שפת C
אל השיא

זיו איילון

לספר זה לא מלווה תוכנה. את קבצי התוכניות תוריד יחד עם הספר או תמצא אותם באתר הוד-עמי

הוצאת הוד-עמי
ספרי מחשבים

עורך ראשי: **יצחק עמיהוד**

עריכה לשונית ועיצוב: **שרה עמיהוד**

עיצוב עטיפה: **ישראל מצגר**

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי עשתה כמיטב יכולתה למסור מידע אודות השמות המסחריים המוזכרים בספר זה ולציין את שמות החברות, המוצרים והשירותים. שמות מסחריים רשומים (registered trademarks) המוזכרים בספר צוינו בהתאמה.

הודעה

ספר זה מיועד לתת מידע אודות מוצרים שונים. נעשו מאמצים רבים לגרום לכך שהספר יהיה שלם ואמין ככל שניתן, אך אין משתמעת מכך כל אחריות שהיא.

המידע ניתן "כמות שהוא" ("as is"). הוצאת הוד-עמי אינה אחראית כלפי יחיד או ארגון עבור כל אובדן או נזק אשר ייגרם, אם ייגרם, מהמידע שבספר זה, או מהתקליטור שמצורף לו.

לשם שטף הקריאה כתוב ספר זה בלשון זכר בלבד. ספר זה מיועד לגברים ונשים כאחד ואין בכוונתנו להפלות או לפגוע בציבור המשתמשים/ות.

- **טלפון: 09-9564716**
- **פקס: 09-9571582**
- **דואר אלקטרוני: info@hod-ami.co.il**
- **אתר באינטרנט: www.hod-ami.co.il**

ספר זה מוקדש לממציא הקפה...

חב אני תודות רבות למשפחתי,
על התמיכה והעידוד שהעניקה לאורך כל הדרך.
תודות מיוחדות שלוחות לאבי, איתן איילון, על ההגהה הקפדנית;
לזיוה קונצמן, על ההערות המקצועיות מאירות-העיניים;
לשלום מנדל, על אמונו ביכולתי ועל שיחות הנפש המעודדות.
תודה.

זיו איילון

11/2000

C Language - to the top

By Ziv Ayalon

© כל הזכויות שמורות

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון: 09-9564716 פקס: 09-9571582
www.hod-ami.co.il
info@hod-ami.co.il

אין להעתיק או לשדר בכל אמצעי שהוא ספר זה או קטעים ממנו בשום צורה ובשום אמצעי אלקטרוני או מכני, לרבות צילום והקלטה, אמצעי אחסון והפצת מידע, ללא אישור בכתב מאת ההוצאה, אלא לשם ציטוט קטעים קצרים בציון שם המקור.

הודפס בישראל 2000

הדפסה 2/2002

All Rights Reserved
HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, 2000, 2002

מסת"ב 9-268-361-965 ISBN

תוכן עניינים מקוצר

מבוא 13

פרק 1: אלגוריתמיקה 27

פרק 2: תרשימי זרימה 61

פרק 3: כלי הפיתוח של השפה 87

פרק 4: אבני היסוד של השפה 101

פרק 5: משתנים 121

פרק 6: משפטי תנאי 177

פרק 7: לולאות 233

פרק 8: פונקציות 293

פרק 9: מערכים 337

פרק 10: מחרוזות 381

נספח א': על שפת C ושפות אחרות 419

נספח ב': הודעות שגיאה נפוצות 427

~~נספח ג': התקליטור המצורף והתוכנה TCLite ..... 445~~

אינדקס 457

תוכן העניינים

מבוא 13
ייחוד הספר 13
דרישות 14
מבנה הספר 14
למי מיועד הספר 15
אופן הלימוד 16
כיצד להשתמש בספר 16
תרגול 16
הדוגמאות 17
מוסכמות 17
ידע מתמטי 20
קבוצות מספרים 20
~~התקליטור המצורף ..... 21~~
~~תוכנת TCLite ..... 21~~
~~התיקיה software\tclite\ ..... 21~~
~~התקנת התוכנה ..... 22~~
~~הפעלת התוכנה ..... 23~~
~~הרצת תוכניות ..... 23~~
~~כתיבת עברית בתוכנית ..... 24~~
~~קוד המקור בתקליטור המצורף ..... 24~~
~~העתקת קבצי קוד המקור לדיסק הקשיח ..... 25~~

פונקציות ספריה חשובות 26

פרק 1: אלגוריתמיקה 27
מושגי תכנות בסיסיים 27
האלגוריתם 28
כללים בתכנון האלגוריתם 29
נכון מבחינה תחבירית 29
מובן לגורם המבצע 29
חד-משמעי 29
נכון מבחינה לוגית 30
אלגוריתמים המיועדים לאדם 30

אלגוריתמים פשוטים 32
הוראת הקלט 32
חוקים במתן שמות למשתנים 33
הוראת הפלט 34
הוראת ההשמה 34
תנאים 38
ביצוע מותנה באלגוריתמים המיועדים לאדם 38
ביצוע מותנה באלגוריתמים המיועדים למחשב 40
התנאי המורכב וגם 43
התנאי המורכב או 45
הוראות תנאי מקוננות 47
לולאות 48
בצע ... פעמים 48
עבור ... מ-... עד ... בצע 51
כל עוד ... בצע 53
בצע ... עד ש 55
אלגוריתמים ומערכים 57
סיכום 59

פרק 2: תרשימי זרימה 61

תרשימי זרימה פשוטים 61
תרשימי זרימה המיועדים לאדם 61
ייצוג פעולות אלגוריתמיות 63
פעולת הקלט 63
פעולת הפלט 64
פעולת ההשמה 66
תרשימי זרימה המיועדים למחשב 66
תרשימי זרימה בעלי תנאים 69
הוראות תנאי מקוננות 75
תרשימי זרימה בעלי לולאות 77
סיכום 85

פרק 3: כלי הפיתוח של השפה 87

סוגי קבצים 87
יצירת הקבצים 88
הידור 88
קישור 89
התוכנית הראשונה 90
יצירת קובץ ההרצה 90
יצירת קובץ ההרצה מסביבת הפיתוח המשולבת 92
כתיבת התוכנית 92

הידור וקישור התוכנית..... 93
הרצת התוכנית..... 94
יצירת קובץ ההרצה משורת הפקודה 94
כתיבת התוכנית..... 94
הידור וקישור התוכנית..... 95
הרצת התוכנית..... 96
עבודה עם המהדר..... 97
הודעות שגיאה 97
הודעות אזהרה..... 98
התנסות עם הודעות המהדר 98
סיכום..... 100

פרק 4: אבני היסוד של השפה 101
תוכנית שלד 101
תווים לבנים..... 103
פלט בסיסי..... 103
ירידה לשורה חדשה 106
הערות..... 107
תווי בקרה..... 110
מרכיבי השפה 113
מילות מפתח 114
מזהים 114
קבועים 114
אופרטורים..... 115
סימני פיסוק..... 115
מהי פונקציה?..... 116
קריאה לפונקציה 117
קבצי ספריה..... 118
סיכום..... 119

פרק 5: משתנים 121
מהו משתנה? 121
הצבת ערכים 122
סוגי משתנים..... 125
המשתנה השלם 125
המשתנה העשרוני..... 126
דיוק בערכים שלמים ועשרוניים..... 128
הגדרת משתנים..... 128
פלט ערכי משתנים..... 130
קליטת ערכים מהמשתמש 137

המשתנה התווי..... 143
פלט תווים..... 145
קליטת תווים..... 147
הפונקציה putch 150
הפונקציה getch..... 151
פלט מורכב באמצעות printf 153
אתחול משתנים..... 158
אופרטורים מקוצרים..... 159
אופרטורי השמה מקוצרים 159
אופרטורי העלאה והפחתה עצמית..... 161
✪ ביטויים מורכבים..... 164
ערך מוחזר מאופרטורי העלאה והפחתה עצמית..... 164
הערך המוחזר מאופרטורי השמה 169
סוגי משתנים נוספים..... 172
הגדרה ואתחול..... 173
השמה..... 173
קלט ופלט..... 174
סיכום..... 174

פרק 6: משפטי תנאי..... 177

ביטויי יחס 177
משפט התנאי if..... 180
משפט התנאי if-else..... 185
קינון משפטי if 191
קינון משפטי if-else..... 194
המשפט switch 203
גלישה..... 207
שורת default..... 212
אופרטורים לוגיים..... 213
אופרטור וגם הלוגי 213
אופרטור או הלוגי 216
אופרטור השלילה הלוגי..... 219
שילוב אופרטורים לוגיים 221
✪ ערכי ביטויי יחס 224
סיכום..... 231

פרק 7: לולאות 233
הלולאה while 233
הלולאה do-while 242
הלולאה for 251
✪ אופרטור הפסיק 262
שילוב לולאות ומשפטי תנאי 266
לולאות מקוננות 270
לולאות אינסופיות 279
✪ משפטי בקרה נוספים 280
המשפט goto 280
המשפט break 283
המשפט continue 287
סיכום 292

פרק 8: פונקציות 293
היתרונות בשימוש בפונקציות 293
הגדרת הפונקציה 294
פונקציות פשוטות 296
סדר ביצוע פונקציות והחזרה מהן 300
פונקציות שמקבלות פרמטרים 303
פונקציות שמחזירות ערכים 313
שימושים נוספים במשפט return 320
הצהרת הפונקציה 322
פונקציות ומשתנים 327
✪ אופני הגדרה והצהרה נוספים 332
השמטת הערך המוחזר 332
השמטת שמות הפרמטרים בהצהרת הפונקציה 333
הסגנון המסורתי להגדרה ולהצהרה על פונקציה 333
סיכום 335

פרק 9: מערכים 337
הצורך במערכים 337
מערכים חד-מימדיים 338
מערכים דו-מימדיים 347
אתחול מערכים 360
אתחול מערך חד-מימדי 360
אתחול מערך דו-מימדי 362
מערכים רב-מימדיים 365
אתחול מערך רב-מימדי 367
שליחת מערכים לפונקציות 371
שליחת מערך חד-מימדי 371

שליחת מערך דו-מימדי..... 377
סיכום..... 379

פרק 10: מחרוזות..... 381

הקבוע המחרוזתי..... 381
המחרוזת כמערך תווי חד-מימדי..... 383
אתחול מחרוזות..... 385
המחרוזות והמשתמש..... 389
פלט מחרוזות..... 389
קליטת מחרוזות..... 391
הפונקציה gets..... 392
פונקציות ספריה לטיפול במחרוזות..... 396
מציאת אורך מחרוזת..... 397
העתקת מחרוזת..... 400
הוספת מחרוזת..... 405
השוואת מחרוזות..... 408
מערכי מחרוזות..... 415
שליחת מחרוזות לפונקציות..... 417
שליחת מחרוזת..... 417
שליחת מערך מחרוזות..... 417
סיכום..... 418

נספח א': על שפת C ושפות אחרות..... 419

סוגי שפות תכנות..... 419
שפת מכונה..... 419
שפת סף..... 420
שפה עילית..... 420
הקשר של שפת C לשפות התכנות האחרות..... 422
התפתחות שפת C..... 422
מ-Algol68 ל-B..... 422
מ-B ל-C..... 423
תקן ANSI C..... 423
יתרונות השפה וחסרונותיה..... 424
תכנות מונחה עצמים באמצעות C++..... 426

נספח ב': הודעות שגיאה נפוצות..... 427

נספח ג': התקליטור המצורף והתוכנה TCLite 445
התיקיות הרלוונטיות לספר..... 446
הפעלת התקליטור 446
קטלוג ספרים 446
התקנת תוכנות..... 447
התוכנות המומלצות להתקנה..... 447
קוד המקור 447
תוכנת TCLite..... 448
התקנת התוכנה 448
הפעלת התוכנה..... 449
הרצת תוכנית עם TCLite 449
עקרונות העבודה..... 450
חלון העריכה 455
הרצת תוכנית דרך שורת DOS 455
כתיבת עברית בתוכנית..... 456
מה עוד בתקליטור?..... 456

אינדקס..... 457

מבוא

ייחוד הספר

הספר "שפת C אל השיא" נכתב עקב מחסור בספרי לימוד נאותים בשפה העברית להוראת שפת C, לדעתי. זכורים לי הקשיים בהם נתקלתי כאשר למדתי לראשונה את השפה מספרים שהניחו כי לקורא ידע קודם בתכנות. מכיון שלא היו בנמצא ספרי לימוד בנושא, נאלצתי להסתפק בספרים אלו, קשים להבנה ככל שהיו.

החל משנת הלימודים תש"ס מאפשר משרד החינוך לענות על שאלון בחינת הבגרות במדעי המחשב בשפת C. עקב כך גבר הצורך בספרי לימוד לשפת C המיועדים למתחילים, שהרי רוב התלמידים הניגשים לבחינת הבגרות במדעי המחשב אינם בעלי ניסיון קודם בתכנות.

"שפת C אל השיא" מיועד במיוחד לתלמידים אלה, בהיותו הספר היחיד המלמד את שפת C על-פי תוכנית הלימודים של מקצוע מדעי המחשב. הספר מיועד לכל אדם העושה את צעדיו הראשונים בתכנות בכלל ובשפת C בפרט, ואינו דורש מהקורא ידע קודם בתכנות. מושגים רבים הנזכרים כמובנים מאליהם בספרי מחשבים אחרים, כגון אלגוריתם, שפת תכנות ומהדר, מוסברים במלואם בספר זה. הפרקים הראשונים עוסקים בהרחבה בתורת האלגוריתמיקה ובתרשימי זרימה, ובכך נבדל הספר מיתר ספרי הלימוד, שמתעלמים לגמרי מנושאים חשובים אלה. למעשה, כל נושא בתוכנית הלימודים מוסבר בספר בבהירות רבה ולעומק. מצד אחד, הספר עוסק בנושא מהרמה הבסיסית והתיאורטית ביותר, ומצד שני, הוא מעמיק בכל תחום ותחום, תוך שימת לב מיוחדת לשגיאות נפוצות.

גם מבחינה דידקטית נבדל הספר מספרי לימוד אחרים. נעשה שימוש רב בתיבות טקסט (מלל) בולטות, המזהירות את הקורא מפני פעולה מסוכנת, מסבות את תשומת ליבו לנושא חשוב וכדומה. למעלה ממאה איורים מלווים את הכתוב וממחישים פעולות תכנות ותהליכים מורכבים, מתארים תחביר של הוראות בשפה, מייצגים את קוד המקור בתרשימי זרימה ועוד. כמאה דוגמאות מלאות, מלוות בהסברים מפורטים, מנחות את הקורא כיצד לשלב את כל רכיבי השפה ליצירת תוכניות שימושיות ויעילות.

דרישות

כדי ללמוד לתכנת עליך להצטייד בשלושה דברים: במהדר של שפת C, בספר לימוד ובכוח רצון. מהדר של שפת C מסופק יחד עם הספר, כך ששתי הדרישות הראשונות כבר נענו. כל שנותר לך הוא להשקיע ולהתמיד. אם תלמד בעקביות ותתרגל את החומר הנלמד, תפנים במהרה את חוקי שפת C ותיווכח עד כמה עיסוק זה מהנה ומספק.

שים לב!

אם ברשותך מהדר C, תוכל להשתמש בו במקום במהדר המסופק עם הספר. קיימים מהדרים רבים לשפת C הפועלים תחת סביבות DOS ו-Windows, והתוכניות המופיעות בספר ירוצו בהצלחה תחת כולם.

מבנה הספר

כאמור, הספר מחולק למספר פרקים, כאשר כל פרק עוסק בנושא אחר בשפת C. להלן תיאור קצר של פרקי הספר:

פרק 1 – אלגוריתמיקה

פרק זה מהווה מבוא למדעי המחשב, ודן באופן תיאורטי באופן בו יש לתכנן הוראות המיועדות למחשב.

פרק 2 – הרקע לשפת C

פרק זה דן בשפות תכנות בכלל וברקע לשפת C בפרט.

פרק 3 – כלי הפיתוח של השפה

פרק זה מסביר כיצד מבצע המחשב הוראות הכתובות בשפת C וכיצד להשתמש במהדר השפה.

פרק 4 – אבני היסוד של השפה

בפרק זה מתחיל התכנות המעשי בשפת C. הפרק מסביר בקצרה כיצד בנויה תוכנית בשפת C, כיצד להציג מלל על המסך ומהם רכיבי השפה.

פרק 5 – משתנים

בפרק זה מוסברים הגדרת משתנים, משפטי השמה ואופרטורים שונים, ופליטת וקליטת ערכים.

פרק 6 – משפטי תנאי

פרק זה עוסק בביטויי יחס, באופרטורים לוגיים ובהוראות מותנות.

פרק 7 – לולאות

פרק זה עוסק בהוראות לביצוע חוזר.

פרק 8 – פונקציות

פרק זה עוסק בהגדרות פונקציות, בהצהרות עליהן ובשליחת פרמטרים אליהן.

פרק 9 – מערכים

פרק זה מסביר כיצד להגדיר, לפנות ולאתחל מערכים בעלי מימדים שונים.

פרק 10 – מחרוזות

פרק זה מסביר כיצד לעבד נתוני מלל באופן ישיר ועל ידי פונקציות הספריה לטיפול במחרוזות.

שים לב!

יש המלמדים את נושא הפונקציות לאחר נושאי המערכים והמחרוזות. תוכל לבחור בסדר שנוח לך, משום שפרקים 9 ו-10 אינם מזכירים את המושג פונקציה, ולכן ניתן ללמדם גם לפני לימוד הפונקציות.

למי מיועד הספר

הספר מיועד לכל אדם המעוניין ללמוד לתכנת בשפת C רבת העוצמה. הספר מתאים הן למי שאין ברשותו ידע קודם בתכנות והן למתכנתים ה"מהגרים" לשפת C משפות תכנות אחרות. מתכונת הספר הוכנה במכוון עבור תלמידים הניגשים לבחינת הבגרות ביסודות מדעי המחשב ברמה של 2 יחידות לימוד, ולכן הוא אינו מקיף את כל התחומים הקשורים לתכנות בשפת C. אך הנושאים המרכזיים בשפת C מועברים בהעמקה רבה – כה רבה למעשה, עד שמספר נושאים אינם נחוצים לבחינת הבגרות.

נושאים אלו מסומנים בתוכן העניינים ולאורך הספר בכוכבית (✪), ואלה הם:

הסעיף **ביטויים מורכבים** בפרק 5, משתנים

הסעיף **ערכי ביטויי יחס** בפרק 6, משפטי תנאי

הסעיף **אופרטור הפסיק** בפרק 7, לולאות

הסעיף **משפטי בקרה נוספים** בפרק 7, לולאות

הסעיף **אופני הגדרה והצהרה נוספים** בפרק 8, פונקציות

אופן הלימוד

הספר מחולק למספר פרקים, כאשר כל פרק עוסק בנושא אחר בשפת C. כדי לתכנת ביעילות בשפת C עליך ללמוד היטב את כולם. הפרקים אינם ערוכים בהכרח בסדר קושי עולה; ייתכן כי פרק מסוים יהיה פשוט יותר מזה שקדם לו. רמת הקושי גם מושפעת מהידע הקודם שברשותך.

התוכניות המוצגות בפרקים הראשונים של הספר הינן פשוטות ולא שימושיות במיוחד, שכן יכולות התכנות שברשותנו מוגבלות. בהמשך הלימוד נתחיל לכתוב תוכניות מורכבות היכולות לשמש אותנו אף לצרכי היום-יום. בחלק מהמקרים נעדיף להציג דוגמאות פשוטות ופחות שימושיות כדי להקל על הלמידה, שהרי עדיף להדגים את הנושא הנלמד בתוכנית קצרה ככל הניתן.

כיצד להשתמש בספר

ניתן ללמוד מהספר בשתי דרכים. אם אינך מכיר את השפה, מומלץ להתחיל בפרק הראשון ולהתקדם, בקצב המתאים לך, אל הפרקים הבאים. לא מומלץ לדלג על פרקים, משום שרובם מסתמכים על חומר שנלמד בפרקים הקודמים להם. אם אתה כבר מכיר את שפת C, תוכל להעמיק בעזרת הספר בנושאים ספציפיים שאינך מכיר או ששכחת, תוך שימוש בתוכן העניינים ובאינדקס.

תרגול

הספר מספק את כל התיאוריה הדרושה לשם הצלחה בבחינת הבגרות במדעי המחשב, אך התרגול בכתיבת תוכניות מלאות חשוב לאין ערוך. מומלץ בחום כי תקליד את כל התוכניות במחשב ותתרשם מפעולתן (הסבר על כתיבת התוכניות והרצתן במחשב מופיע בפרק 3). מלבד העובדה שהקלדה מקלה על זכירת כללי השפה, לאחר שכתבת את התוכנית תוכל לערוך אותה ולבחון את השפעות השינוי על ריצת התוכנית, ובכך לרכוש ידע רב.

הספר "**שפת C – תרגול אל השיא**", שראה אור בהוצאת הוד-עמי, מהווה אמצעי מושלם לתרגול החומר התיאורטי והמעשי הנלמד בספר זה. הספר כולל מאות תרגילים מגוונים: החל משאלות פתוחות, דרך מציאת טעויות במשפטים נתונים וכלה במטלות תכנות מלאות. מבנה הספר תואם במלואו את מבנה הספר הזה, כך שנוח ללמוד עם שניהם בו-זמנית. לאחר קריאת סעיף מסוים בספר זה תוכל לפנות אל אותו סעיף בספר התרגילים ובכך לתרגל ולהפנים את החומר הנלמד.

הדוגמאות

מלבד הסברים וקטעי תוכניות קצרים, הספר כולל כמאה דוגמאות מלאות. דוגמאות אלו משולבות עם החומר התיאורטי הנלמד, והן מאפשרות לחזור עליו וליישם אותו במסגרת של תוכנית מלאה.

כל דוגמה מורכבת מתרגיל, מפתרון מלא ומהסבר. התרגיל הוא המטלה אותה מבצעת הדוגמה. הפתרון הוא בדרך כלל תוכנית מלאה בשפת C, והוא מתבסס על החומר שנלמד באותו סעיף. הסברי הדוגמה הנוגעים לחומר החדש שמוצג בה מפורטים מאוד ומלווים, במידת הצורך, באיורים.

תוכל להשלים את המטלות המוסברות בדוגמאות לפני קריאת פתרונותיהן, ובכך לתרגל את החומר הנלמד.

מוסכמות

ספר הכתוב כגוש מלל רצוף אינו נעים לקריאה. בספר זה נעשה שימוש במוסכמות הבאות כדי להקל על קריאתו:

(1) אם נהנית מהקריאה אזי
(1.1) ספר לחבריך
(2) אחרת
(2.1) ספר לנו

מלל המוצג בגופן כזה מציין רצף הגיוני כלשהו של הוראות.

```
sum = number1 + number2;
average = grades / 10;
```

מלל בלעז המוצג בגופן שרוחב אותיותיו קבוע מציין הוראות הכתובות בשפת C.

כל פרק מכיל מספר דוגמאות, הנפתחות באיור זה.

סימן זה מציין סוף של דוגמה.

כל התוכניות המלאות המוצגות בספר מצויות בתקליטור המסופק עמו. כאשר נציג תוכנית מלאה בשפת C, נכתוב בראשה את שם הקובץ שלה.

FILE.C

כל תוכנית מלאה מוצגת כדוגמה ממוספרת כדי שנוכל לפנות אליה בקלות בהמשך הלימוד. המספר שמשמאל לנקודה מציין את מספר הפרק בו מופיעה התוכנית, והמספר שמימין לנקודה מציין את מספר התוכנית בפרק זה. תוכנית 8.4 היא, לכן, התוכנית הרביעית בפרק השמיני.

תוכנית 8.4

תוכנית מלאה תיכתב במסגרת אפורה ומוצללת.

```
void main()
{

}
```

הדוגמאות המוצגות בספר מוסברות בפירוט רב. כדי להסביר שורה מסוימת, נציג אותה על רקע אפור ואת ההסבר לה נכתוב בהמשך.

```
void main()
```

זוהי הגדרת הפונקציה הראשית.

```
{   }
```

אלה הם סוגריים מסולסלים.

כבר בשלב מוקדם של הלימוד נכתוב תוכניות המציגות הודעות על המסך. כאשר נרצה להדגים את ההודעה שמציגה תוכנית מסוימת, נרשום אותם בתוך מסגרת.

```
Hello there!
```

כאשר ניתקל במושג חדש נרשום אותו באות מודגשת ונציין את המונח הזהה לו באנגלית.

רצף הוראות מכונה **אלגוריתם** (algorithm).

סימן זה מייצג שורות הכתובות בשפת C שאינן מצוינות במפורש. נשתמש בסימן זה בעיקר כאשר אין חשיבות לפעולת שורות אלו או כאשר הסברים אודותיהן יינתנו בהמשך.

```
/* ... */
```

איך לא...

במסגרת כזו תופיע דוגמה לפעולה אסורה.

במחשבה שנייה...

מהי המסגרת הזו?

במסגרת כזו תופיע שאלה ותשובה בצידה.

זהירות!

במסגרת כזו תופיע אזהרה המתריעה מפני ביצוע פעולה מסוכנת.

עצה

במסגרת כזו תופיע המלצה בנושא מסוים.

שים לב!

במסגרת כזו יופיע נושא חשוב שראוי לציינו במיוחד.

ידע מתמטי

מקצוע מדעי המחשב קשור בטבורו למתמטיקה, וגם בבחינת הבגרות במדעי המחשב מופיעות שאלות המצריכות הבנה בסיסית במתמטיקה. הידע המתמטי הנדרש להבנת חומר הלימוד נרכש בחטיבת הביניים, ולכן מושגים מתמטיים פשוטים כגון חזקה, שורש וממוצע אינם מוסברים. זהו אינו ספר לימוד במתמטיקה, אך מושגים מתמטיים מורכבים יותר, כגון עצרת ומשוואה ריבועית, מוסברים בקצרה ובליווי דוגמאות.

במהלך הלימוד נפנה אל קבוצות מספרים מסוימות תוך ציון שמן. כדי שלא נצטרך להסביר את תכונותיה של קבוצה בכל פעם שנדון בה, נתאר עתה את כל קבוצות המספרים הבסיסיות.

קבוצות מספרים

כמות המספרים הקיימים בימינו היא אינסופית. במהלך אלפי השנים האחרונות, חילקו אותם המתמטיקאים למספר קטגוריות. בראשית הדרך נעזרו אנשים במספרים כדי לייצג כמויות של פריטים בהם השתמשו בחיי היומיום. כמויות אלו היו חיוביות ושלמות, ולכן מספרים אלו היו 1, 2, 3, וכדומה. מספרים אלו מכונים בימינו **מספרים טבעיים** (natural numbers).

ההינדים הרחיבו את טווח המספרים שהיו ידועים באותה תקופה בבוחנם את המתרחש כאשר מפחיתים מספר גדול ממספר קטן. קל היה להפחית 2 מ-4, אך הפחתה של 4 מ-2 הניבה מספר שלא ניתן היה לייצגו על ידי המספרים הטבעיים. מספר כזה יוחס לקבוצת מספרים הנקראים **מספרים שליליים** (negative numbers). מספרים אלו הינם מופשטים ביותר, שהרי אין דרך להמחיש כמות שלילית באמצעים העומדים לרשותנו. ואכן, מתמטיקאים רבים סירבו להכיר בהם בזמנו.

פיתגורס, מתמטיקאי יווני שחי במאה השישית לפנה"ס, ומאות תלמידים אותם לימד את רזי המתמטיקה, חקרו רבות את המספרים הטבעיים. מספרים אלו והמספרים השליליים, יחד עם מספרים המייצגים מנה בין מספרים שלמים (היינו, שברים כגון ¾), מכונים **מספרים רציונליים** (rational numbers).

קיימים מספרים אותם לא ניתן לייצג באמצעות יחס בין מספרים טבעיים. מספרים אלו מכונים **מספרים אי-רציונליים** (irrational numbers). מספר אי-רציונלי אינו שלם ואינו שבר, ולכן לא ניתן לכתבו בכתיב עשרוני. מספר אי-רציונלי מפורסם הוא המספר פאי (π), שערכו בדיוק של עשר ספרות אחרי הנקודה העשרונית הוא 3.1415926535. אך ערך זה, מדויק ככל שיהיה, אינו מדויק לגמרי. מכיון שפאי הוא מספר אי-רציונלי, לא קיימת מחזוריות בספרותיו העשרוניות (מחזוריות כזו קיימת במספרים כגון 0.333333 ו-0.123123123), ולכן לא ניתן לכתבו בדיוק מוחלט.

קיומם של מספרים אי-רציונליים הוכח על ידי יאוקלידס, שנולד בערך בשנת 330 לפנה"ס. הוכחה ידועה נוספת של יאוקלידס גורסת שקיימים אינסוף מספרים ראשוניים. **מספר ראשוני** (prime number) הוא מספר שהמחלקים היחידים שלו (כלומר, המספרים

בהם הוא מתחלק ללא שארית) הם 1 והוא עצמו. מספר שאינו ראשוני מכונה **מספר פריק** (reducible number). קיימים אינסוף מספרים ראשוניים ופריקים.

התקליטור המצורף

תוכנת TCLite

לנוחותך תמצא בתקליטור מהדר TCLite של בורלנד, באדיבות חברת Borland. באמצעות מהדר זה תוכל להדר את התוכניות שבספר. להידור תוכניות מורכבות יותר מאשר בספר זה תזדקק למהדר שלם של השפה.

מהדר TCLite הוא גירסה קלה של מהדר Borland C\C++, אשר עוצב על ידי חברת בורלנד, כדי לאפשר תרגול של תוכניות C.

הוראות ההתקנה וההפעלה המלאות נמצאות בתקליטור בקובץ Software\tclite\install.doc

אם מותקן כבר מהדר Borland במחשב, אל תתקין את TCLite – אלא רק העתק את התוכניות מהתקליטור.

התיקיה software\tclite\

תוכנת **TCLite** (Turbo C Lite), מהדר לשפת C\C++ של חברת Borland International, אשר מיועד למחשבי IBM PC ותואמים. זהו מהדר מהיר מאוד המסוגל, לפי דיווחי החברה, להדר מעל 7,000 שורות בדקה.

התוכנה כוללת:

- מהדר (compiler).
- עורך (editor) המשמש לכתיבת ותיקון התוכניות.
- מערכת תפריטים נוחה, לבחירת האופציות השונות במערכת.
- מקשר (linker).
- מערכת איתור שגיאות בהידור.
- מערכת לבניית תוכניות מקבצי מקור רבים.

הודעה

רשיון השימוש לתוכנה מוגבל להדגמה בלבד. התוכנה משווקת כמו שהיא ("**as is**"), ללא אחריות לתוצאות העלולות להיגרם מהשימוש ו/או אי השימוש בתוכנה מכל סיבה שהיא, וכן להפסד כספי, זמן, רכוש ו/או כל הפסד אחר שייגרם, אם ייגרם.

התקנת התוכנה

ההתקנה וההפעלה פשוטים. עקוב אחר הסעיפים הבאים. ניתן להתקין תחת Windows, אך להפעיל, רצוי וצריך תחת **DOS**.

ההתקנה תבוצע בסביבת העבודה DOS, ולצורך ההסבר נתייחס לכונן C: (אותן פעולות תעבודנה גם על כל כונן אחר).

>>אם אתה נמצא ב-Windows - כבה והפעל את המחשב במצב DOS<<

>>בחלון כיבוי המחשב סמן את האפשרות Restart the computer in MS-DOS mode?<<

1. יש לעבור לתיקיית השורש על ידי הפקודה: cd\ (קיצור של change directory), ולהגיע למצב זה: C:\>_.
2. במקרה וזה איננו ניסיון התקנה ראשון - יש למחוק את התיקיה TCLITE, שבתוכה תותקן התוכנה. עשה זאת על ידי הפקודה: deltree tclite, ואז תראה על המסך את המשפט הבא: deltree tclite*.*. לבסוף, הקש <Enter>.

   אם התיקיה לא קיימת, הסמן יעבור מיידית לשורה חדשה וימתין להוראה. אם התיקיה קיימת, תופיע השאלה הזו:

   Delete directory "tclite" and all its subdirectories? [yn] _

   כלומר, "האם אתה בטוח שאתה רוצה לבטל תיקיה זו?" הקש על המקש <Y> ואחר כך על <Enter>. המחשב ידווח: Deleting...tclite.
3. עתה יהיה עליך לעבור לכונן המתאים של התקליטור (למשל X:). המעבר ל- X: - על ידי הפקודה: x: (ואז על המסך יהיה כתוב: x:\>C:). לאחר הקשת <Enter> הסמן המהבהב יופיע אחרי X:\>_.
4. עבור לתיקיה של TCLite: **cd software\tclite**
5. עכשיו הכל מוכן לביצוע ההתקנה. כתוב את הפקודה: **install c:** X:\software\tclite> (שפירושה, התקנה לתוך כונן C:). שים לב: **אל תשכח** לכתוב c:, כלומר לאן יש לבצע את ההתקנה. המחשב ידווח על התקנת התוכנה TCLITE. משך זמן ההתקנה תלוי במהירות המחשב שלך, ויכול להימשך גם מעל 2 דקות. בדרך כלל ההתקנה מהירה יותר.

6. במקרה של תקלה (תקיעת המערכת) יש לבטל את ההתקנה על ידי <Ctrl>+<C>. במקרה זה המחשב ישאל: ?(Y/N) Terminate batch job; כלומר, האם לסיים את העבודה. יהיה עליך להקיש <Y> ו- <Enter>, ואז, בהנחה שיש די זיכרון בדיסק הקשיח, לחזור על שלבים 2-5.

7. לאחר סיום ההתקנה שים את התקליטור בצד ושמור אותו כגיבוי. הפעלת התוכנית והרצת התוכניות נעשית מתוך הדיסק הקשיח.

הפעלת התוכנה

1. יש לחזור מכונן :X לכונן :C, ולהימצא בתיקיית השורש של כונן C.
2. הכניסה ל- C - על ידי הפקודה **c**>\:C המפעילה את קובץ ההפעלה C.BAT. כתוצאה, מקבלים את סביבת העבודה של Turbo C/C++, ובתוכה חלון פתוח בצבע כחול, ששמו: **NONAME00.CPP** (רק בפעם הראשונה שנכנסים לתוכנה). אם הקובץ c.bat לא הועתק לכונן C שלך, העתק אותו ידנית או צור קובץ אצווה חדש. קובץ ההפעלה של התוכנה הוא tc.exe שנמצא ב- \tclite\bin.
3. כדאי לסגור את החלון **NONAME00.CPP** על ידי צמד המקשים <Alt>+<F3> ואז מקבלים חלון אפור ריק, שבראשו רשימת תפריטים על רקע לבן. כרגע אתה מוכן להתחיל לעבוד בעזרת התוכניות שבספר.

שים לב!

ברירת המחדל של המהדר היא לקרוא ולשמור קבצים עם סיומת C או CPP. בחר c.* או cpp.* בהתאם לקבצים (שפת C או שפת C++).

הרצת תוכניות

העתק את התיקיה של התוכניות לפי ההוראות בהמשך.

1. כדי לקבל תוכנית - יש להקיש <F3>. הקשה זו פותחת "חלון שיחה", שכותרתו הראשית: Load a File ("טען קובץ") וכותרת משנה ראשונה הינה: Name ואחריה, בתוך שורה כחולה עם הדגשה ירוקה כתוב: CPP.* (אתה מריץ תוכניות C - ולכן הקש Backspace פעמיים כדי שיישאר רק c.*. אם אינך רואה דבר הקלד c.*, ולאחר מכן הקש <Enter>).
2. תעבור לחלון פנימי שכותרתו Files ("קבצים"), ותראה על רקע בצבע תכלת 2 טורים של שמות קבצים. באופן כללי, מקשי החיצים מאפשרים לעבור עם ה"סמן", שצבעו ירוק בהיר, על כל הקבצים.
3. בחר את התיקיה הרצויה ואת הקובץ הרצוי. הקשה על <Enter> תכניס אותך לתוכנית. כדי לבחור תיקיה אחרת הקש על \.. שמופיע בסוף הרשימה של התוכניות.

4. הקשת <Enter> על שם קובץ תפתח חלון כחול, שכותרתו במרכז הינה: "שם הקובץ", ובתוך החלון יופיע הטקסט של תוכנית זו (או של קובץ זה).
5. כדי להריץ את התוכנית עליך להקיש על צמד המקשים: <Ctrl>+<F9>. המחשב יתרגם את התוכנית לשפת מכונה ויבצע אותה. כדי לראות את פלט התוכנית הקש <Alt>+<F5>. יש להקיש על מקש כלשהו כדי לחזור לחלון התוכנית.
6. המשך ההרצה של תוכניות נוספות מומלץ באופן הבא: לסגור תחילה את חלון התוכנית הנוכחית על ידי צמד המקשים <Alt>+<F3>. אחר כך ניתן לחזור על השלבים 1 עד 5, כאשר בשלב 3 אפשר לבחור תוכנית אחרת.

עצה

בכל שלב בסעיפי הרצת התוכניות אפשר לבטל שלב כלשהו ולחזור לשלב הקודם על ידי המקש <Esc>.

אם המחשב שלכם מחובר לרשת, המלצתנו היא - להקיש F5 בזמן אתחול המערכת כדי לדלג על ביצוע הקבצים autoexec.bat ו-config.sys. אם לא תעשו כך, ייתכן שבתוכניות תראו תווים לא מובנים במקום עברית.

כתיבת עברית בתוכנית

כדי לכתוב עברית בתוכנית ב-tclite:

1. העתק את הקובץ heb.com מהתיקיה בתקליטור software\tclite\ לכונן הקשית הראשי C.
2. הקש **heb** ל-C:\> ו-Enter.
3. היכנס ל-tclite וכאשר תרצה להקליד עברית, הקש — +Ctrl (החזק את Ctrl ולחץ על מקש מינוס). נפתח חלון בו תוכל לכתוב. הקש Enter כדי להכניס את מה שהקלדת לתוכנית.

עיין על התוכנה בקובץ Install.doc בתיקיה software\tclite\ שבתקליטור

קוד המקור בתקליטור המצורף

התוכניות שבספר מצורפות בתקליטור. התוכניות הורצו ונבדקו, אך ייתכן שתמצא טעות כלשהי בספר או בתוכניות שבתקליטור. אנו מבקשים ממך לבדוק את הנושא בעצמך, כחלק מתהליך הלימוד שלך. נודה לך אם תודיע לנו על השגיאה, אם ישנה, כדי שנוכל לתקן בהדפסת המהדורה הבאה.

בתיקיה **books\59299** תמצא את התוכניות בתיקיות לפי מספרי הפרקים.

שים לב, שאם אתה עובד במהדר TCLite - הכל מתבצע בסביבת המהדר. לא יוצרים קובץ EXE ולא יוצאים ל-DOS כדי להריץ אותו. הספר נכתב לשימוש במהדר רגיל ולכן יש לשים לב לשינויים כאשר עובדים ב-TCLite.

העתקת קבצי קוד המקור לדיסק הקשיח

כדי להעתיק את תכולת התיקיה books\59299:

1. לחץ על לחצן התחל, תוכניות, סייר Windows.
2. לחץ על כונן התקליטורים וסמן את התיקיה 59299 שבתיקיה books.
3. גרור אותה לתיקיה Tclite או לתיקיית השורש C.

מכיוון שמקור הקבצים הוא התקליטור, הם מסומנים ל**קריאה בלבד**. יש לשנות מאפיין זה כך:

1. בסייר Windows היכנס לתיקיה בדיסק שבה נמצאים הקבצים שהעתקת (c:\tclite\59299 או אחרת).
2. סמן את כולם על ידי Ctrl+A.
3. הצב את סמן העכבר מעל האזור המסומן ולחץ לחיצה ימנית בעכבר.
4. מתפריט הקיצור בחר **מאפיינים**.
5. בטל את הסימון בתיבה **קריאה בלבד** (דאג שהיא תהיה ריקה).
6. לחץ על **החל**, לחץ **אישור**.

בדוק האם השתנה המאפיין על ידי סימון הקובץ ; לחיצה ימנית ובחירה ב**מאפיינים** בתפריט הקיצור. שים לב שהתיבה של **קריאה בלבד** תהיה ריקה.

התוכניות המצורפות הורצו ונבדקו. עשינו את מירב המאמצים כדי לבדוק את תקינות פעולתן, ולא נותר לנו אלא לאחל לך תרגול מהנה ומועיל.

קרא עוד על התקליטור בנספח ג' בספר ובקובץ ONCD בתקליטור

פונקציות ספריה חשובות

בטבלה מופיעות פונקציות הספריה של שפת C שחשוב להכירן. לצד כל פונקציה מצוינת התוכנית בה היא מוזכרת לראשונה.

פונקציית הספריה	התוכנית	שם התוכנית
printf	תוכנית 4.2	printf.c
scanf	תוכנית 5.4	scanf.c
pow	תוכנית 5.5	scanf2.c
getch / putch	תוכנית 5.8	arith2.c
sqrt	תוכנית 6.3	ifelse2.c
rand / srand	תוכנית 7.8	rand.c
abs	תוכנית 9.4	abs.c
gets	תוכנית 10.2	pal_str.c
strlen	תוכנית 10.3	strlen.c
strcpy	תוכנית 10.4	strcpy.c
strcat	תוכנית 10.5	strcat.c
strcmp	תוכנית 10.6	strcmp.c

פרק 1

אלגוריתמיקה

את לימוד מדעי המחשב נפתח בהיכרות עם מושגים בסיסיים הקשורים בתכנות. לאחר מכן נציג בעיות יומיומיות ובעיות ממוחשבות ונראה כיצד לפותרן באופן תיאורטי, בלי להשתמש במחשב. בפרקים הבאים נשתמש בידע שרכשנו בפרק זה וניישם אותו על ידי תכנות בשפת C.

מושגי תכנות בסיסיים

המחשבים האישיים הם חלק מרכזי בחיינו. הורים משתמשים בהם כדי לפתח את היצירתיות של הפעוטות שלהם; ילדים משחקים בהם בשעות הפנאי; תלמידים כותבים בהם עבודות להגשה; מדענים מסתייעים בהם לחישובים מדעיים מורכבים; חברות מנהלות באמצעותם את חשבונותיהן; מדינות מאחסנות בהם את נתוני אזרחיהן, ועוד.

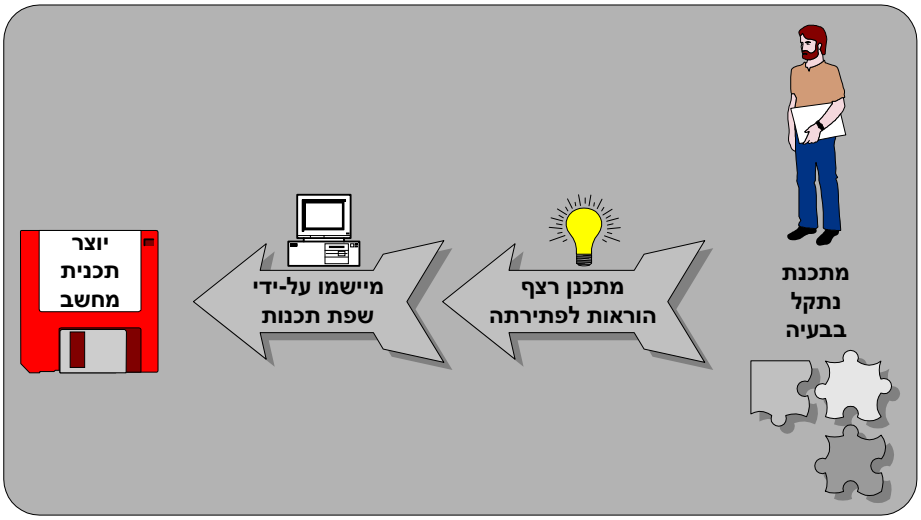
אך לא כל בעיה ניתנת לפתרון על ידי מחשב. למשל, גם המחשבים המשוכללים ביותר לא יכולים להעריך עבודה בהיסטוריה או לקבוע את גילו המשוער של אדם על פי תמונתו – בעוד שכל נער יכול לעשות זאת!

בעיות שאינן מופשטות ושניתן לפותרן בשלבים קבועים וברורים מכונות **בעיות לוגיות**. דוגמאות לבעיות לוגיות הן חישוב ממוצע מספרים, שרטוט מעגל דרך שלוש נקודות נתונות, פתירת משוואות מתמטיות וכדומה. אלו הן בעיות שהמחשב יכול לפותרן, אך לא לפני שנגדיר לו כיצד לעשות זאת.

האדם המגדיר הוראות למחשב מכונה **מתכנת** (programmer). המתכנת אחראי על פיתוח התוכנות המשמשות את משתמשי המחשב, ובידיו כוח רב – ביכולתו לרתום לצידו את כוח העיבוד האדיר של המחשב, ובכך ליצור תוכנות רבות-עוצמה מסוגים שונים.

את ההוראות שעל המחשב לבצע כותב המתכנת ב**שפת תכנות** (programming language). שפת תכנות היא אמצעי דרכו מועברות ההוראות המילוליות, הנכתבות על ידי המתכנת, אל המחשב, בשפה אותה הוא מבין (כפי שנלמד בהמשך, המחשב אינו מבין ישירות את ההוראות שכותב המתכנת). כמו לשפה מדוברת, גם לשפת התכנות חוקים משלה, ואותם יש למלא בקפדנות. כשהחוקים משולבים יחדיו נוצרת **תוכנית מחשב** (computer program), בדומה לשילוב אוצר מילים ותחביר של שפה מדוברת ליצירת משפט מובן. כאשר תופעל תוכנית המחשב, תבוצענה ההוראות המופיעות בה.

תהליך פיתוח טיפוסי של תוכנית מחשב מוצג באיור הבא:



***איור 1.1** – תהליך יצירת תוכנית מחשב*

האלגוריתם

רצף הוראות המיועד לפתרון בעיה לוגית כלשהי מכונה **אלגוריתם** (algorithm). לפני שניגשים למלאכת התכנות עצמה (כלומר, כתיבת ההוראות המיועדות למחשב בשפת תכנות), נהוג לתכנן את האלגוריתם שיבצע את הפעולה המבוקשת. נפתח בדוגמה פשוטה לאלגוריתם המיועד לאדם, המכיל הוראות כיצד לקנות פחית שתייה ממכונת מזון אוטומטית:

(1) בחר את המשקה המבוקש

(2) שלשל מטבעות כנדרש לחריץ המטבעות

(3) לחץ על הכפתור המתאים למשקה המבוקש

(4) קח את פחית השתייה

שים לב!

אלגוריתם זה מיועד, כמובן, לאדם. בדרך כלל יעסקו האלגוריתמים שנפתח בפעולות ממוחשבות, שהרי מטרתנו היא לגרום למחשב לבצע את הוראות האלגוריתם.

כללים בתכנון האלגוריתם

נדון עתה בדרישות שעל האלגוריתם למלא כדי שיהיה נכון, כלומר, כדי שישיג את המטרה לשמה נכתב.

נכון מבחינה תחבירית

הוראות אלגוריתם המיועד למחשב נכתבות, בשלב זה או אחר, בשפת תכנות. כאמור, לשפת התכנות חוקים להם יש לציית בדייקנות. אם לא נציית לחוקים אלה כנדרש, תתקבל **שגיאה תחבירית** (syntax error).

נניח שבשפת תכנות כלשהי חיבור שני מספרים נעשה על ידי התו '+'. למשל, כדי לחשב את סכום המספרים 5 ו-6 יש לכתוב הוראה כזו: 5+6. אם נרצה לבצע הוראה המחשבת סכום זה, אך במקום סימן ה-'+' נרשום בטעות את סימן ה-'@', כך: 5@6, תתקבל הוראה שגויה מבחינה תחבירית.

אדם הקורא משפט ונתקל בטעות תחבירית יכול להשלים את המילים הלא ברורות על פי הקשר המשפט, אך לא כן המחשב: פקודה שגויה מבחינה תחבירית לא תתקבל על ידו.

מובן לגורם המבצע

הגורם המבצע הוא מי שמבצע את האלגוריתם (אדם, מחשב אישי, רובוט וכדומה). כדי שהגורם המבצע את האלגוריתם יוכל לבצע את ההוראות המיועדות לו בהצלחה, עליו להבינן. הדבר נכון הן לגבי אלגוריתמים המיועדים למחשב והן לגבי אלגוריתמים המיועדים לאדם. למשל, ההוראה "חשב את שורשו הריבועי של 1521" היא נכונה אם היא מיועדת למורה למתמטיקה, אך היא אינה נכונה אם היא מיועדת לתלמיד כיתה ב' שזה עתה התחיל את לימודי החשבון.

חד-משמעי

על כל הוראה באלגוריתם להיות חד-משמעית, כלומר, שניתן יהיה לבצעה באופן אחד ויחיד. למשל, האלגוריתם הבא:

(1) קלוט מספר מהמשתמש

(2) אם אפשר, פלוט את שורשו הריבועי

אינו נכון, משום שהביטוי "אם אפשר" ניתן לפירוש באופנים רבים (אם המספר חיובי? אם המספר שלם? אם המספר קטן ממיליון?). באופן דומה, ההוראה "רוץ מספר מטרים",

המיועדת לאדם, אינה נכונה, משום שניתן לבצעה באופנים רבים (שני מטרים? עשרה מטרים?).

נכון מבחינה לוגית

אלגוריתמים יכולים להיות מורכבים מאוד ולהכיל הוראות רבות. כדי שהבעיה הלוגית תיפתר בהצלחה, לא מספיק שההוראות תהיינה נכונות מבחינה תחבירית – עליהן גם, בסופו של דבר, לבצע את המטלה המבוקשת. נסביר זאת על ידי דוגמה פשוטה. בחן את האלגוריתם הבא, המורה למחשב כיצד לחשב ולפלוט ממוצע של שני מספרים נתונים:

(1) חשב את סכום שני המספרים

(2) חלק את התוצאה ב-2

(3) הצג את התוצאה על המסך

אלגוריתם זה הוא מדויק ונכון. עתה, בחן את האלגוריתם הבא:

(1) חשב את סכום שני המספרים

(2) הצג את התוצאה על המסך

(3) חלק את התוצאה ב-2

באלגוריתם זה הוחלפו שתי ההוראות האחרונות, ולכן הוא אינו פולט את ממוצע שני המספרים, אלא את סכומם. כלומר, למרות שההוראה באלגוריתם נכונה מבחינה תחבירית, האלגוריתם אינו פותר בהצלחה את הבעיה הלוגית לשמה הוא נכתב. לכן נאמר כי אלגוריתם זה אינו נכון מבחינה לוגית.

רוב זמנו של המתכנת מושקע באיתור טעויות לוגיות. טעויות תחביריות מתגלות במהרה (כפי שנראה בהמשך, חוקי שפת התכנות מגינים מפניהן), אך לא כן טעויות לוגיות. טעויות לוגיות עלולות לצוץ בכל שלב של הפיתוח, במקומות שלא ציפינו למצאן, והמחשב לא ידחה אותן. המחשב עוקב אחר הוראותינו **בדיוק**, ועלינו לוודא שהן אכן מבצעות את המטלה המבוקשת.

אלגוריתמים המיועדים לאדם

נציג עתה מספר דוגמאות לאלגוריתמים נכונים ושגויים המיועדים לאדם.

דוגמה

הנה דוגמה לאלגוריתם מדויק, המורה לאדם כיצד להיכנס לבית דרך דלתו הנעולה באמצעות מפתח:

(1) הכנס את המפתח לחור המנעול

(2) סובב את המפתח סיבוב שלם בכיוון השעון

(3) הוצא את המפתח מהדלת

(4) לחץ על הידית

(5) דחוף קלות את הדלת

שים לב!

הוראות האלגוריתם הן חד-משמעיות וניתנות בסדר הגיוני, אך הן מיועדות לאדם בלבד (שהרי מחשב אינו מסוגל לבצע פעולות אלו). כלומר, בעוד שהאלגוריתם נכון עבור בן-אדם, הוא נחשב לשגוי מבחינת המחשב.

דוגמה

נפתח עתה אלגוריתם המורה לאדם כיצד לשלוח מכתב בדואר:

(1) הכנס את המכתב למעטפה

(2) הדבק בול בפינה הימנית-עליונה של המעטפה

(3) מלא את פרטי המוען והנמען

(4) שלשל את המכתב לתיבת דואר

דוגמה

בדוגמה זו נבחן אלגוריתם שגוי, האמור להנחות את פלוני כיצד להגיע לתחנת הדלק הקרובה:

(1) המשך לצעוד קדימה עד להצטלבות הרחובות הבאה

(2) פנה שמאלה וצעד כמאה מטרים

(3) פנה ימינה וכעבור זמן קצר היכנס לשביל התחום בפרחים ובוורדים

(4) המשך לצעוד בשביל זה עד שתראה את תחנת הדלק

האלגוריתם שגוי משום שהוראותיו אינן חד-משמעיות, שהרי הביטוי "זמן קצר" ניתן להבנה באופנים שונים.

אלגוריתמים פשוטים

עד כה הצגנו מספר דוגמאות לאלגוריתמים המיועדים לאדם. מעתה נמעיט בדיון על אלגוריתמים "אנושיים", שכן הם אינם מיועדים למחשב – וספר זה מתמקד בפעולות ממוחשבות. כאשר נפתח אלגוריתמים המיועדים למחשב, נשתמש במספר הוראות המורות לו לבצע פעולות בסיסיות. על ידי שילוב הגיוני של הוראות בסיסיות אלו נוכל לפתח אלגוריתמים מורכבים ביותר.

שים לב!

את ההוראות האלגוריתמיות שנכיר בפרק זה לא נוכל להריץ ישירות במחשב, משום שהן אינן כתובות בשפת תכנות המובנת לו. הוראות אלו כתובות בשפה דמוית שפת מחשב, אך חופשית ממנה, הנקראת **פסאודו-קוד** (pseudo code).

הוראת הקלט

בפעולת ה**קלט** (input) מתקבלים נתונים מהאדם המשתמש במחשב, שיכונה מעתה **המשתמש**. בחן, למשל, אלגוריתם המורה למחשב כיצד לפלוט ממוצע של שני מספרים:

(1) חשב את סכום שני המספרים

(2) חלק את התוצאה ב-2

(3) הצג את התוצאה על המסך

אלגוריתם זה הוא חסר משמעות אם שני המספרים אינם ידועים. רק כאשר יידע המחשב מהם הערכים שאת הממוצע שלהם עליו להציג, תופק תועלת מהאלגוריתם. ובכן, שני הערכים בהם תלוי האלגוריתם הם הקלטים, והם מוזנים על ידי המשתמש.

עוצמת הוראות הקלט היא שניתן להפעיל את האלגוריתם תוך שימוש במספרים שונים בכל פעם, ובכך להשתמש בו פעמים רבות. למשל, אם שני המספרים יהיו 8 ו-20, יוצג המספר 14; אם שני המספרים יהיו 100 ו-200, יוצג המספר 150, וכדומה. הפעולות הנעשות בשני המספרים הן קבועות, אך מאחר שהמספרים משתנים, מוצגת בכל פעם תוצאה אחרת.

כל ערך שמתקבל מהמשתמש בפעולת הקלט מוצב ב**משתנה** (variable), המשמש כאמצעי לאחסון נתונים. בכל משתנה ניתן להציב ערך, ובהמשך פעולת האלגוריתם נוכל לקרוא ערך זה ולשנותו. מעובדה זו, שערך משתנה יכול להשתנות, גזור שמו.

משתנים מיוצגים על ידי שמות המורכבים בעיקר מאותיות לועזיות. לכל משתנה יינתן שם המבדיל אותו משאר המשתנים, ובאמצעות שם זה נוכל לפנות אל הערך שבמשתנה.

חוקים במתן שמות למשתנים

קיימים שני חוקים פשוטים בנוגע למתן שמות למשתנים:

- שמות משתנים יכולים להכיל רק אותיות לועזיות, ספרות וקו תחתי (_).
- אין להתחיל שם משתנה בספרה.

שמות משתנים חוקיים הם, למשל, x, abc, number3 ו-floppy_disk. בתוכניות ארוכות מומלץ להעניק למשתנים שמות הגיוניים המרמזים על תפקידם, כגון length, tax_rate ו-perimeter.

איך לא...

להלן מספר דוגמאות לשמות משתנים לא חוקיים:

1stMonth	(אין להתחיל שם משתנה בספרה)
lovely day	(אין רווח בשם משתנה)
Lotsof$	(אין להשתמש בתו '$' בשם משתנה)
ממוצע	(יש להרכיב שם משתנה מאותיות לועזיות)

הנה דוגמה להוראה אלגוריתמית הקולטת ערך מהמשתמש ומציבה אותו במשתנה הנקרא number:

קלוט מספר למשתנה number

כדי לקלוט שלושה מספרים ולהציבם במשתנים a, b ו-c, נכתוב את ההוראות הבאות:

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

ההוראות מוספרו כדי שנוכל לעקוב אחר הסדר בו הן מבוצעות.

שים לב!

בפעולת קלט ניתן לקבל נתונים שונים מהמשתמש, כגון הזזת עכבר, כיבוי והדלקת המדפסת וכדומה. בספר נתמקד בקבלת ערכים מהמקלדת.

הוראת הפלט

בפעולת ה**פלט** (output) נפלטים נתונים אל צג המחשב. כדי לפלוט הודעה פשוטה, כגון "נא הקש שני מספרים", נרשום הוראה אלגוריתמית כזו:

הדפס את ההודעה "נא הקש שני מספרים"

ניתן להציג באמצעות הוראת הפלט גם ערכי משתנים. למשל, כדי להציג את ערך המשתנה number, נרשום הוראה אלגוריתמית כזו:

הדפס את הערך של number

דוגמה

בדוגמה זו נכתוב אלגוריתם הקולט שני מספרים מהמשתמש ופולט אותם בסדר הפוך לסדר בו הוזנו.

(1) קלוט מספר למשתנה num1

(2) קלוט מספר למשתנה num2

(3) הדפס את הערך של num2

(4) הדפס את הערך של num1

אם יוזנו, למשל, המספרים 10 ו-20, ייפלטו המספרים 20 ו-10.

שים לב!

בפעולת פלט ניתן לפלוט נתונים גם למדפסת, לכרטיס קול ועוד. בספר נתמקד בפליטת ערכים אל צג המחשב.

הוראת ההשמה

בפעולת ה**השמה** (assignment) מקבל משתנה ערך כלשהו. שלא כמו בפעולת הקלט, ערך זה אינו מוזן על ידי המשתמש, אלא הוא תוצאה של פעולת חישוב כלשהי.

בכל פעולת השמה מפריד חץ המכוון שמאלה בין שני שדות:

- **משתנה** – משמאל לחץ ובכיוונו יופיע המשתנה בו יוצב הערך.
- **ערך** – מימין לחץ יופיע הערך שיוצב במשתנה.

לדוגמה, הוראה אלגוריתמית המציבה במשתנה x את הערך 15 תיכתב כך:

$x \leftarrow 15$

הערך המוצב במשתנה יכול להכיל פעולות מתמטיות בסיסיות, כגון חיבור, חיסור, כפל וחילוק. למשל, הוראה המציבה במשתנה x את מכפלת המספרים 3 ו-4 תיכתב כך:

$x \leftarrow 3 * 4$

הנה הוראת השמה מעט מורכבת יותר:

$number \leftarrow 4 + 3 * 4 - 1$

כאשר ייחשב המחשב את הערך שיוצב במשתנה number, הוא ינהג על פי סדר הפעולות המקובל במתמטיקה. כלומר, פעולת הכפל תבוצע ראשונה, אחריה תבוצע פעולת החיבור ואחריה – פעולת החיסור (משום שסימן החיבור מופיע לפני סימן החיסור). כלומר, הוראה זו מחושבת באופן הבא:

$number \leftarrow (4 + (3 * 4)) - 1$

ולכן, הערך שיוצב במשתנה number הוא 15.

הטבלה הבאה מציגה את סדר הפעולות על פיו מחושבים ביטויים מתמטיים:

סימן	הפעולה
( )	תחימה בסוגריים
$\sqrt{}$ ^	חזקה, שורש ריבועי
* / %	כפל, חילוק, שארית
+ -	חיבור, חיסור

הטבלה ערוכה בסדר יורד, כלומר, ככל שפעולה קרובה יותר לראש הטבלה, היא מתבצעת קודם. פעולות הנמצאות באותה שורה מבוצעות משמאל לימין. למשל, את הביטוי 3 * 2 / 6 יש לקרוא כך: 3 * (2 / 6), ולכן ערכו הוא 9.

רוב הסימנים המתמטיים המופיעים בטבלה ודאי אינם חדשים לך. נתמקד בשני סימנים מיוחדים:

- הסימן '%' אינו מציין אחוזים, אלא שארית של חלוקת המספר שמשמאלו במספר שמימינו. למשל, ערך הביטוי 6 % 13 הוא 1, משום שהמספר 13 מכיל את המספר 6 פעמיים, והשארית היא 1 (13 - 6 - 6 = 1). כאשר המספר השמאלי בפעולת שארית קטן מהמספר הימני, התוצאה היא המספר השמאלי. למשל, ערך הביטוי 4 % 3 הוא 3.
- הסימן '^' מציין העלאה בחזקה, כאשר המספר שמשמאלו הוא בסיס החזקה והמספר שמימינו הוא מעריך החזקה. למשל, את הביטוי המתמטי 2^3, שערכו 8, נכתוב כ-3 ^ 2.

ניתן להשתמש בערכי משתנים בפעולת ההשמה. למשל, כדי להציב במשתנה avg את ממוצע ערכי המשתנים num1 ו-num2, נכתוב הוראה כזו:

avg ← (num1 + num2) / 2

הסוגריים בהוראה זו הכרחיים, משום שפעולת החילוק מבוצעת לפני פעולת החיבור. אילו נכתבה ההוראה בלי סוגריים, כך:

avg ← num1 + num2 / 2

היה מוצב במשתנה avg סכום הערך שבמשתנה num1 ומחצית הערך שבמשתנה num2, ובזאת, כמובן, לא רצינו.

שים לב!

ההוראה האלגוריתמית avg ← (num1 + num2) / 2 אינה משנה את ערכי המשתנים num1 ו-num2. הערך היחיד שמשתנה בפעולת השמה הוא ערך המשתנה הנמצא משמאל לחץ.

עצה

כאשר מוצב במשתנה ביטוי מתמטי מורכב, השתמש בסוגריים כדי להקל על קריאתו. למשל, במקום הביטוי הבא:

```
7 - 3 * a / b + c
```

כתוב את הביטוי:

```
7 - ((3 * a) / b) + c
```

שים לב כי לא תחמנו את כל הערכים המופיעים בביטוי בסוגריים, כדי לא להקשות על הקריאה שלא לצורך.

אותו משתנה יכול להופיע משני צדי חץ ההשמה. למשל, כדי להעלות ב-2 ערך משתנה מסוים, נניח ששמו x, נכתוב הוראה כזו:

x ← x + 2

וכדי להכפיל ב-2 ערך משתנה בשם d נכתוב הוראה כזו:

d ← d * 2

או הוראה כזו:

d ← d + d

שים לב כי הוראת השמה מתבצעת רק באמצעות חץ, ומשמאלו יכול להופיע משתנה אחד ויחיד.

לכן, ההוראות הבאות אינן נכונות:

a + b ← c

num1 ← num2 ← num3

sum = 10

זהירות!

פעולות שאסורות באלגברה בסיסית אסורות גם במחשב. למשל, חלוקה ב-0 או הוצאת שורש ריבועי ממספר שלילי עלולות להביא לקריסת התוכנית. כמו-כן אין לחשב שארית של חלוקת מספר ב-0. כלומר, אין לחשב את הביטויים הבאים:

num / 0

$\sqrt{-9}$

vase % 0

הטעויות בביטויים אלה מובנות מאליהן, אך חשוב להפיק מהן לקח לגבי ביטויים מורכבים יותר. למשל, אם ברצוננו לחשב את ערכיהם של ביטויים כגון a / b ו-a % b, חשוב לוודא כי ערך המשתנה b אינו 0, ולפני חישוב הביטוי $\sqrt{x}$ יש לוודא כי המשתנה x מכיל ערך חיובי או אפס (0).

דוגמה

בדוגמה זו נכתוב אלגוריתם הפולט את אורך היתר במשולש ישר-זווית שאת אורכי ניצביו מזין המשתמש. בהינתן אורכי הניצבים a ו-b במשולש ישר-זווית, ניתן למצוא את אורך היתר c על ידי המשוואה $c = \sqrt{a^2 + b^2}$.

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) c ← $\sqrt{a^2 + b^2}$

(4) הדפס את הערך של c

אם יוזנו, למשל, המספרים 3 ו-4, ייפלט המספר 5.

האלגוריתם שנכתוב בדוגמה זו יפלוט את שטחו ואת היקפו של מלבן שאת אורכי צלעותיו הזין המשתמש.

(1) קלוט מספר למשתנה length

(2) קלוט מספר למשתנה width

(3) area ← length * width

(4) הדפס את הערך של area

(5) perimeter ← 2 * (length + width)

(6) הדפס את הערך של perimeter

אם הוזנו, למשל, המספרים 5 ו-10, ייפלטו המספרים 50 ו-30.

תנאים

הוראות האלגוריתמים שהוצגו עד כה בוצעו תמיד על פי סדר הופעתן. במקרים רבים נרצה לבצע קבוצת הוראות כלשהי אם תנאי מסוים מתקיים. בסעיף זה נראה כיצד לעשות זאת.

ביצוע מותנה באלגוריתמים המיועדים לאדם

נניח שעלינו לתכנן אלגוריתם המיועד לאדם, הבודק האם יש צורך בקניית כיכר לחם. הוראות אלגוריתם כזה ייראו כך:

(1) אם אין בבית כיכר לחם אזי

(1.1) קנה כיכר לחם

הוראה (1) מורכבת מהמילים "אם... אזי" ומתנאי המופיע ביניהן. הוראה (1.1) היא הוראה אלגוריתמית רגילה. אם התנאי המופיע בהוראה (1) מתקיים (במקרה זה, אם אין בבית כיכר לחם), תבוצע הוראה (1.1) (במקרה זה, "קנה כיכר לחם"). נהוג לומר כי הוראה (1) היא **הוראת תנאי** וכי הוראה (1.1) שייכת לה.

בדוגמה זו נכתוב אלגוריתם עבור אדם המעוניין לקנות כריך במסעדת מזון מהיר. נדגים את הוראת התנאי בכך שניקח בחשבון אפשרות שלאדם אין די כסף לקניית הכריך.

(1) מנה את הכסף שבידיך

(2) מצא כמה עולה כריך

(3) אם בידיך די כסף אזי

(3.1) שלם למוכר

(3.2) קח את הכריך

לעיתים קרובות נרצה לבצע קבוצת הוראות אם התנאי שבהוראת התנאי **אינו** מתקיים. הדבר מודגם באלגוריתם הבא:

(1) אם הכנת את שיעורי הבית למחר אזי

(1.1) צפה בטלוויזיה

(2) אחרת

(2.1) הכן את שיעורי הבית למחר

הוראה (1) בודקת האם הוכנו שיעורי הבית. אם תנאי זה מתקיים, תבוצע ההוראה האלגוריתמית השייכת להוראת התנאי, כלומר, הוראה (1.1). הוראה (2) מורכבת אך ורק מהמילה "אחרת", והוראה (2.1) השייכת לה תבוצע אם התנאי שהופיע בהוראה (1) לא התקיים.

שים לב!

הוראה מסוג "אם... אזי... אחרת" תמיד תבצע אחת (ואחת בלבד) משתי קבוצות ההוראות השייכות לה. בדוגמה זו, הוראה אחת בלבד מההוראות (1.1) ו-(2.1) תתבצע.

דוגמה

בדוגמה זו נכתוב אלגוריתם המנחה אדם כיצד לחייג לביתו מטלפון ציבורי.

(1) הרם את שפופרת הטלפון

(2) אם בידך טלכרט אזי

(2.1) הכנס את הטלכרט לחריץ המתאים

(2.2) חייג את המספר המבוקש

(3) אחרת

(3.1) חייג 142 לשיחת גובייגא

(3.2) אמור את שמך

(3.3) חייג את המספר המבוקש

באלגוריתם זה משויכות שתי הוראות להוראת התנאי (2). אם התנאי המופיע בהוראה זו מתקיים, היינו, אם בידי המחייג כרטיס טלכרט, יבוצעו הוראות (2.1) ו-(2.2). אם תנאי זה אינו מתקיים, יבוצעו ההוראות השייכות להוראה (3), והן הוראות (3.1), (3.2) ו-(3.3).

ביצוע מותנה באלגוריתמים המיועדים למחשב

בעוד שניתן להשתמש בתנאים המיועדים לאדם, לרוב נעסוק בתנאים ממוחשבים. הוראת תנאי אלגוריתמית המורה למחשב לבדוק תנאי מסוים, דומה להוראת התנאי המיועדת לאדם. ההבדל בין השתיים הוא שתנאי ממוחשב תמיד מכיל פעולת יחס כלשהי. הנה, למשל, אלגוריתם הקולט מספר מהמשתמש ומוצא האם הוא מספר שלילי:

(1) קלוט מספר למשתנה number

(2) אם number < 0 אזי

(2.1) הדפס את ההודעה "המספר שהוזן הוא שלילי"

התנאי בהוראה (2) בודק האם ערך המשתנה number קטן מ-0. אם תנאי זה מתקיים, אזי number הוא מספר שלילי, והוראה (2.1) מבוצעת. פעולת הבדיקה "קטן מ..." מיוצגת על ידי הסימן '<'.

נהוג להשתמש בסימני היחס הבאים בעת כתיבת תנאים ממוחשבים:

סימן	משמעות	דוגמה
<	קטן מ...	a < b (האם ערך המשתנה a קטן משל b?)
>	גדול מ...	a > b (האם ערך המשתנה a גדול משל b?)
≤	קטן או שווה ל...	a ≤ b (האם ערך המשתנה a קטן או שווה לשל b?)
≥	גדול או שווה ל...	a ≥ b (האם ערך המשתנה a גדול או שווה לשל b?)
=	שווה ל...	a = b (האם ערך המשתנה a שווה לשל b?)
<>	שונה מ...	a <> b (האם ערך המשתנה a שונה משל b?)

תנאי שמתקיים מכונה **תנאי אמת** (true condition), בעוד שתנאי שאינו מתקיים מכונה **תנאי שקר** (false condition). הטבלה הבאה מציגה מספר תנאי אמת ושקר:

תנאי אמת	תנאי שקר
1 > 10 - 20	3 + 1 = 2 - 1
-2 ≥ -2	5^2 ≤ 3^2
10 <> 7	3 * (2 + 4) = 10
$2 = \sqrt{3+1}$	20 <> 40 / 2

דוגמה

האלגוריתם שנכתוב בדוגמה זו יקלוט שלושה מספרים מהמשתמש ויפלוט את המספר הגדול ביותר מביניהם, תוך שימוש בהוראות תנאי.

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

(4) $x \leftarrow a$

(5) אם $b > x$ אזי

(5.1) $x \leftarrow b$

(6) אם $c > x$ אזי

(6.1) $x \leftarrow c$

(7) הדפס את הערך של x

הוראה (4) מציבה את ערך המשתנה a במשתנה x. הוראה (5) משווה בין ערכי המשתנים b ו-x. אם b > x אז גם b > a, ובמקרה זה מבוצעת הוראה (5.1), המציבה את ערך המשתנה b במשתנה x. לאחר ביצוע הוראות אלו יכיל המשתנה x את הערך הגדול מבין ערכי המשתנים a ו-b. הוראה (6) משווה בין ערכי המשתנים c ו-x. אם c > x אז גם c > b ו-a > c, ובמקרה זה מבוצעת הוראה (6.1), המציבה את ערך המשתנה c במשתנה x. באופן זה, כאשר נפלט ערך המשתנה x בהוראה (7), נפלט הערך הגדול מבין הערכים שהוזנו.

גם באלגוריתמים המיועדים למחשב ניתן לבצע קבוצת הוראות אם התנאי שבהוראת התנאי **אינו** מתקיים. הדבר מודגם באלגוריתם הבא:

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) אם a > b אזי

(3.1) הדפס את ההודעה "a גדול מ-b"

(4) אחרת

(4.1) הדפס את ההודעה "a קטן מ-b או שווה לו"

הוראה (3) בודקת האם ערך המשתנה a גדול מערך המשתנה b. אם תנאי זה מתקיים, תבוצע הוראה (3.1), הפולטת הודעה מתאימה. הוראה (4) מורכבת אך ורק מהמילה "אחרת", והוראה (4.1) השייכת לה תבוצע אם התנאי שהופיע בהוראה (3) לא התקיים.

דוגמה

האלגוריתם הבא שנכתוב יורה למחשב לקלוט מספר מהמשתמש ולפלוט את שורשו הריבועי. לפני הוצאת השורש הריבועי נוודא שהמספר שהוזן אינו שלילי, שהרי אין להוציא שורש ריבועי ממספר שלילי. אם המספר שלילי, נפלוט הודעת שגיאה.

(1) קלוט מספר למשתנה number

(2) אם number < 0 אזי

(2.1) הדפס את ההודעה "שגיאה! אין להוציא שורש ריבועי ממספר שלילי"

(3) אחרת

(3.1) square ← $\sqrt{number}$

(3.2) הדפס את הערך של square

התנאי המורכב וגם

לעיתים ברצוננו לבדוק האם מספר תנאים מתקיימים במקביל. בחן, למשל, את האלגוריתם הפשוט שהוצג בתחילת סעיף זה:

(1) אם אין בבית כיכר לחם אזי

(1.1) קנה כיכר לחם

אלגוריתם זה נכון לגמרי, אך הבה נהפכו לנכון יותר. מאחר שסופרמרקטים סגורים בשעות מסוימות של היום, לפני שנקנה כיכר לחם עלינו לוודא שהסופרמרקט פתוח. אלגוריתם המבצע פעולה זו מוצג להלן:

(1) אם אין בבית כיכר לחם וגם הסופרמרקט פתוח אזי

(1.1) קנה כיכר לחם

אלגוריתם זה שונה מהאלגוריתם הקודם לו בשל תת-ההוראה "וגם...". תת-הוראה זו מוסיפה להוראה (1) תנאי נוסף, והיא מביאה לכך שמשפט (1.1) יתבצע רק אם **שני** התנאים שבהוראה (1) התקיימו.

דוגמה

בדוגמה זו נכתוב אלגוריתם הקולט שלושה מספרים ומוצא האם הם הוזנו בסדר עולה.

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

(4) אם $b > a$ וגם $c > b$ אזי

(4.1) הדפס את ההודעה "המספרים הוזנו בסדר עולה"

(5) אחרת

(5.1) הדפס את ההודעה "המספרים לא הוזנו בסדר עולה"

התנאים של הוראה (4) נבדקים באמצעות תת-ההוראה "וגם", ולכן הוראה (4.1) תבוצע רק אם שניהם מתקיימים. הוראה (5.1) תבוצע אם אחד מהתנאים של הוראה (4) (או שניהם) לא התקיים.

במחשבה שנייה...

האם יכולנו להחליף את תנאיה של הוראה (4) בתנאי a < b < c?

שלא כמו במתמטיקה, התנאי a < b < c נחשב שגוי בשפות תכנות רבות. בעת פיתוח אלגוריתמים נשתדל להשתמש בתחביר פשוט ככל שניתן, כדי שיתאים לכמה שיותר שפות תכנות. לכן נימנע מכתיבת ביטויים מורכבים כגון a < b < c.

ניתן להשתמש ביותר מתנאי "וגם..." אחד בהוראת תנאי. האלגוריתם שהצגנו קודם, למשל,

(1) אם אין בבית כיכר לחם וגם הסופרמרקט פתוח אזי

(1.1) קנה כיכר לחם

ניתן לכתיבה באופן מדויק יותר כך:

(1) אם אין בבית כיכר לחם וגם הסופרמרקט פתוח וגם בידך די כסף לקניית כיכר לחם אזי

(1.1) קנה כיכר לחם

כדי לשלב תנאי נוסף בהוראת התנאי, הוספנו אליה עוד תת-הוראה "וגם...". לכן, הוראה (1.1) תבוצע רק אם **שלושת** התנאים המופיעים בהוראת התנאי מתקיימים.

דוגמה

האלגוריתם שנכתוב בדוגמה זו יקלוט מהמשתמש ארבעה מספרים וימצא אם הם שווים זה לזה.

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

(4) קלוט מספר למשתנה d

(5) אם a = b וגם b = c וגם c = d אזי

(5.1) הדפס את ההודעה "המספרים שווים זה לזה"

(6) אחרת

(6.1) הדפס את ההודעה "המספרים שונים זה מזה"

בהוראה (5) נבדקים שלושה תנאים: a = b, b = c ו-c = d. תנאים אלה נבדקים על ידי תת-ההוראה "וגם...", ולכן משפט (5.1) יבוצע רק אם שלושתם מתקיימים, כלומר, אם ארבעת המספרים שהוזנו שווים זה לזה. הוראה (6.1) תתבצע אם אחד מהתנאים שבהוראה (5) לא התקיים, כלומר, אם אחד מהמספרים שהוזנו שונה מהמספרים האחרים.

התנאי המורכב או

הוראת תנאי המכילה תת-הוראות "וגם..." תבצע את ההוראות השייכות לה אם כל התנאים שבה מתקיימים. לעיתים נרצה לבדוק אם אחד מהתנאים מתקיים, ולא בהכרח כולם. למשל, נהג מכונית יעצור את מכוניתו במקרים רבים; בין היתר, אם הוא מתקרב לרמזור אדום או לתמרור "עצור". אלגוריתם המורה לנהג מתי לעצור את מכוניתו ייראה כך:

(1) אם לפניך רמזור אדום או לפניך תמרור "עצור" אזי

(1.1) עצור את מכוניתך

באלגוריתם זה השתמשנו בתת-ההוראה "או...", ולכן הוראה (1.1) תבוצע אם **אחד** משני התנאים שבהוראה (1) מתקיים. ייתכן כי שני התנאים שבהוראה (1) יתקיימו; גם במקרה זה תתבצע הוראה (1.1). הוראת תנאי המורכבת מתת-ההוראות "או..." תבצע את ההוראות השייכות לה אם אחד או יותר מהתנאים שלה התקיימו.

דוגמה

בדוגמה זו נכתוב אלגוריתם הקולט שני מספרים ומוצא האם אחד מהמספרים מתחלק במספר השני ללא שארית (כלומר, האם אחד מהמספרים הוא כפולה של השני).

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) אם a % b = 0 או b % a = 0 אזי

(3.1) הדפס את ההודעה "אחד מהמספרים מתחלק בשני ללא שארית"

(4) אחרת

(4.1) הדפס את ההודעה "אף אחד מהמספרים אינו מתחלק בשני ללא שארית"

הוראה (3) בודקת האם אחד מהמספרים מתחלק בשני ללא שארית. אם מתקיים אחד משני התנאים של הוראה זו, תתבצע הוראה (3.1), הפולטת הודעה מתאימה. אם שני התנאים אינם מתקיימים, תתבצע הוראה (4.1), הפולטת אף היא הודעה מתאימה.

שים לב!

הביטויים המופיעים בהוראה (3), a % b ו-b % a, מחשבים שארית של חלוקת מספר בערכי המשתנים a ו-b. מכיון שאסור לחשב שארית של חלוקת מספר ב-0, על ערכי המשתנים a ו-b להיות שונים מ-0. בתוכנית זו לא וידאנו זאת; נעשה זאת בהמשך.

ניתן להשתמש ביותר מתנאי "או..." אחד בהוראת תנאי. האלגוריתם שהצגנו קודם, למשל,

(1) אם לפניך רמזור אדום או לפניך תמרור "עצור" אזי

(1.1) עצור את מכוניתך

ניתן לכתיבה באופן מדויק יותר כך:

(1) אם לפניך רמזור אדום או לפניך תמרור "עצור" או כלי רכב נוסע ברחוב אליו אתה נכנס אזי

(1.1) עצור את מכוניתך

כדי לשלב תנאי נוסף בהוראת התנאי, הוספנו אליו עוד תת-הוראה "או...". לכן הוראה (1.1) תבוצע אם אחד או יותר משלושת התנאים של הוראה (1) מתקיימים.

דוגמה

בדוגמה זו נכתוב אלגוריתם הקולט שלושה מספרים ומוצא האם מספר מסוים הוזן יותר מפעם אחת.

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

(4) אם a = b או a = c או b = c אזי

(4.1) הדפס את ההודעה "אותו מספר הוזן יותר מפעם אחת"

(5) אחרת

(5.1) הדפס את ההודעה "שלושת המספרים שונים זה מזה"

התנאים הנבדקים בהוראה (4) משולבים באמצעות תת-ההוראה "או...", ולכן הוראה (4.1) תתבצע אם אחד מהם התקיים, כלומר, אם הוזנו שני מספרים זהים. אם אף אחד משלושת התנאים לא התקיים, אזי הוזנו שלושה מספרים שונים, והוראה (5.1) תתבצע.

הוראות תנאי מקוננות

לעיתים יש צורך לשייך הוראת תנאי להוראת תנאי נוספת. בחן, למשל, את האלגוריתם שהצגנו קודם לקניית לחם בסופרמרקט:

(1) אם אין בבית כיכר לחם וגם הסופרמרקט פתוח וגם בידך די כסף לקניית כיכר לחם אזי

(1.1) קנה כיכר לחם

נוכל לשפר את האלגוריתם אם נבדוק ביתר קפדנות את התנאים הדרושים לקניית כיכר הלחם. אם שלושת התנאים של הוראה (1) מתקיימים, ברור כי ניתן לקנות את כיכר הלחם. אך גם אם רק שני התנאים הראשונים מתקיימים, היינו, אם חסר בביתנו כיכר לחם, הסופרמרקט פתוח, אך אין ברשותנו מספיק כסף, עדיין נוכל לקנות את כיכר הלחם אם נשיג את הכסף הדרוש. לכן, נוכל לשפר את האלגוריתם ולכותבו כך:

(1) אם אין בבית כיכר לחם וגם הסופרמרקט פתוח אזי

(1.1) אם בידך די כסף לקניית כיכר לחם אזי

(1.1.1) קנה כיכר לחם

(1.2) אחרת

(1.2.1) השג את הכסף הדרוש

(1.2.2) קנה כיכר לחם

הוראות (1.1) ו-(1.2) שייכות להוראת התנאי (1), ולכן הן מתבצעות רק אם בביתנו חסר כיכר לחם והסופרמרקט פתוח. הוראה (1.1) בודקת אם ברשותנו מספיק כסף; אם כן, הוראה (1.1.1) מתבצעת, ואנו קונים את כיכר הלחם. אם לא, הוראות (1.2.1) ו-(1.2.2) מתבצעות, ואנו משיגים את הכסף הדרוש ואחר-כך קונים את כיכר הלחם. אם שני התנאים של הוראה (1) מתקיימים, כיכר הלחם נקנה – אך האופן בו הוא נקנה תלוי בתנאי הנבדק בהוראה (1.1).

מבנה זה של שילוב הוראת תנאי בהוראת תנאי אחרת מכונה **קינון** (nesting).

שים לב כי אלגוריתם זה אינו זהה לאלגוריתם הבא:

(1) אם אין בבית כיכר לחם וגם הסופרמרקט פתוח וגם בידך די כסף לקניית כיכר לחם אזי

(1.1) קנה כיכר לחם

(2) אחרת

(2.1) השג את הכסף הדרוש

(2.2) קנה כיכר לחם

הסיבה היא, שבאלגוריתם האחרון כיכר הלחם נקנה בכל מצב, בעוד שבאלגוריתם הקודם לו, כיכר הלחם נקנה רק אם אין בביתנו כיכר לחם והסופרמרקט פתוח.

נדגים שימוש בהוראות תנאי מקוננות על ידי שיפור האלגוריתם שהצגנו מקודם, הקולט מהמשתמש שני מספרים ומוצא האם אחד מהם מתחלק בשני ללא שארית. לשם כך חישבנו את ערכי הביטויים a % b ו-b % a, אך מאחר שאסור לחשב שארית של חלוקת מספר ב-0, עלינו לוודא כי ערכי המשתנים a ו-b שונים מ-0 לפני בדיקת הביטויים. באלגוריתם הקודם לא וידאנו זאת, ולכן הוא אינו נכון לגמרי. עתה נשפרו על ידי בדיקת תקינות הקלטים.

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) אם a = 0 או b = 0 אזי

(3.1) הדפס את ההודעה "קלט לא חוקי!"

(4) אחרת

(4.1) אם a % b = 0 או b % a = 0 אזי

(4.1.1) הדפס את ההודעה "אחד מהמספרים מתחלק בשני ללא שארית"

(4.2) אחרת

(4.2.1) הדפס את ההודעה "אף אחד מהמספרים אינו מתחלק בשני ללא שארית"

הוראה (3.1) פולטת הודעה מתאימה אם הוזן קלט לא חוקי (כלומר, המספר 0). הוראה (4) וההוראות השייכות לה תתבצענה אם ערכי המשתנים a ו-b שונים מ-0. ההוראות השייכות להוראה (4) מוצאות האם אחד מהמספרים שהוזנו מתחלק בשני ללא שארית, ופולטות הודעה מתאימה.

לולאות

לעיתים קרובות נרצה לבצע קבוצת הוראות כלשהי מספר פעמים. הוראות אלגוריתמיות המאפשרות לעשות זאת נקראות **לולאות** (loops). קיימים מספר סוגי לולאות, ובסעיף זה נכירם.

בצע ... פעמים

אם ברצוננו לבצע קבוצת הוראות כלשהי מספר פעמים, נשייך אותה להוראה הנראית כך:

בצע x פעמים

כאשר x הוא ערך קבוע או שם של משתנה. למשל, כדי לפלוט את המילה "שלום" עשר פעמים, נכתוב לולאה כזו:

(1) בצע 10 פעמים

(1.1) הדפס את ההודעה "שלום"

אם נרצה לפלוט מילה זו מספר פעמים הנקבע על ידי המשתמש, נכתוב את ההוראות הבאות:

(1) קלוט מספר למשתנה n

(2) בצע n פעמים

(2.1) הדפס את ההודעה "שלום"

הוראה (1) קולטת מהמשתמש מספר ומציבה אותו במשתנה n. הוראה (2) היא לולאה המתבצעת n פעמים, כלומר, מספר פעמים השווה למספר שהזין המשתמש. הוראה (2.1) מבוצעת בכל פעם שהלולאה מתבצעת, ולכן נפלטת המילה "שלום" n פעמים.

שים לב!

משתנה כגון n, באמצעותו מבצעת הלולאה את ההוראות השייכות לה מספר פעמים מסוים, מכונה **משתנה מונה**.

דוגמה

בדוגמה זו נכתוב אלגוריתם הקולט מספר טבעי מהמשתמש ופולט את כל המספרים הטבעיים עד אליו.

(1) קלוט מספר למשתנה n

(2) $i \leftarrow 1$

(3) בצע n פעמים

(3.1) הדפס את הערך של i

(3.2) $i \leftarrow i + 1$

הוראה (1) קולטת מספר מהמשתמש ומציבה אותו במשתנה n. מספר זה יקבע כמה מספרים טבעיים ייפלטו. הוראה (3) היא לולאה המתבצעת n פעמים. בכל פעם שלולאה זו מתבצעת, מבוצעות הוראה (3.1) הפולטת את ערך המשתנה i, והוראה (3.2) המעלה אותו ב-1.

הוראה (2) מציבה את הערך 1 במשתנה i. לכן, בפעם הראשונה שהוראה (3.1) מתבצעת ערך i הוא 1. בפעם השנייה שהוראה זו מתבצעת ערכו הוא 2 (בגלל הוראת ההשמה (3.2)), בפעם השלישית ערכו 3, וכדומה. הוראה זו מתבצעת n פעמים, ולכן נפלטים המספרים הבאים:

1, 2, 3, ... , n

ואלה הם כל המספרים הטבעיים עד ל-n.

דוגמה

האלגוריתם הבא שנכתוב יקלוט מספר ציונים מהמשתמש ויפלוט את הממוצע שלהם. מספר הציונים שיוזנו ייקבע על ידי המשתמש.

(1) קלוט מספר למשתנה n

(2) sum ← 0

(3) בצע n פעמים

(3.1) קלוט מספר למשתנה grade

(3.2) sum ← sum + grade

(4) average ← sum / n

(5) הדפס את הערך של average

באלגוריתם זה אנו נעזרים בארבעה משתנים. במשתנה n יוצב הערך הראשון שיזין המשתמש, שהוא מספר הציונים שיוזנו. המשתנה sum יכיל את סכום הציונים שיוזנו, והממוצע שלהם יוצב במשתנה average. כל ציון ייקלט בנפרד למשתנה grade.

הוראות (3.1) ו-(3.2) מתבצעות n פעמים. בכל פעם שהוראה (3.1) מתבצעת נקלט מספר מהמשתמש, ומספר זה מוצב במשתנה grade. לאחר כל הזנת ציון מוסיפה הוראה (3.2) את הציון האחרון שהוזן לערך המשתנה sum. המשתנה sum מכיל את הערך 0 לפני ביצוע הלולאה (ההשמה מתבצעת בהוראה (2)), ולכן בסיום הלולאה הוא יכיל את סכום כל הציונים שהוזנו.

כדי לחשב את הממוצע של n הציונים עלינו לחלק את סכומם ב-n, והדבר נעשה בהוראה (4), המציבה במשתנה average את מנת ערך המשתנה sum בערך המשתנה n. הוראה (5) פולטת את ערך המשתנה average, שהוא ממוצע הציונים שהוזנו.

זהירות!

כזכור, אין לבצע חלוקה של מספר ב-0. באלגוריתם זה, אם הערך הראשון שהוזן היה 0, ערך המשתנה n יהיה 0 כאשר נחלק בו את sum. פעולה זו הינה מסוכנת, אך נעלים ממנה עין ונניח שהערך 0 אינו קלט אפשרי בהוראה (1), שהרי במקרה זה לא ייקלט אף ציון, והאלגוריתם יהיה חסר משמעות.

דוגמה

האלגוריתם שנכתוב בדוגמה זו יקלוט עשרה מספרים מהמשתמש ויפלוט את המספר הגדול ביותר מביניהם.

(1) קלוט מספר למשתנה biggest

(2) בצע 9 פעמים

(2.1) קלוט מספר למשתנה num

(2.2) אם num > biggest אזי

(2.2.1) biggest ← num

(3) הדפס את הערך של biggest

הוראה (1) קולטת מהמשתמש מספר ומציבה אותו במשתנה biggest. עתה עלינו לקלוט תשעה מספרים נוספים, ולכן הלולאה המוגדרת בהוראה (2) מתבצעת תשע פעמים. בכל פעם שלולאה זו מתבצעת מבוצעות הוראות (2.1) ו-(2.2). הוראה (2.1) קולטת מהמשתמש מספר ומציבה אותו במשתנה num. הוראה (2.2) בודקת האם המספר האחרון שהוזן גדול מהמספר שבמשתנה biggest. אם כן, מתבצעת הוראה (2.2.1), המציבה במשתנה biggest את המספר האחרון שהוזן. באופן זה, לאחר כל פעם שהלולאה מתבצעת יכיל המשתנה biggest את המספר הגבוה ביותר שהוזן עד כה. כאשר תסתיים הלולאה, יכיל המשתנה biggest את המספר הגבוה ביותר מכל המספרים שהוזנו, ובהוראה (3) ייפלט ערכו.

עבור ... מ-... עד ... בצע

נכיר עתה לולאה נוספת בה נהוג להשתמש כדי לבצע קבוצת הוראות מספר פעמים כלשהו. לולאה זו שונה מהלולאה "בצע ... פעמים" בכך שהיא מעלה ערך משתנה כלשהו ב-1 לאחר כל פעם שההוראות השייכות לה מתבצעות.

הנה דוגמה פשוטה לשימוש בה:

(1) עבור i מ-1 עד 10 בצע

(1.1) הדפס את הערך של i

הלולאה מתבצעת בשלבים הבאים:

1. הוראה (1) מציבה במשתנה i את הערך 1
2. הוראה (1.1) מתבצעת
3. ערך המשתנה i עולה ב-1

שלבים 2 ו-3 מבוצעים שוב ושוב, עד שערך i הוא 10 (כולל). כלומר, בפעם הראשונה שהוראה (1.1) מתבצעת, ערך המשתנה i הוא 1; בפעם השנייה ערכו הוא 2, בפעם השלישית ערכו 3, וכן הלאה – עד לפעם העשירית, בה ערך המשתנה i הוא 10. הוראה (1.1) פולטת בכל פעם את ערך המשתנה i, ולכן מוצגים המספרים הבאים:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

המשתנה המונה בלולאה זו הוא i, אך ניתן לבחור כל משתנה כמשתנה מונה. בלולאה זו תחום ערכי המשתנה המונה עבורם מתבצעת הלולאה הוא מ-1 עד 10, אך ניתן לשנות תחום זה. למשל, כדי לפלוט את המספרים השלמים מ-20 עד 30, נכתוב לולאה כזו:

(1) עבור n מ-20 עד 30 בצע

(1.1) הדפס את הערך של n

בתחילה מוצב במשתנה n הערך 20, והוראה (1.1) מתבצעת עד שערך n יגיע ל-30. לכן ייפלטו המספרים הבאים:

20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

שים לב כי בכל לולאה מסוג זה ערך המשתנה המונה עולה ב-1 לאחר כל פעם שקבוצת ההוראות השייכת ללולאה מתבצעת. תכונה זו של הלולאה היא קבועה – ערך המשתנה המונה תמיד עולה ב-1.

זהירות!

אין לשנות את ערך המשתנה המונה מתוך הלולאה. הלולאה הבאה, למשל:

(1) עבור i מ-1 עד 10 בצע

i ← i * 2 (1.1)

נחשבת ללולאה שגויה.

בדוגמה זו נכתוב אלגוריתם הקולט מהמשתמש שני מספרים ופולט את כל המספרים השלמים הנמצאים ביניהם ומתחלקים ב-3 ללא שארית.

(1) קלוט מספר למשתנה from

(2) קלוט מספר למשתנה to

(3) עבור i מ-from עד to בצע

(3.1) אם i % 3 = 0 אזי

(3.1.1) הדפס את הערך i

הוראות (3.1) ו-(3.1.1) פולטות את ערך המשתנה i אם הוא מתחלק ב-3 ללא שארית. הוראות אלו מבוצעות על ידי הלולאה המוגדרת בהוראה (3), כאשר ערך המשתנה המונה i נע בין כל המספרים השלמים בין from ל-to. באופן זה נפלטים כל המספרים המתחלקים ב-3 ללא שארית הנמצאים בין המספרים שהזין המשתמש.

כל עוד ... בצע

לעיתים לא ידוע מספר הפעמים שיש לבצע קבוצת הוראות כלשהי. במקרה זה, נרצה לבצע את קבוצת ההוראות כל עוד תנאי מסוים מתקיים. נפתח באלגוריתם פשוט המיועד לאדם, המורה לו מתי עליו לאכול:

(1) כל עוד אתה רעב בצע

(1.1) אכול סנדוויץ'

בתחילת ביצוע האלגוריתם נבדקת הוראה (1). הוראה זו היא לולאה, ומופיע בה תנאי המכונה **תנאי לולאה**. אם תנאי הלולאה מתקיים, תבוצענה ההוראות השייכות ללולאה. לאחר ביצוען, ייבדק שוב תנאי הלולאה. אם הוא עדיין מתקיים, ההוראות השייכות ללולאה תבוצענה שוב, וחוזר חלילה. הלולאה תסתיים כאשר תנאי הלולאה לא יתקיים.

תנאי הלולאה באלגוריתם זה מתקיים כל עוד האדם רעב. במקרה זה, תבוצע ההוראה השייכת ללולאה, המורה לאדם לאכול סנדוויץ'. לאחר ביצוע הוראה זו שוב ייבדק תנאי הלולאה, ואם הוא יתקיים, ההוראה השייכת ללולאה תבוצע שוב – וחוזר חלילה.

שים לב!

תנאי הלולאה נבדק לפני שהוראות הלולאה מתבצעות. אם תנאי הלולאה הינו תנאי שקרי תמיד, הוראות הלולאה לא תתבצענה אפילו פעם אחת.

דוגמה

בדוגמה זו נכתוב אלגוריתם הקולט שני מספרים מהמשתמש, כאשר שניהם טבעיים והראשון קטן מהשני, ופולט את סכום המספרים הטבעיים הנמצאים ביניהם (כולל הם עצמם).

(1) קלוט מספר למשתנה from

(2) קלוט מספר למשתנה to

(3) sum ← 0

(4) כל עוד from ≤ to בצע

(4.1) sum ← sum + from

(4.2) from ← from + 1

(5) הדפס את הערך של sum

האלגוריתם מתחיל בקליטת שני מספרים ובהצבתם במשתנים from ו-to. סכום המספרים הטבעיים הנמצאים בין מספרים אלה יחושב במהלך הלולאה ולבסוף ייפלט.

הוראה (4.1) מוסיפה למשתנה sum את ערך המשתנה from, והוראה (4.2) מעלה את ערך המשתנה from ב-1. שתי הוראות אלו מתבצעות כל עוד ערך המשתנה from קטן או שווה לערך המשתנה to, ולכן, כשתיפסק הלולאה, יכיל המשתנה sum את ערך הביטוי הבא:

$$from + (from + 1) + (from + 2) + (from + 3) + \ \ldots \ + to$$

וזהו סכום המספרים הטבעיים הנמצאים בין from ל-to.

דוגמה

האלגוריתם שנכתוב בדוגמה זו יקלוט מספר טבעי מהמשתמש ויפלוט את ספרותיו בסדר הפוך. למשל, אם יוזן המספר 3105, ייפלטו ספרותיו כך: 5013.

(1) קלוט מספר למשתנה num

(2) כל עוד num > 0 בצע

(2.1) digit ← num % 10

(2.2) הדפס את הערך של digit

(2.3) num ← num - digit

(2.4) num ← num / 10

כדי לפלוט את ספרות המספר בסדר הפוך נשתמש באופרטור השארית. ניעזר בביטוי num % 10, המחזיר תמיד את הספרה הימנית ביותר במספר שבמשתנה num. הסיבה לכך היא, שאופרטור השארית מחזיר את שארית חלוקת המספר השמאלי בימני, והשארית של חלוקת מספר כלשהו ב-10 היא ספרת האחדות של אותו מספר.

הוראה (1) קולטת מספר מהמשתמש ומציבה אותו במשתנה num. הוראה (2) היא לולאה המתבצעת כל עוד num > 0. בכל פעם שלולאה זו מתבצעת, הוראה (2.1) מציבה במשתנה digit את הספרה הימנית ביותר במשתנה num והוראה (2.2) פולטת ספרה זו.

לאחר שטיפלנו בספרה הימנית ביותר, ברצוננו לטפל בספרה שמשמאלה. נוכל לעשות זאת בקלות אם נשמיט מהמספר את ספרתו הימנית ביותר. באופן זה, בפעם הבאה שהלולאה תתבצע, הביטוי num % 10 יחזיר את הספרה שמשמאל לספרה בה טיפלנו בפעם זו שהלולאה התבצעה. כדי להשמיט את ספרתו הימנית ביותר של המספר שבמשתנה num נפחית ממנו את ספרת האחדות שלו ונחלקו ב-10. זאת עושות הוראות (2.3) ו-(2.4).

כאשר המשתנה num יכיל מספר חד-ספרתי, הוראה (2.1) תציב במשתנה digit ערך השווה לו. לאחר שייפלט ערך זה, המהווה את הספרה האחרונה שנותרה במספר, ההוראה num ← num - digit תציב במשתנה num את הערך 0, ומאחר שתנאי הלולאה לא יתקיים, האלגוריתם יסתיים.

בצע ... עד ש ...

בדומה ללולאה "כל עוד ... בצע", הלולאה "בצע ... עד ש..." מבצעת קבוצת הוראות בהתאם לקיומו של תנאי מסוים. ההבדל בין השתיים הוא, שהלולאה "כל עוד ... בצע" בודקת את התנאי ואחר כך מבצעת את קבוצת ההוראות (וחוזר חלילה), בעוד שהלולאה "בצע ... עד ש..." מבצעת קודם את קבוצת ההוראות ורק אחר-כך בודקת את התנאי. הנה דוגמה פשוטה לשימוש בלולאה זו:

(1) בצע

(1.1) קלוט מספר למשתנה x

(2) עד ש $x > 0$

באלגוריתם זה, הוראה (1) מתחילה את הלולאה, הוראה (1.1) מהווה את גוף הלולאה, והוראה (2) מסיימת את הלולאה. לולאה זו קולטת מהמשתמש מספרים עד שמוזן מספר חיובי, באופן הבא: ראשית מבוצעת הוראה (1.1); אחר-כך נבדק התנאי שבהוראה (2). אם תנאי זה מתקיים, הלולאה תסתיים; אם לא, הוראה (1.1) תבוצע בשנית, התנאי ייבדק בשנית, וחוזר חלילה.

הטבלה הבאה מציגה את ההבדלים בין הלולאות "כל עוד ... בצע" ו"בצע ... עד ש ...":

הבדל	"כל עוד ... בצע"	"בצע ... עד ש ..."
בדיקת התנאי	לפני ביצוע קבוצת ההוראות השייכת ללולאה	אחרי ביצוע קבוצת ההוראות השייכת ללולאה
מספר הפעמים שהוראות הלולאה מתבצעות	אפס פעמים או יותר	פעם אחת לפחות
סיום הלולאה	כאשר התנאי אינו מתקיים	כאשר התנאי מתקיים

דוגמה

בדוגמה זו נפתח אלגוריתם הקולט עשרה מספרים מהמשתמש ופולט את סכום שורשיהם הריבועיים. לשם כך נשתמש בשתי לולאות: בלולאת "עבור ... מ-... עד ... בצע" ובלולאת "בצע ... עד ש...".

(1) $sum \leftarrow 0$

(2) עבור i מ-1 עד 10 בצע

(2.1) בצע

(2.1.1) קלוט מספר למשתנה num

(2.2) עד ש $num \geq 0$

(2.3) $sum \leftarrow sum + \sqrt{num}$

(3) הדפס את הערך של sum

הוראות (2.1) ו-(2.2) מהוות לולאה מסוג "בצע ... עד ש...", הקולטת מספרים מהמשתמש שוב ושוב, עד שמוזן מספר אי-שלילי. כאשר מוזן מספר כזה, מתבצעת הוראה (2.3), המוסיפה לערך המשתנה sum את שורשו הריבועי של המספר האחרון שהוזן. שים לב כי הלולאה "בצע ... עד ש..." מוודאת שלא מחושב שורש ריבועי של מספר שלילי.

הוראות (2.1)-(2.3) שייכות ללולאה החיצונית המוגדרת בהוראה (2), ולכן הן מתבצעות עשר פעמים. באופן זה מחושב סכום שורשיהם הריבועיים של עשרה מספרים אי-שליליים. לאחר שמסתיימת הלולאה החיצונית מתבצעת הוראה (3), הפולטת את סכום השורשים הריבועיים של המספרים האי-שליליים שהוזנו.

מבנה זה של שילוב לולאה אחת בלולאה אחרת מכונה **קינון לולאות**.

אלגוריתמים ומערכים

מערך (array) הוא טיפוס נתונים המכיל מספר משתנים. לעת עתה נתייחס אל מערך כאל משתנה המהווה רשימה של מספרים. בחן, למשל, את הרשימה הבאה:

numbers	3	7	-5	0	6	4	1

רשימה זו היא למעשה מערך הנקרא numbers. במערך numbers שבעה מספרים, אותם יש לקרוא משמאל לימין. כלומר, המספר הראשון הוא 3, המספר השני הוא 7, וכדומה.

כל מספר נמצא בתא (או באיבר) אחר במערך. כל תא ממוספר ב**אינדקס** (index) משלו, כדי שנוכל להבדילו מתאים אחרים. אינדקס התא הראשון הוא 1, ולכל תא אינדקס הגדול ב-1 מהאינדקס של התא הקודם. במערך numbers, למשל, המספר 3 נמצא בתא בעל האינדקס 1, המספר 7 נמצא בתא בעל האינדקס 2, וכן הלאה – עד למספר 1, הנמצא בתא בעל האינדקס 7.

כדי לפנות אל מספר כלשהו במערך נציין את אינדקס התא שלו, וזאת נעשה על ידי כתיבת שם המערך וציון האינדקס בסוגריים. למשל, פנייה לתא השני במערך numbers תיעשה כך: numbers(2). כל תא במערך נחשב למשתנה רגיל, וניתן לבצע עליו את כל הפעולות האלגוריתמיות שהכרנו. למשל, כדי להוסיף 1 למספר הנמצא בתא השני במערך נכתוב הוראת הצבה כזו:

numbers(2) ← numbers(2) + 1

וכדי לפלוט את המספר שבתא השביעי במערך נכתוב הוראת פלט כזו:

הדפס את הערך של numbers(7)

נהוג להשתמש בלולאת "עבור ... מ-... עד ... בצע..." כדי לפלוט את כל המספרים שבמערך. הלולאה הבא, למשל:

(1) עבור i מ-1 עד 7 בצע

(1.1) הדפס את הערך של numbers(i)

פולטת את כל המספרים שבתאי המערך numbers.

דוגמה

בדוגמה זו נפתח אלגוריתם הקולט עשרה מספרים מהמשתמש ופולט את מכפלת כל אחד מהמספרים ב-2. לשם כך נשתמש בשני מערכים. המערך a יכיל את המספרים שיזין המשתמש ; והמערך b יכיל את מכפלת מספרים אלה ב-2.

(1) עבור i מ-1 עד 10 בצע

(1.1) קלוט מספר למשתנה a(i)

(2) עבור i מ-1 עד 10 בצע

(2.1) b(i) ← a(i) * 2

(2.2) הדפס את הערך של b(i)

הוראה (1) היא לולאה המתבצעת עשר פעמים, ובמהלכה נקלטים מספרים לכל תאי המערך a. גם הוראה (2) היא לולאה המתבצעת עשר פעמים, והוראה (2.1) השייכת לה מציבה בכל תא במערך b את מכפלת המספר שבתא המתאים במערך a ב-2. הוראה (2.2) פולטת את המספרים שבתאי המערך b.

סיכום

- באמצעות שפת התכנות מגדיר המתכנת למחשב רצף הוראות הפותר בעיה לוגית נתונה. רצף הוראות זה מכונה **אלגוריתם**.
- הוראות האלגוריתם צריכות להיות נכונות מבחינה תחבירית ולוגית, חד-משמעיות ומובנות לגורם המבצע אותן.
- משתנה הוא אמצעי לאחסון נתונים.
- הוראת הקלט קולטת ערך מהמשתמש אל תוך משתנה.
- הוראת הפלט מציגה נתונים על צג המחשב.
- הוראת ההשמה מציבה ערך כלשהו במשתנה.
- תנאי מורכב מסימן יחס המופיע בין שני ערכים. התנאי מתקיים אם סימן היחס מבטא נכונה את היחס בין שני הערכים.
- הוראת התנאי "אם... אזי" מבצעת את ההוראות השייכות לה אם התנאי שלה מתקיים.
- הוראת התנאי "אם... אזי... אחרת" מבצעת קבוצת הוראות אחת אם התנאי שלה מתקיים, וקבוצת הוראות אחרת אם הוא אינו מתקיים.
- הוראת תנאי המכילה תת-הוראות "וגם..." תבצע את ההוראות השייכות לה אם כל התנאים שבה מתקיימים.
- הוראת תנאי המכילה תת-הוראות "או..." תבצע את ההוראות השייכות לה אם אחד מהתנאים שבה מתקיים.
- לולאה מבצעת קבוצת הוראות מסוימת מספר פעמים ידוע מראש או בהתאם לתנאי.
- מערך הוא טיפוס נתונים המאגד בתוכו כמה מספרים. כל מספר נמצא בתא אחר במערך, וניתן להתייחס אליו כאילו היה ערך של משתנה רגיל.

פרק 2

תרשימי זרימה

את האלגוריתמים שפיתחנו בפרק הקודם ייצגנו על ידי רשימה ממוספרת של הוראות מילוליות. דרך נוספת לייצג אלגוריתמים היא על ידי תרשימים, המורכבים מצורות הנדסיות בהן רשומות הוראות האלגוריתם, וחיצים המקשרים ביניהן, ובכך מורים על סדר ביצוע ההוראות. בפרק זה נכיר את תרשימי הזרימה ונלמד כיצד לשרטט אותם.

תרשימי זרימה פשוטים

תרשים זרימה (flowchart) מבטא בצורה גרפית את הוראותיו של אלגוריתם מסוים. כאשר בעיה לוגית עומדת בפנינו, נוח לשרטט את תרשים הזרימה כמעין סקיצה ראשונית לפני שניגשים למלאכת התכנות עצמה. כל הוראה באלגוריתם מיוצגת על ידי צורה הנדסית שונה בתרשים הזרימה, ובין כל צורה לצורה נמתח חץ המורה על סדר ביצוע ההוראות.

תרשימי זרימה המיועדים לאדם

רוב תרשימי הזרימה שנפתח מיועדים למחשב, אך לשם ההתנסות נחרוג מהרגלנו ונפתח מספר תרשימי זרימה המיועדים לאדם. בתרשימי זרימה כאלה, כל הוראה באלגוריתם מיוצגת בתוך מלבן.

נפתח בתרשים זרימה המתאר כיצד יש להזמין מזון באמצעות הטלפון. ראשית נקבע את הוראות האלגוריתם עליו יתבסס תרשים הזרימה.

(1) בחר את המזון שברצונך להזמין

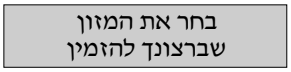
(2) בחר את הרשת שממנה ברצונך להזמין

(3) מצא את טלפון סניף הרשת הקרוב לביתך

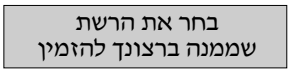
(4) טלפן אל הסניף

(5) מסור את פרטיך ואת בקשתך

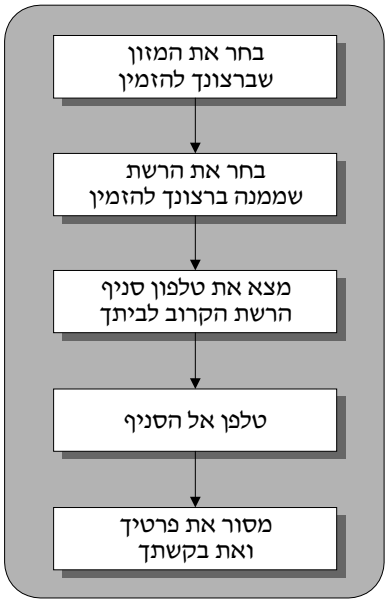
עתה נפתח את תרשים הזרימה המתאים לאלגוריתם זה. נתחיל בכתיבת כל הוראה אלגוריתמית בתוך מלבן. למשל, ההוראה הראשונה תיוצג כך:



ההוראה השנייה תיוצג כך:



וכדומה. באופן זה נייצג כל הוראה במלבן משלה. לאחר מכן נחבר בחץ כל מלבן עם המלבן שבא אחריו. תרשים הזרימה המתקבל מפעולה זו מוצג להלן:



***איור 2.1** – תרשים זרימה להזמנת אוכל באמצעות הטלפון*

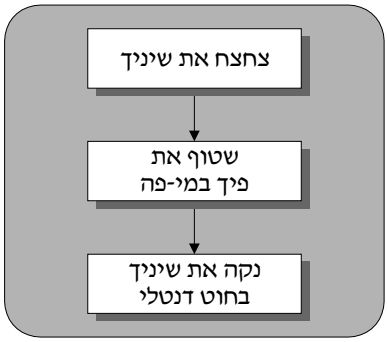
בדוגמה זו נפתח תרשים זרימה המתאים לאלגוריתם לשמירה על היגיינת השיניים. האלגוריתם יכיל את ההוראות הבאות (אותן יש לבצע פעמיים ביום):

(1) צחצח את שיניך

(2) שטוף את פיך במי-פה

(3) נקה את שיניך בחוט דנטלי

כדי לשרטט את תרשים הזרימה נייצג כל הוראה במלבן משלה ונמתח חיצים בין המלבנים שיתקבלו. תרשים הזרימה הסופי מוצג להלן:



***איור 2.2** – תרשים זרימה לשמירה על היגיינת הפה*

ייצוג פעולות אלגוריתמיות

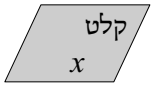
כזכור, אלגוריתמים המיועדים למחשב מורכבים מרצף של הוראות אלגוריתמיות (קלט, פלט, השמה, תנאים ולולאות). כאשר נשרטט תרשימי זרימה המיועדים למחשב, נייצג את ההוראות האלגוריתמיות הללו על ידי צורות הנדסיות שונות.

פעולת הקלט

פעולת קלט מיוצגת בתרשים הזרימה על ידי מקבילית, כאשר בראשה מופיעה המילה "קלט" ובמרכזה – שמות המשתנים בהם יוצבו הערכים הנקלטים. למשל, ההוראה האלגוריתמית הבאה:

קלוט מספר למשתנה x

תיוצג בתרשים הזרימה כך:

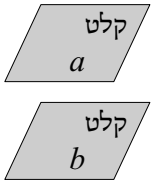


ניתן לשלב מספר הוראות קלט אלגוריתמיות במקבילית אחת. למשל, את ההוראות הבאות:

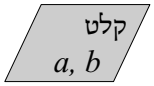
(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

ניתן לייצג בתרשים הזרימה על ידי שתי המקבילויות הבאות, כאשר הן מופיעות אחת אחרי השנייה:



או על ידי המקבילית הבודדת הבאה:



הערכים שיוזנו יוצבו במשתנים שבמקבילית על-פי סדר הופעתם. כלומר, הערך הראשון יוצב במשתנה השמאלי ביותר, הערך השני במשתנה שמימינו וכדומה. בדוגמה זו, אם הזין המשתמש את הערכים 4 ו-6, יכיל המשתנה a את הערך 4 והמשתנה b את הערך 6.

פעולת הפלט

גם פעולת פלט מיוצגת בתרשים הזרימה על ידי מקבילית. בראשה תופיע המילה "פלט" ובמרכזה – ההודעה או המשתנים שברצוננו להציג. הודעה מילולית נתחום בגרשיים. למשל, ההוראה האלגוריתמית הבאה:

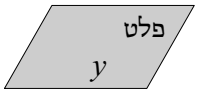
הדפס את ההודעה "שלום"

תיוצג בתרשים הזרימה כך:

פליטת ערך של משתנה תתבצע על ידי כתיבת שמו במרכז המקבילית (בלי גרשיים). למשל, ההוראה האלגוריתמית הבאה:

הדפס את הערך של y

תיוצג בתרשים הזרימה כך:

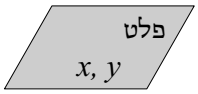


ניתן לפלוט מספר ערכי משתנים בהוראת פלט אחת על ידי הפרדתם בפסיק. למשל, את ההוראות הבאות:

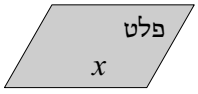
(1) הדפס את הערך של x

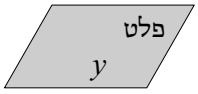
(2) הדפס את הערך של y

ניתן לייצג בתרשים הזרימה על ידי המקבילית הבאה:



גם בפעולת הפלט נפלטים ערכי המשתנים בהתאם לסדר בו הם מופיעים במקבילית, כאשר השמאלי ביותר נפלט ראשון. לכן, המקבילית שלעיל מייצגת פעולה זהה לפעולה שמייצגות שתי המקבילויות הבאות, כשהן מופיעות בזו אחר זו:





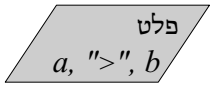
ניתן לפלוט ערכי משתנים יחד עם הודעה מילולית. למשל, את ההוראות הבאות:

(1) הדפס את הערך של a

(2) הדפס את ההודעה ">"

(3) הדפס את הערך של b

נייצג בתרשים הזרימה על ידי המקבילית הבאה:



שים לב כי שמות המשתנים וההודעה הופרדו זה מזה בפסיקים.

זהירות!

הוראת הפלט דומה מאוד להוראת הקלט, ועליך להיזהר לבל תתבלבל ביניהן. לדוגמה, שתי המקבילויות הבאות:

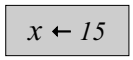
אינן מייצגות הוראות זהות. המקבילית השמאלית קולטת ערך למשתנה sting ואילו הימנית פולטת את ערכו.

פעולת ההשמה

פעולת השמה מיוצגת בתרשים הזרימה על ידי מלבן. למשל, את ההוראה הבאה:

x ← 15

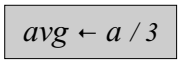
נייצג בתרשים זרימה כך:



ואת ההוראה הבאה:

avg ← a / 3

נייצג בתרשים זרימה כך:



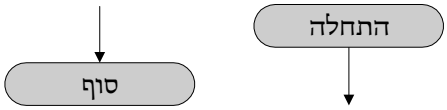
תרשימי זרימה המיועדים למחשב

לאחר שראינו כיצד כל הוראה אלגוריתמית בסיסית מיוצגת בתרשים זרימה, ברצוננו לפתח תרשימי זרימה המייצגים אלגוריתמים ממוחשבים מלאים. אך לפני-כן עלינו להכיר שני מרכיבים חשובים הקיימים בכל תרשים זרימה מלא: אליפסת התחלה ואליפסת סוף. אליפסות אלו מופיעות בכל תרשים זרימה המיועד למחשב, והן תוחמות אותו באופן הבא:

אליפסת ההתחלה מציינת את התחלת האלגוריתם, והיא תופיע תמיד בראש תרשים הזרימה, לפני כל הוראה אחרת.

אליפסת הסוף מציינת את סוף האלגוריתם, והיא תופיע תמיד בסוף תרשים הזרימה, אחרי כל הוראה אחרת.

בין אליפסת ההתחלה לצורה הבאה אחריה ובין אליפסת הסוף לצורה הבאה לפניה נמתח חיצים, כך:



כדוגמה ראשונה נציג תרשים זרימה לאלגוריתם הקולט שני מספרים ופולט את סכומם.
לפני פיתוח תרשים הזרימה נקבע את הוראות האלגוריתם המתאים לו:

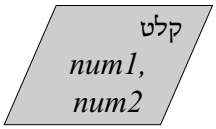
(1) קלוט מספר למשתנה num1

(2) קלוט מספר למשתנה num2

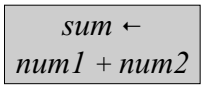
(3) sum ← num1 + num2

(4) הדפס את הערך של sum

עתה נייצג כל הוראה על ידי הצורה ההנדסית המתאימה לה. הוראות (1) ו-(2) תיוצגנה על ידי מקבילית קלט כזו:



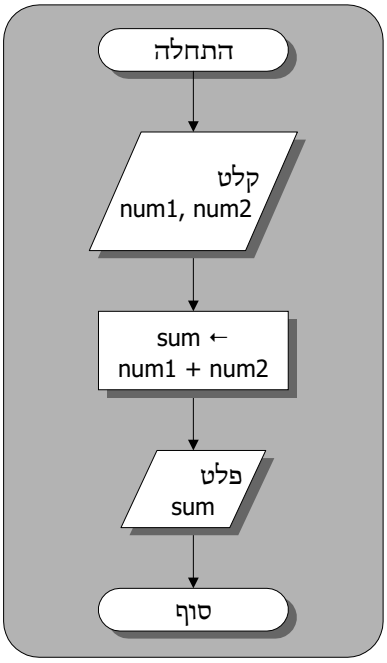
הוראה (3) תיוצג על ידי מלבן כזה:



הוראה (4) תיוצג על ידי מקבילית פלט כזו:

לבסוף נמתח חיצים בין שלוש הצורות ונוסיף לתרשים הזרימה את אליפסות ההתחלה והסוף.

תרשים הזרימה הסופי ייראה כך:



***איור 2.3** – תרשים זרימה לפליטת סכום ערכים*

דוגמה

בדוגמה זו יוצג תרשים זרימה של אלגוריתם לחישוב משכורתה השבועית של מוכרת בבוטיק. משכורת זו תיקבע על-פי שלושה ערכים שיזין המשתמש: כמה ימים עובדת המוכרת בשבוע, כמה שעות היא עובדת בכל יום וכמה משלמים לה עבור שעת עבודה. כרגיל, לפני פיתוח תרשים הזרימה נכתוב את הוראות האלגוריתם אותו הוא מייצג:

(1) קלוט מספר למשתנה days

(2) קלוט מספר למשתנה hours

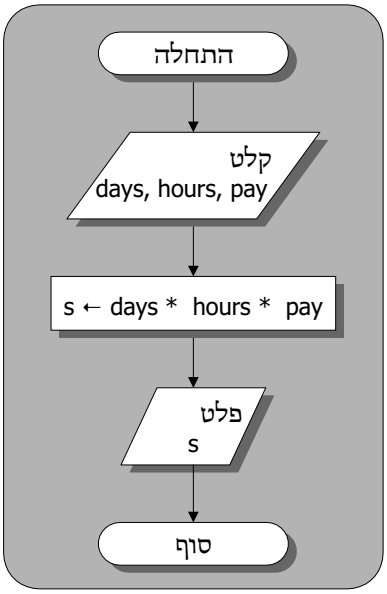
(3) קלוט מספר למשתנה pay

(4) s ← days * hours * pay

(5) הדפס את הערך של s

עתה נייצג כל הוראה באמצעות הצורה ההנדסית המתאימה לה, נוסיף אליפסות התחלה וסוף ונמתח ביניהם חיצים.

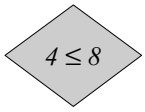
תרשים הזרימה שמתקבל מוצג להלן:



***איור 2.4** – תרשים זרימה לחישוב משכורת חודשית*

תרשימי זרימה בעלי תנאים

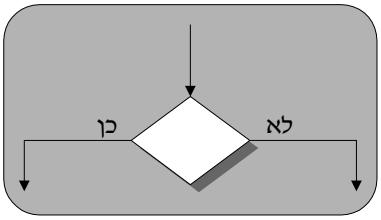
כזכור, כל הוראת תנאי מורכבת מתנאי, על-פיו נקבע אילו הוראות תתבצענה. תנאי מיוצג בתרשים זרימה על ידי מעוין. למשל, התנאי $4 \leq 8$ מיוצג בתרשים זרימה כך:



בתרשים זרימה ניתן לייצג רק הוראות תנאי מסוג "אם... אזי... אחרת". לשם כך, נוציא מהמעוין שני חיצים, כאשר כל חץ יצביע על קבוצת הוראות אחרת, המכונה **ענף** (branch). על-פי התנאי יוחלט איזה ענף הוראותיו יתבצעו: אם התנאי מתקיים, תבוצענה הוראותיו של ענף אחד; אם לא, תבוצענה הוראותיו של הענף האחר.

כדי להבדיל בין שני הענפים עלינו להבדיל בין החיצים המצביעים עליהם. לשם כך נסמן חץ אחד ב**כן** וחץ שני ב**לא**.

הדבר מוצג באיור הבא:



***איור 2.5** – ייצוג כללי של הוראת תנאי בתרשים זרימה*

אם התנאי יתקיים, תבוצענה ההוראות עליהן מצביע חץ ה**כן**. אם התנאי אינו מתקיים, תבוצענה ההוראות עליהן מצביע חץ ה**לא**.

כהרגלנו, נפתח בדוגמה פשוטה שאינה מיועדת למחשב. על סוכנות דוגמנות לוודא כי רק מועמדות שגובהן מעל מטר ושמונים סנטימטרים תתקבלנה לתפקיד המיוחל. אלגוריתם מתאים לבעיה זו ייכתב כך:

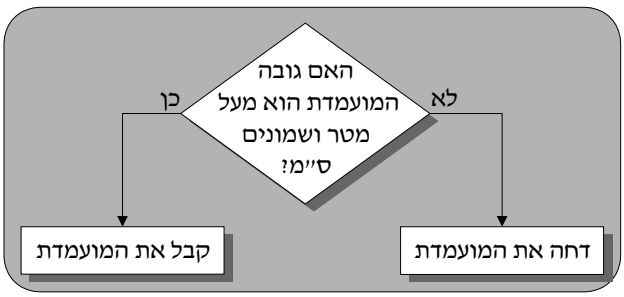
(1) אם גובה המועמדת מעל מטר ושמונים סנטימטרים אזי

(1.1) קבל את המועמדת

(2) אחרת

(2.1) דחה את המועמדת

הוראת תנאי הבודקת האם מועמדת מסוימת מתאימה להיות דוגמנית תיראה כך:



***איור 2.6** – ייצוג הוראת תנאי פשוטה בתרשים זרימה*

בדוגמה זו נפתח תרשים זרימה לאלגוריתם הקולט שני מספרים ופולט את המספר הגדול מביניהם. ראשית, נקבע את הוראות האלגוריתם:

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

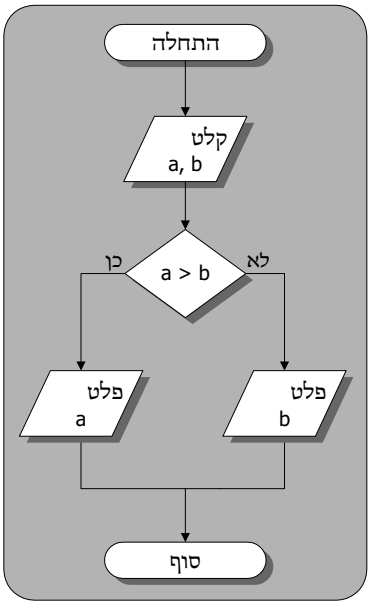
(3) אם a > b אזי

(3.1) הדפס את הערך של a

(4) אחרת

(4.1) הדפס את הערך של b

התנאי a > b ייכתב במעוין, ושני חיצים ייצאו ממנו, כאשר כל חץ מצביע על הוראת פלט אחרת. תרשים זרימה מתאים ייראה כך:



איור 2.7 *– תרשים זרימה לפליטת המספר הגדול מבין שני מספרים*

שים לב!

שים לב כיצד שני החיצים היוצאים מהוראות הפלט משתלבים לחץ אחד, המצביע על אליפסת הסוף. בסוף שני ענפים המפוצלים על ידי הוראת תנאי, חובה לסמן חיצים שיוודאו את המשכיות האלגוריתם.

באלגוריתם הבא, אותו נייצג על ידי תרשים זרימה מתאים, נקלוט מספר מהמשתמש ונציג על המסך את שורשו הריבועי. לפני הוצאת השורש הריבועי נוודא שהמספר שהוזן אינו שלילי, שהרי אין להוציא שורש ריבועי ממספר שלילי. הוראות האלגוריתם ייראו כך:

(1) קלוט מספר למשתנה x

(2) אם $x < 0$ אזי

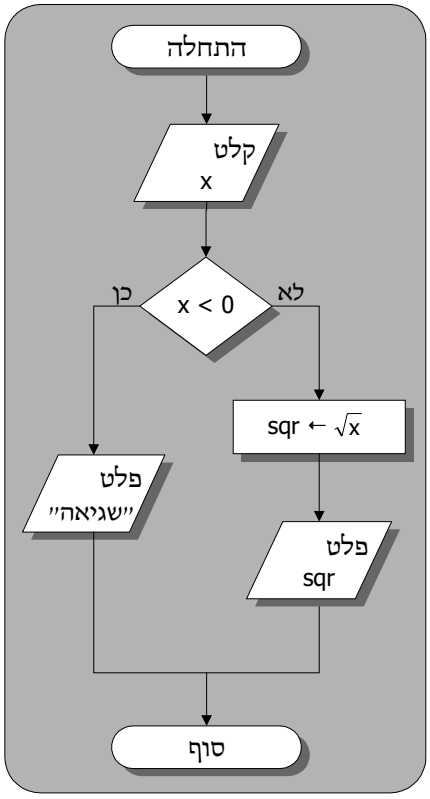
(2.1) הדפס את ההודעה "שגיאה"

(3) אחרת

(3.1) $sqr \leftarrow \sqrt{x}$

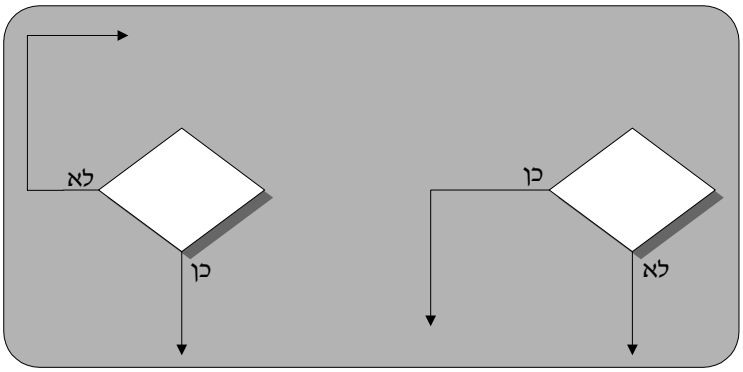
(3.2) הדפס את הערך של sqr

תרשים הזרימה המתאים מוצג באיור הבא:



***איור 2.8** – תרשים זרימה לפליטת שורשו הריבועי של מספר*

את חיצי ה**כן** וה**לא** ניתן להוציא מקודקודים שונים של המעוין. הנה, למשל, שתי דוגמאות לכך:



***איור 2.9** – דוגמאות להוצאת חיצי **כן** ולא*

דוגמה

תרשים הזרימה הבא ייצג אלגוריתם הקולט ארבעה מספרים מהמשתמש ופולט את המספר הגדול ביותר מביניהם. אלה הם שלבי האלגוריתם:

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

(4) קלוט מספר למשתנה d

(5) $x \leftarrow a$

(6) אם $b > x$ אזי

(6.1) $x \leftarrow b$

(7) אם $c > x$ אזי

(7.1) $x \leftarrow c$

(8) אם $d > x$ אזי

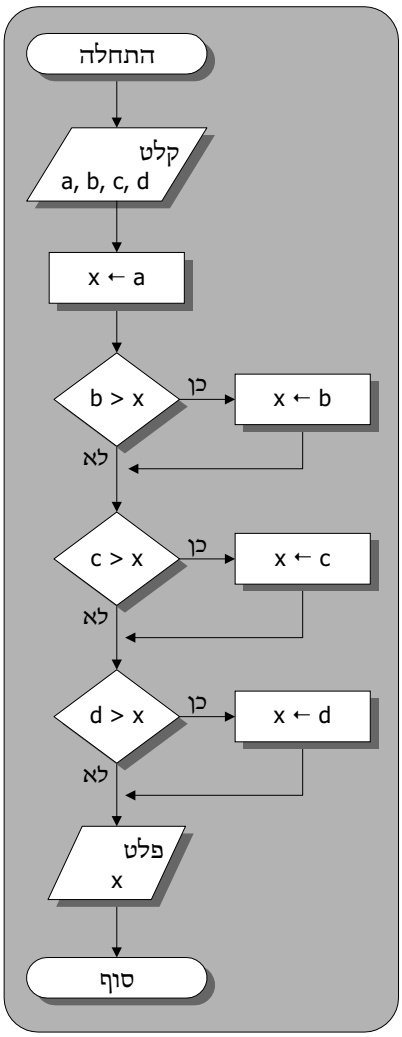
(8.1) $x \leftarrow d$

(9) הדפס את הערך של x

עקרון האלגוריתם הוא שימוש בהוראות תנאי כדי לוודא שבמשתנה המונה x יוצב הערך הגדול מבין כל המשתנים. בהוראה (5) מוצב ערך המשתנה a במשתנה x. בהוראה (6) מושווים ערכי המשתנים x ו-b. אם b > x, אזי גם b > a, ולכן נבצע את הוראת ההשמה x ← b. בהוראה (7) מושווים ערכי המשתנים x ו-c. אם c > x, אזי גם c > b ו-c > a, ולכן נבצע את הוראת ההשמה x ← c. בהוראה (8) מושווים ערכי המשתנים x ו-d. אם d > x, אזי גם d > c, d > b ו-d > a, ולכן נבצע את הוראת ההשמה x ← d.

באופן זה, לאחר שלוש הוראות התנאי, יכיל המשתנה x את המספר הגדול ביותר שהוזן.

לאחר ייצוג כל שלב באמצעות צורה הנדסית מתאימה, יתקבל תרשים הזרימה הבא:



***איור 2.10** – תרשים זרימה לפליטת המספר הגדול מבין ארבעה מספרים*

הוראות תנאי מקוננות

הוראת תנאי אלגוריתמית הבודקת האם שני תנאים מתקיימים במקביל נכתבת באמצעות תת-ההוראה "וגם...". נניח שלפנינו האלגוריתם הבא, הקובע מתי יש לרכוש מטריה:

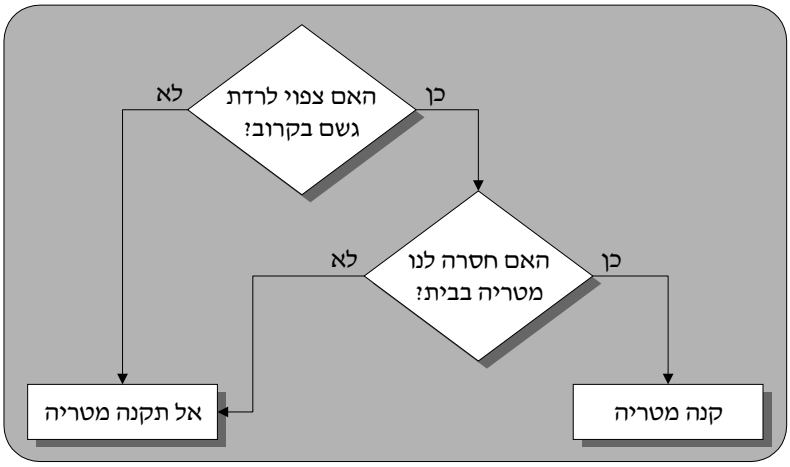
(1) אם צפוי לרדת גשם בקרוב וגם חסרה לנו מטריה בבית אזי

(1.1) קנה מטריה

(2) אחרת

(2.1) אל תקנה מטריה

כדי לייצג הוראת תנאי מסוג "אם... וגם... אזי" בתרשים זרימה נהוג לשייך הוראת תנאי אחת להוראת תנאי אחרת. כלומר, אלגוריתם זה ייוצג בתרשים זרימה כך:



איור 2.11 *– הוראות תנאי הבודקות האם יש צורך לקנות מטריה*

דוגמה

תרשים הזרימה הבא ייצג אלגוריתם הקולט שלושה מספרים מהמשתמש ופולט את הגדול מביניהם. תרשים זרימה זה מבצע פעולה דומה לזו שביצע תרשים הזרימה הקודם שהצגנו, אך הוא עושה זאת על ידי שימוש בהוראות תנאי מקוננות. אלה הם שלבי האלגוריתם:

(1) קלוט מספר למשתנה a

(2) קלוט מספר למשתנה b

(3) קלוט מספר למשתנה c

(4) אם a > b אזי

(4.1) אם a > c אזי

(4.1.1) הדפס את הערך של a

(4.2) אחרת

(4.2.1) הדפס את הערך של c

(5) אחרת

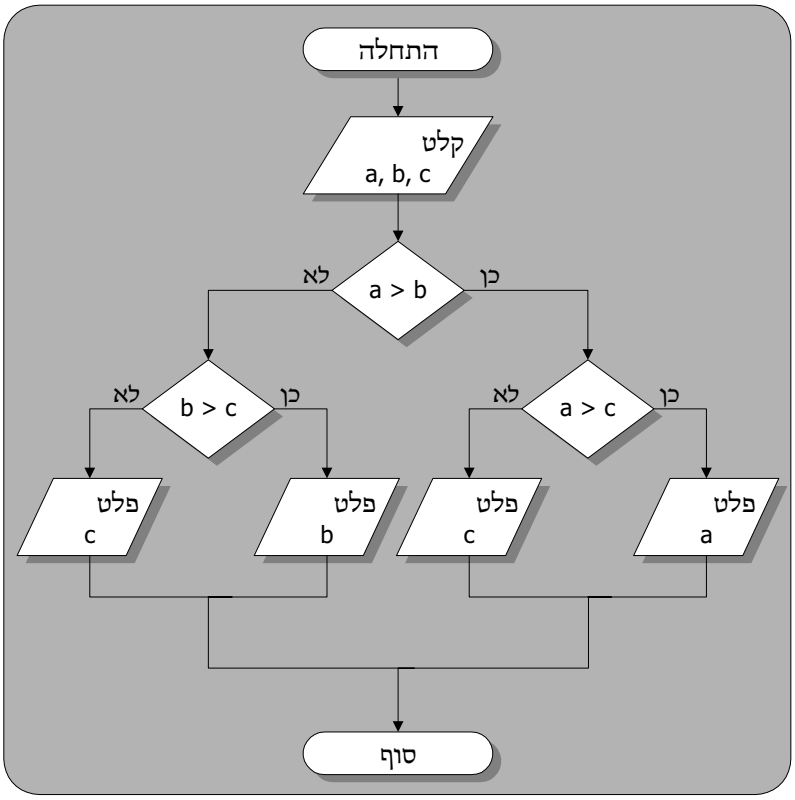
(5.1) אם b > c אזי

(5.1.1) הדפס את הערך של b

(5.2) אחרת

(5.2.1) הדפס את הערך של c

תרשים הזרימה המתאים מוצג באיור הבא:



***איור 2.12** – תרשים זרימה לפליטת המספר הגדול מבין שלושה מספרים*

תרשימי זרימה בעלי לולאות

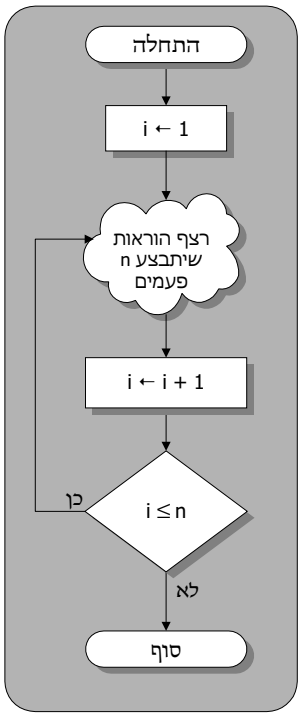
הוראה אלגוריתמית המבצעת רצף הוראות מסוים n פעמים נכתבת כך:

בצע n פעמים

לא ניתן לייצג ישירות הוראה כזו בתרשים זרימה. במקום זאת נבצע לולאה באופן עקיף על ידי הוראת תנאי. אם ברצוננו לבצע רצף הוראות מסוים n פעמים, נעקוב אחר השלבים הבאים:

1. נקבע משתנה מונה, נניח i, ונציב בו את הערך 1.
2. נבחר אילו הוראות ברצוננו לבצע n פעמים ונוסיף אחריהן הוראה המעלה את ערך המשתנה המונה ב-1.
3. נוסיף הוראת תנאי הבודקת האם $i \leq n$.
   - ❖ אם כן, נחזור אל ההוראה הראשונה שיש לבצע n פעמים.
   - ❖ אם לא, נעבור להוראה הבאה.

תרשים זרימה המייצג אלגוריתם זה מוצג באיור הבא:



***איור 2.13** – תבנית בסיסית לשימוש בלולאות*

נתאר את השימוש בלולאות על ידי דוגמה פשוטה הפולטת את המילה "לולאה" ארבע פעמים. המשתנה המונה יהיה counter וההוראה שברצוננו לבצע ארבע פעמים – הוראת הפלט. שלבי האלגוריתם יהיו כדלהלן:

(1) counter ← 1

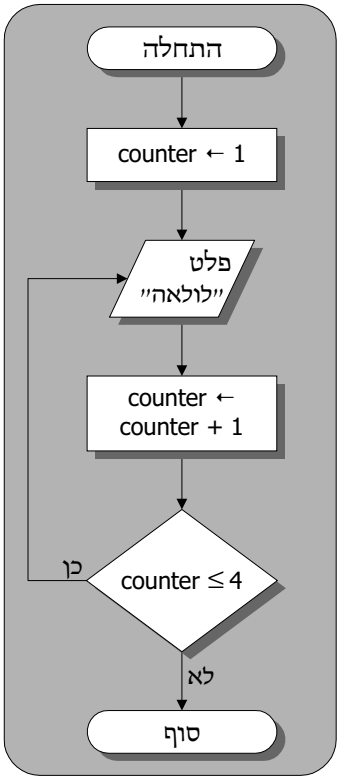
(2) הדפס את ההודעה "לולאה"

(3) counter ← counter + 1

(4) אם counter ≤ 4 אזי

(4.1) חזור להוראה (2)

תרשים זרימה מתאים מוצג באיור הבא:



***איור 2.14** – תרשים זרימה לפליטת המילה "לולאה" ארבע פעמים*

בדוגמה זו נשתמש בפעולת התנאי לשם חזרה על הוראות מסוימות מספר פעמים שאינו ידוע מראש. תרשים הזרימה שנפתח ייצג אלגוריתם הקולט מספרים מהמשתמש שוב ושוב, עד להזנת ערך שלילי. לבסוף, יוצג המספר הגדול ביותר שהוזן. אלה הם שלבי האלגוריתם:

(1) biggest ← 0

(2) קלוט מספר למשתנה n

(3) אם n > biggest אזי

(3.1) biggest ← n

(3.2) חזור להוראה (2)

(4) אם n < 0 אזי

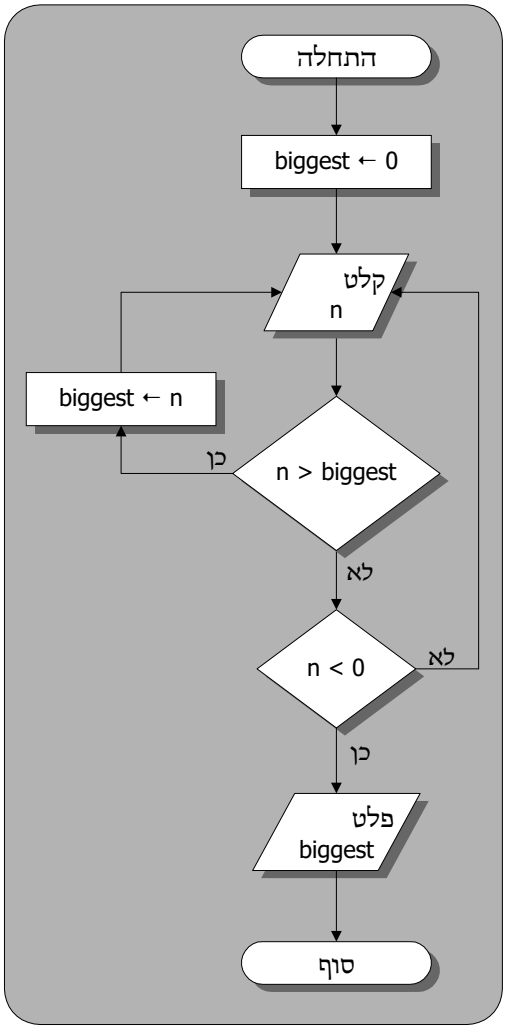
(4.1) הדפס את הערך של biggest

(5) אחרת

(5.1) חזור להוראה (2)

אלגוריתם זה קולט ערכים מהמשתמש שוב ושוב לתוך המשתנה n. הוראה (3.1) מציבה ב-biggest את ערך המשתנה n אם n > biggest, כלומר, אם המספר האחרון שהוזן גדול מהמספר שבמשתנה biggest. לכן יכיל המשתנה biggest את הערך הגדול ביותר שנקלט עד כה. כאשר יוזן מספר שלילי, הוראה (4.1) תפלוט את ערך המשתנה biggest (המכיל את המספר הגבוה ביותר שהוזן), ומאחר שהוראה (5.1) לא תתבצע, האלגוריתם יסתיים.

תרשים הזרימה המתאים לאלגוריתם זה מוצג להלן:



***איור 2.15** – תרשים זרימה לפליטת המספר הגדול מבין מספרים שהוזנו*

באמצעות הלולאות ניתן לבצע חישובים מתמטיים מורכבים. אחת מהפעולות המתמטיות הפשוטות שאינן ניתנות לחישוב בלי שימוש בלולאות היא מציאת **עצרת** (factorial) של מספר נתון.

עצרת של מספר n (בכתיב מתמטי, !n) היא מכפלת כל המספרים השלמים מ-1 ועד n. כלומר, כאשר n הוא מספר טבעי:

$$n! = 1 * 2 * 3 * \ldots * n$$

למשל, עצרת המספר 5 (!5) היא 5 * 4 * 3 * 2 * 1, כלומר 120.

כאשר n הוא 0 העצרת היא 1 (1 = !0).

נפתח עתה תרשים זרימה לאלגוריתם הקולט מהמשתמש מספר טבעי ומציג את העצרת שלו. המספר הנקלט יוצב במשתנה x והמשתנה אותו נפלוט לבסוף יהיה total, שערכו בתחילת האלגוריתם יהיה 1. את ערך המשתנה total נכפיל בכל מספר שלם מ-1 עד x, כך שיכיל את עצרת המספר x.

שלבי האלגוריתם הם אלה:

(1) קלוט מספר למשתנה x

(2) counter ← 1

(3) total ← 1

(4) total ← counter * total

(5) counter ← counter + 1

(6) אם counter ≤ x אזי

(6.1) חזור להוראה (4)

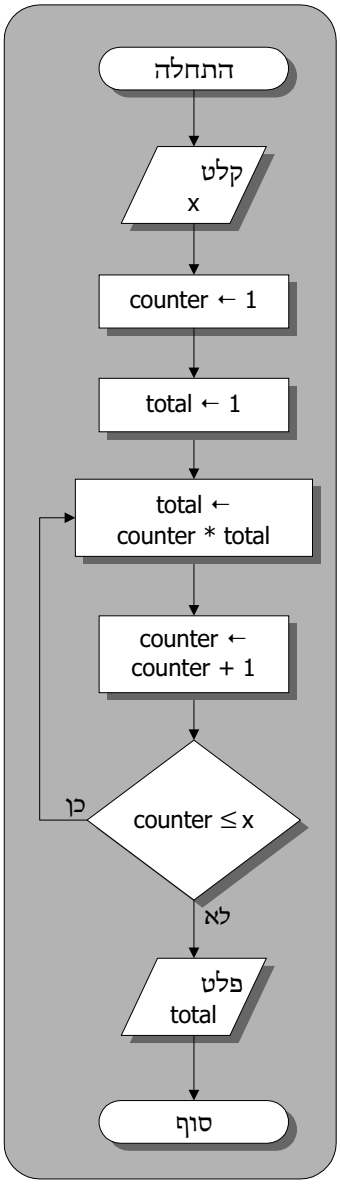
(7) הדפס את הערך של total

הוראה (6.1) מביאה לביצוען של הוראות (4) ו-(5) x פעמים. בהוראה (4) מוצבת מכפלת ערכי המשתנים count ו-total במשתנה total ובהוראה (5) ערך counter עולה ב-1. באופן זה, אם ערך המשתנה x הוא למשל 3, הוראות (4) ו-(5) תתבצענה שלוש פעמים, ובמהלכן יוצב במשתנה total הערך הבא:

$$counter * (counter + 1) * (counter + 2)$$

מאחר ש-counter מאותחל ל-1, הערך המתקבל הוא 3 * 2 * 1, שהוא עצרת המספר 3.

תרשים זרימה מתאים מוצג באיור הבא:



***איור 2.16** – תרשים זרימה לפליטת עצרת של מספר*

נתאר עתה תרשים זרימה מורכב למדי, המציג את סדרת **פיבונאצ'י** (fibonacci). סדרת פיבונאצ'י היא סדרה בה כל איבר שווה לסכום שני האיברים הקודמים לו, כאשר שני האיברים הראשונים בסדרה הם 0 ו-1. בכתיב מתמטי מבוטאת הסדרה כך:

$$a_1 = 0, \quad a_2 = 1,$$
$$a_n = a_{n-2} + a_{n-1}$$

לדוגמה, האיבר השלישי בסדרה שווה ל-$a_1 + a_2$, כלומר ל-1, האיבר הרביעי בסדרה שווה ל-$a_2 + a_3$, כלומר ל-2, וכדומה. אלה הם איברי הסדרה הראשונים:

$$0, \quad 1, \quad 1, \quad 2, \quad 3, \quad 5, \quad 8, \quad 13, \quad 21$$

תרשים הזרימה שנפתח ייצג אלגוריתם המציג את n איברי הסדרה הראשונים, כאשר n הוא ערך שמזין המשתמש. נשתמש במשתנים a1 ו-a2 כדי לייצג שני איברים עוקבים בסדרה, ובמשתנה counter כמונה. המשתנה x ייצג את מספר האיברים שנפלטו עד כה, והוא יאותחל ל-2, משום ששני האיברים הראשונים נפלטים כבר בראשית האלגוריתם (איברים אלה הם קבועים, והסדרה לא מתקיימת בלעדיהם). שלבי האלגוריתם הם כדלהלן:

(1) קלוט מספר למשתנה n

(2) a1 ← 0

(3) a2 ← 1

(4) x ← 2

(5) הדפס את הערך של a1

(6) הדפס את הערך של a2

(7) tmp ← a2

(8) a2 ← a2 + a1

(9) a1 ← tmp

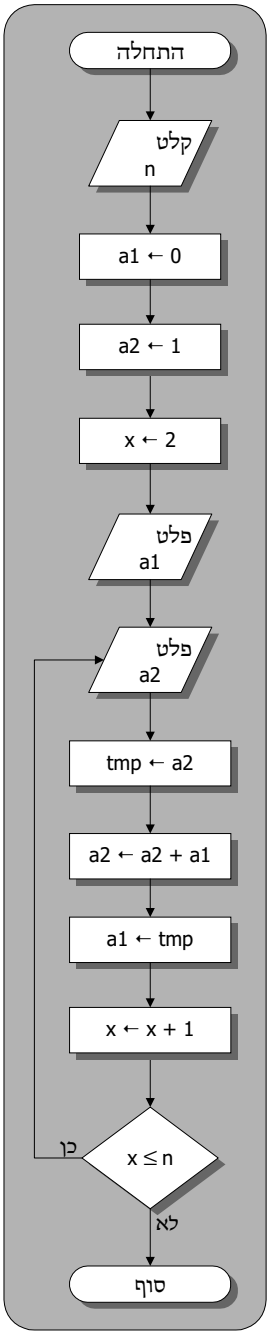
(10) x ← x + 1

(11) אם x ≤ n אזי

(11.1) חזור להוראה (6)

הוראה (11.1) מבצעת את הוראות (6)-(10) n פעמים. בכל פעם נפלט ערך המשתנה a2, ערך משתנה זה נהיה לאיבר הבא בסדרה (על ידי הוספת האיבר הקודם אליו, המיוצג על ידי המשתנה a1) וערך המשתנה a1 נהיה לערך המקורי של a2, לפני השינוי האחרון שבוצע בו.

תרשים הזרימה המתאים מוצג באיור הבא:



***איור 2.17** – תרשים זרימה לפליטת איברים בסדרת פיבונאצ'י*

סיכום

- תרשים זרימה הוא אמצעי גרפי לייצוג אלגוריתם.
- כל הוראה באלגוריתם מיוצגת על ידי צורה אחרת בתרשים הזרימה: הוראות הפלט והקלט – במקבילית; הוראת ההשמה – במלבן; הוראת התנאי – במעוין.
- כל תרשים זרימה מתחיל באליפסת התחלה ומסתיים באליפסת סוף.
- בין צורות תרשים הזרימה נמתחים חיצים הקובעים את סדר ביצוע ההוראות.
- הוראת תנאי המכילה תת-הוראה "וגם..." מיוצגת בתרשים זרימה על ידי הוראות תנאי מקוננות.
- לולאות מיוצגות בתרשים זרימה על ידי הוראת תנאי הבודקת האם תנאי מסוים מתקיים, ובמידת הצורך חוזרת להוראה קודמת.

פרק 3

כלי הפיתוח של השפה

עד כה למדנו כיצד ליישם פתרונות לבעיות לוגיות באמצעות תרשימי זרימה. בקרוב נלמד כיצד ליצור קבצים ממוחשבים, אותם נוכל להריץ במחשב ולהיעזר בהם למטרות שונות. בפרק זה נכיר את כלי הפיתוח של שפת C, המאפשרים יצירת קבצים כאלה. בניגוד לאופי התיאורטי של שני הפרקים הקודמים, בפרק זה יושם דגש על ההיבט הטכני של פיתוח בשפות תכנות.

שים לב!

פרק זה מסתמך על הכתוב בסעיף "סוגי שפות תכנות" בנספח א' של הספר. אם אינך בקיא בסוגי שפות התכנות, מומלץ לקרוא סעיף זה לפני המשך הקריאה בפרק זה.

סוגי קבצים

בפרק זה נצא מנקודת הנחה, שהמתכנת פיתח אלגוריתם לבעיה ממוחשבת כלשהי, וברצונו ליצור קובץ שיבצע את הוראות האלגוריתם. קובץ זה יסופק למשתמש, שיריצו כאשר יחפוץ. הקובץ היחיד אותו מבין המחשב באמת הוא קובץ בשפת מכונה, המכיל נתונים בספרות 0 ו-1. קובץ זה מכיל את ההוראות שעל המחשב לבצע, והמחשב יקרא כל הוראה ויבצעה כמיטב יכולתו.

הקובץ הסופי הדרוש לביצוע ההוראות חייב להיכתב, אם-כן, בספרות 0 ו-1. למזלנו, איננו מוכרחים ללמוד את שפת המכונה המייגעת. במקום זאת, ביכולתנו לכתוב תוכנית המורכבת ממילים מובנות ומסימני פיסוק ברורים – ואותם תתרגם תוכנה מיוחדת לשפת המכונה. על התוכנית שכותב המתכנת למלא אחר כללים שקובעת התוכנה המתרגמת, כדי שתדע כיצד לתרגמה לשפת המכונה.

עתה נכיר את סוגי הקבצים המשתתפים בתהליך בו הופך קובץ המקור בשפת C לקובץ הרצה, אותו מבין המחשב.

- הקובץ הראשון הבא לידי ביטוי בתהליך יצירת קובץ ההרצה הוא **קובץ המקור** (source file). קובץ מקור הוא קובץ טקסט רגיל המכיל הוראות בשפת C. בדרך כלל סיומתו היא c. או h. (code file או source file). תוכן קובץ המקור (כלומר, ההוראות הכתובות בשפת C) נקרא **קוד מקור** (source code).
- קובץ המקור יתורגם לקובץ הכתוב בשפת הסף. קובץ זה, המכונה **קובץ אסמבלי** (assembly file), יכיל פקודות בשפת סף המקבילות לאלו שנכתבו בקובץ המקור בשפת C. בדרך כלל סיומתו היא asm.
- קובץ האסמבלי יתורגם ל**קובץ אובייקטים** (object file). קובץ זה יכיל פקודות בשפת המכונה המקבילות לאלו שהופיעו בקובץ האסמבלי. קובץ האובייקטים כתוב אמנם בשפת המכונה, אך לא ניתן להריצו ולכן אין הוא הקובץ הסופי. בדרך כלל סיומתו היא obj או h.
- קובץ האובייקטים יקושר יחד עם קבצי אובייקטים אחרים לקובץ הסופי, **קובץ ההרצה** (executable file). כאמור, קובץ ההרצה הוא הקובץ אותו יכול להפעיל משתמש התוכנה, והוא כתוב בשפת המכונה. סיומתו היא, בדרך כלל, exe.

יצירת הקבצים

הקבצים שתוארו לעיל לא נוצרים יש מאין. מספר תוכנות אחראיות על יצירתם, החל מתרגום קובץ המקור וכלה ביצירת קובץ ההרצה. כפי שמוצג באיור 3.1, קיימים שני תהליכים מרכזיים בהם נוצר קובץ ההרצה מקובץ המקור: **הידור** (compilation) ו**קישור** (link). במהלך תהליכים אלה נוצרים גם שאר הקבצים שצוינו בסעיף הקודם.

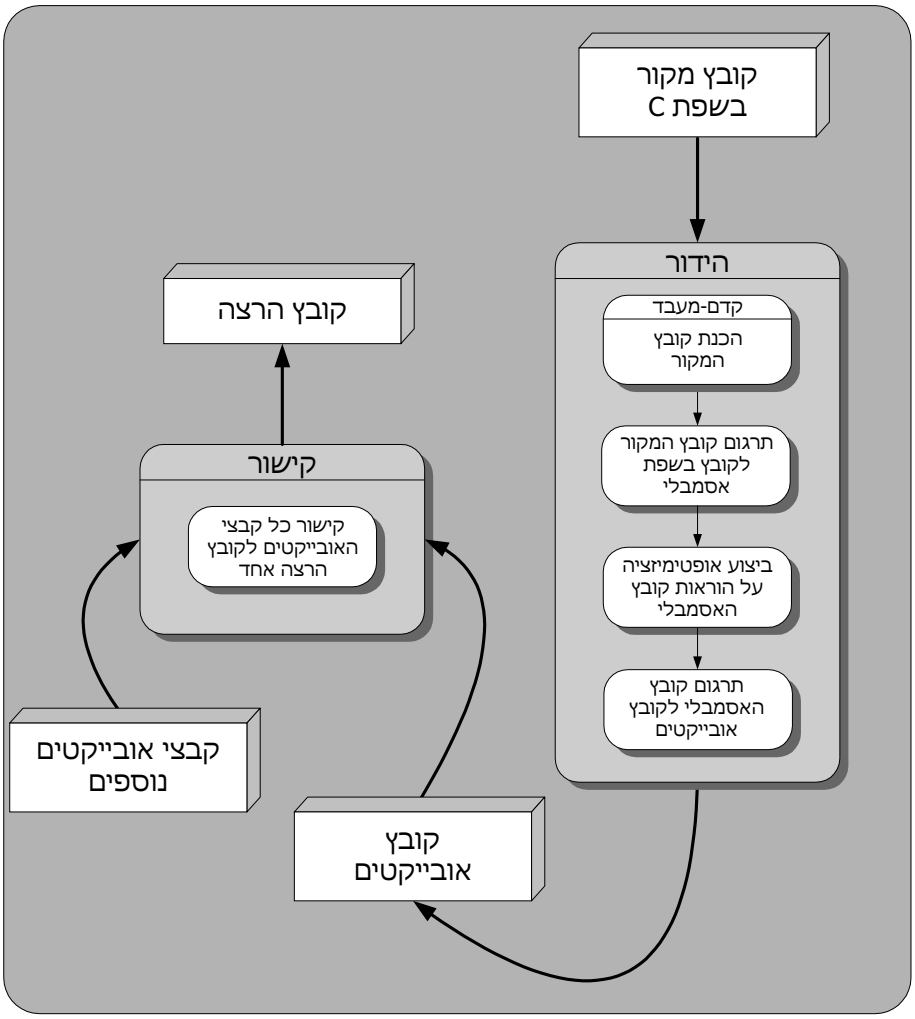
הידור

בתהליך ההידור יוצרת תוכנה הנקראת **מהדר** (compiler) את קובץ האובייקטים מקובץ המקור. בתהליך זה נוצר גם קובץ האסמבלי, אך הוא בדרך כלל נמחק לאחר השימוש בו. המהדר מורכב ממספר תוכנות מובנות, שכולן באות לידי ביטוי בתהליך ההידור.

ראשון מופעל **קדם-המעבד** (preprocessor). מספר פעולות מתבצעות בתהליך זה; בין היתר, הוצאת תווים מיותרים מקובץ המקור. לאחר מכן מבצע המהדר את פעולתו המרכזית: תרגום הקובץ משפת C לשפת הסף. בסיום התהליך נוצר קובץ המכיל פקודות בשפת סף המקבילות לפקודות בשפת ה-C שהופיעו בקובץ המקור.

בשלב זה נעשית, בדרך כלל, פעולת **אופטימיזציה** זו או אחרת על קובץ האסמבלי. בפעולה זו מחפש המהדר דרכים להקטנת גודלו ולשיפור מהירותו של קובץ ההרצה. דרך אחת לעשות זאת היא הוצאת הוראות טפלות מקובץ האסמבלי (למשל, הוראה המתבצעת מספר פעמים, בעוד שניתן לבצעה פעם אחת בלבד בלי שתשתנה פעולת קובץ ההרצה).

עתה באה לידי ביטוי תוכנת האסמבלר הבנויה במהדר, המתרגמת את קובץ האסמבלי לקובץ אובייקטים. בדרך כלל יימחק קובץ האסמבלי לאחר פעולה זו.



איור 3.1 *– תהליך יצירת קובץ הרצה*

קישור

תוכנת הקשר מאחדת את קובץ האובייקטים שהתקבל בסוף תהליך ההידור עם קבצי אובייקטים נוספים ויוצרת מהם את קובץ ההרצה. קבצי קוד האובייקטים הנוספים הם קבצים המסופקים עם כלי הפיתוח של התוכנה, ומכילים פקודות שונות שכבר הודרו לשפת המכונה.

בכל אחד מהשלבים שלעיל עלולה להתרחש תקלה ; למשל, אם שגינו בכתיבת פקודה כלשהי. במקרה כזה ייפסק התהליך ותיפלט הודעה למשתמש שעליו לתקן את הבעיה.

התוכנית הראשונה

התוכניות המופיעות בספר מוצגות כמלל בשפת C, אך את כולן נרצה להפוך לקבצי הרצה ולראות כיצד המחשב מבצע את ההוראות המופיעות בהן. לשם כך עלינו ללמוד כיצד לבצע את תהליכי ההידור והקישור וכיצד לעבוד עם כלי השפה. כדוגמה, ניקח את תוכנית C המופיעה להלן ונהפכה לקובץ הרצה:

תוכנית 3.1 *FIRST.C*

```
1  #include <stdio.h>
2  void main()
3  {
4      int a, b, sum;
5
6      printf("Please enter two numbers: ");
7      scanf("%d %d", &a, &b);
8      sum = a + b;
9      printf("Sum of numbers: %d \n", sum);
10 }
```

תוכנית זו קולטת שני מספרים שלמים מהמשתמש ומציגה על המסך את סכומם. סביר ששורות התוכנית נראות לך מוזרות למדי, אך מובטח כי תבינן בהמשך.

שים לב!

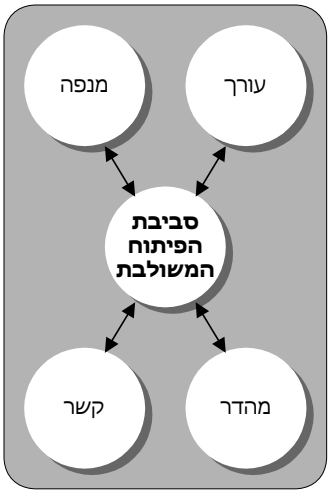
מספרי השורות משמאל אינם חלק מהתוכנית עצמה – הוספנו אותם כדי שנוכל לפנות בקלות לשורות ספציפיות במועד מאוחר יותר; כאשר תכתוב תוכניות משלך במחשב, אל תוסיף מספרי שורות כפי שהם מופיעים בספר.

יצירת קובץ ההרצה

כדי ליצור קובץ הרצה מהתוכנית שלעיל, על כל השלבים המוצגים באיור 3.1 להתבצע בהצלחה. בסעיף זה נראה כיצד לעשות זאת.

לפני שנתפנה למלאכת היצירה, עליך לקבוע אילו כלי פיתוח נמצאים ברשותך. כלי הפיתוח הם הכלים המאפשרים תרגום הוראות הכתובות בשפת C לתוכנית בשפת המכונה, ועמם נמנים המהדר, הקשר וכלי עזר נוספים. כיום קיימים כלי פיתוח רבים לשפת C, המסופקים על ידי יצרנים שונים ברמה מקצועית שונה.

רוב כלי הפיתוח המודרניים של שפת C מורכבים מתוכנה ראשית אחת, המכונה **סביבת פיתוח משולבת** (IDE – Integrated Development Environment). מתוכנה זו ניתן להפעיל את כל הכלים הדרושים לתכנות במהירות ובנוחיות (ראה איור 3.2). חלקה העיקרי של סביבת הפיתוח המשולבת היא תוכנת העריכה, המיועדת במיוחד למתכנתים בשפת C. תוכנת העריכה מבצעת שינויים אוטומטיים במלל המוצג על המסך, כדי לפשט את תהליך כתיבת התוכנית ולהקל על קריאתה. בנוסף, היא מאפשרת הפעלה נוחה של המהדר והקשר: ההידור, הקישור והרצת הקובץ הסופי מבוצעים יחדיו על-פי פקודתך, כך שתהליך היצירה נהיה קל ומהיר יותר.



***איור 3.2** – יחסי הגומלין בין סביבת הפיתוח המשולבת לשאר כלי הפיתוח*

המהדר TCLite המצורף לספר הוא, למעשה, סביבת פיתוח משולבת. אם אין בידך מהדר משוכלל יותר, מומלץ להשתמש במהדר זה.

בניגוד לסביבת הפיתוח המשולבת, קיימים כלי פיתוח המסופקים כתוכנות נפרדות. לשם יצירת קובץ ההרצה באמצעותם, עליך לעבוד עם מספר תוכנות נפרדות: תוכנת עריכה, מהדר וקשר. את התוכנות הללו, כמו את קובץ ההרצה שיווצר מהן, יהיה עליך להריץ ידנית מ**שורת הפקודה** (command prompt). שורת הפקודה היא השורה דרכה ניתן לכתוב פקודות ישירות למערכת ההפעלה, לדוגמה:

```
C:\Windows>
```

יצירת קובץ ההרצה, הן תחת סביבת הפיתוח המשולבת והן תחת שורת הפקודה, אינה מלאכה מסובכת.

עצה
מומלץ להריץ במחשב את כל התוכניות המופיעות בספר. אם תעשה כך, במהרה ייעשה תהליך היצירה בטבעיות רבה.

להלן הנחיות לגבי יצירת קובץ ההרצה תחת סביבת פיתוח משולבת ותחת שורת הפקודה. בדוק אילו כלי פיתוח עומדים לרשותך ובחר בהסבר המתאים.

שים לב!

במהלך הידור תוכנית 3.1, ייתכן שהמהדר שברשותך יפלוט הודעת שגיאה המנוסחת בערך כך:

```
main must have a return type of int
```

כדי לתקן הודעה זו, החלף את שורה (2) בתוכנית 3.1 בשורה הבאה:

```
int main(void)
```

והוסף את השורה הבאה לפני הסוגריים המסולסלים הסוגרים של תוכנית 3.1 (כלומר, בין שורות (9) ו-(10)):

```
return 0;
```

במידת הצורך, עשה דבר דומה גם בתוכניות הבאות שתקליד במחשבך.

יצירת קובץ ההרצה מסביבת הפיתוח המשולבת

עקוב אחר הצעדים הבאים אם ברשותך סביבת פיתוח משולבת. מכיון שקיימות סביבות פיתוח משולבות רבות, לא נפרט כיצד לבצע פעולות טריוויאליות כגון פתיחה או שמירה של קובץ. היעזר במסמכי העזרה המסופקים עם סביבת הפיתוח המשולבת שברשותך כדי לקבל מידע על ממשקה.

כתיבת התוכנית

השלב הראשון ביצירת קובץ ההרצה הוא כתיבת ה-C. פתח את סביבת העבודה המשולבת, צור קובץ חדש וכתוב בו את תוכנית 3.1. בהקלידך, שים לב כי חלק משורות התוכנית נכתבות ימינה יותר משורות אחרות. מותר לעשות כך; כפי שנראה בהמשך, ברוב המקרים אין חשיבות לאופן בו מרווחת התוכנית.

שים לב!

כאשר שורות נכתבות מימין לשורות שמעליהן, נאמר כי הן מוזחות ימינה. נראה בהמשך כי ה**הזחה** (indentation) מסייעת רבות בקריאת התוכנית.

אם בידך עורך משוכלל, ייתכן שמילים מסוימות בתוכנית יופיעו בצבעים שונים. זאת נעשה כדי להקל על ההבחנה בין חלקיה התחביריים השונים של השפה.

בתום כתיבת התוכנית, בחן כל שורה שהקלדת והשווה אותה אל השורה המתאימה בתוכנית 3.1. זכור כי תוכנית מחשב שאינה עוקבת אחר חוקי התחביר של שפת התכנות בה משתמשים אינה נחשבת לנכונה.

עתה, שמור את התוכנית.

עצה

מומלץ להעניק לתוכנית שם המרמז על תפקידה ולתת לו את הסיומת c.; לדוגמה, frame.c. מהדרים מסוימים לא יקבלו קבצים שאינם בעלי סיומת זו.

הידור וקישור התוכנית

אף שניתן להדר ולקשר את התוכנית בנפרד, בדרך כלל נרצה לבצע שתי פעולות אלו יחד. לשם כך נפעיל פקודה הנקראת Link (קַשֵּׁר). פקודה זו תפעיל את המהדר, ואם לא יהיו שגיאות, גם את תוכנת הקשר.

אם חלה שגיאה, בדוק האם מקורה בתוכניתך או לא (ביכולתך להיעזר בהסבר המופיע בהמשך אודות הודעות המהדר). אם השגיאה בתוכניתך, ודא שכתבת את התוכנית כיאות – גם השמטת התו התמים ביותר עלולה לגרור שגיאות רבות! אם מקור ההודעה אינו בתוכניתך אלא בקשר (קרוב לוודאי ששגיאת קשר תתחיל במילים Linker Error), בדוק כי הספריה בה מצויים קבצי הספריה המסופקים עם כלי הפיתוח מוגדרת כהלכה. לרוב נמצאת ספריה זו בתיקיה lib, בתוך תיקיית כלי הפיתוח הראשית. על הקשר לדעת היכן מצויים קבצי הספריה, ואם הספריה המוגדרת לו אינה נכונה, תיפלט הודעת שגיאה בתהליך הקישור.

אם הסתיים תהליך הקישור בהצלחה, יווצר קובץ הרצה ששמו הוא כשם קובץ המקור, פרט לסיומת (.exe).

במחשבה שנייה...

לאחר שנוצר קובץ ההרצה, האם הוא תלוי בקובץ המקור?

קובץ ההרצה נוצר אמנם מקובץ המקור, אך הוא אינו תלוי בו לאחר שנוצר. אם תשנה עתה את קובץ המקור, לא ישתנה קובץ ההרצה. כדי לעדכן את קובץ ההרצה עליך ליצור אותו מחדש, על-פי השלבים שתוארו לעיל.

הרצת התוכנית

עתה, כל שיש לעשות הוא להריץ את התוכנית! לשם כך חפש והפעל פקודה הנקראת Run (הָרֵץ). על המסך תופיע הודעה כדלהלן:

```
Please enter two numbers:
```

בשלב זה התוכנית תושהה ותמתין לקבלת שני ערכים מספריים, המופרדים זה מזה על ידי הקשה על מקש הרווח, על מקש Enter או על מקש Tab. לאחר שנכתבו שני מספרים, השתמש במקש Enter כדי לסיים את פעולת הקלט. על המסך תופיע הודעה המציגה את סכום הערכים שנקלטו. הנה דוגמה לפלט של תוכנית זו:

```
Please enter two numbers: 13 9
Sum of numbers: 22
```

שים לב!

בהמשך הספר נייצג ערכים המוזנים על ידי המשתמש במלל נטוי. כאשר תזין ערכים למחשב הם לא יוצגו על המסך באופן זה; אנו עושים זאת כדי שנוכל להבדיל בקלות בין קלט לפלט.

סביר להניח שאילו בחרנו בפקודה Run לפני שהידרנו וקישרנו את התוכנית, היו מתבצעים שלבי ההידור והקישור אוטומטית. ברוב המקרים כדאי לעשות כך, כדי לקצר את תהליך יצירת קובץ ההרצה. עד כה הדגמנו את השלבים השונים לשם התנסות בכלי הפיתוח.

יצירת קובץ ההרצה משורת הפקודה

עקוב אחר הצעדים הבאים אם הינך עובד באמצעות תוכנות נפרדות לעריכה, להידור ולקישור.

כתיבת התוכנית

השלב הראשון ביצירת קובץ ההרצה הוא הקלדת ה-C. בחר בעורך הנוח לך ביותר, אך שים לב שעל קובץ המקור להישמר בפורמט פשוט, שאינו כולל גופנים שונים. רוב העורכים ששומרים טקסט עם גופנים, כדוגמת Word, מאפשרים לשמור את הקובץ גם בפורמט פשוט, כגון ASCII. כמו-כן ביכולתך להשתמש בתוכנת העריכה Edit המסופקת יחד עם מערכת ההפעלה DOS.

בהקלידך, שים לב שחלק משורות התוכנית נכתבות ימינה יותר משורות אחרות. מותר לעשות כך; כפי שנראה בהמשך, ברוב המקרים אין חשיבות לאופן בו מרווחת התוכנית.

שים לב!

כאשר שורות נכתבות מימין לשורות שמעליהן, נאמר כי הן מוזחות ימינה. נראה בהמשך כי ה**הזחה** (indentation) מסייעת רבות בקריאת התוכנית.

בתום כתיבת התוכנית, בחן כל שורה שהקלדת והשווה אותה אל השורה המתאימה בתוכנית 3.1. זכור כי תוכנית מחשב שאינה עוקבת אחר חוקי התחביר של שפת התכנות בה משתמשים אינה נחשבת לנכונה.

עתה, שמור את התוכנית. מומלץ להעניק לתוכנית שם המרמז על תפקידה ולתת לו את הסיומת c. ; לדוגמה, frame.c. מהדרים מסוימים לא יקבלו קבצים שאינם בעלי סיומת זו. לאחר ששמרת את התוכנית, צא מתוכנת העריכה וחזור לשורת הפקודה של DOS.

הידור וקישור התוכנית

לשם הידור וקישור התוכנית מספיק להפעיל את תוכנת המהדר. לרוב, היא תפעיל מיוזמתה את הקשר. תוכנת המהדר מופעלת משורת הפקודה ועלינו לספק לה ארגומנטים, המציינים את שם הקובץ אותו יש להדר והנחיות כיצד להדרו. זאת נעשה על ידי פקודה הדומה לזו:

```
tcc -Ic:\tc\include -Lc:\tc\lib frame.c
```

באיור 3.3 מוצגים מרכיבי הפקודה. ראשית, מופעלת תוכנת המהדר על ידי ציון שמה, במקרה זה – tcc (שנה שם זה בהתאם למהדר שברשותך). המהדר מקבל מספר נתונים משורת הפקודה: הדגלים I ו-L ושם קובץ המקור אותו עליו לתרגם לקובץ הרצה.

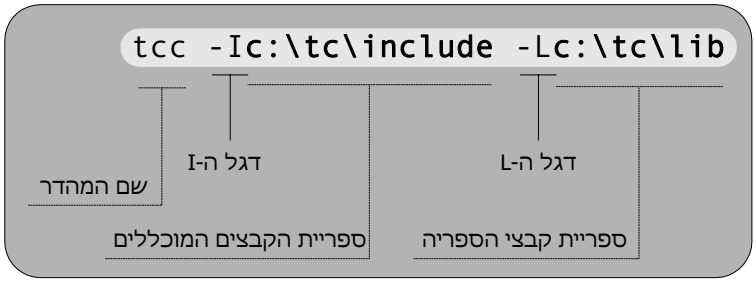
עצה

בדרך כלל לא משתנים הדגלים ממהדר למהדר, אך כדאי לוודא כי תחת המהדר שברשותך דגלים אלה אכן מבצעים את המתואר בהמשך.

שני הדגלים מופרדים זה מזה על ידי רווח. שניהם מתחילים במקף (כך מבדיל המהדר בין דגלים לבין שמות קבצים), ומצורפת אליהם ספריה, ששמה הודגש להקלת הקריאה. תפקידי הדגלים הם אלה:

- דגל ה-I מגדיר למהדר את מיקום הקבצים המוכללים המסופקים עם כלי הפיתוח (קבצים מוכללים הם קבצים המסופקים עם המהדר, כדוגמת הקובץ stdio.h, אליו אנו פונים מתוכניתנו באמצעות ההוראה שבשורה (1)). שנה את שם הספריה בהתאם למיקומה במחשבך (בדרך כלל ספריה זו נקראת include).

❖ דגל ה-L מגדיר למהדר את מיקום קבצי הספריה (קבצי האובייקטים) המסופקים עם כלי הפיתוח. שנה את שם הספריה בהתאם למיקומה במחשבך (בדרך כלל, ספריה זו נקראת lib).



***איור 3.3** – מרכיבי הפקודה המפעילה את המהדר*

אם חלה שגיאה, בדוק האם מקורה בתוכניתך או לא (ביכולתך להיעזר בהסבר המופיע בהמשך הפרק אודות הודעות המהדר). אם השגיאה בתוכניתך, ודא שכתבת את התוכנית כיאות – גם השמטת התו התמים ביותר עלולה לגרור שגיאות רבות! אם מקור ההודעה אינו בתוכניתך אלא בקשר (קרוב לוודאי ששגיאת קשר תתחיל במילים Linker Error), בדוק כי הספריה בה מצויים קבצי הספריה המסופקים עם כלי הפיתוח מוגדרת כהלכה. בדרך כלל ספריה זו היא הספריה lib, הנמצאת בתוך ספריית כלי הפיתוח הראשית. על הקשר לדעת היכן מצויים קבצי הספריה, ואם הספריה המוגדרת לו אינה נכונה, תיפלט הודעת שגיאה בתהליך הקישור.

אם הסתיים תהליך הקישור בהצלחה, יווצר קובץ הרצה ששמו הוא כשם קובץ המקור, פרט לסיומת (exe.). שים לב שאם תשנה עתה את קובץ המקור, לא ישתנה קובץ ההרצה. כדי לעדכן את קובץ ההרצה עליך ליצור אותו מחדש, על-פי השלבים שתוארו לעיל.

הרצת התוכנית

עתה, כל שיש לעשות הוא להריץ את התוכנית! לשם כך כתוב את שם קובץ ההרצה בשורת הפקודה והקש על Enter. על המסך תופיע הודעה כדלהלן:

```
Please enter two numbers:
```

בשלב זה התוכנית תושהה ותמתין לקבלת שני ערכים מספריים, המופרדים זה מזה על ידי הקשה על מקש רווח, על מקש Enter או על מקש Tab. לאחר שנכתבו שני מספרים, השתמש במקש Enter כדי לסיים את פעולת הקלט. על המסך תופיע הודעה המציגה את סכום הערכים שנקלטו. הנה דוגמה לפלט של תוכנית זו:

```
Please enter two numbers: 13 9
Sum of numbers: 22
```

שים לב!

בהמשך הספר נייצג ערכים המוזנים על ידי המשתמש במלל נטוי ומודגש. כאשר תזין ערכים למחשב הם לא יוצגו על המסך באופן זה; אנו עושים זאת כדי שנוכל להבדיל בקלות בין קלט לפלט.

עבודה עם המהדר

המהדר הוא כלי הפיתוח המרכזי ביצירת התוכנית. בסעיפים הבאים נכיר את הודעות השגיאה והאזהרה הנפלטות בזמן הידור התוכנית, ואף נשנה בכוונה את התוכנית שהרצנו בסעיף הקודם כדי לגרום למהדר להציג הודעות שגיאה. במהלך פיתוח תוכניותיך תיתקל לעיתים קרובות בשגיאות ובאזהרות כאלו, ולכן כדאי ללמוד, עוד בראשית הדרך, כיצד להגיב.

הודעות שגיאה

כאשר נתקל המהדר במשפט או בסימן שאינם תואמים את תחביר השפה, תיפלט הודעת שגיאה אחת או יותר ויופסק תהליך יצירת קובץ ההרצה. בדרך כלל תהיה הודעת השגיאה מדויקת ותציג את מקור הבעיה המדויק.

להלן מספר הודעות שגיאה שמציג המהדר כאשר לא עוקבים כהלכה אחר חוקי התחביר של השפה:

```
Statement missing ;
```

הודעה זו מוסרת כי שכחנו להוסיף תו נקודה-פסיק בסוף משפט.

```
Compound statement missing }
```

הודעה זו מוסרת כי הופיעו סוגריים מסולסלים פותחים ללא סוגריים מסולסלים סוגרים (סוגריים אלה באים תמיד בצמד).

בנוסף לתיאור הטעות התחבירית שנעשתה, יודיע המהדר באיזו שורה הוא נתקל בה. כך קל ונוח למצוא את מקור הבעיה ולתקנו.

בעקבות טעויות מסוימות, יציג המהדר הודעה כללית ופחות ספציפית מאלו המופיעות לעיל. זאת ועוד – לעיתים, השורה בה נתקל המהדר בשגיאה אינה השורה בה נעשתה הטעות. במקרה כזה יש לבחון היטב את התוכנית ולחפש את שורש הבעיה.

הודעות אזהרה

סוג נוסף של הודעות הנפלטות מהמהדר הוא הודעות אזהרה. הודעות אזהרה מתריעות על סגנון כתיבה לקוי העלול לגרום לתקלה בלתי רצויה, או על פקודות סרק שלא מבצעות פעולה של ממש בתוכנית אך מגדילות אותה ומאיטות את קצב ריצתה. בדרך כלל לא תופסק פעולת המהדר בעקבות הודעות אזהרה, אך מומלץ בחום להתיחס אליהן כאילו היו הודעות שגיאה; היינו, לבדוק את מקורן ולתקן כנדרש.

עצה

כדי להבטיח את יעילות תוכניתך, ודא שהמהדר שברשותך עובד תחת רמת האזהרות הגבוהה ביותר, כלומר, שהוא פולט אזהרה על כל מקרה אפשרי.

להלן מספר הודעות אזהרה שמציג המהדר כאשר הוא נתקל בקטע תוכנית "חשוד":

```
Unreachable code
```

הודעה זו מוסרת כי מספר משפטים נכתבו בתוכנית, אך אין סיכוי שאי-פעם יתקיימו (לדוגמה, אם הם מותנים בקיום ביטוי תנאי שאף פעם אינו מתקיים).

```
Division by zero
```

הודעה זו מוסרת כי מתבצעת חלוקת מספר ב-0, האסורה לביצוע במחשב כפי שהיא אסורה במתמטיקה.

גם בהודעות האזהרה יודיע המהדר באיזו שורה מקורן.

התנסות עם הודעות המהדר

לאחר שהכרנו מספר הודעות שגיאה ואזהרה, הבה נראה כיצד הן נפלטות. נעשה זאת על ידי שינוי תוכנית הדוגמה כך שתהיה שגויה, ונראה כיצד מגיב לכך המהדר.

ראשית, ערוך את קובץ המקור של תוכנית 3.1 ומחק את הסוגריים המסולסלים הסוגרים המופיעים בשורה (10). נסה להדר את התוכנית המעודכנת על-פי השלבים שהוצגו קודם לכן. המהדר יגיב לתוכנית השגויה על ידי הפסקת תהליך יצירת קובץ ההרצה ופליטת הודעת שגיאה הדומה לזו:

```
Error FIRST.C 9:  Compound statement missing }
```

מרכיבי הודעת השגיאה מוצגים באיור הבא:

***איור 3.4** – מרכיבי הודעת שגיאה טיפוסית*

המהדר ציין הן את מקור הטעות (העדר סוגריים מסולסלים סוגרים) והן את השורה בה מצוי מקור הבעיה. מקור הבעיה הוא שורה (9), מכיון שהיא השורה האחרונה אותה בודק המהדר, ובסיום בדיקתה הוא מצפה להופעת התו הסוגר '}'. המהדר לא רואה בשורה (10) את מקור הבעיה, מאחר שברוב המקרים ריווח התוכנית אינו משמעותי למהדר. מבחינתו, ניתן לכתוב את הסוגריים המסולסלים הסוגרים גם בשורה (9), מייד לאחר תו הנקודה-פסיק.

הבה ניווכח בעצמנו כי המהדר מתעלם מרוב הרווחים שבתוכנית. ראשית החזר את הסוגריים המסולסלים הסוגרים לקובץ המקור, כך שיהיה זהה לתוכנית 3.1. עתה מחק את כל הירידות לשורה חדשה, החל מהשורה השנייה (את השורה הראשונה השאר במקומה) – כך ששורות (2)-(10) תופענה בשורה אחת. עתה תיראה התוכנית כך:

```
#include <stdio.h>
void main() { int a, b, sum; printf("Please enter two numbers: \n");
    scanf("%d %d", &a, &b); sum = a + b; printf("Sum of numbers:
    %d \n", sum); }
```

שים לב!

בשל מגבלות רוחב הדף לא ניתן להציג את השורה הארוכה בלי לפצלה למספר שורות. שים לב לנוהג הקיים גם בהמשך הספר: שורות המופיעות מימין לשורה שמעליהן שלא הסתיימה, הן אינן שורות משל עצמן, אלא המשך השורה שמעליהן. בעורך שברשותך תוכל לנוע בקלות לאורך שורה ארוכה באמצעות חיצי המקלדת או העכבר.

נסה עתה להדר את התוכנית הזו. אם מילאת אחר ההוראות במלואן, תיווכח כי נוצר קובץ הרצה שפעולתו זהה לשל קובץ ההרצה שנוצר קודם. המסקנה היא כי ברוב המקרים, ריווח התוכנית אינו משפיע על אופן היווצרות קובץ ההרצה.

סיכום

- קובץ הכתוב בשפת C מכונה **קובץ מקור**.
- קובץ המופעל במחשב מכונה **קובץ הרצה**. הוא חייב להיכתב בשפת המכונה, המורכבת מהספרות 0 ו-1.
- קובץ המקור מתורגם לקובץ הרצה על ידי תוכנות המהדר והקשר.
- ניתן להפעיל את המהדר ואת הקשר דרך סביבת פיתוח משולבת ודרך שורת הפקודה.
- במהלך פעולות ההידור והקישור עלולות להיפלט הודעות שגיאה ואזהרה. במקרה זה נמצא את מקור הבעיה בקובץ המקור ונתקן את הטעות.

פרק 4

אבני היסוד של השפה

בפרק הקודם למדנו כיצד להשתמש בכלי הפיתוח של שפת C ליצירת קובץ הרצה. את הידע הזה נממש מפרק זה ואילך, כאשר נתחיל את התכנות הממשי בשפת C. במהלך הפרק נציג מספר תוכניות פשוטות ונחקור מעט את רכיביה השונים של השפה. בסוף הפרק כבר תוכל ליצור תוכניות C משלך!

תוכנית שלד

הדרך הטובה ביותר לביסוס הידע הראשוני בשפה היא הצגת תוכנית לדוגמה. מכיון שזו התוכנית הראשונה שננתח (בניגוד לתוכנית 3.1, שהוצגה ללא הסברים כלשהם), ככל שהתוכנית תהא קצרה כך ייטב. התוכנית הקצרה ביותר שניתן לכתוב בשפת תכנות נקראת **תוכנית שלד** (skeleton application). משמה ניתן להבין כי היא משמשת שלד, או מסגרת, לתוכניות מורכבות יותר. ההוראות המופיעות בתוכנית השלד תופענה בכל תוכנית שנכתוב בעתיד.

תוכנית השלד של שפת C היא כדלקמן:

תוכנית 4.1 ***SKELETON.C***

```
void main()
{

}
```

לפני שנתעמק בתוכנית, הקלד אותה בעורך שברשותך ובצע את הפעולות הדרושות כדי ליצור ממנה קובץ הרצה. במידת הצורך, חזור ורענן את זיכרונך בנאמר בפרק 3 בנושא זה. כאשר תריץ את קובץ ההרצה, תיווכח כי לא מתרחש דבר. זאת מפני שהתוכנית היא תוכנית שלד; על-פי הגדרתה, על תוכנית השלד להיות פשוטה ככל הניתן, ומה פשוט יותר מתוכנית שלא עושה דבר?

עתה נתעמק מעט בתוכנית. ראשית, שים לב להבדל בין תרשים זרימה לבין תוכנית בשפת C. בעוד שתרשים זרימה מכיל בדיוק את הוראות האלגוריתם המתאים לו, תוכנית בשפת C מכילה תמיד מספר הוראות בנוסף להוראות האלגוריתם המתאים לה. תוכנית זו, למשל, לא עושה דבר, והיא מתאימה לאלגוריתם ריק מהוראות – אך היא אינה ריקה מתוכן.

אם נבחן את התוכנית בשלמותה, נוכל לומר כי במרכזה פעולה עיקרית אחת: הגדרת הפונקציה הראשית. הפונקציה הראשית היא תת-תוכנית שההוראות המופיעות בה מתבצעות בזו אחר זו בכל פעם שקובץ ההרצה מורץ. אף שבשלב זה אף הוראה לא מופיעה בה, חובה להגדירה בכל תוכנית C.

ננתח עתה בקצרה את מבנה תוכנית השלד.

```
void main()
```

התוכנית נפתחת במילה השמורה void, שלעת עתה נתעלם ממנה. אחריה מופיעה המילה main, שהיא שמה של הפונקציה הראשית. זהו שם קבוע, ולא ניתן להחליפו באחר. צמד הסוגרים () הבא אחר כך קובע כי השם main מציין פונקציה (ולא, למשל, משתנה).

שים לב!

מהדר שפת C מבחין בין אותיות קטנות לגדולות. לכן, את שם הפונקציה main אין לכתוב MAIN, את המילה השמורה void אין לכתוב Void, וכדומה.

```
{

}
```

הסוגריים המסולסלים הפותחים והסוגרים מגדירים את גוף הפונקציה הראשית. עתה, אם נרצה שהוראה כלשהי תתבצע כאשר יורץ קובץ ההרצה, נרשום אותה בגוף הפונקציה הראשית, כלומר, בין הסוגריים המסולסלים. הוראה זו יכולה להיות הוראת קלט, פלט, ועוד – אך עליה להיכתב בשפת C. זאת נעשה כבר בסעיף הבא.

תווים לבנים

תווים לבנים (white-space characters) הם תווים שאינם משפיעים על אופן ביצוע התוכנית, והם נועדו רק להקלה על קריאתה. תווים אלה הם תווי הרווח, הטאב והירידה לשורה חדשה. בתוכנית בשפת C, בכל מקום שמופיע תו לבן בודד, יכול להופיע תו לבן דומה או אחר – אחד או יותר. למשל, את כל תווי הירידה לשורה חדשה שבתוכנית 4.1 ניתן להחליף בתו רווח בודד, כך שתתקבל התוכנית הבאה:

```
void main() { }
```

אף שתוכנית זו זהה לחלוטין לתוכנית 4.1, נעדיף להימנע מהאופן בו היא כתובה, ולרווח את התוכניות שנכתוב כדי להקל על קריאתן.

פלט בסיסי

הצעד הראשון בהפיכת תוכניתנו למעניינת יותר הוא פליטת מלל אל המסך, וכדי לעשות זאת יש להפעיל הוראה מיוחדת (כלומר, לקרוא לה). הוראת הפלט שנכיר, הקרויה printf, מוגדרת בקבצים המצורפים עם כלי הפיתוח, ולכן ניתן להפעילה מתוך תוכניתנו.

בפונקציה printf נשתמש לאורך כל הספר. פשטותה, המאפשרת גם למתכנת המתחיל להשתמש בה, אינה גורעת מעוצמתה, ובהמשך נראה כיצד להשתמש בה כדי לפלוט ערכים למסך במקצועיות.

כדי לקרוא להוראה יש לכתוב את שמה ולציין איזה מלל ברצוננו להציג על המסך. המלל ייתחם בגרשיים וייכתב בין סוגריים המופיעים לאחר שם ההוראה. לדוגמה, כדי לפלוט את המלל המפורסם "hello, world", נפעיל את printf כך:

```
printf("hello, world");
```

שים לב!

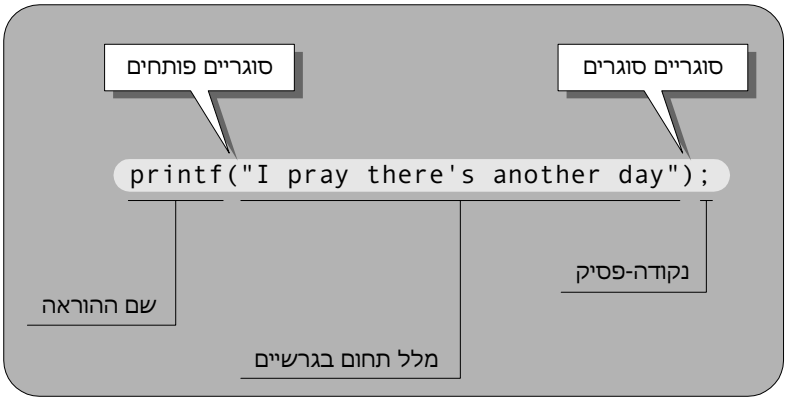
כדי להדגים את השימוש ב-printf בחרנו לכתוב שורה בודדת ולא תוכנית מלאה. נעשה זאת פעמים רבות בהמשך כדי לחסוך במקום וכדי להתמקד בנושא המועבר.

כדי להציג מלל אחר, כל שיש לעשות הוא להחליף את הטקסט התחום בגרשיים באחר. לדוגמה, כדי להציג את ההודעה "C is better than Pascal", נכתוב הוראה כזאת:

```
printf("C is better than Pascal");
```

שים לב לכל הפרטים בקריאה ל-printf. תחילה מופיע שמה, הכתוב באותיות לועזיות קטנות. המלל שברצוננו לפלוט נתחם על ידי גרשיים ומופיע בין סוגריים פותחים וסוגרים. לאחר הסוגריים הסוגרים מופיע תו הנקודה-פסיק, המציין את סופה של

הקריאה להוראה. אי-ציות לכללים אלה עלול לגרום מספר הודעות שגיאה. האיור הבא מסכם את הנאמר לעיל:



***איור 4.1** – אופן הקריאה לפונקציה printf*

את הקריאה ל-printf יש לכתוב בתוך הפונקציה הראשית, כך שתתבצע בכל פעם שמורץ קובץ ההרצה. נראה זאת בדוגמה הבאה.

דוגמה

בדוגמה זו נכתוב תוכנית המציגה משפט קצר על המסך על ידי קריאה ל-printf. כזכור, תוכנית השלד מהווה בסיס לתוכניות מורכבות יותר, ולכן ההוראות שהופיעו בה תופענה גם בתוכנית שלהלן.

תוכנית 4.2 *PRINTF.C*

```
#include <stdio.h>
void main()
{
    printf("Hey, we are learning C here!");
}
```

כרגיל, מומלץ להריץ את התוכנית ולהתרשם מפעולתה. כאשר תעשה כך, תיווכח כי היא מביאה לפליטת המלל הבא:

```
Hey, we are learning C here!
```

בתוכנית זו מופיעות שתי שורות שלא נראו בתוכנית השלד:

```
#include <stdio.h>
```

בשורה זו מופיעה הוראה הגורמת למהדר להתייחס להוראות שונות הנמצאות בקובץ stdio.h (standard input/output). גם קובץ זה כתוב בשפת C, והוא מכיל הוראות הדרושות לתוכניתנו. בהוראת include# זו או אחרת נשתמש כמעט בכל תוכנית שנכתוב, ונעמיק אודותיה בהמשך; בינתיים, דע כי עליך להוסיפה לתוכנית אם ברצונך להשתמש בהוראות קלט ופלט. שים לב שהיא מתחילה בסולמית (#) ושבסופה לא מופיע התו נקודה-פסיק. כמו-כן, על הוראת ה-include# להופיע בשורה משלה, כלומר, שלא תופיע הוראה אחרת לצידה.

```
    printf("Hey, we are learning C here!");
```

שורה זו מכילה קריאה ל-printf, המביאה לפליטת המשפט הקצר על המסך.

בגוף הפונקציה הראשית יכולות להופיע הוראות רבות, ואז תבוצע כל הוראה בנפרד, על-פי סדר הופעתן. לכן, ההוראה הבאה,

```
printf("1 2 3 4 5 6 7 8 9 10");
```

שקולה לשתי ההוראות הבאות, כשהן מופיעות בזו אחר זו ובסדר הנתון:

```
printf("1 2 3 4 5 ");
printf("6 7 8 9 10");
```

שים לב כי ההוראה printf אינה מורידה את סמן הטקסט לשורה חדשה לאחר ביצועה. נראה בהמשך כיצד להציג מלל לאורך מספר שורות.

דוגמה

נדגים את השימוש במספר הוראות printf על ידי התוכנית הבאה:

תוכנית 4.3 *PRINTF2.C*

```
#include <stdio.h>
void main()
{
    printf("Hey, ");
    printf("we are learning ");
    printf("C here");
    printf("!");
}
```

הפלט המתקבל מהרצת תוכנית זו זהה לפלט שהתקבל מהרצת תוכנית 4.2:

```
Hey, we are learning C here!
```

ירידה לשורה חדשה

כשהרצת את שתי התוכניות האחרונות (עשית זאת, נכון?) ודאי שמת לב לבעיה קטנה: סמן הטקסט לא ירד לשורה חדשה אחרי כל קריאה ל-printf. עקב כך, המשפטים הופיעו כל פעם בזה אחר זה. למשל, הפלט המתקבל מהרצת שתי התוכניות האחרונות הוא כזה:

```
Hey, we are learning C here!Hey, we are learning C here!
```

לעיתים קרובות ברצוננו להציג מלל לאורך מספר שורות. כיצד, אם כן, נגרום לסמן הטקסט לרדת לשורה חדשה? ובכן, לשם כך נדפיס על המסך תו מיוחד, הנראה כך: '\n'. הלוכסן ההפוך והאות n, כאשר הם מופיעים בזה אחר זה, נחשבים לתו בודד הגורם לירידה לשורה חדשה כשהוא נפלט. לכן, הקריאה הבאה ל-printf:

```
printf("\n");
```

תוריד את סמן הטקסט לשורה חדשה.

שים לב!

הלוכסן בתו '\n' הוא לוכסן הפוך (backslash) ולא הלוכסן הרגיל בו משתמשים לפעולות חילוק (/).

אם התו '\n' מופיע כחלק ממשפט מלל הנשלח ל-printf, תתבצע ירידה לשורה חדשה במקום בו הוא הופיע. ביצוע המשפט הבא, למשל:

```
printf("I can be myself\nhow 'bout you?");
```

יביא לפליטת מלל הנפרש לאורך שתי שורות, כך:

```
I can be myself
how 'bout you?
```

באותה הוראת printf ניתן לרדת מספר שורות, או לא לרדת לשורה חדשה כלל. למשל:

```
printf("Top of the");
printf(" world\n\nMiddle of nowhere\n");
printf("\nFrom the bottom of my heart");
```

כצפוי, שלוש ההוראות יגרמו לפליטת המלל הבא:

```
Top of the world

Middle of nowhere

From the bottom of my heart
```

בסיום קטע התוכנית יישאר סמן הטקסט בסוף השורה האחרונה שהוצגה, מפני שהתו '\n' לא הופיע בסוף המלל שנשלח ל-printf בפעם האחרונה.

עצה

מומלץ להוריד את סמן הטקסט לשורה חדשה בסוף כל תוכנית.

הערות

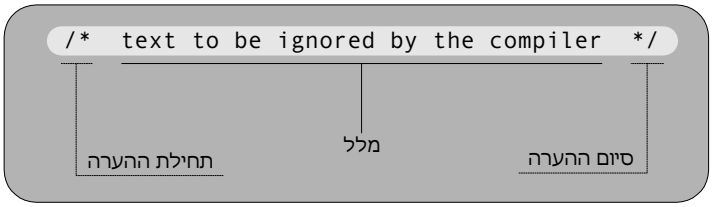
כאשר מפתחים תוכניות קצרות כגון אלו שהוצגו בסעיפים הקודמים, אין קושי בהבנת תפקידה של כל הוראה בתוכנית. אך כשהתוכנית מורכבת וארוכה יותר, קשה לזכור את הסיבה לכתיבתה של כל הוראה והוראה. הקושי אף עולה כשהתוכנית נכתבה לפני זמן ממושך או על ידי מתכנת אחר.

דרך נאה להתמודד עם בעיות אלו היא שימוש ב**הערות** (comments). בשפת C, הערות הן משפטים המיועדים רק לאדם הקורא את התוכנית. מהדר השפה מתעלם ממשפטים אלה ואינו אוכף עליהם את חוקיה התחביריים של השפה, ולכן הם יכולים להכיל כל מלל שהוא. מכיון שהמהדר לא מתרגם הערות לשפת סף ולשפת מכונה, הן אינן משפיעות על קובץ ההרצה (כלומר, פעולת התוכנית).

כיצד נבדיל בין הערות לבין הוראות שברצוננו שהמהדר יתייחס אליהן? ובכן, כל משפט שנרצה להפכו להערה ייתחם בתווים מיוחדים: הוא יתחיל בתווים */ ויסתיים בתווים /*. למשל, המשפט הבא:

```
/* So why don't you slide */
```

נחשב להערה, ואין לו שום משמעות תכנותית. התבנית הכללית לשימוש בהערות מוצגת להלן:



***איור 4.2** – אופן כתיבת הערות*

ניתן לפרוש הערה על-פני מספר שורות, כך:

```
/* This is a bunch of comments,
   spreaded on several lines of code.
   The compiler won't translate them!
*/
```

ואף בהמשך שורה של הוראה אחרת. למשל, השורה הבאה:

```
printf("Hello!");      /* don't let the user fall asleep */
```

מכילה הוראת printf שמתבצעת כרגיל, והערה שאינה משפיעה על התוכנית. עליך לזכור לסיים את מקטע ההערות: כל הערה הנפתחת בתווים */ חייבת להסתיים בתווים /*.

שים לב!

התווים התוחמים את ההערות מורכבים מלוכסן רגיל (בו משתמשים לפעולת חילוק) ולא מלוכסן הפוך (בו משתמשים לכתיבת התו '\n').

דוגמה

נציג עתה תוכנית שלמה הכוללת מספר הוראות והערות:

תוכנית 4.4 *COMMENTS.C*

```
1   /*****************************
2   *         COMMENTS.C          *
3   *  programmed by Ziv Ayalon   *
4   *  All rights reserved! (c)   *
5   *****************************/
6
7   #include <stdio.h>
8   void main()   /* this is the main function */
9   {
10      printf("This is line 1 \n");
11      /* printf("This is line 2 \n"); */
12      printf("This is line 3 \n");
13  }
```

הפלט המתקבל מהתוכנית הוא כדלהלן:

```
This is line 1
This is line 3
```

זכור כי כל המצוי בתחום התווים */ ו-/* אינו מתורגם על ידי המהדר, ולכן אינו נחשב כהוראה בשפת C. זו הסיבה ששורה (11) לא מביאה לפליטת מלל למסך. למעשה, התוכנית שלעיל שקולה לתוכנית הבאה:

```
#include <stdio.h>
void main()
{
    printf("This is line 1 \n");
    printf("This is line 3 \n");
}
```

בתוכנית זו מודגמים שלושה שימושים טיפוסיים להערות:

- שמירה על זכויות יוצרים – כאשר קוד המקור של תוכניתך מופץ לאנשים רבים, ברצונך לוודא כי כולם מודעים לכללים החלים עליו (למשל, שאין להעתיקו או לשנותו). דוגמה להערה המכילה מידע אודות מפתח התוכנה מופיעה בשורות (1)-(5).
- תיאור קוד המקור – מומלץ להוסיף הערות לקוד המקור של תוכניות מורכבות כדי להקל על קריאתן. התוכניות הראשונות שנכתוב יהיו פשוטות למדי, אך בהמשך נפתח תוכניות מורכבות וניעזר רבות בהערות. דוגמה להערה המסבירה הוראת קוד מופיעה בשורה (8).
- ניפוי שגיאות – מלאכת **ניפוי השגיאות** (debugging) עוסקת במציאת שגיאות בתוכנית שלא פעלה כמצופה. אחת הדרכים למציאת מקור הבעיה היא חלוקת התוכנית למספר קטעים והידור כל אחד מהם בנפרד. כך ניתן לבדוק את נכונותו של כל חלק וחלק בתוכנית. אם ברצוננו לגרום למהדר להתעלם מחלק כלשהו בתוכנית, נוח להפוך חלק זה להערה. דוגמה להערה "מחביאה" חלק מהתוכנית מהמהדר מופיעה בשורה (11).

נציג עתה מספר כללים נוספים הנוגעים לשימוש בהערות. ראשית, אין להוסיף תו רווח בין תווי תוחמי ההערות. כלומר, עליהם להיראות כך:

/* */

אך לא כך:

/ * * /

שנית, במקומות מסוימים בתוכנית אין להוסיף הערות. למשל, חל איסור על כתיבת הערה באמצע שמה של הוראה, כפי שמוצג בשורה השגויה שלהלן:

```
pri/*comment*/ntf("Hello!");
```

ולבסוף, אין לשלב הערה בתוך הערה נוספת. למשל:

```
1   /*
2       you can write many things here...
3       or here...
4       but nested comments are not allowed.
5       /* therefore, this line is illegal */
6   */
```

ההערה המופיעה בשורה (5) מצויה בתוך הערה המשתרעת על-פני שורות (1)-(6). מבנה של הערה הנמצאת בתחומה של הערה אחרת מכונה **קינון הערות** (comment nesting) והוא אינו מותר בשפת C.

תווי בקרה

תווי בקרה (escape sequence) הם תווים המבצעים פעולות פלט מיוחדות כאשר הם נפלטים. תו בקרה אחד שהכרנו הוא תו הירידה לשורה חדשה: '\n'. כאשר הוא נמצא במלל הנשלח ל-printf, במקום להציגו על המסך כמו שהוא (לוכסן הפוך והאות n), מתבצעת ירידה לשורה חדשה.

הטבלה הבאה מציגה את תווי הבקרה הנפוצים:

תו בקרה	תפקידו
\n	ירידה לשורה חדשה
\t	מעבר לטאב האופקי הבא
\a	צלצול בפעמון המחשב
\\	הצגת לוכסן הפוך
\"	הצגת גרשיים

קל לראות כי כל תו בקרה מורכב מלוכסן הפוך ומתו המתוסף לו. אף שמדובר בשני תווים נפרדים, כשהם באים יחדיו הם נחשבים לתו בודד.

איך לא...

הקפד לא להוסיף רווח בין הלוכסן ההפוך לבין התו שאחריו. למשל, התו '\ n' לא יביא לירידה לשורה חדשה. כמו כן, התו המופיע לאחר הלוכסן ההפוך חייב להיכתב באות קטנה.

נדון עתה בכל תו בקרה המוצג בטבלה שלעיל. את תו הבקרה '\n' כבר הכרנו, ולכן נדלג לתו הבקרה הבא – '\t'. תו בקרה זה מזיז את סמן הטקסט לטאב האופקי הבא, כלומר, לנקודת הדילוג הבאה. החל מהתו הראשון בשורה, כל תו שמיני נחשב לנקודת דילוג. בחן, למשל, את הוראות הפלט הבאות:

```
printf("12345678901234567890123456789012345678901234567890 \n");
printf("hi,\thow\tare\tyou? \n");
printf("I\tam\tfine,\tthanks. \n");
```

כל הופעה של התו '\t' מזיזה את סמן הטקסט אל התו השמיני הבא בשורה, כך שהפלט המתקבל מיושר בסגנון טבלאי:

```
12345678901234567890123456789012345678901234567890
hi,     how     are     you?
I       am      fine,   thanks.
```

תו הבקרה הבא, '\a', מביא לצלצול בפעמון המחשב כאשר הוא מופיע במלל הנשלח למסך. למשל, הוראת ה-printf הבאה:

```
printf("Nuclear Launch Detected! \a");
```

תציג את המשפט המצמרר על המסך ותשמיע צלצול בפעמון המחשב.

שים לב!

ייתכן שבמחשבך לא תשמע את צלצול הפעמון כיאות. אם ברשותך כרטיס קול, הוא עלול למנוע את השמעת הצליל.

תו הבקרה הבא, '\\', מביא להצגת הלוכסן ההפוך על המסך. למשל, הוראת ה-printf הבאה:

```
printf("Are you tired? Please answer yes\\no: ");
```

תגרום לפליטת המלל הבא:

```
Are you tired? Please answer yes\no:
```

אילו ניסינו לפלוט את המלל בלי שימוש בתו הבקרה '\\', כך:

```
printf("Are you tired? Please answer yes\no: ");
```

היה נפלט המלל הבא:

```
Are you tired? Please answer yes
o:
```

הסיבה לירידת השורה הפתאומית ולהיעלמות המלל "\n" היא הופעת תו הבקרה '\n' בסוף המלל הנשלח ל-printf: "yes**\n**o:".

תו הבקרה האחרון בו נעסוק עתה, '\"', מיועד לפליטת תו הגרשיים. נדגים את השימוש בו בהוראת ה-printf הבאה:

```
printf("The officer said: \"don't rob banks\".");
```

ההוראה מביאה לפליטת המלל הבא:

```
The officer said: "don't rob banks".
```

נוכחנו שאם ברצוננו להציג גרשיים על המסך, חובה להקדימם בלוכסן הפוך. הסיבה לכך היא, שתו הגרשיים לבדו מיועד לתחימת המלל הנשלח ל-printf, ואילו הוראת ה-printf נכתבה כך:

```
printf("The officer said: "don't rob banks".");
```

היתה מתקבלת הודעת שגיאה, שכן הגרשיים באמצע המשפט מציינים את סוף המלל הנשלח ל-printf (המשפט don't rob banks יימצא מחוץ למלל), ולכן המשך השורה נחשב לשגיאה.

דוגמה

נדגים את השימוש בתווי בקרה על ידי התוכנית הבאה:

תוכנית 4.5 *ESC_SEQ.C*

```
/*
 *  This program demonstrates the
 *  use of escape sequence. Enjoy!
 */

#include <stdio.h>
```

```
void main()
{
    /* display the beautiful song */
    printf("\"All day I hear the noise of waters \n");
    printf("\tMaking moan, \n");
    printf("Sad as the sea-bird is, when going \n");
    printf("\tForth alone, \n");
    printf("He hears the winds cry to the Waters' \n");
    printf("\tMonotone.\" \n");

    /* display the writer's name */
    printf("\n\t\t\tJames Joice \\ Chamber Music \n");

    /* ring the bell */
    printf("\a\a");
}
```

תוכנית זו מביאה לפליטת שירו של ג'יימס ג'ויס המפורסם:

```
"All day I hear the noise of waters
        Making moan,
Sad as the sea-bird is, when going
        Forth alone,
He hears the winds cry to the waters'
        Monotone."

                        James Joice \ Chamber Music
```

מרכיבי השפה

כפי ששפה מדוברת מורכבת משמות, מפעלים, ממילות קישור וכדומה – כך גם שפת C נבנית ממספר חלקים. חלקים אלה משולבים, על-פי חוקי התחביר של השפה, ליצירת תוכנית חוקית בשפת C. בסעיף זה נסקור את רכיבי שפת C.

שים לב!

סעיף זה הוא מעין נספח עיוני. מומלץ לקרוא אותו בלי להתעמק בתוכנו יתר על המידה, שכן רוב המושגים המוזכרים בו יוסברו בהמשך טוב יותר.

מילות מפתח

מילת מפתח (keyword) או **מילה שמורה** (reserved word) היא מילה בעלת משמעות מיוחדת למהדר. כל מילה שמורה בנויה במהדר, ולה תפקיד ספציפי ביותר שלא ניתן לשנותו על-פי צרכינו. אין להשתמש במילה שמורה בכל צורה שאינה תואמת בדיוק את תפקידה.

במהלך הלימוד נכיר את רוב המילים השמורות של שפת C, המוצגות להלן:

```
auto        double      int         struct
break       else        long        switch
case        enum        register    typedef
char        extern      return      union
const       float       short       unsigned
continue    for         signed      void
default     goto        sizeof      volatile
do          if          static      while
```

מזהים

מזהה (identifier) הוא מילה המוגדרת על ידי המתכנת. מזהים לדוגמה הם שמות משתנים ושמות פונקציות. פרט למקרים מיוחדים, כל מילה בתוכנית שאינה מילה שמורה היא מזהה. כל מזהה מורכב מרצף של תווים, המייחד אותו ממזהים וממרכיבי שפה אחרים. על כל מזהה להיות שונה בשמו מהמילים השמורות וברוב המקרים – גם מיתר המזהים שבתוכנית. נרחיב אודות החוקים לקביעת שמות מזהים בהמשך.

קבועים

קבוע (constant או literal) הוא ערך שאינו משתנה במהלך ריצת התוכנית. הקבועים נחלקים למספר סוגים:

- קבועים מספריים שלמים, כמו 34 ו-90000.
- קבועים מספריים עשרוניים, כמו 12345.678 ו-45.0.
- קבועים תוויים, כמו 'a\' ו-'n\'. שים לב כי הם תחומים בגרש יחיד בכל צד.
- קבועים מחרוזתיים, כמו "string" ו-"constant". שים לב כי הם תחומים בגרשיים.

אופרטורים

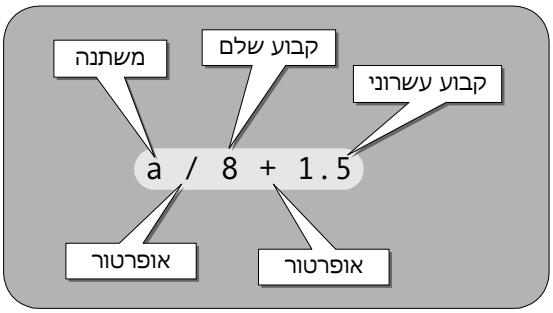
אופרטורים (operators) הם סימנים מיוחדים המאפשרים ליצור ביטויים שונים. בפרק הראשון הכרנו מספר אופרטורים המייצגים פעולות מתמטיות שכיחות: + (חיבור), - (חיסור), * (כפל), / (חילוק) ו-% (שארית). בשפת C אופרטורים רבים (מספרם רב משל מספר המילים השמורות!), ונכירם בהמשך הלימוד.

כאמור, כאשר משלבים אופרטורים וערכים יחדיו, מתקבל **ביטוי** (expression). למשל, כאשר המשתנה b והקבוע המספרי 4 מופיעים בנפרד, אין להם משמעות מיוחדת – אך כאשר נשלבם יחד עם אופרטור כלשהו (למשל, b + 4), יתקבל ביטוי בו ניתן להשתמש בדרכים רבות.

שים לב!

חשוב לציין שתו הנקודה-פסיק אינו מהווה חלק מביטוי. למשל, בעוד ש-b + 4 הוא ביטוי, **;**b + 4 נחשב להוראה.

באיור הבא מוצג ביטוי לדוגמה, המורכב מאופרטורים, מקבועים וממשתנים:



***איור 4.3** – שילוב ערכים ואופרטורים ליצירת ביטוי*

סימני פיסוק

סימני פיסוק (punctuators) הם תווים מיוחדים בעלי שימושים רבים ומגוונים. למרבית סימני הפיסוק יותר מתפקיד אחד, והתפקיד המתאים להם נקבע בהתאם להקשר בו הם מופיעים.

אלה הם סימני הפיסוק הקיימים בשפה:

```
{   }   (   )   [   ]   *   =   #   ...   :   ;   ,
```

במספר סימני פיסוק כבר נתקלנו עד כה; למשל, הסוגריים המסולסלים ותו הנקודה-פסיק.

יש לציין שתי נקודות בנוגע למרכיבי השפה שתוארו להלן. ראשית, בשפת C קיימת הבחנה בין אותיות קטנות לגדולות. לכן, המזהים price, Price ו-PRICE הם שלושה מזהים שונים; ל-printf אין לקרוא Printf; להוראה include# אין לקרוא INCLUDE#; תו הבקרה '\N' לא יגרום לירידת שורה וכדומה.

כמו-כן חשוב לציין, כי מילים המופיעות בתוך מלל או בתוך בלוק הערות אינן מסווגות למרכיבי השפה שתוארו לעיל. למשל, בהוראת הפלט הבאה:

```
printf("How is this evaluated?  a / 8 + 1.5");
```

המספר 1.5 אינו נחשב לקבוע עשרוני, הלוכסן אינו נחשב לאופרטור וכדומה. באופן דומה, ההערה הבאה:

```
/*
 *  These are NOT considered to be keywords:
 *  while, do, for
 */
```

מכילה אמנם מילים הזהות למילים השמורות while, do ו-for, אך הן נמצאות בתחום שאינו נבדק על ידי המהדר, ולכן אינן נחשבות למילים שמורות.

מהי פונקציה?

סדרת הוראות המבצעת מטלה שלמה כלשהי מכונה **פונקציה** (function). המתכנת רשאי להגדיר פונקציות משלו ולקבוע להן הוראות שונות. למשל, פונקציה טיפוסית יכולה לחשב עצרת של מספר כלשהו, וניתן להפעיל אותה עם ערכים שונים. אם נרצה להפעילה מספר פעמים, לא נצטרך לכתוב את כל ההוראות הדרושות לחישוב העצרת שוב ושוב; כל שנעשה הוא לכתוב את שמה, וההוראות השייכות לה יתבצעו באופן אוטומטי.

נמחיש את הנאמר לעיל על ידי דוגמה מחיי היום-יום. אחד הדברים הראשונים שאדם עושה כשהוא קם בבוקר הוא צחצוח שיניו. לשם כך, עליו לבצע פעולות רבות, המזוהות יחדיו תחת המושג "צחצוח שיניים". לכן, נוכל לומר כי המושג "צחצח את שיניך" הוא פונקציה, שהוראותיה הן הפעולות שעושה האדם במהלך הצחצוח (נטילת מברשת, נטילת משחה, לחיצה על השפופרת, מריחת משחה על קצה המברשת וכדומה). עתה, מספיק שנאמר לאדם "צחצח את שיניך", והוא יבצע את כל ההוראות הקשורות למושג זה – בלי שנצטרך לפרט לו כל הוראה והוראה.

שים לב!

פונקציה בשפת C אינה זהה לפונקציה מתמטית. פונקציה בשפת C יכולה אמנם לקבל ערכים ולהחזיר ערכים (כפי שנראה בהמשך), אך לא חלות עליה הגבלות כמו אלו החלות על פונקציות מתמטיות.

בהמשך נלמד כיצד לכתוב פונקציות בעצמנו. בינתיים נשתמש רבות בפונקציות שנכתבו עבורנו. למשל, printf היא פונקציה שנכתבה על ידי מפתחי המהדר. מכיון שהוראותיה הוגדרו כבר על ידי מתכנתים אחרים, כל שצריך לעשות כדי לבצען הוא לקרוא לה בשמה. זאת בדומה לאדם המבין את המושג "צחצח את שיניך" – אין צורך לפרט לו אילו פעולות עליו לבצע לשם צחצוח שיניו; כל שצריך לעשות הוא להורות לו בקצרה: "צחצח את שיניך".

קריאה לפונקציה

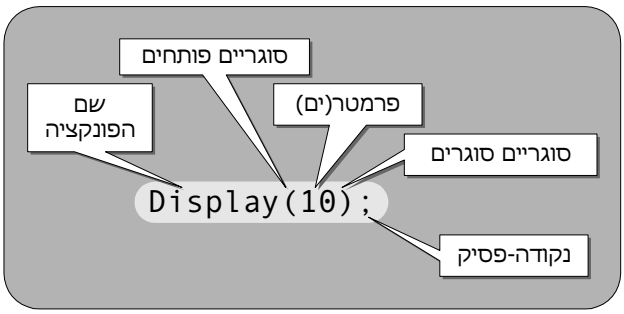
פונקציות הן מרכיב מרכזי בשפה ועלינו לדעת כיצד להפעילן. כדי לגרום לביצוע הוראות פונקציה מסוימת יש לקרוא לה. לעיתים קרובות, בקריאה אל הפונקציה נשלחים אליה מספר ערכים, המכונים **פרמטרים** (parameters). לרוב, מבצעת הפונקציה פעולות שונות באמצעותם (פליטת הפרמטר, חישוב העצרת שלו וכדומה).

הקריאה לפונקציה נעשית על ידי כתיבת שמה והוספת סוגריים פותחים וסוגרים אחריו. אם הפונקציה מקבלת פרמטרים, הם ייכתבו בין הסוגריים ויופרדו בפסיק זה מזה. הקריאה לפונקציה תסתיים לרוב בתו הנקודה-פסיק.

נניח כי קיימת פונקציה בשם Display, ובה הוראות הפולטות למסך את הערך המספרי שנשלח אליה. על-פי החוקים שהכרנו זה עתה, הוראה שתגרום להפעלת הפונקציה ולהצגת המספר 10 על המסך תיראה כך:

```
Display(10);
```

התרשים הבא מתאר אופן קריאה כללי לפונקציה:



***איור 4.4** – קריאה לפונקציה*

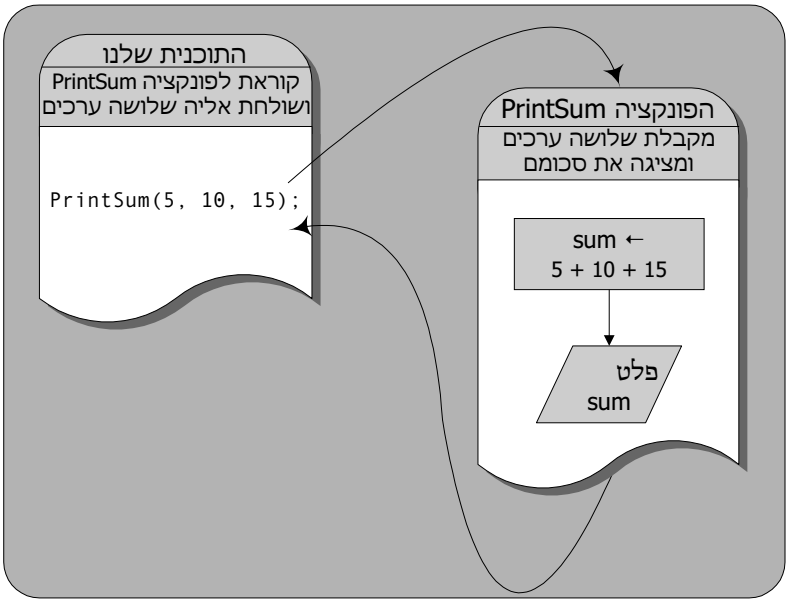
נביא דוגמה נוספת: נניח כי קיימת פונקציה בשם PrintSum, המקבלת שלושה פרמטרים מספריים ומציגה על המסך את סכומם. כדי לקרוא לה נכתוב את שמה, ובין הסוגריים המופיעים אחריו נכתוב שלושה ערכים מספריים, המופרדים זה מזה בפסיק. את ההוראה נסיים בנקודה-פסיק. קריאה מתאימה לפונקציה תיראה כך:

```
PrintSum(5, 10, 15);
```

עצה

מקובל להוסיף רווח בין הפסיק לפרמטר הבא אחריו כדי להקל על הקריאה. מומלץ לסגל סגנון כתיבה תכנותי ברור ככל הניתן, ולנהוג על-פיו לאורך כל התוכנית.

לאחר שהסתיימו הוראות הפונקציה, תתבצע ההוראה שלאחר ההוראה שקראה לה. האיור הבא מדגים כיצד מבוצעות הוראות פונקציה כגון PrintSum כאשר קוראים לה:



***איור 4.5** – קריאה לפונקציה וחזרה ממנה*

גם אם הפונקציה מכילה מאות הוראות, הקריאה לה קלה לכל הדעות. הפונקציה printf, למשל, מכילה הוראות רבות – וכולן מבוצעות עבורנו בעקבות קריאה פשוטה אחת.

קבצי ספריה

כפי שאמרנו, המהדר אינו תומך בפעולות מורכבות כגון פליטת מלל למסך, ביצוע חישובים מורכבים וכדומה. כלומר, לא קיימת הוראה האומרת ישירות למהדר "הצג את הערך הזה-והזה על המסך". אך פעולות אלו הכרחיות לכל מתכנת, ולכן הן מסופקות כהרחבה למהדר: לכל כלי פיתוח בשפת C מצורפים קבצים בהם מוגדרות פונקציות, להן יכול המתכנת לקרוא מתוך תוכניתו. קבצים אלה מכונים **קבצי ספריה** (library files).

קבצי הספריה הם קבצים בשפת מכונה, המשולבים יחד עם תוכניתנו בעת תהליך הקישור. בקבצי הספריה מוגדרות פונקציות רבות ומגוונות, המכונות **פונקציות ספריה** (library functions). בשל השימוש הרב בהן הן מהוות חלק אינטגרלי מהשפה. להלן דוגמאות לפונקציות ספריה שכיחות:

- **פונקציות מתמטיות** – לחישוב לוגריתם, שורש, חזקה וכדומה.
- **פונקציות זמן ותאריך** – לפנייה אל הזמן ואל התאריך במחשב.
- **פונקציות גרפיקה** – להצגת צורות גרפיות על המסך.
- **פונקציות מחרוזות** – לביצוע פעולות על מלל.

העובדה שבמהדר לא מובנות פעולות מורכבות נראית בתחילה כטרדה – אך למעשה, היא מעצימה תוכניות הכתובות בשפת C, וזאת משתי סיבות. ראשית, המהדר קטן וקומפקטי יותר, ולכן מהיר יותר. שנית, הדבר תורם רבות לניידותה של השפה: המהדר מכיר רק הוראות ממוחשבות בסיסיות ביותר, ולכן ניתן להעביר תוכנית ממערכת הפעלה אחת לאחרת בלי לכתוב אותה מחדש. אף שכל מערכת הפעלה מיישמת את הפונקציות שבקבצי הספריה באופן אחר, הקריאה להן מתוך תוכניתנו נשארת זהה, בלי כל קשר למערכת ההפעלה בה אנו עובדים.

סיכום

- כל הוראה שתיכתב בין הסוגריים המסולסלים של הפונקציה הראשית תבוצע כאשר התוכנית מורצת.
- פונקציית הפלט printf מקבלת מלל התחום בגרשיים ומציגה אותו על המסך.
- כאשר התו '\n' נפלט על המסך, סמן הטקסט יורד לשורה חדשה.
- נהוג להשתמש בהערות כדי לתעד את קוד המקור.
- התווים '\n', '\t', '\a', '\\' ו-'\"' מכונים **תווי בקרה**. קל לזהותם על-פי הלוכסן ההפוך בו הם מתחילים.
- מילה שמורה היא מילה המבצעת פעולה קבועה ומוגדרת שמכיר המהדר.
- קבוע הוא ערך שאינו משתנה במהלך התוכנית.
- אופרטור הוא סימן המבצע פעולה מוגדרת כלשהי על ערכים.
- שילוב של אופרטורים וערכים מניב ביטוי.
- פונקציה היא תת-תוכנית המבצעת מטלה שלמה כלשהי, והוראותיה מבוצעות כאשר היא נקראת.

פרק 5

משתנים

בפרק הראשון הכרנו לראשונה את המשתנים, באמצעותם קלטנו ערכים מהמשתמש וביצענו חישובים שונים. המשתנים בשפת C מיועדים לאותה מטרה, אך השימוש בהם הוא מעט יותר מורכב. למשל, ניתן להגדיר משתנים מסוגים שונים ולהציב בהם ערכים באופנים רבים יותר משהכרנו עד כה. השימוש במשתנים אינו קשה כלל, אך הוא ישפר את תוכניותינו עד מאוד.

מהו משתנה?

משתנים (variables) הם, כאמור, אמצעי אחסון לנתונים. גם בשפת C נשתמש תכופות במשתנים. השימוש בהם הכרחי אם ברצוננו לקלוט ערכים מהמשתמש, לבצע חישובים מתמטיים מורכבים ועוד.

משתנים אינם קיימים בתוכניתנו יש מאין. כל משתנה קיים בזיכרון המחשב ותופס שטח מסוים ממנו. בשפת C קיימים מספר סוגי משתנים ולכל סוג גודל משלו, כלומר, שטח שונה שהוא תופס בזיכרון.

במחשבה שנייה...

האם ייתכן מצב בו נגדיר משתנים רבים, שיתפסו את כל שטח הזיכרון?

הסיכוי לכך, לפחות בשלב זה של התכנות, קלוש. גם סוגי המשתנים הגדולים ביותר תופסים שטח מזערי מאוד מסך-כל שטח הזיכרון. למשל, כיום מחשבים חדישים כוללים 128 מיליון תאי זיכרון, בעוד שהמשתנה הגדול ביותר בשפת C תופס, לרוב, כעשרה תאי זיכרון בלבד!

כזכור, כל משתנה מיוצג על ידי שם, המבדיל אותו ממשתנים אחרים. שמות משתנים הם מילים המורכבות מאותיות לועזיות, מספרות ומקו תחתי (_). שמות המשתנים יכולים להכיל תו אחד או תווים רבים (רוב המהדרים יקבלו לפחות 31 תווים בשמות משתנים), אך אסור שיתחילו בספרה. בשלב זה לא נרחיב בנושא, משום שהתעמקנו בו בפרק 1. חזור לנאמר בפרק זה במידת הצורך.

כמו-כן, אין לקרוא למשתנים בשמות של מילים שמורות בשפת C. למשל, שמות המשתנים for ו-if אינם חוקיים, משום שקיימות מילים שמורות כאלו. לעת עתה נקבע, כי אין לקרוא לשני משתנים השייכים לאותה תוכנית באותו שם.

איך לא...

שמות המשתנים הבאים אינם נכונים:

`98salary`	אין להתחיל שם משתנה בספרה
`c++`	התו '+' אינו מותר בשם משתנה
`ben&jerrys`	התו '&' אינו מותר בשם משתנה
ממוצע	אין לקרוא לשם משתנה בעברית

הצבת ערכים

בפרק השני ראינו כיצד להציב ערכים במשתנים באמצעות פעולת הצבה אלגוריתמית, כדוגמת זו:

```
var ← 12
```

כזכור, בכל פעולת הצבה כזו מפריד חץ בין שני שדות: השדה השמאלי הוא משתנה, והשדה הימני הוא ביטוי שערכו מוצב במשתנה.

גם בשפת C ניתן להציב ערכים במשתנים. בדומה להוראת ההצבה שלעיל, בהוראת הצבה בשפת C קיימים שני שדות, משתנה וביטוי, אך הם מופרדים זה מזה על ידי סימן שוויון (=) ולא על ידי חץ. סימן שוויון זה מכונה **אופרטור השמה** (assignment operator). למשל, הוראת הצבה בשפת C השקולה להוראה האלגוריתמית שלעיל נראית כך:

```
var = 12;
```

הוראה בשפת C המציבה ערך במשתנה נקראת **משפט הצבה** או **משפט השמה** (assignment statement). שים לב כי לאחר פעולת ההצבה הוספנו נקודה-פסיק. בינתיים נקבע כי חובה על תו זה להופיע בסופה של כל הוראת הצבה.

כאן המקום להגדיר מהו **משפט** (statement) בשפת C. משפט בשפת C הוא הוראה המבצעת פעולה זו או אחרת, כגון פליטת מלל והצבת ערכים במשתנים. כל משפט (פרט לסוג מיוחד אותו נכיר בהמשך) מסתיים בנקודה-פסיק. התוכנית מורכבת ממשפטים, ואלה מורכבים מחלקים תחביריים נוספים.

כפי שראינו בפרק הראשון, ניתן להציב במשתנים ביטויים מורכבים הכוללים ערכי משתנים אחרים ואופרטורים אריתמטיים. מתוך האופרטורים האריתמטיים שהכרנו בפרק הראשון, באופרטורים הבאים ניתן להשתמש בשפת C:

```
-    +    *    /    %    (    )
```

לעומתם, האופרטורים של חזקה ושורש:

$$\sqrt{\quad} \qquad ^{\wedge}$$

לא קיימים בשפת C, ולכן אסור להשתמש בהם. אם נרצה לחשב חזקה או שורש של מספר כלשהו, נשתמש בפונקציות מיוחדות שנכיר בהמשך.

הביטוי שערכו מוצב במשתנה נכתב בדומה לביטוי שהופיע בפעולת ההצבה האלגוריתמית. למשל, כדי להציב במשתנה x את שארית חלוקת המספר 14 ב-4, נבצע משפט הצבה כזה:

```
x = 14 % 4;      /* x now equals 2 */
```

כדי להציב במשתנה num את מכפלת ערך המשתנה a ב-2, נבצע משפט הצבה כזה:

```
num = a * 2;      /* num now equals a + a */
```

סימן המינוס משמש הן לקבלת הפרש שני מספרים והן לייצוג ערך נגדי של מספר. למשל, המשפט הבא:

```
diff = 10 - 6;      /* diff now equals 4 */
```

מציב במשתנה diff את הפרש המספרים 10 ו-6. לעומתו, המשפט הבא:

```
neg = -num;      /* neg now equals -num */
```

מציב במשתנה neg את הערך הנגדי לערך שבמשתנה num (אם, למשל, המשתנה num מכיל את הערך 7, במשתנה neg יוצב הערך 7-).

שים לב!

אם תבצע השמה בין שני משתנים, ומאוחר יותר ישתנה הערך של אחד מהם, ערך המשתנה השני לא ישתנה. פעולת ההשמה היא חד-פעמית ואינה מבטיחה כי ערכי המשתנים לא ישתנו בהמשך.

סדר הפעולות בשפת C זהה לסדר הפעולות במתמטיקה. למשל, כפי שבמתמטיקה פעולת הכפל קודמת לפעולת החיבור, כך בשפת C קודמת פעולת אופרטור הכפל לזו של אופרטור החיבור. את סדר הפעולות בשפת C נהוג לרשום בטבלה, המציגה אילו מהאופרטורים

מבוצעים קודם. טבלת קדימויות האופרטורים בשפת C היא כזו (הטבלה ערוכה בסדר יורד, כלומר, ככל שהאופרטור קרוב לראש הטבלה, קדימותו גבוהה יותר):

אופרטורים	הערות מיוחדות
( )	
- +	אופרטורי פלוס ומינוס אלה מופיעים לפני ערך בודד וקובעים את סימנו (חיובי או שלילי)
% / *	
- +	אופרטורי פלוס ומינוס אלה מופיעים בין שני ערכים ומבצעים פעולת חיבור או חיסור ביניהם
=	

שים לב!

פעולות מתמטיות הזהות בקדימותן תבוצענה משמאל לימין. למשל, ערך הביטוי 3 * 2 / 6 הוא כשל הביטוי 3 * (2 / 6), כלומר, 9.

לכן, כדי להציב במשתנה average את הממוצע של שלושה ציונים, המיוצגים על ידי המשתנים grade1, grade2 ו-grade3, נבצע משפט השמה הנראה כך:

```
average = (grade1 + grade2 + grade3) / 3;
```

מותר להוסיף סוגריים עודפים מסביב לקבועים, לשמות משתנים ולביטויים, אך רצוי לא להגזים בכך, כדי לא להקשות על קריאת התוכנית. למשל, את המשפט הקודם ניתן לכתוב כך:

```
average = (((grade1) + (grade2) + (grade3)) / (3));
```

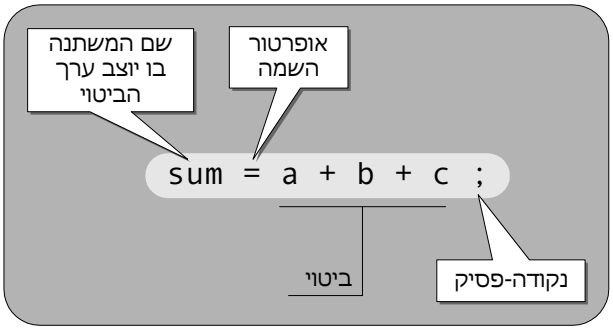
זהירות!

כמו במתמטיקה, גם בשפת C אסור לחלק מספר ב-0. בנוסף, אסור לחשב שארית חלוקת מספר ב-0. לכן, שני משפטי ההשמה הבאים:

```
min = num / 0;      /* dangerous! */
far = low % 0;      /* dangerous! */
```

הם נכונים מבחינה תחבירית, אך מסוכנים. ביצועם עלול לגרום לקריסת התוכנית.

האיור הבא מציג את תבנית הכתיבה של משפט ההשמה:



***איור 5.1** – כתיבת משפט השמה*

איך לא...

משמאל לאופרטור ההשמה חייב להופיע משתנה – ומשתנה אחד ויחיד. משפטי ההשמה הבאים, למשל, אינם נכונים:

```
4 = x;
x + 1 = 7;
```

סוגי משתנים

כל משתנה בשפת C שייך לסוג משתנה כלשהו, אותו בוחר המתכנת בהתאם לצרכיו. בסעיף זה נכיר שני סוגי משתנים:

- משתנה שלם – מיועד לטיפול במספרים שלמים, כגון 95 ו-2020.
- משתנה עשרוני – מיועד לטיפול במספרים בעלי נקודה עשרונית, כגון 12.5. חשוב לציין שכל מספר בו מופיעה נקודה עשרונית נחשב לערך עשרוני – גם המספר (השלם לכאורה) 7.0. המספר 7, לעומתו, נחשב לערך שלם.

המשתנה השלם

במשתנה השלם ניתן להציב ערכים שלמים בלבד, שליליים או חיוביים (ו-0, כמובן). בשפת C קיימים מספר סוגי משתנים שלמים, אך עתה נכיר את הבסיסי והנפוץ ביותר, המכונה בלעז int (מלשון integer – שלם).

סוגי המשתנים השלמים הקיימים בשפה שונים זה מזה בתחום הערכים שניתן להציב בהם. משתנה int יכול להכיל את כל הערכים השלמים בין 32768- ל-32767.

עצה

אל תציב במשתנה ערך החורג מטווח הערכים שניתן להציב בו. זו אינה טעות תחבירית, אך הערך שיוצב בו באמת לא יהיה הערך שרצית.

אף שלא ניתן לאחסן במשתנה שלם ערך עשרוני, הצבת ערך עשרוני בו אינה נחשבת לטעות. בחן, למשל, את המשפט הבא:

```
i = 3.14;      /* i is an int variable */
```

משפט זה חוקי לגמרי, אך המשתנה i לא יכיל את הערך העשרוני 3.14, אלא את הערך השלם 3. כלומר, הערך העשרוני עוגל כלפי מטה לערך שלם, המתאים למשתנה השלם. באופן דומה, אם נבצע את המשפט הבא:

```
i = 7.0 / 4.0;
```

הערך שיוצב במשתנה i הוא 1. הסיבה היא, שהערך העשרוני המתקבל מהביטוי 7.0 / 4.0 הוא 1.75, והוא מעוגל כלפי מטה כדי שהמשתנה השלם יוכל להכילו.

המשתנה העשרוני

בשפת C קיימים מספר סוגי משתנים עשרוניים, ועתה נכיר את הסוג העשרוני הבסיסי, המכונה בלעז float.

במשתנה מסוג float ניתן להציב ערכים עשרוניים שליליים וחיוביים (ו-0, כמובן), בטווח רחב יותר משל המשתנה השלם. משתנה מסוג float יכול להכיל ערכים הנעים בין $3.4 \cdot 10^{-38}$ ל-$3.4 \cdot 10^{38}$ בדיוק של 6 ספרות אחרי הנקודה העשרונית.

שים לב!

לא כל המספרים העשרוניים בטווח זה ניתנים לייצוג. לעיתים תיווכח כי מספר עשרוני אינו מוצג בדיוק, אלא בקירוב. למשל, אם נציב את הערך 4.1 במשתנה עשרוני, ייתכן כי יוצב בו הערך 4.099999.

הדרך הטבעית להציב ערכים במשתנה עשרוני היא כזו:

```
f = 2.71;      /* f is a float variable */
```

אם ננסה להציב במשתנה עשרוני ערך שלם, הערך השלם יומר לערך עשרוני, והוא הערך שיוצב במשתנה העשרוני. למשל, לאחר ביצוע המשפט הבא:

```
f = 3;      /* 3 is promoted to 3.0 */
```

יכיל המשתנה f את הערך העשרוני 3.0.

ניתן לייצג ערכים עשרוניים גם על ידי סימון מעריכי. למשל, בסימון מדעי נהוג לייצג מספר גדול כגון 1230000 כ-$1.23 \cdot 10^6$. בשפת C פעולת החזקה אינה קיימת, אך מספר כזה ניתן לייצוג כ-1.23e6. האות e מציינת את מכפלת המספר שמשמאלה בחזקה שבסיסה 10 והמעריך שלה – המספר שמימינה. המספר שמימינה יכול להיות גם שלילי; למשל, מספר קטן כגון 0.000123 ניתן לכתיבה בשפת C כ-1.23e-4. ערכים אלה הם קבועים לכל דבר.

שים לב!

בשפת C מספרים ארוכים נכתבים בלי פסיקים (למשל, 1230000 ולא 1,230,000).

אופרטור השארית נבדל משאר האופרטורים האריתמטיים בכך שלא ניתן להשתמש בו עם ערכים עשרוניים. למשל, הביטויים הבאים אינם נכונים:

```
8.5 % 4        /* wrong */
7 % 4e3        /* wrong */
```

ואם באופרטור השארית עסקינן, נציין כי יש להימנע מלהשתמש בו עם ערכים שליליים. בחן, למשל, את הביטוי הבא:

```
10 % -3
```

מצד אחד המספר 3- לא נכנס במספר 10 כלל, ולכן נצפה מהביטוי להחזיר את הערך 10. אך מצד שני המספר 3- נכנס 3- פעמים במספר 10, ולכן נצפה מהביטוי להחזיר את הערך 1. דו-משמעות כזו קיימת גם כאשר מחשבים מנת מספרים שלמים שליליים, כפי שמדגים הביטוי הבא:

```
5 / -2
```

כזכור, כאשר מתבצעת חלוקה בין שני ערכים שלמים, התוצאה מעוגלת כלפי מטה. אך כיצד נעגל מטה את מנת המספרים 5 ו-2-? אם נעגל את התוצאה למספר הקטן ביותר, יוחזר הערך 3-. אך אם נעגל את התוצאה למספר הקטן ביותר בערך מוחלט, יוחזר הערך 2-. התשובה לשאלה זו משתנה ממהדר למהדר, ולכן מקובל להימנע גם מחישוב מנת מספרים שליליים.

לעיתים קרובות נצטרך לכתוב תוכנית ולהחליט בעצמנו באילו סוגי משתנים להשתמש. לשם כך יש לחשוב בהיגיון; למשל, משתנה המייצג שנה (כגון 1983) יכול להיות משתנה שלם, אך משתנה המייצג את אורכי צלעותיו של מלבן (כגון 4.5) ראוי להיות משתנה עשרוני.

דיוק בערכים שלמים ועשרוניים

ביטוי שאף אחד מערכיו (קבועים או משתנים) אינו עשרוני מחזיר ערך שלם. למשל, הביטוי הבא:

```
7 / 4
```

מחזיר את הערך 1. שבע רביעיות הן אמנם 1.75, אך שני הערכים בביטוי הינם שלמים, ולכן הערך העשרוני מעוגל כלפי מטה כך שמתקבל ערך שלם. בעת מעבר מערך עשרוני לערך שלם תמיד יעוגל הערך כלפי מטה. למשל, כאשר הופכים את הערך העשרוני 1.999 לערך שלם, יתקבל המספר 1, ולא 2.

ערך הביטוי 4 / 7 יישאר שלם גם אם ננסה להציבו במשתנה עשרוני בשם f, כך:

```
f = 7 / 4;       /* same as:  f = 1; */
```

במקרה זה יכיל המשתנה f את הערך 1.0. המשתנה f הוא אמנם משתנה עשרוני, אך בעת חישוב ערך הביטוי 4 / 7 אין חשיבות לסוג המשתנה בו הוא מוצב. אם ברצוננו להחזיר את הערך המדויק של הביטוי, עלינו להפוך לפחות אחד מערכיו לעשרוני. נוכל, למשל, לעשות זאת כך:

```
7.0 / 4     /* equals 1.75 */
```

עתה, אם ערך ביטוי זה יוצב במשתנה עשרוני, כך:

```
f = 7.0 / 4;       /* same as:  f = 1.75 */
```

הוא יכיל את הערך המדויק 1.75.

הגדרת משתנים

כאשר השתמשנו במשתנה בפרק 1 פשוט כתבנו את שמו בהוראה אלגוריתמית. בשפת C, לעומת זאת, אין להשתמש במשתנה בלי להגדירו קודם-לכן. הגדרת המשתנה קובעת למשתנה את סוגו, ובעקבותיה יוקצה למשתנה שטח מתאים בזיכרון המחשב.

השורה בה מוגדר המשתנה מכונה **שורת הגדרה**, והיא מיידעת את המהדר על סוג המשתנה ועל שמו. היא נכתבת כך: ראשון מופיע סוג המשתנה (int או float), אחריו שם המשתנה, ולבסוף – תו הנקודה פסיק. למשל, כדי להגדיר משתנה שלם בשם age, נכתוב שורת הגדרה כזו:

```
int age;
```

וכדי להגדיר משתנה עשרוני בשם price, נכתוב את שורת ההגדרה הבאה:

```
float price;
```

עתה, כשראינו כיצד להגדיר משתנים ולהציב בהם ערכים, נכתוב תוכנית מלאה שתיישם את הידע הנלמד. תוכנית זו אינה פולטת דבר, אך היא מדגימה היטב את השימוש במשתנים.

תוכנית 5.1 *VARS.C*

```
void main()
{
    int i;       /* define an int variable */
    float f;     /* define a float variable */

    i = 45;
    f = 15.0;
    i = i / f;          /* i now equals 3   */
    f = f - 10.0;       /* f now equals 5.0 */
    i = i * i * i;      /* i now equals 27  */
}
```

```
    int i;       /* define an int variable */
    float f;     /* define a float variable */
```

בשורות אלו מוגדרים משתנה שלם בשם i ומשתנה עשרוני בשם f. זכור כי אין להשתמש במשתנים לפני שמגדירים אותם.

```
    i = 45;
    f = 15.0;
```

בשורות אלו מבוצעים שני משפטי השמה. הראשון מציב במשתנה השלם i את הערך 45, והשני מציב במשתנה העשרוני f את הערך 15.0.

```
    i = i / f;          /* i now equals 3   */
    f = f - 10.0;       /* f now equals 5.0 */
    i = i * i * i;      /* i now equals 27  */
```

שלושת משפטי ההשמה הללו מציבים ערכים שונים במשתנים i ו-f. המשפט הראשון מחלק את ערך המשתנה i בערך המשתנה f, המשפט השני מפחית מערך המשתנה f את הערך העשרוני 10.0 והמשפט השלישי מעלה בשלישית את ערך המשתנה i.

בתוכנית 5.1 הופיעו שורות הגדרת המשתנים בראש התוכנית (כלומר, מייד לאחר הסוגריים המסולסלים הפותחים של הפונקציה main). הדבר לא נעשה במקרה: כאשר ברצוננו להשתמש במשתנים, חובה להגדירם בראשית גוף הפונקציה, כלומר, לפני כל הוראה אחרת שבפונקציה. מסיבה זו, קטע התוכנית הבא אינו חוקי:

```
int a;
a = 4;      /* from this line, variables can't be defined */
int n;      /* wrong! can't appear here */
```

על-פי כלל זה, פונקציית main טיפוסית תיראה כך:

```
void main()
{
    הגדרת משתנים

    משפטים נוספים
}
```

כאשר מגדירים מספר משתנים מאותו סוג, נהוג להגדירם באותה שורת הגדרה, תוך הפרדתם בפסיקים. למשל, שתי שורות ההגדרה הבאות:

```
int number1;
int number2;
```

שקולות לשורת ההגדרה הבאה:

```
int number1, number2;
```

פלט ערכי משתנים

לאחר שראינו כיצד להגדיר משתנים ולהציב בהם ערכים, ברצוננו להציג את ערכיהם על המסך. לשם כך נשתמש בפונקציית הפלט שהכרנו בפרק הקודם, printf, אך באופן מעט שונה.

כזכור, עד עתה קיבלה הפונקציה printf פרמטר אחד, המציין את המלל שברצוננו להציג על המסך. פרמטר זה מכונה **מחרוזת בקרה** (control string). למשל, במשפט הפלט הבא:

```
printf("Madam, I'm Adam");
```

מחרוזת הבקרה היא "Madam, I'm Adam". כדי לפלוט ערך משתנה מסוים, יש לשולחו לפונקציה כפרמטר, לאחר מחרוזת הבקרה, תוך הפרדתם בפסיק. כמו-כן יש לציין במחרוזת הבקרה היכן יש להציגו, וזאת על ידי צירוף תווים מיוחד. בחן, למשל, את משפט הפלט הבא:

```
printf("I am %d years old.", age);
```

אם ערך המשתנה age הוא 17, תפלוט הפונקציה את המשפט:

```
I am 17 years old.
```

כפי שניתן לראות, לפונקציה נשלח פרמטר נוסף, age, וערכו הוא שנפלט על המסך. אף שהסימן '%' והאות d הופיעו במחרוזת הבקרה, הם לא הוצגו על המסך, ובמקומם הוצג ערך המשתנה age. הסיבה לכך היא שצירוף התווים %d הוא צירוף תווים מיוחד. כאשר מופיע צירוף זה במחרוזת הבקרה, יודעת printf שבמקום להציגו על המסך כמו שהוא, עליה להציג מספר שלם כלשהו שנשלח אליה כפרמטר. במקרה שלעיל, זהו המשתנה age.

באופן דומה, אם תבוצע הוראת הפלט הבאה:

```
printf("She sells %d sea shells", num);
```

כאשר ערך המשתנה num הוא 7, ייפלט המלל הבא:

```
She sells 7 sea shells
```

האיור הבא מדגים את הנאמר לעיל:

***איור 5.2** – השימוש בצירוף %d*

צירוף התווים %d שייך לקבוצת **תווי המרה** (conversion characters), המציינים פנייה אל ערכי פרמטרים. תו ההמרה %d מיועד רק למספרים שלמים. כדי לפלוט מספר עשרוני, נשתמש בתו המרה אחר – %f. למשל:

```
printf("The current tax rate is %f percent.", tax_rate);
```

אם המשתנה tax_rate מכיל את הערך 17.3, יוצג המשפט הבא על המסך:

```
The current tax rate is 17.300000 percent.
```

כאשר מופיע התו %f במחרוזת הבקרה, יודעת הפונקציה כי עליה להציג, במקום בו הוא מופיע, ערך עשרוני הנשלח אליה כפרמטר. מספרים בעלי נקודה עשרונית מוצגים בדיוק רב (בדרך כלל שש ספרות עשרוניות לאחר הנקודה), ולכן נפלט המספר עם אפסים כה רבים בסופו. נראה בהמשך כיצד לשנות זאת.

עצה

הקפד להתאים את תווי ההמרה לפרמטרים הנשלחים לפונקציה: אם ברצונך להציג מספר שלם, השתמש בתו ההמרה %d; אם ברצונך להציג מספר עשרוני, השתמש בתו ההמרה %f. שילוב תווי המרה עם ערכים בלתי מתאימים יביא לפלט שגוי.

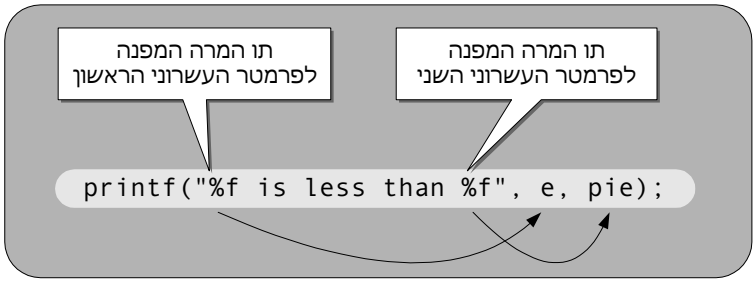
ניתן להציג מספר ערכי משתנים על המסך בהוראת פלט בודדת, על ידי כתיבת מספר תווי המרה במחרוזת הבקרה ושליחת מספר פרמטרים לפונקציה. לתו ההמרה הראשון יותאם הפרמטר השני (הפרמטר הראשון הוא מחרוזת הבקרה), לתו ההמרה השני יותאם הפרמטר השלישי, וכן הלאה. למשל:

```
printf("%f is less than %f", e, pie);
```

אם ערכי המשתנים e ו-pie הם 2.71828 ו-3.14159 בהתאמה, ייפלט המשפט הבא על המסך:

```
2.718280 is less than 3.141590
```

האיור הבא מדגים כיצד שני תווי ההמרה פונים אל ערכי שני הפרמטרים:



***איור 5.3** – השימוש בצירוף %f*

זהירות!

המהדר אינו סופר את מספר הפרמטרים המועברים אל printf, מלבד מחרוזת הבקרה. לכן, אם מספר הפרמטרים אינו תואם את מספר תווי ההמרה שבמחרוזת הבקרה, לא תוזהר על כך – והוראת הפלט תפלוט ערכים בלתי צפויים.

לסיכום, ניתן לומר כי מחרוזת הבקרה מורכבת משלושה סוגי תווים:

- **תווים רגילים** – תווים המוצגים על המסך כמו שהם. למשל, התווים המרכיבים את המחרוזת "hello, world".
- **תווי בקרה** – תווים המורים לפונקציה לבצע פעולה מיוחדת כלשהי כאשר היא נתקלת בהם. למשל, התווים '\n' ו-'\a'.
- **תווי המרה** – תווים המאפשרים להציג פרמטרים. כאשר נתקלת הפונקציה בתו המרה, היא מחפשת פרמטר נוסף שמועבר אליה ומציגה אותו על המסך, במקום שנקבע על-פי תו ההמרה המתאים.

את תו האחוז לא ניתן להציג על המסך רק על ידי כתיבתו במחרוזת הבקרה. כפי שראינו, כשמופיע התו '%' במחרוזת הבקרה, בוחנת הפונקציה את התו שבא אחריו, כדי לראות אם מתקיים צירוף מיוחד כגון %d ו-%f. כדי להציג את תו האחוז על המסך, יש לרשום אותו פעמיים ברציפות, כך: %%. למשל, כדי להציג את המלל:

```
20% of 40 is 8
```

יש לבצע הוראת פלט כזו:

```
printf("20%% of 40 is 8");
```

ולא כזו:

```
printf("20% of 40 is 8");      /* wrong */
```

בהוראת פלט זו, לאחר תו האחוז שבמחרוזת הבקרה נמצא תו רווח, והפונקציה תנסה לפרש את שני התווים כצירוף מיוחד, ולא תפלוט את תו האחוז.

דוגמה

בדוגמה זו נכתוב תוכנית המציגה ערכי משתנים על המסך.

תוכנית 5.2 *PRINTVAR.C*

```
#include <stdio.h>
void main()
{
    /* define variables to hold mass */
    float venus_mass, mars_mass, jupiter_mass, pluto_mass;

    /* assign masses to variables */
    venus_mass = 0.815;
    mars_mass = 0.107;
    jupiter_mass = 318.0;
```

```
    pluto_mass = 0.002;

    printf("Mass of planets relative to earth: \n");
    printf("Venus' mass is %f times earth's mass. \n", venus_mass);
    printf("Mars' mass is %f times earth's mass. \n", mars_mass);
    printf("Jupiter's mass is %f times earth's mass. \n",
        jupiter_mass);
    printf("Pluto's mass is %f times earth's mass. \n",
        pluto_mass);
}
```

פלט התוכנית הוא כדלהלן:

```
Mass of planets relative to earth:
Venus' mass is 0.815000 times earth's mass.
Mars' mass is 0.107000 times earth's mass.
Jupiter's mass is 318.000000 times earth's mass.
Pluto's mass is 0.002000 times earth's mass.
```

```
    float venus_mass, mars_mass, jupiter_mass, pluto_mass;
```

שורת הגדרה זו מגדירה את המשתנים בהם נשתמש בתוכנית. משתנים אלה יכילו את היחס בין מסות כוכבי לכת אחדים למסת כדור-הארץ, ולכן ערכיהם יהיו עשרוניים. מסיבה זו הגדרנו אותם כמשתנים מסוג float.

```
    venus_mass = 0.815;
    mars_mass = 0.107;
    jupiter_mass = 318.0;
    pluto_mass = 0.002;
```

משפטי השמה אלה מציבים ערכים במשתני התוכנית. ערכים אלה הם, כאמור, היחס בין מסות כוכבי לכת אחדים למסת כדור-הארץ.

```
    printf("Venus' mass is %f times earth's mass. \n",
        venus_mass);
    printf("Mars' mass is %f times earth's mass. \n", mars_mass);
    printf("Jupiter's mass is %f times earth's mass. \n",
        jupiter_mass);
    printf("Pluto's mass is %f times earth's mass. \n",
        pluto_mass);
```

משפטים אלה פולטים את ערכי משתני התוכנית. משתנים אלה הם משתנים עשרוניים, ולכן הם נפלטים באמצעות תו ההמרה %f.

מספר הפרמטרים שניתן לשלוח ולהציג באמצעות printf אינו מוגבל (בגבולות ההיגיון). לעיתים קרובות נפלוט שלושה, ארבעה ואף יותר משתנים באותה הוראה. יש רק לזכור להשתמש בתו המרה מתאים עבור כל פרמטר. למשל, בתוכנית 5.2 יכולנו להחליף את חמש הוראות הפלט בהוראת printf בודדת כזו:

```
printf("Mass of planets relative to earth: \nVenus' mass is %f
    times earth's mass. \nMars' mass is %f times earth's
    mass. \nJupiter's mass is %f times earth's mass. \nPluto's mass
    is %f times earth's mass. \n", venus_mass, mars_mass,
    jupiter_mass, pluto_mass);
```

מובן שיש להימנע מהוראות פלט מסורבלות כאלו.

במחשבה שנייה...

כמה פרמטרים נוכל לשלוח לפונקציה?

בדרך כלל לא ניתקל במגבלה על מספר הפרמטרים שניתן לשלוח לפונקציה. כל מהדר התומך בתקן ANSI חייב לאפשר לפונקציה לקבל לפחות 31 פרמטרים.

דוגמה

נציג עתה תוכנית נוספת הפולטת ערכי משתנים. הפעם נבצע מספר חישובים מתמטיים לפני פליטתם.

תוכנית 5.3 *ELEPHANT.C*

```
#include <stdio.h>
void main()
{
    /* define variables to hold ages */
    int mr_elephant, mrs_elephant, jr_elephant;
    float average;

    /* assign elephants' ages and average */
    mr_elephant = 91;
    mrs_elephant = 75;
    jr_elephant = 39;
    average = (mr_elephant + mrs_elephant + jr_elephant) / 3.0;
```

```
    printf("This is the elephants family: \n");
    printf("\tMr. Elephant is %d years old. \n", mr_elephant);
    printf("\tMrs. Elephant is %d years old. \n", mrs_elephant);
    printf("\tElephant Jr. is %d years old. \n", jr_elephant);
    printf("Its average age is %f. \n", average);
}
```

פלט התוכנית הוא כדלהלן:

```
This is the elephants family:
        Mr. Elephant is 91 years old
        Mrs. Elephant is 75 years old
        Elephant Jr. is 39 years old
Its average age is 68.333333.
```

```
    int mr_elephant, mrs_elephant, jr_elephant;
    float average;
```

בשורות אלו מוגדרים המשתנים בהם נשתמש בתוכנית. המשתנים המייצגים את גילאי הפילים הוגדרו כשלמים משום שאין צורך לדייק בהם עד כדי ספרות עשרוניות. לעומתם, המשתנה המייצג את ממוצע גיל הפילים הוגדר כמשתנה עשרוני, כדי שנדע את הממוצע המדויק.

```
    mr_elephant = 91;
    mrs_elephant = 75;
    jr_elephant = 39;
```

בשורות אלו מוצבים ערכים במשתנים המייצגים את גילאי הפילים.

```
    average = (mr_elephant + mrs_elephant + jr_elephant) / 3.0;
```

בשורה זו מחושב ממוצע גילאי הפילים. הביטוי שערכו מוצב במשתנה average מכיל ערך עשרוני (3.0), ולכן מחושב ממוצע הגילאים בדיוק עשרוני.

```
    printf("\tMr. Elephant is %d years old. \n", mr_elephant);
    printf("\tMrs. Elephant is %d years old. \n", mrs_elephant);
    printf("\tElephant Jr. is %d years old. \n", mr_elephant);
```

הוראות הפלט הללו מציגות את גילאי הפילים על המסך. גילאי הפילים מיוצגים על ידי משתנים שלמים, ולכן השתמשנו בתו ההמרה d%.

```
    printf("Its average age is %f.", average);
```

הוראת פלט זו מציגה על המסך את הערך של המשתנה average, המייצג את הגיל הממוצע של הפילים. משתנה זה הוא משתנה עשרוני, ולכן השתמשנו בתו ההמרה f%.

פרט למחרוזת הבקרה, כל הפרמטרים ששלחנו לפונקציה printf היו משתנים, אך ניתן לשלוח כפרמטרים גם קבועים מסוגים שונים. במקרה זה, יוצגו ערכי הקבועים על המסך. למשל, הוראת הפלט הבאה:

```
printf("%d + %d is %d", 4, 5, 9);
```

שקולה להוראה הבאה:

```
printf("4 + 5 is 9");
```

מכיון שכל ביטוי מחזיר ערך קבוע, ניתן לשלוח כפרמטרים אל printf גם ביטויים. לכן, הוראת הפלט הקודמת ניתנת לכתיבה גם כך:

```
printf("%d + %d is %d", 4, 5, 4 + 5);
```

כאמור, כאשר מחושב ערך הפרמטר השלישי, מוחזר הערך 9, ולכן הוראה זו זהה להוראה הקודמת.

קליטת ערכים מהמשתמש

בסעיף זה נלמד כיצד לקלוט ערכים מהמשתמש ולהציבם במשתנים. לשם כך נשתמש בפונקציית הקלט העיקרית בשפת C, הנקראת scanf. אף שתפקידה הפוך לזה של printf, המבנה התחבירי שלהן דומה.

המשתנים בהם ברצוננו להציב את הערכים שיזין המשתמש יישלחו לפונקציה כפרמטרים. למשל, אם ברצוננו לקלוט מספר שלם מהמשתמש ולהציבו במשתנה num, כפי שעושה הוראת הקלט הבאה:

נבצע הוראת scanf כזו:

```
scanf("%d", &num);
```

הפרמטר הראשון שנשלח אל scanf הוא פרמטר מלל, הדומה לפרמטר הראשון שנשלח אל printf, ואף הוא מכונה **מחרוזת בקרה**. מלל זה מורכב בעיקר מתווי המרה, וכל תו המרה מתאים לפרמטר נוסף שנשלח אל הפונקציה. כאשר נתקלת הפונקציה בתו המרה כזה, היא תמתין עד שהמשתמש יזין ערך כלשהו, ותציב ערך זה בפרמטר המתאים. את הערך יזין המשתמש תוך הקשה על המקלדת ולחיצה על מקש Enter.

שים לב!

מכיון שהשתמשנו בתו ההמרה %d, יש להזין ערך שלם. אם יוזן ערך עשרוני, הוא לא יוצב במשתנה num כמצופה.

קיימים שני הבדלים חשובים בין תחבירי הפונקציות printf ו-scanf:

- פרט לתווי המרה, כל תו המופיע במחרוזת הבקרה של printf יוצג על המסך באופן זה או אחר. מכיון שהפונקציה scanf אינה פונקציית פלט, תווי מחרוזת הבקרה שלה שאינם תווי המרה לא יוצגו על המסך. לכן, הקריאה הבאה ל-scanf **לא** תפלוט את המשפט "Enter a value:" על המסך:

```
scanf("Enter a value: %d", &num);      /* wrong */
```

- בשונה מ-printf, scanf מקבלת כפרמטרים את כתובות המשתנים ולא את ערכיהם. הדבר הכרחי כדי שהפונקציה תבצע את מלאכתה כיאות, והוא נעשה על ידי הקדמת שמות המשתנים הנשלחים כפרמטרים בתו '&'. תו זה הוא למעשה אופרטור, המכונה **אופרטור כתובת** (address operator). נוסיפו לפני שמות משתנים כדי לציין שברצוננו לפנות אל כתובותיהם בזיכרון ולא אל ערכיהם.

זהירות!

זכור להקדים כל משתנה הנשלח אל scanf בתו '&'. קל לשכוח להוסיפו – וזו טעות מסוכנת: המהדר לא יתריע מפניה, אך התוכנית לא תעבוד כמצופה. אם, למשל, נכתוב הוראת scanf כזו:

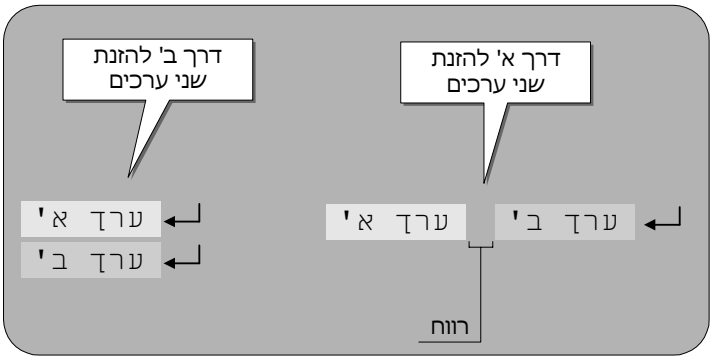
```
scanf("%d", num);     /* wrong, '&' is missing */
```

המהדר לא יתריע על כך, אך הערך שיזין המשתמש לא יוצב במשתנה num.

ניתן לקלוט מספר ערכים מהמשתמש בקריאה בודדת ל-scanf, כך:

```
scanf("%d %d", &num1, &num2);
```

במקרה זה, יכול המשתמש להפריד בין הערכים המוזנים באמצעות מקש Enter או מקש רווח, אך כדי לסיים את הקלט עליו להשתמש במקש Enter. הדבר מוצג באיור הבא:



***איור 5.4** – שתי דרכים זהות להזנת שני ערכים*

כל תוכנית בה נשתמש בפונקציה scanf תתחיל בשורה הבאה:

```
#include <stdio.h>
```

שורה זו, הזהה לשורה שהוספנו לתוכנית כאשר קראנו לפונקציה printf, נדרשת כדי ש-scanf תפעל כהלכה.

דוגמה

נדגים את השימוש ב-scanf על ידי תוכנית קצרה, הקולטת מהמשתמש שני ערכים ופולטת את הממוצע שלהם. לראשונה נכתוב קוד בשפת C שעושה פעולה שימושית כלשהי!

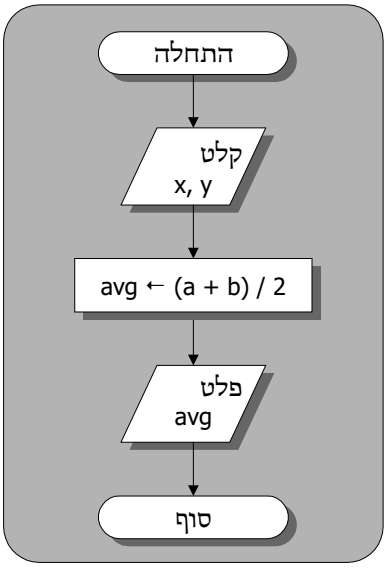
תוכנית 5.4 *SCANF.C*

```
#include <stdio.h>
void main()
{
    int x, y;
    float avg;

    /* print message */
    printf("Please enter two numbers: ");

    /* read two values */
    scanf("%d %d", &x, &y);

    /* calculate and print average */
    avg = (x + y) / 2.0;
    printf("Average of %d and %d is %f. \n", x, y, avg);
}
```

פלט התוכנית הוא כדלהלן:

```
Please enter two numbers: 98 84
Average of 98 and 84 is 91.000000.
```

תרשים הזרימה הבא מציג את פעולתה של תוכנית זו:



***איור 5.5** – תרשים זרימה לפליטת ממוצע של שני מספרים*

```
int x, y;
float avg;
```

בשורות אלו מוגדרים המשתנים בהם נשתמש בתוכניתנו. ברצוננו לקלוט מהמשתמש שני ערכים שלמים, ולכן המשתנים x ו-y, בהם יוצבו הערכים, הוגדרו כמשתנים שלמים. בחישוב הממוצע הנעשה על ידי המשתנה avg ברצוננו לדייק יותר, ולכן הגדרנו את המשתנה avg כמשתנה עשרוני. ביכולתנו לשנות את סוגי המשתנים כאוות נפשנו; עלינו רק להיות מודעים לשינויים שיתרחשו עקב כך (בעיקר הבדלים בדיוק הערכים).

```
printf("Please enter two numbers: ");
```

הוראת פלט זו פולטת הודעה למשתמש כי עליו להזין שני ערכים. השתדל שתוכניותיך יהיו ידידותיות למשתמש במידה סבירה.

```
scanf("%d %d", &x, &y);
```

הוראה זו גורמת להשהיית התוכנית עד להזנת שני ערכים, המוצבים במשתנים x ו-y. שים לב לתו '&' המופיע לפני שמות המשתנים הנשלחים אל scanf כפרמטרים.

```
avg = (x + y) / 2.0;
printf("Average of %d and %d is %f. \n", x, y, avg);
```

בשורות אלו מוצב ממוצע הערכים שהוזנו במשתנה avg, וערך משתנה זה נפלט.

ניתן באמצעות scanf לקלוט ערכים גם למשתנים עשרוניים. לשם כך לא ניעזר בתו ההמרה d%, אלא בתו ההמרה f%. למשל, כדי לקלוט ערך לתוך משתנה עשרוני בשם value, נבצע הוראה כזו:

```
scanf("%f", &value);
```

אם תבוצע הוראה זו ונזין ערך שלם, יוצב במשתנה value הערך העשרוני המתאים לו. למשל, אם נזין את המספר 2, יוצב הערך 2.0 במשתנה value.

זהירות!

כאשר קוראים לפונקציות printf ו-scanf חשוב להקפיד על התאמה בין תו ההמרה לפרמטר המתאים לו. המהדר לא בודק את ההתאמה בין השניים, ואם הם אינם מתאימים, יתקבל פלט לא רצוי.

דוגמה

בדוגמה זו נכתוב תוכנית הקולטת שני מספרים עשרוניים ופולטת את החזקה שלהם, כאשר המספר הראשון הוא בסיס החזקה והמספר השני הוא מעריך החזקה. למשל, אם יוזנו המספרים 2.5 ו-2, תחושב החזקה 2.5^2, וייפלט המספר 6.25.

כדי לחשב חזקה של מספרים עשרוניים נשתמש בפונקציית הספריה pow. פונקציה זו מקבלת שני פרמטרים, כאשר הראשון הוא בסיס החזקה והשני הוא מעריך החזקה, והיא מחזירה את החזקה שלהם. אם, למשל, נקרא לה כך:

```
f = pow(2.5, 2);
```

הפונקציה תחזיר את ערך הביטוי המתמטי 2.5^2, שהוא 6.25, וערך זה יוצב במשתנה f כאילו ביצענו את משפט ההשמה הבא:

```
f = 6.25;
```

שים לב כי הפרמטר הראשון שנשלח לפונקציה pow הוא עשרוני, בעוד שהפרמטר השני הוא שלם. הדבר חוקי – הערך השלם פשוט הופך לערך עשרוני מתאים (במקרה זה, הערך השלם 2 נהיה לערך העשרוני 2.0).

כאשר נשתמש בפונקציה pow נוסיף לראש התוכנית את השורה הבאה:

```
#include <math.h>
```

שורה זו גורמת למהדר להתייחס לנתונים הנמצאים בקובץ math.h. קובץ זה מסופק עם המהדר והוא הכרחי לביצועה התקין של הפונקציה pow.

איך לא...

ההוראה #include <math.h> אינה באה במקום ההוראה #include <stdio.h> ; על שתיהן להופיע בראש קובץ המקור. אין חשיבות לסדר בו הן מופיעות, אך אין לכתוב אותן בשורה אחת. כלומר, השורה הבאה אינה נכונה:

```
#include <stdio.h>      #include <conio.h>
```

תוכנית 5.5 *SCANF2.C*

```
#include <stdio.h>
#include <math.h>
void main()
{
    float base, exp, result;

    printf("Please enter two numbers: ");
    scanf("%f %f", &base, &exp);

    result = pow(base, exp);

    printf("Power is: %f \n", result);
}
```

פלט לדוגמה המתקבל מהרצת התוכנית מוצג להלן:

```
Please enter two numbers: 2.5 2
Power is: 6.250000
```

```
#include <math.h>
```

כאמור, הוראה זו נחוצה בכל תוכנית הקוראת לפונקציה pow. שים לב כי היא מופיעה בנוסף להוראה #include <stdio.h> ולא במקומה. על שתי הוראות ה-#include להופיע בשתי שורות נפרדות.

```
float base, exp, result;
```

בשורה זו מוגדרים שלושה משתנים עשרוניים. במשתנים base ו-exp יוצבו המספרים שיזין המשתמש, כאשר base מייצג את בסיס החזקה ו-exp את מעריך החזקה. במשתנה result תוצב החזקה של שני המספרים, כלומר, $base^{exp}$.

```
scanf("%f %f", &base, &exp);
```

קריאה זו לפונקציה scanf קולטת מהמשתמש שני מספרים ומציבה אותם במשתנים base ו-exp. שים לב כי השתמשנו בתו ההמרה %f (ולא ב-%d) וכי התו '&' מקדים את שמות המשתנים הנשלחים אל scanf כפרמטרים.

```
result = pow(base, exp);
```

משפט השמה זה מציב במשתנה result את הערך המוחזר מהפונקציה pow. פונקציה זו מחזירה את החזקה של הפרמטרים הנשלחים אליה, ולכן מוצבת במשתנה result החזקה $base^{exp}$.

המשתנה התווי

בסעיף זה נכיר סוג משתנה שלישי: **משתנה תווי** (character variable). המשתנה התווי, המכונה char, מיועד בעיקר לטיפול בתווים (כגון 'n\' ו-'+').

שים לב!

כל תו נתחם בגרשיים יחידים (') – בניגוד למחרוזת, הנתחמת בגרשיים רגילים (").

כמו כל משתנה, חובה להגדיר את המשתנה התווי לפני השימוש בו. זאת נעשה בשורת הגדרה הדומה לשורות ההגדרה שהכרנו עד כה. ראשית יופיע סוג המשתנה (char) ואחריו שם המשתנה. את שורת ההגדרה נסיים בנקודה-פסיק. למשל:

```
char ch;      /* define a character variable */
```

הצבת ערכים במשתנים תוויים ופנייה אל ערכיהם דומה להצבה ולפנייה אל ערכי משתנים שלמים או עשרוניים. למשל, כדי להציב את התו 'a' במשתנה התווי ch, נבצע משפט השמה כזה:

```
ch = 'a';
```

במשתנה תווי ניתן להציב את כל התווים הנתמכים על ידי מערכת ההפעלה. תווים אלה נכללים בטבלה הנקראת **טבלת ASCII** (American Standard Code for Information Interchange) ועמם נמנים אותיות, ספרות, תווים גרפיים, תווי בקרה ותווים מיוחדים נוספים.

טבלת ASCII מכילה 256 תווים, ולכל תו משויך ערך מספרי אחר. חלק מהתווים מופיעים להלן, בצירוף ערכיהם המספריים:

```
   32   !  33   "  34   #  35   $  36   %  37   &  38
'  39   (  40   )  41   *  42   +  43   ,  44   -  45
.  46   /  47   0  48   1  49   2  50   3  51   4  52
5  53   6  54   7  55   8  56   9  57   :  58   ;  59
<  60   =  61   >  62   ?  63   @  64   A  65   B  66
C  67   D  68   E  69   F  70   G  71   H  72   I  73
J  74   K  75   L  76   M  77   N  78   O  79   P  80
Q  81   R  82   S  83   T  84   U  85   V  86   W  87
X  88   Y  89   Z  90   [  91   \  92   ]  93   ^  94
_  95   `  96   a  97   b  98   c  99   d 100   e 101
f 102   g 103   h 104   i 105   j 106   k 107   l 108
m 109   n 110   o 111   p 112   q 113   r 114   s 115
t 116   u 117   v 118   w 119   x 120   y 121   z 122
{ 123   | 124   } 125   ~ 126
```

במחשבה שנייה...

מדוע לא הוצגו כל התווים בטבלת ASCII?

32 התווים הראשונים בטבלת ASCII, שערכיהם נעים מ-0 ל-31, הם תווים מיוחדים אשר לא ניתנים להצגה בטבלה כזו. למשל, התו הגורם לצלצול בפעמון המחשב ('\a') ערכו 7 ותו הירידה לשורה חדשה ('\n') ערכו 10. ב-129 התווים האחרונים, שערכיהם נעים מ-127 ל-255, לא נשתמש בדרך כלל ולכן בחרנו שלא להציגם.

התווים שבטבלה מסודרים על-פי הערך המספרי השונה של כל תו. ערך זה חשוב ביותר, ובאמצעותו מובדל כל תו ממשנהו. למעשה, בכל פעם שמופיע בשפת C תו בין גרשיים יחידים, הוא מומר לערך המספרי המתאים לו. למשל, מכיון שהערך המספרי של התו 'v' הוא 118, כאשר מבוצע משפט השמה כזה:

```
ch = 'v';
```

משפט ההשמה שמבצע המהדר באמת ובתמים הוא:

```
ch = 118;
```

מכיון שכך, ערך משתנה תווי לעולם לא יהיה **סימן** של תו, אלא **ערך מספרי** מתאים לו על פי טבלת ASCII. טווח הערכים שניתן להציב במשתנה תווי הוא, אם כן, מ-0 עד 255 – בהתאם לערכים המספריים הקיימים בטבלת ASCII.

ניתן לייצג תו הן על ידי סימנו והן על ידי ערכו המספרי. מכיון שכל קבוע תווי מומר תמיד לערך מספרי, שתי הדרכים זהות. למשל, שני משפטי ההשמה הבאים מבצעים פעולה זהה:

```
ch = 'a';          ch = 97;
```

עצה

בדרך כלל עדיף לציין את סימן התו ולא את ערכו המספרי, כדי להקל על הבנת התוכנית. שהרי, לקורא התוכנית ברור יותר התו 'q' מאשר המספר 113.

תוך זכירת הכלל שמשתנה תווי מכיל את הערך המספרי של התו שהוצב בו ולא את סימנו, ניתן לנחש שקטע התוכנית הבא:

```
char letter;
letter = 'd';      /* same as:  letter = 100; */
printf("letter = %d", letter);
```

יביא לפליטת המלל הבא:

```
letter = 100
```

תו ההמרה %d פולט את הערך המספרי של המשתנה letter, כפי שעשינו פעמים רבות עם ערכים שלמים מסוג int.

פלט תווים

אם ברצוננו לפלוט את הסימן המתאים לערך מספרי כלשהו, נשתמש בפונקציה printf עם תו המרה חדש: %c. בחן, למשל, את הוראת הפלט הבאה:

```
printf("letter = %c", letter);
```

תו ההמרה %c מורה ל-printf לפלוט את הסימן המתאים לערך המספרי שבמשתנה letter. אם המשתנה letter מכיל את הערך 100 (ערך ASCII של האות 'd'), ייפלט המלל הבא:

```
letter = d
```

שים לב!

אם החלטת להשתמש בסימנו של התו, זכור לתחום אותו בגרשיים יחידים. למשל, ניסיון להציב את האות a במשתנה תווי על ידי משפט ההשמה ch = a; מורה למהדר להציב במשתנה ch את ערך המשתנה a, ולא את ערך ASCII המתאים לאות 'a'.

באופן דומה, לאחר ביצוע קטע התוכנית הבא:

```
char ch;
ch = '+';
printf("The ASCII value of %c is %d. \n", ch, ch);
```

יוצג הפלט:

```
The ASCII value of + is 43
```

שני הפרמטרים הנשלחים אל printf זהים (ערכיהם הם 43), אך הם מוצגים על ידי תווי המרה שונים. תו ההמרה %c פולט את הסימן שערכו בטבלת ASCII הוא 43, ואילו תו ההמרה %d פולט את המספר השלם 43.

דוגמה

בדוגמה זו נמחיש כיצד מוחלף ערך תווי כגון 'a' בערך ASCII המתאים לו וכיצד ניתן להציג משתני int ומשתני char כערכים שלמים או כתווים המתאימים לערכים האלה.

תוכנית 5.6

CHARS.C

```
#include <stdio.h>
void main()
{
    char chr;
    int num;

    /* assign values using char symbols */
    chr = 'z';
    num = 'a';
    printf("symbols:       chr = %c, num = %c \n", chr, num);
    printf("ASCII values:  chr = %d, num = %d \n", chr, num);

    /* assign values using numeric values */
    chr = 122;      /* ASCII value of 'z' */
    num = 97;       /* ASCII value of 'a' */
    printf("symbols:       chr = %c, num = %c \n", chr, num);
    printf("ASCII values:  chr = %d, num = %d \n", chr, num);
}
```

פלט התוכנית הוא כדלהלן:

```
symbols:       chr = z, num = a
ASCII values:  chr = 122, num = 97
symbols:       chr = z, num = a
ASCII values:  chr = 122, num = 97
```

```
    char chr;
    int num;
```

בשורות אלו מוגדרים שני משתנים: משתנה תווי בשם chr ומשתנה שלם בשם num. במהלך התוכנית נציב בהם ערכים שונים ונבחן את אשר מתרחש.

```
    chr = 'z';
    num = 'a';
```

משפטי השמה אלה מציבים במשתנים chr ו-num את ערכי ASCII המתאימים לאותיות 'z' ו-'a'. לכן לאחר ההצבות ערכי המשתנים chr ו-num יהיו 122 ו-97, בהתאמה.

```
    printf("symbols:       chr = %c, num = %c \n", chr, num);
```

הוראה זו פולטת את ערכי המשתנים chr ו-num באמצעות תו ההמרה %c, ולכן מוצגים הסימנים המתאימים לערכי ה-ASCII שבהם.

```
    printf("ASCII values:  chr = %d, num = %d \n", chr, num);
```

הוראה זו פולטת את ערכי המשתנים chr ו-num באמצעות תו ההמרה %d, ולכן מוצגים ערכי ASCII שלהם.

```
    chr = 122;
    num = 97;
```

עתה מוצב בכל משתנה ערך מספרי המתאים לתו כלשהו בטבלת ASCII. מכיון שערך התו 'z' הוא 122 וערך התו 'a' הוא 97, שורות אלו זהות לשורות ההצבה הקודמות ולא משנות את ערכי המשתנים.

```
    printf("symbols:       chr = %c, num = %c \n", chr, num);
    printf("ASCII values:  chr = %d, num = %d \n", chr, num);
```

הוראות פלט אלו זהות לשתי הוראות הפלט הקודמות. מכיון שערכי המשתנים chr ו-num לא השתנו מאז פליטתם, המלל שנפלט זהה למלל שנפלט בפעם הקודמת.

קליטת תווים

קליטת ערכים תוויים מהמשתמש תבוצע אף היא באמצעות הפונקציה scanf, על ידי תו ההמרה %c. למשל, המשפט הבא:

```
scanf("%c", &ch);
```

יביא לקליטת תו מהמשתמש ולהצבת ערכו המספרי בטבלת ASCII במשתנה ch.

שים לב!

תו ההמרה %c מציין כי יש להתייחס אל הערך הנקלט מהמשתמש כאל תו ולא כאל מספר טהור. למשל, אם הוזן הערך 0, תו המרה זה קובע כי במשתנה ch יוצב ערך ASCII המתאים לתו '0', שהוא 48. אילו השתמשנו בתו ההמרה %d לקליטת הערך, היה מוצב במשתנה ch הערך 0.

מכיון שההתייחסות אל משתנה תווי היא כאל מספר, ניתן לבצע פעולות אריתמטיות שונות עם ערכים תוויים. בחן את קטע התוכנית הבא:

```
char c = 'A';
printf("The first 3 English letters are: %c %c %c", c, c + 1, c + 2);
```

במשתנה c מוצב ערך ASCII של התו 'A', שהוא 65, וזהו הפרמטר השני שנשלח אל printf. שני הפרמטרים הבאים הם c + 1 ו-c + 2, כלומר, הערכים המספריים 66 ו-67. תווי ההמרה %c מורים לפונקציה לפלוט את תווי ASCII המתאימים לערכים המספריים הללו, ולכן ייפלט המשפט הבא:

```
The first 3 English letters are: A B C
```

דוגמה

נדגים קליטת ערכים תוויים ושימוש באריתמטיקה תווית באמצעות התוכנית הבאה:

תוכנית 5.7 *ARITH.C*

```
#include <stdio.h>
void main()
{
    /* define variables to hold the entered
     * character and the ASCII code difference
     * between small and capital English letters
     */
    char ch, delta;

    /* store in delta the ASCII code difference
     * between small and capital English letters
     */
    delta = 'a' - 'A';
```

```
    printf("Enter a capital English letter: ");
    scanf("%c", &ch);

    printf("The small '%c' letter is '%c'. \n", ch, ch + delta);
}
```

פלט לדוגמה של התוכנית ייראה כך:

```
Enter a capital English letter: Q
The small 'Q' letter is 'q'.
```

```
    delta = 'a' - 'A';
```

בשורה זו מחושב ההפרש בין ערכי ASCII של האות הקטנה 'a' והאות הגדולה 'A'. הפרש זה הוא 65 - 97, כלומר, 32, וזהו הערך שמוצב במשתנה delta. מאחר שהאותיות הלועזיות הקטנות, כמו הגדולות, מופיעות ברצף בטבלת ASCII, הערך 32 הוא ההפרש בין כל אות קטנה לאות הגדולה המתאימה לה. כלומר:

$$'a' - 'A' = 'b' - 'B' = 'c' - 'C' = \ldots = 'z' - 'Z' = 32$$

במחשבה שנייה...

מדוע לא הצבנו ישירות במשתנה delta את הערך 32?

במקום משפט ההשמה שלעיל יכולנו לבצע את משפט ההשמה הבא:

```
delta = 32;      /* same as:  delta = 'a' - 'A'; */
```

בחרנו לא לעשות זאת, משום שהביטוי 'A' - 'a' מרמז על כך שברצוננו לחשב את הפרש ערכי ASCII בין האותיות הקטנות והלועזיות, ואילו המספר 32 הוא מספר סתמי שאינו מיידע את קורא התוכנית על כוונתנו.

```
    scanf("%c", &ch);
```

משפט זה מביא לקליטת ערך תווי למשתנה ch. אילו השתמשנו בתו ההמרה %d, היה על המשתמש להזין ערך מספרי ולא ערך תווי. מכיון שאנו רוצים לקלוט אות, יש להשתמש בתו ההמרה %c.

```
printf("The small '%c' letter is '%c'. \n", ch, ch + delta);
```

בהוראת פלט זו נפלטים סימני התווים המתאימים לערכי ASCII של הפרמטרים ch ו- ch + delta. המשתנה ch מכיל את ערכה המספרי של האות הגדולה שהזין המשתמש. הביטוי ch + delta הוא ערכה המספרי של האות הקטנה של האות שבחר המשתמש. הסיבה לכך היא שהמשתנה delta מכיל את ה"מרחק" בטבלת ASCII של כל אות קטנה מהאות הגדולה המתאימה לה, ואם מוסיפים ערך זה לערך האות הגדולה, מקבלים את ערך האות הקטנה שלה.

הפונקציה putch

הפונקציה putch נועדה אך ורק לפליטת תווים, והשימוש בה פשוט יותר מהשימוש בפונקציה printf. אל פונקציה זו נשלח תו בודד, ותו זה נפלט אל המסך. למשל, אם נקרא לפונקציה כך:

```
putch('a');
```

ייפלט המלל הבא:

```
a
```

מכיון שכל קבוע תווי מוחלף בערך ASCII המתאים לו, את המלל האחרון יכולנו לפלוט גם על ידי המשפט הבא:

```
putch(97);      /* 97 equals 'a' */
```

ובאופן דומה, אם נקרא לפונקציה putch באופן הבא:

```
putch(49);      /* 49 equals '1' */
```

ייפלט התו שערך ASCII שלו הוא 49, כלומר, התו '1':

```
1
```

כדי שנוכל להשתמש בפונקציה putch, עלינו להוסיף לראש התוכנית את ההוראה הבאה:

```
#include <conio.h>
```

הוראה זו גורמת למהדר להתייחס אל נתונים שונים המופיעים בקובץ conio.h. קובץ זה מסופק עם המהדר ומכיל נתונים הכרחיים לביצועה התקין של הפונקציה putch. שים לב כי ההוראה #include <conio.h> מופיעה בנוסף להוראה #include <stdio.h> ולא במקומה, וכי שתיהן נכתבות בשורות נפרדות.

הפונקציה getch

הפונקציה getch נועדה אך ורק לקליטת תווים מהמשתמש, ונהוג להשתמש בה ולא בפונקציה scanf לשם כך. פונקציה זו משהה את ביצוע התוכנית עד שנלחץ מקש כלשהו, וכאשר יקרה הדבר, יוחזר ערך ASCII של מקש זה. פונקציה זו לא מקבלת פרמטרים, אך היא מחזירה את ערך ASCII של המקש שנלחץ. כלומר, אם נקרא לפונקציה באופן הבא:

```
ch = getch();      /* ch is a char variable */
```

ויילחץ המקש '1', שערך ASCII שלו הוא 49, יוצב ערך זה במשתנה ch כאילו ביצענו את משפט ההשמה הבא:

```
ch = 49;
```

ניתן לראות שהערך המוחזר הופיע במקום בו נכתב שם הפונקציה. לכן, אם נבצע את משפט הפלט הבא:

```
printf("ASCII value of the pressed character: %d", getch());
```

והמשתמש ילחץ על המקש '1' כאשר תיקרא הפונקציה getch, יוחלף שמה בערך ASCII של המקש '1', ומשפט הפלט ייראה כך:

```
printf("ASCII value of the pressed character: %d", 49);
```

ולכן יתקבל הפלט הבא:

```
ASCII value of the pressed character: 49
```

כדי להשתמש בפונקציה getch עלינו להוסיף את ההוראה הבאה לראש התוכנית:

```
#include <conio.h>
```

כזכור, זו אותה הוראה שיש להוסיף לתוכנית כאשר משתמשים בפונקציה putch.

במחשבה שנייה...

מתי כדאי לקלוט תווים באמצעות scanf ומתי באמצעות getch?

כדי לענות על שאלה זו יש לציין את ההבדלים בין פעולות הפונקציות scanf ו-getch:

- ערך ASCII של התו הנקלט באמצעות scanf אינו מוצב במשתנה המתאים עד שנלחץ מקש Enter. הפונקציה getch, לעומת זאת, מחזירה את ערך ASCII של התו הנקלט מייד לאחר הלחיצה עליו, כך שאין צורך לסיים את הזנת הקלט באמצעות מקש Enter. מסיבה זו, אגב, לא מתבצעת ירידה לשורה חדשה לאחר ביצוע הפונקציה getch.
- סימנו של תו הנקלט באמצעות הפונקציה scanf מוצג על המסך, בעוד שהפונקציה getch אינה מציגה את המקש שנלחץ על המסך. היא רק מחזירה את ערך ASCII שלו.

דוגמה

בדוגמה זו נכתוב תוכנית המבצעת פעולה דומה לזו שביצעה תוכנית 5.7, אך עתה נשתמש בפונקציות putch ו-getch.

תוכנית 5.8 *ARITH2.C*

```
#include <stdio.h>
#include <conio.h>
void main()
{
    /* define variables to hold the entered
     * character and the ASCII code difference
     * between small and capital English letters
     */
    char ch, delta;

    /* store in delta the ASCII code difference
     * between small and capital English letters
     */
    delta = 'a' - 'A';

    printf("Enter a capital English letter: ");
    ch = getch();
    putch(ch);

    printf("\nThe small '%c' letter is '%c'. \n", ch, ch + delta);
}
```

הנה דוגמה לפלט המתקבל מהרצת תוכנית זו:

```
Enter a capital English letter: Q
The small 'Q' letter is 'q'.
```

```
#include <stdio.h>
#include <conio.h>
```

ההוראה #include <stdio.h> הכרחית משום שהפונקציה printf נקראת מתוך התוכנית. ההוראה #include <conio.h> הכרחית משום שהפונקציות putch ו-getch נקראות מתוך התוכנית.

```
ch = getch();
```

בשורה זו נקראת הפונקציה getch, הממתינה ללחיצה על מקש כלשהו ומחזירה את ערך ASCII שלו, שמוצב במשתנה ch.

שים לב!

הפונקציה getch לא תזהה לחיצה על מקשים שלא קיים עבורם ערך בטבלת ASCII. למשל, המקשים Caps Lock ו-Shift אינם קיימים בטבלת ASCII, ולכן אין ביכולתנו לקלוט לחיצה עליהם (גם לא באמצעות הפונקציה scanf).

```
putch(ch);
```

כזכור, הפונקציה getch אינה פולטת את סימן התו שנלחץ. מכיון שברצוננו להציג את התו שנלחץ על המסך, אנו קוראים לפונקציה putch ושולחים אליה את המשתנה ch, שקודם לכן הוצב בו ערך ASCII של התו שנלחץ.

פלט מורכב באמצעות printf

בפונקציה printf ניתן להשתמש באופנים מורכבים יותר מאלה שהכרנו ובכך לפלוט ערכים באופנים שונים. אחת הפעולות שלעיתים נרצה לבצע היא פליטת ערכים עשרוניים בדיוק הנקבע על ידינו. כזכור, כאשר פלטנו ערך עשרוני באמצעות printf באופן הבא:

```
printf("value: %f", 3.5);
```

הוא הוצג בדיוק של 6 ספרות אחרי הנקודה העשרונית, כך:

```
value: 3.500000
```

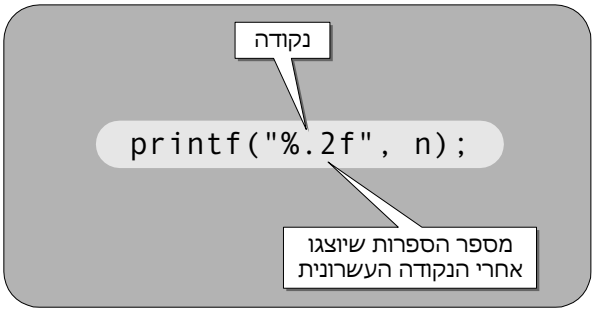
במקרה זה, חמשת האפסים המופיעים אחרי הנקודה העשרונית מיותרים ואף מקשים על קריאת הפלט. למרבה המזל, ביכולתנו לקבוע כמה ספרות עשרוניות יוצגו לאחר הנקודה העשרונית של פרמטר הנשלח ל-printf. כדי לעשות זאת נשנה את תו ההמרה המתאים לפרמטר המבוקש, על ידי הוספת תווים בין שני התווים המרכיבים את תו ההמרה %f. התו הראשון שנוסיף הוא תו הנקודה ('.'); התו השני הוא מספר הספרות שברצוננו לפלוט אחרי הנקודה העשרונית. למשל, כדי לפלוט מספר עשרוני בדיוק של שתי ספרות אחרי הנקודה העשרונית, נשתמש בתו ההמרה %.2f, כפי שמוצג להלן:

```
printf("value: %.2f", 3.5);
```

הפלט המתקבל ממשפט זה הוא כדלהלן:

```
value: 3.50
```

השימוש בתו ההמרה לפליטת פרמטר עשרוני בדיוק משתנה מוצג להלן:



***איור 5.6** – קביעת דיוק של פרמטר עשרוני*

באופן דומה, אם נבצע את משפט הפלט הבא:

```
printf("I'm approximately %.1f years old", 17.342359);
```

ייפלט המלל הבא:

```
I'm approximately 17.3 years old
```

אם הערך שנציין מימין לנקודה בתו ההמרה גדול ממספר הספרות המופיעות אחרי הנקודה העשרונית של הפרמטר המתאים לו, יוספו אפסים בהתאם. למשל, משפט הפלט הבא:

```
printf("%.5f", 1.0);
```

יביא לפליטת המלל הבא:

```
1.00000
```

במחשבה שנייה...

האם printf מתעלמת לגמרי מהספרות העשרוניות שאינן מודפסות?

למעשה, printf אינה מתעלמת מספרות אלו לגמרי. הספרה העשרונית המופיעה לאחר הספרה האחרונה שיש להציג תשמש לעיגול המספר שנפלט. למשל, משפט הפלט הבא:

```
printf("%.1f", 3.36);
```

יפלוט את המספר 3.4 ולא 3.3. הסיבה היא, כאמור, שהספרה '6', המופיעה לאחר הספרה האחרונה שיש להציג ('3'), מביאה לעיגול המספר המוצג כלפי מעלה (עיגול כזה יתבצע עבור הספרות '5' עד '9').

ניתן להשתמש בתווי ההמרה באופן מורכב יותר. נניח שברצוננו להציג כמה מספרים על המסך כך שיהיו מיושרים כלפי ימין, כפי שמוצג בפלט הבא:

```
  10.00
1234.56
   3.14
   0.98
```

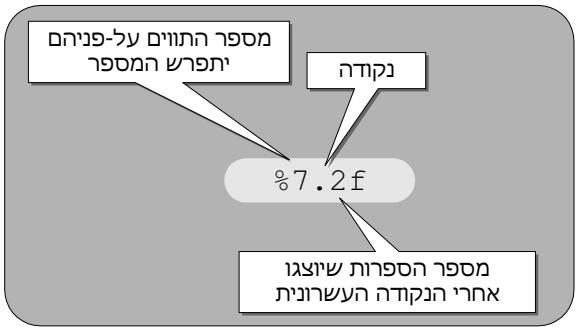
נוכל לעשות זאת על ידי כתיבת מספר משמאל לנקודה המופיעה בתו ההמרה המתאים. מספר זה יקבע את מספר התווים שיתפוס המספר הנפלט. למשל, תו ההמרה %7.2f קובע כי את המשתנה העשרוני המתאים לו יש להציג בדיוק של שתי ספרות אחרי הנקודה העשרונית ובאופן שישתרע על-פני 7 תווים. נניח שנבצע את משפט הפלט הבא:

```
printf("%7.2f", 10.0);
```

הערך שמימין לנקודה המופיעה בתו ההמרה קובע כי המספר 10.0 יוצג עם שתי ספרות אחרי הנקודה העשרונית, כלומר, כ-10.00. מספר זה, כולל הנקודה העשרונית שלו, מורכב מחמישה תווים. אך הערך שמשמאל לנקודה המופיעה בתו ההמרה קובע כי יש לפלוט את המספר על-פני 7 תווים, ולכן יוספו שני תווי רווח לפניו. כלומר, הפלט המתקבל ממשפט פלט זה הוא:

```
  10.00
```

האיור הבא מציג את מרכיבי תו ההמרה המביא לפלט שלעיל:



איור 5.7 *– קביעת מספר התווים שעל-פניהם יתפרש המספר*

אם הערך שמשמאל לנקודה המופיעה בתו ההמרה קטן ממספר התווים מהם מורכב הפרמטר המתאים, הפרמטר ייפלט במלואו ולא יקוצץ. למשל, משפט הפלט הבא:

```
printf("%4.2f", 12345.67);
```

יביא לפליטת המלל הבא:

```
12345.67
```

ניתן לקבוע גם את מספר התווים על-פניהם ישתרע פרמטר שלם. נעשה זאת על ידי כתיבת מספר התווים המבוקש בין תו '%' לתו 'd' של תו ההמרה %d. למשל, כדי לפלוט את המספר 345 כך שישתרע על-פני חמישה תווים, נבצע משפט פלט כזה:

```
printf("%5d", 345);
```

המספר 345 מורכב משלושה תווים, ולכן יתוספו לפניו שני תווי רווח, כפי שמוצג בפלט הבא:

```
  345
```

אם המספר הרשום בין תו '%' לתו 'd' של תו ההמרה %d קטן ממספר התווים מהם מורכב הפרמטר המתאים, הפרמטר ייפלט במלואו ולא יקוצץ. למשל, משפט הפלט הבא:

```
printf("%3d", 12345);
```

יביא לפליטת המלל הבא:

```
12345
```

דוגמה

התוכנית שנכתוב בדוגמה זו תקלוט מספר עשרוני מהמשתמש ותפלוט אותו על המסך באופנים שונים, תוך שימוש בתווי ההמרה שהודגמו עד כה.

תוכנית 5.9 *ADVPRINT.C*

```
#include <stdio.h>
void main()
{
    float num;

    printf("Please enter a number: ");
    scanf("%f", &num);

    printf("\n%f", num);
    printf("\n%.3f", num);
    printf("\n%6.1f", num);
    printf("\n%.0f", num);
    printf("\n%10.5f", num);
    printf("\n%12f \n", num);
}
```

פלט לדוגמה המתקבל מהרצת תוכנית זו מוצג להלן:

```
Please enter a number: 12.3456

12.345600
12.346
  12.3
12
  12.34560
   12.345600
```

```
    printf("\n%f", num);
```

תו ההמרה %f פולט את המספר שהוזן בדיוק ברירת המחדל, שהוא שש ספרות אחרי הנקודה העשרונית. במקרה זה, מוספים אפסים בהתאם מימין לספרותיו העשרוניות, והוא נראה כך: 12.345600.

```
    printf("\n%.3f", num);
```

תו ההמרה %.3f פולט את המספר שהוזן בדיוק של שלוש ספרות אחרי הנקודה העשרונית. המספר 12.3456 בדיוק של שלוש ספרות אחרי הנקודה העשרונית הוא 12.345, אך מאחר שהספרה שמימין לספרה האחרונה שמוצגת היא '6', מתבצע עיגול כלפי מעלה, ונפלט המספר 12.346.

```
    printf("\n%6.1f", num);
```

תו ההמרה %6.1f פולט את המספר שהוזן בדיוק של ספרה אחת אחרי הנקודה העשרונית, ובאופן כזה שהוא מתפרש על-פני שישה תווים. המספר 12.3456 בדיוק של ספרה אחת אחרי הנקודה העשרונית הוא 12.3, ומאחר שמספר זה מורכב מארבעה תווים, שני תווי רווח נפלטים לפניו.

```
    printf("\n%.0f", num);
```

תו ההמרה %.0f פולט את המספר שהוזן בדיוק של 0 ספרות אחרי הנקודה העשרונית, כלומר, בלי נקודה עשרונית כלל.

```
    printf("\n%10.5f", num);
```

תו ההמרה %10.5f פולט את המספר שהוזן בדיוק של חמש ספרות אחרי הנקודה העשרונית, ובאופן כזה שהוא מתפרש על-פני עשרה תווים. המספר 12.3456 בדיוק של חמש ספרות אחרי הנקודה העשרונית הוא 12.34560, ומאחר שמספר זה מורכב משמונה תווים, שני תווי רווח נפלטים לפניו.

```
    printf("\n%12f \n", num);
```

תו ההמרה %12f פולט את המספר שהוזן בדיוק של ברירת המחדל (כלומר, שש ספרות אחרי הנקודה העשרונית), ובאופן כזה שהוא מתפרש על-פני 12 תווים. המספר 12.3456 בדיוק של שש ספרות אחרי הנקודה העשרונית הוא 12.345600, ומאחר שמספר זה מורכב מתשעה תווים, שלושה תווי רווח נפלטים לפניו.

אתחול משתנים

במשתנים מכל הסוגים ניתן להציב ערכים גם בשורה בה הם מוגדרים. למשל, שני המשפטים הבאים:

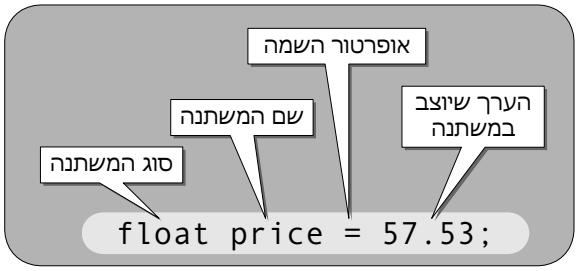
```
int seven;
seven = 7;
```

שקולים למשפט הבא:

```
int seven = 7;
```

השורה שלעיל מגדירה משתנה בשם seven ומציבה בו את הערך 7. אופן הצבה כזה מכונה **אתחול** (initialization), ולכן נאמר כי המשתנה seven מאותחל לערך 7. האתחול מבוצע על ידי כתיבת אופרטור ההשמה לאחר שמו של המשתנה וציון הערך שיוצב בו.

האיור הבא מציג כיצד לאתחל משתנה עשרוני:



איור 5.8 *– אופן אתחול משתנה*

ניתן לאתחל מספר משתנים בשורת הגדרתם, כך:

```
char letter = 'a', digit = '0', hyphen = '-';
```

או לערב משתנים מאותחלים ולא מאותחלים באותה שורת הגדרה:

```
float age = 12.0, height, weight = 41.5;
```

זהירות!

מכיון שלא אתחלנו את ערך המשתנה height, לא ניתן לצפות מה יהיה ערכו לאחר הגדרתו. לפני קריאת ערך ממשתנה וודא תמיד שהוצב בו ערך מתאים.

ניתן לאתחל משתנה לערך של משתנה אחר, המוגדר לפניו, כך:

```
int num = 10;
int num2 = num;
```

ניתן גם לאתחל משתנה לערך של משתנה אחר, המוגדר באותה שורת הגדרה לפניו, כך:

```
int faster = 1000, fastest = faster;
```

אך אין לאתחל משתנה לערך של משתנה אחר, המופיע **אחריו** בשורת ההגדרה. למשל, השורה הבאה אינה נכונה:

```
int faster = fastest, fastest = 1000;    /* wrong! */
```

אופרטורים מקוצרים

בשפת C אופרטורים רבים, מהם הכרנו רק מעטים:

```
-    +    *    /    %    =    (    )
```

בסעיף זה נכיר אופרטורים שימושיים נוספים, המאפשרים לקצר את אורך הביטויים ולהפכם לקריאים יותר.

אופרטורי השמה מקוצרים

קיימים מספר אופרטורים שנועדו לקצר משפטי השמה, והם מכונים **אופרטורי השמה מקוצרים** (compound assignment operators). עמם נמנים, בין היתר, האופרטורים הבאים:

```
-=    +=    *=    /=    %=
```

אופרטורים אלה מאפשרים לקצר את רוב משפטי ההשמה בהם חוזר אותו משתנה משמאל לסימן ההצבה ומימינו. למשל, משפט ההשמה הבא, המעלה את ערך המשתנה tmp ב-5:

```
tmp = tmp + 5;
```

ניתן לכתיבה גם כך:

```
tmp += 5;
```

משפט ההשמה השני הוא משפט השמה מקוצר, הכולל אופרטור השמה מקוצר, והוא מבצע פעולה זהה לזו של משפט ההשמה הראשון. אופרטור השמה מקוצר מורכב מאופרטור השמה ומאופרטור נוסף. הוא מבצע את פעולת האופרטור הנוסף בין המשתנה שמשמאל והביטוי שמימין, ומציב את התוצאה במשתנה שמשמאל.

להלן מספר דוגמאות נוספות לצמדי משפטי השמה שקולים:

```
x = x * 3;      ⟺    x *= 3;
avg = avg / 3;  ⟺    avg /= 3;
a = a % (b + 4); ⟺   a %= b + 4
```

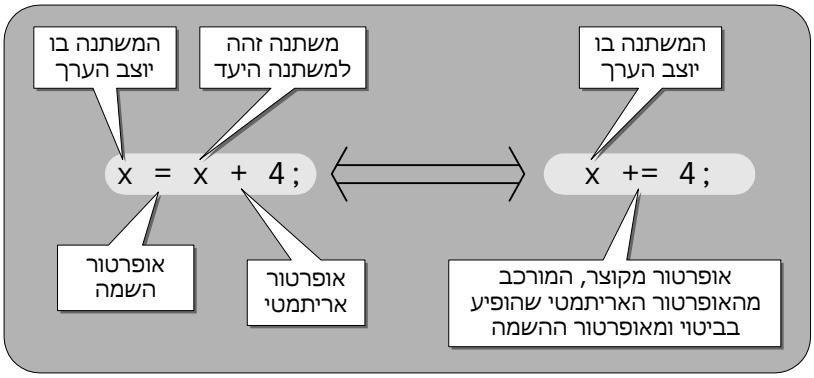
איך לא...

על שני האופרטורים המרכיבים את אופרטור ההשמה להיכתב בלי רווח ביניהם. למשל, המשפט הבא:

```
x + = 4;
```

אינו מכיל אופרטור השמה – הוא מכיל שני אופרטורים נפרדים, '+' ו-'=', והוא שגוי מבחינה תחבירית.

האיור הבא מציג כיצד להשתמש באופרטורי השמה מקוצרים:



איור 5.9 *– השימוש באופרטורי השמה מקוצרים*

שים לב כי לא ניתן לקצר כל משפט השמה בו מופיע אותו משתנה משני צידי אופרטור ההשמה. למשל, משפט ההשמה הבא:

```
x = x * 4 + 5       /* multiply x by 4, then add 5 */
```

אינו שקול למשפט ההשמה הבא:

```
x *= 4 + 5;         /* multiply x by 9 */
```

היזהר לבל תחליף בין האופרטורים המרכיבים את אופרטור ההשמה. למשל, המשפט הבא:

```
x =/ y;      /* not the same as:  x /= y (and illegal) */
```

הוא משפט שגוי. במקרה זה הטעות אינה כה חמורה, כי המהדר לא יקבלה ויפלוט הודעת שגיאה, אך בחן את המשפט הבא:

```
x =- y;      /* this is quite a dangerous statement */
```

משפט זה חוקי למהדרין, אך הוא אינו מפחית מהערך שבמשתנה x את הערך שבמשתנה y. המשפט מציב במשתנה x את הערך המספרי הנגדי לערך שבמשתנה y, והוא שקול למשפט הבא:

```
x = -y;
```

אחד מחסרונותיה של שפת C הוא מספר השילובים הרב שניתן ליצור מהאופרטורים שבה. המשפטים האחרונים הם דוגמה אחת מני רבות לבלבול הנובע מתכונה זו.

אופרטורי העלאה והפחתה עצמית

שני אופרטורים מקוצרים נוספים נועדו לקצר משפטי השמה המעלים או מורידים ערך משתנה ב-1. למשל, במקום המשפט הבא:

```
num += 1;
```

ניתן לכתוב את המשפט הבא:

```
num++;      /* add 1 to num */
```

באופן דומה, המשפט הבא:

```
num -= 1;
```

שקול למשפט הבא:

```
num--;      /* subtract 1 from num */
```

האופרטור ++ מכונה **אופרטור העלאה עצמית** (self increment operator) והאופרטור -- מכונה **אופרטור הפחתה עצמית** (self decrement operator).

איך לא...

אופרטורים אלה יופיעו רק לצד משתנה אחד, ופעם אחת בלבד. למשל, המשפטים הבאים אינם נכונים:

```
4++;
(a + b)++;
y----;
```

כמו-כן, אין להוסיף רווח בין שני הסימנים '+' ו-'-' של האופרטורים הללו. המשפט הבא, למשל, הוא שגוי ואינו מכיל אופרטור העלאה עצמית:

```
var+ +;
```

שני אופרטורים אלה יכולים להופיע גם לפני שם המשתנה, כך:

```
++num;                    --num;
```

משפטים אלה מעלים ומפחיתים את ערך המשתנה num ב-1, בדומה למשפטים בהם הופיעו האופרטורים לאחר שם המשתנה. מיקום האופרטור ביחס למשתנה משפיע על התוכנית במקרים מסוימים, אותם נכיר בהמשך.

דוגמה

בתוכנית זו נדגים את השימוש באופרטורים המקוצרים שלמדנו על ידי קליטת שני ערכים מהמשתמש וביצוע משפטי השמה שונים באמצעותם.

תוכנית 5.10 *SHORT_OP.C*

```
#include <stdio.h>
void main()
{
    int x, y;

    printf("Please enter two numbers: ");
    scanf("%d %d", &x, &y);

    printf("\nbefore\t\tx: %d, y: %d \n", x, y);
    /* add y to x: */
    x += y;
    printf("after x += y\tx: %d, y: %d \n", x, y);
    /* multiply x by y: */
    x *= y;
    printf("after x *= y\tx: %d, y: %d \n", x, y);
    /* subtract x from y: */
    y -= x;
    printf("after y -= x\tx: %d, y: %d \n", x, y);
    /* self increment/decrement: */
    x++;
    y--;
    printf("after x++;y--;\tx: %d, y: %d \n", x, y);
}
```

פלט התוכנית הוא כדלהלן:

```
Please enter two numbers: 10 4

before          x: 10, y: 4
after x += y    x: 14, y: 4
after x *= y    x: 56, y: 4
after y -= x    x: 56, y: -52
after x++;y--;  x: 57, y: -53
```

```
    scanf("%d %d", &x, &y);
```

הוראה זו קולטת מהמשתמש שני ערכים שלמים ומציבה אותם במשתנים x ו-y.

```
    printf("\nbefore\t\tx: %d, y: %d \n", x, y);
```

הוראה זו פולטת את ערכי המשתנים x ו-y. הוראות פלט דומות יופיעו פעמים נוספות בהמשך התוכנית, כדי שנוכל לעקוב אחר השינויים שחלו בערכי המשתנים.

```
    x += y;
```

משפט השמה מקוצר זה מחשב את סכום ערכי המשתנים x ו-y ומציב את התוצאה במשתנה x.

```
    x *= y;
```

משפה השמה מקוצר זה מחשב את מכפלת ערכי המשתנים x ו-y ומציב את התוצאה במשתנה x.

```
    y -= x;
```

משפט השמה מקוצר זה מחשב את הפרש ערכי המשתנים y ו-x ומציב את התוצאה במשתנה y.

```
    x++;
    y--;
```

אלה הם משפטי העלאה והפחתה עצמית, המוסיפים 1 לערך המשתנה x ומפחיתים 1 מערך המשתנה y, בהתאמה.

ביטויים מורכבים

שפת C מאפשרת חופש פעולה נרחב בכל הנוגע לכתיבת ביטויים. את כל האופרטורים שהכרנו ניתן לכתוב זה לצד זה באופנים שלא ראינו עד כה, ובכך לקצר ביטויים ומשפטי השמה. הדבר מעניק עוצמה למתכנת ומקצר מעט את אורך התוכנית, אך יחד עם זאת, מפחית מבהירותה. בסעיף זה נראה כיצד ליצור ביטויים מורכבים מסוגים שונים.

ערך מוחזר מאופרטורי העלאה והפחתה עצמית

כזכור, כאשר משפט מורכב מאופרטור העלאה או הפחתה עצמית וממשתנה, אין זה משנה אם האופרטור מופיע לפני או אחרי שם המשתנה. למשל, שני המשפטים הבאים מבצעים פעולה זהה:

```
my_age++;
++my_age;
```

בסעיף זה נראה מקרים מורכבים יותר של שימוש באופרטורים הללו, בהם סדר הופעתם לצד המשתנה חשוב ביותר.

כאשר הופיע אופרטור העלאה עצמית **לפני** שם משתנה, כך:

```
++a
```

התרחשו שתי פעולות עיקריות. הראשונה העלתה את ערך המשתנה a ב-1. השנייה, ממנה התעלמנו עד כה, החזירה את ערך המשתנה a לאחר ההעלאה ב-1. אם, למשל, המשתנה a אותחל לערך 5, לאחר ביצוע ההעלאה העצמית ערכו יהיה 6 – ויוחזר הערך 6.

במחשבה שנייה...

כיצד ייתכן שהתעלמנו לגמרי מהערך המוחזר מהאופרטור?

כל ביטוי מחזיר ערך, אך לא חובה להשתמש בו. במשפט ההצבה הבא, למשל,

```
num = 4 * 3;
```

השתמשנו בערך המוחזר מהביטוי 4 * 3 בכך שהצבנו אותו במשתנה num. אך יכולנו לוותר על פעולת ההשמה, ולחשב את מכפלת המספרים בלי להשתמש בה כלל, כך:

```
4 * 3;      /* a statement that does nothing */
```

משפט זה לא עושה דבר, אך הוא חוקי למהדרין.

בערך המוחזר מאופרטור העלאה עצמית לא השתמשנו עד כה; למעשה, התעלמנו ממנו לגמרי. אך ניתן להשתמש בו כאילו היה ערך המוחזר מביטויים כגון 3 * 2 ו-3 % 4. בחן, למשל, את שורות הקוד הבאות:

```
scanf("%d", &a);
b = ++a;
```

המשפט הראשון קולט ערך מהמשתמש ומציבו במשתנה a. המשפט השני מעלה את ערך המשתנה a ב-1, ומציב ערך זה במשתנה b. אם, למשל, נבצע שורות אלו ויוזן המספר 5, ערך המשתנה a יעלה ב-1, וערכו לאחר ההעלאה (כלומר, 6) יוצב במשתנה b. באותה מידה, יכולנו לכתוב שורות אלו כך:

```
scanf("%d", &a);
++a;
b = a;
```

כשאופרטור ההפחתה העצמית מופיע לפני שם המשתנה, גם הוא מחזיר את ערך המשתנה לאחר ההפחתה ב-1. למשל, במקום שורות הקוד הבאות:

```
scanf("%d", &a);
--a;
printf("a - 1 is %d", a);
```

ניתן לכתוב:

```
scanf("%d", &a);
printf("a - 1 is %d", --a);
```

המקרה מעט שונה כאשר אופרטורי ההעלאה וההפחתה העצמית מופיעים **לאחר** שם המשתנה. גם במקרה כזה יועלה או יופחת ערך המשתנה ב-1, אך הערך המוחזר יהיה הערך שקדם להעלאה או להפחתה. למשל, אם ערך המשתנה a הוא 4, ונבצע את המשפט הבא:

```
printf("a is %d", a++);
```

אופרטור ההעלאה העצמית יעלה את ערך המשתנה a ב-1, אך יחזיר את ערכו לפני ההעלאה, ולכן ייפלט המלל הבא:

```
a is 4
```

גם אופרטור ההפחתה העצמית, כאשר הוא מופיע לאחר שם המשתנה, מחזיר את ערכו של המשתנה לפני ההפחתה. בחן, למשל, את המשפט הבא, בהנחה שערך המשתנה a הוא 3:

```
num = a-- * 4;
```

אופרטור ההפחתה העצמית יפחית את ערך המשתנה a ל-2, אך יחזיר את הערך לפני ההפחתה. לכן, ערך המשתנה num יהיה 4 * 3, כלומר, 12.

איך לא...

המשתנה שערכו עולה או פוחת ב-1 באמצעות אופרטור ההעלאה או ההפחתה העצמית לא יכול להופיע משמאל לאופרטור ההשמה.
למשל, המשפט הבא אינו חוקי:

```
--b = 5;
```

הטבלה הבאה מציגה את סדר קדימויות האופרטורים שהכרנו (בסדר יורד):

אופרטורים	הערות מיוחדות
-- ++ ( )	אופרטורי העלאה והפחתה עצמית אלה מופיעים לאחר שם המשתנה
++ - + --	אופרטורי פלוס ומינוס אלה מופיעים לפני ערך בודד וקובעים את סימנו (חיובי או שלילי) אופרטורי העלאה והפחתה עצמית אלה מופיעים לפני שם המשתנה
% / *	
- +	אופרטורי פלוס ומינוס אלה מופיעים בין שני ערכים ומבצעים פעולת חיבור או חיסור ביניהם
-= += = %= /= *=	

דוגמה

בדוגמה זו נראה את השימוש באופרטורי ההעלאה וההפחתה העצמית בשילוב עם ביטויים אחרים.

תוכנית 5.11

SELF_OP.C

```
#include <stdio.h>
void main()
{
    int a = 2, b, c;

    b = a--;
    c = ++b;
```

```
    printf("%d \n", b++ * 3);
    printf("%d \n", --c * a++);
    printf("%d \n", ++a + -c * --b);
}
```

פלט התוכנית הוא כדלהלן:

```
9
2
-3
```

```
    int a = 2, b, c;
```

בשורה זו מוגדרים שלושה משתנים שלמים. המשתנה a מאותחל לערך 2.

```
    b = a--;
```

אופרטור ההפחתה העצמית מפחית 1 מערך המשתנה a. מכיון שהאופרטור מופיע לאחר שם המשתנה, הערך המוחזר ממנו הוא ערך המשתנה a לפני ההפחתה ב-1, וערך זה מוצב במשתנה b.

לפני ביצוע המשפט, ערך המשתנה a הוא 2. לכן, אחרי ביצוע המשפט, ערכו יהיה 1, וערך המשתנה b יהיה 2.

```
    c = ++b;
```

אופרטור ההעלאה העצמית מוסיף 1 לערך המשתנה b. מכיון שהאופרטור מופיע לפני שם המשתנה, הערך המוחזר ממנו הוא ערך המשתנה b לאחר ההעלאה ב-1, וערך זה מוצב במשתנה c.

לפני ביצוע המשפט, ערך המשתנה b הוא 2. לכן, אחרי ביצוע המשפט, ערכו יהיה 3, וערך המשתנה c יהיה 3.

```
    printf("%d \n", b++ * 3);
```

הביטוי שערכו נפלט מוסיף 1 לערך המשתנה b. מכיון שאופרטור ההעלאה העצמית מופיע לאחר שם המשתנה, הערך המוחזר ממנו הוא ערך המשתנה b לפני ההעלאה ב-1. ערך זה מוכפל ב-3, והתוצאה מוצגת על המסך.

לפני ביצוע המשפט, ערך המשתנה b הוא 3. לכן, אחרי ביצוע המשפט, ערכו יהיה 4. ערך הביטוי הנפלט הוא 3 * 3, כלומר, 9.

```
    printf("%d \n", --c * a++);
```

הביטוי שערכו נפלט מפחית 1 מערך המשתנה c ומוסיף 1 לערך המשתנה a. מכיון שאופרטור ההפחתה העצמית מופיע לפני שם המשתנה c, הערך המוחזר ממנו הוא ערך

המשתנה c לאחר ההפחתה ב-1. מכיון שאופרטור ההעלאה העצמית מופיע לאחר שם המשתנה a, הערך המוחזר ממנו הוא ערך המשתנה a לפני ההעלאה ב-1. שני הערכים מוכפלים זה בזה, והתוצאה מוצגת על המסך.

לפני ביצוע המשפט, ערך המשתנה a הוא 1 וערך המשתנה c הוא 3. לכן, אחרי ביצוע המשפט, ערך המשתנה a יהיה 2 וערך המשתנה c יהיה 2. תוצאת הביטוי הנפלט היא 1 * 2, כלומר, 2.

```
printf("%d \n", ++a + -c * --b);
```

הביטוי שערכו נפלט מוסיף 1 לערך המשתנה a ומפחית 1 מערך המשתנה b. שני אופרטורי ההעלאה וההפחתה העצמית מופיעים לפני שמות המשתנים, ולכן הם מחזירים את ערכי המשתנים a ו-b לאחר ההעלאה וההפחתה ב-1, בהתאמה.

לפני ביצוע המשפט, ערך המשתנה a הוא 2, ערך המשתנה b הוא 4, וערך המשתנה c הוא 2. לכן, אחרי ביצוע המשפט, ערך המשתנה a יהיה 3 וערך המשתנה b יהיה 3. ערך הביטוי הנפלט הוא (3 * (-2)) + 3, כלומר, 3-.

אופרטור ההעלאה או ההפחתה העצמית מחזיר ערך קבוע, ולכן, בינו לבין שאר הערכים שבביטוי חייבים להפריד אופרטורים. הביטוי הבא, למשל, אינו חוקי:

```
a++ --b     /* wrong */
```

אופרטורי ההעלאה וההפחתה העצמית הם דוגמה מעניינת להבדלים בין ביטויים, הנובעים מתווי רווח. למשל, בעוד שהביטוי הבא שגוי מבחינה תחבירית:

```
a --b       /* wrong */
```

הביטוי שלהלן חוקי למהדרין:

```
a - -b      /* subtract (-b) from a */
```

עתה, בחן את הביטוי הבא:

```
a+++b
```

על ידי הוספת רווחים במקומות שונים, ניתן לקרוא ביטוי זה בשני האופנים הבאים:

```
a++  + b                    a +  ++b
```

שני הביטויים אינם זהים, מכיון שכל אחד מהם מעלה ב-1 ערך משתנה אחר, והערך המוחזר מהביטוי a++ + b קטן ב-1 מהערך המוחזר מהביטוי a + ++b.

ובכן, המהדר ייפרש ביטוי זה כ-a++ + b. הסיבה היא, שכאשר קיימת דו-משמעות בהתייחסות לאופרטורים, קורא המהדר קודם את האופרטור בעל מספר התווים הארוך ביותר; במקרה זה, את האופרטור ++.

הערך המוחזר מאופרטורי השמה

אופרטור ההשמה מחזיר את הערך שהוצב במשתנה שמשמאלו. בחן, למשל, את הביטוי הבא:

```
size = 65
```

ביטוי זה לא רק מציב את הערך 65 במשתנה size – הוא גם מחזיר ערך זה. לכן, אם ננסה להציגו על המסך, כך:

```
printf("%d", size = 65);
```

ייפלט הערך 65, כאילו כתבנו את המשפטים הבאים:

```
size = 65;
printf("%d", size);
```

גם אופרטורי השמה מקוצרים מחזירים את הערך שהוצב במשתנה שמשמאלם. למשל, הביטוי הבא:

```
dist += 60
```

מעלה את ערך המשתנה dist ב-60, ומחזיר את ערכו לאחר ההעלאה. אם, לדוגמה, ערך dist הוא 10, ונבצע את המשפט הבא:

```
printf("distance is %d", dist += 60);
```

ייפלט המלל הבא:

```
distance is 70
```

ניתן לשלב מספר אופרטורי השמה זה לצד זה, באותו ביטוי. למשל, כדי לאפס את ערכי המשתנים a, b ו-c, נוכל לבצע משפט כזה:

```
a = b = c = 0;
```

פעולות השמה מבוצעות בכיוון הפוך לכיוון הקריאה, היינו, מימין לשמאל. לכן, המשפט שלעיל שקול למשפט הבא:

```
a = (b = (c = 0));
```

ראשון מבוצע תת-הביטוי הפנימי, המציב במשתנה c את הערך 0. פעולת השמה זו מחזירה את הערך 0, המוצב במשתנה b. גם פעולת השמה זו מחזירה את הערך 0, המוצב במשתנה a. באופן דומה, ניתן לשלב מספר אופרטורי השמה מקוצרים באותו ביטוי. בחן, למשל, את הביטוי הבא:

```
mul *= add += 4
```

גם את ביטוי זה יש לקרוא מימין לשמאל, כך:

```
mul *= (add += 4)
```

כלומר, קודם מועלה ערך המשתנה add ב-4, וערכו לאחר ההעלאה מוכפל בערך המשתנה mul. למשל, אם לפני בדיקת הביטוי היו שני ערכי המשתנים 3, לאחר ביצועו הערך של add יהיה 7 והערך של mul יהיה 21.

לעיתים, תחימת פעולות השמה בסוגריים משנה את ערך הביטוי. בחן, למשל, את המשפט הבא:

```
y = (x = 3) * 2;
```

במשפט זה מבוצע קודם תת-הביטוי שבסוגריים, המציב את הערך 3 במשתנה x. הערך המוחזר מפעולת השמה זו הוא 3. מכפלת ערך זה ב-2, כלומר 6, מוצבת במשתנה y. אילו השמטנו את הסוגריים מהמשפט, כך:

```
y = x = 3 * 2;      /* same as:  y = (x = (3 * 2)) */
```

היתה מתבצעת פעולת הכפל לפני פעולת ההשמה, והערך שהוצב במשתנה x היה 6, ולא 3.

עצה

מומלץ להימנע מלבצע פעולות השמה רבות באותו ביטוי. בעשותך כך, ייתכן שהתוכנית תגדל מעט, אך היא תהיה בהירה וקריאה יותר. למשל, את המשפט שלעיל עדיף לפצל לשני המשפטים הנפרדים הבאים:

```
x = 3;
y = x * 2;
```

ניתן לשלב אופרטורי העלאה והפחתה מקוצרים עם אופרטורי השמה; למשל:

```
num1 = b *= ++a
```

פעולת אופרטור ההעלאה העצמית קודמת לפעולות השמה, ולכן, ביטוי זה שקול לביטוי הבא:

```
num1 = (b *= (++a));
```

אם לפני ביצוע המשפט ערכי המשתנים a ו-b היו 3 ו-5 בהתאמה, אזי הערך שיוצב במשתנה num1 יהיה 20.

בדוגמה זו נראה את השימוש בביטויים המורכבים ממספר אופרטורי השמה.

תוכנית 5.12

ASSIGN.C

```
#include <stdio.h>
void main()
{
    int a, b;

    a = b = 2;
    a = (b = 1) + 3;
    a = b -= 3;
    b = a *= 2 + 3;

    printf("a = %d, b = %d \n", a, b);
}
```

פלט התוכנית הוא כדלהלן:

```
a = -10, b = -10
```

```
    a = b = 2;
```

משפט השמה זה מורכב משני אופרטורי השמה, שפעולותיהם מבוצעות מימין לשמאל. לכן, הוא שקול למשפט הבא:

```
a = (b = 2);
```

כלומר, ראשית מוצב הערך 2 במשתנה b. הערך המוחזר מפעולת השמה זו הוא 2, וזהו הערך שמוצב במשתנה a.

```
    a = (b = 1) + 3;
```

ראשון מבוצע תת-הביטוי שבסוגריים, המציב את הערך 1 במשתנה b. הערך המוחזר מפעולת השמה זו הוא 1, ולכן, הערך שמוצב במשתנה a הוא 3 + 1, כלומר 4.

```
    a = b -= 3;
```

משפט השמה זה מורכב משני אופרטורי השמה, שפעולותיהם מבוצעות מימין לשמאל. לכן, הוא שקול למשפט הבא:

```
a = (b -= 3);
```

כלומר, ראשית מופחת הערך 3 מהערך שבמשתנה b. לפני ביצוע המשפט, ערך המשתנה b הוא 1, ולכן, הערך המוחזר מפעולת ההפחתה הזו הוא 3 - 1, כלומר, 2-. זהו גם הערך שמוצב במשתנה a.

```
b = a *= 2 + 3;
```

משפט השמה זה מורכב משני אופרטורי השמה, שפעולותיהם מבוצעות מימין לשמאל. מאחר שפעולת חיבור קודמת לפעולות השמה, משפט זה שקול למשפט הבא:

```
b = (a *= (2 + 3));
```

כלומר, ראשית מוכפל ערך המשתנה a ב-5. לפני ביצוע המשפט, ערך המשתנה a הוא 2-, ולכן, הערך המוחזר מפעולת כפל זו הוא 5 * 2-, כלומר, 10-. זהו גם הערך שמוצב במשתנה b.

סוגי משתנים נוספים

בפרק זה הכרנו את סוגי המשתנים char, int ו-float, ולרוב נסתפק בהם. אך שפת C כוללת סוגי משתנים נוספים, השונים מאלה שהכרנו בעיקר בטווח הערכים שניתן להציב בהם. סוגי משתנים אלה הם:

סוג המשתנה		ערך מינימלי	ערך מקסימלי
short	(שלם)	-32768	32767
long	(שלם)	-2147483648	2147483647
double	(עשרוני)	$1.7 * 10^{-308}$	$1.7 * 10^{308}$
long double	(עשרוני)	$3.4 * 10^{-4932}$	$1.1 * 10^{4932}$

סוג המשתנה short זהה בדרך כלל לסוג המשתנה int, אך בתוכניותינו נעדיף להשתמש במשתנים מסוג int כפי שעשינו עד כה.

משתני long מיועדים לאחסון ערכים שלמים כמו משתני int, אך ניתן להציב בהם טווח ערכים גדול יותר מטווח הערכים שניתן להציב במשתני int. מקובל להשתמש במשתני long רק כאשר יש לעבד ערכים שלמים גדולים במיוחד.

משתני double ו-long double מיועדים לאחסון ערכים עשרוניים כמו משתני float, אך ניתן להציב בהם טווח ערכים גדול יותר מטווח הערכים שניתן להציב במשתני float. כמו-כן הדיוק לאחר הנקודה העשרונית גדול יותר במשתני double ו-long double.

הגדרה ואתחול

הגדרת משתנים מהסוגים החדשים שהכרנו נעשית באותו אופן בו הגדרנו משתנים עד כה. ראשית מופיע סוג המשתנה ואחריו שמות המשתנים מאותו סוג, המופרדים זה מזה בפסיקים. למשל, שורת הגדרה המגדירה משתנים מסוג long בשמות micro_sec ו-nano_sec תיכתב כך:

```
long micro_sec, nano_sec;
```

ושורת הגדרה המגדירה משתנים מסוג long double בשמות accuracy ו-precision תיכתב כך:

```
long double accuracy, precision;
```

גם אתחול המשתנים נעשה כרגיל; למשל:

```
short age = 17, height_in_cm = 175;
```

השמה

משפטי השמה הכוללים משתנים מהסוגים החדשים נכתבים כמו משפטי ההשמה שכתבנו עד כה. למשל, כדי להציב במשתנה מסוג double בשם d את הערך 7.5, נבצע משפט השמה כזה:

```
d = 7.5;
```

לרוב משתנים מסוג float יספיקו לנו לצורך חישובינו, ולא נזדקק למשתני double ו-long double. אך משתנים מסוג long הם שימושיים במקרים בהם מטפלים בערכים שלמים הגדולים בערכם המוחלט מ-30000 (בערך). אם, למשל, i הוא משתנה מסוג int ו-l הוא משתנה מסוג long, אז המשפט הבא:

```
i = 100000;      /* legal, but too big for an int variable */
```

אינו מציב במשתנה i את הערך 100000, משום שזהו ערך גדול מדי למשתנה מסוג int. לעומת זאת, המשפט הבא:

```
l = 100000;
```

מציב במשתנה l את הערך 100000, משום שמשתנה long יכול להכיל ערך גדול זה.

אף שמשתני long יכולים להכיל טווח ערכים גדול יותר ממשתני int, הם עדיין משתנים שלמים, ולכן הם אינם יכולים להכיל ערכים עשרוניים. אם נבצע משפט השמה כזה, כאשר l הוא משתנה מסוג long:

```
l = 9.9;
```

יעוגל הערך העשרוני 9.9 כלפי מטה כדי שיתאים למשתנה השלם, ולכן יוצב במשתנה l הערך 9.

קלט ופלט

כזכור, פונקציית הקלט scanf ופונקציית הפלט printf משתמשות בתווי המרה כדי להבחין בין סוגי המשתנים הנשלחים אליהם. תווי ההמרה של המשתנים החדשים שונים במקצת מתווי ההמרה שהכרנו עד כה, והם מוצגים בטבלה הבאה:

סוג המשתנה	תו ההמרה לפונקציה scanf	תו ההמרה לפונקציה printf
short	%hd	%hd
long	%ld	%ld
double	%lf	%f
long double	%Lf	%Lf

למשל, כדי לקלוט מספר למשתנה מסוג long בשם l, נקרא ל-scanf כך:

```
scanf("%ld", &l);
```

וכדי לפלוט את ערך המשתנה l, נקרא ל-printf כך:

```
printf("%ld", l);
```

סיכום

- משתנה הוא אמצעי לאחסון נתונים.
- הוראה המציבה ערך במשתנה מכונה **משפט השמה**.
- בשפת C אופרטורים רבים, והסדר בו הם מבוצעים קבוע ומוגדר בטבלת קדימויות האופרטורים.
- המשתנה השלם (int) מיועד לאחסון ערכים שלמים. המשתנה העשרוני (float) מיועד לאחסון ערכים עשרוניים.
- לפני השימוש במשתנים יש להגדיר אותם. ההגדרה נעשית בשורה הנקראת **שורת הגדרה**.
- פליטת ערכי משתנים נעשית באמצעות הפונקציה printf, כאשר מחרוזת הבקרה שלה מכילה תווי המרה המורים לה כיצד להציג את הפרמטרים הנוספים הנשלחים אליה.
- קליטת ערכים מהמשתמש נעשית באמצעות הפונקציה scanf, שהתחביר שלה דומה לתחביר הפונקציה printf.
- המשתנה התווי (char) מיועד לאחסון ערכים תוויים.

- כל קבוע תווי מוחלף בערך ASCII המתאים לו. למשל, הקבוע 'a' הופך לערך 97.
- הפונקציה putch פולטת את התו המתאים לערך ASCII הנשלח אליה.
- הפונקציה getch מחכה ללחיצה על תו ומחזירה את ערך ASCII שלו. היא אינה מחכה לאישור הקלט על ידי מקש Enter ואינה פולטת את התו שנלחץ.
- הפונקציה printf מאפשרת לנו לקבוע את מספר הספרות העשרוניות שייפלטו עבור פרמטר עשרוני הנשלח אליה. ניתן גם לקבוע את מספר התווים עליהם יתפרש הפרמטר.
- הצבת ערכים בשורת ההגדרה של משתנים מכונה **אתחול**.
- אופרטורי השמה מקוצרים מאפשרים לקצר את רוב משפטי ההשמה בהם מופיע אותו משתנה משני צדי אופרטור ההשמה.
- אופרטור העלאה עצמית מעלה ערך של משתנה ב-1. אופרטור הפחתה עצמית מפחית ערך של משתנה ב-1.
- אופרטור השמה מחזיר את הערך שהוצב במשתנה שמשמאלו.
- הערך המוחזר מאופרטורי העלאה והפחתה עצמית הוא ערך המשתנה לפני או אחרי שינויו, בהתאם למיקום האופרטור.
- נהוג להשתמש במשתנים long ו-double כאשר יש לטפל בערכים שהמשתנים int ו-float אינם יכולים להכיל.

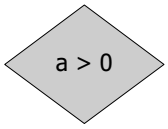
פרק 6

משפטי תנאי

הוראת התנאי שהכרנו בפרק הראשון שיפרה רבות את האלגוריתמים שכתבנו. היכולת לבצע קבוצת הוראות בהתאם לקיומו של תנאי כלשהו אפשרה לנו לפתח אלגוריתמים שימושיים רבים. בפרק זה נבצע דברים דומים בשפת C. נלמד מהם ביטויי אמת ושקר וכיצד להשתמש בהם במשפטי התנאי המגוונים הקיימים בשפת C.

ביטויי יחס

כשרצינו לבדוק האם תנאי כלשהו מתקיים, השתמשנו בהוראת תנאי אלגוריתמית. למשל, כדי לבדוק את נכונות התנאי a > b, נכתבה הוראה המיוצגת כך:



בשפת C קיימות מספר הוראות תנאי, שבוררות בין מספר משפטים ובוחרות באחד מהם בהתאם לנכונות ביטוי כלשהו, המכונה **ביטוי התנאי**. בסעיף זה נעמוד על הכללים בכתיבת ביטויי התנאי, ובהמשך נכיר את הוראות התנאי.

ביטוי התנאי, כפי שמרמז שמו, הוא ביטוי לכל דבר. הוא מורכב מערכים ומאופרטורים ומחזיר ערך בהתאם להם. הערכים המרכיבים את ביטוי התנאי הם ערכים קבועים או ערכי משתנים מסוגים שונים.

האופרטורים כוללים את האופרטורים הפשוטים שהכרנו ואת **אופרטורי היחס** (relational operators) הבאים:

אופרטור	משמעות	דוגמה
<	קטן מ...	a < b (האם ערך המשתנה a קטן משל b?)
>	גדול מ...	a > b (האם ערך המשתנה a גדול משל b?)
<=	קטן או שווה ל...	a <= b (האם ערך המשתנה a קטן או שווה לשל b?)
>=	גדול או שווה ל...	a >= b (האם ערך המשתנה a גדול או שווה לשל b?)
==	שווה ל...	a = b (האם ערך המשתנה a שווה לשל b?)
!=	שונה מ...	a <> b (האם ערך המשתנה a שונה משל b?)

איך לא...

אסור להוסיף רווחים בין תוויו של אופרטור יחס דו-תווי. למשל, אין לכתוב את האופרטור "קטן או שווה ל..." כך: = <.

אופרטורים אלה מוכרים לך מהפרק הראשון, אך חלקם נכתבים מעט אחרת בשפת C:

- אופרטורי ה"קטן או שווה ל..." וה"גדול או שווה ל..." אינם נכתבים כ-≤ וכ-≥.
- אופרטור השוויון מורכב משני סימני '=' ולא מאחד.
- אופרטור ה"שונה מ..." מורכב מהצירוף '=!' ולא מהצירוף '<>'.

זהירות!

אחת מהטעויות השכיחות ביותר בשפת C היא בלבול בין אופרטור השוויון (==) לאופרטור ההשמה (=). טעות זו מסוכנת במיוחד, משום שברוב המקרים לא יתריע המהדר מפניה.

השימוש באופרטורי היחס אינו מפתיע. למשל, כדי לבדוק אם ערך המשתנה a גדול או שווה לערך המשתנה b, נכתוב ביטוי כזה:

```
a >= b
```

כדי לבדוק אם סכום ערכי המשתנים num1 ו-num2 שונה מערך המשתנה total, נכתב ביטוי כזה:

```
num1 + num2 != total
```

אופרטורי היחס נבדקים לאחר שנבדקים האופרטורים האריתמטיים הפשוטים, ולכן ביטוי זה שקול לביטוי הבא:

```
(num1 + num2) != total
```

שים לב!

אף אחד מהביטויים שלעיל אינו מסתיים בנקודה-פסיק. על-פי הגדרתו, ביטוי לעולם לא יסתיים בתו זה.

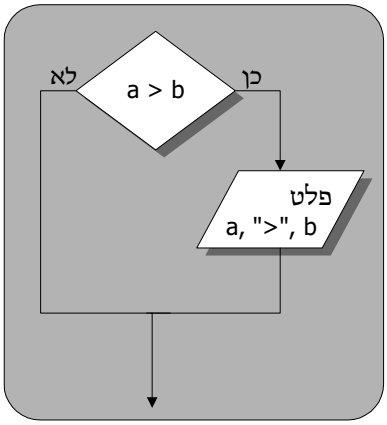
טבלת קדימויות האופרטורים שלמדנו עד כה מוצגת להלן (בסדר יורד):

אופרטורים	הערות מיוחדות
-- ++ ( )	אופרטורי העלאה והפחתה עצמית אלה מופיעים לאחר שם המשתנה
++ - + --	אופרטורי פלוס ומינוס אלה מופיעים לפני ערך בודד וקובעים את סימנו (חיובי או שלילי) אופרטורי העלאה והפחתה עצמית אלה מופיעים לפני שם המשתנה
% / *	
- +	אופרטורי פלוס ומינוס אלה מופיעים בין שני ערכים ומבצעים פעולת חיבור או חיסור ביניהם
< <= > =< >=	
!= ==	
-= += =	
%= /= *=	

קיים הבדל משמעותי בין ביטויים הכוללים אופרטורי יחס לבין ביטויים הכוללים אופרטורים אריתמטיים בלבד. לדוגמה, בעוד שהביטוי המתמטי a + 5 יכול להחזיר ערכים רבים (בהתאם לערך המשתנה a), ביטוי עם אופרטור יחס, כגון a + 5 > 0, יחזיר אחד משני ערכים בלבד: ערך **אמת** אם הביטוי מתקיים או ערך **שקר** אם הביטוי אינו מתקיים. ביטויים כאלה מכונים **ביטויי יחס**, וקל להבדילם מביטויים אריתמטיים בשל הופעת אופרטורי היחס בהם.

משפט התנאי if

בשפת C ניתן לבחור האם לבצע קטע הוראות מסוים או לא באמצעות משפט תנאי הנקרא **משפט if**. בחן, למשל, את תרשים הזרימה הבא, המציג הודעה מתאימה אם התנאי a > b מתקיים:



***איור 6.1** – הוראת תנאי בתרשים זרימה*

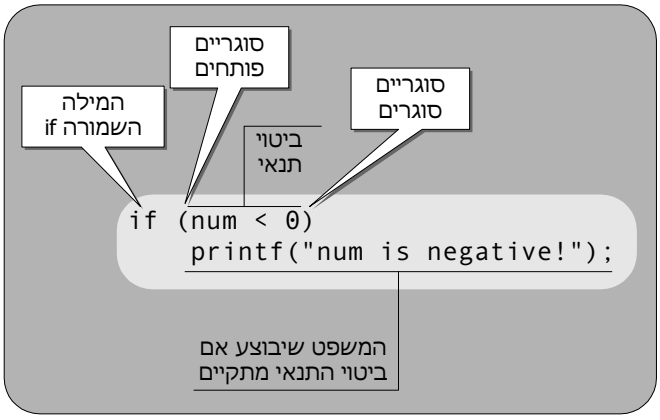
תרשים זרימה זה ניתן ליישום בשפת C כך:

```
1   if (a > b)
2       printf("a > b");
3   /* ... */
```

פעולת משפט if מתחילה בבדיקת הביטוי המופיע בסוגריים שלאחר המילה השמורה if. ביטוי זה הוא **ביטוי התנאי** שהכרנו בסעיף הקודם. אם הוא מתקיים, יבוצע המשפט השייך ל-if, המופיע בשורה (2). אם לאו, תמשיך התוכנית כרגיל, החל מהמשפט המופיע בשורה (3).

לכל משפט if ישויך משפט נוסף אחד בלבד, והוא יתבצע אם ביטוי התנאי מתקיים. משפט זה הוא המשפט המופיע לאחר הסוגריים הסוגרים בהם נכתב ביטוי התנאי. במקרה שלעיל, משפט if משתרע על פני שורות (1) ו-(2), והמשפט השייך לו הוא משפט (2). שים לב כי תו הנקודה-פסיק מופיע רק בסופו של **כל** המשפט, כלומר, לאחר שורה (2), ולא לאחר שורה (1).

התבנית הכללית לשימוש במשפט if מוצגת באיור הבא:



***איור 6.2** – מרכיבי משפט התנאי if*

המשפטים המופיעים לאחר משפט if יבוצעו בלי קשר לנכונות התנאי. למשל, בקטע התוכנית הבא:

```
1   if (total > 100)
2       printf("total is bigger than 100 \n");
3   printf("This statement will ALWAYS execute");
```

הוראת הפלט שבשורה (2) שייכת להוראת התנאי, ולכן היא תתבצע רק אם הביטוי total > 100 יתקיים. לעומתה, הוראת הפלט שבשורה (3) תתבצע תמיד, שכן היא אינה שייכת למשפט if.

זהירות!

אין להוסיף תו נקודה-פסיק לאחר הסוגריים בהם נכתב ביטוי התנאי. משפט if הבא, למשל:

```
if (a > b);      /* note the semicolon */
    printf("a > b");
```

הוא נכון מבחינה תחבירית, אך אינו מבצע את הפעולה לה ציפינו.

התוכנית הבאה שנציג תיעזר במשפטי תנאי כדי להציג את היחס בין שני ערכים שמזין המשתמש.

תוכנית 6.1 *SIMPLEIF.C*

```
#include <stdio.h>
void main()
{
    int num1, num2;
    char ch;

    printf("Please enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    /* is num1 bigger than num2?    */
    if (num1 > num2)
        ch = '>';
    /* is num1 smaller than num2?   */
    if (num1 < num2)
        ch = '<';
    /* is num1 equals num2?         */
    if (num1 == num2)
        ch = '=';

    printf("%d %c %d \n", num1, ch, num2);
}
```

מספר פלטים לדוגמה של תוכנית זו מוצגים להלן:

```
Please enter two numbers: 5 7
5 < 7
```

```
Please enter two numbers: 12 12
12 = 12
```

```
    int num1, num2;
    char ch;
```

שלושה משתנים שלמים מוגדרים בשורות הגדרה אלו. במשתנים num1 ו-num2 יוצבו שני הערכים שיזין המשתמש. המשתנה ch יכיל את אחד מהתווים '<', '>' או '=', בהתאם ליחס בין הערכים שהוזנו, והוא ייפלט על המסך בסוף התוכנית.

```
if (num1 > num2)
    ch = '>';
```

משפט תנאי זה בודק האם ערך המשתנה num1 גדול משל num2. אם הביטוי הלוגי מתקיים, יוצב במשתנה ch התו '>'. זכור שהמשפטים שלאחר משפט if (במקרה זה, המשפטים המופיעים לאחר משפט ההצבה) יתבצעו תמיד, בלי קשר לנכונות ביטוי התנאי.

```
if (num1 < num2)
    ch = '<';
```

משפט זה מציב במשתנה ch את התו '>' אם ערך המשתנה num1 קטן משל num2.

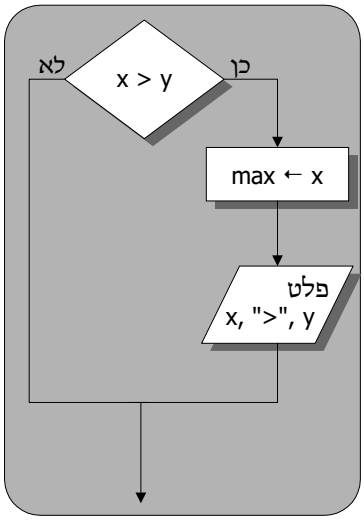
```
if (num1 == num2)
    ch = '=';
```

משפט זה מציב במשתנה ch את התו '=' אם ערכי המשתנים num1 ו-num2 זהים.

```
printf("%d %c %d \n", num1, ch, num2);
```

במשפט זה נפלטים הערכים שהזין המשתמש והתו המיוצג על ידי המשתנה ch. התו נפלט בין שני הערכים והוא מייצג את היחס ביניהם ('>' עבור "הערך משמאל קטן מהערך מימין", '<' עבור "הערך משמאל גדול מהערך מימין" ו-'=' עבור "הערך משמאל שווה לערך מימין").

כאשר פיתחנו תרשימי זרימה, ראינו כי ניתן לבצע יותר מהוראה אחת אם תנאי מסוים מתקיים. למשל, תרשים הזרימה המוצג באיור הבא:



***איור 6.3** – תרשים זרימה חלקי המבצע יותר מפעולה אחת אם ביטוי התנאי מתקיים*

מבצע **שתי** הוראות אם מתקיים ביטוי התנאי. עד כה, לא ראינו כיצד לשייך שני משפטים או יותר אל משפט if אחד. בקטע התוכנית הבא, למשל:

```
if (x > y)
    max = x;
    printf("x > y");
```

רק משפט ההשמה משויך למשפט if. הוראת הפלט אמנם מוזחת ימינה כמו ההוראה שמעליה, אך היא אינה שייכת למשפט התנאי, ולכן היא תתבצע תמיד. נוכל לשייך את שני המשפטים לשתי הוראות תנאי, כך:

```
if (x > y)
    max = x;
if (x > y)
    printf("x > y");
```

סגנון כתיבה כזה הוא חוקי, אך מסורבל ואינו יעיל (וכפי שנראה בהמשך, הוא לעיתים אינו מדויק). דרך טובה יותר היא לשייך מספר משפטים למשפט if אחד על ידי תחימתם בסוגריים מסולסלים. למשל, שילוב שני משפטי התנאי הקודמים למשפט תנאי אחד יבוצע כך:

```
if (x > y) {
    max = x;
    printf("x > y");
}
```

הסוגריים המסולסלים יוצרים משפט מיוחד בשפת C, המכונה **משפט מורכב** (compound statement). משפט מורכב מכיל בתוכו (בין הסוגריים המסולסלים) מספר משפטים; במקרה זה, משפט הצבה ומשפט פלט.

עצה

את המשפטים המופיעים לאחר הסוגריים המסולסלים הפותחים נהוג לכתוב בשורה חדשה תוך הזחה של מספר תווים ימינה. נוהג זה מאפשר לזהות בקלות אילו משפטים שייכים למשפטים אחרים.

חשוב לציין כי המשפט המורכב, יחד עם כל המשפטים שבו, נחשב למשפט בודד. לכן, מנקודת ראותו של משפט if, שייך אליו משפט אחד ויחיד – המשפט המורכב (שכולל בתוכו מספר משפטים).

המשפט המורכב יכול להכיל משפטים רבים, או להיות ריק מתוכן. למשל, משפט if הבא:

```
if (num > 0) {

}
```

בודק האם המשתנה num מכיל ערך חיובי, אך אינו מבצע דבר אם הביטוי מתקיים.

אנו נעדיף להשתמש במשפט המורכב רק כאשר יש לבצע יותר ממשפט אחד אם ביטוי התנאי מתקיים. יחד עם זאת, יש המשתמשים במשפט המורכב תמיד, גם כדי לבצע משפט בודד; למשל:

```
if (a > b) {
    printf("a is bigger than b!");
}
```

זהירות!

המשפט המורכב הוא משפט לכל דבר, אך לא נוסיף תו נקודה-פסיק בסופו (כלומר, לאחר הסוגריים המסולסלים הסוגרים). למשל, קטע הקוד הבא:

```
if (a > b) {
    printf("a is bigger than b");
};      /* note the semicolon */
```

אינו שגוי מבחינה תחבירית, אך הוא עלול לגרום לשגיאות תחביריות בעתיד.

משפט התנאי if-else

עד כה, כשרצינו לבצע הוראות אם ביטוי התנאי **אינו** מתקיים, היה עלינו לכתוב משפט תנאי נוסף שביטויו הפוך (מבחינה לוגית) לביטוי המקורי. למשל, כדי לפלוט הודעה מתאימה אם ערך המשתנה value הוא 0 או לא, נכתוב קטע תוכנית כזה:

```
if (value == 0)
    printf("value is zero");
if (value != 0)
    printf("value is nonzero");
```

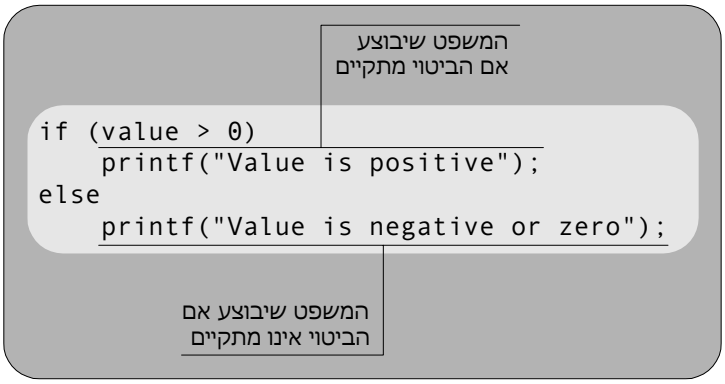
קטע תוכנית זה אינו שגוי מבחינה תחבירית, אך נעדיף לא להשתמש בו, ממספר סיבות. ראשית, שני ביטויי התנאי ייבדקו תמיד, והדבר אינו יעיל מבחינת זמן ריצת התוכנית (בתוכניות מורכבות וארוכות זמן ריצת התוכנית לא יסולא בפז); שנית, לקורא התוכנית לא ברור לגמרי כי הביטוי של משפט התנאי השני הפוך לוגית לזה של משפט התנאי הראשון; ושלישית, אם ישתנה ערך המשתנה value במהלך משפט התנאי הראשון (אם, למשל, ישויך אליו משפט הצבה כגון ;value = 4), עלול להתבצע המשפט השייך למשפט התנאי השני – וזאת בניגוד לכוונתנו המקורית.

עדיף לטפל במקרה זה על ידי משפט if-else, המוסיף למשפט if את האפשרות לביצוע משפט כלשהו אם ביטוי התנאי **אינו** מתקיים. למשל, קטע התוכנית הקודם ייכתב באמצעות משפט if-else כך:

```
if (value == 0)
    printf("value is zero");
else
    printf("value is nonzero");
```

משפט if-else מתחיל כמשפט if רגיל, אך לאחר המשפט השייך ל-if נכתבת המילה השמורה else. אחריה נכתב משפט נוסף, שיתבצע אם ביטוי התנאי אינו מתקיים. המהדר מתחיל בבדיקת ביטוי התנאי: אם הוא מתקיים, יבוצע המשפט השייך לחלק if. אם לאו, יבוצע המשפט השייך לחלק else.

באיור הבא מוצגת התבנית הכללית לשימוש במשפט if-else:



***איור 6.4** – מרכיבי המשפט if-else*

במשפט if-else יבוצע המשפט השייך לחלק if או המשפט השייך לחלק else – אך לא שניהם. למשל, בקטע התוכנית הבא:

```
1   int x = 7;
2   if (x > 0)
3       x = -1;
4   else
5       printf("x isn't positive!");
```

המשפט שבשורה (5) לא יתבצע, משום שבדיקת ביטוי התנאי וההחלטה איזה משפט לבצע (זה השייך לחלק if או זה השייך לחלק else) מתבצעות פעם אחת בלבד. לעומת זאת, בקטע התוכנית הבא:

```
1   int x = 7;
2   if (x > 0)
3       x = -1;
4   if (x <= 0)
5       printf("x isn't positive!");
```

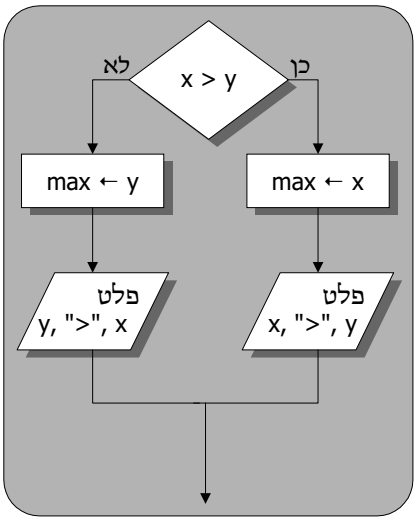
יתבצעו שני המשפטים שבשורות (3) ו-(5), משום שהם שייכים לשני משפטי if נפרדים, וביטויי התנאי שלהם נבדקים בנפרד.

כל else בתוכנית חייב להתקשר אל משפט if שנכתב לפניו. בין if ו-else השייך לו חייב להופיע משפט אחד ויחיד (בזאת נכלל גם המשפט המורכב, כפי שנראה מייד) – לא פחות ולא יותר.

למשל, קטע התוכנית הבא שגוי:

```
if (sex == 1)
    printf("Male");
a = 4;
else   /* error! this else isn't attached to any if! */
    printf("Female");
```

גם חלק if וגם חלק else יכולים להכיל לביצוע שני משפטים או יותר. נעשה זאת על ידי תחימת המשפטים שיש לבצע בסוגריים מסולסלים, כך שייחשבו למשפט מורכב בודד.
למשל, קטע תוכנית השקול לתרשים הזרימה הבא:



***איור 6.5** – תרשים זרימה חלקי בעל משפט תנאי, המכיל יותר ממשפט אחד בכל "ענף"*

ייראה כך:

```
if (x > y) {
    max = x;
    printf("x > y");
}
else {
    max = y;
    printf("y > x");
}
```

בדוגמה זו נציג תוכנית הקולטת שני מספרים מהמשתמש ופולטת את המנה. נשתמש בהוראת if-else כדי לוודא שלא מתבצעת חלוקה ב-0, העלולה להביא לקריסת התוכנית.

תוכנית 6.2 *IFELSE.C*

```
#include <stdio.h>
void main()
{
    float number, divider, result;

    printf("Please enter a number and a divider: ");
    scanf("%f %f", &number, &divider);

    /* check if not dividing by zero */
    if (divider != 0.0) {
        result = number / divider;
        printf("%f/%f = %f \n", number, divider, result);
    }
    else
        printf("Can't divide by zero! \n");
}
```

אם המחלק שהוזן אינו 0, תיפלט מנת המחולק במחלק. למשל:

```
Please enter a number and a divider: 15 3.75
15.000000/3.750000 = 4.000000
```

אך אם המחלק הוא 0, תיפלט הודעת שגיאה. למשל:

```
Please enter a number and a divider: 3 0
Can't divide by zero!
```

```
    float number, divider, result;
```

בשורה זו מוגדרים שלושה משתנים עשרוניים: number, שייצג את הערך המחולק, divider, שייצג את הערך המחלק, ו-result, בו תוצב מנת המחולק במחלק. שלושת המשתנים הוגדרו כעשרוניים לשם הדיוק.

```
    if (divider != 0.0) {
        result = number / divider;
        printf("%f/%f = %f \n", number, divider, result);
    }
```

אם המחלק שונה מ-0 ניתן לחלק בו את המחולק, והדבר נעשה במשפט ההשמה, בו מוצבת המנה במשתנה result. משפט הפלט פולט את הערכים שהזין המשתמש ואת מנתם.

```
    else
        printf("Can't divide by zero! \n");
```

משפט else זה מהווה חלופה למשפט if, ומשפט הפלט השייך לו יתבצע אם ביטוי תנאי if לא התקיים.

במחשבה שנייה...

ניתן להחליף בין שני המשפטים השייכים למשפט if-else אם ביטוי התנאי משתנה בהתאם. כלומר, משפט if-else הבא:

```
if (a % 2 == 0)
    printf("a is even");
else
    printf("a is odd");
```

שקול למשפט הבא:

```
if (a % 2 != 0)
    printf("a is odd");
else
    printf("a is even");
```

דוגמה

התוכנית הבאה שנכתוב תקלוט מהמשתמש מספר ותפלוט את השורש הריבועי על ידי קריאה לפונקציה sqrt, המחשבת את השורש הריבועי של מספר הנשלח אליה. מכיון שאסור להוציא שורש ריבועי ממספר שלילי, נשתמש במשפט if-else כדי לוודא שלא מתבצעת פעולה אסורה כזו.

לפני כתיבת התוכנית נדגים את השימוש בפונקציה sqrt. פונקציה זו מקבלת מספר עשרוני כפרמטר ומחזירה את שורשו הריבועי. למשל, הקריאה הבאה לפונקציה:

```
value = sqrt(20.25);
```

תציב במשתנה sqrt את שורשו הריבועי של 20.25, כלומר, את המספר 4.5. אם נשלח אל הפונקציה ערך שלם, הוא יומר אוטומטית לערך עשרוני. למשל, בקריאה הבאה לפונקציה:

```
value = sqrt(100);
```

מוחזר שורשו הריבועי של המספר העשרוני 100.0, שהוא, כמובן, 10.0.

זהירות!

כמו במתמטיקה, בשפת C אסור לחשב שורש ריבועי של מספר שלילי. לכן, הפונקציה sqrt תחזיר ערך מתאים רק עבור ערכים שאינם שליליים.

תוכנית 6.3 *IFELSE2.C*

```
#include <stdio.h>
#include <math.h>
void main()
{
    float number, sq_root;

    printf("Please enter a positive number: ");
    scanf("%f", &number);

    if (number < 0.0)
        printf("Can't calculate the square root of a negative
            number! \n");
    else {
        sq_root = sqrt(number);
        printf("The square root of %f is %f \n", number, sq_root);
    }
}
```

להלן מספר פלטים המתקבלים מתוכנית זו:

```
Please enter a positive number: 10
The square root of 10.000000 is 3.162278
```

```
Please enter a positive number: -9
Can't calculate the square root of a negative number!
```

```
#include <math.h>
```

שורה זו גורמת למהדר להתייחס אל נתונים המופיעים בקובץ math.h. הנתונים שבקובץ זה מספקים למהדר מידע חיוני הנוגע לפונקציות מתמטיות שונות. אחת מפונקציות מתמטיות אלו היא הפונקציה sqrt בה השתמשנו בתוכניתנו, ולכן עלינו להוסיף הוראה זו לראש התוכנית.

```
    float number, sq_root;
```

בשורת הגדרה זו מוגדרים שני משתנים עשרוניים. במשתנה number יוצב המספר שיזין המשתמש, ושורשו הריבועי של מספר זה יוצב במשתנה sq_root על ידי קריאה לפונקציה sqrt.

```
    if (number < 0.0)
        printf("Can't calculate the square root of a negative
            number! \n");
```

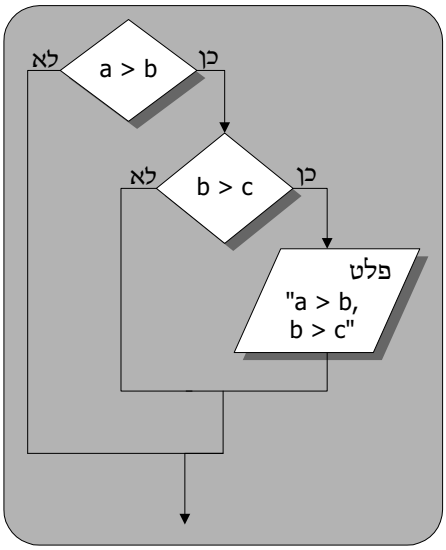
ביטוי התנאי מתקיים אם המספר האחרון שהוזן שלילי. במקרה זה, אין להוציא מהמספר שורש ריבועי, ומשפט הפלט מציג הודעה מתאימה.

```
    else {
        sq_root = sqrt(number);
        printf("The square root of %f is %f \n", number,
            sq_root);
    }
```

משפט else זה יתבצע אם ביטוי התנאי number < 0.0 לא התקיים, כלומר, אם המספר שהוזן אינו מספר שלילי. במקרה זה, ניתן לחשבו את שורשו הריבועי, והדבר נעשה על ידי הקריאה לפונקציה sqrt. משפט הפלט מציג על המסך את המספר ואת שורשו הריבועי.

קינון משפטי if

כזכור, בפרק השני קיננו הוראות תנאי כדי לוודא שהוראה כלשהי תתבצע רק אם שני תנאים שונים מתקיימים. דוגמה לכך מופיעה בתרשים הזרימה המוצג באיור 6.6, הפולט הודעה מתאימה אם ערך המשתנה a גדול מערך המשתנה b ואם ערך המשתנה b גדול מערך המשתנה c.



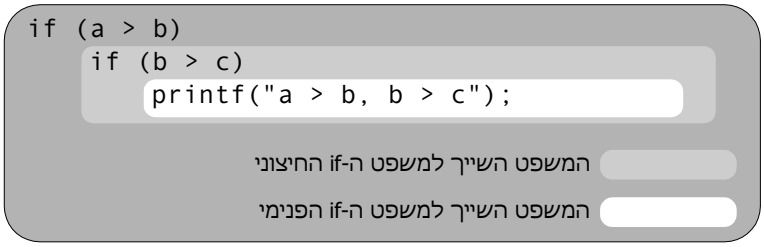
***איור 6.6** – תרשים זרימה חלקי בעל הוראות תנאי מקוננות*

גם בשפת C ניתן לקנן הוראות תנאי זו בזו. מכיון שמשפט if (יחד עם המשפט השייך לו) נחשב למשפט אחד, ניתן לשייכו אל משפט if אחר. למשל, קטע תוכנית המיישם את תרשים הזרימה המוצג באיור 6.6 ייראה כך:

```
1   if (a > b)
2       if (b > c)
3           printf("a > b, b > c");
4   /* ... */
```

קטע תוכנית זה מורכב ממשפט אחד כולל, המאחד בתוכו שני משפטים נוספים. המשפט הכולל הוא משפט if המתחיל בשורה (1). למשפט זה שייך משפט if המתחיל בשורה (2). גם למשפט if השני שייך משפט, והוא משפט הפלט המופיע בשורה (3). ניתן לומר, אם כן, שמשפט if הראשון משתרע על פני שורות (1)-(3) ושמשפט if השני משתרע על פני שורות (2)-(3).

האיור הבא מראה את שייכות המשפטים הללו זה לזה:



***איור 6.7** – קינון משפטי if*

אם ביטויו של משפט התנאי הראשון מתקיים, יתבצע המשפט השייך לו, המופיע בשורה (2). משפט זה הוא משפט if בעצמו, והוא יתבצע כרגיל: ביטויו ייבדק, ואם הוא מתקיים, יבוצע המשפט השייך לו, המופיע בשורה (3). כלומר, המשפט המופיע בשורה (3) יתבצע רק אם מתקיימים ביטויי התנאי a > b ו-b > c. במילים אחרות, אם אחד משני ביטויי התנאי אינו מתקיים, יתבצעו המשפטים שבשורה (4) והלאה.

זהירות!

שלא כמו במתמטיקה, שני הביטויים a > b ו-b > c אינם שקולים לביטוי a > b > c. לכן, אין להחליף את שני משפטי if שבאיור 6.7 במשפט התנאי הבא:

```
if (a > b > c)
    printf("a > b > c");
```

משפט זה חוקי למהדרין, אך פעולתו מפתיעה למדי. נרחיב על כך בהמשך.

הנה דוגמה נוספת, הבודקת האם ערכי שני משתנים שהזין המשתמש חיוביים:

```
scanf("%d %d", &val1, &val2);
if (val1 > 0)
    if (val2 > 0)
        printf("Both numbers are positive!");
```

אם נרצה לבצע יותר ממשפט אחד בתנאי שמתקיימים שני התנאים, ניעזר במשפט המורכב. כזכור, המשפט המורכב הוא משפט בודד המוגדר על ידי סוגריים מסולסלים ויכול להכיל מספר משפטים. למשל, כדי לבצע שני משפטי פלט אם הערכים שהזין המשתמש חיוביים, נכתוב קטע תוכנית כזה:

```
1   if (val1 > 0)
2       if (val2 > 0) {
3           printf("If you see this, ");
4           printf("both numbers are positive!");
5       }
```

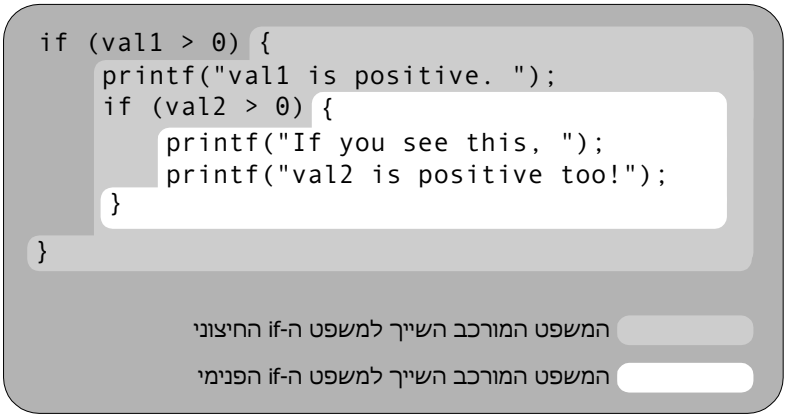
הסוגריים המסולסלים מאגדים את המשפטים שבשורות (3) ו-(4) למשפט מורכב אחד, השייך למשפט if השני. לפיכך, אם יתקיימו שני הביטויים, יתבצעו שני משפטי הפלט.

באופן דומה, אם ברצוננו לבצע מספר משפטים אם מתקיים ביטוי התנאי הראשון (ולא בהכרח ביטוי התנאי השני), עלינו לתחמם בסוגריים מסולסלים. למשל:

```
if (val1 > 0) {
    printf("val1 is positive. ");
    if (val2 > 0) {
        printf("If you see this, ");
        printf("val2 is positive too!");
    }
}
```

למשפט if הראשון שייך משפט אחד בלבד – אך זהו משפט מורכב, המכיל בתוכו שני משפטים נוספים: משפט פלט ומשפט if, שגם לו שייך משפט מורכב. משפט מורכב זה מכיל שני משפטי פלט.

האיור הבא מציג את שייכות המשפטים הללו זה לזה:



***איור 6.8** – שיוך משפט מורכב למשפט if*

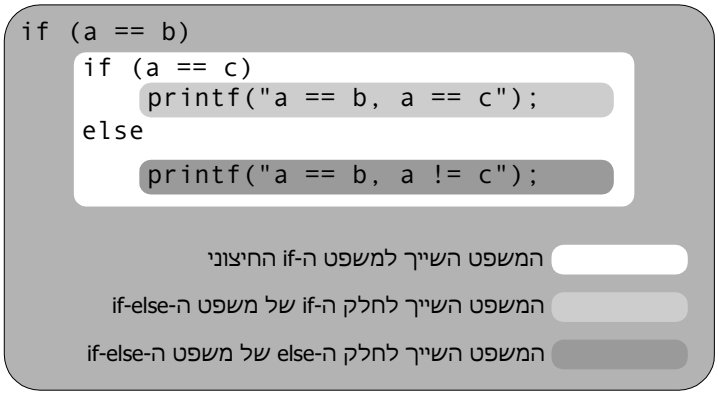
קינון משפטי if-else

כפי שניתן לקנן משפטי if, כך ניתן לקנן משפטי if-else. מכיון שמשפט if-else הוא משפט בודד, ניתן לשייכו למשפט if בלי להשתמש בסוגריים מסולסלים, כך:

```
1   if (a == b)
2       if (a == c)
3           printf("a == b, a == c");
4       else
5           printf("a == b, a != c");
```

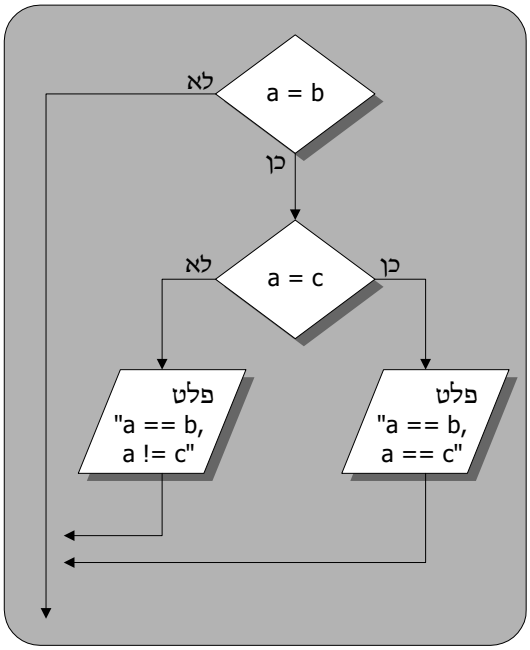
משפט if הראשון מבצע את משפט if-else השייך לו אם ערכי המשתנים a ו-b שווים. משפט if-else המשתרע על פני שורות (2)-(5) משווה את ערכי המשתנים a ו-c ופולט הודעה מתאימה.

האיור הבא מציג את שייכות המשפטים הללו זה לזה:



***איור 6.9** – קינון משפט if-else*

תרשים זרימה המייצג את המשפטים הללו מוצג באיור הבא:



***איור 6.10** – תרשים זרימה חלקי המייצג משפט if-else מקונן*

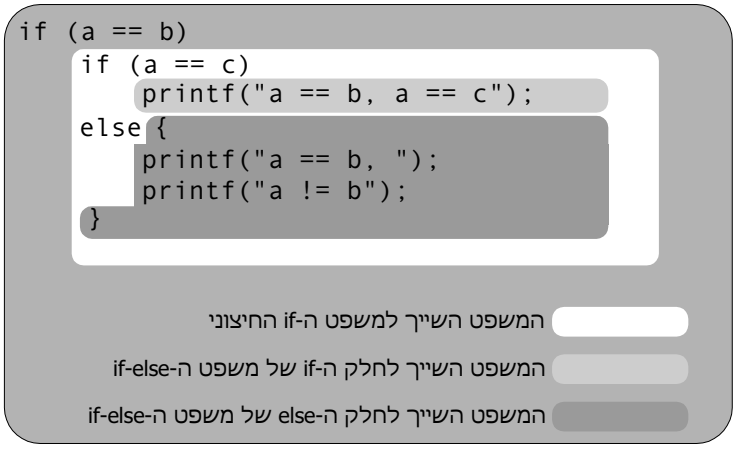
זהירות!

כפי שלא ניתן להחליף את שני הביטויים a > b ו-b > c בביטוי a > b > c, אין להחליף את שני הביטויים a == b ו-a == c בביטוי a == b == c. נסביר את הסיבה לכך בהמשך.

ניתן לשייך משפט מורכב גם אל חלקי if ו-else של משפט if-else מקונן. למשל:

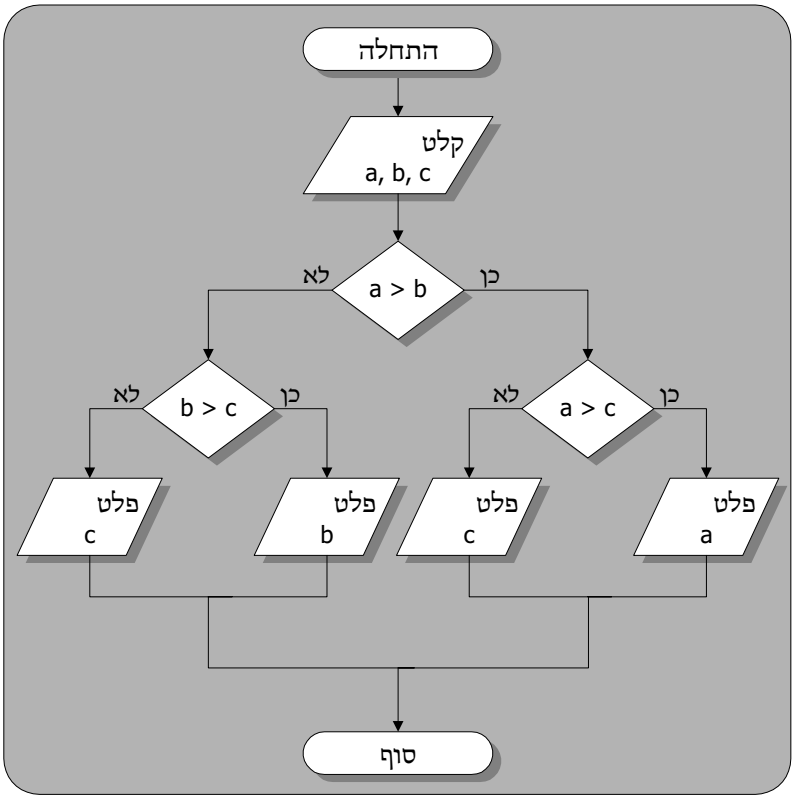
```
if (a == b)
    if (a == c)
        printf("a == b, b == c");
    else {
        printf("a == b, ");
        printf("a != c");
    }
```

שייכות המשפטים הללו זה לזה מוצגת באיור הבא:



***איור 6.11** – שיוך משפט מורכב אל חלק else של משפט if-else מקונן*

נדגים את השימוש בקינון משפטי if-else על ידי תוכנית בעלת משפטי תנאי מקוננים רבים, המיישמת את תרשים הזרימה הבא:



***איור 6.12** – תרשים זרימה לפליטת המספר הגבוה מבין שלושה מספרים*

תרשים זרימה זה הוצג לראשונה בפרק השני, והוא מיישם אלגוריתם לפליטת המספר הגדול מבין שלושה מספרים שמזין המשתמש.

תוכנית בשפת C העושה פעולה דומה תיראה כך (בתוכנית זו אנו מניחים, לשם פשטות ההסבר, כי שלושת המספרים המוזנים שונים זה מזה):

תוכנית 6.4

NEST_IF.C

```
#include <stdio.h>
void main()
{
    int a, b, c;

    printf("Please enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    printf("The biggest number is: ");
    if (a > b)
        if (a > c)
            /* a > b, a > c */
            printf("%d \n", a);
        else
            /* a > b, c > a */
            printf("%d \n", c);
    else
        if (b > c)
            /* b > a, b > c */
            printf("%d \n", b);
        else
            /* b > a, c  > b */
            printf("%d \n", c);
}
```

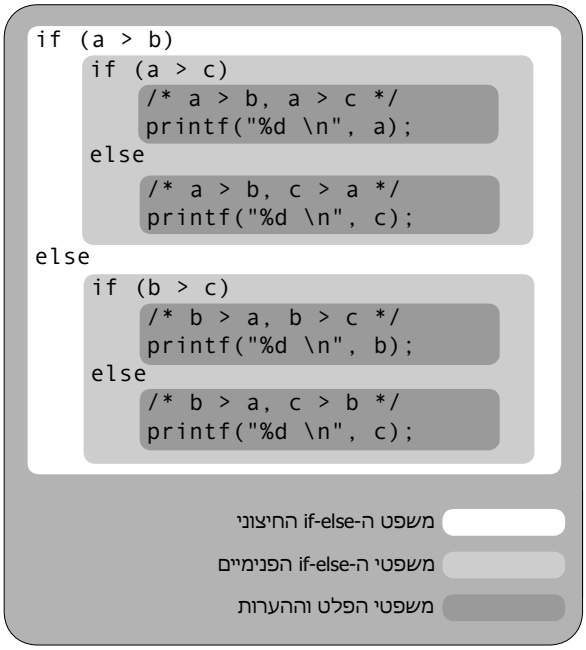
להלן מספר פלטים לדוגמה מתוכנית זו:

```
Please enter three numbers: 100 30 68
The biggest number is: 100
```

```
Please enter three numbers: 4 8 -12
The biggest number is: 8
```

כפי שמרמזות הערות התוכנית, קינון משפטי if-else מאפשר למצוא את המשתנה שערכו הוא הגבוה ביותר. אם הינך מתקשה להבין את העיקרון הכללי על-פיו עובדת התוכנית, התבונן באיור 6.12 או חזור לדוגמה המתאימה בפרק השני.

האיור הבא מציג את שייכות משפטי התנאי המקוננים שבדוגמה זה לזה:



***איור 6.13** – שייכות משפטי התנאי שבתוכנית 6.4 זה לזה*

שים לב!

הערה אינה נחשבת למשפט. לפיכך, למשפט התנאי הבא:

```
if (a > c)
    /* a > b, a > c */
    printf("%d \n", a);
```

שייך משפט אחד בלבד (משפט הפלט).

ניתן לשרשר מספר רב של משפטי if-else זה לזה. בחן, למשל, את קטע התוכנית הבא:

```
1   if (a > b)
2       printf("a is bigger than b");
3   else if (a < b)
4       printf("a is smaller than b");
5   else
6       printf("a equals b");
7   /* ... */
```

קטע תוכנית זה בודק את היחס בין שני משתנים, כפי שעשינו בתוכנית 6.1. אך כאן השתמשנו במשפטי if-else מקוננים, ולכן לאחר שהתקיים אחד מביטויי התנאי, תמשיך התוכנית מהמשפט שבשורה (7), בלי לבדוק את שאר ביטויי התנאי הקיימים עד שורה זו. כאשר הדבר אפשרי, עדיף להשתמש במשפטי if-else מאשר במשפטי if.

דוגמה

בדוגמה זו נשתמש במשפטי if-else מקוננים כדי לפלוט את מספר השורשים שיש לפולינום ממעלה שנייה, שאת מקדמיו מזין המשתמש. כזכור, אם נתון הפולינום הבא, שמקדמיו הם a, b ו-c:

$$ax^2 + bx + c = 0$$

אזי:

- אם $a = 0$ ו-$b \neq 0$, מתקבלת המשוואה $bx + c = 0$, ולה פתרון אחד בלבד.
- אם $a = 0$ ו-$b = 0$, אין פתרונות למשוואה.
- אם $a \neq 0$, יש לחשב את הדיסקרימיננטה של המשוואה, המוגדרת כך:

$$\Delta = \sqrt{b^2 - 4ac}$$

ואז:

- אם $\Delta > 0$, למשוואה שני פתרונות שונים.
- אם $\Delta = 0$, למשוואה שני פתרונות זהים.
- אם $\Delta < 0$, למשוואה אין פתרונות.

תוכנית 6.5 *QUADRAT.C*

```
#include <stdio.h>
void main()
{
    float a, b, c, delta;

    printf("Enter A, B, C for the equation Ax^2 + Bx +C = 0: ");
    scanf("%f %f %f", &a, &b, &c);

    delta = b * b - 4 * a * c;

    if (a == 0)
        if (b == 0)
            printf("There are no roots. \n");
```

```
        else
            printf("There is one root. \n");
    else
        if (delta > 0)
            printf("There are two roots. \n");
        else if (delta < 0)
            printf("There are no roots. \n");
        else
            printf("There are two equal roots. \n");
}
```

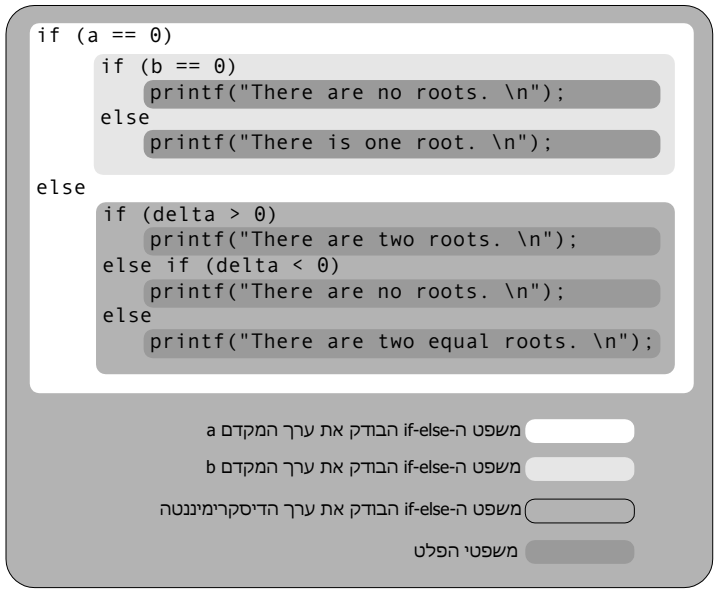
פלטים טיפוסיים של תוכנית זו ייראו כך:

```
Enter A, B, C for the equation Ax^2 + Bx + C = 0: 3 -6 3
There are two equal roots.
```

```
Enter A, B, C for the equation Ax^2 + Bx + C = 0: 0 2 -4
There is one root.
```

```
Enter A, B, C for the equation Ax^2 + Bx + C = 0: 1 1 -12
There are two roots.
```

משפטי if-else דואגים להצגת הודעה מתאימה עבור כל ערך אפשרי של המקדמים a, b ו-c. שייכות המשפטים המרכיבים את משפטי התנאי מוצגת להלן:



***איור 6.14** – שייכות משפטי התנאי שבתוכנית 6.5 זה לזה*

לעיתים, חובה להשתמש בסוגריים מסולסלים בקינון משפטי if-else, אפילו כשיש לבצע רק משפט אחד אם ביטוי התנאי מתקיים או לא. בחן את קטע התוכנית הבא בעיון רב:

```
if (a > 0)
    if (a < 10)
        printf("a > 0, a < 10");
else
    printf("a <= 0");
```

ממבט ראשון נראה קטע התוכנית נכון. הוא פולט הודעה אם ערך המשתנה a גדול מ-0 וקטן מעשר, והודעה אחרת אם ערך המשתנה a קטן או שווה ל-0. אך אם נציב במשתנה a את הערך 100 ונבצע את קטע התוכנית, יתקבל הפלט המפתיע הבא:

```
a <= 0
```

הסיבה לפלט המפתיע נעוצה בעובדה שכל משפט else משתייך, כברירת מחדל, אל משפט if בלי else שלפניו. לכן, קטע התוכנית נראה, למעשה, כך:

```
if (a > 0)
    if (a < 10)
        printf("a > 0, a < 10");
    else
        printf("a <= 0");
```

מכיון שחלק else שייך למשפט if השני ולא לראשון, המשפט השייך לו יבוצע אם ערך המשתנה a גדול או שווה לעשר, ולא אם הוא קטן או שווה ל-0.

כדי לשייך חלק else אל משפט if ש**אינו** הקרוב אליו ביותר, יש להיעזר בסוגריים מסולסלים. למשל, כדי שיתקבל פלט מתאים מהרצת קטע התוכנית הקודם, יש לכתוב אותו כך:

```
if (a > 0) {
    if (a < 10)
        printf("a > 0, a < 10");
}
else
    printf("a <= 0");
```

עתה, הסוגריים המסולסלים מנתקים את חלק else ממשפט if השני, והוא משויך למשפט if הראשון. לכן, משפט הפלט האחרון יתבצע רק אם ערך המשתנה a קטן או שווה ל-0 – כפי שרצינו.

עצה

קטע תוכנית זה ממחיש את הצורך בהזחה נאותה של שורות התוכנית. הקפדה על כך תסייע לך ולכל אדם הקורא את תוכניותיך להבינן.

המשפט switch

switch הוא משפט תנאי רב-עוצמה, המהווה תחליף אלגנטי למשפטי if-else מקוננים רבים. מקובל להשתמש בו כאשר ברצוננו להשוות ערך ביטוי בודד עם ערכים רבים, ולבצע בהתאם להשוואה פעולות שונות. נפתח בדוגמה קצרה. למשל, את קטע התוכנית הבא:

```
if (x == 1)
    printf("x is 1 \n");
else if (x == 2)
    printf("x is 2 \n");
else if (x == 3)
    printf("x is 3 \n");
```

ניתן להחליף במשפט switch כזה:

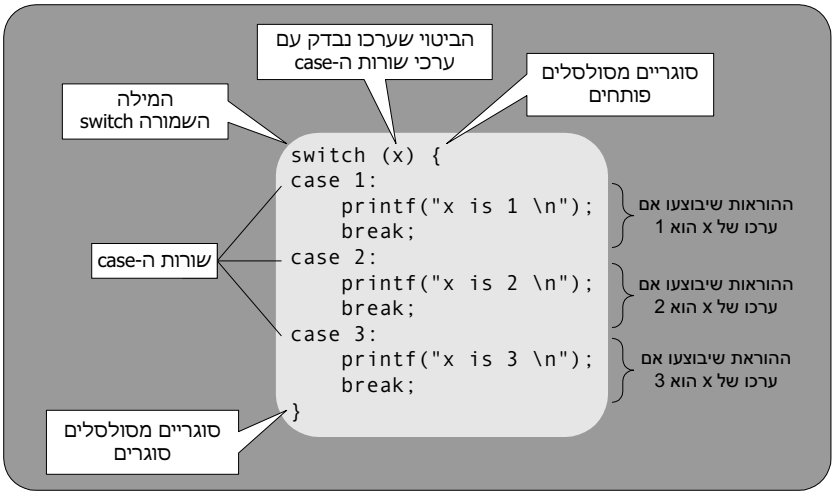
```
1   switch (x) {
2   case 1:
3       printf("x is 1 \n");
4       break;
5   case 2:
6       printf("x is 2 \n");
7       break;
8   case 3:
9       printf("x is 3 \n");
10      break;
11  }
```

המשפט נפתח במילה השמורה switch, ואחריה מופיע ביטוי בסוגריים. בתחילת ביצוע המשפט, ערך ביטוי זה נבדק ומושווה אל כל אחד מערכי שורות case המופיעות בגוף המשפט. בגוף משפט switch שלהלן מופיעות שלוש שורות case: בשורה (2) מופיעה שורת case שערכה 1, בשורה (5) מופיעה שורת case שערכה 2, ובשורה (8) מופיעה שורת case שערכה 3.

אם נמצאה שורת case שערכה שווה לערך הביטוי הנבדק, יתבצעו המשפטים המופיעים תחתיה, בזה אחר זה. אם לאו, יסתיים משפט switch, והמשפטים המופיעים אחריו יבוצעו כרגיל.

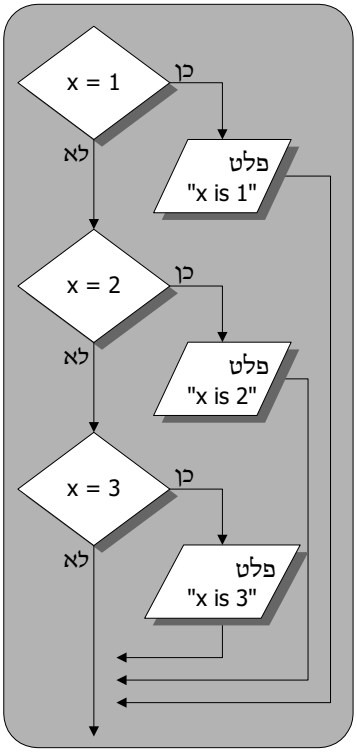
למשל, אם ערך המשתנה x הוא 2, יהיה ערך הביטוי הנבדק 2. שורת case שערכה 2 מופיעה בשורה (5), ולכן, יתבצעו המשפטים המופיעים אחריה: משפט הפלט שבשורה (6) ומשפט break שבשורה (7). כאשר מבוצע משפט break, מסתיימת פעולת משפט switch, וההוראות המופיעות אחריו (במקרה זה, החל משורה (12)) מתבצעות.

האיור הבא מציג את מבנהו של משפט switch:



***איור 6.15** – מרכיבי משפט switch*

תרשים זרימה מתאים למשפט break שלהלן מוצג באיור הבא:



***איור 6.16** –פעולת משפט switch המופיע באיור 6.15*

דוגמה

בדוגמה זו נשתמש במשפט switch פשוט וננתח את מרכיביו.

תוכנית 6.6 *SWITCH1.C*

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char digit;

    printf("Please enter a digit: ");

    digit = getch();
    putch(digit);

    switch (digit) {
    case '1':
        printf("\nYou pressed '1'! \n");
        break;
    case '2':
        printf("\nYou pressed '2'! \n");
        break;
    case '3':
        printf("\nYou pressed '3'! \n");
        break;
    }
}
```

פלט לדוגמה מהתוכנית ייראה כך:

```
Please enter a digit: 3
You pressed '3'!
```

```
#include <conio.h>
```

שורה זו חיונית לתוכניתנו משום שאנו קוראים לפונקציות putch ו-getch ממנה.

```
digit = getch();
putch(digit);
```

נזכיר את הפונקציות getch ו-putch. הפונקציה getch ממתינה ללחיצה על מקש ומחזירה את ערך ASCII שלו, שמוצב במשתנה digit. הפונקציה putch פולטת את התו שמתאים לפרמטר שנשלח אליה, שהוא התו שנלחץ קודם לכן. כלומר, שורות אלו קולטות תו מהמשתמש, מציבות את ערך ASCII שלו במשתנה digit ופולטות אותו.

```
switch (digit) {
    /* ... */
}
```

לאחר המילה השמורה switch מופיע, בסוגריים, הביטוי הנבדק. ביטוי זה חייב להחזיר ערך שלם. הסוגריים המסולסלים תוחמים את השורות הרבות של משפט switch.

שים לב!

אם משפט switch מכיל שורת case בודדת, ניתן לוותר על הסוגריים המסולסלים התוחמים אותה. מובן שבמקרה כזה אין צורך להשתמש במשפט switch, שהרי ניתן להסתפק במשפט if פשוט.

```
case '1':
```

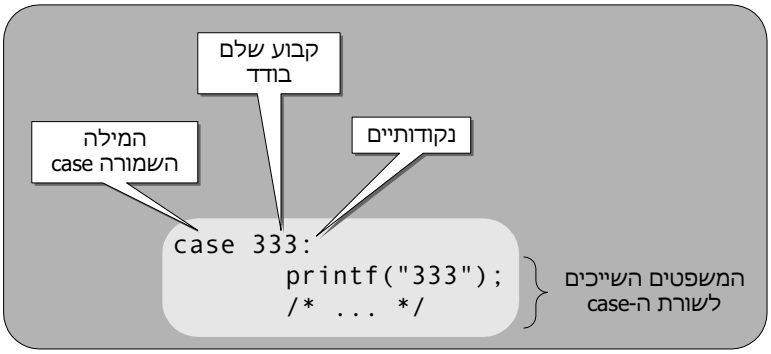
זו שורת case שערכה '1'. אם ערך הביטוי הנבדק שווה לערכה, יתבצעו המשפטים המופיעים אחריה, בזה אחר זה. בתוכנית זו קיימות שלוש שורות case, ולכל אחת ערך שונה. אין להגדיר מספר שורות case בעלות אותו ערך באותו משפט switch.

שורת case נפתחת במילה השמורה case. מייד אחריה מופיע ערך קבוע שלם, אחד ויחיד (ערך שורת case זו הוא '1'; זהו קבוע שלם, משום שסימנו של תו תמיד מוחלף בערך המספרי המתאים לו בטבלת ASCII). מאחר שערך משתנה אינו נחשב לערך קבוע, לא ניתן להשוות, במשפט switch, את ערך הביטוי הנבדק לערך משתנה. למשל, שורת case הבאה שגויה:

```
case num:   /* wrong, constant value required! */
```

לאחר הערך הקבוע תופענה נקודתיים, ואחריהן – המשפט או המשפטים השייכים לשורת case.

האיור הבא מציג את מרכיבי השורה:



***איור 6.17** – מרכיבי שורת case*

משפט switch מאפשר לבדוק רק יחס שוויון בין ערך הביטוי הנבדק לערכי שורות case. לא ניתן, למשל, לבדוק אם ערך הביטוי הנבדק גדול מערך שורת case מסוימת.

```
        printf("\nYou pressed '1'! \n");
        break;
```

אלה הם המשפטים המופיעים מתחת לשורת case הראשונה, ולכן הם מתבצעים אם ערכה הוא כשל ערך הביטוי הנבדק.

משפט break מביא, כאמור, ליציאה ממשפט switch. לרוב הוא יופיע בסוף קבוצת משפטים השייכים לשורת case מסוימת. משפט שיופיע אחריו, כמו משפט הפלט בקטע התוכנית הבא:

```
switch (value) {
case 3:
    /* ... */
    break;      /* quit the switch statement */
    printf("Unreachable!");
}
```

לא יבוצע כלל, מפני שמשפט switch יסתיים מייד לאחר משפט break.

גלישה

כזכור, כאשר ערך הביטוי הנבדק בראש משפט switch שווה לערך שורת case מסוימת, המשפטים שמתחתיה יבוצעו בזה אחר זה, עד שמשפט switch יסתיים באופן זה או אחר. נהוג לצאת ממשפט switch על ידי המשפט break. ואכן, בסוף כל קבוצת משפטים השייכת לשורת case מסוימת השתמשנו במשפט זה.

הבה נראה מה יתרחש אם נשמיט את משפטי break ממשפט switch הראשון שהצגנו (נעשה זאת על ידי הפיכתם להערה):

```
1   switch (x) {
2   case 1:
3       printf("x is 1 \n");
4       /** break;     --comment out the break-- **/
5   case 2:
6       printf("x is 2 \n");
7       /** break;     --comment out the break-- **/
8   case 3:
9       printf("x is 3 \n");
10      /** break;     --comment out the break-- **/
11  }
```

אם ערך המשתנה x הוא 1, ייפלט המלל הבא:

```
x is 1
x is 2
x is 3
```

נסביר עתה את הפלט המפתיע. ראשית, נבחרה שורת case שערכה הוא כשל הביטוי הנבדק. זו שורת case המופיעה בשורה (2), והמשפטים שמתחתיה מבוצעים בזה אחר זה. ראשון מבוצע משפט הפלט המופיע בשורה (3). משפט break שבשורה (4) מופיע בתוך הערה, ולכן, הוא אינו מתבצע. עקב כך, לא מסתיים משפט switch, והמשפטים הבאים מתבצעים על-פי סדר הופעתם – אף על-פי שהם שייכים לשורות case אחרות (שערכיהן שונים מערך הביטוי הנבדק). באופן דומה, משפטי break שבשורות (6) ו-(10) לא מסיימים את משפט switch, ולכן, כל המשפטים שבגוף משפט switch מתבצעים.

באופן דומה, אם ערך המשתנה x הוא 2, ייפלט המלל הבא:

```
x is 2
x is 3
```

במקרה זה, נבחרת שורת case המופיעה בשורה (5), והמשפטים שמתחתיה מבוצעים בזה אחר זה. מאחר שמשפטי break מופיעים כהערות, לא מתבצעת יציאה ממשפט switch בשורות (7) ו-(10), וכל המשפטים שמתחת לשורה (5) מבוצעים. תהליך זה, של מעבר אל שורת case הבאה, מכונה **גלישה**.

זהירות!

טעות נפוצה היא לשכוח להוסיף משפט break לאחר המשפטים השייכים לשורת case מסוימת. במקרה זה, תתבצע גלישה בלתי רצויה והתוכנית לא תפעל כמצופה. לעיתים קל לגלות את הטעות, אך בתוכנית מורכבת הדבר עלול לקחת זמן רב.

הצורך בהוספת משפט break למניעת גלישה לשורות case הבאות נראה בתחילה כטירדה, וברוב המקרים, הוא אכן כך. אך ניתן לנצל את תכונת הגלישה לתועלתנו. בחן, למשל, את קטע התוכנית הבא:

```
if (choice == '1' || choice == '2' || choice == '4')
    printf("You pressed 1, 2 or 4 \n");
else if (choice == '3')
    printf("You pressed 3 \n");
```

קטע קוד זה מבצע פעולה זהה אם ערך המשתנה choice הוא '1', '2' או '4'. לכן, אם נרצה להחליפו במשפט switch מתאים, נוכל לנצל את תכונה הגלישה, כפי שמוצג להלן:

```
1   switch (choice) {
2   case '1':       /* fall through */
3   case '2':       /* fall through */
4   case '4':
5       printf("You pressed 1, 2 or 4 \n");
6       break;
7   case '3':
8       printf("You pressed 3 \n");
9       break;
10  }
```

אם ערך המשתנה choice הוא '1' או '2', תיבחרנה שורות case המופיעות בשורות (2) או (3), בהתאמה. מתחת לשורות אלו לא מופיע משפט break, ולכן מתבצעת גלישה אל המשפטים המופיעים תחת שורת case שבשורה (4). משפט הפלט שבשורה (5) מבוצע ומשפט break שאחריו מביא לסיום משפט switch. אם ערך המשתנה choice הוא '3', יבוצע משפט הפלט שבשורה (8) ומשפט switch יסתיים.

במחשבה שנייה...

האם יש חשיבות לסדר בו מופיעות שורות case?

הסדר משמעותי רק אם מופיע משפט break בסוף כל קבוצות המשפטים השייכים לשורות case. אם לא מופיעים משפטי break, תתבצע גלישה למשפטים שבשורות case הבאות, ולכן הסדר בו הן מופיעות משמעותי.

שים לב!

שורות case לא חייבות להופיע על-פי סדר ערכיהן. למשל, במשפט switch הקודם מופיעה שורת case עם הערך '3' לאחר שורת case עם הערך '4'.

דוגמה

בדוגמה זו נדגים השמטת משפטי break לשם ביצוע גלישה. התוכנית תקלוט מהמשתמש תו, המייצג את ממוצע ציוניו, ותפלוט לאילו קורסים באוניברסיטה הוא יכול להתקבל.

תוכנית 6.7 *SWITCH2.C*

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char grade;

    printf("What is your grades average? \n");
    printf("'a' for 100-95 \n'b' for 94-85 \n");
    printf("'c' for 84-70 \n'd' for 69-55 \n");
    printf("'f' for below 55 \n");

    grade = getch();
    putch(grade);

    printf("\nCourses you can take: \n");

    switch (grade) {
    case 'f':
        printf("No courses available! \n");
        break;

    case 'a':
        printf("programming \n");
        /* fall through */
    case 'b':
        printf("medicine \n");
        /* fall through */
    case 'c':
        printf("law \n");
        /* fall through */
    case 'd':
        printf("history \n");
        break;
    }
}
```

פלט לדוגמה מהתוכנית ייראה כך:

```
What is your grades average?
'a' for 100-95
'b' for 94-85
'c' for 84-70
'd' for 69-55
'f' for below 55
b
Courses you can take:
medicine
law
history
```

```
    case 'f':
        printf("No courses available! \n");
        break;
```

שורת case זו תיבחר אם ערך המשתנה grade הוא 'f'. במקרה זה, ייפלט המלל המתאים ומשפט break יביא לסיום משפט switch.

```
    case 'a':
        printf("programming \n");
        /* fall through */
    case 'b':
        printf("medicine \n");
        /* fall through */
    case 'c':
        printf("law \n");
        /* fall through */
    case 'd':
        printf("history \n");
        break;
```

שורות case הללו מופיעות בסדר מתוכנן מראש. אם ערך המשתנה grade הוא 'a', רשאי הסטודנט להירשם לכל הקורסים, ויש לבצע את ארבעת משפטי הפלט. אם ערך המשתנה grade הוא 'b', רשאי הסטודנט להירשם לכל הקורסים פרט לקורס תכנות, ויש לבצע את שלושת משפטי הפלט האחרונים. אם ערך המשתנה grade הוא 'c', רשאי הסטודנט להירשם לקורסי משפטים והיסטוריה בלבד, ויש לבצע את שני משפטי הפלט האחרונים. אם ערך המשתנה grade הוא 'd', רשאי הסטודנט להירשם רק לקורס היסטוריה, ויש לבצע את משפט הפלט האחרון בלבד.

כל זאת מתאפשר בקלות בזכות תכונת הגלישה של משפט switch. בהעדר משפטי break, מתבצעים המשפטים השייכים לשורות case התחתונות גם כשנבחרו שורות case שמעליהן. לפיכך הוצבו שורות case המתאימות למקצועות הקשים יותר מעל לשורות case המתאימות למקצועות הקלים. באופן זה, יראה הפלט כי סטודנט הרשאי לגשת לקורס קשה יכול לגשת לכל הקורסים הקלים יותר.

במחשבה שנייה...

מהי חשיבותו של משפט break האחרון במשפט switch זה?

למעשה, ניתן לוותר על כל משפט break שמופיע אחרון במשפט switch. הסיבה היא, שמשפט switch מסתיים בכל מקרה לאחר שהמשפט האחרון שבגופו התבצע. כלומר, עם או בלי משפט break האחרון, יסתיים משפט switch.

שורת default

default היא מילה שמורה, שיכולה להופיע רק בגופו של משפט switch. המשפטים המופיעים מתחתיה יבוצעו אם לא נמצאה שורת case שערכה שווה לערך הביטוי הנבדק. למשל, את קטע התוכנית הבא:

```
if (num == 0)
    printf("num is 0 \n");
else if (num == 1)
    printf("num is 1 \n");
else
    printf("num is neither 0 nor 1 \n");
```

ניתן להחליף במשפט switch כזה:

```
switch (num) {
case 0:
    printf("num is 0 \n");
    break;
case 1:
    printf("num is 1 \n");
    break;
default:
    printf("num is neither 0 nor 1 \n");
    break;
}
```

שים לב!

לא חובה להוסיף את משפט default למשפטי switch, אך הוא אינו יכול להופיע יותר מפעם אחת באותו משפט switch.

ניתן לבצע גלישה גם באמצעות default. למשל, אם נכתוב את משפט switch הקודם כך:

```
switch (num) {
default:
    printf("num is neither 0 nor 1 \n");
case 0:
    printf("num is 0 \n");
    break;
case 1:
    printf("num is 1 \n");
    break;
}
```

וערך המשתנה num יהיה 3, המשפטים שמתחת לשורת default יבוצעו. מכיון שלא מופיע משפט break לאחר משפט הפלט הראשון, תתרחש גלישה לשורת case שערכה 0, ויוצג הפלט הבא:

```
num is neither 0 nor 1
num is 0
```

אופרטורים לוגיים

ניתן לחלק את האופרטורים בהם השתמשנו עד כה לשניים: אופרטורים אריתמטיים (כגון * ו-%) ואופרטורים יחסיים (כגון < ו-=>). נכיר עתה סוג נוסף של אופרטורים: **אופרטורים לוגיים** (logical operators). האופרטורים הלוגיים מופיעים לצד ביטוי אחד או יותר, ומחזירים, כמו אופרטורי היחס, אחד משני ערכים: **אמת** או **שקר**. הביטוי בו הם מופיעים מכונה **ביטוי לוגי** (logical expression).

אופרטור וגם הלוגי

ביטויי התנאי שהכרנו עד כה הכילו מספר אופרטורים אריתמטיים ואופרטור יחס אחד בלבד. אם, למשל, ברצוננו לבדוק אם ערך המשתנה a גדול מ-10, נכתוב משפט תנאי כזה:

```
if (a > 10)
    printf("a > 10");
```

אם ברצוננו לבדוק, בנוסף לתנאי שלעיל, אם ערך המשתנה a קטן מ-100, עלינו להוסיף משפט תנאי נוסף:

```
if (a > 10)
    if (a < 100)
        printf("a > 10, a < 100");
```

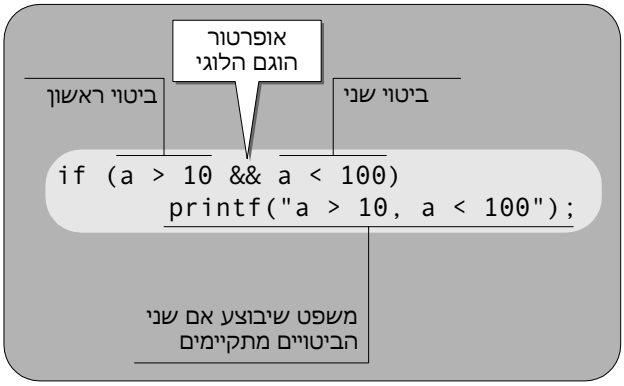
כאשר יש לוודא כי מספר רב של תנאים מתקיימים, הוספת משפטי if רבים גורמת לסרבול. בשפת C קיימת דרך נוחה יותר לבדיקת קיומם של תנאים מרובים, וזאת על ידי **אופרטור וגם הלוגי** (logical AND operator). אופרטור זה מופיע בין שני תת-ביטויים

ומאחד אותם לביטוי אחד, המתקיים רק אם שני תת-הביטויים מתקיימים. אופרטור וגם הלוגי מורכב משני תווי '&', כך: &&. בחן, למשל, את משפט התנאי הבא:

```
if (a > 10 && a < 100)
    printf("a > 10, a < 100");
```

ביטוי התנאי מכיל עתה שני תת-ביטויים, a > 10 ו-a < 100, המופרדים זה מזה על ידי אופרטור **וגם** הלוגי. שני תת-הביטויים נהפכים לביטוי אחד, שמתקיים רק כאשר a > 10 וגם a < 100. קטע זה שקול, אם כן, לקטע הקודם לו, אך בהיר וקצר ממנו בהרבה.

השימוש באופרטור **וגם** הלוגי מודגם באיור שלהלן:



***איור 6.18** – השימוש באופרטור וגם הלוגי*

איך לא...

אין להוסיף רווחים בין שני תווי '&' המרכיבים את אופרטור וגם הלוגי. כמו-כן היזהר לבל תכתוב תו '&' אחד במקום שניים. כמעט בכל ביטוי שניתן להשתמש באופרטור && ניתן להשתמש באופרטור &, אך התוצאה, כפי שנראה בהמשך, אינה זהה.

כפי שניתן לקנן מספר משפטי תנאי כדי לוודא שמספר תנאים מתקיימים, כך ניתן לכתוב ביטוי תנאי בעל מספר אופרטורי וגם לוגיים. למשל, את קטע התוכנית הבא:

```
if (a > 10)
    if (a < 100)
        if (a != b)
            printf("a > 10, a < 100, a != b");
```

ניתן (ורצוי) להחליף במשפט התנאי הבודד הבא:

```
if (a > 10 && a < 100 && a != b)
    printf("a > 10, a < 100, a != b");
```

דוגמה

בדוגמה זו נמחיש את השימוש באופרטור **וגם** הלוגי. בתוכנית זו ייקלט תו מהמשתמש ותיפלט הודעה בנוגע לסוגו – האם הוא אות לועזית גדולה, אות לועזית קטנה, ספרה או תו אחר.

תוכנית 6.8 *LOGICAND.C*

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char ch;

    printf("Please enter a character: ");

    ch = getch();
    putch(ch);

    printf("\nYou've pressed a");

    if (ch >= 'A' && ch <= 'Z')
        printf(" capital letter. \n");
    else if (ch >= 'a' && ch <= 'z')
        printf(" small letter. \n");
    else if (ch >= '0' && ch <= '9')
        printf(" digit. \n");
    else
        printf("n unidentified character. \n");
}
```

הנה מספר פלטים לדוגמה מהתוכנית:

```
Please enter a character: R
You've pressed a capital letter.
```

```
Please enter a character: e
You've pressed a small letter.
```

```
Please enter a character: 8
you've pressed a digit.
```

```
    printf("You've pressed a");
```

כל ההודעות שעלולות להיפלט במהלך ריצת התוכנית מתחילות באותן המילים, שנפלטות בהוראת פלט זו. בסופה לא יורד סמן הטקסט לשורה חדשה, וההודעות הבאות שייפלטו יופיעו לאחר שורה זו.

```
    if (ch >= 'A' && ch <= 'Z')
        printf(" capital letter. \n");
```

אופרטור וגם הלוגי גורם לביטוי התנאי להתקיים רק אם ערכו המספרי של התו שהוזן הוא בין ערכו המספרי של התו 'A' לערכו המספרי של התו 'Z', כלומר, אם הוזנה אות לועזית גדולה. במקרה זה תתבצע הוראת הפלט שלעיל.

```
    else if (ch >= 'a' && ch <= 'z')
        printf(" small letter. \n");
```

אופרטור וגם הלוגי גורם לביטוי התנאי להתקיים רק אם ערכו המספרי של התו שהוזן הוא בין ערכו המספרי של התו 'a' לערכו המספרי של התו 'z', כלומר, אם הוזנה אות לועזית קטנה. במקרה זה תתבצע הוראת הפלט שלעיל.

```
    else if (ch >= '0' && ch <= '9')
        printf(" digit. \n");
```

אופרטור וגם הלוגי גורם לביטוי התנאי להתקיים רק אם ערכו המספרי של התו שהוזן הוא בין ערכו המספרי של התו '0' לערכו המספרי של התו '9', כלומר, אם הוזנה ספרה. במקרה זה תתבצע הוראת הפלט שלעיל.

```
    else
        printf("n unidentified character. \n");
```

זהו חלק else הפנימי ביותר של משפטי if-else המקוננים. המשפט השייך לו יתבצע רק אם כל ביטויי if הקודמים לא התקיימו; היינו, אם התו שהוזן אינו אות לועזית גדולה, אות לועזית קטנה או ספרה.

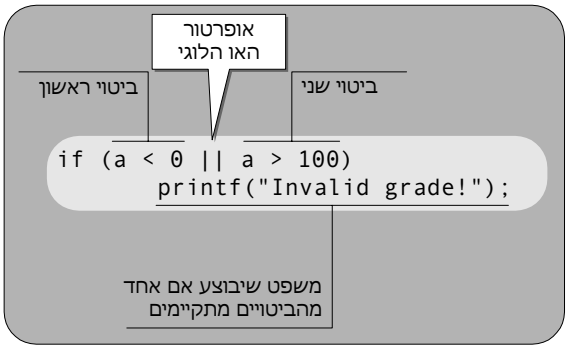
אופרטור או הלוגי

אופרטור לוגי חשוב נוסף הוא **אופרטור או הלוגי** (logical OR operator), המופיע אף הוא בין שני תת-ביטויים ומאחד אותם לביטוי אחד. מספיק שאחד משני תת-הביטויים יתקיים – והביטוי הכולל יתקיים אף הוא. אופרטור או הלוגי מורכב משני תווי '|', כך: ||. בחן, למשל, את קטע התוכנית הבא:

```
scanf("%d", &ch);
if (ch == 'a' || ch == 'A')
    printf("You've pressed the first English letter");
```

ביטוי התנאי מכיל שני תת-ביטויים, ch == 'a' ו-ch == 'A', המופרדים זה מזה על ידי אופרטור או הלוגי. שני תת-הביטויים הופכים לביטוי אחד, שמתקיים אם ערך המשתנה ch הוא 'a' או 'A'.

השימוש באופרטור או הלוגי מודגם באיור הבא:



***איור 6.19** – השימוש באופרטור או הלוגי*

איך לא...

אין להוסיף רווחים בין שני תווי '|' המרכיבים את אופרטור וגם הלוגי. כמו-כן, היזהר לבל תכתוב תו '|' אחד במקום שניים. כמעט בכל ביטוי שניתן להשתמש באופרטור || ניתן להשתמש באופרטור |, אך התוצאה אינה זהה.

ניתן להשתמש באופרטור או הלוגי עם מספר תת-ביטויים, כפי שמוצג להלן:

```
c = getch();
if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
    printf("You've entered a vowel letter!");
```

בקטע תוכנית זה מושווה התו שהוזן לאימות הקריאה הלועזיות. אם התו הוא אכן אם קריאה, תיפלט הודעה מתאימה. אילו רצינו לכתוב קטע תוכנית דומה בלי להשתמש באופרטור או הלוגי, היה עלינו לעשות זאת כך:

```
c = getch();
if (c == 'a')
    printf("You've entered a vowel letter!");
else if (c == 'e')
    printf("You've entered a vowel letter!");
else if (c == 'i')
    printf("You've entered a vowel letter!");
else if (c == 'o')
    printf("You've entered a vowel letter!");
else if (c == 'u')
    printf("You've entered a vowel letter!");
```

או כך:

```
c = getch();
switch (c) {
case 'a':
case 'e':
case 'i':
case 'o':
case 'u':
    printf("You've entered a vowel letter!");
    break;
}
```

קל לראות עד כמה השימוש באופרטורים הלוגיים מפשט את תוכניותינו.

דוגמה

בתוכנית הבאה שנציג נתרגל את השימוש באופרטור או הלוגי. התוכנית תקלוט שני ציונים מהמשתמש ותבדוק את נכונותם: אם הציונים אינם בתחום הדרוש (מ-0 עד 100), תיפלט הודעת שגיאה מתאימה. אם הציונים מתאימים, יחושב וייפלט הממוצע שלהם.

תוכנית 6.9 *LOGICOR.C*

```
#include <stdio.h>
void main()
{
    int math_grade, physics_grade;
    float average;

    printf("Please enter two grades: ");
    scanf("%d %d", &math_grade, &physics_grade);

    if (math_grade < 0 || math_grade > 100 ||
        physics_grade < 0 || physics_grade > 100)
        printf("Invalid input! Grade must range from 0 to 100. \n");
    else {
        average = (math_grade + physics_grade) / 2.0;
        printf("Average of grades is %f. \n", average);
    }
}
```

להלן מספר פלטים לדוגמה מהתוכנית:

```
Please enter two grades: 100 87
Average of grades is 93.500000.
```

```
Please enter two grades: 78 120
Invalid input! Grade must range from 0 to 100.
```

```
    if (math_grade < 0 || math_grade > 100 ||
        physics_grade < 0 || physics_grade > 100)
        printf("Invalid input! Grade must range from 0 to
            100. \n");
```

ביטוי התנאי שלעיל מורכב מארבעה תת-ביטויים. אם אחד מהציונים שהוזנו חורג מטווח הציונים המותר, יתקיים אחד מתת-הביטויים והוראת הפלט, המציגה הודעה מתאימה, תתבצע.

```
    else {
        average = (math_grade + physics_grade) / 2.0;
        printf("Average of grades is %f. \n", average);
    }
```

חלק else שלעיל יתבצע רק אם כל הציונים שהוזנו אינם חורגים מטווח הציונים המותר. במשפט המורכב השייך ל-else ממוצע הציונים מחושב ונפלט.

אופרטור השלילה הלוגי

האופרטור הלוגי השלישי שנכיר הוא **אופרטור השלילה הלוגי** (logical negation operator), המופיע משמאל לביטוי בודד והופך את ערכו הלוגי. כלומר, אם אופרטור השלילה מופיע לצד ביטוי שמתקיים, הוא הופך אותו לביטוי שאינו מתקיים, ואם הוא מופיע לצד ביטוי שאינו מתקיים, הוא הופך אותו לביטוי שמתקיים. אופרטור השלילה מורכב מתו סימן-קריאה בודד, כך: !.

למשל, כדי להפוך את ערכו הלוגי של הביטוי הבא:

```
3 < 6        /* this expression yields TRUE */
```

נשתמש באופרטור השלילה כך:

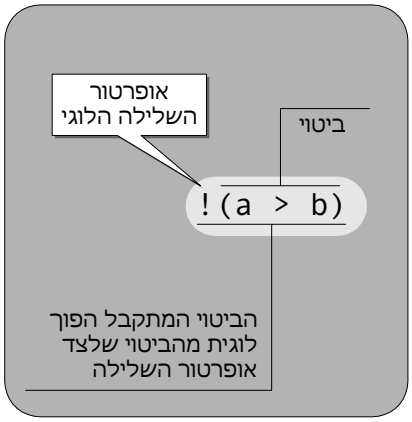
```
!(3 < 6)     /* this expression yields FALSE */
```

שים לב כיצד נתחם הביטוי 6 > 3 בסוגריים כאשר השתמשנו באופרטור השלילה. חשוב לתחום בסוגריים את הביטוי שאת ערכו הלוגי רוצים להפוך. אם נוותר על הסוגריים ונכתוב את הביטוי כך:

```
!3 < 6        /* confusing; same as:  (!3) < 6 */
```

יתייחס אופרטור השלילה הלוגי אל הקבוע 3, ולא אל הביטוי 6 > 3.

השימוש באופרטור השלילה הלוגי מודגם באיור הבא:



***איור 6.20** – השימוש באופרטור השלילה הלוגי*

ניתן להשתמש באופרטור השלילה גם עם ביטויים מורכבים יותר. בחן, למשל, את הביטוי הבא:

```
ch >= '0' && ch <= '9'
```

ביטוי זה מתקיים כאשר המשתנה ch מכיל את ערכה המספרי של אחת הספרות. אם נרצה לכתוב ביטוי שקול לביטוי זה, תוך שימוש באופרטור השלילה, נעשה זאת כך:

```
!(ch < 0 || ch > 9)
```

הביטוי שבסוגריים מתקיים כאשר המשתנה ch **אינו** מכיל את ערכה המספרי של אחת הספרות. אופרטור השלילה הלוגי, ההופך את ערכו הלוגי של הביטוי שמימינו, גורם לביטוי להתקיים כאשר המשתנה ch **כן** מכיל את ערכה המספרי של אחת הספרות.

ניתן לכתוב את אופרטור השלילה לצד אופרטור שלילה נוסף. בחן, למשל, את הביטוי הבא:

```
!(!(4 != 10))
```

הביטוי שבסוגריים החיצוניים, (10 != 4)!, הוא ביטוי שקר, משום שאופרטור השלילה הופך את ערכו הלוגי של הביטוי שבסוגריים הפנימיים. אך לצד ביטוי השקר מופיע אופרטור שלילה נוסף, שהופך את ערכו – מביטוי שקר לביטוי אמת. לכן, הביטוי הכולל הוא ביטוי אמת.

עצה

את הביטוי ((10 =! 4)!)! ניתן לכתוב גם ללא סוגריים חיצוניים, כך:

```
!!(4 != 10)
```

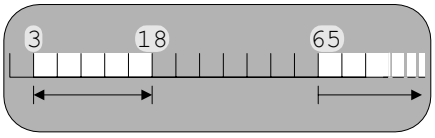
בחרנו להוסיף סוגריים שלא לצורך כדי להקל על הבנת הביטוי.
מומלץ לנהוג כך בשימוש בביטויים מורכבים.

שילוב אופרטורים לוגיים

נניח שאדם זכאי להנחה על כרטיס נסיעה באוטובוס אם גילו נע בין 3 ל-18 או עולה על 65. אם ברצוננו לבדוק האם הוא זכאי להנחה ולפלוט הודעה מתאימה, נוכל להשתמש בשני משפטי תנאי, כך:

```
if (age >= 65.0)
    printf("Your bus ticket is cheaper");
if (age >= 3.0 && age <= 18.0)
    printf("Your bus ticket is cheaper");
```

כרגיל, נעדיף לאחד שני משפטי תנאי למשפט תנאי בודד. כפי שמוצג באיור הבא, לא ניתן לפתח ביטוי תנאי השקול לביטוי שלעיל תוך שימוש רק באופרטורי וגם לוגיים או רק באופרטורי או לוגיים:



***איור 6.21** – גילאי הזכאים להנחה על כרטיסי אוטובוס*

לפיכך, עלינו לפתח ביטוי תנאי שיכיל את שניהם. ביטוי תנאי כזה מוצג להלן:

```
(age >= 65.0) || (age >= 3.0 && age <= 18.0)
```

ביטוי התנאי מורכב עתה משני תת-ביטויים התחומים בסוגריים: הראשון בודק האם גיל הנוסע הוא 65 (שנים) ומעלה. השני בודק האם גיל הנוסע הוא בין 3 ל-18 (שנים). שני תת-הביטויים משולבים באמצעות אופרטור או הלוגי, כך שאם אחד מהם מתקיים, גם הביטוי הכולל יתקיים.

נציג עתה את טבלת קדימויות האופרטורים שהכרנו עד כה:

אופרטורים	הערות מיוחדות
( ) ++ --	אופרטורי העלאה והפחתה עצמית אלה מופיעים לאחר שם המשתנה
+ - ++ -- !	אופרטורי פלוס ומינוס אלה מופיעים לפני ערך בודד וקובעים את סימנו (חיובי או שלילי) אופרטורי העלאה והפחתה עצמית אלה מופיעים לפני שם המשתנה
* / %	
+ -	אופרטורי פלוס ומינוס אלה מופיעים בין שני ערכים ומבצעים פעולת חיבור או חיסור ביניהם
< <= > >=	
== !=	
&&	
||	
= += -=	
*= /= %=	

ניתן להסיק מהטבלה שפעולת אופרטור וגם הלוגי קודמת לפעולת אופרטור או הלוגי.
לכן, את ביטוי התנאי הקודם ניתן לכתוב בלי סוגריים, כך:

```
age >= 65.0 || age >= 3.0 && age <= 18.0
```

דוגמה

נדגים שילובי אופרטורים לוגיים בתוכנית הבאה, המקבלת כקלט נתוני מועמד ופולטת אם הוא ראוי להתקבל לפקולטת מדעי המחשב:

תוכנית 6.10 *ANDOR.C*

```
#include <stdio.h>
void main()
{
    /* iq     = intelligence quotient
     * matric = matriculations average
     * psycho = psychometry grade
     */
```

```
    int iq, matric, psycho;

    printf("Please enter your IQ, matriculations average and
        psychometry grade: ");
    scanf("%d %d %d", &iq, &matric, &psycho);

    if ((iq > 170) || (matric > 110 && psycho > 720) ||
        (matric > 95 && psycho > 750) || (iq > 155 && matric > 90))
        printf("You've been accepted to the computer science
            faculty! \n");
    else
        printf("You've been rejected from the computer science
            faculty! \n");
}
```

להלן מספר פלטים לדוגמה מהתוכנית:

```
Please enter your IQ, matriculation average and psychometry grade:
150 120 760
You've been accepted to the computer science faculty!
```

```
Please enter your IQ, matriculation average and psychometry grade:
130 95 700
You've been rejected from the computer science faculty!
```

```
    int iq, matric, psycho;
```

בשורת הגדרה זו מוגדרים שלושה משתנים שלמים: iq, שייצג את מנת המשכל של המועמד, matric, שייצג את ממוצע ציוני בחינות הבגרות שלו, ו-psycho, שייצג את הציון הפסיכומטרי שלו.

```
    if ((iq > 170) || (matric > 110 && psycho > 720) ||
     (matric > 95 && psycho > 750) || (iq > 155 && matric > 90))
        printf("You've been accepted to the computer science
            faculty! \n");
```

משפט התנאי שלעיל בודק, באמצעות אופרטורי או לוגיים, אם המועמד מקיים אחד מהתנאים הבאים:

- מנת המשכל עולה על 170
- ממוצע ציוני בחינות הבגרות עולה על 110 וציון הפסיכומטרי עולה על 720
- ממוצע ציוני בחינות הבגרות עולה על 95 וציון הפסיכומטרי עולה על 750
- מנת המשכל עולה על 155 וממוצע ציוני בחינות הבגרות עולה על 90

מועמד שעונה על אחת מהדרישות מתקבל לפקולטת מדעי המחשב, והודעה מתאימה נפלטת על המסך.

```
    else
        printf("You've been rejected from the computer science
            faculty! \n");
```

חלק else יתבצע אם ביטוי התנאי לא התקיים, כלומר, אם המועמד אינו עונה על דרישות הקבלה. במקרה כזה, לא מתקבל המועמד לפקולטת מדעי המחשב, והודעה מתאימה נפלטת על המסך.

ערכי ביטויי יחס

עד כה קבענו כי כל ביטוי יחס יכול להחזיר אחד משני ערכים בלבד: אמת או שקר. למעשה, ערכים אלה מיוצגים בשפת C באמצעות מספרים רגילים, כך שערך ביטוי שמתקיים הוא 1, וערך ביטוי שאינו מתקיים הוא 0. למשל, ערך הביטוי הבא:

```
2 < 4
```

הוא 1, ולכן, אם נבצע את המשפט הבא:

```
bool = 2 < 4;
```

יוצב במשתנה bool הערך 1. באופן דומה, אם נבצע את המשפט הבא:

```
printf("Value is: %d", 3 == 5);
```

הביטוי 3 == 5 אינו מתקיים (ערכו 0), ולכן ייפלט המלל הבא:

```
Value is: 0
```

בדוגמה זו נדגים את השימוש בביטויי יחס ובערכים המוחזרים מהם:

תוכנית 6.11 *MORE_EXP.C*

```
#include <stdio.h>
void main()
{
    int a, b, c;

    printf("Please enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    printf("a > b:  %d \n", a > b);
    printf("a + 1 == b - c:  %d \n", a + 1 == b - c);
    printf("-a - 4 <= b - ++c:  %d \n", -a - 4 <= b - ++c);
    printf("a - 1 > (b - c != 6):  %d \n", a - 1 > (b - c != 6));
}
```

פלטים לדוגמה מהתוכנית מוצגים להלן:

```
Please enter three numbers: 1 2 3
a > b:  0
a + 1 == b - c:  0
-a - 4 <= b - ++c:  1
a - 1 > (b - c != 6):  0
```

```
Please enter three numbers: 4 2 -1
a > b:  1
a + 1 == b - c:  0
-a - 4 <= b - ++c:  1
a - 1 > (b - c != 6):  1
```

```
    printf("a > b:  %d \n", a > b);
```

משפט זה פולט את הערך 1 אם ערך המשתנה a גדול מערך המשתנה b, ואת הערך 0 אם לאו.

```
    printf("a + 1 == b - c:  %d \n", a + 1 == b - c);
```

משפט זה פולט את הערך 1 אם ערך הביטוי a + 1 שווה לערך הביטוי b - c, ואת הערך 0 אם לאו.

```
printf("-a - 4 <= b - ++c:  %d \n", -a - 4 <= b - ++c);
```

משפט זה פולט את הערך 1 אם ערך הביטוי a - 4- קטן או שווה לערך הביטוי b - ++c, ואת הערך 0 אם לאו. שים לב שאופרטור ההעלאה העצמית מוסיף 1 לערך המשתנה c ומחזיר את ערכו לאחר ההעלאה ב-1.

```
printf("a - 1 > (b - c != 6):  %d \n", a - 1 > (b - c != 6));
```

משפט זה פולט את הערך 1 אם ערך הביטוי a - 1 גדול מערך הביטוי הלוגי b - c != 6 ואת הערך 0 אם לאו.

ניתן לערב אופרטורי יחס ואופרטורים אריתמטיים עם אופרטורים לוגיים. בחן, למשל, את הביטוי הבא:

```
6 >= 1 && 6 == 7 - 1
```

פעולת אופרטורי היחס קודמת לפעולת אופרטור וגם הלוגי, ולכן ביטוי זה מחושב כך:

```
(6 >= 1) && (6 == 7 - 1)
```

תת-הביטוי שמשמאל לאופרטור וגם הלוגי מתקיים, מכיון ש-1 =< 6, והוא מחזיר את הערך 1. גם תת-הביטוי שמימין לאופרטור וגם הלוגי מתקיים, ולכן הביטוי הכולל מתקיים אף הוא. הנה דוגמה נוספת:

```
3 != 8 / 2 - 1 || (a = 0)
```

פעולת החילוק קודמת לפעולת אופרטור או הלוגי, ולכן ביטוי זה מחושב כך:

```
(3 != 8 / 2 - 1) || (a = 0)
```

תת-הביטוי שמשמאל לאופרטור או הלוגי אינו מתקיים, ולכן הוא מחזיר את הערך 0. תת-הביטוי שמימין לאופרטור או הלוגי מציב את הערך 0 במשתנה a, וזהו הערך המוחזר ממנו. שני הערכים ביניהם מופיע אופרטור או הלוגי הם ערכי שקר, ולכן גם הביטוי הכולל אינו מתקיים.

במחשבה שנייה...

האם הסוגריים בביטוי האחרון הכרחיים?

אם נוותר על הסוגריים התוחמים את פעולות ההשמה בביטוי האחרון, כך:

```
3 != 8 / 2 - 1 || a = 0
```

יתקבל ביטוי שגוי מבחינה תחבירית. הסיבה היא, שאופרטור ההשמה מתבצע לאחר שמתבצעים שאר האופרטורים שבביטוי. לכן ביטוי זה שקול לביטוי הבא:

```
(3 != 8 / 2 - 1 || a) = 0
```

מכיון שמשמאל לאופרטור ההשמה חייב להופיע משתנה אחד ויחיד, זהו משפט שגוי.

כאשר מופיע אופרטור השלילה, נבדק קודם הערך שמימין לו, ואחר כך נהפך ערכו. למשל, את הביטוי הבא:

```
!!4
```

יש לקרוא כך:

```
!(!(4))
```

כלומר, כך:

```
!(0)
```

ביטוי זה מחזיר את הערך 1, ולכן הוא ביטוי אמת.

דוגמה

בדוגמה זו נדגים את השימוש בביטויים לוגיים מורכבים, הכוללים אופרטורים מסוגים שונים:

תוכנית 6.12 *LOGIC.C*

```
#include <stdio.h>
void main()
{
    int a, b, c;

    printf("Please enter three numbers: ");
    scanf("%d %d %d", &a, &b, &c);

    printf("!c > 0 || a > b - 1:  %d \n",!c > 0 || a > b - 1);
    printf("!!(a == b):  %d \n", !!(a == b));
    printf("c && (a * b) || 1:  %d \n", c && (a * b) || 1);
    printf("(a == 0 || b == 0) && (a > c || c > b):  %d \n",
        (a == 0 || b == 0) && (a > c || c > b));
}
```

פלטים לדוגמה מהתוכנית מוצגים להלן:

```
Please enter three numbers: 1 2 0
!c > 0 || a > b - 1:  1
!!(a == b):  0
c && (a * b) || 1:  1
(a == 0 || b == 0) && (a > c || c > b):  0
```

```
Please enter three numbers: 2 3 2
!c > 0 || a > b - 1:  0
!!(a == b):  0
c && (a * b) || 1:  1
(a == 0 || b == 0) && (a > c || c > b):  0
```

```
    printf("!c > 0 || a > b - 1:  %d \n", !c > 0 || a > b - 1);
```

משפט זה פולט את ערך הביטוי הלוגי השקול לביטוי הבא:

```
((!c) > 0) || (a > (b - 1))
```

```
    printf("!!(a == b):  %d \n", !!(a == b));
```

משפט זה פולט את ערך הביטוי הלוגי השקול לביטוי הבא:

```
!(!(a == b))     /* actually, returns 1 if a == b */
```

```
    printf("c && (a * b) || 1:  %d \n", c && (a * b) || 1);
```

משפט זה פולט את ערך הביטוי הלוגי השקול לביטוי הבא:

```
(c && (a * b)) || 1
```

שים לב!

ביטוי זה מבצע פעולת **או** בין שני תת-ביטויים, כאשר תת-הביטוי הימני מתקיים תמיד. לכן גם הביטוי הכולל תמיד יתקיים.

```
    printf("(a == 0 || b == 0) && (a > c || c > b):  %d \n",
        (a == 0 || b == 0) && (a > c || c > b));
```

משפט זה פולט את ערך הביטוי הלוגי השקול לביטוי הבא:

```
((a == 0) || (b == 0)) && ((a > c) || (c > b)))
```

נציג כעת מספר ביטויי יחס מבלבלים בשפת C. ראשית, בחן את הביטוי הבא:

```
-2 < -1 < 0
```

בשפת המתמטיקה, ביטוי זה מציין ש-1- גדול מ-2- וקטן מ-0. זוהי, כמובן, טענה נכונה. אך בשפת C זהו ביטוי שקר! נבדוק מדוע. ערכים המופיעים משני צדי אופרטורי יחס נבדקים תמיד משמאל לימין, ולכן, הביטוי שלהלן שקול לביטוי הבא:

```
(-2 < -1) < 0
```

ראשון נבדק תת-הביטוי 1- < 2-. תת-ביטוי זה מתקיים ומחזיר את הערך 1. כלומר, הביטוי המתקבל הוא:

```
1 < 0
```

וזהו ביטוי שקר. ביטוי מבלבל נוסף הוא הביטוי הבא:

```
5 == 5 == 5
```

אף ששלושת הערכים המושווים שווים זה לזה, בשפת C ביטוי כזה הוא ביטוי שקר. מאחר שערכים המופיעים משני צדי אופרטורי יחס נבדקים משמאל לימין, הביטוי שלעיל שקול לביטוי הבא:

```
(5 == 5) == 5
```

ראשית נבדק תת-הביטוי 5 == 5. תת-ביטוי זה מתקיים ומחזיר את הערך 1. כלומר, הביטוי המתקבל הוא:

```
1 == 5
```

וזהו, כמובן, ביטוי שקר.

כאשר נכתוב ביטויים לוגיים ונשלב בהם ערכים מספריים, הערך 0 ייחשב לערך שקרי, וכל ערך פרט ל-0 ייחשב לערך אמת. למשל, הביטוי הבא:

```
4 && -7
```

הוא ביטוי אמת, משום שהערכים 4 ו-7- נחשבים לערכי אמת (ואופרטור וגם הלוגי מחזיר ערך אמת כאשר שני הערכים שמצדדיו הם ערכי אמת). לעומת זאת, הביטוי הבא:

```
4 && 0
```

אינו מתקיים, משום שהערך 0 נחשב לערך שקרי. באופן דומה, הביטוי הבא:

```
!5
```

יחזיר את הערך 0, משום שהערך 5 נחשב לערך אמת, והערך הלוגי ההפוך לערך אמת הוא ערך שקר, שערכו 0.

שים לב!

את משפט התנאי הבא:

```
if (a != 0)      /* ... */
```

ניתן (ולעיתים מקובל) להחליף במשפט התנאי:

```
if (a)      /* ... */
```

ביטוי התנאי השני בודק את ערך המשתנה a. כל ערך פרט ל-0 נחשב לערך אמת, ולכן, אם הערך של a שונה מ-0, הביטוי יתקיים. אם ערך a שווה ל-0, הביטוי לא יתקיים. בדיקה זו זהה לבדיקה המבוצעת על ידי ביטוי התנאי הראשון. באופן דומה, את משפט התנאי הבא:

```
if (a == 0)      /* ... */
```

ניתן (ולעיתים מקובל) להחליף במשפט התנאי:

```
if (!a)      /* ... */
```

בפרק הקודם הזהרנו מפני בלבול בין אופרטור השוויון (==) לאופרטור ההשמה (=). בחן, למשל, את הביטוי הבא:

```
if (a = b)      /* instead of:  a == b */
    printf("a equals b");
```

במקרה זה, מכיון שאופרטור ההשמה מחזיר את הערך שהוצב במשתנה שמשמאלו, ערך ביטוי התנאי יהיה ערך המשתנה b. לכן, אם ערך המשתנה b שונה מ-0, יתקיים ביטוי התנאי, ואם ערך המשתנה b שווה ל-0, ביטוי התנאי לא יתקיים. זו טעות מסוכנת ביותר, שקשה לאתרה.

טעות מסוכנת נוספת מוצגת במשפט הבא:

```
a == 5;          /* instead of:  a = 5; */
```

במקרה זה התכוון המתכנת לבצע פעולת השמה, אך בטעות השתמש באופרטור השוויון. המהדר לא יציב את הערך 5 במשתנה a, אלא ישווה בין הערכים ויחזיר ערך 1 או 0 בהתאם לתוצאת ההשוואה. בערך זה לא נעשה כלל שימוש, והמהדר מתעלם ממנו וממשיך למשפט הבא.

סיכום

- ביטוי יחס הוא ביטוי שמכיל אופרטור יחס ומחזיר ערכי אמת או שקר.
- משפט if מבצע את המשפט השייך לו אם ביטוי התנאי שלו מתקיים.
- כל משפט יכול להיות משפט מורכב, ובכך להכיל מספר משפטים נוספים.
- משפט if-else מבצע משפט אחד אם ביטוי התנאי שלו מתקיים, ומשפט אחר אם לאו.
- ניתן לשייך משפטי תנאי אל משפטי תנאי אחרים.
- משפט switch משווה ערך ביטוי שלם אל הקבועים של שורות case השייכות לו, ומבצע את המשפטים המופיעים לאחר השורה שהקבוע שלה מתאים לערך זה.
- משפט break מאפשר לצאת ממשפט switch.
- שורת default תיבחר אם אף אחד מערכי שורות case אינו שווה לערך הביטוי שנבדק במשפט switch. במקרה זה, יתבצעו המשפטים המופיעים אחריה.
- אופרטורים לוגיים מחזירים ערכי אמת או שקר בלבד.
- אופרטור וגם הלוגי מחזיר ערך אמת אם שני הביטויים שמשני צדיו מתקיימים.
- אופרטור או הלוגי מחזיר ערך אמת אם אחד מהביטויים שמשני צדיו מתקיים.
- אופרטור השלילה הלוגי הופך את ערכו הלוגי של הביטוי שמימינו.
- ביטוי אמת מחזיר את הערך 1 וביטוי שקר מחזיר את הערך 0.
- כל ערך פרט ל-0 נחשב לביטוי אמת.

פרק 7

לולאות

ההיכרות עם משפטי הפלט והקלט, המשתנים ומשפטי התנאי אפשרה לנו לכתוב תוכניות מורכבות למדי. הגיעה העת להרחיב עוד יותר את יכולות תוכניתנו, וזאת על ידי הכרת הלולאות. את הלולאות בשפת C מרכיבים משפטים מיוחדים, ובפרק זה נכירם.

הלולאה while

בשפת C קיימות שלוש **לולאות** (loops) השונות בתחבירן אך מבצעות, בסופו של דבר, פעולה דומה: ביצוע משפט מסוים מספר פעמים. משפט זה הוא המשפט השייך ללולאה.

הלולאה הראשונה שנכיר היא **הלולאה while**. לולאה זו מבצעת את המשפט השייך לה כל עוד תנאי מסוים מתקיים, והיא דומה ללולאה "כל עוד ... בצע" שהכרנו בפרק הראשון. בחן, למשל, את הלולאה המוצגת באלגוריתם הבא:

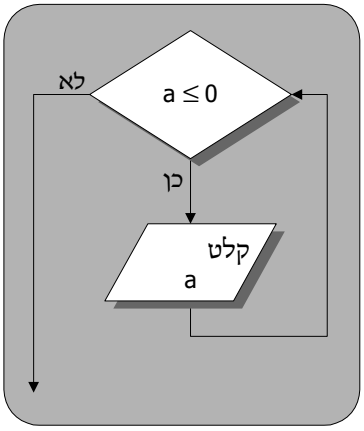
(1) כל עוד $a \leq 0$ בצע

(1.1) קלוט מספר למשתנה a

לולאה זו קולטת מספרים מהמשתמש שוב ושוב, עד שמוזן ערך חיובי. לולאת while המתאימה ללולאה זו תיראה כך:

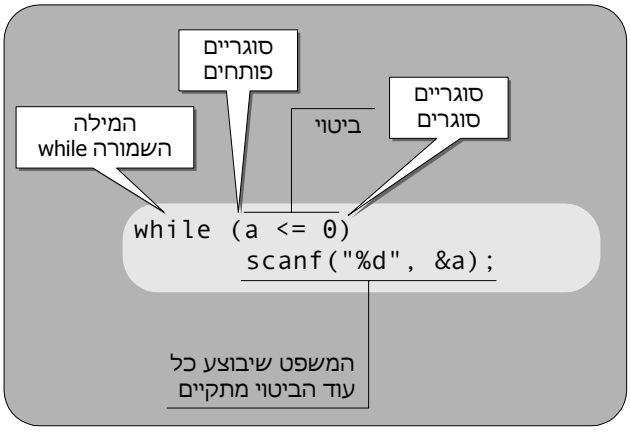
```
1   while (a <= 0)                /* as long as a <= 0 ... */
2       scanf("%d", &a);          /* ... execute this statement */
3   /* ... */
```

כפי שמוצג באיור 7.1, לולאת while שלהלן מתחילה בבדיקת הביטוי 0 => a. אם הביטוי מתקיים, יבוצע משפט הקלט והביטוי ייבדק שוב. אם הוא עדיין מתקיים, יבוצע שוב משפט הקלט והביטוי ייבדק בפעם השלישית. תהליך זה יתבצע שוב ושוב, עד שהביטוי לא יתקיים. כאשר יקרה הדבר, תסתיים הלולאה, והמשפטים שבשורה (3) ואילך יתבצעו כרגיל. הביטוי לא יתקיים כאשר יוזן ערך הגדול מ-0, ולכן הלולאה תסתיים רק כאשר יזין המשתמש מספר חיובי.



***איור 7.1** – לולאה הקולטת ערכים עד שמוזן ערך חיובי*

השימוש בלולאת while מוצג באיור הבא:



***איור 7.2** – השימוש בלולאה while*

מבחינה תחבירית, בנויה לולאת while משני חלקים. החלק הראשון מורכב מהמילה השמורה while ומביטוי בסוגריים, שיכונה מעתה **ביטוי הלולאה**. החלק השני מורכב מהמשפט השייך ללולאה, המתבצע כל עוד ביטוי הלולאה מתקיים.

זהירות!

אין להוסיף נקודה-פסיק לאחר הסוגריים הסוגרים של ביטוי הלולאה, כפי שמוצג להלן:

```
while (a == 0);      /* dangerous! */
    scanf("%d", &a);
```

זו טעות מסוכנת, משום שהמהדר לא יזהיר אותנו מפניה, אך הלולאה לא תתבצע כנדרש.

ביצוע בודד של המשפט השייך ללולאה מכונה **איטרציה** (iteration).

כדי לבצע מספר משפטים כל עוד ביטוי הלולאה מתקיים, נתחום אותם בסוגריים מסולסלים, כך שיווצר מהם משפט מורכב. הנה דוגמה לכך:

```
int counter = 4;
while (counter >= 1) {
    printf("%d \n", counter);
    counter--;
}
```

בראשית הקטע מוגדר משתנה שלם בשם counter, המאותחל ל-4. הלולאה מתבצעת כל עוד ערך counter גדול או שווה ל-1. נדמה עתה את פעולתה:

לפני הכניסה הראשונה ללולאה נבדק ביטוי הלולאה. מכיון שביטוי זה מתקיים, המשפט המורכב השייך ללולאה מתבצע, ולכן ערך המשתנה counter נפלט ומופחת ב-1.

לאחר האיטרציה הראשונה של הלולאה ערך המשתנה counter הוא 3. ביטוי הלולאה נבדק בשנית, ומאחר שהוא מתקיים, מתבצע המשפט המורכב השייך ללולאה, ולכן ערך המשתנה counter נפלט ומופחת ב-1.

לאחר האיטרציה השנייה של הלולאה ערך המשתנה counter הוא 2. ביטוי הלולאה נבדק בשלישית, ומאחר שהוא מתקיים, מתבצע המשפט המורכב השייך ללולאה, ולכן ערך המשתנה counter נפלט ומופחת ב-1.

לאחר האיטרציה השלישית של הלולאה ערך המשתנה counter הוא 1. ביטוי הלולאה נבדק ברביעית, ומאחר שהוא מתקיים, מתבצע המשפט המורכב השייך ללולאה, ולכן ערך המשתנה counter נפלט ומופחת ב-1.

לאחר האיטרציה הרביעית של הלולאה ערך המשתנה counter הוא 0. ביטוי הלולאה נבדק בחמישית, ומאחר שהוא אינו מתקיים, מסתיימת הלולאה. בסך-הכל מתבצע משפט הפלט ארבע פעמים, ומביא לפליטת המלל הבא:

```
4
3
2
1
```

קטע תוכנית שפעולתו הפוכה לזו של קטע התוכנית הקודם ייראה כך:

```
int counter = 1;
while (counter <= 4) {
    printf("%d \n", counter);
    counter++;
}
```

עתה, מאותחל המשתנה counter לערך 1. ערך המשתנה counter עולה ב-1 בכל איטרציה, וכאשר יגיע לערך 5, לא יתקיים ביטוי הלולאה והלולאה תסתיים. כמו הלולאה הקודמת, גם לולאה זו מתבצעת ארבע פעמים, אך ערך המשתנה counter עולה בכל איטרציה, ולא יורד. לכן, הפלט המתקבל הוא זה:

```
1
2
3
4
```

דוגמה

בדוגמה זו נשתמש בלולאת while כדי להציג עשר כפולות ראשונות של מספר שמזין המשתמש. למשל, אם יזין המשתמש את הערך 4, יוצגו המספרים 0, 4, 8, 12 וכדומה.

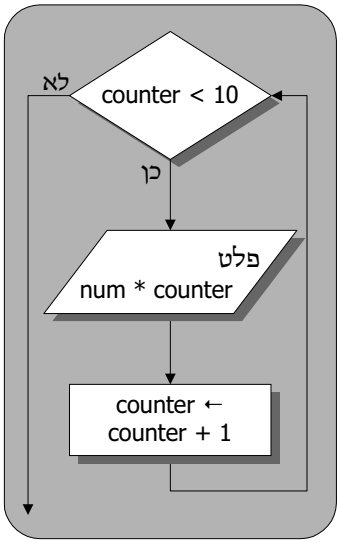
תוכנית 7.1 *MULT.C*

```
#include <stdio.h>
void main()
{
    int num, counter = 0;

    printf("Please enter a number: ");
    scanf("%d", &num);

    while (counter < 10) {
        printf("%d ", num * counter);
        counter++;
    }
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter a number: 7
0 7 14 21 28 35 42 49 56 63
```

תרשים הזרימה החלקי הבא מייצג את הלולאה שבתוכנית זו:



איור 7.3 *– לולאה לפליטת עשר הכפולות הראשונות של המספר num*

```
int num, counter = 0;
```

בשורת הגדרה זו מוגדרים שני משתנים שלמים. במשתנה num יוצב המספר שמזין המשתמש. במשתנה counter נשתמש כמונה, וניעזר בו כדי לבצע את הלולאה עשר פעמים.

```
while (counter < 10) {
    printf("%d ", num * counter);
    counter++;
}
```

ביטוי הלולאה מתקיים כל עוד ערך המשתנה counter קטן מ-10. המשתנה counter מאותחל ל-0 וערכו עולה ב-1 בכל איטרציה, כך שהלולאה מתבצעת, בסך-הכל, עשר פעמים.

משפט הפלט שבגוף הלולאה (כלומר, בין הסוגריים המסולסלים), פולט את מכפלת הערך שבמשתנה num בערך שבמשתנה counter. המשתנה counter מאותחל ל-0 וערכו עולה ב-1 בכל איטרציה, ולכן נפלטות עשר כפולותיו הראשונות של המספר num.

זהירות!

זכור להוסיף לגופה של כל לולאה משפט המשנה את ערך המשתנה המונה. אם תשכח לעשות זאת, ביטוי הלולאה יתקיים תמיד, והלולאה תתבצע אינסוף פעמים. נרחיב על כך בהמשך.

בתוכנית הבאה נשתמש בלולאת while כדי לחשב עצרת של מספר שמזין המשתמש.

תוכנית 7.2

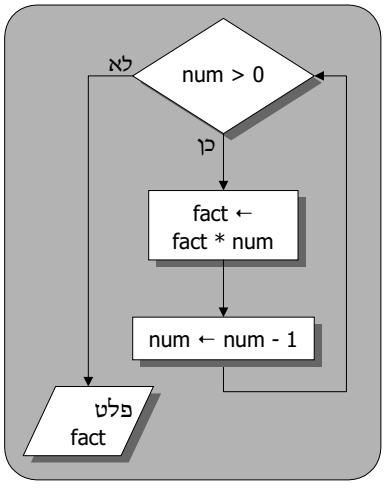
FACT.C

```
#include <stdio.h>
void main()
{
    int num;
    long fact = 1;

    printf("Please enter a number: ");
    scanf("%d", &num);

    while (num > 0) {
        fact *= num;
        num--;
    }

    printf("Factorial is: %ld \n", fact);
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter a number: 12
Factorial is: 479001600
```

תרשים הזרימה הבא מייצג את אופן חישוב ופליטת עצרת המספר שהוזן:



איור 7.4 – לולאה לפליטת עצרת של מספר

```
int num;
long fact = 1;
```

שני משתנים אלה ישמשו אותנו למציאת העצרת. המספר שיזין המשתמש יוצב במשתנה num, והעצרת שלו תחושב על ידי המשתנה fact. בחרנו להשתמש במשתנה מסוג long לחישוב העצרת, משום שהעצרת של מספרים נמוכים גבוהה לאין שיעור. אם נשתמש במשתנה מסוג int לחישוב העצרת, לא נוכל לחשב עצרת של מספר הגבוה מ-7. עם משתנה מסוג long, לעומת זאת, ניתן לחשב עצרות של מספרים גבוהים יותר.

```
while (num > 0) {
    fact *= num;
    num--;
}
```

לולאה זו תסתיים כאשר ערך המשתנה num לא יהיה חיובי. בכל איטרציה פוחת ערך המשתנה num ב-1, ולכן, מספר הפעמים שהלולאה תתבצע שווה למספר שהזין המשתמש.

בכל איטרציה של הלולאה מוכפל ערך המשתנה fact בערך המשתנה num. המשתנה fact מאותחל לערך 1, ולכן, לאחר האיטרציה הראשונה הוא יכיל את ערך המשתנה num. באיטרציה השנייה יוכפל ערכו בערך המשתנה num - 1. באיטרציה השלישית יוכפל ערכו בערך המשתנה num - 2, וכן הלאה. בסופו של דבר, יכיל המשתנה fact את הערך של הביטוי הבא:

$$num * (num - 1) * (num - 2) * (num - 3) * \ldots * 1$$

שהוא עצרת המספר num.

במחשבה שנייה...

מה יתרחש אם יזין המשתמש את המספר 0?

אם יוזן הערך 0, לא יתקיים ביטוי הלולאה והלולאה לא תתבצע כלל. במקרה זה, ערך המשתנה fact יהיה הערך בו הוא אותחל – 1. ואכן, עצרת המספר 0 היא 1.

```
printf("Factorial is: %ld \n", fact);
```

משפט זה מביא לפליטת הערך של המשתנה fact, המכיל את העצרת המבוקשת. זכור כי משתני long נפלטים באמצעות תו ההמרה %ld ולא באמצעות %d.

בדוגמה זו נחשב חזקה של שני מספרים שלמים באמצעות לולאת while. התוכנית שנכתוב תקבל שני ערכים, המייצגים את בסיס החזקה ואת מעריך החזקה, ותחשב את התוצאה.

במחשבה שנייה...

מהו בסיס החזקה ומהו מעריך החזקה?

בסיס החזקה הוא המספר המוכפל, ומעריך החזקה הוא המספר המכפיל אותו. למשל, בביטוי 2^3, 2 הוא בסיס החזקה ו-3 הוא מעריך החזקה.

תוכנית 7.3

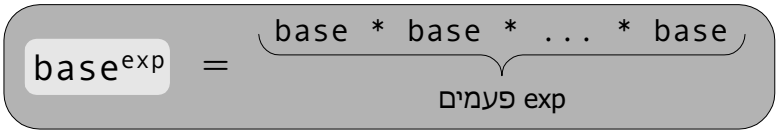
POWER.C

```
#include <stdio.h>
void main()
{
    int base, exp, power = 1;

    printf("Enter a base and an exponent: ");
    scanf("%d %d", &base, &exp);

    while (exp > 0) {
        power *= base;
        exp--;
    }

    printf("The power is: %d \n", power);
}
```

הנה פלט לדוגמה מהתוכנית:

```
Enter a base and an exponent: 5 3
The power is: 125
```

התוכנית מוצאת את החזקה על ידי כפילת בסיס החזקה בעצמו, מספר פעמים השווה למעריך החזקה. האיור הבא מדגים זאת:



***איור 7.5** – מציאת ערך ביטוי מסוג $base^{exp}$*

כדי לחשב את החזקה השתמשנו בלולאת while המתבצעת exp פעמים. בכל איטרציה שלה מוכפל ערך משתנה העזר power, המאותחל ל-1, בערך המשתנה base. באופן זה, עם סיום הלולאה יימצא במשתנה העזר ערך השווה לחזקה הרצויה.

```
    while (exp > 0) {
        /* ... */
```

לולאה זו מתבצעת כל עוד ערך המשתנה exp גדול מ-0. בכל איטרציה של הלולאה מופחת ערך exp ב-1, ולכן הלולאה מתבצעת exp פעמים.

```
        power *= base;
```

משפט זה מכפיל את ערך המשתנה power בערך המשתנה base. משפט זה מופיע בגוף לולאה, ולכן הוא מתבצע exp פעמים. לאחר שתסתיים הלולאה, יכיל המשתנה power את החזקה המבוקשת, באופן המוצג באיור 7.5.

```
        exp--;
```

משפט זה, המתבצע בכל איטרציה של הלולאה, מפחית את ערך המשתנה exp ב-1. זאת נעשה כדי שהלולאה תתבצע, בסך-הכל, exp פעמים.

כמו כל ביטוי, יכול ביטוי הלולאה להכיל מספר תת-ביטויים. למשל, כדי לוודא שערך שמזין המשתמש נמצא בתחום ערכים מסוים, נכתוב קטע תוכנית כזה:

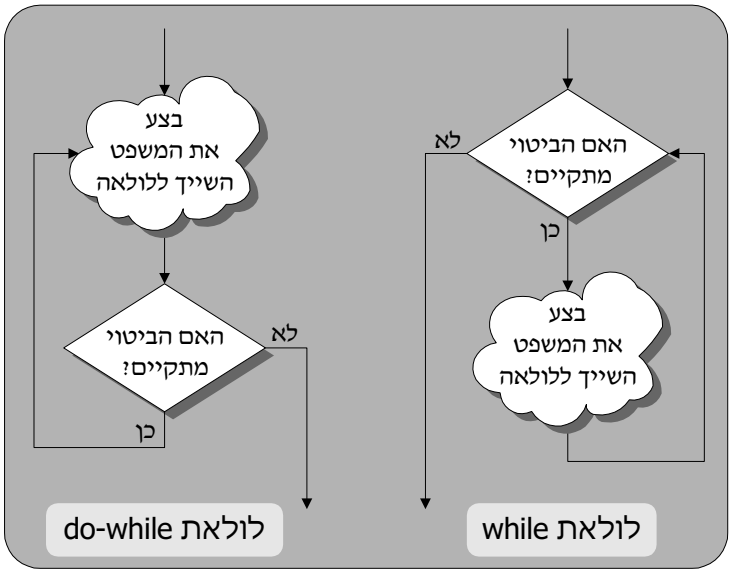
```
scanf("%d", &grade);
while (grade < 0 || grade > 100) {
    printf("Input Error. Enter grade from 0 to 100: ");
    scanf("%d", &grade);
}
```

לולאת while מתבצעת כל עוד ערך המשתנה grade אינו בין 0 ל-100, ובכך מאלצת את המשתמש להזין ערך בתחום זה.

הלולאה do-while

נכיר כעת לולאה נוספת, הדומה ללולאה while. לולאה זו נקראת **לולאת do-while**, וגם לה משתייך משפט המתבצע כל עוד ביטוי הלולאה מתקיים.

שתי הלולאות שונות בסדר בו הן בודקות את ביטוי הלולאה ומבצעות את המשפט השייך להן. לולאת while בודקת את ביטוי הלולאה לפני שהיא מבצעת את המשפט השייך לה, ואילו לולאת do-while מבצעת את המשפט השייך לה ורק אחר כך בודקת את ביטוי הלולאה. הבדל זה יובהר מהתבוננות בשני תרשימי הזרימה הבאים:



***איור 7.6** – תרשימי זרימה חלקיים המייצגים את הלולאות while ו-do-while*

שים לב!

לולאת while בודקת את ביטוי הלולאה לפני ביצוע המשפט השייך לה, ולכן ייתכן שהמשפט לא יתבצע אפילו פעם אחת. לעומתה, מבצעת לולאת do-while את המשפט השייך לה ורק אז בודקת את הביטוי, ולכן המשפט תמיד יתבצע פעם אחת לפחות. לולאת do-while עדיפה על לולאת while כאשר יש לבצע את המשפט השייך לה פעם אחת לפחות.

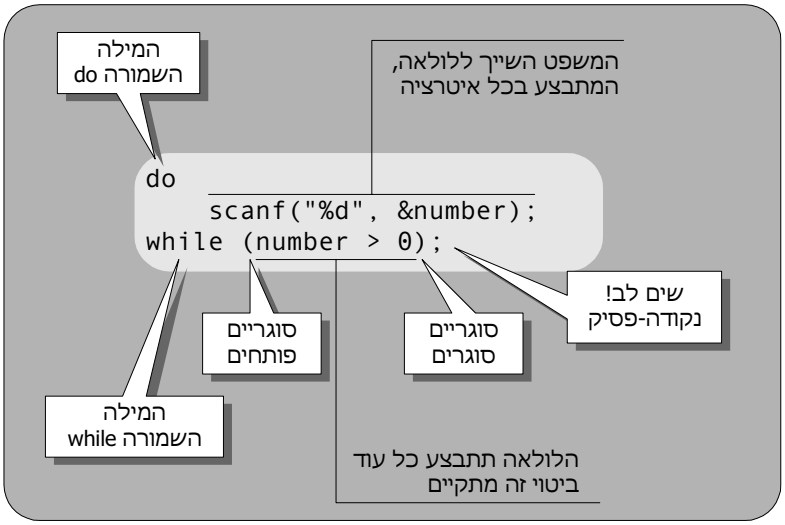
לולאת do-while פשוטה נראית כך:

```
do
    scanf("%d", &a);
while (a <= 0);
```

תחילה מבוצע משפט הקלט השייך ללולאה. אחר כך נבדק ביטוי הלולאה. אם הוא מתקיים, תתבצע הלולאה שוב, כלומר, המשפט יבוצע פעם נוספת וביטוי הלולאה ייבדק שנית. אם הוא עדיין מתקיים, תתבצע הלולאה שוב – וחוזר חלילה. כאשר לא יתקיים הביטוי, תסתיים הלולאה.

מבחינה תחבירית, בנויה לולאת do-while משלושה חלקים. החלק הראשון מורכב מהמילה השמורה do, הפותחת את הלולאה. החלק השני מורכב מהמשפט השייך ללולאה, המבוצע כל עוד ביטוי הלולאה מתקיים. החלק השלישי מורכב מהמילה השמורה while ומביטוי הלולאה, המופיע בסוגריים.

האיור הבא מציג את השימוש בלולאת do-while:



***איור 7.7** – השימוש בלולאה do-while*

זהירות!

החלק השלישי של לולאת do-while מזכיר את לולאת while, אך כאן הוא מהווה חלק מלולאת do-while ולא חלק מלולאת while. זו גם הסיבה שבסופו מופיע התו נקודה-פסיק. כאשר הוא היווה חלק מלולאת while, תו הנקודה-פסיק לא הופיע בסופו.

המשפט השייך ללולאת do-while יכול להיות גם משפט מורכב, כפי שמודגם להלן:

```
do {
    printf("Enter a positive number: ");
    scanf("%d", &num);
} while (num <= 0);
```

בלולאה זו מתבקש המשתמש להזין ערך חיובי. משפט הפלט ומשפט הקלט יתבצעו שוב ושוב, עד שיוזן ערך הגדול מ-0.

דוגמה

בדוגמה זו נפתח משחק ניחושים פשוט באמצעות לולאת do-while. על המשתמש לנחש מספר שלם שנקבע מראש, בין 1 ל-10. במהלך הלולאה נקלוט מהמשתמש את ניחושיו, ונמשיך בכך עד שינחש נכונה.

תוכנית 7.4 *GUESS.C*

```
#include <stdio.h>
void main()
{
    int num, guess = 7;

    printf("Welcome to the guessing game. \n");
    printf("A number between 1 and 10 has \n");
    printf("been chosen. Try to guess it! \n");

    printf("\nEnter your guess: \n");
    do
        scanf("%d", &num);
    while (num != guess);

    printf("Congratulations! You've found it! \n");
}
```

הנה פלט התוכנית:

```
Welcome to the guessing game.
A number between 1 and 10 has
been chosen. Try to guess it!

Enter your guess:
10
3
8
7
Congratulations! You've found it!
```

```
    int num, guess = 7;
```

שני משתנים שלמים מוגדרים בשורה זו. במשתנה num יוצב בכל פעם ניחושו של המשתמש. המשתנה guess יכיל את המספר הנבחר, שאותו צריך המשתמש לנחש.

```
    do
        scanf("%d", &num);
    while (num != guess);
```

בכל איטרציה של לולאה זו מוצב הניחוש שמזין המשתמש במשתנה num. הלולאה תתבצע כל עוד ערכי המשתנים num ו-guess שונים זה מזה, כלומר, כל עוד ניחושו של המשתמש שונה מהמספר שנבחר.

כמו כל ביטוי, גם ביטוי הלולאה do-while יכול להכיל מספר תת-ביטויים. למשל, אם ברצוננו לקלוט מהמשתמש ערכים המייצגים גילאי נהגים, עד שמוזן גיל של נהג הזכאי לביטוח רכב בתנאים מועדפים, נשתמש בלולאה כזו:

```
do {
    printf("Enter your age: ");
    scanf("%d", &age);
} while (age < 25 || age > 55);
```

לולאה זו מתבצעת כל עוד הגיל המוזן קטן מ-25 או גדול מ-55, ובכך מאלצת את המשתמש להזין גיל שנמצא בתחום המבוקש.

דוגמה

התוכנית הבאה שנכתוב תתבסס על תוכנית 7.4, אך נשפר אותה כדי להפוך את המשחק למעניין יותר. עתה נשתמש בביטוי לולאה מורכב כדי להגביל את מספר ניסיונות הניחוש. אם לא יצליח המשתמש לנחש את המספר הנבחר בחמישה ניסיונות, תסתיים הלולאה ותיפלט הודעה מתאימה.

תוכנית 7.5 *GUESS2.C*

```
#include <stdio.h>
void main()
{
    int num, guess = 7, guess_left = 5;

    printf("Welcome to the guessing game. \n");
    printf("A number between 1 and 10 has \n");
    printf("been chosen. Try to guess it! \n");
```

```
    do {
        printf("Enter your guess (%d remaining): ", guess_left);
        scanf("%d", &num);
        guess_left--;
    } while (num != guess && guess_left > 0);

    if (num == guess)
        printf("Congratulations! You've found it! \n");
    else
        printf("You've failed to find the number! \n");
}
```

פלט לדוגמה מהתוכנית ייראה כך:

```
Welcome to the guessing game.
A number between 1 and 10 has
been chosen. Try to guess it!
Enter your guess (5 remaining): 4
Enter your guess (4 remaining): 8
Enter your guess (3 remaining): 2
Enter your guess (2 remaining): 10
Enter your guess (1 remaining): 3
You've failed to find the number!
```

```
    int num, guess = 7, guess_left = 5;
```

במשתנים num ו-guess נשתמש כפי שהשתמשנו בהם בתוכנית 7.4. המשתנה guess_left, לעומת זאת, לא הופיע בתוכנית הקודמת. משתנה זה מאותחל ל-5 ולאחר כל הזנת ניחוש מופחת ערכו ב-1. באופן זה, הוא יכיל את מספר הניחושים שנותרו למשתמש.

```
    do {
        /* ... */
    } while (num != guess && guess_left > 0);
```

לולאה זו, במהלכה נקלטים הניחושים מהמשתמש, מתבצעת עד שהמשתמש ניחש נכונה את המספר הנבחר, או עד שלא נותרו לו ניסיונות ניחוש.

```
        printf("Enter your guess (%d remaining): ", guess_left);
```

משפט פלט זה מתבצע בכל איטרציה, והוא מבקש מהמשתמש להזין את ניחושו ומודיע לו כמה ניחושים נותרו לו.

```
        guess_left--;
```

כאמור, משתנה זה מייצג את מספר ניסיונות הניחוש שנותרו למשתמש. מאחר שבכל איטרציה מנחש המשתמש מספר אחד, ערכו מופחת ב-1 בסוף כל איטרציה.

```
    if (num == guess)
        printf("Congratulations! You've found it! \n");
    else
        printf("You've failed to find the number! \n");
```

הלולאה שבתוכניתנו מסתיימת בשני מקרים: או שהמשתמש ניחש נכונה את המספר הנבחר, או שהמשתמש לא הצליח לנחשו בחמישה ניסיונות. לכן, לאחר היציאה מהלולאה, עלינו לבדוק האם המספר האחרון שהזין המשתמש הוא המספר הנבחר. אם כן, אזי הצליח לנחש את המספר, ויש לפלוט הודעה מתאימה. אם לאו, אזי לא הצליח המשתמש לנחש את המספר הנבחר בחמישה ניסיונות, וגם על כך יש לפלוט הודעה מתאימה.

מטבע הדברים, ניתן לשלב משפטי תנאי בתוך המשפט המורכב השייך ללולאה. למשל, גוף הלולאה הבאה כולל משפט if-else, הבודק האם המספר שהוזן הוא זוגי או אי-זוגי ופולט הודעה מתאימה:

```
do {
    printf("Enter a number: ");
    scanf("%d", &num);
    if (num % 2 == 0)
        printf("The number is even! \n");
    else
        printf("The number is odd! \n");
} while (num != 0);
```

דוגמה

בדוגמה זו נשפר שנית את משחק הניחושים שהוצג בתוכניות 7.4 ו-7.5. עתה, לאחר כל ניחוש לא מוצלח, נודיע למשתמש אם המספר הנכון גבוה או נמוך מהמספר שניחש.

תוכנית 7.6 *GUESS3.C*

```
#include <stdio.h>
void main()
{
    int num, guess = 7, guess_left = 5;

    printf("Welcome to the guessing game. \n");
    printf("A number between 1 and 10 has \n");
    printf("been chosen. Try to guess it! \n");
```

```
    do {
        printf("Enter your guess (%d remaining): ", guess_left);
        scanf("%d", &num);
        if (num > guess)
            printf("Wrong guess... try a smaller number! \n");
        else if (num < guess)
            printf("Wrong guess... try a bigger number! \n");
        guess_left--;
    } while (num != guess && guess_left > 0);

    if (num == guess)
        printf("Congratulations! You've found it! \n");
    else
        printf("You've failed to find the number! \n");
}
```

פלט התוכנית ייראה בערך כך:

```
Welcome to the guessing game.
A number between 1 and 10 has
been chosen. Try to guess it!
Enter your guess (5 remaining): 5
Wrong guess... try a bigger number!
Enter your guess (4 remaining): 8
Wrong guess... try a smaller number!
Enter your guess (3 remaining): 7
Congratulations! You've found it!
```

```
        if (num > guess)
            printf("Wrong guess... try a smaller number! \n");
```

לאחר כל פעם שהמשתמש הזין את ניחושו, בודק משפט התנאי האם הניחוש גבוה מהמספר הנבחר. אם כן, נפלטת הודעה המייעצת למשתמש להזין מספר נמוך יותר.

```
        else if (num < guess)
            printf("Wrong guess... try a bigger number! \n");
```

אם המספר שניחש המשתמש נמוך מהמספר הנבחר, נפלטת הודעה המייעצת לו להזין מספר גבוה יותר.

דוגמה

בדוגמה הבאה נשתמש בלולאת do-while כדי לקלוט מספר ציונים מהמשתמש ולחשב את הממוצע שלהם. לשם כך נשתמש בשני משתני עזר: האחד יכיל את סכום כל הציונים שנקלטו והשני יכיל את מספר הציונים שנקלטו. באופן זה, כשנחלק את האחד בשני, נקבל את ממוצע הציונים.

כדי שהתוכנית תהיה גמישה יותר, נקבע כי אין הגבלה על מספר הציונים שניתן להזין. אם-כן, כיצד נדע מתי להפסיק את קליטת הציונים? לשם כך נקבע כי אם יוזן הערך -1 תיפסק קליטתם והממוצע שלהם ייפלט.

תוכנית 7.7

GRADES.C

```
#include <stdio.h>
void main()
{
    int grade = 0, counter = 0;
    float sum = 0;

    printf("Please enter grades, end with -1: \n");
    do {
        scanf("%d", &grade);

        if (grade != -1) {
            sum += grade;
            counter++;
        }
    } while (grade != -1);

    printf("The average of the grades is %f \n", sum / counter);
}
```

פלט אפשרי של התוכנית הוא כדלהלן:

```
Please enter grades, end with -1:
88
76
95
82
94
-1
The average of the grades is 87.000000
```

```
do {
    /* ... */
} while (grade != -1);
```

לולאה זו היא גרעין התוכנית. במהלכה נקלטים הציונים מהמשתמש וסכומם מוצב במשתנה sum. בנוסף, בכל איטרציה של הלולאה עולה ערך משתנה העזר counter ב-1, ובכך הוא מכיל את מספר הציונים שהוזנו. מאוחר יותר נשתמש בשני משתני העזר הללו כדי למצוא את ממוצע הציונים.

ביטוי הלולאה מתקיים כאשר ערך המשתנה grade אינו 1-. בכל איטרציה מוצב הערך שמזין המשתמש במשתנה grade. באופן זה, יפסיק המשתמש את תהליך קליטת הציונים על ידי הזנת הערך 1-.

```
    scanf("%d", &grade);
```

זהו המשפט הראשון בגוף הלולאה, והוא אחראי על קליטת ערך שלם מהמשתמש והצבתו במשתנה grade.

```
    if (grade != -1) {
        sum += grade;
        counter++;
    }
```

המשפטים המופיעים בגופו של משפט התנאי שלעיל יתבצעו רק אם הערך שהזין המשתמש הוא ציון. המשפט הראשון מוסיף את הערך שבמשתנה grade, כלומר, את הציון האחרון שהוזן, לערך שבמשתנה sum. באופן זה, המשתנה sum יכיל את סכום הציונים שהוזנו. המשפט השני מעלה את ערך המשתנה counter ב-1, כך שעם סיום הלולאה יכיל משתנה זה את מספר הציונים שהוזנו.

```
printf("The average of the grades is %f \n", sum / counter);
```

משפט פלט זה פולט את ממוצע הציונים שהזין המשתמש. המשתנה העשרוני sum, המכיל את סכום הציונים שהוזנו, מחולק במשתנה counter, המכיל את מספר הציונים שהוזנו – והתוצאה המתקבלת מוצגת על המסך בדיוק עשרוני (על ידי תו ההמרה f%).

במחשבה שנייה...

מה יתרחש אם לא הוזן כל ציון?

אם הערך היחיד שהזין המשתמש הוא 1-, היינו, אם לא הוזן כל ציון, ערך counter יהיה 0 כאשר נחלק בו את sum. בתוכנית זו התעלמנו מסכנת החילוק ב-0, משום שהוראת הפלט מורה למשתמש בפירוש להזין סדרת ציונים (ורק אחר כך 1-). בכל זאת, בתוכניות מורכבות יותר מומלץ לקחת בחשבון את כל המצבים ולא לסמוך על המשתמש שקרא היטב את ההוראות.

הלולאה for

הלולאה for היא הלולאה השכיחה והגמישה ביותר בשפת C. בדומה ללולאות הקודמות, גם לה משתייך משפט המבוצע מספר פעמים. נפתח בדוגמה פשוטה של לולאת for, המבצעת את משפט הפלט השייך לה שלוש פעמים:

```
for (x = 1 ; x <= 3 ; x++)
    printf("A man, a plan, a cat, a canal - Panama? \n");
```

הלולאה מביאה לפליטת המלל הבא:

```
A man, a plan, a cat, a canal - Panama?
A man, a plan, a cat, a canal - Panama?
A man, a plan, a cat, a canal - Panama?
```

לולאה זו מתאימה ללולאה האלגוריתמית הבאה:

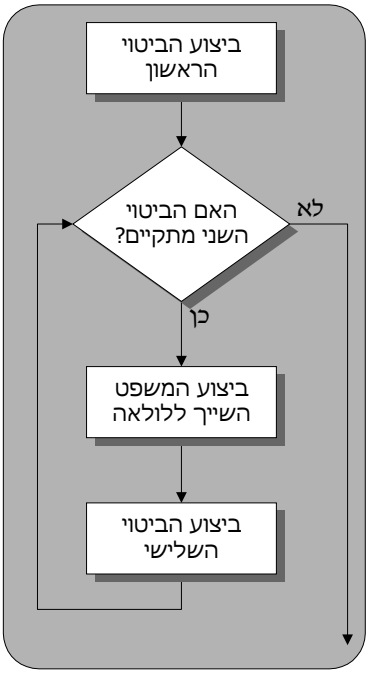
(1) עבור x מ-1 עד 3 בצע

(1.1) הדפס את ההודעה "A man, a plan, a cat, a canal - Panama?"

לולאת for נפתחת במילה השמורה for, ובסוגריים, מופרדים בתווי נקודה-פסיק זה מזה, מופיעים שלושה ביטויים. הלולאה מתיחסת אל הביטויים הללו באופן הבא:

- הביטוי השמאלי מתבצע פעם אחת בלבד, **לפני הכניסה הראשונה** ללולאה.
- הביטוי האמצעי הוא ביטוי הלולאה, שנבדק **לפני כל פעם** שמבוצע המשפט השייך ללולאה. אם ביטוי הלולאה מתקיים, מבוצע המשפט השייך ללולאה; אם לאו, הלולאה מסתיימת.
- הביטוי הימני מתבצע **לאחר כל פעם** שמבוצע המשפט השייך ללולאה.

שלבי הביצוע של לולאת for מוצגים בתרשים הזרימה הבא:



איור 7.8 *– תרשים זרימה חלקי המתאר את שלבי ביצוע לולאת for*

נבחן את הנאמר לעיל על-פי קטע התוכנית הקודם:

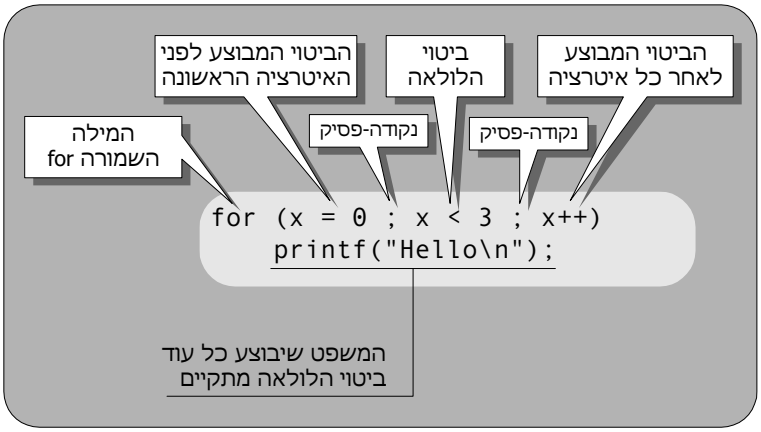
ראשון מבוצע הביטוי השמאלי, המציב את הערך 1 במשתנה x. מייד לאחר מכן נבדק הביטוי השני, ביטוי הלולאה, המתקיים כל עוד ערך המשתנה x קטן או שווה ל-3. בשלב זה, ערך המשתנה x הוא 1, ומכיון שהתנאי מתקיים, המשפט השייך ללולאה מבוצע. לאחר מכן מבוצע הביטוי השלישי, המעלה את ערך המשתנה x ב-1 (כרגע, ערכו הוא 2).

עתה נבדק שוב ביטוי הלולאה, ומכיון שהוא מתקיים, המשפט השייך ללולאה מבוצע שנית. אחר מבוצע הביטוי השלישי, המעלה את ערך המשתנה x ב-1 (כרגע, ערכו הוא 3).

עתה נבדק שוב ביטוי הלולאה, ומכיון שהוא מתקיים, המשפט השייך ללולאה מבוצע בפעם השלישית. לאחר מכן מבוצע הביטוי השלישי, המעלה את ערך המשתנה x ב-1 (כרגע, ערכו הוא 4).

עתה נבדק שוב ביטוי הלולאה, אך הוא אינו מתקיים, ולכן הלולאה מסתיימת.

האיור הבא מציג את מרכיבי לולאת for:



***איור 7.9** – השימוש בלולאה for*

נציג עתה לולאת for נוספת, המציגה את המספרים מ-1 עד 4:

```
for (i = 1 ; i <= 4 ; i++)
    printf("%d \n", i);
```

לפני ביצוע האיטרציה הראשונה, מוצב הערך 1 במשתנה i. ביטוי הלולאה מתקיים כל עוד ערך i קטן או שווה ל-4, ומאחר שערך i עולה ב-1 לאחר כל איטרציה, המשפט השייך ללולאה מתבצע ארבע פעמים. משפט זה פולט את ערך המשתנה i, ולכן, הלולאה פולטת את המלל הבא:

```
1
2
3
4
```

שים לב!

את לולאת for שלעיל ניתן להחליף בלולאת while כזו:

```
i = 1;
while (i <= 4) {
    printf("%d \n", i);
    i++;
}
```

הלולאות while ו-for דומות מאוד זו לזו, ולעיתים קשה להחליט באיזו מהן לבחור. מומלץ להשתמש בלולאת for כאשר מספר הפעמים שעליה להתבצע ידוע ואינו משתנה במהלך ביצועה.

ניתן לשייך אל לולאת for משפט מורכב, ובכך לבצע יותר ממשפט אחד מספר פעמים. למשל, כדי לקלוט מהמשתמש עשרה מספרים ולפלוט את סכומם, נבצע לולאת for כזו:

```
sum = 0.0;      /* sum is a float variable */
for (i = 0 ; i < 10 ; i++) {
    scanf("%ld", &number);
    sum += number;
}
```

את ביטויי לולאת for ניתן לשנות בהתאם לצורך. למשל, אם ברצוננו להציג את המספרים מ-100 עד 150, נבצע לולאה כזו:

```
for (i = 100 ; i <= 150 ; i++)
    printf("%d ", i);
```

לפני הכניסה הראשונה ללולאה מוצב במשתנה i הערך 100. ערכו נפלט בכל פעם שמבוצע משפט הלולאה, ומייד אחר כך הוא עולה ב-1. כאשר ערכו יהיה גבוה מ-150, לא יתקיים ביטוי הלולאה, והלולאה תסתיים.

דוגמה

בתוכנית זו נכתוב תוכנית הפולטת באמצעות לולאת for עשרה מספרים אקראיים. פעולה זו הינה מורכבת יותר משנראה במבט ראשון, שהרי ברצוננו לקבל מספר שונה בכל איטרציה של הלולאה ובכל פעם שהתוכנית רצה.

כדי לקבל מספר אקראי נשתמש בפונקציה rand, שאינה מקבלת פרמטרים אך מחזירה מספר אקראי בין 0 ל-32767. למשל, אם נקרא לפונקציה זו פעמיים, כך:

```
i = rand();
j = rand();
```

יכילו המשתנים i ו-j ערכים אקראיים (סביר שיהיו שונים זה מזה) בין 0 ל-32767. אם ברצוננו לקבל מספר אקראי בתחום מסוים, ניעזר באופרטור השארית. למשל, כדי לקבל מספר אקראי הנע בין 0 ל-9, נקרא לפונקציה rand באופן הבא:

```
i = rand() % 10;
```

פעולת השארית המבוצעת על הערך המוחזר מהפונקציה rand מבטיחה כי המספר שיוצב במשתנה i ינוע בין 0 ל-9. באופן כללי ניתן לומר כי הפעולה rand() % n, כאשר n הוא מספר טבעי כלשהו, מחזירה מספר אקראי בין 0 ל-n - 1.

אם נרצה לקבל מספר אקראי בתחום שאינו מתחיל מ-0, נוסיף ערך קבוע כלשהו לתוצאה המוחזרת מהקריאה ל-rand ומאופרטור השארית. למשל, כדי להציב במשתנה i מספר אקראי הנע בין 10 ל-19 נבצע את ההוראות הבאות:

```
i = rand() % 10;      /* assigns value between 0 and 9 */
i += 10;              /*  + 10  ==>  between 10 and 19 */
```

משפט ההשמה הראשון מציב במשתנה i ערך הנע בין 0 ל-9. משפט ההשמה השני מעלה את ערך המשתנה i ב-10, ובכך הוא נהיה בין 10 ל-19. מכיון שפעולת אופרטור השארית קודמת לפעולת אופרטור החיבור, את שני משפטים ההשמה האחרונים ניתן לקצר למשפט בודד כזה:

```
i = rand() % 10 + 10;      /* assigns value between 10 and 19 */
```

הבה נבדוק את פעולתה של הפונקציה rand על ידי שימוש בלולאת for כזו:

```
for (i = 0 ; i < 10 ; i++)
    printf("%4d", rand() % 10);
```

פלט אפשרי המתקבל מלולאה זו הוא כדלהלן:

```
2   2   4   7   2   8   3   2   4   5
```

לכאורה נראה שהלולאה פעלה בהצלחה, שהרי נפלטו ערכים אקראיים. אך אם נריץ שנית ושלישית את התוכנית בה נכתבה הלולאה, יתקבל הפלט הבא:

```
2   2   4   7   2   8   3   2   4   5
```

```
2   2   4   7   2   8   3   2   4   5
```

כלומר, בכל פעם שהתוכנית מורצת נפלטים ערכים זהים. כדי לעמוד על הסיבה לכך נתאר את פעולתה של הפונקציה rand. בפעם הראשונה שפונקציה זו נקראת, היא מחזירה את המספר הראשון מרצף אקראי כלשהו של מספרים. בפעם השנייה שהיא נקראת, היא מחזירה את המספר השני מרצף זה, וכדומה. מאחר שהרצף הינו אקראי, גם המספרים המוחזרים ממנו הם אקראיים.

הבעיה היא, שרצף המספרים אינו משתנה בכל פעם שהתוכנית רצה מחדש. לכן בכל פעם שנפעיל את התוכנית, הקריאות לפונקציה rand יחזירו את אותם המספרים. כדי לפתור בעיה זו עלינו לדאוג שרצף המספרים יורכב ממספרים אחרים בכל פעם שהתוכנית תרוץ. זאת נעשה על ידי קריאה לפונקציה נוספת, הנקראת srand. פונקציה זו מקבלת ערך כלשהו ומשנה את רצף המספרים בהתאם אליו. אך גם ערך זה צריך להשתנות בכל פעם שהתוכנית רצה, ולכן נשתמש ברכיב המחשב הבסיסי ביותר שמשתנה בכל פעם שתוכנית כלשהי רצה, והוא שעון המחשב.

הפונקציה srand תיקרא באופן הבא:

```
srand(time(0));
```

הפרמטר הנשלח אל srand, time(0), הוא למעשה ערך המוחזר מפונקציה נוספת הנקראת time. פונקציה זו מחזירה את מספר השניות שחלפו מאז הראשון בינואר, 1970. בכל פעם שתרוץ התוכנית ישתנה מספר זה, ולכן הערך שיישלח אל הפונקציה srand יהיה שונה. באופן זה ישתנה גם רצף המספרים, והמספרים שיוחזרו מהפונקציה rand אכן יהיו אקראיים.

שים לב!

הפונקציות rand ו-srand דורשות את הוספת ההוראה הבאה לתוכנית:

```
#include <stdlib.h>
```

הפונקציה time דורשת את הוספת ההוראה הבאה לתוכנית:

```
#include <time.h>
```

בכל תוכנית בה נשתמש במספרים אקראיים נוסיף את שתי השורות הללו.

תוכנית 7.8

RAND.C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main()
{
    int i;

    srand(time(0));

    printf("Random numbers between 10 and 100: \n");
    for (i = 0 ; i < 10 ; i++)
        printf("%4d", rand() % 91 + 10);
    printf("\n");
}
```

הנה מספר פלטים המתקבלים מהרצת תוכנית זו:

```
Random numbers between 10 and 100:
  43  30  96  95  50  17  49  64  56  30
```

```
Random numbers between 10 and 100:
  25  87  20  31  59  28  62  77  19  17
```

```
Random numbers between 10 and 100:
  91  75  12  49  37  20  18  50  30  60
```

```
#include <stdlib.h>
```

שורה זו גורמת למהדר להתייחס לנתונים המופיעים בקובץ stdlib.h. עלינו להוסיף שורה זו לתוכנית בשל הקריאות לפונקציות rand ו-srand.

```
#include <time.h>
```

שורה זו גורמת למהדר להתייחס לנתונים המופיעים בקובץ time.h. עלינו להוסיף שורה זו לתוכנית בשל הקריאה לפונקציה time.

```
    srand(time(0));
```

הקריאה לפונקציה time מחזירה את מספר השניות שחלפו מאז הראשון בינואר, 1970. מספר זה נשלח אל הפונקציה srand, שמשנה את רצף המספרים ממנו יילקחו המספרים האקראיים. באופן זה בכל פעם שתרוץ התוכנית תיקרא srand עם ערך אחר, ורצף המספרים יורכב ממספרים אחרים.

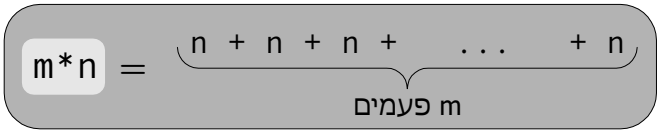
```
    for (i = 0 ; i < 10 ; i++)
        printf("%4d", rand() % 91 + 10);
```

לפני האיטרציה הראשונה של לולאה זו מוצב במשתנה i הערך 0. לאחר כל פעם שמבוצע המשפט השייך ללולאה מבוצע הביטוי השלישי של הלולאה, המעלה את ערך המשתנה i ב-1. הלולאה מתבצעת כל עוד ערך המשתנה i קטן מ-10, ולכן היא מתבצעת בסך-הכל עשר פעמים.

בכל איטרציה של הלולאה נפלט מספר אקראי אחר. הקריאה לפונקציה rand מחזירה מספר בין 0 ל-32767, אך השימוש באופרטור השארית מגביל מספר זה כך שיהיה בין 0 ל-90. השימוש באופרטור החיבור גורם למספר לנוע בין 10 ל-100, כפי שרצינו.

דוגמה

בדוגמה זו נשתמש בלולאת for כדי לבצע פעולת כפל בין שני מספרים שלמים ואי-שליליים בלי להשתמש באופרטור הכפל (*). לשם כך נקלוט שני מספרים מהמשתמש ונבחר משתנה עזר המאותחל ל-0. למשתנה העזר נוסיף את המספר השני שהוזן שוב ושוב, מספר פעמים השווה למספר הראשון שהוזן. האיור הבא מדגים זאת:



***איור 7.10** – מציאת מכפלת שני מספרים בלי להשתמש בפעולת כפל*

```
#include <stdio.h>
void main()
{
    int num1, num2, mult = 0, i;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    for (i = 1 ; i <= num1 ; i++)
        mult += num2;
    printf("%d * %d = %d \n", num1, num2, mult);
}
```

פלט לדוגמה מהתוכנית ייראה כך:

```
Enter two numbers: 4 6
4 * 6 = 24
```

```
    int num1, num2, mult = 0, i;
```

בשורה זו מוגדרים ארבעה משתנים שלמים. במשתנים num1 ו-num2 יוצבו המספרים שיזין המשתמש, ומכפלתם תוצב, בסופו של דבר, במשתנה העזר mult. המשתנה i הוא משתנה מונה וניעזר בו בלולאת for.

```
    for (i = 1 ; i <= num1 ; i++)
        /* ... */
```

לפני הכניסה הראשונה ללולאה מוצב במשתנה i הערך 1. לאחר כל איטרציה עולה ערכו ב-1. ביטוי הלולאה מתקיים כל עוד ערך המשתנה i קטן או שווה לערך המשתנה num1, ולכן הלולאה מתבצעת, בסך הכל, num1 פעמים.

במחשבה שנייה...

האם התוכנית תפעל כהלכה עבור ערכים שליליים?

אם הערך הראשון שמוזן אינו מספר חיובי תתעורר בעיה. הסיבה לכך היא, שהלולאה לא תתבצע אפילו פעם אחת (ביטוי הלולאה לא יתקיים), ולכן ערך המשתנה mult יהיה 0. למשל, אם יזין המשתמש את המספרים 2- ו-3, תוצאת מכפלתם תהא 0.

```
        mult += num2;
```

זהו המשפט האחראי על חישוב מכפלת המספרים. משפט זה, המוסיף לערך המשתנה mult את ערך המשתנה num2, מבוצע num1 פעמים, וכך, בסוף הלולאה, יכיל המשתנה mult את סכום שני המספרים.

נציג עתה מספר לולאות for שביטוייהן שונים מאלה שראינו עד כה. ראשית, ביטוי הלולאה יכול להכיל מספר תת-ביטויים, המופרדים על ידי אופרטורים לוגיים. למשל:

```
for (num = 999 ; num % 2 == 0 || num % 3 == 0 ; num++)
    printf("%d ", num);
```

כמו-כן, הביטוי השלישי לא חייב להעלות את ערך המשתנה ב-1. למשל, אם ברצוננו לפלוט את המספרים הזוגיים בין 100 ל-150, נבצע לולאה כזו:

```
for (i = 100 ; i <= 150 ; i += 2)
    printf("%d ", i);
```

בנוסף, המשתנים המופיעים בביטויי הלולאה לא חייבים להיות משתנים שלמים. למשל, כדי להציג את כל השברים בין 0 ל-3 המורכבים מרבעים, נכתוב לולאה כזו:

```
for (d = 0.0 ; d <= 3.0 ; d += 0.25)
    printf("%f \n", d);
```

למעשה, ניתן להשמיט כל אחד מביטויי לולאת for. אם נשמיט את הביטוי הראשון, לא תבצע הלולאה דבר לפני האיטרציה הראשונה שלה. אם נשמיט את הביטוי השלישי, לא תבצע הלולאה דבר לאחר כל איטרציה שלה. ניתן להשמיט גם את הביטוי השני, אך זהו מקרה מיוחד ונדון בו בהמשך.

בחן, למשל, את הלולאה הבאה, הפולטת את כל האותיות הלועזיות הקטנות:

```
for (ch = 'a' ; ch <= 'z' ; ch++)
    printf("%c ", ch);
```

יכולנו לוותר לגמרי על הביטוי הראשון ולכתוב את הלולאה באופן הבא:

```
for (          ; ch <= 'z' ; ch++)
    printf("%c ", ch);
```

הלולאה נכונה מבחינה תחבירית, אך היא אינה פועלת כהלכה. זאת משום שבלולאה שהצגנו לפניה הוצב במשתנה ch, באמצעות ביטוי for הראשון, הערך 'a' – אך עתה הוצאנו את הביטוי הראשון מהלולאה, ולכן לא קיים ערך מתאים במשתנה ch. ביצוע הלולאה כאשר ערך המשתנה ch אינו ידוע עלול להביא לפלט מפתיע ובלתי-רצוי. לכן, לפני הכניסה ללולאה יש להוסיף משפט המציב במשתנה ch את הערך 'a':

```
ch = 'a';
for (          ; ch <= 'z' ; ch++)
    printf("%c ", ch);
```

עתה תפעל הלולאה כהלכה.

עצה

השתדל להימנע מהוצאת ביטויים מהסוגריים של לולאת for. מומלץ לנצל את מלוא יכולתה של הלולאה ולבצע שלושה ביטויים הגיוניים במקומות המיועדים לכך. אם אין צורך בביצוע פעולות לפני הכניסה הראשונה ללולאה או לאחר כל פעם שמבוצע המשפט השייך ללולאה, מומלץ להשתמש בלולאת while.

ניתן לוותר גם על הביטוי השלישי, כפי שמודגם להלן:

```
for (ch = 'a' ; ch <= 'z' ;     )
    printf("%c ", ch);
```

כמו הלולאה הקודמת, גם לולאה זו נכונה מבחינה תחבירית, אך היא אינה פועלת כהלכה. זאת משום שהוצאנו את הביטוי השלישי, המעלה את ערך המשתנה ch ב-1 לאחר כל איטרציה. מאחר שערך המשתנה ch אינו משתנה, ביטוי הלולאה יתקיים תמיד, והלולאה תתבצע שוב ושוב, ללא הפסקה. כדי לתקן את הלולאה, עלינו להוסיף לגופה משפט העושה דבר זהה לזה שעשה הביטוי השלישי:

```
for (ch = 'a' ; ch <= 'z' ;     ) {
    printf("%c", ch);
    ch++;
}
```

עתה תבצע הלולאה את המוטל עליה.

ניתן לוותר גם על הביטוי הראשון וגם על הביטוי השלישי, כפי שמודגם להלן:

```
ch = 'a';
for (         ; ch <= 'z' ;     ) {
    printf("%c ", ch);
    ch++;
}
```

שים לב!

בכל לולאות for האחרונות, אף על-פי שהשמטנו את ביטוייהן, לא ויתרנו על תווי הנקודה-פסיק המופיעים בין הסוגריים. זכור כי בין הסוגריים של לולאת for תמיד יופיעו שני תווי נקודה-פסיק.

נדגים את השימוש בלולאות for שביטוייהן חסרים בתוכנית הבאה, המציגה על המסך את הספרות, האותיות הקטנות והאותיות הגדולות, עם ערכי ASCII שלהן:

תוכנית 7.10 *LETTERS.C*

```
#include <stdio.h>
void main()
{
    char ch;

    printf("\nDigits: \n");
    /* using a normal 'for' loop */
    for (ch = '0' ; ch <= '9' ; ch++)
        printf("'%c'=%d\t", ch, ch);

    printf("\nCapital letters: \n");
    /* using a non-initialized 'for' loop */
    ch = 'A';
    for (          ; ch <= 'Z' ; ch++)
        printf("'%c'=%d\t", ch, ch);

    printf("\nSmall letters: \n");
    /* using a non-incremented 'for' loop */
    for (ch = 'a' ; ch <= 'z' ;      ) {
        printf("'%c'=%d\t", ch, ch);
        ch++;
    }
}
```

פלט התוכנית מוצג להלן (פלט חלקי):

```
Digits:
'0'=48    '1'=49    '2'=50    '3'=51    '4'=52
..  ..  ..  ..
Capital letters:
'A'=65    'B'=66    'C'=67    'D'=68    'E'=69
..  ..  ..  ..
Small letters:
'a'=97    'b'=98    'c'=99    'd'=100   'e'=101
..  ..  ..  ..
```

במחשבה שנייה...

על כמה שורות יתפרס הפלט של תוכנית זו?

אף שלא השתמשנו בתו הבקרה '\n', פלט התוכנית יתפרס על מספר שורות. זאת משום שסמן הטקסט יורד אוטומטית לשורה חדשה כשהוא מגיע לסוף שורה.

```
for (ch = '0' ; ch <= '9' ; ch++)
    printf("'%c'=%d\t", ch, ch);
```

לולאה זו משתמשת במשתנה ch כדי לפלוט את עשר הספרות ואת ערכי ASCII המתאימים להן. משפט הפלט מתבצע כ-10 פעמים ובמהלכן נפלטים כל ספרה (באמצעות תו ההמרה %c) וערך ASCII המתאים לה (באמצעות תו ההמרה %d). השימוש בתו הבקרה '\t' נועד ליישור הפלט בצורה מסודרת.

```
ch = 'A';
for (          ; ch <= 'Z' ; ch++)
    printf("'%c'=%d\t", ch, ch);
```

לולאה זו משתמשת במשתנה ch כדי לפלוט את האותיות הלועזיות הגדולות ואת ערכי ASCII המתאימים להן. מאחר שהשמטנו את הביטוי הראשון של לולאה זו, הצבנו במשתנה ch ערך מתאים מחוץ לגוף הלולאה.

```
for (ch = 'a' ; ch <= 'z' ;      ) {
    printf("'%c'=%d\t", ch, ch);
    ch++;
}
```

לולאה זו משתמשת במשתנה ch כדי לפלוט את האותיות הלועזיות הקטנות ואת ערכי ASCII המתאימים להן. מאחר שהשמטנו את הביטוי השלישי של לולאה זו, הוספנו אל גופה משפט המעלה את ערך המשתנה ch ב-1.

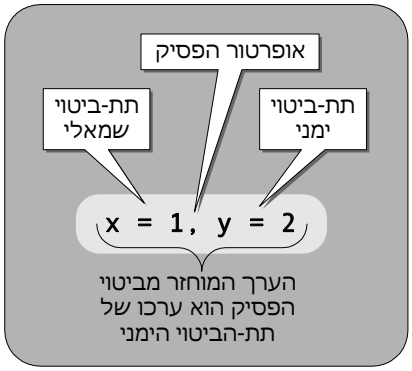
אופרטור הפסיק

לסימן הפסיק בשפת C מספר תפקידים. בכל הפסיקים שהופיעו בתוכניותינו עד כה השתמשנו כדי להפריד בין פרמטרים הנשלחים ל-printf ול-scanf. פסיקים אלה נחשבים, מבחינת המהדר, לסימני פיסוק. אך סימן הפסיק יכול להופיע במספר הקשרים אחרים. בהקשרים אלה תפקידו שונה לגמרי, והוא הופך לאופרטור.

אופרטור הפסיק (comma operator) מופיע בין שני תת-ביטויים, מבצע את השמאלי, אחר כך את הימני, ומחזיר את ערך הביטוי הימני. למשל, הביטוי הבא:

```
x = 1, y = 2
```

מורכב משני תת-ביטויים, הנמצאים בין אופרטור הפסיק. ראשית, מבוצע תת-הביטוי x = 1, המציב את הערך 1 במשתנה x. אחר כך מבוצע תת-הביטוי y = 2, המציב את הערך 2 במשתנה y. הערך המוחזר מהביטוי הוא ערך תת-הביטוי הימני, כלומר, 2. ביטוי המכיל אופרטור פסיק מכונה **ביטוי פסיק**. מרכיביו של ביטוי הפסיק שלעיל מוצגים באיור הבא:



***איור 7.11** – מרכיבי ביטוי פסיק*

במחשבה שנייה...

מהי קדימותו של אופרטור הפסיק?

אופרטור הפסיק הוא בעל הקדימות הנמוכה ביותר מכל האופרטורים בשפת C. כלומר, בהיעדר סוגריים, פעולתו תתבצע לאחר פעולת כל שאר האופרטורים שבביטוי.

באופרטור הפסיק ניתן להשתמש במספר אופנים מעניינים. בחן, למשל, את משפט התנאי הבא:

```
if (selection == 1) {
    a = 20;
    b = 30;
}
```

במשפט תנאי זה עלינו להשתמש בסוגריים מסולסלים, מאחר שברצוננו לבצע שני משפטים אם ביטוי הלולאה מתקיים. דרך נוספת לכתוב משפט זה, על ידי אופרטור הפסיק, היא כזו:

```
if (selection == 1)
    a = 20, b = 30;
```

עתה, המשפט השייך ללולאה מכיל ביטוי פסיק, המכיל את שני תת-הביטויים. תת-ביטויים אלה מבוצעים משמאל לימין, ומציבים את הערך 20 במשתנה a ואת הערך 30 במשתנה b. במקרה זה לא השתמשנו בערך המוחזר מביטוי הפסיק; כזכור, לא חובה להשתמש בערכים המוחזרים מביטויים.

ביטוי הפסיק יכול להיות גם ביטוי לולאה. למשל, אם ברצוננו לקלוט מספרים מהמשתמש ולחשב את סכומם עד שיוזן הערך 0, נבצע את שורות הקוד הבאות:

```
sum = 0;
do {
    scanf("%d", &n);
    sum += n;
} while (n != 0);
```

שורות קוד אלו ניתן להחליף בשורות הבאות:

```
sum = 0;
do
    scanf("%d", &n);
while (sum += n, n != 0);
```

עתה, ביטוי הלולאה מכיל את אופרטור הפסיק, המבצע את תת-הביטויים המופיעים משני צדיו, משמאל לימין. ראשון מבוצע תת-הביטוי sum += n, המוסיף למשתנה sum את הערך האחרון שהזין המשתמש. אחר כך מבוצע תת-הביטוי n != 0, המחזיר ערך אמת אם הערך האחרון שהזין המשתמש אינו 0. הערך המוחזר מביטוי הפסיק הוא ערך תת-הביטוי השני (n != 0), ואת הערך הזה בודקת הלולאה. כלומר, אם תת-הביטוי n != 0 מתקיים, יחזיר ביטוי הפסיק ערך אמת, והלולאה תתבצע פעם נוספת. אם תת-הביטוי אינו מתקיים, כלומר, אם המשתמש הזין את הערך 0, יחזיר ביטוי הפסיק ערך שקר, והלולאה תסתיים.

עצה

לא מקובל להשתמש באופרטור הפסיק באופנים המוצגים לעיל. גם אם הוא יכול לחסוך מספר שורות מתוכניתך, השתדל להימנע מהשימוש בו.

נניח שברצוננו לפלוט את הערך המוחזר מביטוי הפסיק הראשון שהוצג. אם נעשה זאת כך:

```
printf("result: %d", a = 20, b = 30);
```

יתקבל הפלט הבא:

```
result: 20
```

אך קודם קבענו כי ביטוי הפסיק a = 20, b = 30 מחזיר את הערך 30! הסיבה לפלט השגוי היא, שבקריאה לפונקציה, כאשר מופיע תו הפסיק בין שני ביטויים, הוא אינו נחשב לאופרטור (המבצע את הביטויים משמאל לימין ומחזיר את ערך הביטוי הימני), אלא לסימן פיסוק (השולח כל ביטוי כפרמטר נפרד לפונקציה). לכן, אל משפט הפלט הקודם נשלחו שני הביטויים, והביטוי הראשון נפלט באמצעות תו ההמרה %d. הביטוי הראשון הוא ביטוי השמה, המחזיר את הערך שהוצב במשתנה, ולכן נפלט הערך 1.

כדי להפוך את תפקיד הפסיק מסימן פיסוק לאופרטור, עלינו לתחום את ביטוי הפסיק בסוגריים, כך:

```
printf("result: %d", (a = 20, b = 30));
```

עתה יודע המהדר כי יש להתייחס אל הפסיק כאל אופרטור ולא כאל סימן פיסוק. ואכן, הפלט המתקבל מביצוע משפט פלט זה הוא ערך תת-הביטוי הימני:

```
result: 30
```

לאופרטור הפסיק זיקה מיוחדת ללולאת for. כזכור, הביטוי הראשון של לולאת for מתבצע לפני הכניסה הראשונה ללולאה. נהוג לאתחל באמצעותו את ערך המשתנה המונה. באמצעות אופרטור הפסיק, ניתן לאתחל שני משתנים בבת אחת. למשל, את לולאת for הבאה, המחשבת את עצרת המספר n:

```
fact = 1;
for (i = 2 ; i <= n ; i++)
    fact *= i;
```

ניתן לכתוב כך:

```
for (fact = 1, i = 2 ; i <= n ; i++)
    fact *= i;
```

עתה, הביטוי הראשון של לולאת for הוא ביטוי פסיק, המכיל שני תת-ביטויים. תת-ביטויים אלה מתבצעים בזה אחר זה, ולכן, לפני האיטרציה הראשונה, מוצב במשתנה fact הערך 1, ובמשתנה i – הערך 2.

ניתן להשתמש באופרטור הפסיק גם בביטוי השלישי של הלולאה for, ובכך לבצע שתי פעולות לאחר כל איטרציה. למשל, את לולאת for הקודמת ניתן לכתוב כך:

```
for (fact = 1, i = 2 ; i <= n ; fact *= i, i++) {
    /* note: empty body */
}
```

הביטוי השלישי בלולאה זו מתבצע בסופה של כל איטרציה, והוא מכיל שני תת-ביטויים. תת-ביטויים אלה מתבצעים על-פי סדר הופעתם; לכן, בסוף כל איטרציה, מבוצע הביטוי fact *= i, ואחריו, הביטוי ++i. סדר זה זהה לסדר בו התבצעו פעולות אלו בשתי הלולאות הקודמות.

במחשבה שנייה...

האם חשוב סדר הופעת הביטויים fact *= i ו-++i בביטוי השלישי של לולאת for שלעיל?

סדר זה חשוב ביותר. אילו הופיע הביטוי ++i לפני הביטוי fact *= i, היתה מחושבת מכפלת המספרים הבאה:

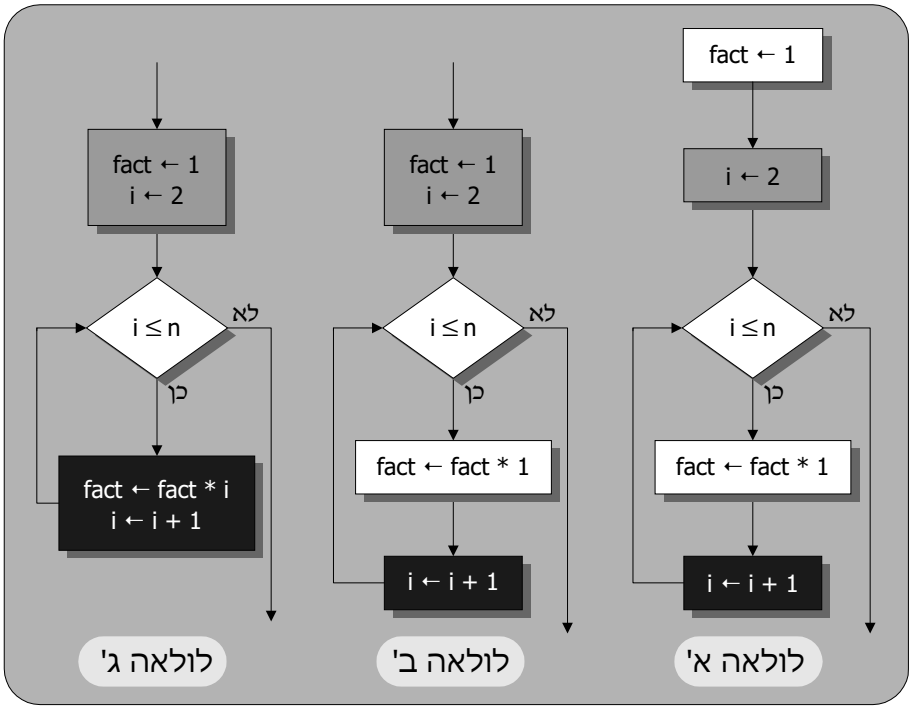
$3 * 4 * \ldots * n * (n + 1)$

במקום מכפלת המספרים הזו:

$2 * 3 * 4 * \ldots * n$

ובזאת, כמובן, לא רצינו.

האיור הבא מציג תרשימי זרימה חלקיים לשלוש לולאות for האחרונות. באיור זה, מלבן אפור-כהה מסמל את הביטוי הראשון של כל לולאה, ומלבן שחור – את הביטוי השלישי.



***איור 7.12** – לולאות for שקולות זו לזו*

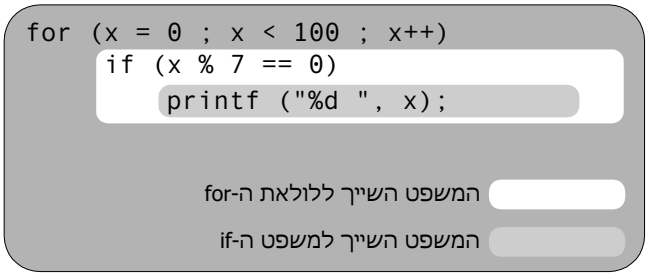
שילוב לולאות ומשפטי תנאי

כזכור, הוראת התנאי if היא משפט בודד. מכיון שלכל לולאה משתייך משפט אחד, ניתן לשייך לה משפט תנאי בלי לתחום אותו בסוגריים מסולסלים. דוגמה לכך מופיעה להלן:

```
for (x = 0 ; x < 100 ; x++)
    if (x % 7 == 0)
        printf("%d ", x);
```

המשפט השייך ללולאה הוא משפט התנאי, שלו שייך משפט הפלט. בכל איטרציה של הלולאה נבדק ביטוי התנאי, המתקיים אם ערך המשתנה x הוא אחת מכפולות המספר 7. אם ביטוי התנאי מתקיים, יתבצע משפט הפלט.

שייכות המשפטים הללו מוצגת באיור הבא:



***איור 7.13** – שיוך משפט if ללולאה*

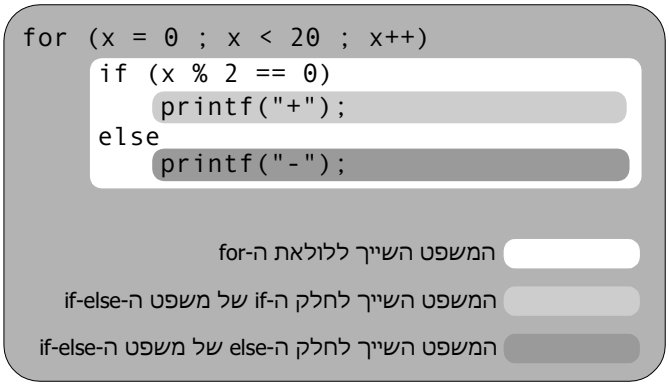
גם את הוראת התנאי if-else, הנחשבת למשפט בודד, ניתן לשייך ללולאה בלי להיעזר בסוגריים מסולסלים. למשל, אם ברצוננו להציג על המסך את תווי הפלוס והמינוס לסירוגין, נכתוב לולאה כזו:

```
for (x = 0 ; x < 20 ; x++)
    if (x % 2 == 0)
        printf("+");
    else
        printf("-");
```

הלולאה תפלוט את התווים הבאים:

```
+-+-+-+-+-+-+-+-+-+-+-+-
```

שייכות המשפטים שלעיל מוצגת באיור הבא:



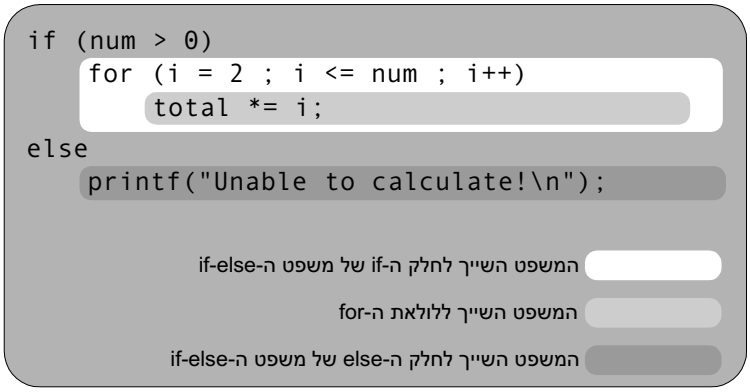
***איור 7.14** – שיוך משפט if-else ללולאה*

כל לולאה, יחד עם המשפט השייך לה, נחשבת למשפט בודד. לכן ניתן לשייך לולאה למשפט if בלי להשתמש בסוגריים מסולסלים, כפי שמודגם להלן:

```
total = 1;
scanf("%d", &num);
if (num > 0)
    for (i = 2 ; i <= num ; i++)
        total *= i;
else
    printf("Unable to calculate! \n");
```

לולאת for שלעיל מחשבת את עצרת המספר num, אותה יכיל, לאחר ביצוע הלולאה, המשתנה total. לולאה זו שייכת למשפט התנאי, ולכן היא תתבצע רק אם ביטויו יתקיים, כלומר, אם הוזן מספר חיובי. אם הוזן מספר שאינו חיובי, תיפלט הודעת שגיאה מתאימה.

שייכות המשפטים שלעיל מוצגת באיור הבא:



***איור 7.15** – שיוך לולאה למשפט תנאי*

דוגמה

בדוגמה הבאה נשייך מספר לולאות for למשפטי if-else מקוננים. התוכנית תקלוט תו מהמשתמש ותציג על המסך את כל התווים השייכים לקבוצתו (קיימות שלוש קבוצות תווים: ספרות, אותיות לועזיות קטנות ואותיות לועזיות גדולות). אם הזין המשתמש תו שאינו שייך לאחת משלוש הקבוצות, תיפלט הודעה מתאימה.

תוכנית 7.11 PRINTCHR.C

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char ch, i;

    printf("\nPlease enter a letter or a digit: ");

    ch = getch();
    putch(ch);
    printf("\n");

    if (ch >= '0' && ch <= '9')
        /* a digit was selected */
        for (i = '0' ; i <= '9' ; i++)
            printf("%c", i);
    else if (ch >= 'A' && ch <= 'Z')
        /* a capital letter was selected */
        for (i = 'A' ; i <= 'Z' ; i++)
            printf("%c", i);
    else if (ch >= 'a' && ch <= 'z')
        /* a small letter was selected */
        for (i = 'a' ; i <= 'z' ; i++)
            printf("%c", i);
    else
        printf("Invalid input! \n");
}
```

מספר פלטים אפשריים של התוכנית מופיעים להלן:

```
Please enter a letter or a digit: 4
0123456789
```

```
Please enter a letter or a digit: q
abcdefghijklmnopqrstuvwxyz
```

```
Please enter a letter or a digit: #
Invalid input!
```

```
    if (ch >= '0' && ch <= '9')
        /* a digit was selected */
        for (i = '0' ; i <= '9' ; i++)
            printf("%c", i);
```

הלולאה השייכת למשפט התנאי שלעיל תתבצע אם התו שהזין המשתמש הוא ספרה. לולאה זו מתבצעת 10 פעמים, ובכל איטרציה שלה יכיל המשתנה i ערך של ספרה אחרת (מ-'0' עד '9'). משפט הפלט השייך ללולאה מציג את התווים המתאימים לערכי המשתנה i, כלומר, את כל הספרות.

```
    else if (ch >= 'A' && ch <= 'Z')
        /* a capital letter was selected */
        for (i = 'A' ; i <= 'Z' ; i++)
            printf("%c", i);
```

הלולאה השייכת למשפט התנאי שלעיל תתבצע אם התו שהזין המשתמש הוא אות לועזית גדולה. לולאה זו מתבצעת 26 פעמים, ובכל איטרציה שלה יכיל המשתנה i את הערך של אות לועזית גדולה אחרת (מ-'A' עד 'Z'). משפט הפלט השייך ללולאה מציג את התווים המתאימים לערכי המשתנה i, כלומר, את כל האותיות הלועזיות הגדולות.

```
    else if (ch >= 'a' && ch <= 'z')
        /* a small letter was selected */
        for (i = 'a' ; i <= 'z' ; i++)
            printf("%c", i);
```

הלולאה השייכת למשפט התנאי שלעיל תתבצע אם התו שהזין המשתמש הוא אות לועזית קטנה. לולאה זו מתבצעת 26 פעמים, ובכל איטרציה שלה יכיל המשתנה i את הערך של אות לועזית קטנה אחרת (מ-'a' עד 'z'). משפט הפלט השייך ללולאה מציג את התווים המתאימים לערכי המשתנה i, כלומר, את כל האותיות הלועזיות הקטנות.

```
    else
        printf("Invalid input! \n");
```

משפט הפלט שלעיל יתבצע אם אף אחד ממשפטי התנאי הקודמים לא התקיים; היינו, אם התו שהזין המשתמש אינו ספרה או אות לועזית.

לולאות מקוננות

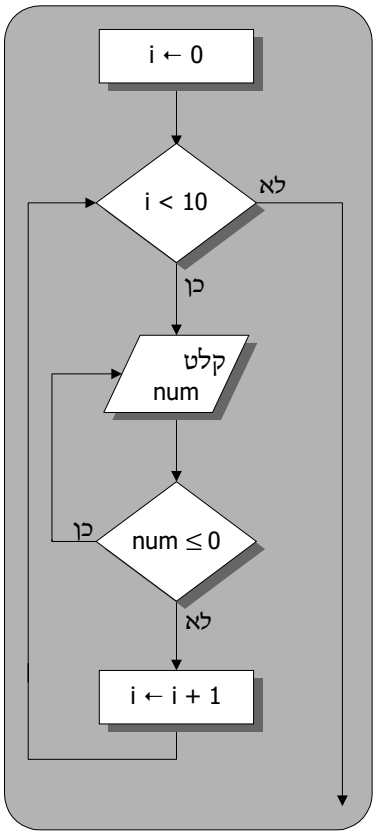
לולאה המופיעה בתוך גופה של לולאה אחרת מכונה **לולאה מקוננת**. נפתח בדוגמה פשוטה: נניח שברצוננו לקלוט מספר מהמשתמש ולוודא שהוא חיובי. לשם כך, נוח להשתמש בלולאת do-while כזו:

```
do
    scanf("%d", &num);
while (num <= 0);
```

עתה יש לקלוט עשרה מספרים מהמשתמש ולוודא שכולם חיוביים. לשם כך, יש לבצע את לולאת do-while הזו עשר פעמים. זאת נעשה על ידי שיוך שלה ללולאה נוספת, המתבצעת עשר פעמים.

```
for (i = 0 ; i < 10 ; i++) {
    do
        scanf("%d", &num);
    while (num <= 0);
}
```

באיור הבא מוצג תרשים זרימה המייצג את פעולת הלולאות הללו:



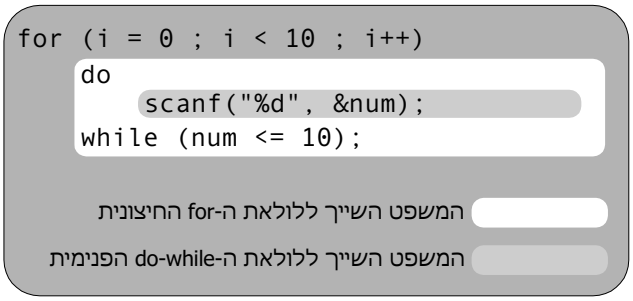
***איור 7.16** – קינון לולאת do-while בלולאת for*

כפי שניתן לראות מתרשים הזרימה, קטע התוכנית מורכב משתי לולאות: הלולאה החיצונית היא לולאת for פשוטה, המתבצעת עשר פעמים. הלולאה הפנימית היא לולאת do-while, המתבצעת שוב ושוב עד שהוזן ערך חיובי. מאחר שהלולאה הפנימית משוייכת ללולאה החיצונית, היא מתבצעת כל עוד ביטוי הלולאה של הלולאה החיצונית מתבצע – כלומר, עשר פעמים.

כזכור, כל לולאה, יחד עם המשפט השייך לה, מהווה משפט בודד. לכן, את לולאת do-while הפנימית שלעיל ניתן לשייך ללולאת for בלי להשתמש במשפט מורכב, כך:

```
for (i = 0 ; i < 10 ; i++)
    do
        scanf("%d", &num);
    while (num <= 0);
```

שייכות המשפטים שלעיל מוצגת באיור הבא:



***איור 7.17** –קינון לולאת do-while בלולאת for*

נציג דוגמה נוספת, המציגה על המסך ריבוע מלא ברוחב ובגובה של n כוכביות. למשל, אם n שווה לשלוש, יתקבל הפלט הבא:

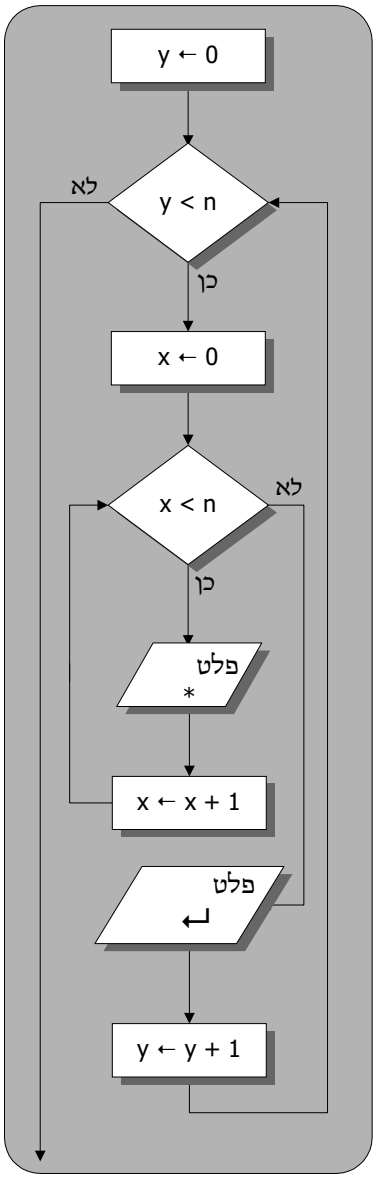
```
***
***
***
```

לשם הצגת הריבוע, עלינו להשתמש בשתי לולאות. לולאה אחת תציג, בכל איטרציה שלה, כוכבית אחת. בסך-הכל היא תתבצע n פעמים, כך שתציג, בסופו של דבר, שורת כוכביות אחת על המסך. לולאה זו תיראה כך:

```
for (x = 0 ; x < n ; x++)   /* draws one line of * */
    printf("*");
```

לולאה זו מציגה שורת כוכביות בודדת, וכדי להתאים לגובה הרצוי של הריבוע, עליה להתבצע n פעמים. לזאת תדאג הלולאה הנוספת, אליה תשויך לולאה זו. אין לשכוח לרדת לשורה חדשה לאחר הצגת כל שורת כוכביות. הלולאה הסופית המתקבלת, המוצגת גם באיור 7.18, נראית כך:

```
for (y = 0 ; y < n ; y++) {
    for (x = 0 ; x < n ; x++)
        printf("*");
    printf("\n");
}
```



איור 7.18 *– קינון לולאת for בלולאת for*

בדוגמה זו נשתמש בלולאות מקוננות כדי להציג על המסך את לוח הכפל.

תוכנית 7.12 *MULBOARD.C*

```
#include <stdio.h>
void main()
{
    int x, y;

    for (y = 1 ; y <= 10 ; y++) {
        for (x = 1 ; x <= 10 ; x++)
            printf("%d\t", y * x);
        printf("\n");
    }
}
```

פלט התוכנית (תוך צמצום רווחים) מוצג להלן:

```
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
..    ..    ..    ..    ..    ..
10  20  30  40  50  60  70  80  90  100
```

```
        for (x = 1 ; x <= 10 ; x++)
            printf("%d\t", y * x);
```

לולאה פנימית זו מתבצעת עשר פעמים, והיא אחראית על הצגת שורה בודדת של לוח הכפל. בכל איטרציה שלה נפלטת מכפלת ערכי המשתנים x ו-y, ומתבצע מעבר לטאב האופקי הבא, על ידי פליטת תו הבקרה '\t'.

```
    for (y = 1 ; y <= 10 ; y++) {
        /* ... */
```

זו הלולאה החיצונית, המתבצעת אף-היא עשר פעמים. בכל איטרציה שלה מבוצע משפט הפלט השייך ללולאה הפנימית עשר פעמים, כאשר ערך המשתנה x עולה מ-1 ל-10 בהפרשים של 1. כלומר, עבור כל ערך של y, משפט פלט זה מתבצע עשר פעמים, כשערכי המשתנה x הם 1, 2, 3, ... , 10. מאחר שמשפט הפלט פולט את מכפלת ערכי המשתנים x ו-y, בכל איטרציה של הלולאה החיצונית מוצגות עשר הכפולות החיוביות הראשונות של המספר שבמשתנה y. הלולאה החיצונית מתבצעת כשערך המשתנה y עולה מ-1 ל-10 בהפרשים של 1, ולכן מוצגות כל הכפולות של המספרים מ-1 עד 10 – וזהו לוח הכפל.

```
        printf("\n");
```

משפט פלט זה מבוצע בסוף כל איטרציה של הלולאה החיצונית, והוא מוריד את סמן הטקסט לשורה חדשה לאחר כל פעם שמוצגת שורה בודדת של לוח הכפל.

דוגמה

המספרים הפלינדרומיים (palindromic numbers) הם מספרים שריתקו מתמטיקאים במשך זמן רב. מספר פלינדרומי הוא מספר שנקרא מההתחלה לסוף באותו אופן שהוא נקרא מהסוף להתחלה. דוגמאות למספרים פלינדרומים הן 9, 4884, 15851 ו-8813200023188.

בדוגמה זו נשתמש בלולאות מקוננות כדי להציג את המספרים הפלינדרומיים הראשונים. נעשה זאת בעזרת אלגוריתם פשוט, המאפשר ליצור מספר פלינדרומי מכמעט כל מספר טבעי num. על-פי האלגוריתם, אם ברצוננו ליצור מספר פלינדרומי ממספר num שאינו פלינדרומי, יש להוסיף ל-num את המספר ההפוך לו בסדר ספרותיו, ולחזור על פעולה זו עד שיתקבל מספר פלינדרומי.

נדגים את פעולת האלגוריתם על המספר הלא-פלינדרומי 84. נוסיף לו את המספר ההפוך לו (48), ונקבל 132. גם 132 אינו מספר פלינדרומי, ולכן נמשיך ונוסיף לו את המספר ההפוך לו (231). המספר המתקבל הוא 363, והוא מספר פלינדרומי.

נדגים את פעולת האלגוריתם על מספר נוסף, 195. נוסיף לו את המספר ההפוך לו (591), ונקבל את המספר 786. נוסיף ל-786 את המספר ההפוך לו (687), ונקבל את המספר 1473. נוסיף ל-1473 את המספר ההפוך לו (3741), ונקבל את המספר 5214. נוסיף ל-5214 את המספר ההפוך לו (4125), ונקבל את המספר 9339 – והוא המספר הפלינדרומי המתאים למספר 195.

במחשבה שנייה...

האם אלגוריתם זה מתאים לכל המספרים הטבעיים?

למעשה, אין תשובה לשאלה זו. עדיין לא הוכח שניתן ליצור מספרים פלינדרומיים מכל המספרים הטבעיים. למשל, חוקרים ניסו להפוך את המספר 196 למספר פלינדרומי באמצעות אלגוריתם זה, ונכשלו גם לאחר שביצעו 9.48 מיליון (!) שלבים כגון אלה המתוארים לעיל.

```
#include <stdio.h>
void main()
{
    int i, num, rev, s;

    /* rev is the palindromic number of num
     * s is a temporary variable
     */

    printf("The palindromic numbers that match \n
        the first 78 natural numbers are: \n");

    for (i = 1 ; i <= 78 ; i++) {
        num = i;

        rev = 0;
        do {
            num += rev;

            rev = 0;
            for (s = num ; s > 0 ; s /= 10)
                rev = rev * 10 + s % 10;
        } while (num != rev);

        printf("#%d --> %d \t", i, num);
    }
    printf("\n");
}
```

פלט חלקי של התוכנית מוצג להלן (הרווחים שלהלן אינם זהים לרווחי הפלט שיתקבל תחת מחשבך):

```
The palindromic numbers that match
the first 78 natural numbers are:
#1 --> 1      #2 -->  2     #3 -->  3     #4 -->  4
#5 -->  5     #6 --> 6      #7 --> 7      #8 --> 8
#9 -->  9     #10 --> 11    #11 --> 11    #12 --> 33
 ..     ..     ..     ..     ..     ..     ..     ..
#73 -->  121 #74 --> 121   #75 --> 363   #76 --> 484
#77 -->  77   #78 --> 4884
```

שים לב!

בחרנו לפלוט דווקא 78 מספרים פלינדרומיים משום שהמספר הפלינדרומי המתאים לערך 79 הוא 44044, ותחת רוב מערכות ההפעלה ערך זה חורג מטווח הערכים שניתן להציב במשתנה מסוג int.

```
    int i, num, rev, s;
```

בשורה זו מוגדרים המשתנים בהם נשתמש בתוכניתנו. המשתנה i הוא משתנה מונה של הלולאה החיצונית, המתבצעת 78 פעמים ומציגה את המספרים הפלינדרומים. בכל איטרציה שלה יקבל המשתנה num את הערך של i, ולאחר ביצוע לולאת do-while הפנימית הוא יכיל את המספר הפלינדרומי המתאים ל-i. מספר זה יאוחסן גם במשתנה rev, כפי שנראה בהמשך. המשתנה s הוא משתנה זמני, שיכיל את ערך num במהלך חישוב המספר הפלינדרומי.

```
    for (i = 1 ; i <= 78 ; i++) {
        /* ... */
```

זו הלולאה החיצונית, המתבצעת 78 פעמים. בסוף כל איטרציה שלה ייפלט המספר הפלינדרומי המתאים ל-i.

```
        do {
            /* ... */
        } while (num != rev);
```

לולאה פנימית זו מתבצעת עד שנמצא המספר הפלינדרומי המתאים למשתנה i. נרחיב אודות גופה ואודות ביטוי הלולאה שלה בהמשך.

```
            num += rev;
```

משפט זה מוסיף למספר שבמשתנה num את המספר ההפוך לו, שנמצא במשתנה rev. על-פי האלגוריתם, לאחר שפעולה זו מתבצעת, יש לבדוק האם num הוא מספר פלינדרומי. זאת ייעשה בהמשך.

כאשר num הוא מספר פלינדרומי, אין להוסיף לו את המספר ההפוך לו. ייתכן מצב בו num יהיה מספר פלינדרומי לפני הכניסה ללולאת do-while (זה יקרה כאשר i הוא מספר פלינדרומי), ולכן, עבור כל ערך של i, בפעם הראשונה שמשפט זה מבוצע, ערך rev הוא 0. כך ייבדק בהמשך האם num הוא מספר פלינדרומי, בלי שהוסף אליו המספר ההפוך לו בסדר ספרותיו.

```
rev = 0;
for (s = num ; s > 0 ; s /= 10)
    rev = rev * 10 + s % 10;
```

שורות אלו מציבות ב-rev מספר ההפוך בסדר ספרותיו למספר שבמשתנה s (הזהה למספר שבמשתנה num). הדבר נעשה במספר שלבים.

הספרה הימנית ביותר של המספר שבמשתנה s מתקבלת על ידי חישוב שאריתו ב-10. ערך s מחולק ב-10 לאחר כל איטרציה, ובכך מתבטלת ספרתו האחרונה. לכן, אם בכל איטרציה נחשב את השארית של s ב-10, נקבל את ספרותיו של המספר num מימין לשמאל, וזהו רצף ספרות היוצר מספר הפוך למספר num.

המשפט השייך ללולאת for מוצא את הספרה האחרונה של המספר שבמשתנה s, ומוסיף אותה מימין לספרות המספר שבמשתנה rev. לכן, לאחר ביצוע הלולאה, יכיל המשתנה rev את המספר ההפוך בסדר ספרותיו למספר שבמשתנה num.

הטבלה הבאה מציגה את האופן בו מקבל המשתנה rev את המספר ההפוך למספר שבמשתנה s, אם הוא 2836:

ערכו של s לפני הכניסה ללולאה:	ערכו של rev לפני הכניסה ללולאה:
2 8 3 6	0

מספר איטרציה	הספרה s % 10	ערכו של rev לאחר הכפלתו ב-10	ערכו של rev לאחר הוספת הספרה s % 10	ערכו של s לאחר חלוקתו ב-10
1	6	0	6	2 8 3
2	3	6 0	6 3	2 8
3	8	6 3 0	6 3 8	2
4	2	6 3 8 0	6 3 8 2	0

ערכו של s בתום הלולאה:	ערכו של rev בתום הלולאה:
0	6 3 8 2

```
        } while (num != rev);
```

לאחר ביצועה של לולאת for הפנימית, הוצב במשתנה rev המספר ההפוך בספרותיו למספר שבמשתנה num. אם שני המספרים הללו זהים, אזי לפנינו מספר פלינדרומי, והלולאה הפנימית מסתיימת.

```
        printf("#%d –> %d \t", i, num);
```

משפט זה פולט את ערך המשתנה i ואת המספר הפלינדרומי המתאים לו.

לולאות אינסופיות

כאשר ביטוי הלולאה של לולאה מתקיים תמיד, המשפט השייך לה יבוצע אינסוף פעמים. במקרה זה, נכנה את הלולאה **לולאה אינסופית** (infinite loop). למשל, לולאת while הבאה היא לולאה אינסופית:

```
while (1 != 0)  /* this expression is always TRUE */
    printf("This will fill up your screen! \n");
```

עצה

ברוב המקרים ניתן להפסיק משפט המתבצע ללא הפסקה (כגון משפט הפלט שלעיל) באמצעות לחיצה בו-זמנית על המקשים Ctrl ו-Break.

גם לולאת do-while יכולה להיות אינסופית, כפי שמוצג להלן:

```
do {
    /* ... */
} while (1 == 1);
```

לרוב, כניסה ללולאה אינסופית תגרום לתקיעת התוכנית או לקריסתה. לכן, היזהר בכתיבת ביטויי הלולאות וודא שאינם מתקיימים תמיד. במקרים שלעיל הטעות ברורה לעין, אך בחן את הלולאה הבאה:

```
for (i = 1 ; i <= 10 ; i--)
    printf("%d ", i);
```

המתכנת התכוון לפלוט את עשרת המספרים הטבעיים הראשונים, אך במקום להעלות את ערך המשתנה המונה בסוף כל איטרציה, הוא הפחית אותו. ביטוי הלולאה מתקיים תמיד, שכן 1 קטן מ-10, וכל שכן מספר הקטן מ-1. לכן משפט הפלט יתבצע ללא הפסקה, עד שתסתיים התוכנית באופן מלאכותי.

מקרה נוסף בו לולאת for תתבצע אינסוף פעמים הוא כאשר לא מצוי ביטוי שני בלולאה. למשל, בלולאה הבאה:

```
for (i = 0 ;       ; i++)
    printf("Here we go again... \n");
```

חסר ביטוי הלולאה, ולכן משפט הפלט יבוצע אינסוף פעמים.

שים לב!

לולאת for היא הלולאה היחידה שניתן להשמיט את ביטוי הלולאה שלה. הלולאה הבאה, למשל, אינה נכונה:

```
while (    )      /* wrong, expression needed */
    /* ... */
```

משפטי בקרה נוספים

בשפת C קיימים שלושה משפטי בקרה פשוטים, המאפשרים לדלג מיידית אל משפטים אחרים בתוכנית. לרוב, יבוצעו משפטים אלה מתוך משפטי תנאי, כדי לקטוע את רצף התוכנית הסדיר.

עצה

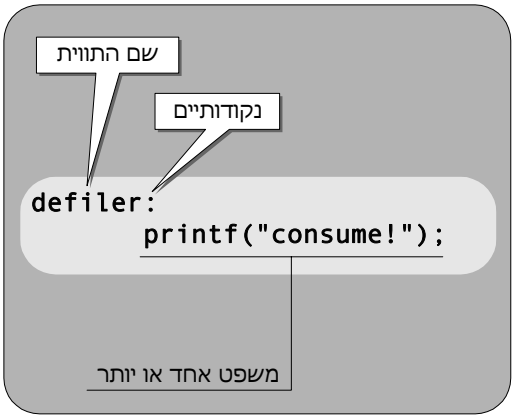
משפטי הבקרה שנכיר מייד מקשים על הבנת התוכנית, ומומלץ להימנע מהשימוש בהם ככל הניתן. למעשה, יש האוסרים כליל את השימוש בהם.

המשפט goto

המשפט goto מאפשר לדלג אל משפטים אחרים בתוכניתנו. לפני השימוש בו, עלינו להגדיר **תווית** (label). תווית היא מזהה הנכתב במקום מסוים בתוכנית, וניתן לעבור למקום זה כאשר נחפוץ.

שם התווית יכול להכיל אותיות לועזיות, ספרות וקו תחתי, אך אין להתחילו בספרה. לאחר שם התווית מופיעות נקודתיים, ואחריהן משפט אחד או יותר.

האיור הבא מציג את מרכיבי התווית:



***איור 7.19** – מרכיבי תווית*

הנה דוגמה למספר תוויות חוקיות:

```
YesOrNo:
    printf("Yes or no? ");


get_3_numbers:
    printf("Please enter 3 numbers: ");
    scanf("%d %d %d", &x, &y, &z);
```

באפשרותנו להגדיר מספר תוויות בתוכנית, כדי שנוכל לדלג אל משפטים שונים. במקרה זה, יש להעניק לכל תווית שם שונה. ניתן להגדיר תווית בשם הזהה לשם של משתנה, אך מומלץ להימנע מכך, כדי למנוע בלבול-יתר. אין להגדיר תווית בשם של מילה שמורה.

כדי לדלג אל תווית מסוימת, נבצע משפט goto. כשנעשה זאת, יבוצעו כל המשפטים המופיעים לאחר התווית. לאחר הדילוג, לא נוכל לחזור אל המשפט ממנו דילגנו.

משפט goto מורכב מהמילה השמורה goto ומשם התווית אליה ברצוננו לדלג. כמו כל משפט, הוא מסתיים בנקודה-פסיק. למשל, משפט goto המדלג אל התווית get_3_numbers נראה כך:

```
goto get_3_numbers;
```

שים לב!

התווית אליה אנו מדלגים יכולה להופיע לפני או אחרי משפט goto המדלג אליה.

נציג דוגמה פשוטה לשימוש ב-goto. אם נבצע את קטע התוכנית הבא:

```
1   printf("Hello there. \n");
2   goto finish;
3   printf("Will you see this? \n");
4
5   finish:
6       printf("finished! \n");
7       printf("... \n");
8       printf("Are you still here? \n");
```

יתקבל הפלט הבא:

```
Hello there.
finished!
...
Are you still here?
```

בתחילה, מבוצע משפט הפלט המופיע בשורה (1). המשפט הבא אחריו הוא משפט goto, המביא לדילוג אל התווית הנקראת finish. תווית זו נמצאת בשורה (5), והמשפטים הבאים אחריה מבוצעים, בזה אחר זה. משפט הפלט המופיע בשורה (3) לא יתבצע לעולם.

ניתן לכתוב מספר תוויות בזו אחר זו, כפי שמוצג בקטע התוכנית הבא:

```
print1:
    printf("dragon ");
print2:
print3:
    printf("slay");
    printf("er");
```

אם נדלג אל התווית print1, ייפלט המלל הבא:

```
dragon slayer
```

ואם נדלג אל התווית print2 או print3, ייפלט המלל הבא:

```
slayer
```

משפט goto נחשב ל"כבשה השחורה" של שפת C, והשימוש בו מעיד על סגנון תכנותי גרוע (לכל היותר). קשה להבין תוכנית הכוללת משפטי goto רבים, ולכן חשוב להימנע מהם בכל מחיר.

במחשבה שנייה...

האם קיימים מקרים בהם השימוש ב-goto הכרחי?

התשובה הינה חד-משמעית – לא. ניתן לוותר על כל משפטי goto בכל תוכנית על ידי שימוש נכון בלולאות (ובפונקציות, אותן נכיר בהמשך). אם הגעת למצב בו אינך מוצא דרך להימנע ממשפט goto, יש להניח שתוכניתך אינה כתובה כהלכה.

יש לציין, עם זאת, מקרה בודד בו השימוש ב-goto יכול להקל במעט על כתיבת התוכנית. מדובר ביציאה מלולאות מקוננות רבות, למשל:

```
for ( ... ) {
    for ( ... ) {
        while( ... ) {
            if ( ... )
                goto end;      /* simple jump */
        }
    }
}
/* ... */
end:
    /* ... */
```

המשפט break

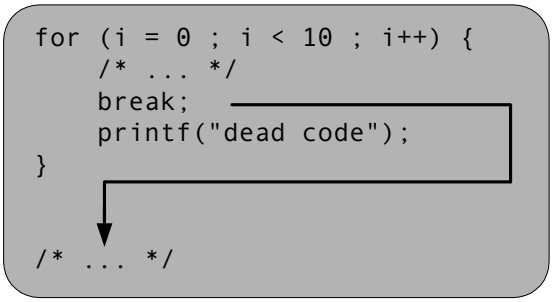
במשפט break נתקלנו לראשונה כאשר דנו במשפט switch. ראינו כי כאשר משפט break מבוצע בתוך גופו של משפט switch, משפט switch מסתיים. באופן דומה, על ידי משפט break ניתן לסיים גם לולאות. אם נבצעו בתוך גופה של לולאה, היא תסתיים מייד – גם אם ביטוי הלולאה שלה מתקיים. כדי לבצע משפט זה, יש לרשום את המילה השמורה break, ואחריה – נקודה-פסיק.

איך לא...

משפט break יכול להופיע רק בתוך לולאה או בתוך משפט switch. לכן, קטע התוכנית הבא אינו נכון מבחינה תחבירית:

```
if (a > b) {
    /* ... */
    if (a > c)
        break;  /* wrong, break isn't allowed here
*/
}
```

האיור הבא מציג את פעולת המשפט break:



*איור **7.20** – פעולת המשפט break*

במחשבה שנייה...

מהי פעולתו של משפט הפלט באיור 7.20?

משפט זה לא יתבצע כלל. הסיבה היא, שמשפט break יוצא מיידית מהלולאה, ולכן המשפטים המופיעים אחריו בגוף הלולאה לעולם לא יתבצעו. מהדרים מסוימים יזהירו אותך ממקרה כזה.

קטע התוכנית הבא מדגים שימוש במשפט break:

```
1   int a, b;
2   printf("Enter pairs of numbers or 0 to quit: \n");
3   for ( ; ; ) {
4       scanf("%d %d", &a, &b);
5       if (a == 0 || b == 0)
6           break;      /* jump to line 9 */
7       printf("result: %d", a / b);
8   }
9   printf("I hope you've learned some math!");
```

הנה דוגמה לפלט המתקבל מקטע תוכנית זה:

```
Enter pairs of numbers or 0 to quit:
15 3
result: 5
20 10
result: 2
0 0
I hope you've learned some math!
```

הגרעין של קטע תוכנית זה הוא הלולאה האינסופית. במהלך לולאה זו, נקלטים שני מספרים שלמים מהמשתמש ונפלטת מנתם. משפט התנאי המופיע בשורה (5) מבצע את משפט break אם אחד מהמספרים שהוזנו הוא 0. משפט break מסיים את הלולאה, ומשפט הפלט שבשורה (9) מתבצע. באופן זה מתבצעות קליטת המספרים ופליטת מנתם שוב ושוב, עד להזנת הערך 0.

עצה

יכולנו לכתוב את קטע התוכנית הקודם במספר דרכים בלי להשתמש במשפט break. מומלץ להימנע משימוש במשפט זה ככל הניתן.

כאשר מבצעים את משפט break בתוך לולאות מקוננות, תסתיים הלולאה הפנימית ביותר בה הוא נמצא. נדגים את הדבר בדוגמה הבאה.

דוגמה

בדוגמה זו נשתמש במשפט break בלולאות מקוננות. התוכנית שנכתוב תקלוט מהמשתמש מספר ותבקש ממנו להזין את שורשו הריבועי. אם ישגה המשתמש בתשובתו, הוא יתבקש להזין שוב את שורש המספר, וחוזר חלילה – עד שתשובתו תהיה נכונה. קליטת המספרים שאת שורשם הריבועי על המשתמש לחשב תתבצע שוב ושוב, עד שיוזן מספר שלילי.

תוכנית 7.14 *BREAK.C*

```
#include <stdio.h>
#include <math.h>
void main()
{
    int num, result;

    for ( ; ; ) {
        printf("Please enter a number (negative number to quit): ");
        scanf("%d", &num);
        if (num < 0)
            break;
        printf("\tSquare root of %d is: ", num);
```

```
        for ( ; ; ) {
            scanf("%d", &result);
            if (result == sqrt(num))
                break;
            printf("\tWrong; try again: ");
        }
        printf("\tCorrect! \n\n");
    }
}
```

הנה פלט לדוגמה מהתוכנית:

```
Please enter a number (negative number to quit): 9
        Square root of 9 is: 3
        Correct!

Please enter a number (negative number to quit): 25
        Square root of 25 is: 10
        Wrong; try again: 50
        Wrong; try again: 5
        Correct!

Please enter a number (negative number to quit): -1
```

```
#include <math.h>
```

הוראה זו גורמת למהדר להתייחס אל נתונים המופיעים בקובץ math.h. הוראה זו הכרחית משום שתוכניתנו קוראת לפונקציה sqrt.

```
    int num, result;
```

בשני המשתנים השלמים הללו נשתמש כדי לקלוט ערכים מהמשתמש. במשתנה num יוצב המספר שאת שורשו הריבועי יצטרך המשתמש לחשב. המספר שהמשתמש חושב שהוא שורשו הריבועי של המספר num יוצב במשתנה result.

```
    for ( ; ; ) {
        printf("Please enter a number (negative number to
            quit): ");
        scanf("%d", &num);
        if (num < 0)
            break;
        /* ... */
```

בלולאה אינסופית זו נקלט המספר שאת שורשו הריבועי יצטרך המשתמש לחשב. אם יוזן מספר שלילי, תסתיים הלולאה באמצעות משפט break והתוכנית תסתיים. אם המספר שהוזן אינו שלילי, יתבצעו המשפטים הבאים שבגוף הלולאה.

```
        for ( ; ; ) {
            scanf("%d", &result);
            if (result == sqrt(num))
                break;
            printf("\tWrong; try again: ");
        }
```

בלולאה אינסופית זו מזין המשתמש את המספר ששווה, לדעתו, לשורשו הריבועי של המספר num. אם המשתמש צדק, יבוצע משפט break ולולאה פנימית זו תסתיים. אם הוא טעה, יבוצע משפט הפלט, המודיע למשתמש כי תשובתו אינה נכונה, ותתבצע איטרציה נוספת של לולאה זו. באופן זה ייקלטו מספרים מהמשתמש שוב ושוב, עד שיוזן ערך נכון.

```
        printf("\tCorrect! \n\n");
```

משפט זה מבוצע לאחר שהסתיימה הלולאה האינסופית הפנימית, כלומר, לאחר שהזין המשתמש תשובה נכונה. במקרה זה, תוצג הודעה מתאימה, ואיטרציה נוספת של הלולאה החיצונית תתבצע. באיטרציה זו ייקלט מספר נוסף מהמשתמש, ואם הוא אינו שלילי, המשתמש יתבקש להזין את שורשו הריבועי – וחוזר חלילה.

המשפט continue

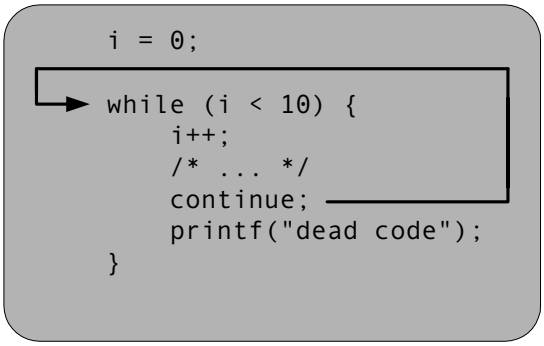
ניתן לראות במשפט continue קרוב משפחה רחוק של משפט break. בעוד שמשפט break מביא לסיום הלולאה, משפט continue מביא לסיום האיטרציה הנוכחית שלה. לאחר מכן תתבצע הלולאה כרגיל: ביטוי הלולאה ייבדק, ואם הוא יתקיים, תבוצע איטרציה נוספת של הלולאה – וחוזר חלילה. כדי לבצע משפט זה, יש לכתוב את המילה השמורה continue, ואחריה – נקודה-פסיק.

איך לא...

משפט continue יכול להופיע רק בתוך לולאה. לכן, קטע התוכנית הבא אינו נכון מבחינה תחבירית:

```
if (i == j) {
    /* ... */
    if (j == m)
        continue;      /* wrong, continue is
                          not allowed here   */
}
```

הפעולה של משפט continue מוצגת באיור הבא:



איור 7.21 – פעולת המשפט continue

במחשבה שנייה...

מהי פעולתו של משפט הפלט באיור 7.21?

משפט זה לא יתבצע כלל. הסיבה היא, שמשפט continue מסיים מיידית את האיטרציה הנוכחית של הלולאה, ולכן המשפטים המופיעים אחריו בגוף הלולאה לעולם לא יתבצעו. מהדרים מסוימים יזהירו אותך ממקרה כזה.

הנה דוגמה פשוטה לשימוש במשפט continue:

```
1   i = 0;
2   while (i < 10) {
3       printf("%d ", i);
4       i++;
5       continue;      /* jump to line 2 */
6       printf("Hello! \n");    /* dead code */
7   }
```

הפלט המתקבל מקטע תוכנית זה הוא כדלהלן:

```
0 1 2 3 4 5 6 7 8 9
```

שני המשפטים הראשונים בגופה של הלולאה פולטים את ערך המשתנה i ומעלים אותו ב-1. לאחר ביצועם מתבצע משפט continue, המפסיק את האיטרציה הנוכחית של הלולאה ומתחיל את האיטרציה הבאה שלה (החל משורה (2)). משפט זה מתבצע בכל איטרציה, ולכן משפט הפלט שבשורה (6) לא יתבצע לעולם.

הנה דוגמה מעט מורכבת יותר לשימוש במשפט continue, הקולטת מהמשתמש עשרה מספרים ומחשבת את סכומם של החיוביים שבהם:

```
1   i = 0;
2   sum = 0;
3   while (i < 10) {
4       i++;
5       scanf("%d", &num);
6       if (num <= 0)
7           continue;      /* jump to line 3 */
8       sum += num;   /* this statement executes only if num > 0
*/
9   }
```

משפט התנאי שבשורה (6) מבצע את משפט continue אם המספר האחרון שהוזן אינו חיובי. כאשר יבוצע משפט continue, תיפסק האיטרציה הנוכחית של הלולאה ותתחיל האיטרציה הבאה שלה: ביטוי הלולאה שבשורה (3) ייבדק; אם הוא יתקיים, ייקלט מספר נוסף מהמשתמש, ואם לא, תסתיים הלולאה. משפט ההשמה שבשורה (7) יתבצע רק אם משפט continue לא התבצע, כלומר, אם הוזן ערך חיובי.

כאשר יבוצע משפט continue מתוך לולאת for, תיפסק האיטרציה הנוכחית שלה, ויבוצע הביטוי השלישי של הלולאה. אחר כך ייבדק ביטוי הלולאה, ואם הוא יתקיים, תבוצע הלולאה פעם נוספת. נדגים זאת על ידי לולאת for פשוטה, הפולטת את כל המספרים האי-זוגיים בין 1 ל-20:

```
for (i = 1 ; i <= 20 ; i++) {
    if (i % 2 == 0)
        continue; /* jump to the 3rd expression of the for statement */
    printf("%d ", i); /* this statement executes only if i is odd */
}
```

אם ערך המשתנה i זוגי, יבוצע משפט continue. עקב כך, תסתיים האיטרציה הנוכחית של הלולאה, הביטוי השלישי שלה יתבצע, ביטוי הלולאה ייבדק, ואם הוא יתקיים, תבוצע הלולאה פעם נוספת. משפט הפלט יתבצע רק אם משפט continue לא התבצע, כלומר, אם ערך i הוא אי-זוגי.

כאשר מבצעים את משפט continue בתוך לולאות מקוננות, תסתיים האיטרציה של הלולאה הפנימית ביותר בה הוא נמצא. נדגים את הדבר בדוגמה הבאה.

דוגמה

בדוגמה זו נשתמש במשפט continue בלולאות מקוננות. התוכנית שנכתוב תפלוט את כל המספרים הפריקים (כלומר, שאינם ראשוניים) מ-2 עד 20. מספר פריק הוא מספר שמתחלק ללא שארית במספר אחד לפחות שאינו 1 או הוא עצמו.

במחשבה שנייה...

האם המספר 1 הוא מספר ראשוני או פריק?

הדעות חלוקות בנוגע לפריקותו של המספר 1; יש המגדירים אותו כמספר ראשוני ויש המגדירים אותו כפריק.

תוכנית 7.15

CONTINUE.C

```
#include <stdio.h>
#include <math.h>
void main()
{
    int num, div;

    printf("The reducible numbers between 2 and 20 are: \n");

    for (num = 2 ; num <= 20 ; num++)
        for (div = 2 ; div <= sqrt(num) ; div++) {
            if (num % div != 0)
                continue;
            printf("%d ", num);
            break;
        }
    printf("\n");
}
```

פלט התוכנית הוא כדלהלן:

```
The reducible numbers between 2 and 20 are:
4 6 8 9 10 12 14 15 16 18 20
```

```
#include <math.h>
```

הוספנו לתוכנית הוראה זו, המורה למהדר להתייחס אל הנתונים שבקובץ math.h, משום שהשתמשנו בפונקציה sqrt.

```
    int num, div;
```

המשתנה num הוא משתנה מונה, בו נשתמש כדי לבצע את הלולאה החיצונית. המשתנה div הוא משתנה עזר, שאת תפקידו נראה בהמשך.

```
    for (num = 2 ; num <= 20 ; num++)
        /* ... */
```

לולאה זו מציבה במשתנה num את הערך 2, ומעלה אותו ב-1 בסוף כל איטרציה, עד שערכו גבוה מ-20. בכל איטרציה שלה, ייבדק המספר שבמשתנה num, ואם הוא מספר פריק, הוא ייפלט על המסך.

```
        for (div = 2 ; div <= sqrt(num) ; div++) {
            /* ... */
```

לולאה פנימית זו מתבצעת בכל איטרציה של הלולאה החיצונית, כלומר, עבור כל מספר בין 2 ל-20. היא בודקת האם מספר זה הוא פריק, ואם כן, פולטת אותו על המסך. כדי לדעת אם מספר נתון הינו פריק, נבדוק האם הוא מתחלק ללא שארית באחד מהמספרים השלמים בין 2 לשורשו הריבועי (שהרי כל מספר מתחלק ב-1 ללא שארית, ואף מספר לא מתחלק ללא שארית במספר הגדול משורשו הריבועי). אם נמצא שכן, אזי לפנינו מספר פריק.

במהלך איטרציות הלולאה, ערכי המשתנה div הם כל הערכים בהם יש לחלק את num כדי למצוא אם הוא פריק או ראשוני. אם נמצא ש-num מתחלק באחד מהערכים הללו ללא שארית, נדע שהוא מספר פריק.

```
            if (num % div != 0)
                continue;
```

משפט התנאי הזה מבצע את משפט continue אם המספרים שבמשתנים num ו-div אינם מתחלקים זה בזה ללא שארית.

```
            printf("%d ", num);
            break;
```

אם התבצע משפט continue, תסתיים מייד האיטרציה הנוכחית של הלולאה הפנימית. לכן, שני המשפטים שלהלן יתבצעו רק אם ביטוי התנאי לא התקיים, כלומר, אם המספר שבמשתנה num מתחלק במספר שבמשתנה div ללא שארית. במקרה זה, המספר שבמשתנה num הוא מספר פריק, ויש לפלוט אותו על המסך. כמו-כן, אין צורך בבדיקות נוספות לגבי מספר זה, ולכן מבוצע משפט break, המסיים את הלולאה הפנימית ומביא לבדיקת פריקותו של המספר הבא.

סיכום

- הלולאה while מבצעת את המשפט השייך לה כל עוד ביטוי הלולאה שלה מתקיים.
- הלולאה do-while דומה ללולאה while, אך המשפט השייך לה מבוצע לפני כל פעם שנבדק ביטוי הלולאה שלה.
- הלולאה for מכילה שלושה ביטויים בין הסוגריים שלה. הביטוי הראשון יבוצע לפני הפעם הראשונה שמבוצע המשפט השייך לה. הביטוי השני הוא ביטוי הלולאה שלה. הביטוי השלישי יבוצע לאחר כל פעם שמבוצע המשפט השייך לה.
- ניתן להשמיט כל אחד מביטויי הלולאה for, אך שני תווי נקודה-פסיק תמיד יופיעו בין הסוגריים שלה.
- אופרטור הפסיק מבצע את הביטוי שמשמאלו, אחר כך את הביטוי שמימינו, ומחזיר את הערך של הביטוי שמימינו.
- משפטי תנאי ולולאות הם משפטים בודדים, ולכן ניתן לשייכם זה לזה.
- מבנה של לולאה השייכת ללולאה אחרת מכונה **קינון לולאות**.
- לולאה אינסופית היא לולאה שביטוי הלולאה שלה מתקיים תמיד.
- המשפט goto מדלג למקום בתוכנית בו הוגדרה תווית. יש להימנע מהשימוש בו.
- המשפט break יוצא ממשפט switch ומהלולאה הפנימית ביותר בה הוא נמצא. מומלץ להימנע מלהשתמש בו בלולאות.
- המשפט continue מסיים את האיטרציה הנוכחית של הלולאה הפנימית ביותר בה הוא נמצא. יש להימנע מהשימוש בו.

פרק 8

פונקציות

פונקציה היא תת-תוכנית הכוללת הוראות, המתבצעות כאשר היא נקראת. עד כה נוכחנו בפעולתה של הפונקציה main, הנקראת אוטומטית בכל פעם שמורץ קובץ ההרצה, ואז מבוצעות כל ההוראות המופיעות בה. בפרק זה נלמד כיצד להגדיר פונקציות משלנו, שיכילו הוראות הגיוניות כלשהן. לפונקציות אלו נוכל לקרוא בכל שלב, ולהיעזר בהן כדי לקצר ולייעל את תוכניותינו.

היתרונות בשימוש בפונקציות

גם בתוכניות הפשוטות ביותר שכתבנו עד כה קראנו לפונקציות. למשל, printf ו-scanf הן שתי פונקציות שכיחות בהן השתמשנו רבות. פונקציות אלו הוגדרו עבורנו על ידי מפתחי המהדר, ולכן כל שיש לעשות הוא לקרוא להן. בתוכניות ארוכות נוהג המתכנת להגדיר פונקציות משלו ולקרוא להן בעת הצורך. לשימוש בפונקציות מספר יתרונות:

- על ידי חלוקת תוכנית ארוכה למספר פונקציות קצרות, שכל אחת מבצעת מטלה שלמה אחרת, קל יותר להבין ולשנות את התוכנית.
- ניתן בקלות להשתמש בפונקציות שפותחו על ידי מתכנת אחר. כל שיש לעשות הוא להעתיק אותן לקובץ התוכנית ולקרוא להן כפי שקראנו ל-printf ול-scanf, בלי "להמציא את הגלגל מחדש".
- כל ההוראות שבגוף הפונקציה מתבצעות אוטומטית כאשר היא נקראת, על ידי הוראה בודדת. באופן זה אין צורך לכתוב את כל הוראות הפונקציה בכל פעם מחדש. לדוגמה, הפונקציה printf מכילה מספר רב של שורות; דמיין לעצמך תוכנית הכוללת שורות אלו שוב ושוב, בכל פעם שיש להציג מלל כלשהו על המסך!

- התוכנית קטנה כאשר משתמשים בפונקציות, שהרי, גם אם נקרא לפונקציה מסוימת פעמים רבות, הוראותיה יופיעו בקובץ ההרצה פעם אחת בלבד (היכן שנמצא גוף הפונקציה בקובץ המקור), ולא בכל פעם שהפונקציה נקראת. לולא השתמשנו בפונקציות, היה עלינו לחזור ולכתוב הוראות אלו בכל פעם שהפונקציה היתה צריכה להיקרא.
- אם נרצה במועד כלשהו לשנות את אופן ביצועה של הפונקציה, נצטרך לשנות רק את ההוראות המופיעות בגופה – הקריאות לפונקציה לא ישתנו. לולא השתמשנו בפונקציות, היה עלינו לבצע את השינוי במקומות רבים בתוכנית.

קיים חסרון אחד בשימוש בפונקציות, והוא איטיות קובץ ההרצה. קריאה לפונקציה וביצוע הוראותיה אורכים זמן רב יותר מאשר ביצוע הוראותיה לו הופיעו בקוד המקור שוב ושוב, כלומר, בלי הקריאה לה. אך מגרעת המהירות כמעט שאינה מורגשת, והיא מתגמדת לעומת היתרונות הרבים בשימוש בפונקציות.

הגדרת הפונקציה

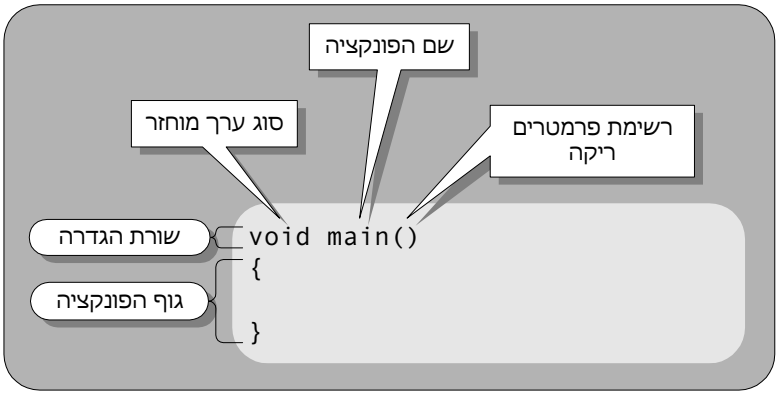
במקרים רבים נרצה להשתמש בפונקציות משלנו, כלומר, להגדיר פונקציה הכוללת הוראות מסוימות ולבצען בעת הצורך על ידי קריאה לפונקציה. לפני שנוכל לקרוא לפונקציה, עלינו להגדירה. הגדרת הפונקציה מורכבת משני חלקים:

- שורת ההגדרה – שורת ההגדרה מבדילה פונקציה אחת מרעותה. היא קובעת את מאפייני הפונקציה הבאים:
  - סוג הערך המוחזר – קיימות פונקציות שמחזירות ערכים (כגון getch) ופונקציות שאינן מחזירות ערכים. כאשר פונקציה מחזירה ערך, עלינו לקבוע מראש את סוגו (char, int וכדומה). בערך המוחזר מפונקציה ניתן להשתמש באופנים שונים; הנה דוגמה לשימוש בערך המוחזר מהפונקציה getch:

```
ch = getch();   /* call getch and use its returned value */
```

  - שם הפונקציה – לכל פונקציה שם שונה, ובאמצעות כתיבתו היא מתבצעת.
  - רשימת פרמטרים – קיימות פונקציות שמקבלות פרמטרים כאשר הן נקראות (כגון printf) ומבצעות עליהם פעולות שונות, ופונקציות שאינן מקבלות פרמטרים, כלומר, שרשימת הפרמטרים שלהן ריקה (כגון getch).
- גוף הפונקציה – גוף הפונקציה מורכב מצמד סוגריים מסולסלים המופיע לאחר שורת ההגדרה. המשפטים שיופיעו בתוכו יתבצעו כאשר תיקרא הפונקציה.

main, כזכור, היא פונקציה לכל דבר, ולכן היא מוגדרת על-פי הנאמר לעיל. הדבר מוצג באיור הבא:



***איור 8.1** – מרכיבי הגדרת הפונקציה main*

הבה נבחן לעומק את הגדרת הפונקציה main. שורת ההגדרה שלה נפתחת במילה השמורה void. מילה זו מציינת שהפונקציה אינה מחזירה כל ערך. לאחר מכן מופיע שם הפונקציה, main. שם הפונקציה הוא מזהה לכל דבר, וחלים עליו החוקים הרגילים בנתינת שמות למשתנים. היינו, שם פונקציה יכול להכיל ספרות, אותיות לועזיות וקו תחתי (_), אך אין להתחילו בספרה. כמו-כן אין לקרוא לשתי פונקציות בשם זהה.

עצה

נהוג להתחיל את שם הפונקציה באות לועזית גדולה, ואם הוא מורכב מכמה מילים, להתחיל כל מילה באות גדולה. למשל, פונקציה המחשבת ממוצע של ציונים תיקרא CalculateGradesAverage. לעומת זאת, שמות משתנים יורכבו מאותיות לועזיות קטנות, ואם הם מורכבים ממספר מילים, נפריד ביניהן בקו תחתי. למשל, משתנה לחישוב ממוצע ציונים ייקרא grades_average.

הקפדה על הנאמר לעיל תקל על הבנת התוכנית, שכן נוכל לקבוע אם כל מזהה בתוכנית הוא משתנה או פונקציה רק על ידי בחינת אותיותיו.

הסוגריים המופיעים לאחר שם הפונקציה מכילים את רשימת הפרמטרים שלה. בהגדרת הפונקציה main לא מופיע דבר בין הסוגריים, כלומר, רשימת הפרמטרים ריקה. הדבר מציין שהפונקציה אינה מקבלת ערכים כאשר היא נקראת (כמו הפונקציה getch).

לאחר שורת ההגדרה מופיע גוף הפונקציה, המורכב מסוגריים מסולסלים ומהוראות המופיעות ביניהם. גוף פונקציה יכול להכיל אלפי משפטים או אף לא משפט אחד; יהיה אשר יהיה, חובה להגדירו על ידי סוגריים מסולסלים.

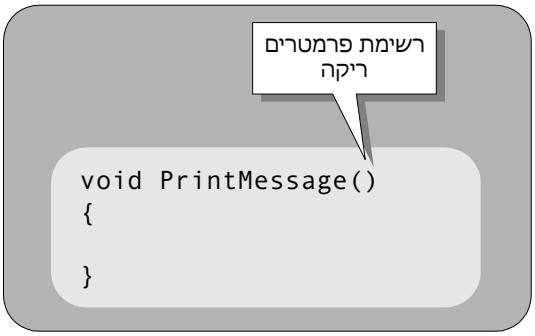
שים לב!

לאחר שורת ההגדרה של הפונקציה לא מופיע תו נקודה-פסיק.

פונקציות פשוטות

לפני שנעמיק בפונקציות מורכבות, נציג מספר דוגמאות להגדרת פונקציות שאינן מקבלות פרמטרים. פונקציות כאלו נועדו, בדרך-כלל, לביצוע פעולה קבועה. דוגמה לפונקציה שאינה מקבלת פרמטרים היא getch, המבצעת פעולה זהה בכל פעם שהיא נקראת – המתנה ללחיצת מקש והחזרת ערך ASCII המתאים לו. מכיון שאופן ביצוע הקלט זהה בכל קריאה ל-getch, אין צורך לשלוח אליה פרמטרים. זאת בניגוד לפונקציה scanf, שיכולה לקלוט בכל פעם מספר משתנה של ערכים, ולכן יש לשלוח אליה פרמטרים שיגדירו כיצד תתבצע.

כאשר פונקציה אינה מקבלת פרמטרים, נאמר כי רשימת הפרמטרים שלה ריקה. רשימת פרמטרים ריקה מיוצגת על ידי סוגריים פותחים וסוגרים בלבד, כפי שמוצג באיור הבא:



איור 8.2 *– הגדרת פונקציה שאינה מקבלת פרמטרים*

איך לא...

חובה לציין סוגריים פותחים וסוגרים בכל הגדרת פונקציה, בין אם היא מקבלת פרמטרים ובין אם לאו. למשל, הגדרת הפונקציה הבאה אינה נכונה:

```
void PrintMessage      /* wrong, () required */
{
    /* ... */
}
```

הבה נוסיף מספר משפטים לגוף הפונקציה שלעיל:

```
void PrintMessage()
{
    printf("Hello, and welcome to my program. \n");
    printf("Creator: Mr. Unknown, 1999-2000. \n");
    printf("Telephone number: 555-1234, LA, USA. \n\n");
}
```

עצה

שים לב כי בדומה למשפטים השייכים לפונקציה main, המשפטים השייכים לפונקציה PrintMessage הוזחו ימינה. מומלץ לנהוג כך, משום שבאופן זה קל להבחין אילו משפטים שייכים לכל פונקציה.

כדי לבצע את הוראות הפונקציה יש לקרוא לה. גם בקריאה לפונקציה חובה לרשום צמד סוגריים לאחר שמה. לפונקציה שלעיל, למשל, יש לקרוא כך:

```
PrintMessage();      /* invoke the function PrintMessage */
```

בכל פעם שתיקרא הפונקציה ייפלט המלל הבא:

```
Hello, and welcome to my program.
Creator: Mr. Unknown, 1999-2000.
Telephone number: 555-1234, LA, USA.
```

המהדר יודע כי הפונקציה PrintMessage אינה מחזירה ערך ואינה מקבלת פרמטרים, מפני שכך ציינו בשורת ההגדרה. עתה, לאורך כל התוכנית יש להתייחס אל הפונקציה בהתאם. לכן שורות הקריאה הבאות לפונקציה אינן נכונות:

```
ch = PrintMessage();      /* PrintMessage returns no value */
PrintMessage(12);         /* PrintMessage receives no parameters */
void PrintMessage();      /* don't use void when invoking */
```

אילו השמטנו את הסוגריים המופיעים לאחר שם הפונקציה בקריאה אליה, כך:

```
PrintMessage;      /* dangerous! */
```

לא היתה מתקבלת הודעת שגיאה מהמהדר, אך הפונקציה לא היתה נקראת. נסביר בהמשך מדוע.

דוגמה

עד כה ראינו כיצד להגדיר פונקציה פשוטה וכיצד לקרוא לה. בדוגמה זו נכתוב תוכנית מלאה הקוראת לפונקציה PrintMessage שהוגדרה לעיל.

תוכנית 8.1 *MESSAGE.C*

```
#include <stdio.h>

void PrintMessage()
{
    printf("Hello, and welcome to my program. \n");
    printf("Creator: Mr. Unknown, 1999-2000. \n");
    printf("Telephone number: 555-1234, LA, USA. \n\n");
}

void main()
{
    int n;

    printf("How many times would you like to see the output? ");
    scanf("%d", &n);

    while (n > 0) {
        PrintMessage();     /* invoke PrintMessage */
        n--;
    }
}
```

הנה דוגמה לפלט המתקבל מהרצת תוכנית זו:

```
How many times would you like to see the output? 3
Hello, and welcome to my program.
Creator: Mr. Unknown, 1999-2000.
Telephone number: 555-1234, LA, USA.

Hello, and welcome to my program.
Creator: Mr. Unknown, 1999-2000.
Telephone number: 555-1234, LA, USA.

Hello, and welcome to my program.
Creator: Mr. Unknown, 1999-2000.
Telephone number: 555-1234, LA, USA.
```

```
void PrintMessage()
{
    /* ... */
}

void main()
{
    /* ... */
}
```

לפני שנתעמק במשפטי התוכנית, שים לב כי הפונקציה PrintMessage מוגדרת לפני הפונקציה main. הדבר הכרחי, משום שהמהדר צריך לדעת את המאפיינים של פונקציה (הנקבעים בהגדרתה) לפני שהוא נתקל במשפט הקורא לה. המהדר בודק את התוכנית מלמעלה למטה, ומכיון שהמשפט הקורא לפונקציה PrintMessage מצוי בפונקציה main, עלינו להגדיר את PrintMessage לפני main.

איך לא...

אין להגדיר פונקציה אחת בתוך פונקציה אחרת. למשל, לא יכולנו להגדיר את הפונקציה PrintMessage בתוך הפונקציה main, כך:

```
void main()
{
    void PrintMessage()     /* wrong */
    {

    }
}
```

```
    printf("How many times would you like to see the output? ");
    scanf("%d", &n);
```

בשורות אלו מתבקש המשתמש להזין את מספר הפעמים שברצונו לראות את הפלט המתקבל מהקריאה לפונקציה PrintMessage.

```
    while (n > 0) {
        /* ... */
        n--;
    }
```

לולאה זו מתבצעת מספר פעמים השווה למספר שהזין המשתמש.

```
        PrintMessage();
```

משפט זה, המתבצע בכל איטרציה של הלולאה, קורא לפונקציה PrintMessage, ובכך מבצע את ההוראות שבה.

סדר ביצוע פונקציות והחזרה מהן

הפונקציות בשפת C אינן נקראות על-פי סדר הופעתן בקובץ המקור. בחן, למשל, את תוכנית 8.1. הוראות הפונקציה main מתבצעות לפני הוראות הפונקציה PrintMessage, אף ש-main מוגדרת לאחר PrintMessage.

ההוראות הראשונות שמתבצעות הן, כאמור, ההוראות המופיעות בפונקציה main. אם אחת מההוראות הללו תכיל קריאה לפונקציה שנייה, יתבצעו הוראות הפונקציה השנייה, ולאחר מכן היא תסתיים. כאשר תסתיים הפונקציה השנייה, יתבצעו הוראות הפונקציה main המופיעות לאחר ההוראה שקראה לפונקציה השנייה.

מתוך הפונקציה השנייה ניתן לקרוא לפונקציה שלישית, ולאחר שהוראות הפונקציה השלישית יתבצעו, היא תסתיים, ויתבצעו הוראות הפונקציה השנייה המופיעות לאחר ההוראה שקראה לפונקציה השלישית.

דוגמה

התוכנית שנכתוב בדוגמה זו תבצע פעולה זהה לזו של תוכנית 8.1, אך היא תכיל מספר רב יותר של פונקציות, שתקראנה אחת לשנייה בהתאם לצורך. לא נהוג לחלק תוכנית כה קצרה לפונקציות רבות, אך נעשה זאת עתה כדי לעמוד על הסדר בו מבוצעות ומסתיימות פונקציות.

תוכנית 8.2 *MESSAGE2.C*

```
#include <stdio.h>

void PrintNewLine()
{
    printf("\n");
}

void PrintMessage()
{
    printf("Hello, and welcome to my program.");
    PrintNewLine();         /* advance to new line */
    printf("Creator: Mr. Unknown, 1999-2000.");
    PrintNewLine();         /* advance to new line */
    printf("Telephone number: 555-1234, LA, USA.");
    PrintNewLine();         /* advance to new line */
    PrintNewLine();         /* advance to new line */
}
```

```
void ReadNumber()
{
    int n;

    printf("How many times would you like to see the output? ");
    scanf("%d", &n);

    while (n > 0) {
        PrintMessage();     /* display output */
        n--;
    }
}

void main()
{
    ReadNumber();           /* invoke ReadNumber */
}
```

הפלט המתקבל מהרצת תוכנית זו זהה לפלט המתקבל מהרצת תוכנית 8.1.

```
void PrintNewLine()
{
    /* ... */
}

void PrintMessage()
{
    /* ... */
}

void ReadNumber()
{
    /* ... */
}

void main()
{
    /* ... */
}
```

בתוכנית זו, הפונקציה main קוראת לפונקציה ReadNumber, הפונקציה ReadNumber קוראת לפונקציה PrintMessage, והפונקציה PrintMessage קוראת לפונקציה PrintNewLine. מאחר שהמהדר מחייב שפונקציה תוגדר לפני שמתבצעת קריאה אליה, הפונקציה ReadNumber מוגדרת לפני הפונקציה main, הפונקציה PrintMessage מוגדרת לפני הפונקציה ReadNumber, והפונקציה PrintNewLine מוגדרת לפני הפונקציה PrintMessage.

```
void PrintNewLine()
{
    printf("\n");
}
```

זוהי הגדרת הפונקציה PrintNewLine, המורידה את סמן הטקסט לשורה חדשה כאשר היא נקראת. נזכיר כי חובה לרשום את הסוגריים המסולסלים המציינים את גוף הפונקציה גם אם גוף הפונקציה מכיל הוראה אחת בלבד.

```
void PrintMessage()
{
    printf("Hello, and welcome to my program.");
    PrintNewLine();         /* advance to new line */
    printf("Creator: Mr. Unknown, 1999-2000.");
    PrintNewLine();         /* advance to new line */
    printf("Telephone number: 555-1234, LA, USA.");
    PrintNewLine();         /* advance to new line */
    PrintNewLine();         /* advance to new line */
}
```

זוהי הגדרת הפונקציה PrintMessage. לאחר כל קריאה ל-printf, המציגה משפט מלל אחר על המסך, נקראת הפונקציה PrintNewLine, המורידה את סמן הטקסט לשורה חדשה.

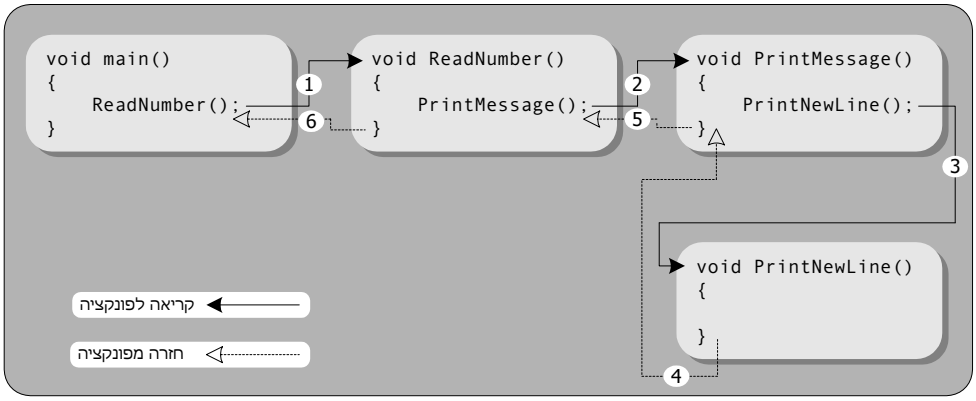
```
void ReadNumber()
{
    int n;

    printf("How many times would you like to see the output? ");
    scanf("%d", &n);

    while (n > 0) {
        PrintMessage();     /* display output */
        n--;
    }
}
```

זוהי הגדרת הפונקציה ReadNumber, המכילה הוראות הזהות לאלו שהכילה פונקציית main בתוכנית 8.1. כדי לבצע הוראות אלו, יש לקרוא לפונקציה ReadNumber מתוך הפונקציה main.

```
void main()
{
    ReadNumber();           /* invoke ReadNumber */
}
```

פונקציית main מכילה הוראה אחת בלבד, הקוראת לפונקציה ReadNumber. הפונקציה ReadNumber קולטת מספר מהמשתמש וקוראת לפונקציה PrintMessage מספר פעמים השווה לו. הפונקציה PrintMessage, בתורה, קוראת לפונקציה PrintNewLine כדי לרדת לשורה חדשה.

סדר הקריאה לפונקציות שבתוכנית זו והחזרה מהן מוצג באיור הבא:



איור 8.3 *– סדר הקריאה והחזרה מהפונקציות שבתוכנית 8.2*

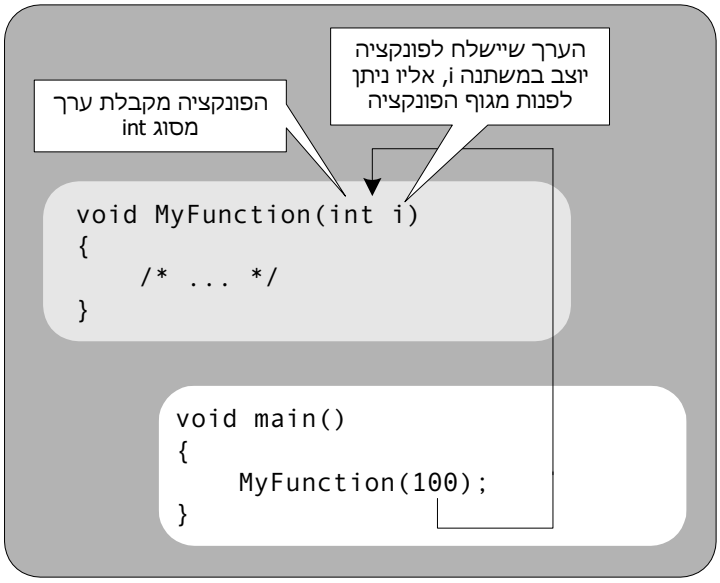
פונקציות שמקבלות פרמטרים

פונקציות שמקבלות פרמטרים הן פונקציות אליהן נשלחים ערכים כאשר הן נקראות. הפונקציות שהגדרנו עד כה לא קיבלו פרמטרים, וציינו זאת בכך שהשארנו את רשימות הפרמטרים שלהן ריקות. כדי לציין שפונקציה מקבלת פרמטר כלשהו, ראשית עלינו לקבוע את סוגו (int ,char וכדומה). לאחר מכן נגדיר משתנה מתאים בשורת ההגדרה של הפונקציה, כלומר, בין הסוגריים המופיעים לאחר שמה. הנה, למשל, הגדרה של פונקציה המקבלת פרמטר בודד מסוג int:

```
void MyFunction(int i)
{
    /* ... */
}
```

במשתנה i יוצב הערך שיישלח אל הפונקציה כאשר היא תיקרא.

האיור הבא מציג זאת:



איור 8.4 *– הגדרת פונקציה שמקבלת פרמטר מסוג int והקריאה לה*

ניתן לפנות אל המשתנה i מתוך גוף הפונקציה ולבצע עליו את כל הפעולות שביצענו על משתנים עד כה. למשל, כדי למצוא האם ערך המשתנה i הוא חיובי ולפלוט הודעה מתאימה, נכתוב את הפונקציה MyFunction כך:

```
void MyFunction(int i)
{
    if (i > 0)
        printf("%d is positive! \n", i);
}
```

עתה, אם תיקרא הפונקציה באופן הבא:

```
MyFunction(100);
```

יוצב הערך 100 במשתנה i, ויתקבל הפלט הבא:

```
100 is positive!
```

ואם הפונקציה תיקרא כך:

```
MyFunction(-5);
```

יוצב הערך 5- במשתנה i, ולא יתקבל כל פלט.

נגדיר עתה פונקציה בשם PrintEven, המקבלת פרמטר שלם מסוג int ופולטת את המספרים הזוגיים מ-0 עד אליו.

```
void PrintEven(int n)
{
    int i;
    for (i = 0; i <= n; i += 2)
        printf("%d ", i);
}
```

אם תיקרא הפונקציה באופן הבא:

```
PrintEven(20);
```

יתקבל הפלט הבא:

```
0 2 4 6 8 10 12 14 16 18 20
```

מכיון שהגדרנו משתנה בשורת ההגדרה של הפונקציה, המהדר יצפה שהיא תקבל פרמטר מתאים בכל קריאה אליה. לכן לא נוכל לקרוא ל-PrintEven באופנים הבאים:

```
PrintEven();           /* parameter required! */
PrintEven(10, 20);     /* only one parameter! */
```

אך נוכל לקרוא לה כך:

```
PrintEven(9.5);
```

במקרה זה, ימיר המהדר את הערך העשרוני 9.5 לערך מסוג int. כפי שראינו בפרקים הקודמים, פעולה זו מתבצעת על ידי התעלמות מהחלק העשרוני של המספר. לכן המשתנה n יכיל את הערך 9, וייפלט המלל:

```
0 2 4 6 8
```

שים לב!

אם נשלח משתנה כפרמטר לפונקציה, נניח ששמו a, וערך הפרמטר המתאים לו ישתנה במהלך הפונקציה, ערך a לא ישתנה בהתאם. לעת עתה לא נרחיב בנושא זה.

דוגמה

בדוגמה זו נגדיר שתי פונקציות המקבלות פרמטרים ונקרא להן באופן מתאים. פונקציה אחת תקבל פרמטר מסוג int ותפלוט את העצרת שלו ; הפונקציה השנייה תקבל פרמטר מסוג int ותפלוט את סכום המספרים הטבעיים עד אליו.

תוכנית 8.3

PARAM.C

```
#include <stdio.h>

void PrintFactorial(int i)
{
    long fact;

    for (fact = 1 ; i > 1 ; i--)
        fact *= i;

    printf("Factorial: %ld \n", fact);
}

void PrintSum(int s)
{
    int sum;

    for (sum = 0 ; s > 0 ; s--)
        sum += s;

    printf("Sum: %d \n", sum);
}

void main()
{
    int n;

    printf("Please enter a non-negative number: ");
    scanf("%d", &n);

    PrintFactorial(n);
    PrintSum(n);
}
```

הנה דוגמה לפלט המתקבל מהרצת תוכנית זו:

```
Please enter a non-negative number: 10
Factorial: 3628800
Sum: 55
```

```
void PrintFactorial(int i)
{
    /* ... */
}
```

זוהי הגדרת הפונקציה PrintFactorial. פונקציה זו מקבלת פרמטר בודד מסוג int ופולטת את העצרת שלו.

```
    long fact;

    for (fact = 1 ; i > 1 ; i--)
        fact *= i;
```

אלו הן ההוראות השייכות לפונקציה PrintFactorial, ולאחר ביצוען יכיל המשתנה fact את עצרת המספר i. סוג המשתנה fact הוא long כדי שנוכל לחשב את העצרת של יותר מספרים. המשפט השייך ללולאת for מציב את מכפלת ערכי המשתנים i ו-fact במשתנה fact. במשתנה fact מוצב הערך 1 לפני האיטרציה הראשונה של הלולאה, ובסוף כל איטרציה יורד הערך של i ב-1. לכן הערך שמחושב על ידי המשתנה fact הוא:

$$i * (i - 1) * (i - 2) * (i - 3) * \ldots * 2$$

וערך זה הוא עצרת המספר i.

```
void PrintSum(int s)
{
    /* ... */
}
```

זוהי הגדרת הפונקציה PrintSum. פונקציה זו מקבלת פרמטר בודד מסוג int ופולטת את סכום המספרים הטבעיים עד אליו.

```
    int sum;

    for (sum = 0 ; s > 0 ; s--)
        sum += s;
```

אלו הן ההוראות השייכות לפונקציה PrintSum, ולאחר ביצוען יכיל המשתנה sum את סכום המספרים הטבעיים עד ל-s. במשתנה sum מוצב הערך 0 לפני האיטרציה הראשונה של הלולאה. בכל איטרציה של הלולאה, מתוסף ערך המשתנה s לערך המשתנה sum. ערך s פוחת ב-1 בסוף כל איטרציה, ולכן הערך שמחושב על ידי המשתנה sum הוא:

$$s + (s - 1) + (s - 2) + \ldots + 1$$

וערך זה הוא סכום המספרים הטבעיים עד ל-s.

```
void main()
{
    int n;

    printf("Please enter a non-negative number: ");
    scanf("%d", &n);

    /* ... */
```

אלו הן ההוראות השייכות לפונקציה main, ולאחר ביצוען ייקלט ערך שלם לתוך המשתנה n.

```
    PrintFactorial(n);
    PrintSum(n);
```

בשורות אלו מתבצעות קריאות לפונקציות PrintFactorial ו-PrintSum, כאשר אל שתיהן נשלח הפרמטר n. הפונקציה PrintFactorial תפלוט את עצרת המספר n, והפונקציה PrintSum – את סכום המספרים הטבעיים עד אליו. נזכיר כי ערך המשתנה n אינו משתנה לאחר שורות אלו, אף שהפרמטרים המתאימים לו בפונקציות PrintFactorial ו-PrintSum משתנים.

פונקציות יכולות לקבל יותר מפרמטר אחד. כדי להגדיר פונקציה המקבלת מספר פרמטרים, יש להפריד כל פרמטר ופרמטר על ידי פסיק. למשל, פונקציה המקבלת שלושה פרמטרים, מסוג int, float ו-char בהתאמה, תוגדר כך:

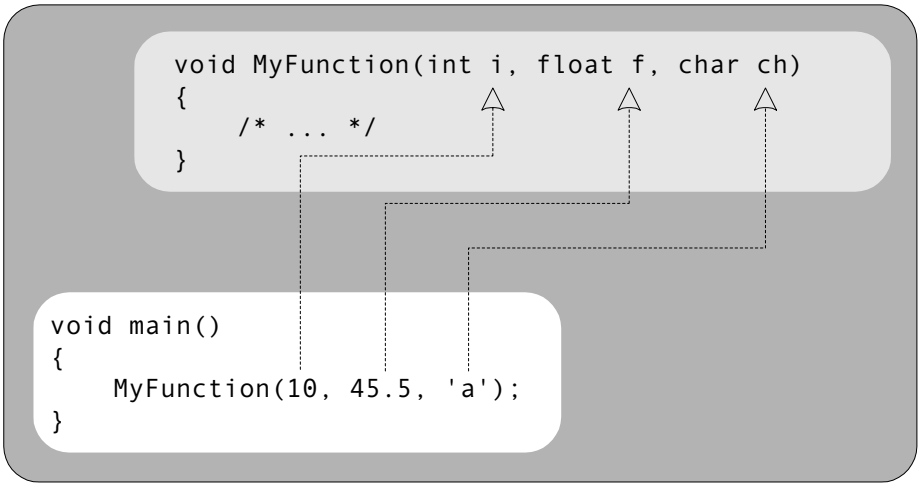
```
void MyFunction(int i, float f, char ch)
{
    /* ... */
}
```

שורת ההגדרה של פונקציה זו כוללת שלושה משתנים, ולכן, בקריאה לפונקציה יהיה עלינו לשלוח שלושה ערכים, כמודגם להלן:

```
MyFunction(10, 45.5, 'a');
```

כל ערך הנשלח אל הפונקציה יוצב בפרמטר המתאים לו.

הדבר מוצג באיור הבא:



***איור 8.5** – שליחת מספר פרמטרים לפונקציה*

אם נקרא לפונקציה MyFunction עם מספר שונה של פרמטרים, יפלוט המהדר הודעת שגיאה.

במחשבה שנייה...

אם ניתן לשלוח לפונקציה מסוימת מספר קבוע של פרמטרים, כיצד ייתכן שהפונקציות printf ו-scanf מקבלות מספר משתנה של פרמטרים?

בשפת C ניתן להגדיר פונקציות המקבלות מספר משתנה של פרמטרים. השימוש בהן מורכב יותר מהשימוש בפונקציה רגילה, ולכן לא נדון בו בשלב זה.

משתנים כגון i, f ו-ch, המוגדרים בשורת ההגדרה של הפונקציה, מכונים **משתנים פורמליים** (formal variables). משתנים המוגדרים בגוף הפונקציה מכונים **משתנים לוקליים** (local variables), ועד לפרק זה השתמשנו רק במשתנים כאלה.

על משתנים פורמליים חלים חוקי המשתנים הרגילים הנוגעים לנתינת שמות, לביצוע פעולות עם אופרטורים וכדומה. אך בהגדרתם יש לשים לב לחוקים הבאים:

- ניתן להגדיר משתנים לוקליים מאותו סוג בלי לציין את סוגם בכל פעם מחדש; למשל:

```
int a, b, c;
```

אך אין לעשות זאת בעת הגדרת משתנים פורמליים. לצד כל משתנה פורמלי חובה לציין את סוגו, אפילו אם סוג זה זהה לסוג המשתנה הפורמלי הקודם לו. לכן שורת ההגדרה הבאה אינה נכונה:

```
void MyFunction(int a, b, c)      /* wrong */
```

ובמקומה יש לכתוב:

```
void MyFunction(int a, int b, int c)
```

- אין לאתחל משתנים פורמליים בעת הגדרתם. לכן שורת ההגדרה הבאה אינה נכונה:

```
void MyFunction(int a = 1, int b = 2, int c = 3)      /* wrong */
```

- לאחר הגדרת משתנים לוקליים יש להוסיף נקודה-פסיק. לא כן ברשימת הפרמטרים, בה מופרד כל משתנה פורמלי ממשנהו על ידי פסיק, ותו הנקודה-פסיק לא מופיע כלל. לכן שורת ההגדרה הבאה אינה נכונה:

```
void MyFunction(char ch; int i)      /* wrong */
```

דוגמה

בדוגמה זו נכתוב תוכנית הקולטת ארבעה מספרים מהמשתמש וקוראת לפונקציות מתאימות, הפולטות את המספר הגדול ביותר ואת המספר הקטן ביותר שהוזן.

תוכנית 8.4 *MAX_MIN.C*

```
#include <stdio.h>

void PrintMax(float a, float b, float c, float d)
{
    float max = a;

    if (b > max)
        max = b;
    if (c > max)
        max = c;
    if (d > max)
        max = d;
    printf("Biggest number: %f \n", max);
}
```

```
void PrintMin(float l, float m, float n, float o)
{
    float min = l;

    if (m < min)
        min = m;
    if (n < min)
        min = n;
    if (o < min)
        min = o;
    printf("Smallest number: %f \n", min);
}

void main()
{
    float num1, num2, num3, num4;

    printf("Please enter four numbers: ");
    scanf("%f %f %f %f", &num1, &num2, &num3, &num4);

    PrintMax(num1, num2, num3, num4);
    PrintMin(num1, num2, num3, num4);
}
```

פלט לדוגמה מתוכנית זו מוצג להלן:

```
Please enter four numbers: 88 -3 88.5 2.22
Biggest number: 88.500000
Smallest number: -3.000000
```

```
void PrintMax(float a, float b, float c, float d)
{
    /* ... */
}
```

זוהי הגדרת הפונקציה PrintMax, המקבלת ארבעה ערכים עשרוניים ופולטת את הגדול מביניהם.

```
    float max = a;

    if (b > max)
        max = b;
    if (c > max)
        max = c;
    if (d > max)
        max = d;
```

אלו הן ההוראות השייכות לפונקציה PrintMax, ולאחר ביצוען יוצב במשתנה max הערך הגבוה מבין ערכי המשתנים a, b, c ו-d. האלגוריתם בו עשינו שימוש בשורות אלו מוסבר בפירוט בפרק השני.

```
void PrintMin(float l, float m, float n, float o)
{
    /* ... */
}
```

זוהי הגדרת הפונקציה PrintMin, המקבלת ארבעה ערכים עשרוניים ופולטת את הקטן מביניהם.

```
    float min = l;

    if (m < min)
        min = m;
    if (n < min)
        min = n;
    if (o < min)
        min = o;
```

אלו הן ההוראות השייכות לפונקציה PrintMin, ולאחר ביצוען יוצב במשתנה min הערך הנמוך מבין ערכי המשתנים a, b, c ו-d. האלגוריתם בו השתמשנו בשורות אלו דומה לאלגוריתם בו השתמשנו בפונקציה PrintMax.

```
void main()
{
    float num1, num2, num3, num4;

    printf("Please enter four numbers: ");
    scanf("%f %f %f %f", &num1, &num2, &num3, &num4);

    /* ... */
```

אלו הן ההוראות השייכות לפונקציה main, ולאחר ביצוען ייקלטו מהמשתמש ארבעה ערכים עשרוניים לתוך המשתנים num1, num2, num3 ו-num4.

```
    PrintMax(num1, num2, num3, num4);
    PrintMin(num1, num2, num3, num4);
```

בשורות אלו מתבצעות קריאות לפונקציות PrintMax ו-PrintMin, כאשר אל שתיהן נשלחים הערכים שהזין המשתמש.

פונקציות שמחזירות ערכים

בכל הגדרות הפונקציות שראינו עד כה הופיעה המילה void לפני שמות הפונקציות, ובכך ידע המהדר כי הן אינן מחזירות ערכים. כאשר קוראים לפונקציה שמחזירה ערך, ההוראות שבה מתבצעות כרגיל, ובשלב כלשהו היא מסתיימת ומחזירה ערך כלשהו. אנו מתייחסים אל ערך מוחזר זה כאל ערך קבוע שנכתב במקום בו נכתב שם הפונקציה. נדגים זאת על ידי הפונקציה getch. נניח שקראנו לה באופן הבא:

```
char key;
/* ... */
key = getch();      /* wait for key press */
```

אם נלחצה האות 'a', שערך ASCII שלה הוא 97, תחזיר הפונקציה getch את הערך 97, וערך זה יוצב במשתנה key. במקרה זה, משפט ההשמה ;()key = getch שקול למשפט הבא:

```
key = 97;      /* same as:  key = getch();  if 'a' was pressed */
```

לעומת זאת, המשפט הבא אינו נכון מבחינה תחבירית:

```
getch() += 3;      /* wrong */
```

זאת משום שערך המוחזר מפונקציה נחשב לערך קבוע, ואם, למשל, נלחצה האות 'a', משפט ההשמה שלעיל שקול למשפט הבא:

```
97 += 3;      /* wrong */
```

וזהו, כמובן, משפט שגוי.

לאחר שהזכרנו כיצד לקרוא לפונקציה שמחזירה ערך, ברצוננו להגדיר פונקציה כזו. ראשית, עלינו לבחור את סוג הערך שיוחזר מהפונקציה. סוג זה יכול להיות char, int, וכדומה, והוא ייכתב לפני שם הפונקציה, במקום המילה השמורה void. למשל, פונקציה המחזירה ערך מסוג int תיכתב כך:

```
int MyFunction()
{
    /* ... */
}
```

שים לב!

בעוד שפונקציה רשאית לקבל מספר רב של פרמטרים, היא יכולה להחזיר ערך אחד לכל היותר.

החזרת ערך מפונקציה נעשית על ידי משפט return. לעת עתה, נגדיר משפט זה כמשפט המורכב מהמילה השמורה return ומערך המופיע לצידה. כאשר יבוצע משפט זה, תסתיים הפונקציה ויוחזר הערך המופיע בו. בחן, למשל, את הפונקציה הבאה:

```
int MyFunction()
{
    return 5;      /* exit MyFunction and return 5 */
}
```

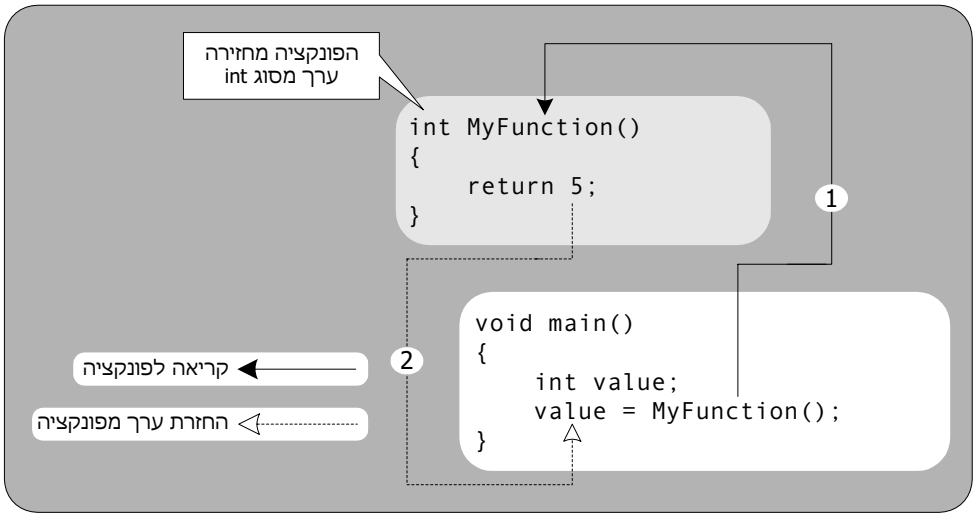
אם תיקרא הפונקציה באופן הבא:

```
value = MyFunction();
```

יבוצע משפט return שבגוף הפונקציה, המביא ליציאה ממנה ולהחזרת הערך 5. ערך זה יוצב במשתנה value, כאילו בוצע המשפט הבא:

```
value = 5;
```

אופן ביצוע הפונקציה והחזרת הערך ממנה מוצג באיור הבא:



***איור 8.6** – החזרת ערך מפונקציה*

באופן דומה, המשפט הבא:

```
printf("returned value: %d", MyFunction());
```

קורא לפונקציה MyFunction, המחזירה את הערך 5, ולכן הוא שקול למשפט הבא:

```
printf("returned value: %d", 5);
```

זהירות!

חשוב לזכור להחזיר ערך מפונקציה שמחזירה ערך. אם, למשל, נגדיר את הפונקציה הבאה:

```
int MyFunction()
{
    /* note: no return statement is used */
}
```

ונקרא לפונקציה MyFunction כך:

```
value = MyFunction();      /* dangerous! */
```

לא ניתן יהיה לצפות את הערך שיוצב במשתנה value. שים לב כי רוב המהדרים יפלטו הודעת שגיאה המוסרת כי הפונקציה MyFunction לא מחזירה ערך, אף שבשורת הגדרתה צוין שהיא מחזירה ערך מסוג int.

דוגמה

בדוגמה זו נכתוב תוכנית הקולטת ציונים מהמשתמש ומחשבת את הממוצע שלהם תוך שימוש בפונקציות המחזירות ערכים.

תוכנית 8.5 *GRADES2.C*

```
#include <stdio.h>

int GetGradesNum()
{
    int count;

    printf("This program reads grades and calculates their average.");
    printf("\nHow many grades would you like to enter? ");
    scanf("%d", &count);

    return count;
}

int GetGrade()
{
    int grade;
```

```
    printf("Please enter a grade: ");
    scanf("%d", &grade);

    return grade;
}

void main()
{
    int average = 0, i, grades_num;

    grades_num = GetGradesNum();

    for (i = 0 ; i < grades_num ; i++)
        average += GetGrade();

    average /= grades_num;
    printf("Average of grades: %d \n", average);
}
```

פלט לדוגמה מהתוכנית מוצג להלן:

```
This program reads grades and calculates their average.
How many grades would you like to enter? 5
Please enter a grade: 85
Please enter a grade: 96
Please enter a grade: 78
Please enter a grade: 81
Please enter a grade: 90
Average of grades: 86
```

```
int GetGradesNum()
{
    /* ... */
}
```

זוהי הגדרת הפונקציה GetGradesNum, הקולטת מהמשתמש את מספר הציונים שיוזנו ומחזירה מספר זה.

```
    int count;

    printf("This program reads grades and calculates their
        average.");
    printf("\nHow many grades would you like to enter? ");
    scanf("%d", &count);
```

אלו הן ההוראות השייכות לפונקציה GetGradesNum, ולאחר ביצוען יוצב במשתנה count ערך שמזין המשתמש. ערך זה יקבע כמה ציונים ייקלטו.

```
    return count;
```

משפט return זה מביא לסיום הפונקציה GetGradesNum ולהחזרת הערך שבמשתנה count ממנה.

```
int GetGrade()
{
    /* ... */
}
```

זוהי הגדרת הפונקציה GetGrade, הקולטת מהמשתמש ציון בודד ומחזירה אותו.

```
    int grade;

    printf("Please enter a grade: ");
    scanf("%d", &grade);
```

אלו הן ההוראות השייכות לפונקציה GetGrade, ולאחר ביצוען יוצב במשתנה grade ציון שמזין המשתמש.

```
    return grade;
```

משפט return זה מביא לסיום הפונקציה GetGrade ולהחזרת הערך שבמשתנה grade ממנה.

```
void main()
{
    int grades_num, i, average = 0;

    /* ... */
```

שורות אלו מהוות את ראשיתה של הגדרת הפונקציה main. במשתנה grades_num יוצב מספר הציונים שיש לקלוט. המשתנה i הוא משתנה מונה בו נשתמש כדי לקלוט את הציונים. המשתנה average יכיל בתחילה את סכום הציונים שיש לקלוט. כדי לחשב את ממוצע הציונים נחלק את ערך המשתנה average, המכיל את סכומם, בערך המשתנה grades_num, המכיל את מספרם.

```
    grades_num = GetGradesNum();
```

בשורה זו נקראת הפונקציה GetGradesNum. פונקציה זו מחזירה את מספר הציונים שיש לקלוט, וערך זה מוצב במשתנה grades_num.

```
    for (i = 0 ; i < grades_num ; i++)
        average += GetGrade();
```

לולאה זו מתבצעת grades_num פעמים, כלומר, מספר פעמים השווה למספר שהזין המשתמש במהלך ביצוע הפונקציה GetGradesNum. בכל איטרציה של הלולאה מתבצעת קריאה לפונקציה GetGrade, הקולטת ציון בודד מהמשתמש ומחזירה אותו. הערך המוחזר ממנה, כלומר, הציון האחרון שהוזן, מתוסף לערך המשתנה average.

```
    average /= grades_num;
    printf("Average of grades: %d \n", average);
```

לאחר ביצוע הלולאה יכיל המשתנה average את סכום הציונים שהוזנו. עתה עלינו לחלקו במספר הציונים שהוזנו כדי למצוא את הממוצע שלהם. מספר הציונים שהוזנו מוצב במשתנה grades_num. לאחר החלוקה נפלוט את ערך average על המסך.

בדוגמה זו נקרא לפונקציה שגם מקבלת פרמטר וגם מחזירה ערך. פונקציה זו תקבל מספר כפרמטר ותחזיר את הערך 1 אם הוא מספר ראשוני ואת הערך 0 אם הוא מספר פריק. נשתמש בפונקציה זו כדי לפלוט על המסך את n המספרים הראשוניים הראשונים, כאשר n הוא מספר שמזין המשתמש.

המספר 1 אינו מספר ראשוני, ולכן נתעלם ממנו בתוכניתנו. כלומר, פליטת המספרים הראשוניים תתחיל במספר הראשוני 2.

תוכנית 8.6

IS_PRIME.C

```
#include <stdio.h>
#include <math.h>

int IsPrime(int num)
{
    int i;

    for (i = 2 ; i <= sqrt(num) ; i++)
        if (num % i == 0)
            return 0;     /* num is reducible, so return 0 */

    return 1;     /* num is prime, so return 1 */
}

void main()
{
    int i, prime_count;

    printf("How many prime numbers would you like to see? ");
    scanf("%d", &prime_count);

    i = 2;
    while (prime_count > 0) {
        if (IsPrime(i) == 1) {
            printf("%d ", i);
            prime_count--;
        }
        i++;
    }

    printf("\n");
}
```

פלט לדוגמה מהתוכנית מוצג להלן:

```
How many prime numbers would you like to see? 8
2 3 5 7 11 13 17 19
```

```
#include <math.h>
```

הוספנו שורה זו לתוכנית משום שאנו קוראים לפונקציה sqrt ממנה.

```
int IsPrime(int num)
{
    /* ... */
}
```

זוהי הגדרת הפונקציה IsPrime, המקבלת מספר כפרמטר ומחזירה את הערך 1 אם הוא מספר ראשוני ואת הערך 0 אם הוא מספר פריק.

```
    for (i = 2 ; i <= sqrt(num) ; i++)
        if (num % i == 0)
            return 0;      /* num is reducible, so return 0 */
```

לולאה זו בודקת האם קיים מספר i בו num מתחלק ללא שארית. הלולאה מתבצעת כאשר ערך i עולה מ-2 לשורש הריבועי של num, משום שכל מספר מתחלק ב-1 ללא שארית, ואף מספר לא מתחלק במספר הגדול משורשו הריבועי ללא שארית. אם נמצא מספר בו num מתחלק ללא שארית, אזי num מספר פריק, ומשפט return מסיים את הפונקציה ומחזיר ממנה את הערך 0.

```
    return 1;      /* num is prime, so return 1 */
```

משפט זה יבוצע רק אם משפט return 0; לא התבצע, כלומר, אם לא נמצא מספר בו num מתחלק ללא שארית. במקרה זה, num הוא מספר ראשוני, ומשפט return שלעיל מסיים את הפונקציה ומחזיר ממנה את הערך 1.

```
void main()
{
    int prime_count, i;

    /* ... */
```

שורות אלו מהוות את ראשיתה של הפונקציה main. המשתנה prime_count יכיל את מספר המספרים הראשוניים שיש להציג. מספר זה יוזן על ידי המשתמש. המשתנה i הוא משתנה מונה, בו נשתמש כדי להציג את המספרים הראשוניים.

```
    printf("How many prime numbers would you like to see? ");
    scanf("%d", &prime_count);
```

בשורות אלו מתבקש המשתמש להזין כמה מספרים ראשוניים ברצונו שייפלטו.

```
    while (prime_count > 0) {
        /* ... */
```

לולאה זו מתבצעת כל עוד ערך המשתנה prime_count גדול מ-0. כפי שנראה מייד, הערך של prime_count פוחת ב-1 עבור כל מספר ראשוני שנפלט. באופן זה הלולאה תתבצע עד שמספר המספרים הראשוניים שייפלטו ישתווה לערך שהזין המשתמש.

```
        if (IsPrime(i) == 1) {
            /* ... */
```

ביטוי התנאי IsPrime(i) == 1 קורא לפונקציה IsPrime ושולח אליה את המספר i. כאשר הפונקציה מסתיימת, מושווה הערך המוחזר ממנה ל-1. אם i הוא מספר ראשוני, הפונקציה תחזיר את הערך 1, וביטוי התנאי יתקיים. אם i הוא מספר פריק, הפונקציה תחזיר את הערך 0, וביטוי התנאי לא יתקיים.

```
            printf("%d ", i);
            prime_count--;
```

משפטים אלה יתבצעו אם ביטוי התנאי IsPrime(i) == 1 מתקיים, כלומר, אם i הוא מספר ראשוני. במקרה זה ייפלט הערך של המשתנה i והערך של המשתנה prime_count, המייצג את מספר המספרים הראשוניים שנפלטו, יפחת ב-1.

```
    i = 2;
    while (primes_count > 0) {
        /* ... */
        i++;
    }
```

בכל איטרציה של לולאת while נבדק האם i הוא מספר ראשוני. ברצוננו לבדוק את ראשוניותם של כל המספרים הטבעיים החל מ-2, ולכן מוצב הערך 2 במשתנה i לפני הכניסה ללולאה. כמו-כן, לאחר כל איטרציה של הלולאה עולה הערך של i ב-1, ובאופן זה נבדקת בכל איטרציה ראשוניותו של מספר טבעי אחר.

שימושים נוספים במשפט return

במשפט return ניתן להשתמש גם מתוך פונקציות שאינן מחזירות ערכים. כאשר נעשה זאת, הפונקציה תסתיים אך לא יוחזר ממנה כל ערך. משפט return המופיע בפונקציה שאינה מחזירה ערך ייכתב ללא ערך לצדו, כך:

```
return;     /* exit a function that returns no value */
```

איך לא...

אין לנסות להחזיר ערך באמצעות משפט return מתוך פונקציה שאינה מחזירה ערך. למשל, קטע התוכנית הבא אינו נכון:

```
void MyFunction()
{
    return 5; /* wrong, returned value isn't allowed */
}
```

הנה דוגמה לפונקציה שאינה מחזירה ערך המשתמשת במשפט זה:

```
void PrintSquareRoot(float n)
{
    if (n < 0.0)      /* if n is negative */
        return;       /* quit the function */
                      /* otherwise, print its square root */
    printf("Square root of %f: %f \n", n, sqrt(n));
}
```

פונקציה זו מקבלת פרמטר ופולטת את שורשו הריבועי. כזכור, אין לחשב את שורשו הריבועי של מספר שלילי, ולכן משפט התנאי מביא לסיום הפונקציה אם הפרמטר שלילי. אם הפרמטר אינו שלילי, לא יתקיים משפט התנאי, ומשפט הפלט יציג את שורשו הריבועי.

איך לא...

אין לבצע משפט return בלי לציין ערך מתוך פונקציה שמחזירה ערך כלשהו. למשל, קטע התוכנית הבא אינו נכון:

```
int MyFunction()
{
    return;      /* wrong, returned value needed */
}
```

אם הערך שפונקציה מחזירה אינו מתאים לסוג הערך שהמהדר מצפה ממנה להחזיר, תתבצע המרה בהתאם. ערך זה יכול להיות ערך המוחזר מביטוי או מפונקציה אחרת. לדוגמה:

```
return a > 0 && a < 100;    /* returns a logical value (1 or 0) */
return i++;                 /* returns the value of i */
return AnotherFunction();   /* calls the function AnotherFunction
                               and returns the value it returns */
```

במחשבה שנייה...

האם חובה להשתמש בערך המוחזר מפונקציות?

כפי שאין צורך להשתמש בערך המוחזר מביטוי, ניתן להתעלם מהערך שפונקציה מחזירה. למשל, לפונקציה IsPrime המוגדרת בתוכנית 8.6 ניתן לקרוא כך:

```
IsPrime(i);      /* call IsPrime but ignore
                    the value it returns    */
```

כמעט בכל המקרים שקראנו לפונקציות עד כה התעלמנו מהערכים שהן מחזירות. רוב פונקציות הספרייה, כגון printf, scanf ו-gets, מחזירות ערכים, אך ברוב המקרים לא נשתמש בהם.

הצהרת הפונקציה

בתחילת הפרק ציינו שהגדרת פונקציה הקוראת לפונקציה אחרת צריכה להופיע אחרי הגדרת הפונקציה האחרת. כלומר, אם פונקציה בשם MyFunction קוראת לפונקציה AnotherFunction, עליהן להופיע בסדר הבא:

```
void AnotherFunction()
{
    /* ... */
}

void MyFunction()
{
    AnotherFunction();
}
```

ולא כך:

```
void MyFunction()
{
    AnotherFunction();      /* wrong */
}

void AnotherFunction()
{
    /* ... */
}
```

הסברנו זאת בכך שהמהדר בודק את התוכנית מלמעלה למטה, ולפני שהוא נתקל בקריאה לפונקציה כלשהי, עליו לדעת מהם מאפייניה. מאפיינים אלה נקבעים בשורת ההגדרה של הפונקציה, ולכן יש להגדירה לפני שמתבצעת קריאה אליה.

בתוכניות שכתבנו עד כה, כאשר נקראה פונקציה אחת מתוך פונקציה שנייה, הוגדרה הראשונה לפני השנייה. אך במקרים מסוימים לא נוכל לעמוד בדרישה זו. הנה מספר דוגמאות למקרים כאלה:

- כאשר התוכנית מורכבת ממספר קבצים.
- כאשר שתי פונקציות קוראות זו לזו.
- כאשר נרצה להגדיר את הפונקציות בסדר שאינו תלוי בסדר בן הן נקראות, כדי להקל על הבנת התוכנית.

נניח שבידינו פונקציה בשם MyFunction, שברצוננו להגדיר **לאחר** פונקציה אחרת שקוראת לה. במקרה זה, נכתוב עבור הפונקציה MyFunction שורה מיוחדת, הנקראת **הצהרת הפונקציה** (function declaration) או **אבטיפוס הפונקציה** (function prototype). שורה זו תכיל את כל מאפייני הפונקציה (כלומר, את סוג הערך המוחזר ממנה, את שמה ואת פרטי הפרמטרים שהיא מקבלת), אך לא את גופה. כאשר המהדר ייתקל בה, היא תיידע אותו לגבי מאפייני הפונקציה.

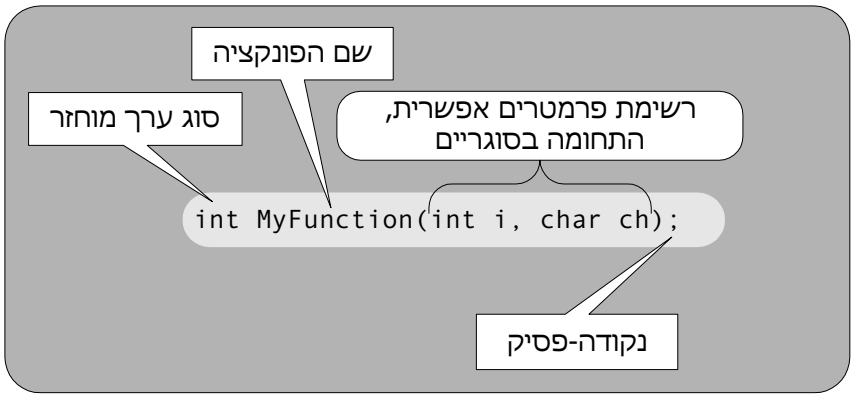
הצהרת הפונקציה נכתבת בדומה לשורת ההגדרה של הפונקציה. למשל, על הפונקציה המוגדרת כך:

```
int MyFunction(int i, char ch)     /* definition of MyFunction */
{
    /* ... */
}
```

נצהיר באופן הבא:

```
int MyFunction(int i, char ch);    /* declaration of MyFunction */
```

האיור הבא מדגים זאת:



איור 8.7 – *הצהרה על פונקציה*

שים לב!

הצהרת הפונקציה חייבת להתאים לשורת ההגדרה של הפונקציה, כלומר, מאפייני הפונקציה צריכים להיות זהים בשתיהן. ההבדל היחיד הקיים בין השתיים הוא שהצהרת הפונקציה מסתיימת בתו נקודה-פסיק.

נהוג לכתוב את הצהרת הפונקציה בראש התוכנית, מייד לאחר הוראות #include. באופן זה, המהדר ייתקל בה לפני שיקרא את יתר שורות התוכנית, וכבר בשלב זה ידע את מאפייני הפונקציה. עתה נוכל לקרוא לפונקציה מכל פונקציה אחרת בתוכנית, ונוכל להגדירה לפני או אחרי אילו פונקציות שנרצה. לכן, קטע התוכנית הבא הינו חוקי לגמרי:

```
int MyFunction(int i, char ch);     /* declaration */

void AnotherFunction()
{
    MyFunction(10, 'a');     /* correct statement because of
                                the declaration of MyFunction */
}

int MyFunction(int i, char ch)      /* definition */
{
    /* ... */
}
```

בדרך-כלל נצהיר על כל פונקציה שאנו מגדירים, אפילו אם היא אינה נקראת מפונקציה שמוגדרת לפניה. נוהג זה מאפשר לקורא התוכנית לסקור בקצרה את פונקציות התוכנית, ואם מאוחר יותר נרצה לשנות את סדר הגדרתן או להוסיף קריאות להן ממקומות שונים בתוכנית, נוכל לעשות זאת בנקל.

שים לב!

ניתן להצהיר על פונקציה מספר פעמים (נראה בהמשך מקרים בהם הדבר הכרחי), אך הגדרת הפונקציה, כלומר, ציון ההוראות השייכות לה בין סוגריים מסולסלים, חייבת להיעשות פעם אחת ויחידה עבור כל פונקציה.

דוגמה

התוכנית הבאה שנכתוב תבצע פעולה זהה לזו שביצעה תוכנית 8.2, אך היא תעשה שימוש בהצהרות פונקציות. כזכור, תוכנית 8.2 קלטה מספר מהמשתמש ופלטה מלל מסוים מספר פעמים השווה לו.

```
#include <stdio.h>

void PrintNewLine();        /* declaration */
void PrintMessage();        /* declaration */
void ReadNumber();          /* declaration */

void main()
{
    ReadNumber();           /* invoke ReadNumber */
}

void ReadNumber()
{
    int n;

    printf("How many times would you like to see the output? ");
    scanf("%d", &n);

    while (n > 0) {
        PrintMessage();     /* invoke PrintMessage */
        n--;
    }
}

void PrintMessage()
{
    printf("Hello, and welcome to my program.");
    PrintNewLine();         /* advance to new line */
    printf("Creator: Mr. Unknown, 1999-2000.");
    PrintNewLine();         /* advance to new line */
    printf("Telephone number: 555-1234, LA, USA.");
    PrintNewLine();         /* advance to new line */
    PrintNewLine();         /* advance to new line */
}

void PrintNewLine()
{
    printf("\n");
}
```

הפלט המתקבל מהרצת תוכנית זו זהה לפלט המתקבל מהרצת תוכנית 8.2.

```
void PrintNewLine();        /* declaration */
void PrintMessage();        /* declaration */
void ReadNumber();          /* declaration */
```

בשורות אלו מוצהרות הפונקציות PrintNewLine, PrintMessage ו-ReadNumber. מאחר שהצהרנו עליהן בראשית התוכנית, ניתן לקרוא להן מתוך כל פונקציה שהיא, ואין חשיבות לסדר בו הן מוגדרות.

אין חשיבות לסדר בו מוצהרות הפונקציות; בתוכנית זו, למשל, הצהרת הפונקציה ReadNumber יכלה להופיע לפני הצהרת הפונקציה PrintNewLine.

במחשבה שנייה...

מדוע לא הצהרנו על הפונקציה main?

הפונקציה main היא הפונקציה היחידה עליה לא מקובל להצהיר. הסיבה לכך היא, שרק במקרים נדירים ביותר היא נקראת מתוך פונקציה אחרת (נרחיב אודות נושא זה בהמשך).

```
void main()
{
    /* ... */
}

void ReadNumber()
{
    /* ... */
}

void PrintMessage()
{
    /* ... */
}

void PrintNewLine()
{
    /* ... */
}
```

שים לב כי הסדר בו מוגדרות הפונקציות שונה מהסדר בו הן הוגדרו בתוכנית 8.2. הסיבה לכך נעוצה בהצהרות המופיעות בראש התוכנית, המאפשרות לנו להגדיר את הפונקציות באיזה סדר שנחפוץ, בלי תלות בסדר בו הפונקציות קוראות זו לזו.

פונקציות ומשתנים

עד לפרק זה הוגדרו כל המשתנים בפונקציה main. אך כבר מתחילת הפרק ראינו כי ניתן להגדיר משתנים בפונקציות אחרות. בסעיף זה נעסוק בקשר של משתנים עם הפונקציות בהן הם מוגדרים.

כזכור, משתנים לוקליים הם משתנים המוגדרים בתוך גוף פונקציה כלשהי. למשל, הפונקציה הבאה:

```
void MyFunction()
{
    int i;      /* i is a local variable */
}
```

מכילה משתנה לוקלי בשם i. נהוג לומר כי המשתנה i הוא לוקלי לפונקציה MyFunction. למעשה, כל המשתנים בהם השתמשנו עד כה היו משתנים לוקליים (בין אם הם הוגדרו בפונקציה main ובין אם בפונקציה אחרת).

משתנה לוקלי, כמו כל משתנה בשפת C, תופס שטח כלשהו בזיכרון המחשב. השטח שתופס משתנה לוקלי מוקצה כאשר נקראת הפונקציה בה הוא מוגדר, והוא משתחרר כאשר הפונקציה מסתיימת. פעולות הקצאת השטח ושחרורו תבוצענה מחדש עבור כל קריאה לפונקציה.

מאחר שמשתנה לוקלי קיים בזיכרון רק בזמן ביצוע הפונקציה בה הוא מוגדר, לא ניתן לפנות אליו מחוץ לפונקציה זו. כלומר, כל משתנה לוקלי שייך לפונקציה בה הוא מוגדר – ורק לה. לכן, קטע התוכנית הבא שגוי:

```
void MyFunction()
{
    int i;
}

void AnotherFunction()
{
    i = 5;      /* wrong, i does not exist here! */
}
```

הסיבה היא, שהמשתנה i מוגדר בפונקציה MyFunction, ולא ניתן לפנות אליו מגוף הפונקציה AnotherFunction. נהוג לומר כי המשתנה i לוקלי לפונקציה MyFunction.

מכיון שכל משתנה לוקלי שייך רק לפונקציה בה הוא מוגדר, ניתן להגדיר משתנים לוקליים בעלי אותו שם בפונקציות שונות, והם ייחשבו למשתנים נפרדים.

לכן, אם נקרא בזו אחר זו לשתי הפונקציות הבאות:

```
void First()
{
    int n;      /* this n is local to First */
    n = 5;      /* assign value to the variable local to First */
}

void Second()
{
    int n;                /* this n is local to Second */
    printf("%d", n);     /* print the variable local to Second */
}
```

ייפלט ערך בלתי צפוי. הסיבה היא, שמשפט ההשמה משפיע על המשתנה n שמוגדר בפונקציה First בלבד. המשתנה n שמוגדר בפונקציה Second שונה מהמשתנה n שמוגדר בפונקציה First, והוא אינו מושפע ממשפט השמה זה. משפט הפלט פולט את ערכו של n השייך לפונקציה Second בלי שהוצב בו ערך כלשהו, ולכן לא ניתן לצפות את הפלט שיתקבל.

דוגמה

בדוגמה זו נראה כיצד מתייחס המהדר אל משתנים לוקליים ופורמליים בשמות זהים המוגדרים בפונקציות שונות. התוכנית שנכתוב תקלוט מהמשתמש מספר חד-ספרתי ותפלוט את מאה המספרים הטבעיים הראשונים שאינם מתחלקים בו ללא שארית ושאף אחת מספרותיהם אינה שווה לו. ניעזר בפונקציה המקבלת את המספר החד-ספרתי ומספר טבעי, ומחזירה את הערך 1 אם המספר הטבעי מתחלק במספר החד-ספרתי ללא שארית או אם אחת מספרות המספר הטבעי שווה למספר החד-ספרתי.

שים לב!

על רעיון זה מבוסס המשחק "1-2-3-בום", בו כל שחקן אומר בקול רם את המספר הטבעי שבא אחרי המספר הטבעי שאמר השחקן הקודם. שחקן שהמספר אליו הגיע מתחלק ב-3 ללא שארית או שאחת מספרותיו היא 3, צריך לומר "בום" במקום את המספר עצמו. שחקן שלא אמר "בום" כשהיה צריך, או שאמר "בום" שלא לצורך, נפסל.

תוכנית 8.8 *BOOM.C*

```
#include <stdio.h>

int IsBoom(int i, int boom);

void main()
{
    int i, boom;

    printf("Please enter a 1-digit number: ");
    scanf("%d", &boom);

    for (i = 1 ; i <= 100 ; i++)
        if (IsBoom(i, boom) == 1)
            printf("X\t");
        else
            printf("%d\t", i);
    printf("\n");
}

int IsBoom(int i, int boom)
{
    if (i % boom == 0)
        return 1;

    while (i > 0) {
        if (i % 10 == boom)
            return 1;
        i /= 10;
    }

    return 0;
}
```

פלט חלקי המתקבל מהרצת תוכנית זו מוצג להלן (הרווחים שלהלן אינם זהים לרווחי הפלט שיתקבל במחשב):

```
Please enter a 1-digit number: 3
1       2       X       4       5       X       7       8       X       10
11      X       X       14      X       16      17      X       19      20
X       22      X       X       25      26      X       28      29      X
..              ..              ..              ..              ..
        ..              ..              ..              ..              ..
```

```
int IsBoom(int i, int boom);

void main()
{
    /* ... */
}

int IsBoom(int i, int boom)
{
    /* ... */
}
```

הצהרת הפונקציה IsBoom מאפשרת לפונקציה main לקרוא לפונקציה IsBoom, אף ש-IsBoom מוגדרת אחרי main.

```
void main()
{
    int i, boom;

    printf("Please enter a 1-digit number: ");
    scanf("%d", &boom);

    /* ... */
```

אלו הן ההוראות השייכות לפונקציה main, ולאחר ביצוען ייקלט מספר לתוך המשתנה boom. זהו המספר החד-ספרתי שיקבע אילו מספרים טבעיים ייפלטו בהמשך.

```
    for (i = 1 ; i <= 100 ; i++)
        if (IsBoom(i, boom) == 1)
            printf("X\t");
        else
            printf("%d\t", i);
```

בלולאה זו עולה ערך המשתנה i מ-1 ל-100 בהפרשים של 1. משפט התנאי השייך ללולאה קורא לפונקציה IsBoom ושולח אליה שני פרמטרים, i ו-boom. הפרמטר i הוא מספר טבעי כלשהו בין 1 ל-100, והפרמטר boom הוא המספר החד-ספרתי שהזין המשתמש.

הפונקציה IsBoom תחזיר את הערך 1 אם הפרמטר i מתחלק בפרמטר boom ללא שארית או אם אחת מספרות הפרמטר i היא הפרמטר boom. במקרה זה אין להציג את המספר i, ובמקום זאת נפלט התו 'X'. אם הפונקציה IsBoom החזירה ערך השונה מ-1, אזי הפרמטר i אינו מתחלק בפרמטר boom ללא שארית ואף אחת מספרותיו של הפרמטר i אינה הפרמטר boom. במקרה זה יש לפלוט את המספר i.

משפט התנאי מתבצע עבור כל ערך של i, ובאופן זה נפלטים כל המספרים השלמים בין 1 ל-100 שאינם מתחלקים ב-boom ללא שארית ושאף אחת מספרותיהם אינה boom.

```
int IsBoom(int i, int boom)
{
    /* ... */
```

זוהי תחילתה של הגדרת הפונקציה IsBoom. שים לב כי שמות הפרמטרים שהיא מקבלת, i ו-boom, זהים לשמות המשתנים הלוקליים שהוגדרו בפונקציה main. כפי שכל משתנה לוקלי שייך אך ורק לפונקציה בה הוא מוגדר, כך גם משתנים פורמליים מוגדרים בנפרד עבור כל פונקציה. לכן המשתנים הפורמליים i ו-boom נחשבים למשתנים שונים מהמשתנים הלוקליים i ו-boom המוגדרים בפונקציה main.

```
    if (i % boom == 0)
        return 1;
```

משפט תנאי זה מסיים את הפונקציה ומחזיר ממנה את הערך 1 אם המשתנה i מתחלק במשתנה boom ללא שארית.

```
    while (i > 0) {
        if (i % 10 == boom)
            return 1;
        i /= 10;
    }
```

לולאה זו בודקת האם אחת מספרותיו של המספר i היא המספר boom. ביטויו של משפט התנאי מתקיים אם ספרתו הימנית ביותר של i, המתקבלת מחישוב שארית חלוקתו ב-10, היא המספר boom. במקרה זה מסתיימת הפונקציה ומוחזר ממנה הערך 1. אם ביטוי התנאי לא התקיים, יש לבדוק את הספרה הבאה במספר i. ספרה זו היא הספרה הנמצאת משמאל לספרה האחרונה שבדקנו, והיא תיבדק באיטרציה הבאה אם נשמיט את ספרתו הימנית ביותר של המספר i. זאת נעשה על ידי חילוק ערך המשתנה i ב-10.

```
    return 0;
```

משפט זה יתבצע אם לא התבצעו משפטי return הקודמים, כלומר, אם המספר i אינו מתחלק במספר boom ללא שארית ואם אף אחת מספרות המספר i היא המספר boom. במקרה זה יש לפלוט את המספר על המסך, ולכן מוחזר הערך 0.

אופני הגדרה והצהרה נוספים

נדון עתה במספר אופנים נוספים בהם ניתן להגדיר פונקציות ולהצהיר עליהן.

עצה

אופני ההגדרה וההצהרה שנציג להלן היו מקובלים עד שנקבע תקן ANSI C, בו הוכר אבטיפוס הפונקציה כחלק מהשפה. השימוש באופנים אלה נדיר ביותר, אך יתכן שתיתקל בו במהלך קריאת תוכניות שנכתבו על ידי מתכנתי C אחרים. בתוכניות שאתה כותב, הימנע מסגנונות הכתיבה המתוארים בסעיף זה.

השמטת הערך המוחזר

אם נגדיר או נצהיר על פונקציה בלי לציין את סוג הערך אותו היא מחזירה, היא תיחשב כפונקציה המחזירה ערך מסוג int. לכן, את שורות הקוד הבאות, המצהירות ומגדירות פונקציה המחזירה ערך מסוג int:

```
int MyFunction();      /* declaration */

int MyFunction()       /* definition */
{
    /* ... */
}
```

ניתן להחליף בשורות הקוד הבאות:

```
MyFunction();      /* declaration */

MyFunction()       /* definition */
{
    /* ... */
}
```

עצה

גם אם סוג הערך המוחזר מפונקציה הינו int, מומלץ לציין זאת במפורש בהצהרה ובהגדרה שלה.

השמטת שמות הפרמטרים בהצהרת הפונקציה

כאשר המהדר נתקל בהצהרה על פונקציה המקבלת פרמטרים, הוא אינו דורש לדעת את שמותיהם, אלא רק את סוגם. לכן, הצהרת הפונקציה הבאה:

```
float GetValue(char ch, int i, float f);
```

ניתנת לכתיבה גם כך:

```
float GetValue(char, int, float);    /* specifying types only */
```

סגנון כתיבה זה אינו מקובל, ואנו נעדיף להשתמש בשורות הצהרה מלאות, הכוללות גם את שמות המשתנים.

איך לא...

בהגדרת הפונקציה חובה לציין את שמות הפרמטרים, שכן אחרת לא נוכל לגשת אליהם. לכן, הגדרת הפונקציה הבאה אינה נכונה:

```
void MyFunction(char, int)      /* wrong */
{
    /* ... */
}
```

הסגנון המסורתי להגדרה ולהצהרה על פונקציה

קיימת דרך נוספת להגדרה ולהצהרה על פונקציות, שהיתה מקובלת לפני שנקבע תקן ANSI C. על-פיה, את שורות ההצהרה וההגדרה הבאות:

```
void MyFunction(float a, float b);  /* declaration */

void MyFunction(float a, float b)   /* definition */
{
    /* ... */
}
```

יש לכתוב כך:

```
void MyFunction();      /* declaration */

void MyFunction(a, b)      /* definition */
float a;
float b;
{
    /* ... */
}
```

על-פי דרך זו, גם אם הפונקציה מקבלת פרמטרים, אין לציינם בהצהרה עליה. בשורת ההגדרה של הפונקציה ייכתבו הפרמטרים פעמיים: בתוך הסוגריים ייכתבו רק שמות המשתנים, ואילו אחריהם, אך לפני גוף הפונקציה, יצוינו סוגי המשתנים ושמותיהם, מופרדים זה מזה על ידי נקודה-פסיק.

במחשבה שנייה...

מהו ההבדל בין שני סגנונות ההגדרה וההצהרה הללו?

לאחר שהמהדר בודק הצהרת פונקציה בה לא מצוינים הפרמטרים, הוא אינו יודע כמה פרמטרים היא מקבלת. לכן, אם נקרא לפונקציה זו ולא נשלח אליה אף פרמטר, או אם נקרא לה ונשלח אליה עשרים פרמטרים, לא תיפלט הודעת שגיאה – אך התוכנית לא תפעל כמצופה.

בנוסף, המהדר יניח שכל הפרמטרים הנשלחים אל הפונקציה הם מסוג int, וכזהו גם הערך אותו היא מחזירה. אם נשלח אל הפונקציה פרמטרים שאינם מסוג int, או אם הפונקציה תנסה להחזיר ערך שאינו מסוג int, לא תפעל התוכנית כמצופה.

מכיון שסגנון ההגדרה וההצהרה המיושן עלול לגרום לתקלות, לא נהוג להשתמש בו.

סיכום

- נהוג להשתמש בפונקציות כדי לחלק מטלה ארוכה לתת-מטלות קצרות יותר.
- כדי ליצור פונקציה משלנו עלינו להגדיר אותה. ההגדרה מורכבת משורת הגדרה ומגוף הפונקציה.
- שורת ההגדרה מיידעת את המהדר על מאפייני הפונקציה: סוג הערך שהיא מחזירה, שמה והפרמטרים שהיא מקבלת.
- גוף הפונקציה מורכב ממספר הוראות התחומות בסוגריים מסולסלים.
- לפני שפונקציה נקראת על המהדר להכיר את מאפייניה. פונקציה שקוראת לפונקציה אחרת תוגדר אחריה.
- כאשר הפונקציה נקראת ההוראות שבגוף שלה מבוצעות. כאשר פונקציה מסתימת, מבוצעת ההוראה המופיעה לאחר ההוראה שקראה לה.
- כאשר נקרא לפונקציה המקבלת פרמטרים נשלח אליה ערכים מתאימים. ערכים אלה יוצבו במשתנים המתאימים שהוגדרו בשורת ההגדרה של הפונקציה.
- פונקציה יכולה להחזיר ערך על ידי המשפט return.
- שורת הצהרה של פונקציה זהה לשורת ההגדרה שלה פרט לכך שהיא מסתיימת בנקודה-פסיק.
- כאשר קיימות הצהרות על כל הפונקציות בתוכנית, אין חשיבות לסדר בו הן מוגדרות.

פרק 9

מערכים

עד עתה הכרנו מספר מרכיבים בשפת C ופיתחנו באמצעותם תוכניות שימושיות רבות. מרכיב חשוב לתוכניות רבות הוא היכולת לקלוט מהמשתמש נתונים רבים ולעבד אותם בדרכים שונות. בפרק זה נכיר את המערכים, המאפשרים לנו לעשות זאת בפשטות וביעילות.

הצורך במערכים

לעיתים קרובות עלינו לטפל בנתוני קלט רבים. אם, למשל, ברצוננו לקלוט עשרה ערכים מהמשתמש ולחשב את סכומם, נוכל לעשות זאת בקלות על ידי לולאה כזו:

```
int i, num, sum = 0;
for (i = 0 ; i < 10 ; i++) {
    scanf("%d", &num);
    sum += num;
}
```

עתה, נניח שבהמשך התוכנית עלינו לפנות שוב אל הערכים שהוזנו (למשל, כדי להציגם על המסך). במקרה זה יהיה עלינו למצוא תחליף ללולאה, משום שבכל איטרציה שלה נמחק הערך הקודם שהוזן. בלית-ברירה, ניאלץ להגדיר עשרה משתנים שונים, לקלוט ערכים לתוכם בנפרד ולחשב את סכומם, כך:

```
int num1, num2, num3, num4, num5, num6, num7, num8, num9,
    num10, sum = 0;
scanf("%d%d%d%d%d%d%d%d%d%d", &num1, &num2, &num3, &num4,
    &num5, &num6, &num7, &num8, &num9, &num10);
sum = num1 + num2 + num3 + num4 + num5 + num6 + num7 +
    num8 + num9 + num10;
```

באופן זה, הטיפול בערכי המשתנים הוא מגושם ביותר – אך זהו האופן היחיד בו ניתן לפנות בבטחה אל כל הערכים שהוזנו. הסרבול גובר כאשר עלינו לקלוט מספר רב יותר של ערכים, למשל, 20, 30 ואף 100.

כאשר התוכנית מכילה משתנים רבים בעלי מאפיין משותף, נוכל (ונרצה) להחליפם בטיפוס נתונים בודד, שיכיל את כל ערכיהם ויאפשר לנו לפנות אליהם בקלות. בשפת C, זהו ה**מערך** (array).

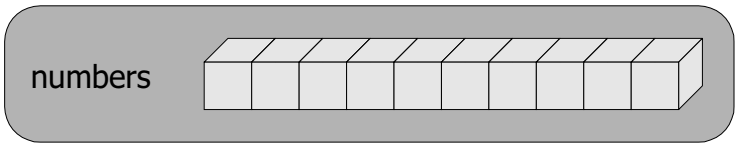
מערכים חד-מימדיים

מערך הוא משתנה המכיל מספר **איברים** (elements). לכל איבר ערך משלו, וניתן לפנות אליו כאילו היה משתנה רגיל. כל איברי המערך חייבים להיות מאותו הסוג (למשל, שלמים, עשרוניים או תווים), וזהו גם סוג המערך. כמו לכל משתנה, גם למערך יש שם, גם הוא תופס שטח בזיכרון וגם אותו יש להגדיר לפני השימוש.

הגדרת מערך זהה להגדרת משתנה רגיל, פרט לכך שיש לציין את מספר איבריו בסוגריים מרובעים לאחר שמו. למשל, שורת הגדרה כזו:

```
int numbers[10];
```

מגדירה מערך בשם numbers, המכיל עשרה איברים שלמים. מערך זה הוא משתנה לכל דבר; למשל, על שורת הגדרתו להופיע לפני כל משפט אחר, אין לקרוא לו בשמות של משתנים אחרים, וכדומה. אם נרצה לשרטט את איברי המערך, נפרוש אותם על-פני שורה כך:



איור 9.1 *– איברי מערך חד-מימדי בן עשרה איברים*

מערך כגון numbers, שפרישת איבריו יוצרת שורה בודדת, מכונה **מערך חד-מימדי** (one-dimensional array).

כדי להגדיר מערך בן מאה איברים עשרוניים בשם values, נכתוב שורת הגדרה כזו:

```
float values[100];
```

ניתן להגדיר מערכים ומשתנים רגילים באותה שורה. למשל:

```
char letters[26], ch, digits[10], div = '/';
```

איך לא...

גודל המערך חייב להיות מספר טבעי וקבוע. לכן אין להגדיר מערך בגודל n איברים (כאשר n הוא משתנה), כך:

```
int n = 10;
int ri[n];      /* wrong, constant value required
*/
```

ההתייחסות אל כל איבר במערך היא כאל משתנה רגיל, כלומר, ניתן להציב בו ערכים ולקרוא ממנו ערכים. כדי לפנות אל איבר מסוים במערך, נכתוב את שם המערך, ובסוגריים מרובעים את מיקומו של האיבר במערך. מיקום זה נקרא **אינדקס** (index), והוא מספר שלם המתחיל מ-0. כלומר, במערך בעל n איברים, אינדקס האיבר הראשון הוא 0, אינדקס האיבר השני הוא 1, וכן הלאה – עד לאינדקס האיבר ה-n, שהוא n - 1.

לדוגמה, כדי להציב את המספר 12.3 באיבר הראשון של המערך values, נבצע משפט השמה כזה:

```
values[0] = 12.3;     /* first element <==> index 0 */
```

וכדי לפלוט את ערך האיבר השלישי במערך זה, נבצע משפט פלט כזה:

```
printf("The third element is %f", values[2]);
```

זהירות!

קל להתבלבל מהעובדה שאינדקס האיבר הראשון במערך הוא 0 ולא 1. אם נטעה ונתייחס, למשל, אל מיקום האיבר העשירי במערך כ-10, המהדר לא יזהיר אותנו. טעות כזו הינה מסוכנת ביותר, והיא עלולה אף לגרום לקריסת התוכנית ומערכת ההפעלה.

שים לב כי בכל פנייה לאיבר במערך, חובה לציין את האינדקס שלו. לכן המשפט הבא אינו חוקי:

```
values = 0.123;     /* wrong, index required! */
```

באופן דומה, אין לבצע פעולת השמה בין מערכים בלי לפנות אל איבריהם. למשל, משפט ההשמה הבא אינו נכון:

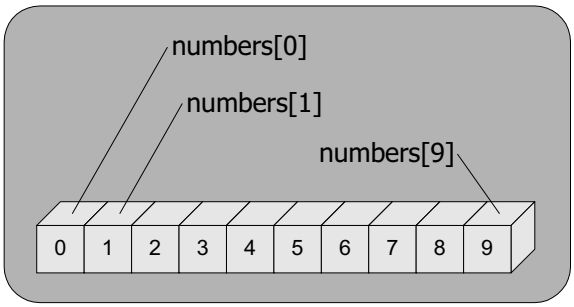
```
array2 = array1;      /* wrong! */
```

אם ברצוננו להעתיק את תוכן המערך array1 למערך array2, עלינו לבצע פעולות השמה בין כל אחד מאיבריהם.

אף שגודל המערך בשורת ההגדרה חייב להיות מספר קבוע, ניתן לפנות אל איבר במערך גם על ידי משתנה. למשל, הלולאה הבאה מציבה בכל איבר במערך numbers את ערך האינדקס שלו:

```
for (i = 0 ; i < 10 ; i++)
    numbers[i] = i;
```

האיור הבא מציג את ערכי איברי המערך numbers לאחר ביצוע הלולאה:



***איור 9.2** – איברי המערך numbers לאחר הצבת ערכים בהם*

זכור כי הפנייה לאיבר העשירי נעשית באמצעות האינדקס 9, ולא 10. אילו כתבנו את הלולאה כך:

```
for (i = 0 ; i <= 10 ; i++) /* dangerous, note the <= */
    numbers[i] = i;
```

היה מתבצע משפט ההשמה גם כאשר ערך המשתנה i הוא 10, והיתה מתבצעת פנייה לאיבר שכלל לא שייך למערך, כך:

```
numbers[10] = 10;     /* dangerous! */
```

זו פעולה מסוכנת, שעלולה לגרום לקריסת התוכנית ואף לקריסת מערכת ההפעלה.

זהירות!

המהדר לא יזהיר אותנו אם נפנה לאיבר שאינו שייך למערך.

דוגמה

נתרגל את השימוש במערכים בדוגמה הבאה, הקולטת עשרה מספרים מהמשתמש, מציגה אותם בסדר הפוך לסדר בו הוזנו ופולטת את סכומם.

תוכנית 9.1 *REVERSE.C*

```
#include <stdio.h>
void main()
{
    int values[10], i, sum = 0;

    printf("Please enter ten values: ");

    for (i = 0 ; i < 10 ; i++) {
        scanf("%d", &values[i]);
        sum += values[i];
    }

    printf("Values in reversed order: ");
    for (i = 9 ; i >= 0 ; i--)
        printf("%d ", values[i]);

    printf("\nSum of values: %d \n", sum);
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter ten values: 2 4 6 8 10 12 14 16 18 20
Values in reversed order: 20 18 16 14 12 10 8 6 4 2
Sum of values: 110
```

```
    int values[10], i, sum = 0;
```

בשורת הגדרה זו מוגדר מערך בן עשרה איברים בשם values, בו יאוחסנו הערכים שיזין המשתמש. במשתנה sum נשתמש לחישוב סכום הערכים המוזנים. המשתנה i הוא משתנה מונה פשוט.

```
    for (i = 0 ; i < 10 ; i++) {
         scanf("%d", &values[i]);
         sum += values[i];
    }
```

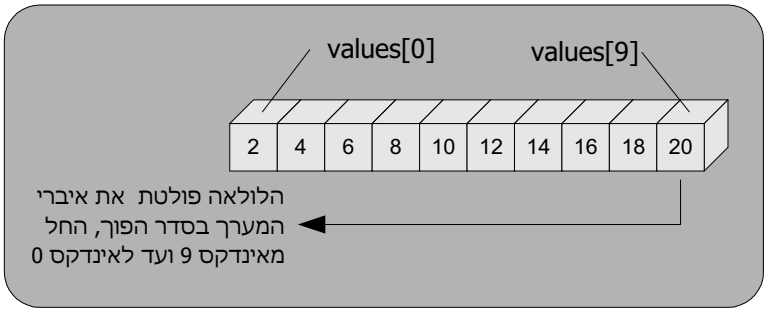
לולאה זו מתבצעת עשר פעמים, ובכל איטרציה שלה נקלט ערך מהמשתמש. כל ערך מוצב באיבר אחר במערך values, ומתוסף לערך המשתנה sum, שיכיל, עם סיום הלולאה, את סכום כל הערכים שהוזנו.

שים לב!

לאיבר בודד במערך יש להתייחס כאל משתנה רגיל. לכן, הקדמנו את שמו באופרטור הכתובת (&) כאשר שלחנו אותו לפונקציה scanf.

```
    for (i = 9 ; i >= 0 ; i--)
        printf("%d ", values[i]);
```

גם לולאה זו מתבצעת עשר פעמים, והיא פולטת את ערכי איברי המערך values מהסוף להתחלה. אינדקס האיבר האחרון במערך הוא 9, וזהו הערך המוצב במשתנה i לפני האיטרציה הראשונה. בכל איטרציה נפלט האיבר בעל האינדקס i, ובסוף כל איטרציה יורד הערך של i ב-1. באופן זה, נפלטים בסדר הפוך הערכים שהזין המשתמש. האיור הבא מדגים את פעולת הלולאה עבור הערכים המופיעים בפלט לדוגמה:



***איור 9.3** – לולאה לפליטת ערכי מערך חד-מימדי מהסוף להתחלה*

דוגמה

גם בדוגמה זו נתרגל פנייה אל איברי מערך. התוכנית שנכתוב תקלוט מהמשתמש מילה בת חמישה תווים, ותפלוט אותה תוך הפיכת אותיותיה הקטנות לגדולות ואותיותיה הגדולות – לקטנות.

תוכנית 9.2

WORDS.C

```
#include <stdio.h>
#include <conio.h>
void main()
{
    char string[5], delta;
    int i;
```

```
    printf("Please enter a 5-letter word: ");
    for (i = 0 ; i < 5 ; i++) {
        string[i] = getch();
        putch(string[i]);
    }

    delta = 'a' - 'A';

    printf("\nThe new word is: ");
    for (i = 0 ; i < 5 ; i++) {
        if (string[i] >= 'a' && string[i] <= 'z')
            string[i] -= delta;
        else if (string[i] >= 'A' && string[i] <= 'Z')
            string[i] += delta;
        printf("%c", string[i]);
    }
    printf("\n");
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter a 5-letter word: HappY
The new word is: hAPPy
```

שים לב!

אם יוזן תו שאינו אות, הוא לא ישתנה. למשל, אם נזין את התווים הבאים:

```
a^8^b
```

ייפלטו התווים:

```
A^8^B
```

```
    char string[5], delta;
    int i;
```

בשורת ההגדרה הראשונה מוגדר מערך תווי בן חמישה איברים, בו יוצבו התווים שיזין המשתמש. את תפקיד המשתנה delta נסביר מייד. במשתנה השלם i נשתמש כמונה ללולאה הקולטת את התווים.

```
    for (i = 0 ; i < 5 ; i++) {
        string[i] = getch();
        putch(string[i]);
    }
```

לולאה זו קולטת את חמשת התווים מהמשתמש. את התווים יש להזין בזה אחר זה; אם נפריד ביניהם על ידי מקש רווח או מקש Enter, ייחשב מקש זה לאחד מחמשת התווים המוזנים.

```
delta = 'a' - 'A';
```

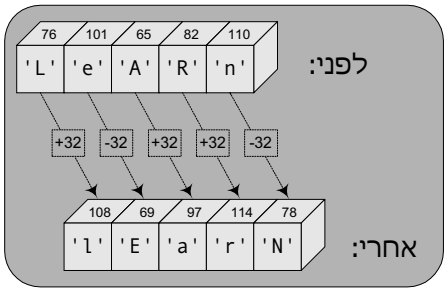
בשורה זו מוצב במשתנה delta ההפרש בין הערכים המספריים של האותיות 'a' ו-'A'. מאוחר יותר, כאשר נוסיף או נחסר ערך זה מתווי המחרוזת, נוכל להפוך אות גדולה לאות קטנה, ולהיפך. הסבר נוסף על הנושא מופיע בתוכנית 5.6.

```
for (i = 0 ; i < 5 ; i++) {
    /* ... */
```

לולאה זו מתבצעת חמש פעמים, והיא אחראית על פליטת התווים שהזין המשתמש, וזאת לאחר הפיכת כל אות קטנה לגדולה ולהיפך.

```
    if (string[i] >= 'a' && string[i] <= 'z')
        string[i] -= delta;
```

אם איבר המערך string שמיקומו i הינו אות לועזית קטנה, ברצוננו להפוך אותו לאות לועזית גדולה. זאת נעשה על ידי כך שנפחית ממנו את ערך delta, המייצג את המרחק בטבלת ASCII בין האותיות הקטנות לגדולות. פעולה זו מומחשת באיור הבא:



איור 9.4 *– איברי המערך string לפני ואחרי הפיכת כל האותיות הקטנות לגדולות ולהיפך (המספרים מעל לתווים מציינים את ערכיהם המספריים בטבלת ASCII)*

```
    else if (string[i] >= 'A' && string[i] <= 'Z')
        string[i] += delta;
```

אם איבר המערך string שמיקומו i הינו אות לועזית גדולה, ברצוננו להפוך אותו לאות לועזית קטנה. זאת נעשה על ידי כך שנוסיף לו את ערך delta, המייצג את המרחק בטבלת ASCII בין האותיות הקטנות לגדולות. גם פעולה זו מומחשת באיור 9.4.

```
    printf("%c", string[i]);
```

משפטי התנאי הקודמים בדקו האם איבר המערך string שמיקומו i הינו אות קטנה או גדולה, ושינו אותו בהתאם. עתה, יש לפלוט את התו. הדבר נעשה בשורה זו.

שיטה פשוטה לשיפור ציונים בבחינה היא הפיכת הציון הגבוה ביותר בכיתה ל-100 על ידי הכפלתו במספר מסוים, והכפלת שאר הציונים באותו מספר. שיטה זו מכונה **נרמול**. בדוגמה זו נקלוט עשרה ציונים מהמשתמש ונעזר בשיטה זו כדי לשפרם.

הבה נדגים את שיטת הנרמול לפני שנתעמק בתוכנית. נניח שהציון הגבוה ביותר היה 80. ראשית, כדי להפכו ל-100, יש להכפילו ב-100/80, כלומר, ב-1.25. לאחר שנהפכו ל-100, נכפיל את שאר הציונים ב-1.25. למשל, ציון 40 יהפוך ל-50.

תוכנית 9.3 *GRADES3.C*

```
#include <stdio.h>
void main()
{
    int grades[10], better_grades[10], i, max = 0, improvement = 0;

    printf("Please enter ten grades: ");

    for (i = 0 ; i < 10 ; i++) {
        scanf("%d", &grades[i]);
        if (grades[i] > max)
            max = grades[i];
    }

    printf("\nBefore\t\tAfter \nfactor:\t\tfactor: \n");

    for (i = 0 ; i < 10 ; i++) {
        better_grades[i] = grades[i] * 100 / max;
        printf("%d\t-->\t%d \n", grades[i], better_grades[i]);
        improvement += better_grades[i] - grades[i];
    }

    improvement /= 10;
    printf("\nAverage improvement: %d \n", improvement);
}
```

פלט אפשרי מהתוכנית הוא כזה:

```
Please enter ten grades: 0 70 65 50 80 60 40 75 60 30

Before          After
factor:         factor:
0       -->     0
70      -->     87
65      -->     81
50      -->     62
80      -->     100
60      -->     75
40      -->     50
75      -->     93
60      -->     75
30      -->     37

Average improvement: 13
```

```
    int grades[10], better_grades[10], i, max = 0,
        improvement = 0;
```

בשורה זו מוגדרים שני מערכים בני עשרה איברים, בשמות grades ו-better_grades. במערך grades יאוחסנו הציונים המוזנים ובמערך better_grades יאוחסנו הציונים לאחר הנרמול. המשתנה i הוא משתנה מונה פשוט. במשתנה max יוצב הציון הגבוה ביותר. המשתנה improvement יכיל, בסוף התוכנית, את השיפור הממוצע בציונים.

```
    for (i = 0 ; i < 10 ; i++) {
        scanf("%d", &grades[i]);
        if (grades[i] > max)
            max = grades[i];
    }
```

לולאה זו מתבצעת עשר פעמים, והיא קולטת ציונים לתוך איברי המערך grades. משפט ה-if מציב במשתנה max את הציון הגבוה ביותר שהוזן עד כה.

```
    for (i = 0 ; i < 10 ; i++) {
        better_grades[i] = grades[i] * 100 / max;
        printf("%d\t->\t%d \n", grades[i], better_grades[i]);
        improvement += better_grades[i] - grades[i];
    }
```

לולאה זו מאחסנת באיברי המערך better_grades את הציונים המנורמלים של הציונים שבמערך grades. משפט הפלט מציג בטבלה את הציונים הרגילים ואת הציונים המנורמלים שלהם. כמו-כן, השיפור בכל ציון (כלומר, במה עלה כל ציון בעקבות הנרמול שלו) מתוסף למשתנה improvement.

במחשבה שנייה...

מה יתרחש אם כל הציונים שהוזנו יהיו 0?

אם כל הציונים הינם 0, יהיה גם ערך המשתנה num 0, ותתבצע חלוקה ב-0, העלולה לגרום לקריסת התוכנית. לשם הפשטות הנחנו בתוכנית זו שהדבר בלתי אפשרי.

```
    improvement /= 10;
```

עם סיום לולאת for הקודמת, מכיל המשתנה improvement את סך-כל השיפור בציונים. כדי לדעת את ממוצע השיפור בציונים, עלינו לחלקו במספר הציונים. זאת עשינו במשפט ההשמה שלעיל.

מערכים דו-מימדיים

המערכים החד-מימדיים מיועדים לאחסון נתונים הניתנים לייצוג בשורה (שהרי, לשורה מימד אחד בלבד). למשל, הם מתאימים היטב לייצוג ציוני תלמידים במבחן מסוים. אך אם נרצה לייצג ציוני תלמידים בשלושה מבחנים, כך שלכל תלמיד יהיו שלושה ציונים, המערך החד-מימדי לא יספק אותנו. לשם כך נשתמש ב**מערך דו-מימדי** (two-dimensional array), המכונה גם **מטריצה** (matrix). המערך הדו-מימדי מוסיף מימד נוסף למערך החד-מימדי, ואת איבריו נוח לרשום בטבלה. באופן זה, ביכולתנו לטפל במבני נתונים מורכבים יותר.

נניח שמורה מעוניין לרשום בטבלה את ציוני עשרת תלמידיו בשלושה מבחנים שונים. המורה מחליט שכל שורה תייצג את ציוניו של תלמיד אחר, וכל עמודה – את ציוני כל התלמידים במבחן מסוים.

טבלה כזו מוצגת באיור הבא:

	ציון א'	ציון ב'	ציון ג'
תלמיד א'			
תלמיד ב'			
תלמיד ג'			
תלמיד ד'			
תלמיד ה'			
תלמיד ו'			
תלמיד ז'			
תלמיד ח'			
תלמיד ט'			
תלמיד י'			

***איור 9.5** – ייצוג איברי מערך דו-מימדי בטבלה*

קל להגדיר מערך דו-מימדי העונה לדרישות הטבלה שלעיל. לשם כך יש לציין את סוג המערך, את שמו ואת מספר השורות והעמודות שבו. השניים האחרונים יופיעו בתוך סוגריים מרובעים נפרדים. לכן, הגדרת מערך המתאים לטבלה זו מבוצעת כך:

```
int grades[10][3];      /* 10 rows * 3 columns */
```

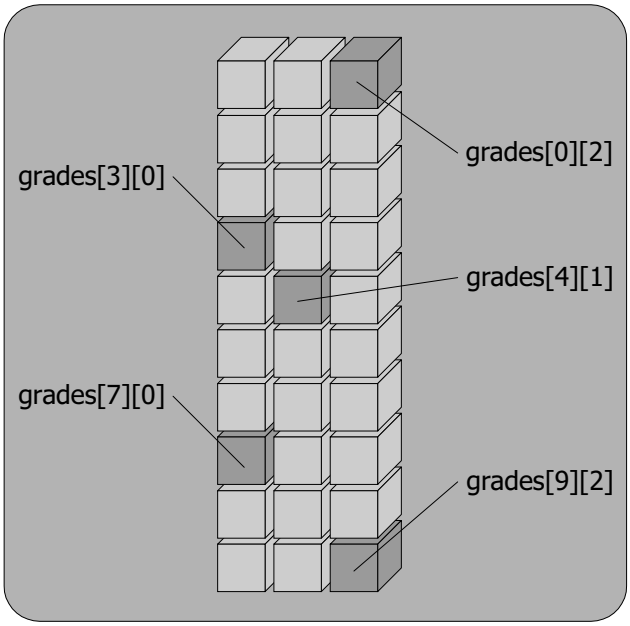
שים לב!

מספר האיברים במערך דו-מימדי הוא מכפלת מספר שורותיו בעמודותיו.

כדי לגשת לאיבר במערך דו-מימדי, לא די בציון אינדקס אחד. מאחר שמיקומו של איבר נקבע על-פי השורה והעמודה בהן הוא נמצא, יש לציין שני אינדקסים: האינדקס הראשון יקבע את השורה בה נמצא האיבר המבוקש, והאינדקס השני – את העמודה. כל אינדקס יתחיל מ-0, ויימצא בסוגריים מרובעים משלו. אם נרצה, למשל, לפלוט את ערך האיבר הנמצא בשורה הרביעית ובעמודה השנייה במערך grades, נכתוב משפט כזה:

```
printf("%d", grades[3][1]);   /* 4th row, 2nd column */
```

האיור הבא מציג את האינדקסים המייצגים את מיקומם של איברים נבחרים במערך grades:



***איור 9.6** – מיקומי איברים נבחרים במערך הדו-מימדי grades*

זהירות!

בפניה לאיבר במערך דו-מימדי, זכור כי מספר השורה קודם למספר העמודה. למשל, איבר המערך grades שנמצא בשורה השנייה ובעמודה הראשונה הוא array[1][0], ולא array[0][1].

איך לא...

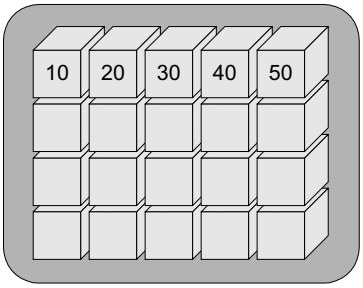
מכיון שמיקומו של איבר במערך דו-מימדי מוגדר על ידי שני אינדקסים, המשפטים הבאים שגויים מבחינה תחבירית:

```
grades = 33;       /* wrong, two indexes required */
grades[3] = 33;    /* wrong, two indexes required */
```

נניח שמוגדר מערך תווי בשם chs, המכיל ארבע שורות וחמש עמודות. אם נרצה לקלוט ערכים מהמשתמש ולהציבם באיברי השורה הראשונה, נבצע את הלולאה הבאה:

```
for (i = 0 ; i < 5 ; i++)
    chs[0][i] = getch();
```

מכיון שהערכים נקלטים לאיברים הנמצאים בשורה בודדת, האינדקס המציין את מספר השורה הוא קבוע. לעומתו, האינדקס המציין את מספר העמודה משתנה. בשורה הראשונה קיימים חמישה איברים, המיוצגים על ידי האינדקסים 0-4. לכן, משפט הקלט מתבצע חמש פעמים, כאשר המשתנה i מכיל את הערכים 0, 1, 2, 3 ו-4. אם, למשל, יוזנו הערכים 10, 20, 30, 40 ו-50, יתקבל המערך שאיבריו מוצגים באיור הבא:

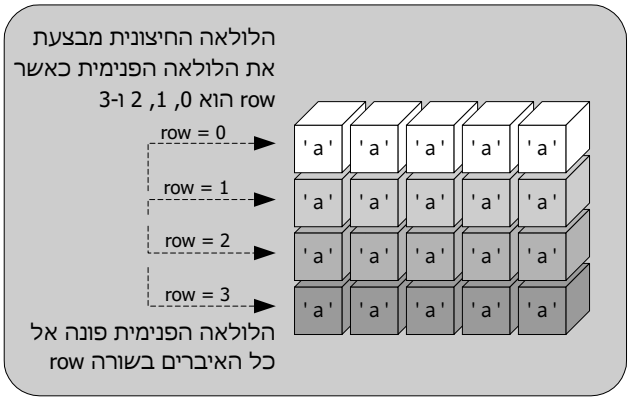


איור 9.7 *– ערכי איברי המערך chs לאחר קליטת ערכים אל איברי השורה הראשונה*

כדי להציב ערכים בכל איברי המערך, יהיה עלינו לשנות הן את אינדקס השורה והן את אינדקס העמודה. לשם כך נשתמש בלולאות מקוננות, כפי שמודגם להלן:

```
for (row = 0 ; row < 4 ; row++)
    for (col = 0 ; col < 5 ; col++)
        chs[row][col] = 'a';
```

משפט ההשמה מציב את הערך 'a' באיבר שבשורה row ובטור col. הלולאה הפנימית מתבצעת 5 פעמים, והיא מביאה להצבת הערך 'a' בכל איברי השורה row. את הלולאה הזאת מבצעת הלולאה החיצונית 4 פעמים, כאשר ערך המשתנה row הוא 0, 1, 2, ו-3 – ובאופן זה, הערך 'a' מוצב בכל איברי המערך. הדבר מודגם באיור הבא:

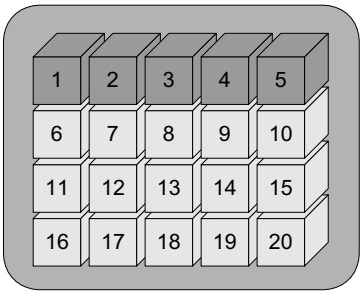


איור 9.8 *– לולאות מקוננות המציבות ערך בכל איברי המערך*

באופן דומה, אם נבצע את קטע התוכנית הבא:

```
for (row = 0 ; row < 4 ; row++)
    for (col = 0 ; col < 5 ; col++)
        scanf("%d", &chs[row][col]);
```

ונזין את כל המספרים השלמים בין 1 ל-20, בסדר עולה, יתקבל המערך שאיבריו מוצגים באיור הבא:



איור 9.9 *– ערכי איברי המערך chs לאחר קליטת ערכים אל כל איבריו*

דוגמה

התוכנית שנכתוב בדוגמה זו תקלוט עשרה מספרים שלמים מהמשתמש, חלקם חיוביים וחלקם שליליים, ותפלוט את ערכיהם המוחלטים של מספרים אלה. בתוכנית זו נשתמש במערך דו-מימדי בן שתי שורות ועשר עמודות. באיברי השורה הראשונה יאוחסנו המספרים שיזין המשתמש, ובאיברי השורה השנייה יאוחסנו ערכיהם המוחלטים של מספרים אלה.

ניתן לחשב ערך מוחלט של מספר באמצעות משפט תנאי פשוט, אך בתוכנית זו נעשה זאת על ידי קריאה לפונקציית הספריה abs. פונקציה זו מקבלת כפרמטר מספר שלם מסוג int ומחזירה את הערך המוחלט שלו. למשל, הקריאה הבאה לפונקציה:

```
i = abs(-10);
```

תציב במשתנה i את הערך 10. גם אם נקרא לפונקציה כך:

```
i = abs(10);
```

יוצב במשתנה i הערך 10.

כדי שנוכל להשתמש בפונקציה abs נוסיף לראש התוכנית את שורת include# הבאה:

```
#include <math.h>
```

שורה זו גורמת למהדר להתייחס אל נתונים הנמצאים בקובץ math.h, והיא דרושה לשם ביצועה התקין של הפונקציה abs.

שים לב!

הפונקציה abs תפעל כהלכה רק עבור ערכים שלמים מסוג int. כדי לחשב ערך מוחלט של מספר עשרוני נשתמש בפונקציה fabs, וכדי לחשב ערך מוחלט של מספר שלם מסוג long נשתמש בפונקציה labs. הפונקציות fabs ו-labs דומות לפונקציה abs – גם הן מקבלות פרמטר מספרי בודד ומחזירות את ערכו המוחלט, וגם הן דורשות את הוספת השורה #include <math.h> לתוכנית.

תוכנית 9.4

ABS.C

```
#include <stdio.h>
#include <math.h>
void main()
{
    int array[2][10], i;

    printf("Please enter ten values: ");
    for (i = 0 ; i < 10 ; i++) {
        scanf("%d", &array[0][i]);
        array[1][i] = abs(array[0][i]);
    }

    printf("Absolute values: ");
    for (i = 0 ; i < 10 ; i++)
        printf("%d ", array[1][i]);
}
```

פלט לדוגמה המתקבל מתוכנית זו מוצג להלן:

```
Please enter ten values: 1 -9 5 0 -66 -12 3 -4 -8 10
Absolute values: 1 9 5 0 66 12 3 4 8 10
```

```
#include <math.h>
```

כאמור, שורה זו הכרחית כדי שנוכל לקרוא לפונקציה abs מתוכניתנו.

```
    int array[2][10], i;
```

בשורה זו מוגדר מערך דו-מימדי בשם array, המכיל שתי שורות ועשר עמודות. באיברי השורה הראשונה יוצבו המספרים שיזין המשתמש, ואיברי השורה השנייה יכילו את ערכיהם המוחלטים של מספרים אלה. המשתנה i המוגדר בשורה זו הוא משתנה מונה פשוט.

```
for (i = 0 ; i < 10 ; i++) {
    scanf("%d", &array[0][i]);
    /* ... */
```

לולאה זו מתבצעת 10 פעמים, ובכל איטרציה שלה נקלט מספר לאיבר אחר במערך array. שים לב לפנייה אל איבר המערך array[0][i]. בכל פעם שהלולאה מתבצעת הערך המופיע בצמד הסוגריים המרובעים הראשון הוא 0 – רק הערך המופיע בצמד הסוגריים המרובעים השני משתנה. כך מוצבים הערכים המוזנים בכל איברי השורה הראשונה (בעלת האינדקס 0) שבמערך array.

```
    array[1][i] = abs(array[0][i]);
```

לאחר כל פעם שנקלט מספר לאיבר המערך array[0][i], הערך המוחלט שלו מוצב באיבר המתאים בשורה השנייה במערך – כלומר, באיבר המערך array[1][i]. מציאת הערך המוחלט מתבצעת על ידי שליחת איברי השורה הראשונה לפונקציה abs. הטבלה הבאה מציגה את איברי המערך array לאחר הזנת הציונים וחישוב ערכיהם המוחלטים:

שורה 0	1	-9	5	0	-66	-12	3	-4	-8	10
שורה 1	1	9	5	0	66	12	3	4	8	10

```
for (i = 0 ; i < 10 ; i++)
    printf("%d ", array[1][i]);
```

לולאה זו מתבצעת 10 פעמים, והיא פולטת את ערכי איברי השורה השנייה (בעלת האינדקס 1) במערך array. שורה זו מכילה את הערכים המוחלטים של המספרים שהוזנו.

דוגמה

התוכנית הבאה שנכתוב תקלוט ערכים מהמשתמש, המייצגים את ערכי איבריו של מערך דו-מימדי בן שלוש שורות ושלוש עמודות, ותפלוט הודעה מתאימה אם המערך הוא ריבוע קסם.

במחשבה שנייה...

מהו ריבוע קסם?

ריבוע קסם הוא מערך דו-מימדי שסכומי האיברים בכל שורה, בכל עמודה ובאלכסונים הראשי והמשני שווים. הנה דוגמה לשני ריבועי קסם:

5	1	6
5	4	3
2	7	3

12	1	11
7	8	9
5	15	4

תוכנית 9.5

SQUARE.C

```
#include <stdio.h>
void main()
{
    int sq[3][3], row, col, is_magic = 1, sum;

    printf("Please enter nine numbers: \n");

    /* read values */
    for (row = 0 ; row < 3 ; row++)
        for (col = 0 ; col < 3 ; col++)
            scanf("%d", &sq[row][col]);

    /* check the diagonals */
    sum = sq[0][0] + sq[1][1] + sq[2][2];
    if (sq[2][0] + sq[1][1] + sq[0][2] != sum)
        is_magic = 0;

    /* check the columns and the rows */
    if (is_magic == 1)
        for (row = 0 ; row < 3 ; row++)
            if (sq[row][0] + sq[row][1] + sq[row][2] != sum ||
                sq[0][row] + sq[1][row] + sq[2][row] != sum)
                is_magic = 0;

    if (is_magic == 1)
        printf("This is a magic square! \n");
    else
        printf("This isn't a magic square! \n");
}
```

הנה פלט לדוגמה מהתוכנית:

```
Please enter nine numbers:
5 1 6
5 4 3
2 7 3
This is a magic square!
```

```
    int sq[3][3], row, col, is_magic = 1, sum;
```

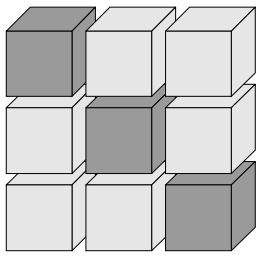
בשורה זו מוגדר מערך דו-מימדי בשם sq, בעל שלוש שורות ושלוש עמודות. במערך זה יאוחסנו הערכים שיזין המשתמש. במשתנים row ו-col נשתמש כמונים, כדי לפנות אל איברי המערך השונים. המשתנה is_magic מאותחל לערך 1, ואם יימצא, במהלך התוכנית, כי המערך אינו ריבוע קסם, יוצב בו הערך 0. את תפקיד המשתנה sum נראה בהמשך.

```
    for (row = 0 ; row < 3 ; row++)
        for (col = 0 ; col < 3 ; col++)
            scanf("%d", &sq[row][col]);
```

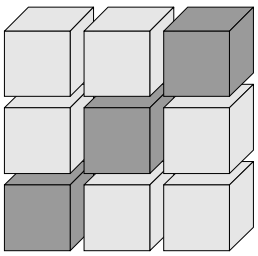
לולאות אלו מביאות לקליטת תשעה ערכים מהמשתמש. ערכים אלה מוצבים באיבריו השונים של המערך sq.

```
    sum = sq[0][0] + sq[1][1] + sq[2][2];
```

בשורה זו אנו פונים אל איברי האלכסון הראשי שבמערך (אלה הם האיברים הכהים שבאיור 9.10–א'), ומציבים את סכום ערכיהם במשתנה sum. בהמשך נשווה את ערך המשתנה sum אל סכומי ערכי האיברים בכל שורה, בכל עמודה ובאלכסון המשני. אם הם יהיו שונים זה מזה, נדע כי הערכים שנקלטו אינם יוצרים ריבוע קסם.



איור 9.10–א'



איור 9.10–ב'

```
    if (sq[2][0] + sq[1][1] + sq[0][2] != sum)
        is_magic = 0;
```

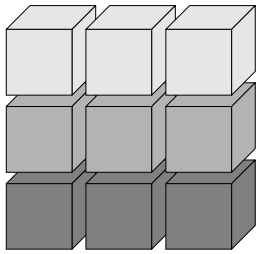
במשפט זה מושווה ערך המשתנה sum, המכיל את סכום ערכי האיברים של האלכסון הראשי, עם סכום ערכי האיברים של האלכסון המשני (אלה הם האיברים הכהים שבאיור 9.10–ב'). אם הסכומים שונים זה מזה, אזי אין בפנינו ריבוע קסם, והערך 0 מוצב במשתנה is_magic.

```
    if (is_magic == 1)
        /* ... */
```

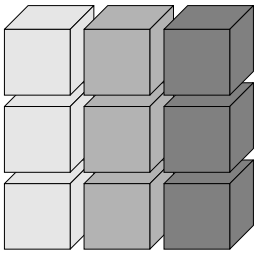
אם ערך המשתנה is_magic הוא 0, אזי מצאנו במשפט התנאי הקודם שהמערך אינו ריבוע קסם, ולכן אין צורך בבדיקות נוספות. משפט תנאי זה מוודא כי הלולאה השייכת לו, הבודקת את איברי השורות והעמודות, לא תתבצע לשווא (כלומר, אם ידוע כבר שהמערך אינו ריבוע קסם).

```
        for (row = 0 ; row < 3 ; row++)
            if (sq[row][0] + sq[row][1] + sq[row][2] != sum ||
                sq[0][row] + sq[1][row] + sq[2][row] != sum)
                is_magic = 0;
```

ביטוי התנאי של משפט התנאי מורכב משני תת-ביטויים. תת-הביטוי הראשון מסכם בכל פעם את איבריה של שורה אחרת במערך. כשערך row הוא 0, יסוכמו איברי השורה הראשונה ([2][0]+[1][0]+[0][0]); כשערך row הוא 1, יסוכמו איברי השורה השנייה ([2][1]+[1][1]+[0][1]); וכשערך row הוא 2, יסוכמו איברי השורה השלישית ([2][2]+[1][2]+[0][2]). בכל איטרציה נבדקת בתת-ביטוי זה שורה אחרת במערך, כפי שמוצג באיור 9.10ג'. סכומי איברי השורות מושווים אל ערך המשתנה sum, המכיל את סכום ערכי האיברים באלכסון הראשי. אם הם שונים זה מזה, אזי אין בפנינו ריבוע קסם, והערך 0 מוצב במשתנה is_magic.



איור 9.10ג'



איור 9.10ד'

תת-הביטוי השני מסכם בכל פעם את איבריה של עמודה אחרת במערך. כשהערך של row הוא 0, יסוכמו איברי העמודה הראשונה ([0][2]+[0][1]+[0][0]); כשהערך של row הוא 1, יסוכמו איברי העמודה השנייה ([1][2]+[1][1]+[1][0]); וכשהערך של row הוא 2, יסוכמו איברי העמודה השלישית ([2][2]+[2][1]+[2][0]). בכל איטרציה נבדקת בתת-ביטוי זה עמודה אחרת במערך, כפי שמוצג באיור 9.10ד'. סכומי איברי העמודות מושווים אל ערך המשתנה sum, המכיל את סכום ערכי האיברים באלכסון הראשי. אם הם שונים זה מזה, אזי אין בפנינו ריבוע קסם, והערך 0 מוצב במשתנה is_magic.

דוגמה

בדוגמה זו נתרגל פנייה אל איברי מערכים דו-מימדיים. התוכנית שנכתוב תקלוט את ציוניהם של עשרה תלמידים בשלושה מבחנים, ותחשב את ממוצע הציונים של כל תלמיד ואת ממוצע הציונים בכל מבחן.

תוכנית 9.6 *GRADES4.C*

```
#include <stdio.h>
void main()
{
    int grades[10][3], st_avg[10], gr_avg[3], row, col;

    for (col = 0 ; col < 3 ; col++)
        gr_avg[col] = 0;

    for (row = 0 ; row < 10 ; row++) {
        st_avg[row] = 0;
        printf("Please enter the grades of student #%d: ",
            row + 1);
        for (col = 0 ; col < 3 ; col++) {
            scanf("%d", &grades[row][col]);
            st_avg[row] += grades[row][col];
            gr_avg[col] += grades[row][col];
        }
        st_avg[row] /= 3;
    }
    printf("\nSt. #\tGrade 1\tGrade 2\tGrade 3\tAverage");
    printf("\n-----\t-------\t-------\t-------\t-------");
    for (row = 0 ; row < 10 ; row++) {
        printf("\n  %d\t", row + 1);
        for (col = 0 ; col < 3 ; col++)
            printf("  %d\t", grades[row][col]);
        printf("  %d\t", st_avg[row]);
    }

    printf("\n-----\t-------\t-------\t-------\t------- \n
        Average:");
    for (col = 0 ; col < 3 ; col++) {
        gr_avg[col] /= 10;
        printf("  %d\t", gr_avg[col]);
    }
    printf("\n");
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter the grades of student #1: 100 85 80
Please enter the grades of student #2: 87 83 85
Please enter the grades of student #3: 79 81 75
Please enter the grades of student #4: 100 95 95
Please enter the grades of student #5: 83 70 75
Please enter the grades of student #6: 65 60 59
Please enter the grades of student #7: 85 87 84
Please enter the grades of student #8: 91 90 70
Please enter the grades of student #9: 73 84 70
Please enter the grades of student #10: 90 80 85

St. #   Grade 1 Grade 2 Grade 3 Average
-----   ------- ------- ------- -------
  1       100     85      80      88
  2       87      83      85      85
  3       79      81      75      78
  4       100     95      95      96
  5       83      70      75      76
  6       65      60      59      61
  7       85      87      84      85
  8       91      90      70      83
  9       73      84      70      75
  10      90      80      85      85
-----   ------- ------- ------- -------
Average:  85      81      77
```

```
    int grades[10][3], st_avg[10], gr_avg[3], row, col;
```

בשורה זו מוגדר מערך דו-מימדי בשם grades, בעל עשר שורות ושלוש עמודות. במערך זה יאוחסנו ציוני התלמידים. במערך החד-מימדי st_avg יאוחסנו ממוצעי ציוניו של כל תלמיד. במערך החד-מימדי gr_avg יאוחסנו ממוצעי הציונים בכל מבחן. במשתנים row ו-col נשתמש כדי לפנות אל איברי המערכים.

```
    for (col = 0 ; col < 3 ; col++)
        gr_avg[col] = 0;
```

לולאה זו מאפסת את איברי המערך gr_avg. בהמשך נוסיף לכל איבר במערך זה את אחד משלושת ציוניו של כל תלמיד. כך, כל איבר במערך יכיל סכום ציונים במבחן אחר. כשנחלק כל איבר במספר הציונים שהוזנו, יתקבל ממוצע הציונים בכל מבחן.

```
    for (row = 0 ; row < 10 ; row++) {
        /* ... */
```

לולאה זו אחראית על קליטת הציונים מהמשתמש. בכל איטרציה שלה נקלטים ציוניו של תלמיד אחר, וכך מתמלאת שורה נוספת במערך grades.

```
        st_avg[row] = 0;
```

במשפט זה מתאפס איבר המערך st_avg הנמצא באינדקס row. בהמשך יכיל כל איבר במערך זה את סכום שלושת ציוניו של אחד התלמידים. כשנחלק כל איבר בשלוש, נקבל את ממוצע ציוניו של התלמיד.

```
        for (col = 0 ; col < 3 ; col++) {
            scanf("%d", &grades[row][col]);
            st_avg[row] += grades[row][col];
            gr_avg[col] += grades[row][col];
        }
```

בלולאה זו נקלטים שלושת ציוניו של אחד התלמידים. לולאה זו מתבצעת עשר פעמים, וכך נקלטים כל ציוני התלמידים. כל אחד מהציונים הנקלטים מתוסף לאיבר המתאים במערכים st_avg ו-gr_avg.

```
        st_avg[row] /= 3;
```

לאחר שלולאת for הפנימית הסתיימה, מכיל איבר המערך st_avg הנמצא באינדקס row את סכום שלושת ציוניו של אחד התלמידים. עתה, כדי למצוא את הממוצע של ציונים אלה, יש לחלק את ערך האיבר בשלוש.

```
    for (row = 0 ; row < 10 ; row++) {
        printf("\n  %d\t", row + 1);
        for (col = 0 ; col < 3 ; col++)
            printf("  %d\t", grades[row][col]);
        printf("  %d\t", st_avg[row]);
    }
```

לולאה זו אחראית על פליטת ציוני התלמידים והממוצע שלהם. משפט הפלט הראשון מציג את מספר התלמיד, לולאת for הפנימית פולטת את שלושת ציוניו, ומשפט הפלט האחרון מציג את ממוצע ציוניו. כל זאת מתבצע עשר פעמים – פעם אחת עבור כל תלמיד.

שים לב!

מספרי התלמידים, כפי שהם מיוצגים במערך, נעים מ-0 עד 9. אך למען נוחיות המשתמש, נעדיף להציגם מ-1 עד 10. לכן, איננו פולטים את ערך המשתנה row, אלא את הביטוי row + 1.

```
    for (col = 0 ; col < 3 ; col++) {
        gr_avg[col] /= 10;
        printf("  %d\t", gr_avg[col]);
    }
```

לולאה זו אחראית על פליטת ממוצע הציונים בכל מבחן. סכומי עשרת הציונים בכל אחד משלושת המבחנים נמצאים באיברי המערך gr_avg, וכדי למצוא את הממוצע שלהם, יש לחלק את ערכי האיברים הללו בעשר.

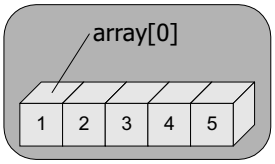
אתחול מערכים

אתחול מערך חד-מימדי

כמו משתנים רגילים, כך גם מערכים ניתנים לאתחול בשורת הגדרתם. לשם כך, נציין מספר ערכים מאתחלים ולא ערך מאתחל בודד, שהרי מערך מורכב ממספר איברים. כל ערך מאתחל יוצב באיבר אחר במערך, החל מהאיבר בעל האינדקס 0. את הערכים המאתחלים נתחום בסוגריים מסולסלים ונרשום אותם לאחר אופרטור ההשמה, כך:

```
int array[5] = { 1, 2, 3, 4, 5 };
```

שים לב כי הערכים המאתחלים מופרדים זה מזה בפסיקים. ערכים אלה מוצבים באיברי המערך לפי סדר הופעתם בסוגריים המסולסלים. לכן, לאחר שורת האתחול שלהלן, ייראה המערך array כפי שמוצג באיור הבא:



***איור 9.11** – ערכי איברי המערך array לאחר אתחולו*

הערכים המאתחלים חייבים להופיע בסוגריים מסולסלים, ולכן שורת ההגדרה הבאה אינה נכונה:

```
int array[5] = 1, 2, 3, 4, 5;     /* wrong! */
```

במחשבה שנייה...

מהו תפקידם של הסוגריים המסולסלים באתחול מערך?

תפקיד הסוגריים המסולסלים בהקשר הוא תחימת הערכים המאתחלים. אין לו קשר עם תפקיד הסוגריים המסולסלים ביצירת משפט מורכב.

אם ננסה לאתחל מערך במספר ערכים הגדול ממספר איבריו, יפלוט המהדר הודעת שגיאה. כלומר, אין לכתוב שורת הגדרה כגון זו:

```
float nums[3] = { 1.1, 2.2, 3.3, 4.4 };   /* wrong! */
```

לעומת זאת, מותר לאתחל מערך במספר ערכים הקטן ממספר איבריו. במקרה זה, האיברים האחרונים, להם לא קיים ערך מאתחל, יאותחלו לערך 0. לכן, ערכי איברי המערך המוגדר באופן הבא:

```
int many_numbers[5] = { 100, 200, 300 };
```

הם 100, 200, 300, 0 ו-0, בהתאמה.

אם ברצוננו לאפס את כל איבריו של מערך מסוים, נוכל לכתוב רק את הערך 0 בין הסוגריים המסולסלים. למשל, מערך המוגדר כך:

```
int zeros[10] = { 0 };
```

מכיל עשרה איברים, שבכולם הערך 0. אתחול כזה הוא מהיר ובהיר יותר מאשר לולאה העושה דבר זהה.

איך לא...

בין הסוגריים המסולסלים חייב להופיע ערך מאתחל אחד לפחות. לכן, שורת ההגדרה הבאה אינה נכונה:

```
int zeros[10] = { };
```

כאשר מאתחלים מערך, ניתן להשמיט את גודלו, כלומר, לרשום לאחר שם המשתנה סוגריים מרובעים פותחים וסוגרים כשביניהם לא מופיע דבר. במקרה זה, יהיה מספר איבריו כמספר הערכים המאתחלים. לדוגמה, שורת ההגדרה הבאה:

```
char digits[] = { '0', '1', '2', '3' };
```

מגדירה מערך תווי בעל ארבעה איברים – מספר השווה למספר הערכים המאתחלים.

איך לא...

השמטת גודל המערך אפשרית רק כאשר הוא מאותחל בערך אחד לפחות. כלומר, שורת ההגדרה הבאה אינה נכונה:

```
char letters[];
```

ניתן לשלב אתחול מערך עם הגדרת משתנים אחרים; למשל:

```
int num1, num2 = 3, numbers[4] = { 1, 2, 3, 4 }, i;
```

חשוב לציין שהצבת ערכים מרובים במערך, כפי שמוצג לעיל, אפשרית רק כאשר הוא מאותחל בשורת הגדרתו. למשל, קטע התוכנית הבא אינו נכון:

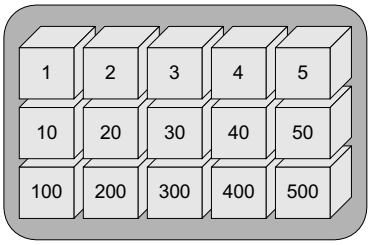
```
float heights[5];
/* ... */
heights = { 1.81, 1.92, 1.70, 2.0, 1.75 };  /* wrong! */
```

אתחול מערך דו-מימדי

באתחול מערך דו-מימדי יש לדאוג להצבת ערכים הן בשורותיו והן בעמודותיו. הדבר לא קשה כלל ועיקר; למשל, שורת ההגדרה הבאה מאתחלת את כל איבריו של מערך בן שלוש שורות וחמש עמודות:

```
int numbers[3][5] = { 1,   2,   3,   4,   5,
                      10,  20,  30,  40,  50,
                      100, 200, 300, 400, 500 };
```

הערך המאתחל הראשון יוצב באיבר המיוצג על ידי האינדקסים [0][0], הערך המאתחל השני – באיבר [1][0], השלישי – באיבר [2][0], וכדומה. לאחר שתמולא שורת איברים, תמולא השורה הבאה אחריה, באותו אופן. לכן, לאחר אתחולו, ייראה המערך numbers כפי שמוצג באיור הבא:



איור 9.12 *– ערכי איברי המערך numbers לאחר אתחולו*

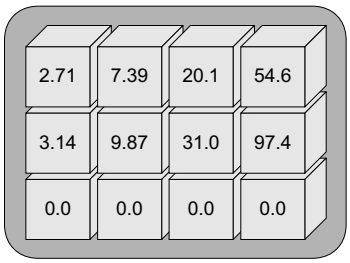
שים לב כי בין הערכים המאתחלים הוספנו רווחים ואף ירדנו לשורה חדשה. באופן זה, מיקומי הערכים המאתחלים רומזים על מיקומי האיברים בהם הם מוצבים. סגנון כתיבה כזה מקל על הבנת התוכנית, אך יכולנו, באותה מידה, להגדיר את המערך כך:

```
int numbers[3][5] = { 1, 2, 3, 4, 5, 10, 20, 30, 40,
                      50, 100, 200, 300, 400, 500 };
```

כמו באתחול מערך חד-מימדי, אם מספר הערכים המאתחלים קטן ממספר איברי המערך הדו-מימדי, יאופסו ערכי האיברים הנותרים. לכן, שורת ההגדרה הבאה:

```
float consts[3][4] = { 2.71, 7.39, 20.1, 54.6,
                       3.14, 9.87, 31.0, 97.4 };
```

מציבה את הערך 0.0 בכל האיברים שבשורה השלישית, כפי שמוצג באיור הבא:



איור 9.13 *– ערכי איברי המערך consts לאחר אתחולו*

באופן דומה, אם נגדיר מערך דו-מימדי ונאתחל אותו בערך 0 בלבד, כך:

```
int numbers[3][5] = { 0 };
```

יאופסו כל איבריו.

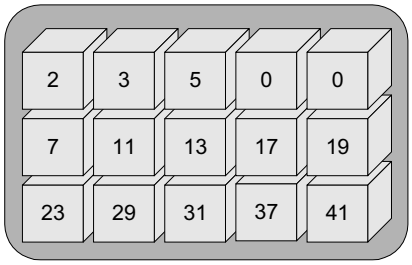
ניתן לתחום כל שורה במערך הדו-מימדי בצמד נוסף של סוגריים מסולסלים. שורת האתחול הבאה מדגימה זאת:

```
int values[3][5] = { { 1,   2,   3,   4,   5   },
                     { 10,  20,  30,  40,  50  },
                     { 100, 200, 300, 400, 500 } };
```

בשורה זו מוגדר מערך שאיבריו זהים לאיברי המערך המוצג באיור 9.12. במקרה זה, הוספת הסוגריים המסולסלים לא השפיעה על איברי המערך – אך בחן את שורת ההגדרה הבאה:

```
int primes[3][5] = { { 2,  3,  5           },
                     { 7,  11, 13, 17, 19  },
                     { 23, 29, 31, 37, 41  } };
```

מאחר ששני ערכים הושמטו מהשורה הראשונה, שני האיברים האחרונים בשורה זו מאופסים. לכן, איברי המערך נראים כפי שמוצג באיור הבא:



***איור 9.14** – ערכי איברי המערך primes לאחר אתחולו*

שים לב!

בין כל שתי שורות של ערכים מאתחלים התחומות בסוגריים מסולסלים מופיע פסיק.

איך לא...

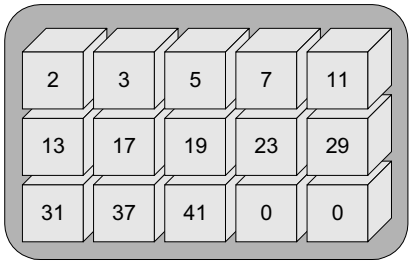
ניתן לתחום על ידי סוגריים מסולסלים נוספים רק שורות – ושורה אחת בכל פעם. לכן, שורת ההגדרה הבאה אינה נכונה:

```
int numbers[3][5] = { { 1, 2, 3 },      /* wrong */
                      { 4, 5, 6 },
                      { 7, 8, 9 },
                      { 10, 11, 12 },
                      { 13, 14, 15 } };
```

אם היינו מאתחלים מערך זהה באותם ערכים, אך מוותרים על הסוגריים המסולסלים, כך:

```
int primes2[3][5] = { 2,  3,  5,
                      7,  11, 13, 17, 19,
                      23, 29, 31, 37, 41  };
```

היו מאופסים שני האיברים האחרונים בשורה האחרונה, כפי שמוצג באיור הבא:



איור 9.15 *– ערכי איברי המערך primes2 לאחר אתחולו*

השינוי בערכי איברי המערך נובע מכך שבלי סוגריים מסולסלים, הערכים המאתחלים מוצבים באיברי המערך על-פי סדר הופעתם. הערך 7 מופיע לאחר הערך 5, והוא מוצב באיבר הבא אחריו. באופן דומה, שאר הערכים המאתחלים מוצבים באיברי המערך השונים. בסופו של דבר, מאותחל המערך ב-13 ערכים, והאיברים האחרונים הנותרים – שניים במספר – מאופסים.

באתחול מערכים דו-מימדיים ניתן להשמיט את גודל המערך הראשון, המייצג את מספר השורות במערך. במקרה זה, ייקבע גודל זה בהתאם למספר הערכים המאתחלים ולמספר העמודות. בחן, למשל, את המערך הבא:

```
char ch[][4] = { 'a', 'b', 'c', 'd',
                 'A', 'B', 'C', 'D',
                 '1', '2', '3', '4' };
```

מערך זה מכיל 4 עמודות ומאותחל ל-12 ערכים. לכן, כדי שכל הערכים יוצבו באיברי המערך, מספר שורותיו יהיה 3.

איך לא...

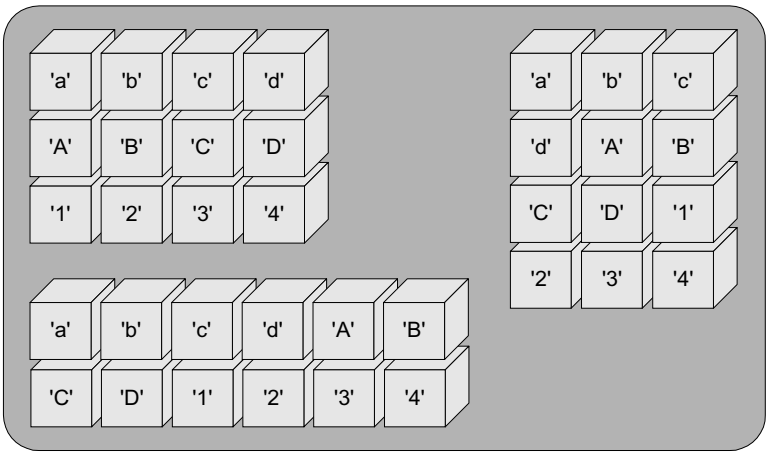
באתחול מערכים דו-מימדיים ניתן להשמיט רק את גודל המערך הראשון. לכן, שורת ההגדרה הבאה אינה נכונה:

```
int naturals[3][] = { 1, 2, 3,      /* wrong */
                      4, 5, 6,
                      7, 8, 9 };
```

אילו השמטנו גם את גודל המערך השני, המייצג את מספר העמודות במערך, לא יכולנו לקבוע את מימדיו המדויקים. לדוגמה, אל המערך הבא (המאותחל באופן שגוי):

```
char ch2[][] = { 'a', 'b', 'c', 'd',      /* wrong! */
                 'A', 'B', 'C', 'D',
                 '1', '2', '3', '4' };
```

ניתן לייחס מימדים רבים, כגון אלה המוצגים להלן:

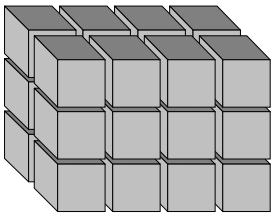


איור 9.16 *– מימדים אפשריים למערך בן 12 איברים*

מערכים רב-מימדיים

לעיתים יש צורך במבנה נתונים המכיל כמה מימדים. במקרה כזה, ניתן להגדיר מערך בעל מספר רב של מימדים. מערך כזה נקרא **מערך רב-מימדי** (multidimensional array).
למעשה, גם המערך הדו-מימדי הוא סוג של מערך רב-מימדי. עתה, נתאר מערכים בעלי שלושה מימדים ויותר.

כדי להגדיר מערך בעל n איברים, עלינו לציין ערכים בתוך n זוגות סוגריים מרובעים, המופיעים לאחר שם המערך. למשל, כדי להגדיר מערך תלת-מימדי המייצג את הצורה הבאה:

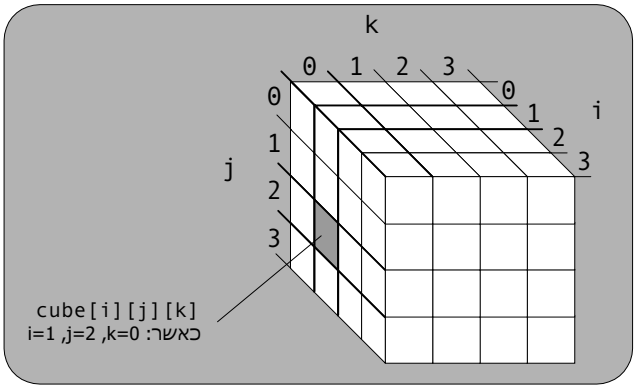


איור 9.17 *– צורה המתאימה למערך תלת-מימדי*

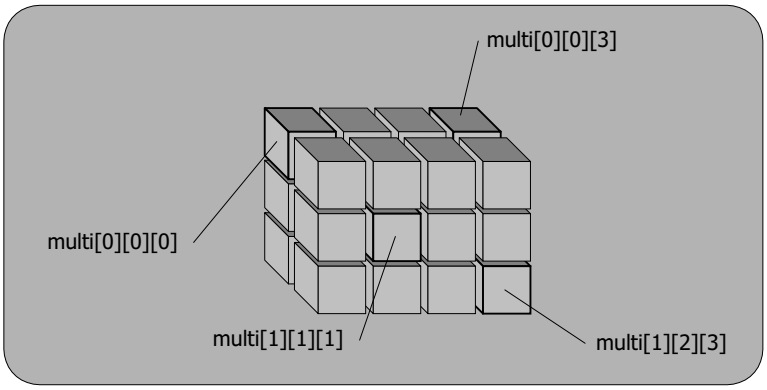
נכתוב שורת הגדרה כזו:

```
int multi[2][3][4];
```

כאשר מביטים בצורה, ניתן לומר כי הערך הראשון שבסוגריים מציין את עומקה, הערך השני – את גובהה, והערך השלישי – את רוחבה. שני האיורים הבאים מדגימים זאת:



איור 9.18 *– התייחסות אל איבר במערך תלת-מימדי*



איור 9.19 *– איברים נבחרים במערך התלת-מימדי multi*

במחשבה שנייה...

מהו מספר המימדים המירבי של מערך?

אין הגבלה (בגבולות ההיגיון) על מספר מימדיו של מערך, אך רק במקרים נדירים נזדקק למערך בעל יותר משני מימדים.

כאשר ברצוננו לקלוט ערכים מהמשתמש ולאחסנם במערך רב-מימדי, נהוג להשתמש בלולאותfor מקוננות, שמספרן הוא כשל מספר מימדי המערך. למשל, כדי לקלוט מספרים מהמשתמש ולהציבם באיברי המערך multi, נבצע קטע תוכנית כזה:

```
for (depth = 0 ; depth < 2 ; depth++)
    for (height = 0 ; height < 3 ; height++)
        for (width = 0 ; width < 4 ; width++)
            scanf("%d", &multi[depth][height][width]);
```

באופן דומה, אם נגדיר מערך בעל חמישה מימדים, כך:

```
int five[3][4][2][5][3];
```

ונרצה להציב באיבריו ערכים המוזנים על ידי המשתמש, נבצע קטע תוכנית כזה:

```
for (a = 0 ; a < 3 ; a++)
    for (b = 0 ; b < 4 ; b++)
        for (c = 0 ; c < 2 ; c++)
            for (d = 0 ; d < 5 ; d++)
                for(e = 0 ; e < 3 ; e++)
                    scanf("%d", &five[a][b][c][d][e]);
```

אתחול מערך רב-מימדי

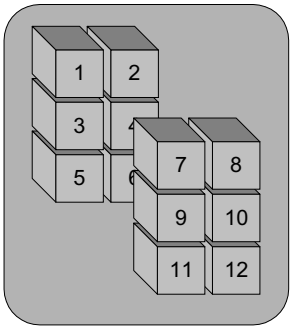
אתחול מערכים רב-מימדיים נעשה בדומה לאתחול מערכים חד-מימדיים ודו-מימדיים – כתיבת הערכים המאתחלים בין סוגריים מסולסלים. למשל, שורת ההגדרה הבאה:

```
int numbers[2][3][2] = { 1, 2, 3, 4,  5,  6, 7, 8, 9, 10, 11, 12 };
```

מגדירה מערך תלת-מימדי, ומציבה באיבריו את הערכים המאתחלים, באופן הבא:

איבר	ערך	איבר	ערך
[0][0][0]	1	[1][0][0]	7
[0][0][1]	2	[1][0][1]	8
[0][1][0]	3	[1][1][0]	9
[0][1][1]	4	[1][1][1]	10
[0][2][0]	5	[1][2][0]	11
[0][2][1]	6	[1][2][1]	12

ערכי האיברים הללו מוצגים באיור הבא:



איור 9.20 *– איברי המערך התלת-מימדי numbers לאחר אתחולו*

ואם נגדיר מערך בעל ארבעה מימדים, באופן הבא:

```
int four[2][3][2][2] = { 1,  2,  3,  4,  5,  6,
                         7,  8,  9,  10, 11, 12,
                         13, 14, 15, 16, 17, 18,
                         19, 20, 21, 22, 23, 24 } ;
```

יוצבו הערכים המאתחלים באיברי המערך באופן הבא:

איבר	ערך	איבר	ערך
[0][0][0]	1	[1][0][0]	13
[0][0][1]	2	[1][0][1]	14
[0][1][0]	3	[1][1][0]	15
[0][1][1]	4	[1][1][1]	16
[0][0][0]	5	[1][0][0]	17
[0][0][1]	6	[1][0][1]	18
[0][1][0]	7	[1][1][0]	19
[0][1][1]	8	[1][1][1]	20
[0][0][0]	9	[1][2][0]	21
[0][0][1]	10	[1][2][1]	22
[0][1][0]	11	[1][1][0]	23
[0][1][1]	12	[1][1][1]	24

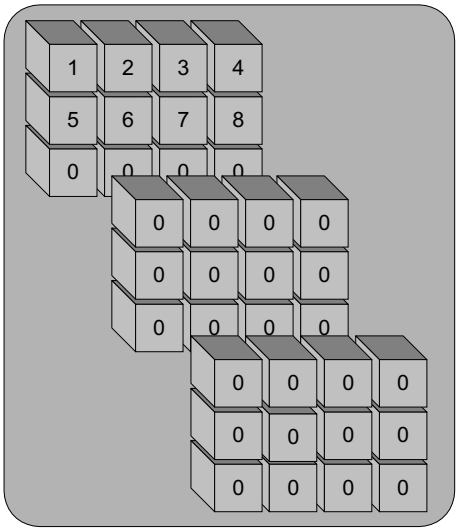
שים לב!

לא ניתן לייצג בתרשים את איברי מערך בעל יותר משלושה מימדים, ולכן נסתפק בטבלה זו.

אם מספר הערכים המאתחלים קטן ממספר איברי המערך הרב-מימדי, יתר האיברים יאותחלו לערך 0. למשל, לאחר שורת ההגדרה הבאה:

```
int values[3][3][4] = { 1, 2, 3, 4, 5, 6, 7, 8 };
```

ייראה המערך values כפי שמוצג באיור הבא:



איור 9.21 – *איברי המערך התלת-מימדי values לאחר אתחולו*

באופן דומה, שורת ההגדרה הבאה:

```
float averages[2][3][4][5] = { 0.0 };
```

מגדירה מערך בעל ארבעה מימדים ומאפסת את כל איבריו.

אם יש צורך באיפוס חלק מהאיברים האחרונים שבמימד מסוים, ניתן לתחום את הערכים המאתחלים שלו בסוגריים מסולסלים נוספים, ולוותר על הערכים המאתחלים המתאימים. לדוגמה, שורת האתחול הבאה:

```
int grades[4][2][2] = { 90, 78, 0,  0,
                        77, 79, 82, 0,
                        61, 70, 0,  0,
                        99, 0,  0,  0 };
```

שקולה לשורת האתחול הזו:

```
int grades[4][2][2] = { { 90, 78 },
                        { 77, 79, 82 },
                        { 61, 70 },
                        { 99 } };
```

כאשר מאתחלים מערכים רב-מימדיים, ניתן להשמיט את גודל המימד הראשון, והשטח שיוקצה למערך יהיה בהתאם למספר הערכים המאתחלים ולמספר המימדים האחרים של המערך.

בחן, למשל, מערך המוגדר כך:

```
char stuff[][2][3][2] =
    {  'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
       'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
       's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0',
       '1', '2', '3', '4', '5', '6', '7', '8', '9'  };
```

מספר הערכים המאתחלים הוא 36. גדלי המימדים הידועים הם 2, 3 ו-2, שמכפלתם היא 12. לכן, המימד הראשון של המערך יהיה 3.

שים לב!

עתה נוכל לדייק בניסוח חוק השמטת גודל המערך: כאשר מאתחלים מערך, בלי קשר למספר מימדיו, ניתן להשמיט רק את גודל המימד הראשון שלו.

שליחת מערכים לפונקציות

כפי שניתן לשלוח משתנים מסוגים שונים לפונקציות, כך ניתן לשלוח מערכים לפונקציות. בסעיף זה נלמד כיצד לשלוח מערכים חד-מימדיים ודו-מימדיים לפונקציות.

שליחת מערך חד-מימדי

נניח שקיים מערך המוגדר כך:

```
int array[10];
```

כדי לשלוח אותו לפונקציה בשם MyFunction, נבצע משפט כזה:

```
MyFunction(array);
```

איך לא...

בעת שליחת מערך לפונקציה אין לציין אינדקס (שהרי, בכך אנו פונים אל איבר במערך). אילו ציינו אינדקס, כך:

```
MyFunction(array[n]);
```

היה נשלח לפונקציה MyFunction איבר המערך array שמיקומו n, וזהו משתנה רגיל מסוג int ולא מערך מסוג int.

עתה נדון בהגדרת הפונקציה MyFunction. פונקציה זו מקבלת כפרמטר מערך מסוג int ולא משתנה int רגיל, ולכן לא נוכל להגדירה כך:

```
void MyFunction(int a)      /* wrong */
{
    /* ... */
}
```

בהגדרה זו הפרמטר a הוא משתנה רגיל מסוג int. כדי לציין שהוא מערך ולא משתנה int רגיל, נרשום לאחר שמו סוגריים מרובעים פותחים וסוגרים, בלי ערך ביניהם, כך:

```
void MyFunction(int a[])
{
    /* ... */
}
```

שים לב!

בעת הגדרת מערך חד-מימדי כמשתנה פורמלי, אין צורך לציין את מספר איבריו. לו רשמנו מספר טבעי כלשהו בין הסוגריים המרובעים המופיעים לאחר שם הפרמטר a, לא היתה זו טעות, אך הדבר לא היה משפיע כלל על פעולת הפונקציה. למשל, הגדרות הפונקציות הבאות הן זהות **לגמרי**:

```
void Function1(int a[])        {  /* ... */  }
void Function2(int a[10])      {  /* ... */  }
void Function3(int a[9999])    {  /* ... */  }
```

הפנייה אל איברי מערך מתוך הפונקציה אליה הוא נשלח תתבצע כפי שראינו עד כה. הנה, למשל, הגדרת פונקציה בשם PrintArray, המקבלת כפרמטר מערך ופולטת את ערכי עשרת איבריו הראשונים:

```
void PrintArray(int a[])
{
    int i;

    for (i = 0 ; i < 10 ; i++)
        printf("%d ", a[i]);
}
```

אם נבצע את שורות הקוד הבאות:

```
int array[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
PrintArray(array);
```

יתקבל, כצפוי, הפלט הבא:

```
1 2 3 4 5 6 7 8 9 10
```

המעניין הוא, שלפונקציה המקבלת מערך חד-מימדי כפרמטר ניתן לשלוח כל מערך חד-מימדי מאותו סוג – בלי חשיבות למספר האיברים שבו. על כך מרמזת העובדה שבהגדרת הפונקציה PrintArray לא מצוין גודל המערך. כלומר, אל הפונקציה PrintArray ניתן לשלוח כל אחד מהמערכים הבאים:

```
int array1[15] =  { 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5
};
int array2[12] =  { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
int array3[100] = { 0 };
```

כמובן שבכל מקרה ייפלטו רק עשרת האיברים הראשונים בכל מערך (כך נקבע בגוף הפונקציה).

העובדה שאין חשיבות למספר האיברים של מערך חד-מימדי הנשלח לפונקציה מאפשרת לכתוב תוכניות גמישות מאוד, משום שניתן לקרוא לפונקציות ולשלוח אליהן בכל פעם מערכים בגדלים שונים.

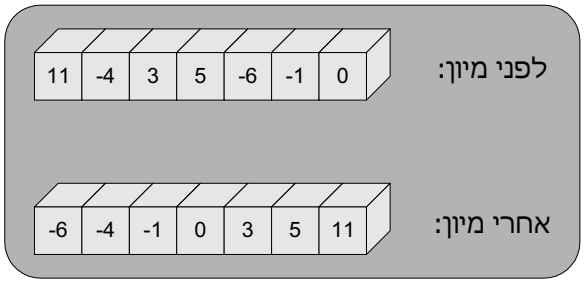
מכיון שאין דרך לדעת את מספר האיברים של מערך הנשלח לפונקציה מתוך אותה פונקציה, נהוג לשלוח לכל פונקציה המקבלת מערך פרמטר נוסף, המציין את מספר האיברים של אותו מערך. הנה, למשל, פונקציה בשם SumArray, המקבלת כפרמטר מערך ומחזירה את סכום איבריו:

```
int SumArray(int array[], int n)
{
    int i, sum = 0;

    for (i = 0 ; i < n ; i++)
        sum += array[i];
    return sum;
}
```

הפרמטר n קובע את מספר האיברים במערך array, ובהתאם לו מחושב סכום איברי המערך. ניתן לשלוח לפונקציה זו כל מערך חד-מימדי מסוג int, כפי שמודגם להלן:

```
int r1[2] =  { 1, 2 };
int r2[10] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 };
int r3[50] = { 2, 4 };
int s1, s2, s3;

s1 = SumArray(r1, 2) ;     /* assigns the value 3  */
s2 = SumArray(r2, 10);     /* assigns the value 88 */
s3 = SumArray(r3, 50);     /* assigns the value 6  */
```

בתוכנית זו נכתוב פונקציה המקבלת מערך חד-מימדי כפרמטר וממיינת את איבריו בסדר עולה. האיור הבא מציג מערך שאיבריו ממוינים ולא ממוינים:



***איור 9.22** – מערך שאיבריו ממויינים ולא ממויינים*

קיימים אלגוריתמים רבים למיון מערכים, ובדוגמה זו נתמקד באלגוריתם המכונה **מיון בועות** (bubble sort). הפונקציה שנכתוב תשתמש באלגוריתם זה כדי למיין את המערך הנשלח אליה בסדר עולה.

במחשבה שנייה...

כיצד תוכל הפונקציה שנכתוב לשנות את איברי המערך הנשלח אליה?

כאשר למדנו אודות הפונקציות קבענו כי פונקציה אינה יכולה לשנות ערך של פרמטר הנשלח אליה. טענה זו נכונה פרט למקרים בהם נשלח מערך לפונקציה. במקרים אלה יכולה הפונקציה לשנות את איבריו.

תוכנית 9.7

BUBBLE.C

```
#include <stdio.h>

void SortArray(int a[], int n);

void main()
{
    int array[7] = { 11, -4, 3, 5, -6, -1, 0 };
    int i;
```

```
    printf("\nBefore sorting:\t");
    for (i = 0 ; i < 7 ; i++)
        printf("%4d", array[i]);

    SortArray(array, 7);

    printf("\nAfter sorting:\t");
    for (i = 0 ; i < 7 ; i++)
        printf("%4d", array[i]);
}

void SortArray(int a[], int n)
{
    int i, j, temp;

    for (i = 0 ; i < n - 1 ; i++)
        for (j = n - 1 ; j > 0 ; j--)
            if (a[j - 1] > a[j]) {
                temp = a[j - 1];
                a[j - 1] = a[j];
                a[j] = temp;
            }
}
```

פלט התוכנית הוא כדלהלן:

```
Before sorting:   11  -4   3   5  -6  -1   0
After sorting:    -6  -4  -1   0   3   5  11
```

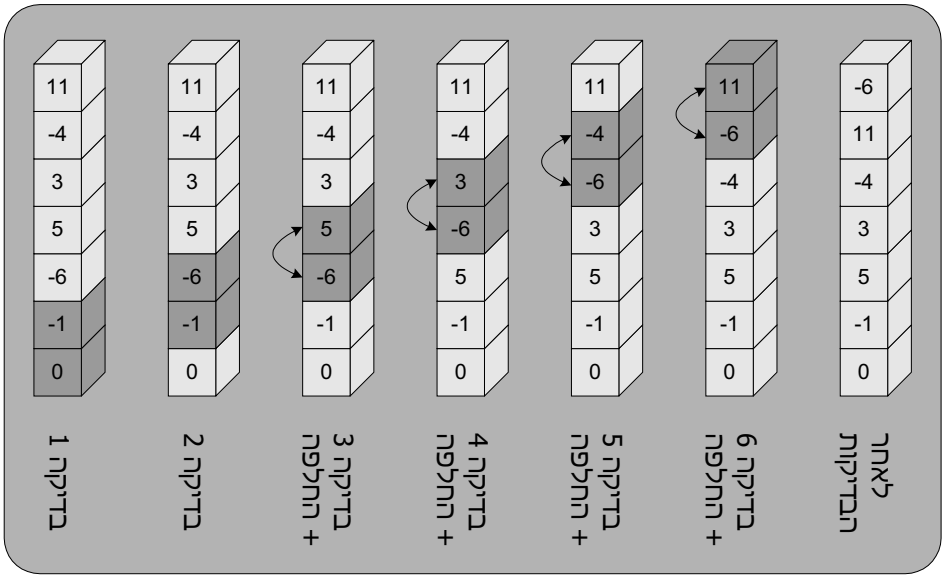
```
void SortArray(int a[], int n);

void SortArray(int a[], int n)
{
    /* ... */
}
```

הפונקציה SortArray מציינת את המערך הנשלח אליה, המיוצג על ידי הפרמטר a. הפרמטר השני n הוא מספר טבעי הקובע את מספר האיברים במערך a. שים לב כי גם בעת ההצהרה על הפונקציה SortArray מופיעים סוגריים מרובעים פותחים וסוגרים לאחר שם הפרמטר a.

```
for (j = n - 1 ; j > 0 ; j--)
    if (a[j - 1] > a[j]) {
        temp = a[j - 1];
        a[j - 1] = a[j];
        a[j] = temp;
    }
```

לולאת for פנימית זו מתבצעת n - 1 פעמים, החל מאיבר האחרון במערך ועד לאיבר השני בו (שהאינדקס שלו הוא 1). משפט התנאי השייך לה בודק האם קיימים שני איברים סמוכים המופיעים בסדר שגוי, כלומר, האם קיים איבר הגדול מהאיבר שבא אחריו. אם נמצאו שני איברים כאלה, ברצוננו להחליף ביניהם – וזאת עושים משפטי ההשמה השייכים למשפט התנאי.

האיור הבא מציג את פעולת לולאת for הפנימית בפעם הראשונה שהלולאה החיצונית מבצעת אותה:



איור 9.23 *– פעולת לולאת for הפנימית*
(האיברים הכהים הם האיברים המושווים בכל איטרציה. החיצים מצביעים על איברים מוחלפים)

מהאיור ניתן לראות כי לאחר הפעם הראשונה שהלולאה החיצונית ביצעה את הלולאה הפנימית, האיבר הקטן ביותר עבר לתחילת המערך.

```
for (i = 0 ; i < n - 1 ; i++)
```

לולאה זו מבצעת את הלולאה הפנימית n - 1 פעמים.

לפני שנעמוד על הסיבה לכך, נציג את השלבים במיון המערך, כלומר, את איברי המערך לאחר כל פעם שלולאה זו מבצעת את הלולאה הפנימית:

ללא מיון	שלב 1	שלב 2	שלב 3	שלב 4	שלב 5	שלב 6
11	-6	-6	-6	-6	-6	-6
-4	11	-4	-4	-4	-4	-4
3	-4	11	-1	-1	-1	-1
5	3	-1	11	0	0	0
-6	5	3	0	11	3	3
-1	-1	5	3	3	11	5
0	0	0	5	5	5	11

מהטבלה ניתן לראות כי לאחר כל פעם שהלולאה החיצונית מבצעת את הלולאה הפנימית, איבר נוסף עובר למקום הראוי לו במערך. לכן, כאשר הלולאה הפנימית תתבצע n - 1 פעמים, ימוקמו כל איברי המערך במקום המתאים להם, כלומר, בסדר עולה.

שים לב!

לאחר כל פעם שהלולאה החיצונית מבצעת את הלולאה הפנימית, איבר נוסף מוצא את מיקומו במערך, ובפעמים הבאות שהלולאה הפנימית תתבצע, מיקומו לא ישתנה. לכן ניתן לשפר את מהירות הביצוע של המיון, ולאחר כל פעם שהלולאה החיצונית מבצעת את הלולאה הפנימית, לא לסרוק איבר נוסף במערך. כלומר, בפעם השנייה לסרוק n - 2 איברים, בפעם השנייה לסרוק n - 3 איברים, וכן הלאה. הדבר יתבצע על ידי שינוי הלולאה הפנימית ללולאה הבאה:

```
for (j = n - 1 ; j > i ; j--)
    /* ... */
```

שליחת מערך דו-מימדי

שליחת מערך דו-מימדי לפונקציה דומה לשליחת מערך חד-מימדי. בשורת ההגדרה של הפונקציה, לצד שם הפרמטר המתאים למערך הדו-מימדי, יש לציין שני צמדי סוגריים מרובעים. צמד הסוגריים הראשון יהיה ריק וצמד הסוגריים השני יכיל את מספר העמודות במערך. נניח שמוגדר מערך דו-מימדי כך:

```
int array[10][4];
```

שליחת מערך זה לפונקציה בשם MyFunction תיעשה באופן הבא:

```
MyFunction(array, 10);
```

והפונקציה MyFunction תוגדר כך:

```
void MyFunction(int a[][4], int lines)
{
    /* ... */
}
```

אף שניתן לרשום כל ערך טבעי בין הסוגריים המרובעים הראשונים שלצד הפרמטר a והדבר לא ישפיע כלל על פעולת הפונקציה, המספר המופיע בין הסוגריים המרובעים השניים חשוב ביותר – וחובה עליו להופיע. יתרה מכך, כל מערך הנשלח אל הפונקציה MyFunction חייב להכיל 4 עמודות (ולהיות מסוג int, כמובן). מספר השורות שבו אינו חשוב כלל. כלומר, אל הפונקציה MyFunction ניתן לשלוח כל אחד מהמערכים הבאים:

```
int array1[2][4];
int array2[10][4];
int array3[999][4];
```

הנה דוגמה לפונקציה המקבלת מערך דו-מימדי מסוג int בן 4 עמודות ו-n שורות ופולטת את כל איבריו:

```
void PrintArray(int a[][4], int n)
{
    int i, j;

    for (i = 0 ; i < n ; i++) {
        for (j = 0 ; j < 4 ; j++)
            printf("%d ", a[i][j]);
        printf("\n");
    }
}
```

לפונקציה זו ניתן לשלוח כל מערך מסוג int בן 4 עמודות, ולכן הקריאות הבאות חוקיות:

```
PrintArray(array1, 2);
PrintArray(array2, 10);
PrintArray(array3, 999);
```

סיכום

- מערך הוא משתנה המכיל מספר איברים מסוג זהה. כל איבר במערך נחשב למשתנה רגיל מסוג זה.
- גודל המערך נקבע בשורת הגדרתו בין סוגריים מרובעים. זהו מספר טבעי וקבוע.
- אל איבר במערך נפנה על ידי ציון מיקומו במערך. מיקום זה מכונה **אינדקס**, והוא נכתב בין סוגריים מרובעים.
- אינדקס איברי המערך מתחיל מ-0.
- מערך חד-מימדי הוא מערך שאת איבריו נוח לשרטט בשורה.
- מערך דו-מימדי הוא מערך שאת איבריו נוח לשרטט בטבלה.
- הגדרת מערך דו-מימדי ופנייה אל איבר במערך דו-מימדי נעשות על ידי ציון שני ערכים בשני זוגות של סוגריים מרובעים.
- ניתן לאתחל מערך על ידי כתיבת ערכים מאתחלים בין סוגריים מסולסלים.
- אם מספר הערכים המאתחלים קטן ממספר איברי המערך, יתר האיברים יאותחלו לערך 0.
- ניתן להשמיט את גודל המימד הראשון של המערך כאשר מאתחלים אותו, ואז הוא ייקבע בהתאם למימדים הידועים ובהתאם למספר הערכים המאתחלים.
- ניתן להגדיר מערכים בעלי מימדים רבים. בעת הגדרת המערך ובעת פנייה אל איבר בו נשתמש ב-n זוגות של סוגריים מרובעים, כאשר n הוא מספר מימדי המערך.
- אל פונקציה המקבלת כפרמטר מערך חד-מימדי ניתן לשלוח כל מערך חד-מימדי מאותו סוג.
- אל פונקציה המקבלת כפרמטר מערך דו-מימדי ניתן לשלוח כל מערך דו-מימדי מאותו סוג, המכיל אותו מספר עמודות.

פרק 10

מחרוזות

כפי שבוודאי שמת לב, עד כה קלטנו מהמשתמש מספרים ותווים בודדים, אך לא מילים ומשפטי מלל. לכאורה, נראה הדבר מוזר, שהרי בעוד שביצענו פעולות מורכבות למדי, לא יכולנו לבצע פעולה פשוטה כגון קליטת שם המשתמש. ובכן, קליטת מלל מהמשתמש ועיבודו הינם מורכבים יותר מקליטת ערכים מספריים ועיבודם, ולשם כך עלינו ליישם את הידע שנלמד בפרק הקודם. הכל אודות זאת – בפרק זה.

הקבוע המחרוזתי

קבוע מחרוזתי (string constant או string literal) הוא מלל התחום בגרשיים. נתקלנו בקבועים מחרוזתיים כבר בפרק הרביעי, כאשר שלחנו אותם כפרמטרים לפונקציה printf כדי להציגם על המסך. הנה מספר דוגמאות של קבועים מחרוזתיים:

```
"is it heaven or hell?"
"I'll be back..."
""
```

כל הקבועים המחרוזתיים מורכבים מתווים. תו בודד רשום בין גרש יחיד מכל צד, אך כאשר מספר תווים מקובצים לקבוע מחרוזתי, הם מתוחמים בגרשיים. למשל, התווים הבודדים היוצרים את שם מדינתנו מופיעים בין גרשיים יחידים:

```
'I'  's'  'r'  'a'  'e'  'l'
```

אך אם נאחדם למחרוזת, הם יירשמו בין גרשיים, כך:

```
"Israel"
```

שים לב שאין לרשום מספר תווים רצופים בין גרשיים יחידים (פרט לתווי בקרה). למשל, השורה הבאה אינה נכונה:

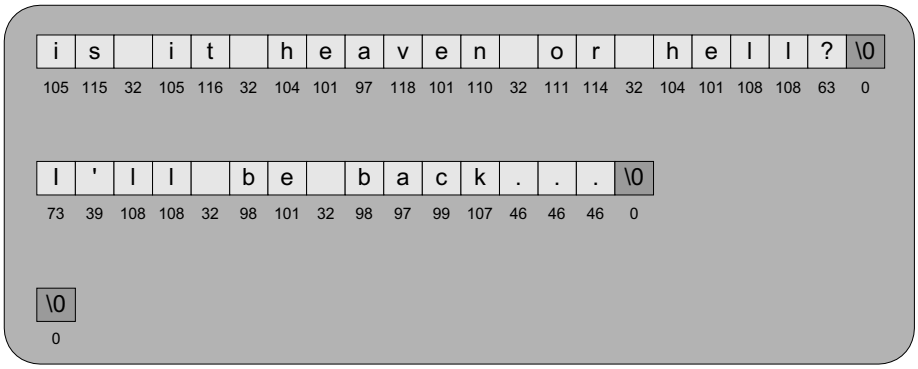
```
'Israel'      /* wrong, this is not a string! */
```

כל קבוע מחרוזתי מורכב מתו **נוסף**, מלבד התווים הרגילים שמרכיבים אותו. זהו תו שערך ASCII שלו הוא 0, ונהוג לייצגו כך: '\0'. תו זה אינו נראה על המסך כאשר רושמים מלל בין גרשיים, אך הוא מתוסף אל הקבוע המחרוזתי באופן אוטומטי. תו זה מציין את סוף הקבוע המחרוזתי, וכפי שנראה בהמשך, הוא חשוב לאין ערוך.

שים לב!

התו '\0' מכונה **תו הנול** (null character). מאחר שסימנו של תו מוחלף תמיד בערך המספרי המתאים לו בטבלת ASCII, ניתן לרשום במקום התו '\0' את המספר 0. אנו נעדיף לייצגו כ-'\0', כדי להדגיש שמדובר בתו המסיים מחרוזת. אחת היא במה תבחר, שים לב כי התו '\0', שערכו 0, שונה מהתו '0', שערכו 48!

האיור הבא מציג את התווים המרכיבים את הקבועים המחרוזתיים שהצגנו זה עתה:



***איור 10.1** – תווים המרכיבים קבועים מחרוזתיים*
(המספרים מתחת לתווים מציינים את ערכיהם המספריים בטבלת ASCII)

שים לב שהתו '\0' מופיע בסופו של כל קבוע מחרוזתי. ניתן לומר כי מספר התווים המרכיבים קבוע מחרוזתי תמיד גדול ב-1 ממספר התווים הגלויים לעין כאשר הוא מוצג על המסך.

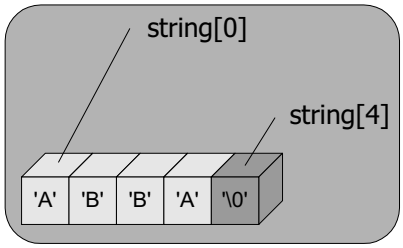
המחרוזת כמערך תווי חד-מימדי

הקבוע המחרוזתי כשמו כן הוא – קבוע. אם נרצה לעבד מלל באופנים שונים (למשל, לקלוט אותו מהמשתמש, לשנות את תוויו ועוד), יהיה עלינו להכיר משתנה בו ניתן לאחסן קבועים מחרוזתיים. משתנה כזה מכונה **מחרוזת** (string). המחרוזת היא מערך תווי חד-מימדי, שכל אחד מאיבריו מכיל תו אחר של הקבוע המחרוזתי. אין לשכוח את התו '\0' – גם עליו להופיע באיבר המתאים במחרוזת.

למשל, כדי להגדיר מחרוזת שתוכל לאחסן את הקבוע המחרוזתי "ABBA", נגדיר מערך תווי בן חמישה איברים, נציב בארבעת איבריו הראשונים את התווים 'A', 'B', 'B' ו-'A', ובאיברו האחרון – את התו '\0'. נבצע זאת כך:

```
char string[5];
string[0] = 'A';
string[1] = 'B';
string[2] = 'B';
string[3] = 'A';
string[4] = '\0';
```

מראה המחרוזת string לאחר ביצוע שורות אלו מוצג באיור הבא:



***איור 10.2** – איברי מחרוזת המכילה את תווי הקבוע המחרוזתי "ABBA"*

איך לא...

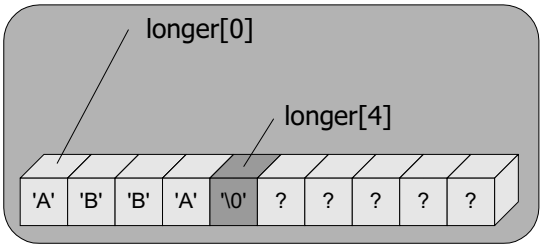
אין להציב קבועים מחרוזתיים ישירות במחרוזת. לכן, קטע התוכנית הבא אינו חוקי:

```
char string[5];
/* ... */
string = "ABBA";     /* wrong! */
```

נוכל להגדיר מחרוזת גדולה יותר, ולהציב את תווי הקבוע המחרוזתי "ABBA" רק בחמשת איבריה הראשונים. למשל, לאחר שורות הקוד הבאות:

```
char longer[10];
longer[0] = 'A';
longer[1] = 'B';
longer[2] = 'B';
longer[3] = 'A';
longer[4] = '\0';
```

תיראה המחרוזת longer כך:



איור 10.3 *– איברי מחרוזת ארוכה יותר המכילה את תווי הקבוע המחרוזתי "ABBA"*
(ערכי האיברים המיוצגים על ידי סימן שאלה אינם ידועים)

במחשבה שנייה...

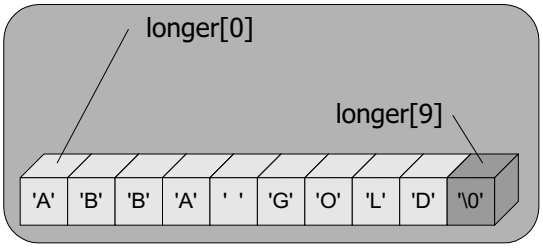
מהי החשיבות בהוספת התו '\0'?

אילו הצבנו במחרוזת longer את התווים 'A', 'B', 'B' ו-'A' בלי להשתמש בתו '\0', היינו מוגבלים בפעולות שיכולנו לבצע באמצעותה. בהמשך נראה כי הוספת התו '\0' מאפשרת לעבד את המחרוזת בדרכים רבות.

ביכולתנו להציב ערכים גם בחמשת איבריה האחרונים של המחרוזת longer, ובכך להאריך את הקבוע המחרוזתי שהיא מייצגת. למשל, אם נבצע את שורות הקוד הבאות, לאחר קטע התוכנית הקודם:

```
longer[4] = ' ';
longer[5] = 'G';
longer[6] = 'O';
longer[7] = 'L';
longer[8] = 'D';
longer[9] = '\0';
```

תיראה המחרוזת longer כך:



איור 10.4 *– איברי מחרוזת המכילה את הקבוע המחרוזתי "ABBA GOLD"*

אתחול מחרוזות

בפרק הקודם ראינו כי ניתן לאתחל כל מערך בשורת הגדרתו. המחרוזת, בהיותה מערך תווי חד-מימדי, אינה יוצאת דופן. דרך אחת לאתחל אותה היא כפי שלמדנו בפרק הקודם, כתיבת התווים שיש להציב בה כערכים מאתחלים. למשל, כדי להגדיר מחרוזת ולהציב בה את תווי הקבוע המחרוזתי "to err is human", נכתוב שורת הגדרה כזו:

```
char saying[16] = { 't', 'o', ' ', 'e', 'r', 'r', ' ', 'i',
                    's', ' ', 'h', 'u', 'm', 'a', 'n', '\0' };
```

שיטת אתחול כזו הינה מסורבלת למדי, ולמזלנו, קיימת שיטה נוחה יותר. במקום לכתוב את כל תווי הקבוע המחרוזתי בנפרד, ניתן להציב במחרוזת ישירות את הקבוע המחרוזתי. למשל, את שורת ההגדרה הקודמת ניתן לכתוב כך:

```
char saying[16] = "to err is human";
```

באופן זה, מוצבים אוטומטית כל התווים שבקבוע המחרוזתי "to err is human" באיברים המתאימים של המחרוזת saying. חשוב לציין, כי משפט השמה כזה הינו חוקי רק בשורת ההגדרה של המחרוזת, ובמקום אחר בתוכנית אסור לבצעו. למשל, קטע התוכנית הבא אינו נכון:

```
char saying[16];
/* ... */
saying = "to err is human";   /* wrong assignment! */
```

כזכור, כאשר מאתחלים מערך חד-מימדי ומשמיטים את גודלו, מספר איבריו נהיה כמספר הערכים המאתחלים. באופן דומה, אם נאתחל מחרוזת בקבוע מחרוזתי בלי לציין את גודלה, יהיה מספר איבריה כמספר התווים שבקבוע המחרוזתי (כולל התו '\0'). למשל, שורת ההגדרה הבאה:

```
char food[] = "loaf";
```

מגדירה מחרוזת בת חמישה איברים, המכילים את התווים 'l', 'o', 'a', 'f' ו-'\0', בהתאמה.

איך לא...

ניתן להשמיט את גודל המחרוזת רק כאשר היא מאותחלת במחרוזת אחרת. לכן, שורת ההגדרה הבאה אינה נכונה:

```
char food[];  /* wrong, initialize values required */
```

בתוכנית זו נגדיר מחרוזת ונאתחל אותה בתווי הקבוע המחרוזתי "!Howdy". לאחר מכן נסתמך על כך שהאיבר האחרון במחרוזת הוא '\0', כדי לפלוט את תוויה בהדרגה.
המחרוזת תיפלט מספר פעמים השווה למספר התווים שבה (לא כולל התו '\0'). בפעם הראשונה היא תיפלט בשלמותה. בפעם השנייה היא תיפלט החל מהתו השני שבה. בפעם השלישית היא תיפלט החל מהתו השלישי שבה, וכן הלאה.

תוכנית 10.1 *PARTS.C*

```
#include <stdio.h>
void main()
{
    char string[] = "Howdy!";
    int i, j;

    printf("\n");
    for (i = 0 ; string[i] != '\0' ; i++) {
        for (j = i ; string[j] != '\0' ; j++)
            printf("%c", string[j]);
        printf("\n");
    }
}
```

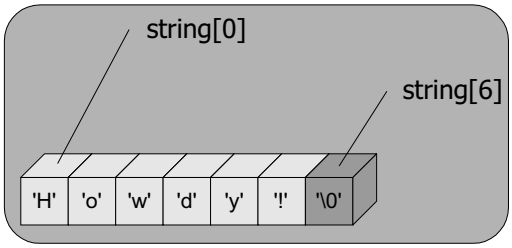
פלט התוכנית נראה כך:

```
Howdy!
owdy!
wdy!
dy!
y!
!
```

```
    char string[] = "Howdy!";
    int i, j;
```

בשורה זו מאותחלת המחרוזת string בתווי הקבוע המחרוזתי "!Howdy". בקבוע מחרוזתי זה 7 תווים (כולל התו '\0').

זה גם מספר האיברים של המחרוזת string, הנראית כך:



איור 10.5 *– איברי המחרוזת string לאחר אתחולה*

במשתנים i ו-j נשתמש כמשתנים מונים.

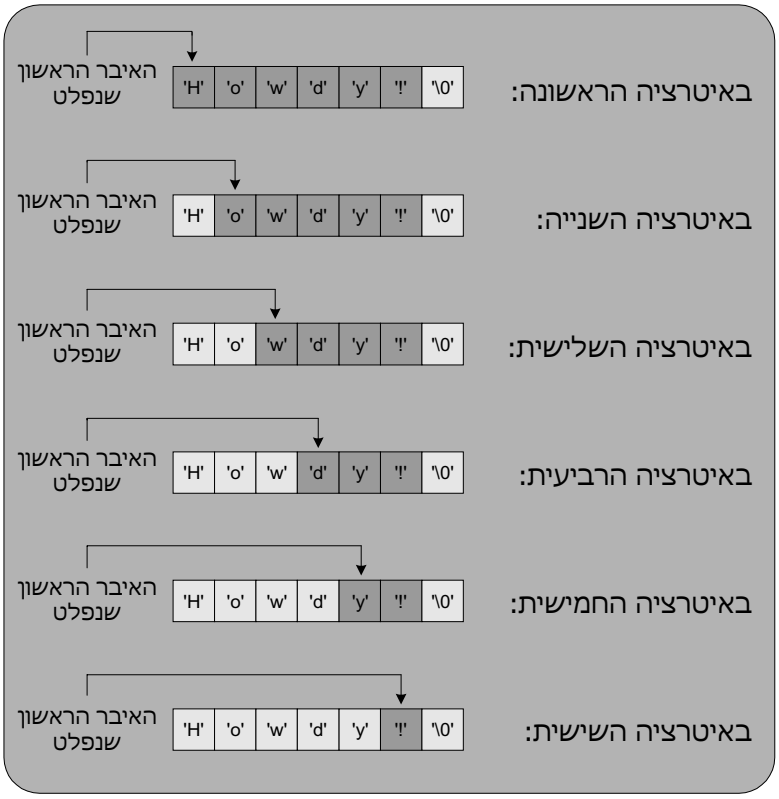
```
for (i = 0 ; string[i] != '\0' ; i++) {
    /* ... */
```

ערך המשתנה i הוא 0 לפני האיטרציה הראשונה, ובסוף כל איטרציה, הוא עולה ב-1. ביטוי הלולאה בודק בכל איטרציה איבר אחר במחרוזת, והוא אינו מתקיים כאשר נמצא איבר שערכו '\0'. אינדקס האיבר שערכו '\0' הוא 6, ולכן הלולאה תתבצע 6 פעמים, כאשר ערך i עולה מ-0 ל-5 בהפרשים של 1.

```
    for (j = i ; string[j] != '\0' ; j++)
        printf("%c", string[j]);
```

לולאה פנימית זו אחראית על פליטת איברי המערך string. ראשית, נזכיר את הפלט שברצוננו לקבל. ברצוננו לפלוט את המחרוזת string מספר פעמים השווה למספר התווים שבה (לא כולל התו '\0'). בפעם הראשונה יש לפלוט את המחרוזת בשלמותה. בפעם השנייה יש לפלוט את המחרוזת החל מהתו השני שלה. בפעם השלישית יש לפלוט את המחרוזת החל מהתו השלישי שלה, וכן הלאה.

האיור הבא מדגים זאת:



***איור 10.6** – אופן בדיקת איברי המחרוזת string בכל איטרציה*
(האיברים הכהים הם האיברים שנפלטים על המסך)

הלולאה הפנימית פולטת בשורה בודדת את תווי המחרוזת string, החל מהאיבר שמיקומו j. לפני כל כניסה ראשונה ללולאה זו, מוצב במשתנה j ערך המשתנה i, העולה ב-1 לאחר כל פעם שנפלטה שורה של המחרוזת. לכן, בכל פעם, האיבר הראשון שנפלט הוא האיבר שאחרי האיבר הקודם שנפלט ראשון.

המחרוזות והמשתמש

מחרוזות שונות ממערכים חד-מימדיים רגילים בכך שניתן לפלוט את איבריהן על המסך ולקלוט אליהם ערכים מהמשתמש בלי להשתמש בלולאות. בסעיף זה נלמד כיצד לעשות זאת.

פלט מחרוזות

כאשר נפלוט מחרוזת מסוימת, יוצגו על המסך כל התווים המרכיבים אותה, פרט לתו האחרון שבה – התו '\0'. כדי לפלוט מחרוזת, כל שיש לעשות הוא לשלוח אותה כפרמטר נוסף אל הפונקציה printf, ולציין במקום המבוקש במחרוזת הבקרה תו המרה מתאים. תו ההמרה המתאים למחרוזות הוא תו המרה חדש, הנראה כך: %s. למשל, אם נבצע את השורות הבאות:

```
char message[] = "I see thee";
printf("%s", message);    /* note: no index specified */
```

ייפלט המלל הבא:

```
I see thee
```

תו ההמרה %s מיועד במיוחד למחרוזות, ולכן הפרמטר המתאים לו אינו איבר במחרוזת message (איבר במחרוזת הוא תו, ולתו מתאים תו ההמרה %c). אילו ביצענו את משפט הפלט כך:

```
printf("%s", message[0]);    /* don't specify index! */
```

לא היתה מתקבלת הודעת שגיאה מהמהדר, אך היה נפלט מלל מוזר ובלתי-רצוי.

שים לב!

רק למחרוזות קיים תו המרה המאפשר לפלוט את ערכי כל איבריהן במשפט פלט בודד. אם ברצונך לפלוט את כל איבריו של מערך שאינו מחרוזת, עליך להשתמש בלולאות.

הפונקציה printf פולטת את כל תווי המחרוזת הנשלחת אליה, עד שהיא נתקלת בתו '\0'. במחרוזת message תו זה נמצא באיבר האחרון, אך אם נציב את הערך '\0' באיבר השישי שלה, כך:

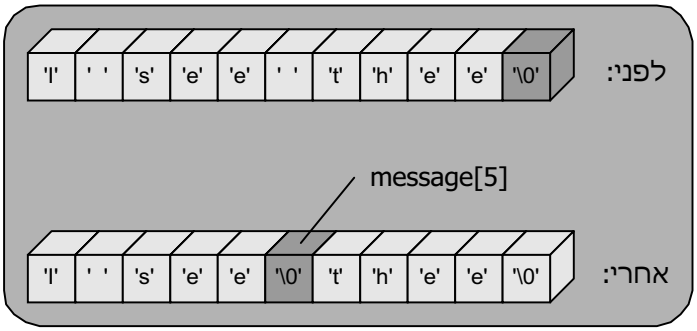
```
message[5] = '\0';
```

ונשלח אותה לפונקציה printf כפי שעשינו קודם, יתקבל הפלט הבא:

```
I see
```

ההסבר לפלט הוא שהפונקציה printf אינה פולטת את התווים המופיעים לאחר התו '\0' הראשון בו היא נתקלת. לאחר משפט ההשמה נמצא תו זה באיבר השישי, ולכן התווים שבאיברים הבאים אחריו אינם נפלטים.

האיור הבא מציג את איברי המחרוזת message לפני ואחרי השמת הערך '\0' באיבר השישי:



איור 10.7 *– איברי המחרוזת message לפני ואחרי השמת הערך '\0' באיבר החמישי*

זהירות!

אם נשלח אל printf מחרוזת שאינה מסתיימת בתו '\0', ייפלטו על המסך ערכים בלתי-צפויים. נסביר בהמשך מדוע הדבר קורה; בינתיים, וודא תמיד שכל מחרוזת הנפלטת למסך מסתיימת בתו זה.

ניתן לפלוט על ידי printf גם קבועים מחרוזתיים. למשל, ביצוע משפט הפלט הבא:

```
printf("%s", "Men with manes are meaner than all men");
```

יביא לפליטת המלל:

```
Men with manes are meaner than all men
```

ולאחר ביצוע משפט כזה:

```
printf("%s is %c healthy %s %c", "coke", '!', "4", 'u');
```

יתקבל המלל הבא:

```
coke is ! healthy 4 u
```

במחשבה שנייה...

מהו ההבדל בין הקבועים '4' ו-"4"?

הקבוע '4' הוא קבוע תווי והקבוע "4" הוא קבוע מחרוזתי. אם נרצה לפלוט אותם באמצעות printf, נתאים ל-'4' את תו ההמרה %c ול-"4" את תו ההמרה %s. שים לב שהקבוע "4", בהיותו קבוע מחרוזתי, מורכב משני תווים ('4' ו-'\0'), בעוד שהקבוע '4' מורכב מתו בודד.

קליטת מחרוזות

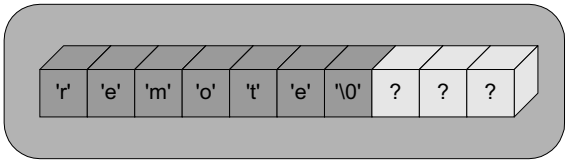
קליטת מחרוזות מהמשתמש נעשית אף היא בקלות. לשם כך נשלח מחרוזת כפרמטר נוסף אל הפונקציה scanf, ונציין במחרוזת הבקרה שלה את תו ההמרה %s. למשל, קטע התוכנית הבא קולט מלל מהמשתמש ומציבו באיברי המחרוזת input:

```
char input[10];
printf("Please enter a string: ");
scanf("%s", &input);      /* note: no index specified */
```

הפלט המתקבל מביצוע שורות אלו הוא כדלהלן:

```
Please enter a string: remote
```

לאחר הזנת המלל "remote", ייראו איברי המערך כך:



***איור 10.8** – המחרוזת input לאחר קליטת ערכים*
(האיברים הכהים הם האיברים שאת ערכיהם שינתה scanf)

ניתן לראות שהפונקציה scanf הציבה באיברי המחרוזת את התווים שהוזנו, ובסופם – את התו '\0'. שאר איברי המחרוזת לא שונו (ערכיהם של איברים אלה אינם ידועים, מאחר שהמחרוזת לא אותחלה).

שים לב!

רק למחרוזות קיים תו המרה המאפשר לקלוט מספר ערכים (תווים) מהמשתמש ולהציבם באיבריהן במשפט קלט בודד. אם ברצונך לקלוט ערכים ולהציבם באיבריו של מערך שאינו מחרוזת, השתמש בלולאות.

המחרוזת שנשלחה אל scanf מכילה עשרה איברים. באיבר אחד עלינו לאחסן את התו '\0', ולכן, ניתן לקלוט באמצעותה עד תשעה תווים מהמשתמש. אם יזין המשתמש מחרוזת בת יותר מתשעה תווים, הדבר עלול לגרום לקריסת התוכנית ומערכת ההפעלה.

עצה

וודא תמיד כי כל מחרוזת אליה נקלט מלל מהמשתמש מכילה די איברים. למשל, כדי לקלוט את שם משפחתו של המשתמש, נשתמש במחרוזת שאורכה 20 תווים (ייתכן שהשם שיזין המשתמש יהיה "dan", וייתכן שיהיה "rumpelstiltskin"!).

שים לב כי לא שלחנו אל scanf את אחד מאיברי המחרוזת input. אילו ביצענו את משפט הקלט הבא:

```
scanf("%s", &input[2]);      /* legal, but confusing */
```

היה מוצב המלל הנקלט החל מהאיבר השלישי במערך. נסביר את הסיבה לכך בהמשך.

כאשר נשלחת מחרוזת לפונקציית קלט בלי שמצוין אינדקס, לא חובה להקדים את שמה באופרטור הכתובת. לכן, שני משפטי הקלט הבאים מבצעים פעולה זהה:

```
scanf("%s", &input);          scanf("%s", input);
```

לא מקובל להשתמש באופרטור הכתובת לשווא, ולכן נעדיף להשמיטו כאשר נשלח מחרוזת לפונקציית קלט.

זהירות!

ניתן לוותר על אופרטור הכתובת רק כאשר שולחים מחרוזת לפונקציה בלי לציין אינדקס. במקרים הבאים, למשל, חובה להשתמש באופרטור הכתובת:

```
scanf("%c", &ch);
scanf("%d", &array[index]);
```

הפונקציה gets

עד כה השתמשנו בתו הרווח כדי להפריד בין שני הערכים המוזנים. למשל, כאשר תבוצע הוראת קלט כזו:

```
scanf("%d %d", &num1, &num2);      /* read two values */
```

נזין שני מספרים ונפריד אותם בתו רווח. באופן דומה, אם נזין מחרוזת המכילה תו רווח, הוא ייחשב כמפריד בין שתי מחרוזות מוזנות. למשל, אם במקום המחרוזת "remote" היינו מזינים את המחרוזת "be there", היתה scanf מניחה שהזנו שתי מחרוזות נפרדות ("be" ו-"there"), ורק את הראשונה ("be") היתה מציבה במחרוזת input.

לא ניתן לעקוף מגבלה זו, כלומר, על ידי הפונקציה scanf לא ניתן לקלוט מלל המורכב מרווחים. כדי להתגבר על כך נכיר פונקציית קלט חדשה, שתאפשר לקלוט מחרוזות המכילות תווי רווח. פונקציה זו נקראת gets, והיא אף פשוטה יותר לביצוע מאשר scanf. כל שיש לעשות כדי לקלוט באמצעותה מחרוזת, הוא לשלוח אותה אליה כפרמטר. למשל, במקום הקריאה הבאה ל-scanf:

```
scanf("%s", string);
```

נבצע משפט כזה:

```
gets(string);
```

הפונקציה gets תציב במחרוזת את כל התווים המוזנים, כולל רווחים, עד להקשה על מקש Enter. לבסוף היא תציב באיבר המתאים במחרוזת את התו '\0'. כאשר קולטים מחרוזות העלולות להכיל תווי רווח, נהוג להשתמש ב-gets ולא ב-scanf.

איך לא...

לא ניתן לקלוט יותר ממחרוזת אחת באמצעות הוראת gets בודדת. המשפט הבא, למשל, אינו נכון:

```
gets(string1, string2, string3);      /* wrong */
```

כמו printf ו-scanf, הפונקציה gets מצריכה את הוספת ההוראה #include <stdio.h> לראש התוכנית.

דוגמה

בדוגמה זו נכתוב תוכנית הקולטת מחרוזת מהמשתמש ומוצאת האם היא מחרוזת פלינדרומית. מחרוזת פלינדרומית היא מחרוזת הנקראת מההתחלה לסוף כפי שהיא נקראת מהסוף להתחלה. הנה מספר דוגמאות למחרוזות פלינדרומיות:

```
x      alla      123m5m321      britaatirb
```

תוכנית 10.2

PAL_STR.C

```
#include <stdio.h>
void main()
{
    char string[20];
    int length, is_palin = 1, i;

    printf("Please enter a string: ");
    gets(string);
```

```
    length = 0;
    while (string[length] != '\0')
        length++;
    length--;

    for (i = 0 ; i < length / 2 && is_palin == 1; i++)
        if (string[i] != string[length - i])
            is_palin = 0;

    if (is_palin == 1)
        printf("\"%s\" is a palindromic string! \n", string);
    else
        printf("\"%s\" is not a palindromic string! \n", string);
}
```

הנה מספר פלטים המתקבלים מהתוכנית:

```
Please enter a string: aquauqa
"aquauqa" is a palindromic string!
```

```
Please enter a string: xyz xyz
"xyz xyz" is not a palindromic string!
```

```
    char string[20];
    int length, is_palin = 1, i;
```

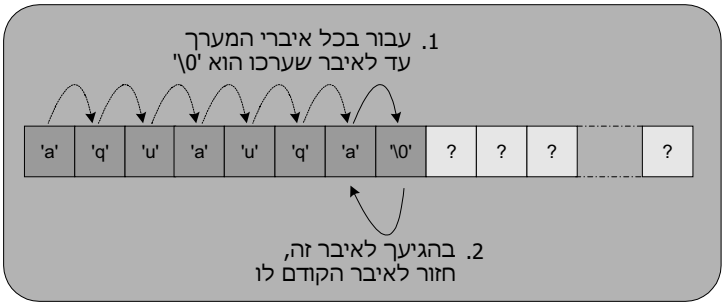
במחרוזת string יוצב המלל שיזין המשתמש. מאחר שבמחרוזת זו עשרים איברים, ניתן לקלוט מהמשתמש עד 19 תווים (זכור כי על אחד מאיברי המחרוזת להכיל את הערך '\0'). איננו יודעים כמה תווים יזין המשתמש, אך בהמשך נחשב זאת באמצעות המשתנה length. המשתנה is_palin מאותחל לערך 1, ואם במהלך התוכנית נראה כי המחרוזת אינה פלינדרומית, נציב בו את הערך 0. כך נוכל לדעת, בסוף התוכנית, אם המחרוזת הינה פלינדרומית או לא. המשתנה i הוא משתנה מונה פשוט.

```
    gets(string);
```

בשורה זו מבוצעת הוראת הקלט gets, הממתינה להזנת מחרוזת ומציבה את תוויה באיברי המחרוזת string. אם שורה זו היתה מוחלפת בקריאה לפונקציה scanf והיתה מוזנת מחרוזת המכילה תווי רווח, לא היתה התוכנית פועלת כהלכה.

```
length = 0;
while (string[length] != '\0')
    length++;
length--;
```

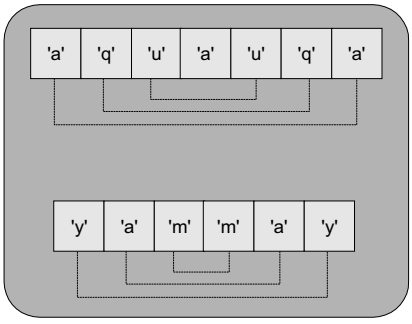
במשפטים אלה מחושב אינדקס איבר המחרוזת string המכיל את התו האחרון שהזין המשתמש. אינדקס זה מוצב במשתנה length, באופן הבא: ראשית, מוצב הערך 0 במשתנה length. לולאת while מתבצעת כל עוד איבר המחרוזת string במיקום length אינו '\0', ולכן, כאשר היא תסתיים, יהיה length אינדקס האיבר שערכו '\0'. המשפט הבא, המפחית את ערך length ב-1, מציב בו את אינדקס האיבר המכיל את התו האחרון שהזין המשתמש. האיור הבא מדגים את פעולת המשפטים הללו:



***איור 10.9** – מציאת אינדקס האיבר המכיל את התו האחרון שהוזן*

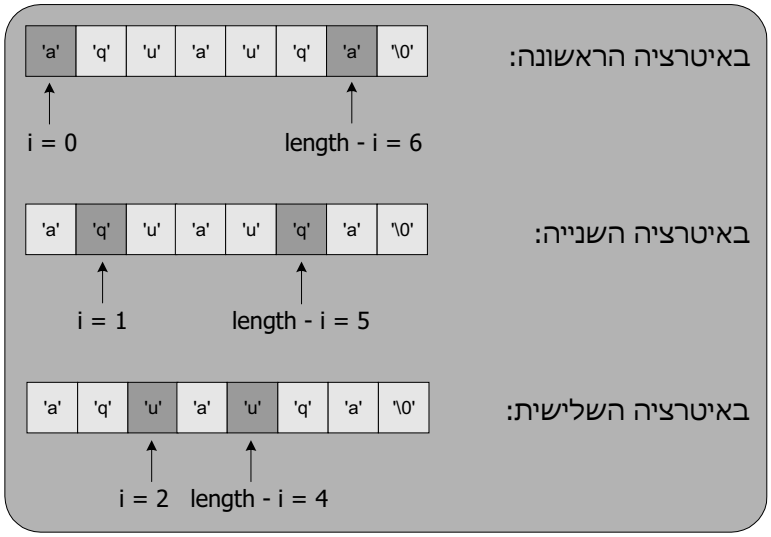
```
for (i = 0 ; i < length / 2 && is_palin == 1; i++)
    if (string[i] != string[length - i])
        is_palin = 0;
```

במחרוזת פלינדרומית כל שני איברים הסימטריים למרכז המחרוזת שווים. זוגות כאלה מוצגים באיור 10.10. לולאה זו מוצאת האם המחרוזת string הינה מחרוזת פלינדרומית על ידי בדיקת כל שני איברים הסימטריים למרכז המחרוזת. אם יימצאו שני איברים סימטריים השונים זה מזה, נדע כי המחרוזת אינה פלינדרומית.



***איור 10.10** – במחרוזת פלינדרומית כל שני איברים הסימטריים למרכז המחרוזת שווים*

לפני האיטרציה הראשונה של הלולאה, מכילים איברי המחרוזת string במיקומים i ו-length את התו הראשון והאחרון שהזין המשתמש, בהתאמה. בכל איטרציה נבדקים האיברים בעלי האינדקסים i ו-length - i, ולאחר כל איטרציה עולה ערך i ב-1. באופן זה, נבדקים זוגות האיברים הבאים:



***איור 10.11** – האיברים הסימטריים במחרוזת string הנבדקים בכל איטרציה*

ביטוי הלולאה מוודא שזוג האיברים במיקומים i ו-length - i נבדק עד שמגיעים למרכז המחרוזת. אם נמצאו שני איברים סימטריים השונים זה מזה, אזי המחרוזת אינה פלינדרומית, והערך 0 יוצב במשתנה is_palin. במקרה זה אין צורך בבדיקות נוספות של הלולאה, ואכן ביטוי הלולאה לא יתקיים יותר (בשל אופרטור הוגם הלוגי, הדורש כי המשתנה is_palin יכיל את הערך 1).

פונקציות ספריה לטיפול במחרוזות

ניתן לבצע פעולות רבות במחרוזות תוך יישום הידע שרכשנו, אך הדבר עלול להיות מסורבל במקצת. פעולות שגרתיות, כגון מציאת אורכה של מחרוזת או העתקת מחרוזת אחת לאחרת, מבוצעות פעמים רבות בתוכניות מורכבות. בכל פעם שנבצען, נצטרך להגדיר משתנים נוספים ולבצע מספר משפטים, ובסופו של דבר, תוכניתנו תהיה מסורבלת וארוכה יתר על המידה. למזלנו, פעולות כאלו ניתן לבצע על ידי קריאה פשוטה וקצרה לפונקציות מתאימות.

פונקציות לעיבוד מחרוזות, כמו פונקציות קלט ופלט, אינן כלולות במהדר שפת C. כפיצוי על כך, קבצי הספריה המסופקים עם המהדר מכילים פונקציות רבות ומגוונות לטיפול

במחרוזות. לפונקציות אלו נקרא מתוכניתנו, כפי שקראנו ל-printf, ל-scanf ול-gets. בסעיף זה נכיר כמה מהנפוצות שבהן.

חשוב לציין שרוב הפונקציות לעיבוד מחרוזות מסתמכות על כך שכל מחרוזת מסתיימת בתו '\0'. אם נשלח לפונקציות אלו מחרוזת שאינה כוללת תו זה, לא תתקבל התוצאה לה ציפינו, והתוכנית אף עלולה לקרוס.

כאשר נקרא לפונקציות ספריה המטפלות במחרוזות, נוסיף לתוכניתנו את הוראת קדם-המעבד הבאה:

```
#include <string.h>
```

הוראה זו גורמת למהדר להתייחס אל נתונים שונים הנמצאים בקובץ string.h. קובץ זה מכיל מידע לגבי פונקציות המחרוזות בהן נשתמש, והכללתו נחוצה לשם ביצועה התקין של התוכנית.

שים לב!

הוראה זו תופיע לפני או אחרי ההוראה #include <stdio.h>; היא אינה באה במקומה.

מציאת אורך מחרוזת

לעיתים קרובות נרצה למצוא אורך של מחרוזת נתונה. נוכל לבצע זאת על ידי לולאה ומשתנה מונה, אך נוח יותר לקרוא לפונקציית ספריה מיוחדת, המבצעת פעולה זו בשבילנו. פונקציה זו נקראת strlen; היא מקבלת מחרוזת כפרמטר ומחזירה את מספר התווים שבה, עד לתו '\0' הראשון (לא כולל). אם, למשל, נקרא לה באופן הבא:

```
strlen("spleen")      /*  returns 6 */
```

יוחזר הערך השלם 6. בערך המוחזר מהפונקציה נשתמש כאילו היה ערך קבוע רגיל. נוכל, למשל, להציבו במשתנה:

```
length = strlen("spleen");      /* assign 6 to length */
```

או לשלוח אותו כפרמטר לפונקציית פלט:

```
printf("length of \"spleen\": %d", strlen("spleen"));
```

ביצוע שורה זו יביא לפליטת המלל הבא:

```
length of "spleen": 6
```

נהוג לקרוא לפונקציה strlen כדי לבדוק אורך מחרוזת שאינה קבוע מחרוזתי (שהרי אורכו של קבוע מחרוזתי אף הוא קבוע, וניתן לדעת אותו מראש). הנה דוגמה לכך:

```
char str[20];
int length;
printf("Please enter a string: ");
gets(str);
length = strlen(str);
printf("You've entered a %d-char long string.", length);
```

הפלט המתקבל מקטע תוכנית זה הוא כדלהלן:

```
Please enter a string: appendix
You've entered a 8-char long string.
```

דוגמה

בדוגמה זו נתרגל את השימוש בפונקציה strlen. התוכנית שנכתוב תקלוט מהמשתמש חמש מחרוזות קצרות, ותפלוט את האורך של כל מחרוזת ואת מספר התווים שנקלטו בסך-הכל.

תוכנית 10.3 *STRLEN.C*

```
#include <stdio.h>
#include <string.h>
void main()
{
    char string[20];
    int lengths[5], total_len = 0, i;

    for (i = 0 ; i < 5 ; i++) {
        printf("Please enter string #%d: ", i + 1);
        gets(string);

        lengths[i] = strlen(string);
        total_len += lengths[i];
    }
```

```
    printf("\nThe lengths of the strings you've entered are: \n");
    for (i = 0 ; i < 5 ; i++)
        printf("string #%d: %d \n", i + 1, lengths[i]);
    printf("Total characters read: %d \n", total_len);
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter string #1: learn
Please enter string #2: C
Please enter string #3: right
Please enter string #4: now
Please enter string #5: !!

The lengths of the strings you've entered are:
string #1: 5
string #2: 1
string #3: 5
string #4: 3
string #5: 2
Total characters read: 16
```

```
#include <string.h>
```

בשורה זו מופיעה הוראת קדם-מעבד הגורמת למהדר להתייחס אל הכתוב בקובץ string.h. זכור להוסיף שורה זו לכל תוכנית בה מתבצעת קריאה לפונקציות ספריה של מחרוזות.

```
    char string[20];
    int lengths[5], total_len = 0, i;
```

במחרוזת string יוצב המלל שיזין המשתמש. באיברי המערך lengths נציב את אורכה של כל אחת מחמש המחרוזות שיזין המשתמש. המשתנה total_len יכיל את מספר התווים שהוזנו, כלומר, את סכום ערכי איברי המערך lengths. המשתנה i הוא משתנה מונה פשוט.

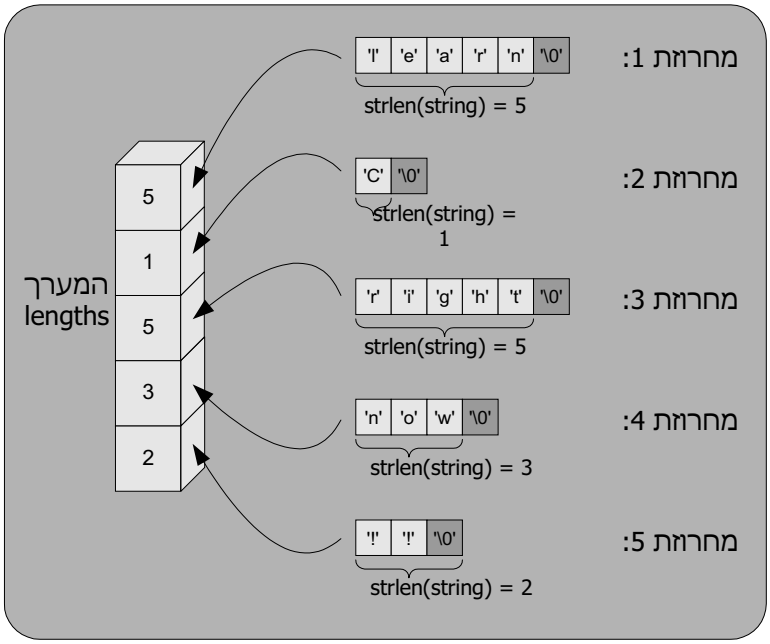
```
    for (i = 0 ; i < 5 ; i++) {
        printf("Please enter string #%d: ", i + 1);
        gets(string);
        /* ... */
```

בכל אחת מחמש האיטרציות של לולאה זו, מתבקש המשתמש להזין מחרוזת. מחרוזת זו מוצבת במחרוזת string.

```
        lengths[i] = strlen(string);
        total_len += lengths[i];
```

משפט ההשמה הראשון קורא לפונקציה strlen כדי לחשב את אורך המחרוזת האחרונה שנקלטה. אורך זה מוצב באיבר המתאים במערך lengths, ומתוסף לערך המשתנה total_len.

פעולת משפט ההשמה הראשון מוצגת באיור הבא:



איור 10.12 *– הצבת אורך כל אחת מהמחרוזות הנקלטות באיבר מתאים במערך lengths*

```
    for (i = 0 ; i < 5 ; i++)
        printf("string #%d: %d \n", i + 1, lengths[i]);
    printf("Total characters read: %d \n", total_len);
```

בשורות אלו נפלטים ערכי איברי המערך lengths, המכילים את אורכי המחרוזות שנקלטו. כמו-כן נפלט ערך המשתנה total_len, המכיל את סכום כל אורכי המחרוזות שהוזנו.

העתקת מחרוזת

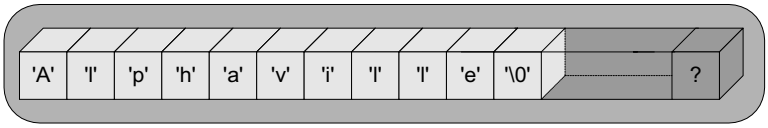
ראינו כי לאחר שהוגדרה מחרוזת, אין להציב בה ישירות מחרוזת אחרת. אם נרצה, למשל, לקלוט שם מהמשתמש ולהעתיקו לשלוש מחרוזות שונות, יהיה עלינו להשתמש בלולאה ולבצע פעולת השמה בין כל תו ותו שבמחרוזות.

ניתן להימנע מפעולה מייגעת זו על ידי שימוש בפונקציית ספריה הנקראת strcpy. פונקציה זו מקבלת שתי מחרוזות כפרמטרים, ומעתיקה לראשונה את השנייה. המחרוזת הראשונה מכונה **מחרוזת היעד**, ואליה מועתקת המחרוזת השנייה. המחרוזת השנייה

מכונה **מחרוזת המקור**, והיא אינה משתנה במהלך הקריאה לפונקציה. למשל, כדי להעתיק למחרוזת band את הקבוע המחרוזתי "Alphaville", נבצע את המשפט הבא:

```
strcpy(band, "Alphaville");
```

הפונקציה strcpy תעתיק את כל תווי הקבוע המחרוזתי, כולל את התו '\0', אל האיברים המתאימים במחרוזת band. לאחר ביצוע משפט זה, ייראו איברי המחרוזת band כך:



***איור 10.13** – איברי המחרוזת band לאחר העתקת תווי הקבוע המחרוזתי "Alphaville" אליה*

זהירות!

המהדר אינו בודק האם מחרוזת היעד מכילה די איברים כדי להכיל את מחרוזת המקור. אם, למשל, נבצע את קטע התוכנית השגוי הבא:

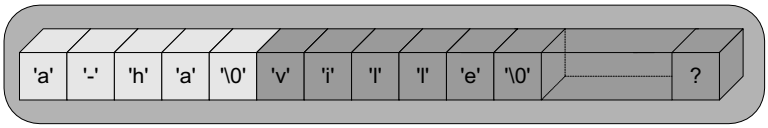
```
char target[5];
strcpy(target, "Hello there!");      /* dangerous!
*/
```

לא תיפלט הודעת שגיאה או אזהרה, אך התוכנית עלולה לקרוס.

אם נבצע שנית את הפונקציה strcpy, באופן הבא:

```
strcpy(band, "a-ha");
```

ישתנו חמשת האיברים הראשונים במחרוזת band. בארבעת האיברים הראשונים יוצבו התווים 'a', 'h', '-', ו-'a' בהתאמה, ובאיבר החמישי יוצב התו '\0'. כלומר, איברי המחרוזת band ייראו כך:



***איור 10.14** – strcpy מעתיקה את כל איברי מחרוזת המקור, עד לאיבר '\0' (כולל), למחרוזת היעד*

איך לא...

חשוב לזכור שמחרוזת היעד נשלחת ראשונה אל strcpy. אם נחליף את סדר שליחת הפרמטרים לפונקציה, ונכתוב משפט כזה:

```
strcpy("a-ha", band);      /* dangerous! */
```

לא תתקבל הודעת שגיאה או אזהרה, אך התוכנית עלולה לקרוס.

בדוגמה זו נתרגל את השימוש בפונקציה strcpy. התוכנית שנכתוב תקלוט מהמשתמש חמש מחרוזות, ולבסוף, תפלוט את הארוכה בהן.

תוכנית 10.4

STRCPY.C

```
#include <stdio.h>
#include <string.h>
void main()
{
    char str[20], longest[20];
    int i;

    printf("Please enter five strings: \n");

    for (i = 0 ; i < 5 ; i++) {
        gets(str);

        if (i == 0 || strlen(str) > strlen(longest))
            strcpy(longest, str);
    }

    printf("The longest string is: %s \n", longest);
}
```

פלט לדוגמה מהתוכנית מוצג להלן:

```
Please enter five strings:
Iconizer
DaTime & controls
Code
Monster
Employees
The longest string is: DaTime & controls
```

```
    char str[20], longest[20];
    int i;
```

בשורה זו מוגדרות שתי מחרוזות. במחרוזת str תאוחסן כל מחרוזת שיזין המשתמש, ובמחרוזת longest תאוחסן המחרוזת הארוכה ביותר שהוזנה עד כה. המשתנה i הוא משתנה מונה פשוט.

```
    for (i = 0 ; i < 5 ; i++) {
        gets(str);
        /* ... */
```

לולאה זו אחראית על קליטת המחרוזות ומציאת הארוכה בהן. בתחילת כל איטרציה נקלטת מהמשתמש מחרוזת בודדת אל המחרוזת str.

```
        if (i == 0 || strlen(str) > strlen(longest))
            strcpy(longest, str);
```

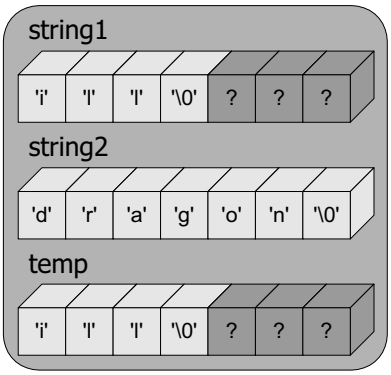
בכל איטרציה ברצוננו לאחסן את המחרוזת הארוכה ביותר שנקלטה עד כה במחרוזת longest. לזאת אחראי תת-הביטוי השני של ביטוי התנאי, הבודק האם המחרוזת האחרונה שהוזנה ארוכה יותר מהמחרוזת המאוחסנת ב-longest. אם כן, ברצוננו להציב אותה ב-longest, וזאת נעשה על ידי הקריאה לפונקציה strcpy.

כאשר נבדק ביטוי התנאי באיטרציה הראשונה של הלולאה, אורך המחרוזת longest אינו ידוע (שהרי לא אתחלנו אותה). לכן, באיטרציה זו בלבד, ברצוננו להציב את המחרוזת שהוזנה ב-longest, בלי קשר לאורכה. לזאת אחראי תת-הביטוי הראשון של ביטוי התנאי, המתקיים אם ערך המשתנה המונה i הוא 0.

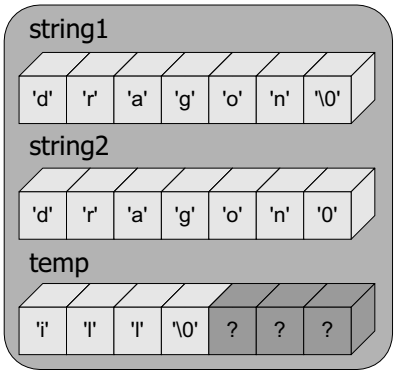
ניתן להיעזר בפונקציה זו כדי להחליף בין איבריהן של שתי מחרוזות. הדבר מודגם בקטע התוכנית הבא:

```
char string1[7] = "ill", string2[7] = "dragon", temp[7];
strcpy(temp, string1);
strcpy(string1, string2);
strcpy(string2, temp);
```

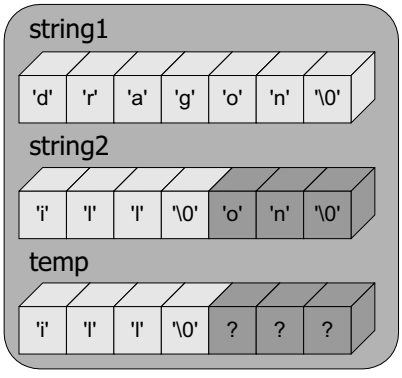
הקריאות לפונקציה strcpy מחליפות בין איבריהן של שתי המחרוזות. הקריאה הראשונה מעתיקה את איברי המחרוזת string1 ל-temp (ראה איור 10.15–א'). הקריאה הבאה מעתיקה את איברי המחרוזת string2 ל-string1 (ראה איור 10.15–ב'). הקריאה האחרונה מעתיקה את איברי המחרוזת temp, המכילים את תווי המחרוזת הראשונה שהוזנה, ל-string2 (ראה איור 10.15–ג'). לאחר שלוש הקריאות הללו, יכילו איברי המחרוזת string1 את תווי המחרוזת השנייה שהוזנה, ואיברי המחרוזת string2 יכילו את תווי המחרוזת הראשונה שהוזנה.



איור 10.15–א'



איור 10.15–ב'



איור 10.15–ג'

הוספת מחרוזת

כאשר השתמשנו בפונקציה strcpy, הועתקה מחרוזת המקור אל תחילתה של מחרוזת היעד, ובכך מחקה את תוויה הראשונים. לעיתים ברצוננו להוסיף מחרוזת אחת לסופה של מחרוזת אחרת, ובכך ליצור איחוד של השתיים. נניח שבידינו שתי מחרוזות: האחת מכילה את שמו הפרטי של המשתמש, והשנייה – את שם משפחתו. כדי לקבל מחרוזת המכילה את שמו המלא, נוסיף את המחרוזת השנייה למחרוזת הראשונה.

לשם כך נשתמש בפונקציית ספריה נוספת, הנקראת strcat. פונקציה זו מקבלת שתי מחרוזות, ומוסיפה את השנייה לראשונה. המחרוזת הראשונה מכונה **מחרוזת היעד**, ואליה מועתקת המחרוזת השנייה. המחרוזת השנייה מכונה **מחרוזת המקור**, והיא אינה משתנה במהלך הקריאה לפונקציה. הנה דוגמה לשימוש ב-strcat:

```
char proverb[38] = "the power of pen exceeds ";
char end[] = "the sword's.";
strcat(proverb, end);      /* add 'end' to 'proverb' */
printf("%s", proverb);
```

הפונקציה strcat מוסיפה למחרוזת proverb את המחרוזת end, ולכן הפלט המתקבל משורות אלו הוא:

```
the power of pen exceeds the sword's.
```

זהירות!

המהדר אינו בודק האם מחרוזת היעד מכילה די איברים כדי להכיל את מחרוזת המקור. אם, למשל, נבצע את קטע התוכנית השגוי הבא:

```
char target[10] = "Hello, ";
strcat(target, "world!");     /* dangerous! */
```

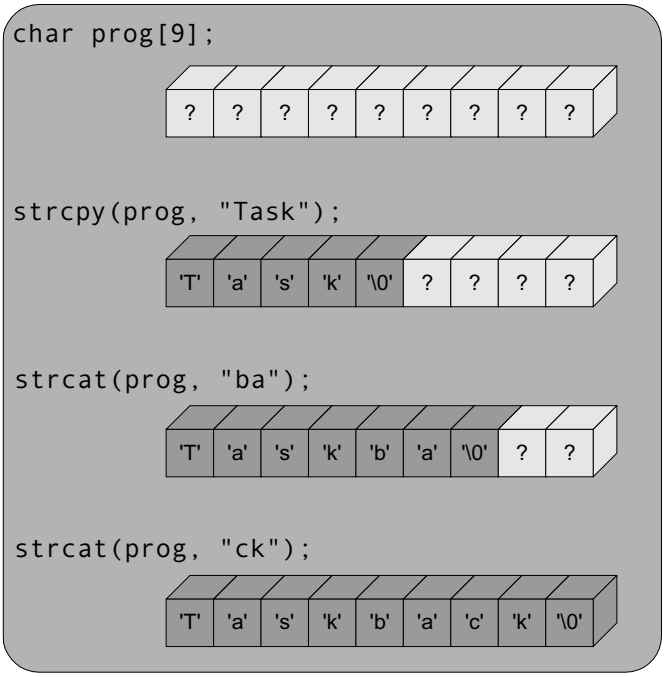
לא תיפלט הודעת שגיאה או אזהרה, אך התוכנית עלולה לקרוס.

הנה דוגמה מעט מורכבת יותר:

```
char prog[9];
strcpy(prog, "Task");
strcat(prog, "ba");
strcat(prog, "ck");
```

בתחילה מוגדרת מחרוזת בת 9 איברים, ומוצבים בה איברי קבוע המחרוזת "Task". פונקציית strcat הראשונה מוסיפה למחרוזת זו את הקבוע המחרוזתי "ba", ופונקציית strcat השנייה מוסיפה לה את הקבוע המחרוזתי "ck". לאחר שורות אלו, תכיל המחרוזת prog את איברי הקבוע המחרוזתי "Taskback".

האיור הבא מציג את איברי המחרוזת prog בשלבים השונים:



***איור 10.16** – איברי המחרוזת prog לאחר הקריאות השונות לפונקציות strcpy ו-strcat*

איך לא...

חשוב לזכור שמחרוזת היעד נשלחת ראשונה אל strcat. אם נחליף את סדר שליחת הפרמטרים לפונקציה, ונכתוב משפט כזה:

```
strcat("world! ", target);      /* dangerous! */
```

לא תתקבל הודעת שגיאה או אזהרה, אך התוכנית עלולה לקרוס.

דוגמה

בדוגמה זו נתרגל את השימוש בפונקציה strcat. התוכנית שנכתוב תקלוט מהמשתמש מחרוזות בזו אחר זו ותאחדן למחרוזת אחת, תוך הקפדה שלא יועתקו אליה יותר תווים משהיא יכולה להכיל.

```
#include <stdio.h>
#include <string.h>
void main()
{
    char string[41] = "", temp[41];
    int free_chars = 40;

    while (free_chars > 0) {
        do {
            printf("Please enter a string, up to %d chars: ",
                free_chars);
            gets(temp);
        } while (strlen(temp) > free_chars);
        strcat(string, temp);
        free_chars -= strlen(temp);
    }

    printf("Final string: \n%s \n", string);
}
```

פלט לדוגמה מהתוכנית נראה כך:

```
Please enter a string, up to 40 chars: "futile life
Please enter a string, up to 28 chars:  is the s
Please enter a string, up to 19 chars: ame as early
Please enter a string, up to 7 chars:  death"
Final string:
"futile life is the same as early death"
```

```
    char string[41] = "", temp[41];
    int free_chars = 40;
```

את המחרוזות שיזין המשתמש נוסיף למחרוזת string. בכל פעם, תוצב המחרוזת שיזין המשתמש במחרוזת temp, ואם נוכל להוסיפה אל המחרוזת string, נעשה זאת. המשתנה free_chars מציין את מספר התווים הפנויים במחרוזת string. ערכו ישתנה לאחר כל פעם שנוסיף את המחרוזת האחרונה שהוזנה אל המחרוזת string.

```
    while (free_chars > 0) {
        /* ... */
```

לולאה זו אחראית על קליטת המלל מהמשתמש והוספתו למחרוזת string, והיא מתבצעת כל עוד יש בה די איברים פנויים. לאחר שיזין המשתמש 40 תווים, יהיה ערך המשתנה free_chars 0, והלולאה תיפסק.

```
        do {
            printf("Please enter a string, up to %d chars: ",
                free_chars);
            gets(temp);
        } while (strlen(temp) > free_chars);
```

בגוף הלולאה הזאת נקלט מלל מהמשתמש אל המחרוזת temp. אם אורך המלל שנקלט גדול מערך המשתנה free_chars, המייצג את מספר התווים הפנויים במחרוזת string, אין להוסיף אותו למחרוזת string. במקום זאת, יש לקלוט מהמשתמש מלל קצר יותר. הדבר מתבצע על ידי ביטוי הלולאה, המתקיים רק אם ניתן להוסיף את המחרוזת temp למחרוזת string.

```
        strcat(string, temp);
```

פונקציית strcat זו מוסיפה את המלל האחרון שהזין המשתמש אל המחרוזת string. שים לב כי היא תתבצע רק אם במחרוזת string די תווים פנויים כדי להכיל מלל זה.

```
        free_chars -= strlen(temp);
```

משפט השמה זה מפחית מערך המשתנה free_chars את מספר התווים שהזין המשתמש בפעם האחרונה. משתנה זה מאותחל בשורת הגדרתו ל-40, ולכן, בכל פעם הוא יכיל את מספר התווים הפנויים במחרוזת string.

השוואת מחרוזות

פעולה נוספת שברצוננו לעשות היא השוואת מחרוזות מבחינה אלפביתית, כלומר, למצוא, כאשר יש שתי מחרוזות, מי מהן קודמת לשנייה בסדר האלפביתי.

פעולה זו נראית משנית למדי, אך נראה בהמשך שיש לה שימושים רבים. למעשה, אנו מבצעים אותה פעמים רבות גם בחיי היומיום. נניח שברצוננו לחפש במילון את הפירוש למילה "compiler", ופתחנו אותו במילה "computer". כדי להחליט האם לחפש בהמשך המילון או לחזור אחורנית, נשווה כל זוג אותיות מתאימות בשתי המילים, עד שנגיע לזוג אותיות השונות זו מזו. במקרה זה, ארבע האותיות הראשונות בשתי המילים זהות, אך האות החמישית שונה, ומאחר שהאות 'u' באה לאחר האות 'i', נדע כי עלינו לחזור אחורנית.

פונקציה המבצעת פעולה דומה נקראת strcmp. פונקציה זו מקבלת כפרמטרים שתי מחרוזות ומחזירה ערך שלם, הקובע מי מהן קודמת לשנייה בסדר האלפבית. לעת עתה, נחלק את הערך המוחזר ממנה לשלושה תחומים. ערך שלילי מציין שהמחרוזת הראשונה נמצאת, בסדר אלפביתי, לפני השנייה. למשל, הקריאה הבאה לפונקציה:

```
order = strcmp("appendix", "liver");     /* result:  < 0 */
```

מציבה במשתנה order ערך קטן מ-0. ערך חיובי מציין שהמחרוזת הראשונה נמצאת, בסדר אלפביתי, לאחר השנייה.

למשל, הקריאה הבאה לפונקציה:

```
order = strcmp("vein", "vain");      /* result:  > 0 */
```

מציבה במשתנה order ערך גדול מ-0. ואם המחרוזת הראשונה זהה למחרוזת השנייה, כמו בקריאה הבאה לפונקציה:

```
order = strcmp("lung", "lung");      /* result:  = 0 */
```

יוצב במשתנה order הערך 0.

נהוג להשתמש בפונקציה זו במשפטי תנאי. למשל, אם ברצוננו לקלוט שתי מחרוזות מהמשתמש ולפלוט אותן על-פי סדר האלפבית, נכתוב קטע תוכנית כזה:

```
char str1[20], str2[20];
printf("Please enter two strings: \n");
gets(str1);
gets(str2);
printf("The strings in lexicographic relation: \n");
if (strcmp(str1, str2) > 0)
     /* str1 appears after str2 in the dictionary */
     printf("%s \n%s \n", str2, str1);
else
     /* str1 appears before str2 in the
      * dictionary, or they are equal  */
     printf("%s \n%s \n", str1, str2);
```

הנה מספר פלטים המתקבלים מהרצת שורות קוד אלו:

```
Please enter two strings:
quarter
quadrant
The strings in lexicographic relation:
quadrant
quarter
```

```
Please enter two strings:
I believe I can touch the sky
I believe I can fly
The strings in lexicographic relation:
I believe I can fly
I believe I can touch the sky
```

איך לא...

אין להשוות ישירות בין שתי מחרוזות. אילו כתבנו את משפט התנאי כך:

```
if (str1 > str2)      /* legal, but confusing */
    /* ... */
```

לא היתה נפלטת הודעת שגיאה, אך התוכנית לא היתה פועלת כהלכה. באופן דומה, משפט התנאי:

```
if (str1 == str2)
    /* ... */
```

לא בודק האם המחרוזות str1 ו-str2 שוות זו לזו.

דוגמה

בדוגמה זו נתרגל את השימוש בפונקציה strcmp. התוכנית שנכתוב תקלוט מהמשתמש שלוש מחרוזות, ותפלוט אותן בסדר אלפביתי עולה.

תוכנית 10.6 *STRCMP.C*

```
#include <stdio.h>
#include <string.h>
void main()
{
    char str1[20], str2[20], str3[20], temp[20];

    printf("Please enter three strings: \n");
    gets(str1);
    gets(str2);
    gets(str3);

    if (strcmp(str2, str3) > 0) {    /* swap str2 and str3 */
        strcpy(temp, str2);
        strcpy(str2, str3);
        strcpy(str3, temp);
    }

    if (strcmp(str1, str2) > 0) {    /* swap str1 and str2 */
        strcpy(temp, str1);
```

```
        strcpy(str1, str2);
        strcpy(str2, temp);
    }

    if (strcmp(str2, str3) > 0) {    /* swap str2 and str3 */
        strcpy(temp, str2);
        strcpy(str2, str3);
        strcpy(str3, temp);
    }

    printf("\nThe strings in lexicographic relation: \n%s\n%s\n%s\n",
        str1, str2, str3);
}
```

פלט לדוגמה מהתוכנית מוצג להלן:

```
Please enter three strings:
loving america
living in america
leaving america

The strings in lexicographic relation:
leaving america
living in america
loving america
```

שים לב!

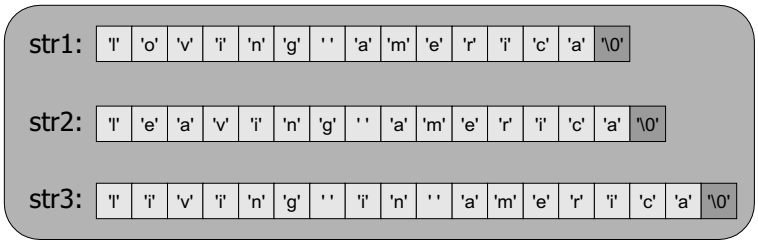
הקלט מורכב אך ורק מאותיות קטנות. לעת עתה, אל תשלח אל strcmp מחרוזות המכילות עירוב של אותיות קטנות וגדולות. נסביר בהמשך מדוע.

```
    char str1[20], str2[20], str3[20], temp[20];
```

בשורה זו מוגדרות ארבע המחרוזות בהן נשתמש בתוכניתנו. במחרוזות str1, str2 ו-str3 יוצבו שלוש המחרוזות שיזין המשתמש. במהלך התוכנית, נמיין את המחרוזות על-פי סדר האלפבית, כך שהראשונה תאוחסן במחרוזת str1, השנייה תאוחסן במחרוזת str2, והאחרונה – במחרוזת str3. אם נראה שיש צורך להחליף בין שתי מחרוזות, נשתמש במחרוזת temp כמחרוזת עזר.

```
    if (strcmp(str2, str3) > 0) {    /* swap str2 and str3 */
        strcpy(temp, str2);
        strcpy(str2, str3);
        strcpy(str3, temp);
    }
```

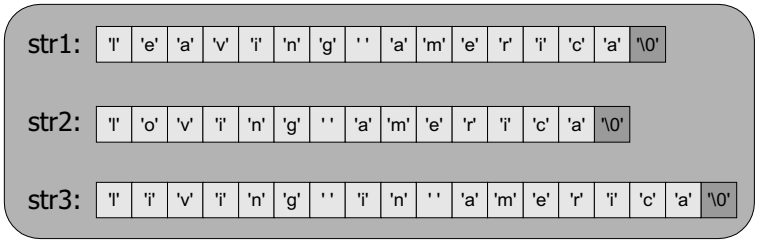
משפט תנאי זה בודק האם המחרוזת המאוחסנת ב-str2 נמצאת, בסדר אלפביתי, אחרי המחרוזת המאוחסנת ב-str3. אם ביטוי התנאי מתקיים, ברצוננו להחליף בין שתי המחרוזות, והדבר נעשה על ידי שלוש הקריאות לפונקציה strcpy. המחרוזת המאוחסנת ב-str2 היא "living in america", והמחרוזת המאוחסנת ב-str3 היא "leaving america". ביטוי התנאי מתקיים, ולכן שתי המחרוזות מוחלפות. לאחר שורות הקוד האלו, ייראו איבריהן כך:



איור 10.17–א' *– איברי שלוש המחרוזות לאחר ההחלפה בין str2 ו-str3*

```
    if (strcmp(str1, str2) > 0) {    /* swap str1 and str2 */
        strcpy(temp, str1);
        strcpy(str1, str2);
        strcpy(str2, temp);
    }
```

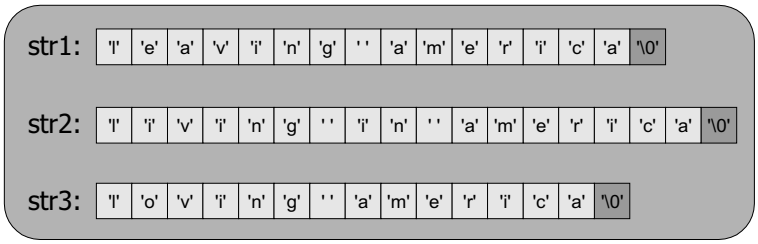
משפט תנאי זה בודק האם המחרוזת המאוחסנת ב-str1 נמצאת, בסדר אלפביתי, אחרי המחרוזת המאוחסנת ב-str2. אם ביטוי התנאי מתקיים, ברצוננו להחליף בין שתי המחרוזות, והדבר נעשה על ידי שלוש הקריאות לפונקציה strcpy. המחרוזת המאוחסנת ב-str1 היא "loving america", ולאחר ההחלפה הקודמת, המחרוזת המאוחסנת ב-str2 היא "leaving america". ביטוי התנאי מתקיים, ולכן שתי המחרוזות מוחלפות. לאחר שורות הקוד האלו, ייראו איבריהן כך:



איור 10.17–ב' *– איברי שלוש המחרוזות לאחר ההחלפה בין str1 ו-str2*

```
    if (strcmp(str2, str3) > 0) {    /* swap str2 and str3 */
        strcpy(temp, str2);
        strcpy(str2, str3);
        strcpy(str3, temp);
    }
```

משפט תנאי זה בודק האם המחרוזת המאוחסנת ב-str2 נמצאת, בסדר אלפביתי, אחרי המחרוזת המאוחסנת ב-str3. אם ביטוי התנאי מתקיים, ברצוננו להחליף בין שתי המחרוזות, והדבר נעשה על ידי שלוש הקריאות לפונקציה strcpy. לאחר שתי ההחלפות הקודמות, המחרוזת המאוחסנת ב-str2 היא "loving america", והמחרוזת המאוחסנת ב-str3 היא "living in america". ביטוי התנאי מתקיים, ולכן מוחלפות שתי המחרוזות. לאחר שורות הקוד האלו, ייראו איבריהן כך:



איור 10.17ג' *– איברי שלוש המחרוזות לאחר ההחלפה השנייה בין המחרוזות str2 ו-str3*

```
    printf("\nThe strings in lexicographic relation:
        \n%s\n%s\n%s\n", str1, str2, str3);
```

לאחר ביצוע שלושת משפטי התנאי, מכילות המחרוזות str1, str2 ו-str3 את המחרוזות שהוזנו, על-פי סדרן האלפביתי. עתה, כל שנותר הוא לפלוט אותן על המסך.

עד כה התייחסנו אל הערך המוחזר מ-strcmp כחיובי, כשלילי או כאפס. עתה נראה מהו הערך המדויק המוחזר מפונקציה זו. נפתח במספר דוגמאות. אם נקרא ל-strcmp כך:

```
strcmp("cannon", "avocado")
```

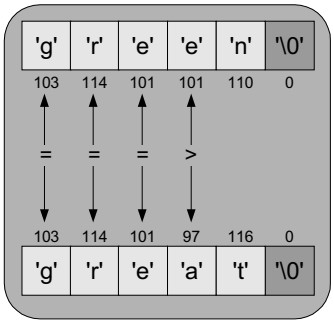
יוחזר הערך 2. ואם נקרא לה כך:

```
strcmp("jerk", "mosquito")
```

יוחזר הערך 3-. בדיקה קצרה תגלה שהערך המוחזר הוא ההפרש בין **הזוג הראשון של התווים שאינם זהים** בשתי המחרוזות. כדי לחשב הפרש זה, יש לבדוק את ערכי ASCII של כל תו ולחסר בין השניים. למשל, בדוגמה הראשונה, ההפרש בין התווים 'c' ו-'a' הוא 97 - 99, כלומר, 2; בדוגמה השנייה, ההפרש בין התווים 'j' ו-'m' הוא 109 - 106, כלומר, 3-. באופן דומה, אם נקרא ל-strcmp כך:

```
strcmp("green", "great")
```

יוחזר ההפרש בין זוג התווים הראשון שאינם זהים – 'e' ו-'a'. הפרש זה הוא 97 - 101, כלומר, 4. תהליך החישוב עבור שתי מחרוזות אלו מוצג באיור הבא:



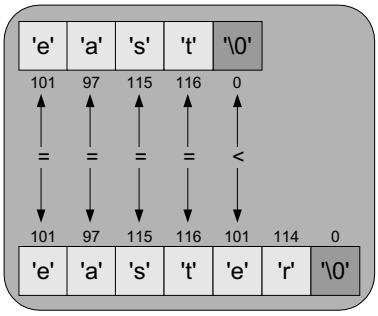
***איור 10.18** – חישוב הפרשם של תווים מתאימים על ידי הפונקציה strcmp*

חישוב הפרשי התווים נעשה על-פי טבלת ASCII, והדבר עלול לגרום למספר בלבולים. למשל, אם נקרא לפונקציה strcmp כך:

```
strcmp("Banana", "avocado")
```

נצפה שיוחזר ערך חיובי, שהרי האות 'B' נמצאת לאחר האות 'a' בסדר האלפביתי. בדיקה קצרה תגלה שהערך המוחזר הינו שלילי. הסיבה היא, שבטבלת ASCII האותיות הלועזיות הגדולות נמצאות לפני האותיות הלועזיות הקטנות, ולכן ערכה המספרי של האות 'B' קטן מזה של האות 'a'. לפיכך, כאשר חיסרנו בין שני הערכים, התקבל ערך שלילי.

באיור 10.19 מושוות שתי מחרוזות השונות באורכן, כאשר תווי המחרוזת הקצרה זהים לתווים הראשונים של המחרוזת הארוכה. זוג התווים הראשון שאינו זהה הוא זוג התווים החמישי, '\0' ו-'e'. הפרשם של תווים אלה הוא שלילי, ולכן, מבחינה לקסיקוגרפית, המחרוזת "east" נמצאת לפני המחרוזת "easter".



***איור 10.19** – חישוב הפרש תווים מתאימים על ידי הפונקציה strcmp*

מערכי מחרוזות

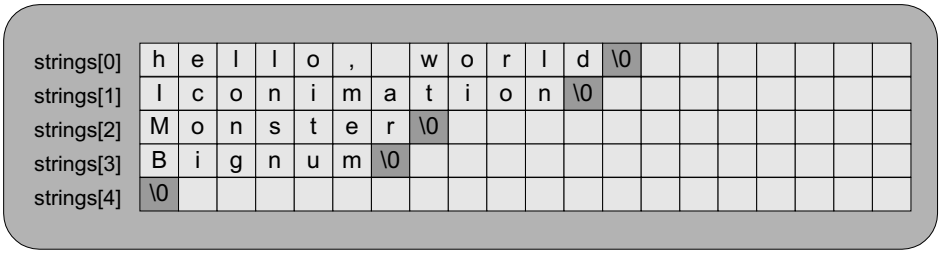
כשיש לטפל במספר רב של מחרוזות נוח להשתמש במערך של מחרוזות. כפי שמחרוזת היא מערך תווי חד-מימדי, מערך של מחרוזות הוא מערך תווי דו-מימדי. בחן, למשל, מערך המוגדר כך:

```
char strings[5][20];
```

זהו מערך תווי דו-מימדי, בעל 5 שורות ו-20 עמודות. כל אחת משורותיו היא מערך תווי חד-מימדי, ולכן ניתן להתייחס אליה כאל מחרוזת, ולבצע הוראות כגון אלו:

```
strcpy(strings[0], "hello, world");
strcpy(strings[1], "Iconimation");
strcpy(strings[2], "Monster");
strcpy(strings[3], "Bignum");
strcpy(strings[4], "");
```

האיור הבא מציג את איברי המערך הדו-מימדי strings לאחר ביצוע הוראות אלו:



***איור 10.20** – איברי המערך המחרוזתי strings*

דוגמה

בדוגמה זו נכתוב תוכנית הקולטת חמש מחרוזות מהמשתמש ופולטת אותן בסדר הפוך לסדר בו נקלטו.

תוכנית 10.7 *STRINGS.C*

```
#include <stdio.h>
void main()
{
    char strings[5][20];
    int i;
```

```
    printf("Please enter five strings: \n");
    for (i = 0 ; i < 5 ; i++)
        gets(strings[i]);

    printf("\nThe strings in reversed order: \n");
    for (i = 4 ; i >= 0 ; i--)
        printf("%s \n", strings[i]);
    printf("\n");
}
```

הנה פלט לדוגמה המתקבל מהרצת התוכנית:

```
Please enter five strings:
chasing dreams
all across
the country
hoping
for better days

The strings in reversed order:
for better days
hoping
the country
all across
chasing dreams
```

```
    char strings[5][20];
    int i;
```

בשורות אלו מוגדרים המשתנים בהם נשתמש בתוכנית. המערך התווי הדו-מימדי strings מכיל 5 מחרוזות בנות 20 תווים. המשתנה i הוא משתנה מונה פשוט.

```
    for (i = 0 ; i < 5 ; i++)
        gets(strings[i]);
```

לולאה זו קולטת חמש מחרוזות ומאחסנת כל אחת מהן בשורה נפרדת במערך המחרוזתי strings. המחרוזת הראשונה שתוזן תוצב בשורה הראשונה במערך strings, המחרוזת השנייה – בשורה השנייה, וכן הלאה.

```
    for (i = 4 ; i >= 0 ; i--)
        printf("%s \n", strings[i]);
```

לולאה זו פולטת את המחרוזות המאוחסנות במערך הדו-מימדי strings. תחילה נפלטת המחרוזת בעלת האינדקס 4, אחר-כך המחרוזת בעלת האינדקס 3, וכן הלאה – עד למחרוזת בעלת האינדקס 0. באופן זה נפלטות המחרוזות בסדר הפוך לסדר בו הוזנו.

שליחת מחרוזות לפונקציות

ניתן בקלות לשלוח מחרוזות לפונקציות. בסעיף זה נראה כיצד לשלוח לפונקציות מחרוזות ומערכי מחרוזות.

שליחת מחרוזת

מחרוזת היא למעשה מערך תווי חד-מימדי, ולכן ניתן לשולחה לפונקציה באותו אופן בו ניתן לשלוח מערך חד-מימדי רגיל. הנה, למשל, הגדרה של פונקציה המקבלת מחרוזת ופרמטר שלם n ופולטת את המחרוזת n פעמים:

```
void PrintString(char str[], int n)
{
    int i;

    for (i = 0 ; i < n ; i++)
        printf("%s \n", str);
}
```

אם נקרא לפונקציה באופן הבא:

```
PrintString("aloha!", 4);
```

ייפלט המלל הבא:

```
aloha!
aloha!
aloha!
aloha!
```

ניתן, כמובן, לשלוח לפונקציה מחרוזת רגילה ולא קבוע מחרוזתי.

שליחת מערך מחרוזות

מערך של מחרוזות הוא למעשה מערך תווי דו-מימדי, ולכן ניתן לשולחו לפונקציה באותו אופן בו ניתן לשלוח מערך דו-מימדי רגיל. הנה, למשל, הגדרה של פונקציה המקבלת מערך מחרוזות ופרמטר שלם n ופולטת את המחרוזות שבו ואת אורכיהן:

```
void PrintStrings(char strings[][20], int n)
{
    int i;

    for (i = 0 ; i < n ; i++)
        printf("%s (len: %d) \n", strings[i],
strlen(strings[i]));
}
```

שים לב!

אל פונקציה זו חובה לשלוח מערך שמספר עמודותיו (כלומר, מספר התווים המרבי שכל אחת ממחרוזותיו יכולה להכיל) הוא 20.

אם נקרא לפונקציה באופן הבא:

```
char strings[4][20];
strcpy(strings[0], "hello!");
strcpy(strings[1], "how do");
strcpy(strings[2], "you");
strcpy(strings[3], "feel?");
PrintStrings(strings, 4);
```

יתקבל הפלט הבא:

```
hello! (len: 6)
how do (len: 6)
you (len: 3)
feel? (len: 5)
```

סיכום

- מלל שנתחם בגרשיים מכונה **קבוע מחרוזתי**.
- כל קבוע מחרוזתי מסתיים בתו '\0', שאינו נראה לעין בעת כתיבת הקבוע המחרוזתי או בעת פליטתו.
- מחרוזת היא מערך תווי חד-מימדי.
- כאשר מציבים תווים במחרוזת אחד-אחד, יש לזכור להציב את התו '\0'.
- ניתן לאתחל מחרוזת על ידי כתיבת תווים בנפרד או על ידי הצבה ישירה של קבוע מחרוזתי.
- פליטת מחרוזות תיעשה על ידי הפונקציה printf.
- קליטת מחרוזות תיעשה על ידי הפונקציות scanf ו-gets. עדיף להשתמש ב-gets כאשר הקלט עלול להכיל רווחים.
- הפונקציה strlen מחזירה את מספר התווים עד לתו '\0' במחרוזת הנשלחת אליה.
- הפונקציה strcpy מעתיקה את המחרוזת השנייה הנשלחת אליה אל המחרוזת הראשונה.
- הפונקציה strcat מעתיקה את המחרוזת השנייה הנשלחת אליה אל תחילתה של המחרוזת הראשונה.
- הפונקציה strcmp מחזירה את ההפרש בין זוג התווים השונה הראשון בין שתי המחרוזות הנשלחות אליה.

נספח א'

על שפת C ושפות אחרות

בנספח זה נעשיר את הידע הכללי שלנו בנוגע לשפות התכנות הקיימות כיום, תוך שימת דגש על שפת C. נסווג את שפות התכנות למספר סוגים, נסקור את התפתחות שפת C ואת יתרונותיה וחסרונותיה ונבחן בקצרה את עולם התכנות המונחה עצמים ב-C++.

סוגי שפות תכנות

ספרנו מתמקד בשפת C, אך קיימות כיום שפות תכנות רבות מסוגים שונים. כמו לשפות מדוברות, לכל שפת תכנות תחביר ו"אוצר מילים" משלה. רוב שפות התכנות פותחו לשם מטרה מסוימת, ולהן יתרונות בתחומים להם יועדו – וחסרונות בתחומים אחרים.

חשוב להכיר את הקשר בין שפות התכנות השונות. לא נוכל להעמיק בכל אחת מהן, שכן הדבר יצריך כתיבת ספרים רבים. במקום זאת, נסתפק בסיווג שפות התכנות השונות לסוגים הבאים:

שפת מכונה

שפת המכונה היא השפה בה כתוב כל **קובץ הרצה** (executable file). קובץ הרצה הוא קובץ המופעל ישירות על ידי מערכת ההפעלה, להבדיל מקבצים הפועלים תחת תוכנות ספציפיות (למשל, קבצי מלל של מעבדי תמלילים או קבצי תמונות ממוחשבות של תוכנות גרפיות).

קובץ ההרצה מכיל הוראות למחשב, המיוצגות על ידי רצף ארוך של ספרות 0 ו-1. רק רצף כזה – ולא פקודות מילוליות כלשהן – יכול המחשב להבין באמת. השפה המורכבת מהספרות 0 ו-1 נקראת **שפת מכונה** (machine language). רצף הספרות 0 ו-1 נשלח אל ה**מיקרו-מעבד** (microprocessor). המיקרו-מעבד הוא מעגל חשמלי מרכזי במחשב

האחראי על ביצוע פעולות כגון פליטת נתונים למסך ולמדפסת, קליטת נתונים מהמקלדת וקריאה וכתיבה לזיכרון. המיקרו-מעבד מתייחס אל הספרה "1" כאל אות אלקטרוני מופעל ואל הספרה "0" כאל אות אלקטרוני מכובה, ודואג לבצע את הפעולות המוגדרות על-פי ספרות אלו.

בעבר נאלצו מתכנתים לפתח תוכניות שלמות באמצעות רצפי מספרים כאלו, תוך פנייה ישירה לרכיבי מחשב כגון תאי זיכרון והתקני חומרה. שער בנפשך כי הצגת משפט תמים על המסך דרשה כתיבה של מאות ספרות כאלו!

שפת סף

כתיבת תוכנית בשפת המכונה היתה מייגעת ומפרכת, ולכן הומצאה **שפת הסף** (low-level language) הקרויה גם **אסמבלי** (assembly). שפת הסף החליפה כל הוראה בשפת המכונה, שיוצגה על ידי רצף ארוך של ספרות 0 ו-1, בפקודה מילולית קצרה (כדוגמת ADD, MOV ו-LOOP). כל פקודה בשפת הסף מקבילה לפקודה בשפת המכונה, אך קל יותר להבינה ולזכרה.

גם כשפותחה שפת הסף יכול היה המחשב להבין רק צירופי 0 ו-1, ולכן עלה צורך בתרגום התוכנית שנכתבה בשפת הסף לרצף ספרות בשפת המכונה. מטלת תרגום זו מבצעת תוכנה הקרויה **אסמבלר** (assembler).

שים לב!

כיום מפותחות תוכנות מעטות בלבד בשפת סף. עמן ניתן למנות מערכות הפעלה (כגון DOS), מהדרים (כגון Turbo Pascal) ותוכנות נוספות שמהירות ריצתן חשובה לאין ערוך (כגון תוכנות לחימוש טילים גרעיניים).

שפה עילית

מאוחר יותר פותחו **שפות עיליות** (high-level language), המאפשרות תכנות נוח ופשוט יותר מאשר בשפת הסף. תוכניות הכתובות בשפה עילית קלות יותר לכתיבה, לקריאה ולשינוי; שלא כמו פקודות בשפת הסף, כל פקודה בשפה העילית משלימה מטלה הגיונית אחת שלמה, ולא פעולת מיקרו-מעבד אחת. לדוגמה, אם לשם פליטת מילה על המסך יש לכתוב שש פקודות בשפת סף (ומאות ספרות בשפת מכונה), בשפה עילית הדבר ייעשה בפקודה בודדת.

קיימות שפות עיליות רבות המתאימות לתפקידים שונים: לחישובים מדעיים, לבינה מלאכותית, לגרפיקה ועוד. מהמיגוון הרחב יכול המתכנת לבחור לעצמו את השפה המתאימה ביותר לצרכיו.

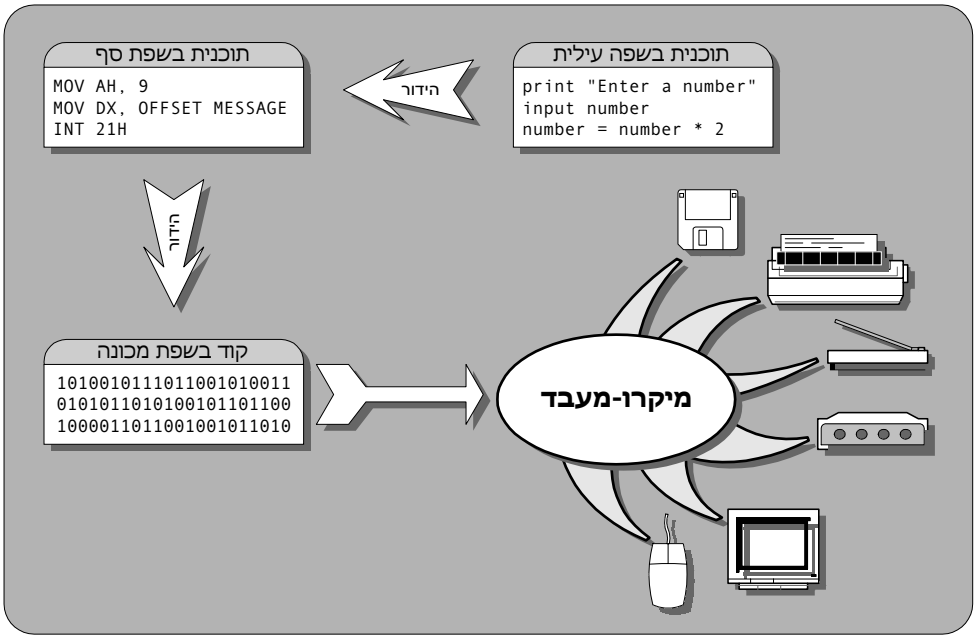
גם תוכנית שנכתבה בשפה עילית יש לתרגם לשפת מכונה, וזאת עושות תוכנות תרגום הנקראות **מהדר** (compiler) או **מתרגם** (interpreter). איור א.1 מציג את השלבים בהם מועברות הוראותיה של תוכנית שנכתבה בשפה עילית אל המיקרו-מעבד.

במחשבה שנייה...

מדוע לא לכתוב את כל התוכניות בשפות עיליות?

תוכנית הכתובה בשפה עילית תרוץ לאט יותר מתוכנית הכתובה בשפת סף או בשפת מכונה. זאת משום שהתוכנה המתרגמת תוכנית בשפה עילית לשפת מכונה אינה מושלמת, ואינה יכולה להשיג ביעילותה מתכנת מנוסה המתכנת ישירות בשפת סף. אך חסרון המהירות בדרך כלל אינו מורגש, ולכן רוב התוכנות נכתבות כיום בשפות עיליות.

כמו-כן, השפה העילית מוגבלת ביכולתה. באמצעותה לא ניתן, לדוגמה, לפנות ישירות אל תאי זיכרון או אל התקני חומרה במחשב. גם מגבלה זו אינה כה משמעותית, שכן בדרך כלל אין צורך בפנייה אל "מחשכי" המחשב.



איור א.1 *– תהליך העברת הוראות בשפה עילית למיקרו-המעבד*

הקשר של שפת C לשפות התכנות האחרות

שפת C אינה שפת מכונה או שפת סף, ולכן אין מנוס מלשייכה לקטגוריית השפות העיליות. אף שתכונות רבות של שפת C דומות לאלו של שפות עיליות אחרות (למשל, טיפוסי משתנים רבים, משפטי בקרה נוחים ותחביר מובן), אין לשכוח את עוצמתה, מהירותה ומימדיה המצומצמים של תוכנית C טיפוסית. לפיכך, נעדיף למקם את שפת C בין שפת הסף לשפות העיליות, ולכנותה שפת-ביניים (middle-level language).

התפתחות שפת C

פיתוחה של שפת C לא היה מיידי. השפה פותחה בשלבים רבים, תוך הפקת לקחים משפות תכנות שקדמו לה. למעשה, אפילו שפת C שהוצגה לראווה בשנת 1972 לא זהה לגמרי לשפת C בה מתכנתים כיום. בסעיף זה נעקוב אחר פיתוחן של שפות תכנות שהשפיעו על שפת C ואחר פיתוחה של שפת C מאז הוצגה לראשונה.

מ-Algol68 ל-B

ראשיתה של שפת C נעוצה בשפת התכנות Algol68, שפותחה בתחילת שנות ה-60 באירופה. היא הופיעה מספר שנים לאחר שפת Fortran, והיתה מעט יותר מתוחכמת ממנה. מתכנניה השקיעו מאמץ רב בשיפור תחבירה ובעשייתה מודולרית ככל שאפשר, אך חרף המאמצים, Algol68 לא היתה פופולרית בארה״ב.

בשנת 1963 פותחה באוניברסיטאות לונדון וקיימברידג׳ שפת CPL (Compiled Programming Language), במטרה לשפר את אופייה הפשטני של Algol68, שניסה לתת מענה לתחומים ממוחשבים רבים מדי. שפת CPL היתה מגושמת למדי ותחבירה נשאר מסובך כשל Algol68, ולכן גם היא לא היתה פופולרית במיוחד.

בשנת 1967 תכנן מרטין ריצ׳רדס מאוניברסיטת קיימברידג׳ את שפת BCPL (Basic Compiled Programming Language). שפת BCPL תוכננה במטרה לשפר את שפת CPL, ואף שבתחילת שנות ה-70 פותחו באמצעותה מספר פרויקטים חשובים, היא לא היוותה פריצת דרך מכרעת.

בשנת 1969 נכתבה הגרסה הראשונה של מערכת ההפעלה UNIX במעבדות בל שבניו-ג׳רסי, בשפת-סף של מחשב DEC PDP-7. UNIX נחשבה לסביבה ידידותית ומקצועית, ובמהרה פותחה עבורה שפה עילית, שנועדה לפיתוח שפות תכנות חדשות. קן תומפסון, מתכנת במעבדות בל, התלהב מפיתוחה והמציא ב-1970 שפה עילית משלו למערך UNIX. שפה זו נקראה B – על שמה של שפת BCPL או על שמה של שפת BON (שנוצרה על ידי תומפסון מוקדם יותר).

מ-B ל-C

לאחר שיישם תומפסון את שפת B על מחשב ה-DEC PDP-7 שברשותו, הגיע לידיו מחשב מדגם חדש יותר, DEC PDP-11. עד לאותו זמן שימשה שפת B לפיתוח יישומי UNIX רבים, אך הארכיטקטורה המתקדמת של מחשב ה-DEC PDP-11 התנגשה באופייה המיושן של השפה. למשל, האופן בו עיבדה שפת B נתוני מלל היה מגושם למדי בהתחשב במחשב החדש. זאת ועוד – מכיון שמהדר השפה בו השתמשו במחשבי ה-DEC PDP-7 לא יצר הוראות בשפת מכונה, תוכניות שהודרו בשפת B היו איטיות בהרבה מתוכניות שפותחו בשפת-סף. גורמים אלו העלו את הצורך בפיתוח שפת תכנות חדשה ומשופרת.

משימה זו לקח על עצמו דניס ריצ'י, מתכנת ותיק במעבדות בל. בשנת 1971 הוא הרחיב את שפת B בכך שהוסיף לה סוגי נתונים חדשים, שיפר את האופן בו טיפלה בנתוני מלל ושכתב את המהדר, כך שיחולל הוראות המתאימות למחשב החדש. הרחבה זו כונתה על ידי ריצ'י New B, והיא דמתה למדי לשפת C המוכרת לנו כיום. היות והיא היתה שפת-מעבר בין B ל-C ותו-לא, לא נכתבו עבורה מסמכי תיעוד מלאים. לאחר מספר התנסויות ושיפורים, החליט ריצ'י כי ראוי להעניק לה שם חדש: C.

בשנת 1978 יצא לאור הספר הראשון אודות שפת C, שנכתב על ידי דניס ריצ'י ורעהו בריאן קרניגהן. רבים מתיחסים אליו כאל "התנ"ך של שפת C" ומאמינים שהוא הספר הטוב ביותר שפורסם אודותיה עד היום. ואכן, הספר שימש במשך למעלה מעשר שנים כמסמך התיעוד המרכזי לשפה. למעשה, תקן C הישן, על-פיו נכתבו מאות אלפי שורות קוד, הסתמך כה רבות על ספרם של ריצ'י וקרניגהן, עד שכונה **תקן K&R** (Kernighan & Ritchie).

תקן ANSI C

בשנות ה-80 המוקדמות נהייתה C לשפה פופולרית ביותר, והיא נפוצה במהירות אל מחוץ למעבדות בל. מתכנתים החלו לתכנת בה תחת מערכות הפעלה שונות, וכאשר גופים עצמאיים פיתחו מהדרי C לפי צרכיהם הפרטיים, נוצרו גרסאות שונות של השפה. חלק מההבדלים בין הגרסאות היו משמעותיים ביותר, וגרעו מאחידותה של השפה. לכן, בשנת 1983 הקים מכון התקנים האמריקני, הקרוי ANSI (American National Standards Institute), ועדה לקביעת תקן אוניברסלי, שתפתור את בעיית חוסר העקביות בשפת C.

לוועדה זו ניתן השם X3J11, ובשנת 1990, לאחר דיונים ארוכים, הוצגו מסקנותיה במסמך ארוך בן מאתיים עמודים. הוועדה הוסיפה לשפה מספר מרכיבים (בעיקר משפת C++), החליטה לגבי מספר סוגיות שהדעות עליהן היו חלוקות וקבעה אילו קבצי ספריה ישויכו לשפה. קבצי ספריה אלו העשירו את השפה ואפשרו לה לתקשר במקצועיות עם מערכת ההפעלה. תקן השפה כונה **תקן ANSI C**, וזמן קצר לאחר שפורסם, אימץ אותו מכון התקנים הבינלאומי, הקרוי ISO (International Standards Organization). שמו הרשמי של התקן הוא **תקן ISO/IEC 9899:1994**, אך בפי כל הוא מכונה **תקן ANSI C**.

בעוד שכל המהדרים המודרניים תומכים בתקן ANSI C, רובם תומכים גם בתקן K&R הישן, כדי לא להפוך תוכניות שנכתבו על-פיו לבלתי-שמישות. בספרנו נתמקד בתקן ANSI C, אך במהלך הלימוד נציין את ההבדלים המשמעותיים בין שני התקנים.

יתרונות השפה וחסרונותיה

לשפת C יתרונות רבים, ובזכותם היא אחת השפות הדומיננטיות ביותר כיום. יתרונותיה המרכזיים מוצגים להלן:

- **קומפקטיות** – הודות למהדר השפה הקומפקטי, תוכנית הכתובה נכונה בשפת C תהיה קטנה מתוכנית הכתובה בשפה עילית אחרת. בניגוד למהדרים של שפות עיליות אחרות, מהדר שפת C מכיל מספר מצומצם ביותר של פקודות. בנוסף, לא בנויות בו פונקציות הבנויות במהדרי שפות עיליות אחרות, ולכן תוכנית C טיפוסית תופסת כמות זיכרון מזערית.

במחשבה שנייה...

כיצד נקבעת קומפקטיות המהדר?

המדד העיקרי לבדיקת קומפקטיות המהדר הוא מספר המילים השמורות אותו הוא מזהה. מילה שמורה היא הוראה בשפה המורה למהדר לבצע פעולה מוגדרת כלשהי. ככל שמספר המילים השמורות רב יותר, כך המהדר מסורבל יותר. בשפת C 32 מילים שמורות, וזהו מספר קטן ביותר לעומת מספר המילים השמורות בשפות תכנות עיליות אחרות (בשפת פסקל 49 מילים שמורות, בשפת C++ 48 מילים שמורות, ובשפת עדה 62 מילים שמורות).

- **מהירות** – הודות למהדר המתקדם של השפה, תוכניות C רצות בקצב מהיר במיוחד. למשל, רצף פקודות החוזר על עצמו בתוכנית לא יתורגם לשפת מכונה בכל פעם מחדש, אלא פעם אחת בלבד. בנוסף, מהדר C כולל אזהרות מיוחדות המסייעות למתכנת בייעול תוכניתו. אזהרות אלו, לדוגמה, מסבות את תשומת-לב המתכנת אל משתנים אשר הגדיר אך לא עשה בהם שימוש, מתריעות מפני רצף פקודות שלעולם לא יתבצע ועוד.
- **עוצמה** – שפת C מאפשרת גישה ישירה לתאי זיכרון, כמעט כמו שפת הסף. בנוסף, מהדר השפה תומך באופן מלא בעבודה בסיביות, והופך אותה לשפה טבעית לפיתוח מערכות הפעלה, מהדרים ויישומים מתקדמים נוספים.
- **חופשיות** – למתכנת בשפת C נתונה יד חופשית בכל הנוגע להמרות משתנים ולפנייה לתאי זיכרון. פעולות רבות שאסורות ברוב שפות התכנות העיליות הינן חוקיות לגמרי בשפת C.

- **שפה מובנית** – למרות החופשיות הרבה הניתנת למתכנת, המזכירה את אופייה המתירני של שפת הסף, שפת C כוללת משפטי בקרה (משפטי תנאי ולולאות) מגוונים, כמו אלו הקיימים בשפות עיליות אחרות. קרבתה הרבה לשפת הסף לא מונעת משפת C תחביר עשיר ומובן, המאפשר למתכנת להתרכז בבעיה הניצבת לפניו ולקרוא את קוד המקור בקלות.
- **קישור** – בשפת C ניתן לחלק תוכנית אחת למספר חלקים שונים ולאחדם לקובץ הרצה אחד כאשר כל חלק מושלם. הדבר מאפשר לחלק תוכנית גדולה למספר מתכנתים, כך שכל מתכנת עובד בו-זמנית על חלק שונה ממנה. באופן זה ניתן להדר כל חלק בנפרד, בלי להדר את כל התוכנית בכל פעם מחדש, ובכך לחסוך זמן יקר.
- **ניידות** – שפת C היא אחת מהשפות הניידות ביותר כיום. מכיון שמהדרי שפת C זהים תחת מערכות הפעלה רבות, כל שדרוש כדי להמיר תוכנית שנכתבה תחת מערכת הפעלה אחת לאחרת הוא להדרה במהדר הרץ תחת מערכת ההפעלה המבוקשת. שפת הסף, למשל, אינה כזו: תוכנית שנכתבה בשפת סף של DOS תצטרך להיכתב מחדש כדי שתוכל לרוץ תחת מערכת ההפעלה UNIX.
- **קרבה לשפות נוספות** – שפת C מהווה את הבסיס לשפת C++ ותחבירה דומה לשל Java. מתכנת הלומד את שפת C יקנה לעצמו כלים רבים הדרושים ללמידת C++ ו-Java, שהן שפות תכנות פופולריות ורבות-עוצמה.

ליתרונות אלו משקל רב ביותר בעולם התכנות. שפת C מהווה שפת-מפתח ברוב תחומי המחשב, אך קיימים מספר תחומים שלא יועדו לה; לדוגמה, תחום הבינה המלאכותית.

לשפת C אין חסרונות רבים, אך ניתן לייחס לחלק מיתרונותיה גם חסרונות:

- ליברליות מהדר השפה, אף שהיא מהווה יתרון, יכולה להתפרש גם כחסרון. מתכנת מתחיל עשוי לכתוב משפטים שגויים שמפניהם לא יזהיר המהדר. משפטים שגויים כאלו עלולים לגרום לקריסת התוכנית ואף לקריסת מערכת ההפעלה.
- יש הטוענים כי תוכנית הכתובה בשפת C קשה יותר לקריאה מתוכניות הכתובות בשפות עיליות אחרות, בשל האופרטורים (סימנים כגון +, -, <, > ו-=) הרבים וסימני הפיסוק הרבים הקיימים בה. על טעם ועל ריח אין להתווכח, אך אין ספק כי מתכנת המורגל בפיתוח תוכניות C לא יראה בתחביר שלה גורם מפריע לקריאת התוכניות. להיפך; בתוכנית ארוכה, המכילה עשרות אלפי מילים, דווקא סימני הפיסוק בולטים ביותר.

שים לב!

מילים שלמות (כגון Begin ו-End בפסקל) מופיעות בשפת C כסימנים (כדוגמת { ו-}) וכאופרטורים. למעשה, מהדר שפת C מכיר יותר אופרטורים ממילים!

- לרוב, משולבים האופרטורים הרבים בשפת C לשם ביצוע פעולות שונות. הודות לצירופים הרבים האפשריים, קל להתבלבל בין אופרטור זה או אחר. במקרים רבים לא יתריע המהדר, בשל מתירנותו הרבה, על כך שהאופרטור שנבחר אינו מתאים למטרה בה חפצנו.

תכנות מונחה עצמים באמצעות C++

תיאוריית ה**תכנות מונחה העצמים** (OOP – Object Oriented Programming) גורסת כי את התוכנית יש לאגד לקבוצות נתונים המכונות **מחלקות** (classes). המחלקה, המהווה את בסיסה של שפת C++, מאגדת בתוכה משתנים ופונקציות בעלי מכנה משותף. באמצעות המחלקה ניתן לבצע פעולות רבות ומגוונות, ובכך להקל על כתיבת תוכניות ארוכות ועל ביצוע שינויים בהן.

C++ פותחה על ידי ביארן סטראוסטרפ ממעבדות בל בתחילת שנות ה-80. סטראוסטרפ שאף לשדרג את שפת C כך שתתמוך בתכנות מונחה העצמים, תוך שמירה על יתרונותיה הרבים ושיפור אחדים מחלקיה. שלא כמו שפות מונחות עצמים אחרות, C++ מאפשרת לכתוב תוכניות שאינן מיישמות במלואן את טכניקת התכנות מונחה העצמים. באופן זה קל למתכנת המתחיל לתרגל את הטכניקה בשלבים, עד שכל התוכנית נכתבת על-פיה.

ניתן לכתוב ספרים רבים על טכניקת התכנות מונחה העצמים בלי להזכיר כלל את שפת C++. ספר זה אינו מתמקד בתכנות מונחה עצמים, ולכן לא נעמיק בנושא יתר על המידה.

ספרים בנושא זה ועוד ראה בקטלוג שבסוף הספר ובתקליטור המצורף.

נספח ב'

הודעות שגיאה נפוצות

בנספח זה מוצגות הודעות שגיאה נפוצות שמציג מהדר שפת C. אם תיתקל במהלך כתיבת תוכניותיך בהודעת שגיאה שמקורה אינה ברור לך, עיין בסעיף זה.

עצה

אם המהדר מודיע כי קימת שגיאה בשורה מסוימת והנך בטוח כי היא נכונה מבחינה תחבירית, בחן את השורה שמעליה. ייתכן כי מקור השגיאה הוא בה.

שורת #include שגויה

Bad file name format in include directive

הודעה זו תוצג אם שם קובץ המצוין בהוראת #include אינו תחום בסוגריים משולשים או בגרשיים.

```
#include stdio.h      ✗ Bad file name format in include directive ✗
void main()
{

}
```

הגדרה שגויה — Body has already been defined for this function

הודעה זו תוצג אם גוף של אותה פונקציה מופיע יותר מפעם אחת, כלומר, אם היא מוגדרת מספר פעמים. יש להגדיר פונקציה פעם אחת בלבד.

```
void f1()
{

}

void f1()
{       ✗ Body has already been defined for this function ✗

}
```

הצבה שגויה — Cannot assign 'XXX' to 'YYY' in function ZZZ

הודעה זו תוצג אם ננסה להציב ערך בלתי חוקי במשתנה מסוים. בדרך כלל יקרה הדבר אם ננסה להציב מחרוזות במשתנים מסוג מספרי (כגון int ו-float).

- XXX – סוג הערך אותו מנסים להציב במשתנה
- YYY – סוג המשתנה בו מנסים להציב את XXX
- ZZZ – הפונקציה בה מתבצעת ההצבה השגויה

```
void main()
{
    int i;
    i = "word";       ✗ Cannot assign 'char *' to 'int' in function main() ✗
}
```

Case outside of switch in function XXX

case במקום אסור

הודעה זו תוצג אם הופיעה שורת case במקום לא מתאים. שורת case יכולה להופיע רק בתוך משפט switch.

❖ XXX – הפונקציה בה מופיעה שורת case השגויה

```
void main()
{
    int i = 3;

    switch (i) {
    case 1:
        i++;
        break;
    case 2:
        i--;
        break;
    }
    case 3:      ✗ Case outside of switch in function main() ✗
        i = 0;
}
```

Case statement missing : in function XXX

חסרות נקודתיים

הודעה זו תוצג אם הופיעה שורת case בלי נקודתיים. לאחר הקבוע השייך לשורת case חייבות להופיע נקודתיים.

❖ XXX – הפונקציה בה נכתבת שורת case השגויה

```
void main()
{
    int n = 1;

    switch (n) {
    case 1      ✗ Case statement missing : in function main() ✗
        n = 2;
        break;
```

```
    case 2:
        n = 1;
        break;
    }
}
```

Character constant too long in function XXX

קבוע תווי שגוי

הודעה זו תוצג אם נכתוב קבוע תווי שמכיל תווים רבים מדי. בין הגרשיים התוחמים קבוע תווי יש לכתוב תו אחד (למשל, 'a') או שניים (למשל, '\n') בלבד.

- XXX – הפונקציה בה מופיע הקבוע התווי השגוי

```
void main()
{
    char ch;
    ch = 'hello';       ✗ Character constant too long in function main() ✗
}
```

Compound statement missing } in function XXX

סוגריים מסולסלים
סוגרים חסרים

הודעה זו תוצג אם הופיעו סוגריים מסולסלים פותחים ללא סוגריים מסולסלים סוגרים. סוגריים אלו צריכים לבוא בזוגות, כלומר, לכל סוגריים מסולסלים פותחים יש להתאים סוגריים מסולסלים סוגרים.

- XXX – הפונקציה בה חסרים הסוגריים המסולסלים הסוגרים

```
void main()
{       ✗ Compound statement missing } in function main() ✗
```

הודעה זו תוצג אם יימצא משתנה במקום בו אמור להופיע ערך קבוע. ערך קבוע הכרחי כאשר קובעים גודל מערך בשורת הגדרתו וכאשר מציינים קבוע של שורת case. לכן, אם n הוא משתנה, אין להגדיר מערך בגודל n איברים ואין להגדיר שורת case שהקבוע שלה הוא n.

שים לב!

הודעה זו עלולה להיפלט גם אם נשתמש בערכים עשרוניים (כגון 3.5 ו-4.0) במקומות שתוארו לעיל.

❖ XXX – הפונקציה בה אמור להופיע הערך הקבוע

```
void main()
{
    int n = 26;
    char array[n];      ✘ Constant expression required in function main() ✘
}
```

```
void main()
{
    int n = 2, m = 2;

    switch (m) {
    case n:      ✘ Constant expression required in function main() ✘
        m++;
        break;
    case 3:
        m--;
        break;
    }
}
```

Declaration is not allowed here — הצהרה אסורה

הודעה זו תוצג אם נגדיר משתנה במקום אסור. את כל המשתנים יש להגדיר מיד לאחר הסוגריים המסולסלים הפותחים בלוק של הוראות, בלי שמופיעות לפניהם הוראות כלשהן (כגון הוראת השמה וקריאה לפונקציה).

```
#include <conio.h>
void main()
{
    char ch;
    ch = getch();
    int i;      ✗ Declaration is not allowed here ✗
}
```

Default outside of switch in function XXX — default במקום אסור

הודעה זו תוצג אם תופיע שורת default במקום לא מתאים. שורת default יכולה להופיע רק בתוך משפט switch.

- XXX – הפונקציה בה מופיעה שורת default השגויה

```
void main()
{
    char l = 'x';

    switch (l) {
    case 'a':
        l += 2;
        break;
    case 'c':
        l -= 2;
        break;
    }
    default:      ✗ default outside of switch in function main() ✗
        l = 0;
}
```

חלוקה אסורה באפס

Division by zero in function XXX

הודעה זו תוצג אם יופיע ביטוי המחשב מנת ערך באפס או שארית של מנת ערך באפס. שתי הפעולות הללו אסורות.

- XXX – הפונקציה בה מופיע הביטוי השגוי

```
void main()
{
    int i;
    i = 4 / 0;       ✘ Division by zero in function main() ✘
}
```

```
void main()
{
    int i;
    i = 4 % 0;       ✘ Division by zero in function main() ✘
}
```

לולאת do-while שגויה

Do statement must have while in function XXX

הודעה זו תוצג אם לולאת do-while אינה מכילה חלק while. הודעה זו עלולה לצוץ אם מופיע תו נקודה-פסיק לאחר הסוגריים המסולסלים הסוגרים את גוף הלולאה.

- XXX – הפונקציה בה מופיעה הלולאה השגויה

```
void main()
{
    do {

    }
}      ✘ Do statement must have while in function main() ✘
```

```
void main()
{
    do {
        /* note the semicolon below */
    };      ✘ Do statement must have while in function main() ✘
    while (1 == 1);
}
```

כפילות של שורת case — Duplicate case in function XXX

הודעה זו תוצג אם הופיעו שתי שורות case או יותר בעלות אותו ערך קבוע באותו משפט switch. במשפט switch הערכים הקבועים של שורות ה-case חייבים להיות שונים זה מזה.

❖ XXX – הפונקציה בה מופיעה שורת case השגויה

```
void main()
{
    int i = 2;

    switch (i) {
    case 0:
        i++;
        break;
    case 1:
        i--;
        break;
    case 0:      ✘ Duplicate case in function main() ✘
        i++;
        break;
    }
}
```

הודעה זו תוצג אם נכתב ביטוי שאינו נכון מבחינה תחבירית. בדרך כלל קורה הדבר כאשר שני ערכים או שני אופרטורים מופיעים אחד אחרי השני, כאשר לא מופיעים סוגריים כנדרש או כאשר המשפט הקודם לביטוי לא הסתיים בנקודה-פסיק.

❖ XXX – הפונקציה בה נכתב הביטוי השגוי

```
void main()
{
    int i;
    i = 4 5;          ✘ Expression syntax in function main() ✘
}
```

```
void main()
{
    int i;
    i = 2 - / 4;      ✘ Expression syntax in function main() ✘
}
```

```
void main()
{
    int i;
    i = 1     /* note: no semicolon */
    i = 2 + 1;        ✘ Expression syntax in function main() ✘
}
```

```
#include <stdio.h>
void main()
{
    printf "Hello";   ✘ Expression syntax in function main() ✘
}
```

קריאה שגויה לפונקציה

Extra parameter in call to XXX in function YYY

הודעה זו תוצג אם בעת הקריאה לפונקציה XXX נשלחו אליה יותר פרמטרים משהיא אמורה לקבל. אל כל פונקציה יש לשלוח מספר פרמטרים כפי שנקבע בהגדרתה.

- XXX – הפונקציה הנקראת עם מספר פרמטרים רב מדי
- YYY – הפונקציה ממנה מתבצעת הקריאה ל-XXX

```
#include <conio.h>
void main()
{
    char ch;
    ch = getch('a');        ✗ Extra parameter in call to getch() in function main() ✗
}
```

משפט return שגוי

Function should return a value in function XXX

הודעה זו תוצג אם יופיע משפט return שאינו מחזיר ערך (; return) מתוך פונקציה שאינה מחזירה void. כאשר פונקציה מחזירה ערך, חובה לציין במשפט ה-return ערך מוחזר.

- XXX – הפונקציה בה הופיע משפט ה-return השגוי

```
int MyFunction()
{
    return ;        ✗ Function should return a value in function MyFunction ✗
}
```

שימוש שגוי בערך עשרוני

Illegal use of floating point in function XXX

הודעה זו תוצג אם נשתמש שלא כשורה בערך עשרוני. הדבר יכול לקרות, למשל, אם ננסה לחשב שארית של מנת מספרים עשרוניים.

- XXX – הפונקציה בה נעשה שימוש שגוי בערך עשרוני

```
void main()
{
    int i;
    i = 4 % 2.5;        ✘ Illegal use of floating point in function main() ✘
}
```

הצבה שגויה

Lvalue required in function XXX

הודעה זו תוצג אם ננסה לשנות ערך של מערך (בלי לציין אינדקס של איבר כלשהו בו) או של קבוע.

- XXX – הפונקציה בה מבוצעת ההשמה השגויה

```
void main()
{
    int array[10];

    array = 5;        ✘ Lvalue required in function main() ✘
```

```
void main()
{
    7 = 10;           ✘ Lvalue required in function main() ✘
}
```

```
void main()
{
    3++;              ✘ Lvalue required in function main() ✘
}
```

Misplaced break in function XXX

break במקום אסור

הודעה זו תוצג אם משפט break הופיע במקום אסור. משפט break יכול להופיע רק בתוך משפט switch או בתוך לולאה.

- XXX – הפונקציה בה נכתב משפט break השגוי

```
void main()
{
    int a = 20, b = 2;
    if (a > 0) {
        if (b == 0)
            break;      ✗ Misplaced break in function main() ✗
        a /= b;
    }
}
```

Misplaced continue in function XXX

continue במקום אסור

הודעה זו תוצג אם משפט continue הופיע במקום אסור. משפט continue יכול להופיע רק בתוך לולאה.

- XXX – הפונקציה בה נכתב משפט ה-continue השגוי

```
void main()
{
    int a = 20, b = 1;
    if (a > 0) {
        b++;
        if (b == 0)
            continue;      ✗ Misplaced continue in function main() ✗
        a /= b;
    }
}
```

else במקום אסור

Misplaced else in function XXX

הודעה זו תוצג אם משפט else נכתב במקום אסור. משפט else חייב להופיע לאחר המשפט השייך למשפט if אליו הוא מתקשר.

- XXX – הפונקציה בה נכתב משפט else השגוי

```
void main()
{
    int x = 1, y = 0;

    if (x == 1)
        x++;
        y++;     /* this statement causes the error */
    else     ✘ Misplaced else in function main() ✘
        x--;
}
```

הצהרה כפולה על משתנה

Multiple declaration for 'XXX' in function YYY

הודעה זו תוצג אם אותו משתנה הוגדר יותר מפעם אחת באותו בלוק של סוגריים מסולסלים. יש להגדיר כל משתנה פעם אחת בלבד בכל בלוק כזה.

- XXX – המשתנה שמוגדר יותר מפעם אחת
- YYY – הפונקציה בה מוגדר XXX

```
void main()
{
    int n;
    float n;      ✘ Multiple declaration for 'n' in function main() ✘
}
```

הצהרת פונקציה שגויה

Parameter names are used only with a function body

הודעה זו תוצג אם הצהרת פונקציה מכילה רק את שמות הפרמטרים ולא את סוגם. בהצהרת פונקציה חובה לציין את סוג הפרמטרים.

```
void MyFunction(i);       ✗ Parameter names are used only with a function body ✗

void MyFunction(int i)
{

}
```

נקודה-פסיק חסר

Statement missing ; in function XXX

הודעה זו תוצג אם נכתבה הוראה שלא הסתיימה בנקודה-פסיק. לאחר כל הוראה (כגון קריאה לפונקציה ופעולת השמה) חובה להוסיף נקודה-פסיק.

- XXX – הפונקציה בה מופיעה ההוראה השגויה

```
void main()
{
    int x;
    x = 4
}           ✗ Statement missing ; in function main() ✗
```

```
#include <stdio.h>
void main()
{
    printf("%d", 1)
}           ✗ Statement missing ; in function main() ✗
```

Subscripting missing] in function XXX

סוגריים מרובעים
סוגרים חסרים

הודעה זו תוצג אם פנינו אל איבר במערך על-ידי ציון האינדקס שלו בין סוגריים מרובעים פותחים בלבד. כל אינדקס חייב להופיע בין סוגריים מרובעים פותחים וסוגרים.

- XXX – הפונקציה בה מופיעה הפנייה השגויה

```
void main()
{
    int array[10];
    array[0 = 4;      ✗ Subscripting missing ] in function main() ✗
}
```

Switch selection expression must be of integral type in function XXX

ביטוי switch
שגוי

הודעה זו תוצג אם הביטוי הנבדק בין הסוגריים של משפט switch מחזיר ערך עשרוני. ביטוי זה חייב להחזיר ערך שלם.

- XXX – הפונקציה בה מופיע משפט switch השגוי

```
void main()
{
    float f = 4.5;

    switch (f) {      ✗ Switch selection expression must be of
                        integral type in function main() ✗
    case 1:
        f++;
        break;
    case 2:
        f--;
        break;
    }
}
```

קריאה שגויה לפונקציה

Too few parameters in call to XXX in function YYY

הודעה זו תוצג אם בעת הקריאה לפונקציה XXX נשלחו אליה פחות פרמטרים משהיא אמורה לקבל. אל כל פונקציה יש לשלוח מספר פרמטרים כפי שנקבע בהגדרתה.

- XXX – הפונקציה הנקראת עם מספר פרמטרים מועט מדי
- YYY – הפונקציה ממנה מתבצעת הקריאה ל-XXX

```
#include <stdio.h>
void main()
{
    printf();       ✘ Too few parameters in call to printf() in function main() ✘
}
```

שורות default רבות מדי

Too many default cases in function XXX

הודעה זו תוצג אם משפט switch מכיל שורות default רבות מדי. משפט switch יכול להכיל שורת default אחת לכל היותר.

- XXX – הפונקציה בה מופיע משפט ה-switch השגוי

```
void main()
{
    int i;
    switch (i) {
    case 1:
        i++;
        break;
    default:
        i += 2;
    default:      ✘ Too many default cases in function main() ✘
        i = 0;
    }
}
```

קריאה שגויה לפונקציה

Type mismatch in parameter 'XXX' in call to 'YYY' in function ZZZ

הודעה זו תוצג אם בעת הקריאה לפונקציה XXX נשלחו אליה ערכים בלתי מתאימים. סוגי הערכים הנשלחים אל פונקציה חייבים להתאים אל סוגי הפרמטרים שלה.

- XXX – הפרמטר שסוגו לא מתאים לערך שנשלח
- YYY – הפונקציה שנקראת באופן שגוי
- ZZZ – הפונקציה בה מתבצעת הקריאה ל-YYY

```
#include <stdio.h>
void main()
{
    printf(100);       ✘ Type mismatch in parameter '   format'
                          in call to 'printf' in function main() ✘
}
```

לא ניתן לפתוח קובץ

Unable to open include file 'XXX'

הודעה זו תוצג אם שורת include# אינה מצליחה לפתוח את הקובץ הרשום בין הסוגריים המשולשים המופיעים לצידה. קובץ זה חייב להימצא בספריה שמוגדרת מראש למהדר (בדרך כלל היא נקראת include).

- XXX – שם הקובץ אותו לא ניתן לפתוח

```
#include <hohoho.h>       ✘ Unable to open include file 'hohoho.h' ✘
void main()
{

}
```

Unterminated string or character constant in function XXX

הודעה זו תוצג אם הופיע קבוע מחרוזתי או תווי הנפתח בגרש (') או בגרשיים (") אך לא הסתיים בתו זה. קבועים מסוגים אלה חייבים להיתחם בגרשיים משני צדדיהם.

```
#include <stdio.h>
void main()
{
    char ch;
    ch = 'a;      ✗ Unterminated string or character constant in function main() ✗
}
```

נספח ג'

נספח ג' אינו רלוונטי. אין תוכנה שמצורפת לספר זה. קבצי התוכניות מצורפים לספר וכן נמצאים באתר הוד-עמי

התקליטור המצורף והתוכנה TCLite

תקליטור זה מופעל באופן אוטומטי עם הכנסתו לכונן.

אם התקליטור אינו מופעל באופן אוטומטי:

1. נדא שהתקליטור בכונן.
2. בשולחן העבודה לחץ לחיצה כפולה על סמל **המחשב שלי**.
3. לחץ לחיצה כפולה על סמל **כונן התקליטורים**.
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**.

התקליטור כולל:

- קטלוג ספרי מחשבים בהוצאת **הוד-עמי**
- תוכנת TCLite וקבצי קוד מקור של ספר זה.
- תוכנות/תוכניות שירות מועילות
- שאלות נפוצות - לגבי רכישת ספרים בהוצאה
- אודות החברה
- דרכי התקשרות להוצאת **הוד-עמי**

שים לב!

אם מנהל התקן כונן התקליטורים המותקן הוא 16 סיביות - ייתכן שתראה רק 8 תווים ראשונים של שם הקובץ (במקרה ובמקור הוא ארוך יותר). **הסיבה:** כונני תקליטורים במהירות 4x עובדים עם מנהל התקן שעבד בסביבת DOS ו-Windows 3.11 ויכול לעבוד גם עם Windows 9x, למעט היכולת לזהות קבצים עם שמות ארוכים. **הפתרון:** להתקין מנהל התקן 32 סיביות (אם קיים), או לקנות כונן תקליטורים חדש ולוודא שמצורף אליו מנהל התקן 32 סיביות.

קרא קובץ ONCD שנמצא בתקליטור לגבי תכולת התקליטור ולהוראות התקנה של חלק מהתוכנות המצורפות.

התיקיות הרלוונטיות לספר

Books\59299 שכוללת את קבצי קוד המקור

Software\Tclite שכוללת את המהדר

הפעלת התקליטור

במשך כל זמן ההפעלה חייב התקליטור להיות בכונן. הוראות התקנה והפעלה מפורטות נוספות נמצאות בקובץ **ONCD.PDF** בתקליטור.

כדי לקרוא את תוכן הקובץ ONCD.PDF (הכולל פירוט מעוצב של תוכן התקליטור), עליך להתקין את תוכנת Adobe Acrobat Reader מהתפריט **התקנת תוכנות**. בנוסף, תשמש אותך תוכנה זו לקריאה והדפסה של הפרקים לדוגמה, אותם תמצא בקטלוג ספרי המחשבים בהוצאת **הוד-עמי**. התוכנה מאפשרת עיון והדפסת המסמך המוצג, כמו גם חיפוש בעברית ובאנגלית. בעזרת תוכנה זו תוכל גם לקרוא את המסמכים אותם מפרסמת ההוצאה באתר האינטרנט.

קטלוג ספרים

בחלק זה תוכל לעיין בקטלוג הספרים בהוצאת **הוד-עמי**.

עיון בקטלוג ספרי המחשבים של הוצאת **הוד-עמי** כולל:

- תוכן עניינים מלא של הספר
- פרק לדוגמה
- אינדקס לחלק מהספרים

את כל המידע ניתן לראות וגם להדפיס. לצורך זה יש להתקין את תוכנת Acrobat Reader מהתפריט **התקנת תוכנות**.

כדי לצפות בקטלוג העדכני ביותר ולצפות במבצעים המיוחדים מומלץ לבקר באתר האינטרנט של ההוצאה בכתובת www.hod-ami.co.il. הקטלוג מומלץ לצפייה ברזולוציית מסך של 600x800. הקטלוג מומלץ לצפייה באמצעות Internet Explorer 5 ומעלה. התוכנה מצורפת בתקליטור. כדי להתקין אותו בחר בתפריט **התקנת תוכנות** ובחר באפשרות **תוכנות נוספות**.

התקנת תוכנות

התקליטור כולל מיגוון תוכנות שירות, חלקן תוכנות חופשיות (freeware) וחלקן תוכנות שיתופיות (shareware). התוכנות מסופקות כמו שהן (as is). הוצאת הוד-עמי אינה אחראית כלפי יחיד או ארגון עבור כל נזק, אשר ייגרם, אם ייגרם, משימוש בתקליטור. השימוש בתקליטור הוא על אחריותו הבלעדית של המשתמש. **הוצאת הוד-עמי אינה תומכת בתוכנות המסופקות בתקליטור זה.** קרא על התוכנות המצורפות בתקליטור בקובץ ONCD.PDF. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם. הוצאת הוד-עמי אינה אחראית, בשום צורה שהיא, לאופן ולטיב התוכנות המותקנות.

התוכנות המומלצות להתקנה

1. תוכנת Adobe Acrobat Reader כדי שניתן יהיה לעיין וגם להדפיס את הפרקים לדוגמה ואת תוכן העניינים של כל ספר בהוצאת הוד-עמי.
2. תוכנת **TCLite** (מהדר שפת C).

להתקנת שאר התוכנות יש ללחוץ על **התקנת תוכנות**, לאשר את תנאי השימוש בתקליטור וללחוץ על **תוכנות נוספות**. פירוט רחב יותר לגבי חלק מהתוכנות המצורפות בתקליטור תוכל לקרוא בקובץ ONCD.PDF אשר בתקליטור.

קוד המקור

בתקליטור נמצאים קבצי תוכניות. התוכניות נמצאות תחת בתיקיות לפי מספרי הפרקים.

לכל ספר יש מספר ייחודי, הבנוי מה-דאנאקוד, המופיע על העטיפה האחורית של הספר, והוא מתחיל בספרות 59 (לדוגמה 59315).

מספר הפריט של ספר זה הוא: **59299**

התקנת קוד המקור פירושה **העתקה** (לא התקנה) של הקבצים מהתקליטור לדיסק הקשיח במחשב באישי שלך.

לאחר שיועתק קוד המקור הוא יימצא בדיסק שבחרת, בתיקיה HodAmiBooks\59299.

שים לב!

כאשר עובדים במהדר TCLite הכל מתבצע בסביבת המהדר. לא יוצרים קובץ EXE ולא יוצאים ל-DOS כדי להריץ אותו. הספר נכתב לשימוש במהדר רגיל ולכן יש לשים לב לשינויים כאשר עובדים ב-TCLite.

תוכנת TCLite

תוכנת **TCLite** (Turbo C Lite), מהדר לשפת C של חברת Borland International, אשר מיועד למחשבי IBM PC ותואמים, הפועלים תחת מערכת הפעלה MS-DOS. זהו מהדר מהיר מאוד המסוגל, לפי דיווחי החברה, להדר מעל 7,000 שורות בדקה.

התוכנה כוללת:

- מהדר (compiler).
- עורך (editor) המשמש לכתיבת ותיקון התוכניות.
- מערכת תפריטים נוחה, לבחירת האופציות השונות במערכת.
- מקשר (linker).
- מערכת איתור שגיאות בהידור.
- מערכת לבניית תוכניות מקובצי מקור רבים.

אם מותקן כבר מהדר Borland במחשב, אל תתקין את TCLite אלא רק העתק את התוכניות מהתקליטור.
ההתקנה וההפעלה פשוטים. הוראות התקנה והפעלה מלאות נמצאות בתקליטור בקובץ Software\Tclite\TCLiteGuide.pdf

התקנת התוכנה

התהליך הבא מעתיק את קבצי התוכנה לכונן הדיסק הקשיח C: שלך, לתיקיה בשם Tclite הנוצרת באופן אוטומטי.

אם התקליטור לא הופעל אוטומטית בצע שלבים 1-4. אם הופעל - עבור לשלב 5.

1. ודא שהתקליטור נמצא בכונן התקליטורים.
2. בשולחן העבודה לחץ לחיצה כפולה על הסמל **המחשב שלי**.
3. לחץ לחיצה כפולה על סמל כונן התקליטורים.
4. לחץ לחיצה כפולה על הקובץ **hodami.exe**.
5. לחץ על **התקנת תוכנות**.
6. לחץ על לחצן האפשרויות ליד **אני מאשר את תנאי השימוש בתקליטור**.
7. לחץ **המשך**.
8. לחץ על **TCLite**.
9. התוכנה מותקנת בתיקיה **C:\tclite**.

בסיום שלב ההתקנה מועתק קובץ בשם c.bat לתיקיית השורש של כונן C. לנוחיותך ניתן ליצור **קיצור דרך** על שולחן העבודה (ראה ספרים בנושא Windows).

הפעלת התוכנה

1. לחץ לחיצה כפולה על קיצור הדרך ששמו TCLite על שולחן העבודה.
2. סביבת העבודה של TCLite מופיעה בחלון MS-DOS Prompt - TC, ובתוכה חלון (מסמך) בצבע כחול, ששמו: **NONAME00.CPP** (רק בפעם הראשונה שנכנסים לתוכנה).
3. כדאי לסגור את החלון **NONAME00.CP** ע"י צמד המקשים <F3>+<Alt>, ומקבלים חלון אפור ריק, שבראשו רשימת תפריטים על רקע לבן. כרגע אתה מוכן להתחיל לעבוד בעזרת תוכניות הספר.
4. בחר בתפריט **Options** באפשרות **Environment** ולאחר מכן באפשרות **Editor**.
5. בתיבת הדו-שיח **Editor Options** בשדה **Default Extension** הקלד **C** במקום CPP. לחץ OK.
6. בחר בתפריט **Options** באפשרות **Save** כדי לשמור את השינויים שביצעת.

שים לב!

בכל שלב בסעיפי הרצת התוכניות אפשר לבטל שלב כלשהו ולחזור לשלב הקודם ע"י המקש <Esc>.

ברירת המחדל של ה- Compiler היא לקרוא ולשמור קבצי C/CPP. אתם כמובן קוראים קבצי C וכותבים קבצי C.

זהירות!

אם המחשב שלכם מחובר לרשת, המלצתנו - בזמן אתחול המערכת להקיש F5 כדי לדלג על ביצוע הקבצים autoexec.bat ו-config.sys אם לא תעשו כך, ייתכן שבתוכניות תראו תווים לא מובנים במקום עברית.

הרצת תוכנית עם TCLite

1. כדי לקרוא לקובץ תוכנית קיימת יש לבחור בתפריט **File** באפשרות **Open**. נפתחת תיבת דו-שיח בשם: **Load a File**.
2. התיבה Name מכילה את הערך ***.C** (או ***.CPP**, בהתאם למה שבחרת בתפריט Options, Environment, Editor). המשמעות היא שיוצגו בפניך כל התיקיות וכל הקבצים עם סיומת **.C** (או סיומת **.CPP**).
3. אם קבצי התוכניות אינם נמצאים בכונן C, אלא בכונן **E**, לדוגמה, יהיה עליך להקליד בתיבה Name את רצף התווים הבא: **E:*.c** ולהקיש Enter.

4. בתיבה **Files** תוכל לראות את כל שמות התיקיות שבדיסק ואת שמות הקבצים לפי מה שרשום בתיבה **Name** (סיומת C או סיומת CPP). היעזר בפס הגלילה האופקי כדי לשוטט בתיבה Files.
5. כדי לעבור לתיקיה, סמן את התיקיה ולחץ **Open**.
6. כדי לחזור לתיקיה ממנה הגעת, סמן את רצף התווים **\..** ולחץ **Open**. התו \.. נמצא בסוף רשימת הקבצים. אם יש הרבה קבצים באותה תיקיה לא כולם מופיעים בתיבה Files. השתמש בפס הגלילה האופקי כדי להגיע לסוף הרשימה.
7. כדי לפתוח קובץ, סמן את הקובץ ולחץ **Open**.
8. לאחר פתיחת קובץ תוכנית, תוכן הקובץ יופיע בחלון על רקע כחול שכותרתו היא שם הקובץ כולל נתיב התיקיות.
9. להרצת הקובץ יש לבחור בתפריט **Run** באפשרות **Run**.
10. כדי לראות את פלט התוכנית הקלד **Alt+F5** או בחר בתפריט **Window** באפשרות **Output**. היעזר בפסי הגלילה האנכי והאופקי של חלון Output כדי לראות את תוכנו.
11. חלון פעיל הוא חלון שהינך רואה ובפינתו השמאלית העליונה מופיע ריבוע[■]. כדי לסגור את החלון יש לנקוט באחת מהפעולות הבאות:
    - בחירה בתפריט **Window** באפשרות **Close**.
    - לחיצה אחת באמצעות העכבר על הריבוע.
12. מעבר בין חלונות (Message, Output, Watch, User Screen) באמצעות תפריט **Window**.
13. ליצירת קובץ תוכנית חדש יש לבחור בתפריט **File** באפשרות **New**.
14. לשמירת קובץ יש לבחור בתפריט **File** באפשרות **Save**.
15. ליציאה ממהדר Tclite בחר בתפריט **File** באפשרות **Quit**.
16. סגור את החלון על ידי לחיצה על ☒.

קרא קובץ Software\TCLite\TCLiteGuide.pdf שנמצא בתקליטור ללימוד עקרונות העבודה ב-TCLite.

עקרונות העבודה

שיטת העבודה במערכת הפיתוח מבוססת על **מערכת תפריטים** (Menus) שמתוכם בוחרים באפשרויות השונות. בהתאם לבחירה, מופיעים זה על גבי זה, חלונות המשמשים לקליטת נתונים מהמשתמש ולהצגת אפשרויות נוספות.

קבלת עזרה (HELP), מקש F1

בכל עת ניתן לקבל מהמערכת דברי הסבר (באנגלית) המתאימים לשלב העבודה שבו נמצאים. ב**הקשת F1**, המערכת תגיב בהקרנת חלון המכיל דברי הסבר על מקום הימצאך

והאפשרויות העומדות לרשותך באותו שלב עבודה. ייתכן ותופענה מילות מפתח אשר תאפשרנה לקבל הסברים נוספים הקשורים לאותו נושא. כאשר תציב את הסמן על אחת מהמילים האלו ותקיש Enter, יוצג ההסבר המתאים.

כל הקשה על **Alt+F1** תחזיר אותך לחלון העזרה הקודם. הקשה על F1 בזמן שמוקרן חלון עזרה כלשהו, תגרום למערכת להקרין חלון עזרה ראשי, שממנו תוכל לבחור נושא כלשהו ולקבל עליו חלון הסבר.

שים לב!

כאשר ברצוננו לקבל עזרה אודות פונקציה מסוימת, יש להקליד את שם הפונקציה, לדאוג שהסמן יעמוד על הפונקציה, ולהקיש Ctrl+F1. כך נקבל עזרה מותאמת לפונקציה ובדרך כלל תוכנית דוגמה, המסבירה את השימוש בפונקציה.

מקש Esc משמש ליציאה מחלונות העזרה בחזרה לנקודה ממנה נכנסת אליהם.

התפריט הראשי - Main Menu

לתפריט הראשי ניתן להגיע מייד לאחר הרצת טורבו C, או על ידי הקשה על **F10** מנקודה כלשהי בזמן העבודה במערכת. עובדה זו (F10-Main Menu) מצוינת בשורה התחתונה בכל עת.

בזמן היותך בתפריט הראשי, אחד הפריטים (בשורה עליונה) יהיה מודגש (מואר). ביכולתך להציב את הסמן על פריט אחר באמצעות מקשי החיצים ולהקיש Enter, או שתוכל להקיש את האות הראשונה של כל פריט (לדוגמה F - עבור File).

אם אינך נמצא בתפריט הראשי תוכל לבחור פריט מסוים על ידי הקשת Alt והאות הראשונה של הפריט (לדוגמה Alt+F - עבור File).

לפניך תיאור קצר של הפריטים בתפריט הראשי:

- **F**ile - טיפול בקבצים, טעינה, שמירה, החלפת כונן ו/או מדריך ברמת DOS, יציאה זמנית ל DOS וסיום העבודה עם TCLite.
- **E**dit - מצב עריכה של התוכנית הנמצאת בחלון העריכה (העליון).
- **S**earch - חיפוש והחלפת תווי טקסט בקובץ התוכנית.
- **R**un - הרצת תוכנית, לאחר ביצוע הידור וקישור.
- **C**ompile - הפעלת שלבי ההידור והקישור.
- **D**ebug - הרצת התוכנית צעד אחר צעד, קביעת נקודות עצירה, והצגה בערכי משתנים בזמן ריצה.
- **P**roject - הגדרת הידור עבור תוכנית שקטעים שלה נמצאים במספר קבצי מקור.
- **O**ptions - הצגת האפשרויות השונות הקיימות במערכת הפיתוח המוכללת. טיפול במיקום קבצי הספריה וקבצי ההגדרות.

Window - טיפול בחלונות המסך השונים.

Help - מערכת עזרה לסביבת הפיתוח.

שים לב!

מומלץ ללמוד את הקיצורים השונים המצויינים בתפריטים על מנת לחסוך את זמן הגישה לתפריטים.

עתה נבצע סקירה כללית של מירב האפשרויות במסגרת התפריטים אשר תוארו לעיל. הסברים נוספים ניתן למצוא במערכת העזרה. צירופי המקשים אשר יופיעו בסוגריים יבצעו ישירות את פעולת התפריט.

תפריט File

Open (F3) - לאחר הבחירה תפתח תיבת דו-שיח אשר לתוכה ניתן להקיש שם תוכנית כולל המסלול המלא שלה, אם צריך. על ידי הקשת Tab ניתן לעבור לחלקים השונים של תיבת הדו-שיח.

New - פתיחת חלון לתוכנית חדשה אשר רק על ידי שמירה (Save) ומתן שם לתוכנית תיצר קובץ חדש.

Save (F2) - אפשרות זו מיועדת לשמירת הקובץ המצוי בחלון העריכה. אם עדיין לא ניתן שם לקובץ תיפתח תיבת דו-שיח שבה ניתן להקיש את שם הקובץ הרצוי.

Save as - שמירת קובץ קיים בשם אחר. הקובץ הקודם לא יימחק. על הדיסק נמצאים עכשיו שני קבצים זהים לחלוטין, אך בעלי שם שונה.

Save all - שמירת כל הקבצים הפתוחים.

Change dir - שינוי כונן/ספריית העבודה. יש לבחור ב-"OK" בתיבת הדו-שיח כדי שהשינוי יתבצע.

Print - הדפסת התוכנית למדפסת.

Get info - הצגת חלון מידע אשר מכיל פרטים אודות התוכנית והזיכרון שבשימוש.

DOS shell - יציאה זמנית לסביבת DOS. על ידי הקלדת Exit ניתן לחזור לסביבת הפיתוח.

Quit (Alt+x) - סיום הפעילות ב-TCLite והחזרת הפיקוח למערכת ההפעלה.

תפריט Edit

Restore line - שחזור שורת קוד לאחר שינוי או מחיקה.

Cut (Shift+Del) - גזירת קטע קוד מסומן (בלוק) אל תוך ה-clipboard.

Copy (Ctrl+Ins)	העתקת קטע קוד מסומן (בלוק) אל תוך ה-clipboard.
Paste (Shift+Ins)	הדבקת קטע קוד אשר הועתק או נגזר אל תוכנית במקום שעומד הסמן.
Copy example	העתקת תוכנית דוגמה אשר מופיעה ב-Help אל תוך התוכנית.
Show clipboard	מראה מה שיש בלוח.
Clear (Ctrl+Del)	מחיקת קטע קוד מסומן.

פעולות עריכה בחלון הפיתוח

חיצים	תנועת הסמן לכל הכיוונים.
Page Up/Page Down	קפיצת דף מסך מטה/מעלה.
Ctrl+Page Up	עבור לתחילת התוכנית.
Ctrl+Page Down	עבור לסוף התוכנית.
Ctrl+y	מחיקת שורה.
backspace (←)	מחיקת תו קודם.
Delete	מחיקת תו נוכחי (התו שהסמן נמצא מעליו).
חיצים+Shift	סימון קטע קוד (הקטע הרצוי יואר בצבע אפור בהיר).

תפריט Search

Find	תיבת דו-שיח לחיפוש, בה יש להקליד מחרוזת לחיפוש. בחירת "OK" תאשר את ביצוע החיפוש.
Replace	בחירת מחרוזת לחיפוש ומחרוזת להחלפה בתיבת הדו-שיח.
Search again (Ctrl+l)	חזור על פעולת החיפוש/החלפה האחרונה.
Go to line number	עבור לשורה על ידי הקלדת השורה הנבחרת בתיבת הדו-שיח.
ועוד...	

תפריט Run

Run (Ctrl+F9)	ביצוע הידור לתוכנית, והרצה מתוך סביבת הפיתוח (ניתן לראות את פלט התוכנית על ידי הקשת Alt+F5).
Program reset (Ctrl+F2)	הפסקת ריצת התוכנית כאשר נמצאים בריצת צעד-צעד (שורה אחר שורה). אפשר גם להקיש F7.
Go to cursor (F4)	הרצת חלק מהתוכנית, עד למקום הסמן.
Trace into (F7)	הרצת התוכנית שורה אחר שורה, כולל כניסה לפונקציות.

Step over (F8) הרצת התוכנית שורה אחר שורה, לא כולל כניסה לפונקציות.

Arguments העברת ארגומנטים לתוכנית כאשר עובדים עם שורת הפקודה.

תפריט Compile

Compile to OBJ (Alt+F9) בצע הידור לתוכנית.

Make EXE file (F9) צור קובץ ריצה exe.

Link EXE file (Ctrl+F9) בצע הידור והרץ את התוכנית.

Build all בצע הידור לכל הקבצים המשתתפים בפרויקט.

תפריט Debug

Inspect (Alt+F4) פתיחת חלון בו ניתן לראות את ערך המשתנה בתהליך ריצה, שלב אחר שלב.

Evaluate/modify (Ctrl+F4) פתיחת חלון בו ניתן לראות ולשנות ערך של משתנה בתהליך ריצה, שלב אחר שלב.

Call stack (Ctrl+F3) הצצה אל תוכן המחסנית.

watches\Add watch (Ctrl+F7) חלון בו ניתן לצפות בערך המשתנים בזמן ריצה (שלב אחר שלב).

Toggle breakpoint (Ctrl+F8) קביעת נקודות עצירה בתוכנית (שורה מוארת בצבע אדום). התוכנית תרוץ עד לנקודת העצירה הראשונה במשך ריצת התוכנית. הקשה על Ctrl+F9 תביא אותנו לנקודת העצירה השנייה, וכך הלאה עד לסוף התוכנית. קביעת נקודת עצירה על שורה שכבר קבענו לה נקודת עצירה, תבטל את נקודת העצירה.

תפריט Project

Open project פתיחת פרויקט חדש.

Close project סגירת פרויקט.

Add item הוספת קובץ מקור לפרויקט.

Delete item מחיקת קובץ מקור מהפרויקט.

תפריט Options

Directories קביעת מסלולים לספריית המערכת ולספריית העבודה שניצור תחת TCLite.

Environment קביעת תכונות סביבת העבודה: העדפות, סביבת העריכה ומצב העכבר.

תפריט Window

Size/Move (Ctrl+F5)	מסגרת החלון מוצגת בצבע ירוק ובעזרת חיצים + Shift ניתן לשנות את גודל חלון העבודה. בעזרת החיצים בלבד ניתן גם להזיז את חלון העבודה על פני המסך.
Zoom (F5)	הגדלת חלון העבודה למסך מלא. בהקשה נוספת יחזור החלון לגודלו הקודם.
Next (F6)	דילוג בין החלונות הפתוחים.
Close (Alt+F3)	סגירת חלון.
Output	הצגת חלון ריצת התוכנית.
User screen (Alt+F5)	מעבר אל חלון ריצת התוכנית.
List (Alt+0)	הצגת רשימת החלונות הפתוחים בתיבת דו-שיח.

חלון העריכה

בכל פעם שיוצרים קובץ חדש, או כשפותחים קובץ קיים נפתח חלון העריכה.

הרצת תוכנית דרך שורת DOS

בכל פעם ששלב ההידור-קישור עובר ללא כישלון, נוצר קובץ הנושא את שם קובץ התוכנית עם סיומת EXE. זהו קובץ הריצה. כדי לצאת ל-DOS הקש Alt+X, או בחר בפריט **Q**uit שבתפריט **F**ile.

שים לב!

ניתן להריץ תוכנית דרך שורת DOS אחרי הידור ב-Borland בלבד.

בשורת DOS הקש את הפקודה: dir שם_קובץ.* ותוכל לראות שקיימים 4 קבצים בעלי שם כזה, הנושאים סיומות שונות:

.BAK מצב הקובץ לפני עדכונים בחלון העריכה (בתנאי שהמתג Backup Source Files מופעל).

.C קובץ התוכנית.

.EXE קובץ הריצה.

.OBJ קובץ זמני של המהדר (שלב ביניים לפני הקישור לספריות המערכת).

נחזור ונזכיר, שפרק זה מהווה מבוא תמציתי למהדר TCLite ונועד לשרת את התרגול בשפת C בעת הלימוד בספר בלבד.

כתיבת עברית בתוכנית

כדי לכתוב עברית בתוכנית ב-TCLite:

1. העתק את הקובץ heb.com מהספריה בתקליטור software\tclite לספריה הראשית בדיסק הקשיח (בדרך כלל C).
2. הקש במצב DOS - **heb** <C:\> ו-Enter.
3. היכנס ל-tclite וכאשר תרצה להקליד עברית, הקש Ctrl+ (החזק את Ctrl ולחץ על מקש מינוס). נפתח חלון בו תוכל לכתוב. הקש Enter כדי להכניס את מה שהקלדת לתוכנית.

קרא קובץ ONCD שנמצא בתקליטור לגבי תכולת התקליטור ולהוראות התקנה של חלק מהתוכנות המצורפות.

מה עוד בתקליטור?

הוצאת **הוד-עמי** מפיצה תוכנות אלו כבונוס ללקוחות ההוצאה, ואינה מתיימרת לגבות תשלום עבור התוכניות המצורפות ו/או לתמוך בהם.

שים לב!

השימוש בתקליטור זה הוא על אחריותו הבלעדית של המשתמש. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם. הוצאת **הוד-עמי** אינה אחראית, בכל צורה שהיא, לאופן ולטיב התוכנות המותקנות.

בכל שאלה לגבי תוכנה הנמצאת בתקליטור, יש לפנות למפתחי התוכנה (כל תוכנה בנפרד) כפי שמצוין בקבצי העזרה של התוכנה המדוברת.

הקבצים הם גרסאות ShareWare ו-FreeWare.

גרסת ShareWare מאפשרת לך, המשתמש, לבדוק את יעילות התוכנה ואת תאימותה לעבודה אותה אתה מבצע. אם נמצאה התוכנה מתאימה לצרכיך, עליך לשלם למפתחיה תשלום סמלי (לפי הרשום בקבצי העזרה של כל תוכנה ותוכנה בנפרד) כדי לקבל רישיון מלא לשימוש בה. קבלת רישיון לשימוש בתוכנה יפתח בפניך מיגוון אפשרויות שלא עמדו לרשותך בהפעלת גרסת ה-ShareWare.

תוכנת TCLite

תוכנת **TCLite** (Turbo C Lite), מהדר לשפת C של חברת Borland International, אשר מיועד למחשבי IBM PC ותואמים, הפועלים תחת מערכת הפעלה MS-DOS. זהו מהדר מהיר מאוד המסוגל, לפי דיווחי החברה, להדר מעל 7,000 שורות בדקה.

התוכנה כוללת:

- ❖ מהדר (compiler).
- ❖ עורך (editor) המשמש לכתיבת ותיקון התוכניות.
- ❖ מערכת תפריטים נוחה, לבחירת האופציות השונות במערכת.
- ❖ מקשר (linker).
- ❖ מערכת איתור שגיאות בהידור.
- ❖ מערכת לבניית תוכניות מקבצי מקור רבים.

אם מותקן כבר מהדר Borland במחשב, אל תתקין את TCLite אלא רק העתק את התוכניות מהתקליטור.

ההתקנה וההפעלה פשוטים. עקוב אחר הסעיפים הבאים.

התקנת התוכנה

הוראות התקנה אלו נמצאות גם בתקליטור בקובץ install.doc (נכתב ב-Word).

ההתקנה תבוצע מסביבת DOS ולצורך ההסבר נתייחס לכונן C: (אותן פעולות תעבודנה גם על כל כונן אחר).

>>**אם אתה נמצא ב-Windows - כבה והפעל את המחשב במצב DOS**<<

>>**בחלון ביבוי המחשב סמן את האפשרות**

>>Restart the computer in MS-DOS mode?

1. יש לעבור לספריית השורש - על ידי הפקודה: cd\ (קיצור של change directory), ולהגיע למצב זה: C:\>_.
2. במקרה וזה איננו ניסיון התקנה ראשון - יש למחוק את הספריה TCLITE, בה הותקנה התוכנה. עשה זאת על ידי: deltree tclite, כתוב כך: C:\>deltree tclite. לבסוף, הקש <Enter>.

אם הספריה לא קיימת - הסמן יעבור מיידית לשורה חדשה וימתין להוראה הבאה. אם הספריה קיימת - תופיע השאלה הבאה:

Delete directory "tclite" and all its subdirectories? [yn] _

כלומר, "האם אתה בטוח שאתה רוצה לבטל ספריה זו?" הקש על המקש <Y> ואח"כ על <Enter>. המחשב ידווח: Deleting...tclite.

3. עתה יהיה עליך לעבור לכונן המתאים של התקליטור (למשל X:). המעבר ל- X: - על ידי הפקודה: **X:** (ואז על המסך יהיה כתוב: X:\>). לאחר הקשת <Enter> הסמן המהבהב יופיע אחרי X:\>_.

4. עבור לספריה software\tclite על ידי הקלדת הפקודה:

X:\>**cd software\tclite**

5. עכשיו הכל מוכן לביצוע ההתקנה. כתוב את הפקודה:

X:\software\tclite>**install C:**

(שפירושה, התקנה לתוך כונן C:). שים לב: אם תשכח לכתוב C:, כלומר לאן יש לבצע את ההתקנה - פשוט תזור שוב על הפקודה עם 2 חלקיה). המחשב ידווח, הן על התקנת התוכנה TCLITE - גרסה של שפת Turbo C++. משך זמן ההתקנה תלוי במהירות המחשב שלך, ויכול להימשך אפילו מעל 2 דקות.

6. במקרה של תקלה (תקיעת המערכת) יש לבטל את ההתקנה ע"י <Ctrl>+<C>. במקרה זה המחשב ישאל: Terminate batch job (Y/N)?, כלומר, האם לסיים את העבודה. יהיה עליך להקיש <Y> ו- <Enter>, ואז, בהנחה שיש די זיכרון בדיסק הקשיח, לחזור על שלבים 2-5.

7. לאחר סיום ההתקנה (והעתקת קבצי קוד המקור) שים את התקליטור בצד ושמור אותו כגיבוי. הפעלת התוכנית והרצת התוכניות נעשית מתוך הדיסק הקשיח.

הפעלת התוכנה

1. יש לחזור מכונן X: לכונן C:, ולהימצא בספריית השורש, על ידי הפקודה C: והקשת Enter. אם אינך נמצא ב- C:\> הקש cd\ ו-Enter.

2. הכניסה למהדר על ידי הפקודה **c**<C:\ המפעילה את קובץ ההפעלה C.BAT. כתוצאה, מקבלים את סביבת העבודה של Turbo C/C++, ובתוכה חלון פתוח בצבע כחול, ששמו: **NONAME00.CPP** (רק בפעם הראשונה שנכנסים לתוכנה).

3. כדאי לסגור את החלון **NONAME00.CPP** ע"י צמד המקשים <Alt>+<F3>, ואז מקבלים חלון אפור ריק, שבראשו רשימת תפריטים על רקע לבן. כרגע אתה מוכן להתחיל לעבוד בעזרת תוכניות הספר.

שים לב!

ברירת המחדל של ה- Compiler היא לקרוא ולשמור קבצי C/CPP. אתם כמובן קוראים קבצי C וכותבים קבצי C.

הרצת תוכניות

1. כדי לקבל תוכנית - יש להקיש <F3>. הקשה זו פותחת "חלון שיחה", שכותרתו הראשית: Load a File ("טען קובץ") וכותרת משנה ראשונה היא: Name: ואחריה, בתוך שורה כחולה עם הדגשה ירוקה כתוב: *.CPP. אם 5 תווים אלה (*.cpp) מופיעים - הקש Backspace פעמיים כדי שיישאר רק *.c. אם אינך רואה דבר הקלד *.c, ולאחר מכן הקש <Enter>.

2. תעבור לחלון פנימי שכותרתו Files ("קבצים"), ועל רקע בצבע תכלת מצויים 2 טורים של שמות קבצים. באופן כללי, מקשי החיצים מאפשרים לטייל עם ה"סמן", שצבעו ירוק בהיר, לכל הקבצים.

3. למעשה, מה שמוצג כרגע על המסך הוא שמות הקבצים/תוכניות שבספריה programs. הקשה על <Enter> תכניס אותך לתוכנית. כדי לבחור ספריה אחרת (mesimot או targilim) הקש על \.. שמופיע בסוף הרשימה של התוכניות.

4. הקשת <Enter> על שם קובץ תפתח חלון כחול, שכותרתו במרכז הינה: שם הקובץ, ובתוך החלון יופיע הטקסט של תוכנית זו (או של קובץ זה).

5. כדי להריץ את התוכנית עליך להקיש על צמד המקשים: <Ctrl>+<F9>. המחשב יתרגם את התוכנית לשפת מכונה ויבצע אותה. כדי לראות את פלט התוכנית הקש <Alt>+<F5>. יש להקיש על מקש כלשהו כדי לחזור לחלון התוכנית.

6. המשך ההרצה של תוכניות נוספות מומלץ באופן הבא: תחילה לסגור את חלון התוכנית הנוכחית ע"י צמד המקשים <Alt>+<F3>. אח"כ ניתן לחזור על השלבים 1 עד 5, כאשר בשלב 3 אפשר לבחור תוכנית אחרת.

שים לב!

בכל שלב בסעיפי הרצת התוכניות אפשר לבטל שלב כלשהו ולחזור לשלב הקודם ע"י המקש <Esc>.

זהירות!

אם המחשב שלכם מחובר לרשת, המלצתנו - בזמן אתחול המערכת להקיש F5 כדי לדלג על ביצוע הקבצים autoexec.bat ו-config.sys אם לא תעשו כך, ייתכן שבתוכניות תראו תווים לא מובנים במקום עברית.

עקרונות העבודה

שיטת העבודה במערכת הפיתוח מבוססת על **מערכת תפריטים** (Menus) שמתוכם בוחרים באפשרויות השונות. בהתאם לבחירה, מופיעים זה על גבי זה, חלונות המשמשים לקליטת נתונים מהמשתמש ולהצגת אפשרויות נוספות.

קבלת עזרה (HELP), מקש F1

בכל עת ניתן לקבל מהמערכת דברי הסבר (באנגלית) המתאימים לשלב העבודה שבו נמצאים. **בהקשת F1**, המערכת תגיב בהקרנת חלון המכיל דברי הסבר על מקום הימצאך והאפשרויות העומדות לרשותך באותו שלב עבודה. ייתכן ותופענה מילות מפתח אשר תאפשרנה לקבל הסברים נוספים הקשורים לאותו נושא. כאשר תציב את הסמן על אחת מהמילים האלו ותקיש Enter, יוצג ההסבר המתאים.

כל הקשה על **Alt+F1** תחזיר אותך לחלון העזרה הקודם. הקשה על F1 בזמן שמוקרן חלון עזרה כלשהו, תגרום למערכת להקרין חלון עזרה ראשי, שממנו תוכל לבחור נושא כלשהו ולקבל עליו חלון הסבר.

שים לב!
כאשר ברצוננו לקבל עזרה אודות פונקציה מסוימת, יש להקליד את שם הפונקציה, לדאוג שהסמן יעמוד על הפונקציה, ולהקיש Ctrl+F1. כך נקבל עזרה מותאמת לפונקציה ובדרך כלל תוכנית דוגמה, המסבירה את השימוש בפונקציה.

מקש Esc משמש ליציאה מחלונות העזרה בחזרה לנקודה ממנה נכנסת אליהם.

התפריט הראשי - Main Menu

לתפריט הראשי ניתן להגיע מייד לאחר הרצת טורבו C, או על ידי הקשה על **F10** מנקודה כלשהי בזמן העבודה במערכת. עובדה זו (F10-Main Menu) מצוינת בשורה התחתונה בכל עת.

בזמן היותך בתפריט הראשי, אחד הפריטים (בשורה עלינה) יהיה מודגש (מואר). ביכולתך להציב את הסמן על פריט אחר באמצעות מקשי החיצים ולהקיש Enter, או שתוכל להקיש את האות הראשונה של כל פריט (לדוגמה F - עבור File).

אם אינך נמצא בתפריט הראשי תוכל לבחור פריט מסוים על ידי הקשת Alt והאות הראשונה של הפריט (לדוגמה Alt+F - עבור File).

לפניך תיאור קצר של הפריטים בתפריט הראשי:

File - טיפול בקבצים, טעינה, שמירה, החלפת כונן ו/או מדריך ברמת DOS, יציאה זמנית ל-DOS וסיום העבודה עם TCLite.

Edit - מצב עריכה של התוכנית הנמצאת בחלון העריכה (העליון).

Search - חיפוש והחלפת תווי טקסט בקובץ התוכנית.

Run - הרצת תוכנית, לאחר ביצוע הידור וקישור.

Compile - הפעלת שלבי ההידור והקישור.

Debug - הרצת התוכנית צעד אחר צעד, קביעת נקודות עצירה, והצצה בערכי משתנים בזמן ריצה.

Project - הגדרת הידור עבור תוכנית שקטעים שלה נמצאים במספר קבצי מקור.

Options - הצגת האפשרויות השונות הקיימות במערכת הפיתוח המוכללת. טיפול במיקום קבצי הספריה וקבצי ההגדרות.

Window - טיפול בחלונות המסך השונים.

Help - מערכת עזרה לסביבת הפיתוח.

שים לב!

מומלץ ללמוד את הקיצורים השונים המצויינים בתפריטים על מנת לחסוך את זמן הגישה לתפריטים.

עתה נבצע סקירה כללית של מירב האפשרויות במסגרת התפריטים אשר תוארו לעיל. הסברים נוספים ניתן למצוא במערכת העזרה.

צירופי המקשים אשר יופיעו בסוגריים יבצעו ישירות את פעולת התפריט.

תפריט File

Open (F3) - לאחר הבחירה תפתח תיבת דו-שיח אשר לתוכה ניתן להקיש שם תוכנית כולל המסלול המלא שלה, אם צריך. על ידי הקשת Tab ניתן לעבור לחלקים השונים של תיבת הדו-שיח.

New - פתיחת חלון לתוכנית חדשה אשר רק על ידי שמירה (Save) ומתן שם לתוכנית תייצר קובץ חדש.

Save (F2) - אפשרות זו מיועדת לשמירת הקובץ המצוי בחלון העריכה. אם עדיין לא ניתן שם לקובץ תפתח תיבת דו-שיח שבה ניתן להקיש את שם הקובץ הרצוי.

Save as - שמירת קובץ קיים בשם אחר.

הקובץ הקודם לא יימחק. על הדיסק נמצאים עכשיו שני קבצים זהים לחלוטין, אך בעלי שם שונה.

Save all	שמירת כל הקבצים הפתוחים.
Change dir	שינוי כונן/ספריית העבודה. יש לבחור ב-"OK" בתיבת הדו-שיח כדי שהשינוי יתבצע.
Print	הדפסת התוכנית למדפסת.
Get info	הצגת חלון מידע אשר מכיל פרטים אודות התוכנית והזיכרון שבשימוש.
DOS shell	יציאה זמנית לסביבת DOS. על ידי הקלדת Exit ניתן לחזור לסביבת הפיתוח.
Quit (Alt+x)	סיום הפעילות ב-TCLite והחזרת הפיקוח למערכת ההפעלה.

תפריט Edit

Restore line	שיחזור שורת קוד לאחר שינוי או מחיקה.
Cut (Shift+Del)	גזירת קטע קוד מסומן (בלוק) אל תוך ה-clipboard.
Copy (Ctrl+Ins)	העתקת קטע קוד מסומן (בלוק) אל תוך ה-clipboard.
Paste (Shift+Ins)	הדבקת קטע קוד אשר הועתק או נגזר אל תוכנית במקום שעומד הסמן.
Copy example	העתקת תוכנית דוגמה אשר מופיעה ב-Help אל תוך התוכנית.
Show clipboard	מראה מה שיש בלוח.
Clear (Ctrl+Del)	מחיקת קטע קוד מסומן.

פעולות עריכה בחלון הפיתוח

חיצים	תנועת הסמן לכל הכיוונים.
Page Up/Page Down	קפיצת דף מסך מטה/מעלה.
Ctrl+Page Up	עבור לתחילת התוכנית.
Ctrl+Page Down	עבור לסוף התוכנית.
Ctrl+y	מחיקת שורה.
backspace (↵)	מחיקת תו קודם.
Delete	מחיקת תו נוכחי (התו שהסמן נמצא מעליו).
Shift+חיצים	סימון קטע קוד (הקטע הרצוי יואר בצבע אפור בהיר).

תפריט Search

Find תיבת דו-שיח לחיפוש, בה יש להקליד מחרוזת לחיפוש. בחירת "OK" תאשר את ביצוע החיפוש.

Replace בחירת מחרוזת לחיפוש ומחרוזת להחלפה בתיבת הדו-שיח.

Search again (Ctrl+I) חזור על פעולת החיפוש/החלפה האחרונה.

Go to line number עבור לשורה על ידי הקלדת השורה הנבחרת בתיבת הדו-שיח.

ועוד...

תפריט Run

Run (Ctrl+F9) ביצוע הידור לתוכנית, והרצה מתוך סביבת הפיתוח (ניתן לראות את פלט התוכנית על ידי הקשת Alt+F5).

Program reset (Ctrl+F2) הפסקת ריצת התוכנית כאשר נמצאים בריצת צעד-צעד (שורה אחר שורה). אפשר גם להקיש F7.

Go to cursor (F4) הרצת חלק מהתוכנית, עד למקום הסמן.

Trace into (F7) הרצת התוכנית שורה אחר שורה, כולל כניסה לפונקציות.

Step over (F8) הרצת התוכנית שורה אחר שורה, לא כולל כניסה לפונקציות.

Arguments העברת ארגומנטים לתוכנית כאשר עובדים עם שורת הפקודה.

תפריט Compile

Compile to OBJ (Alt+F9) בצע הידור לתוכנית.

Make EXE file (F9) צור קובץ ריצה exe.

Link EXE file (Ctrl+F9) בצע הידור והרץ את התוכנית.

Build all בצע הידור לכל הקבצים המשתתפים בפרויקט.

תפריט Debug

Inspect (Alt+F4) פתיחת חלון בו ניתן לראות את ערך המשתנה בתהליך ריצה, שלב אחר שלב.

Evaluate/modify (Ctrl+F4) פתיחת חלון בו ניתן לראות ולשנות ערך של משתנה בתהליך ריצה, שלב אחר שלב.

Call stack (Ctrl+F3) הצצה אל תוכן המחסנית.

watches\Add watch (Ctrl+F7)	חלון בו ניתן לצפות בערך המשתנים בזמן ריצה (שלב אחר שלב).
Toggle breakpoint (Ctrl+F8)	קביעת נקודות עצירה בתוכנית (שורה מוארת בצבע אדום). התוכנית תרוץ עד לנקודת העצירה הראשונה במשך ריצת התוכנית. הקשה על Ctrl+F9 תביא אותנו לנקודת העצירה השנייה, וכך הלאה עד לסוף התוכנית. קביעת נקודת עצירה על שורה שכבר קבענו לה נקודת עצירה, תבטל את נקודת העצירה.

תפריט Project

Open project	פתיחת פרויקט חדש.
Close project	סגירת פרויקט.
Add item	הוספת קובץ מקור לפרויקט.
Delete item	מחיקת קובץ מקור מהפרויקט.

תפריט Options

Directories	קביעת מסלולים לספריית המערכת ולספריית העבודה שניצור תחת TCLite.
Environment	קביעת תכונות סביבת העבודה: העדפות, סביבת העריכה ומצב העכבר.

תפריט Window

Size/Move (Ctrl+F5)	מסגרת החלון מוצגת בצבע ירוק ובעזרת חיצים + Shift ניתן לשנות את גודל חלון העבודה. בעזרת החיצים בלבד ניתן גם להזיז את חלון העבודה על פני המסך.
Zoom (F5)	הגדלת חלון העבודה למסך מלא. בהקשה נוספת יחזור החלון לגודלו הקודם.
Next (F6)	דילוג בין החלונות הפתוחים.
Close (Alt+F3)	סגירת חלון.
Output	הצגת חלון ריצת התוכנית.
User screen (Alt+F5)	מעבר אל חלון ריצת התוכנית.
List (Alt+0)	הצגת רשימת החלונות הפתוחים בתיבת דו-שיח.

חלון העריכה

בכל פעם שיוצרים קובץ חדש, או כשפותחים קובץ קיים נפתח חלון העריכה.

הרצת תוכנית דרך שורת DOS

בכל פעם ששלב ההידור-קישור עובר ללא כישלון, נוצר קובץ הנושא את שם קובץ התוכנית עם סיומת EXE. זהו קובץ הריצה. כדי לצאת ל-DOS הקש Alt+X, או בחר בפריט **Q**uit שבתפריט **F**ile.

שים לב!
ניתן להריץ תוכנית דרך שורת DOS אחרי הידור ב-Borland בלבד.

בשורת DOS הקש את הפקודה: dir קובץ_שם.* ותוכל לראות שקיימים 4 קבצים בעלי שם כזה, הנושאים סיומות שונות:

- **BAK**. מצב הקובץ לפני עדכונים בחלון העריכה (בתנאי שהמתג Backup Source Files מופעל).
- **C**. קובץ התוכנית.
- **EXE**. קובץ הריצה.
- **OBJ**. קובץ זמני של המהדר (שלב ביניים לפני הקישור לספריות המערכת).

נחזור ונזכיר, שפרק זה מהווה מבוא תמציתי למהדר TCLite ונועד לשרת את התרגול בשפת C בעת הלימוד בספר בלבד.

כתיבת עברית בתוכנית

כדי לכתוב עברית בתוכנית ב-TCLite:

1. העתק את הקובץ heb.com מהספריה בתקליטור software\tclite לספריה הראשית בדיסק הקשיח (בדרך כלל C).
2. הקש במצב DOS - **heb** ו-Enter<C:\.
3. היכנס ל-tclite ובאשר תרצה להקליד עברית, הקש Ctrl+ (החזק את Ctrl ולחץ על מקש מינוס). נפתח חלון בו תוכל לכתוב. הקש Enter כדי להכניס את מה שהקלדת לתוכנית.

קרא קובץ ONCD שנמצא בתקליטור לגבי תכולת התקליטור ולהוראות התקנה של חלק מהתוכנות המצורפות.

מה עוד בתקליטור?

הוצאת **הוד-עמי** מפיצה תוכנות אלו כבונוס ללקוחות ההוצאה, ואינה מתיימרת לגבות תשלום עבור התוכניות המצורפות ו/או לתמוך בהם.

שים לב!

השימוש בתקליטור זה הוא על אחריותו הבלעדית של המשתמש. המוצרים המותקנים בתקליטור זה מסופקים באחריות החברות המייצרות אותם. הוצאת **הוד-עמי** אינה אחראית, בכל צורה שהיא, לאופן ולטיב התוכנות המותקנות.

בכל שאלה לגבי תוכנה הנמצאת בתקליטור, יש לפנות למפתחי התוכנה (כל תוכנה בנפרד) כפי שמצוין בקבצי העזרה של התוכנה המדוברת.

הקבצים הם גרסאות ShareWare ו-FreeWare.

גרסת ShareWare מאפשרת לך, המשתמש, לבדוק את יעילות התוכנה ואת תאימותה לעבודה אותה אתה מבצע. אם נמצאה התוכנה מתאימה לצרכיך, עליך לשלם למפתחיה תשלום סמלי (לפי הרשום בקבצי העזרה של כל תוכנה ותוכנה בנפרד) כדי לקבל רישיון מלא לשימוש בה. קבלת רישיון לשימוש בתוכנה יפתח בפניך מיגוון אפשרויות שלא עמדו לרשותך בהפעלת גרסת ה-ShareWare.

אינדקס

-- 166
- 123
" 35, 41, 56, 110, 124, 167, 178, 179, 222
115
#include
stdio.h 104, 139
string.h 397, 399
% 123, 124, 127
%= 160, 170
d% 137
& 138, 392
&& 213
{ } 102
( ) 115, 117, 123, 124
/* */ 109
* 115, 123
*= 160, 170
... 115
/ 123, 124, 126, 128
/= 160, 170
: 115
; 115, 122, 251
[ ] 115, 338, 348, 366
^ 123, 127
|| 216
+ 123
++ 165
+= 160, 170
-= 160, 170
= 115, 122, 159, 160, 170, 360, 362, 367, 385

A

address operator (&) 138, 392
arguments 264
arithmetic
  expression 34, 123, 128
  operators 123, 127, 128
array
  initialization 360, 362, 367
assembler 88
assembly 420
assembly file 88
assignment 115
  expression 123, 160, 170
  operator 122, 159, 160, 170, 360, 362, 367, 385
  statement 122
auto 114

B

BCPL 422
break statement 114, 207, 283

C

C 422
case statement 114, 206
char 114
command prompt 91, 94
compilation 88, 93, 95
compound assignment operators 160, 170
conditional 40
const 114
constant 110, 114, 125, 126, 128
continue statement 114, 287
control string 137
conversion characters 131, 133
  %d 137

D

debugging 109
default statement 114, 212
do 114
double 114, 126, 128

E
185 ,114 else statement
114 enum
97 error
133 ,110 escape sequence
222 ,179 ,178 ,167 ,124 ,110 ,41 ,35 escape sequence
419 executable file
115 expression
128 ,123 ,34 arithmetic
170 ,160 ,123 assignment
114 extern

F
114 float
279 ,114 for statement
116 function
call
264 ,117 arguments
264 ,117 ,115 comma
119 library
391 input
389 output
139 ,104 stdio.h
399 ,397 string.h
102 main

G
280 ,114 goto statement
280 label

H
420 high-level language

I
92 IDE
114 if statement
94 ,92 indentation
initialization
array
367 multi-dimensional
360 dimensional-one
63 ,34 ,32 input
137 ,130 ,125 ,114 int

K

114 keyword

L

280 label statement

library

119 file

95 ,93 ,89 link

89 linker

213 (&&) logical-and operator

216 (||) logical-or operator

114 long

loop

49 counter

283 nesting

420 low-level language

M

419 machine language

419 microprocessor

367 multi-dimensional array

N

nesting

110 comments

283 loop statements

O

88 object file

360 one-dimensional array

operators

392 ,138 (&) address

128 ,127 ,123 arithmetic

385 ,367 ,362 ,360 ,170 ,160 ,159 ,122 assignment

262 comma

logical

213 (&&) and

216 (||) or

263 ,222 ,179 ,167 ,123 ,35 precedence

166 self decrement

165 self increment

88 optimization

64 output

P
palindromic
275 numbers
393 strings
166 postfix
165 prefix

R
114 register
320 ,114 return

S
166 self decrement operator
165 self increment operator
114 short
114 signed
114 sizeof
87 source code
statement
122 assignment
114 static
139 ,104 stdio.h
string
114 constant
137 ,130 control
library functions
405 strcat
408 strcmp
400 strcpy
397 strlen
393 palindromic
399 ,397 string.h
114 struct
114 switch statement
207 break
206 case
212 default

T
114 typedef

U

union 114

unsigned 114

V

variable

array 338

double 126, 128

int 125, 130, 137

void 114

volatile 114

W

warning 98

while statement 114

white-space characters 103

א
אופטימיזציה 88
אופרטורים operators 115
אריתמטיים 123, 127, 128
העלאה עצמית 161, 165
הפחתה עצמית 161, 166
השמה 122, 159, 169, 360, 362, 367, 385
מקוצרים 159
יחס 178
כתובת (&) 138, 392
לוגיים
או (||) 216
וגם (&&) 213
שילוב 221, 226
שלילה (!) 219
סדר פעולות 35, 123, 167, 179, 222, 263
פסיק 262
אזהרה 98
איבר 338
איטרציה iteration 235
אלגוריתם algorithm 27, 28
אליפסת התחלה 66
אליפסת סוף 66
אסמבלר 88, 420
ארגומנט 117, 264
אתחול
מערך 367
דו-מימדי 362
חד-מימדי 360
רב-מימדי 367

ב
ביטוי 115
for 251, 259, 265
אריתמטי 34, 123, 128
השמה 123, 160, 170
יחס 177
לוגי, שקר 224
לולאה
do-while 242
for 251, 280

מורכב 164
תנאי 177
בעיה לוגית 27

ג

גלישה 207
גרשיים
' 145, 381
" 103, 381

ד

דגל
I - 95
L - 95

ה

הגדרה
מערך דו-מימדי 348
מערך חד-מימדי 338
מערך רב-מימדי 366
שורת הגדרה 128
הודעות מהדר
אזהרה 98
שגיאה 93, 96, 97, 98, 427
הוספת מחרוזת 405
הזחה 92, 94 indentation
הידור 88, 93, 95 compilation
הערות 107 comments
מקוננות 110
שימושים 109
העתקת מחרוזת 400
השוואת מחרוזות 408
השמה 34 assignment

ח

חזקה 123, 127, 240
חיבור 123
חילוק 123, 124, 126, 128
חיסור 123

ט

טבלת ASCII 143, 144
טעות תחבירית 29

י
יצירת קובץ הרצה
מסביבת פיתוח משולבת 92
משורת הפקודה 94

כ
כלי פיתוח 87
כפל 123

ל
לולאה 48
do-while 242
for 251, 280
while 233, 242
אינסופית 279
בצע...עד ש... 55
בצע...פעמים 48
כל עוד...בצע 53
מונה 49
מקוננות 56, 270, 283
משפטי תנאי, שילוב 266
עבור...מ-....עד...בצע 51

מ
מהדר 88, 97, 421 compiler
אופטימיזציה 88
אסמבלר 88, 420
הידור 88
קדם-מעבד 88 preprocessor
מונה 49
מזהה 114 identifier
חוקים בנתינת שמות 33, 122
מחלקות 426
מחרוזות פלינדרומיות 393
מחרוזת 381 string
אורך 397
אתחול 385
בקרה 130, 137
הוספה 405
העתקה 400
השוואה 408
מערך 415

מערך תווי חד-מימדי 383
פונקציות ספריה 396
strcat 405
strcmp 408
strcpy 400
strlen 397
פלט 389
קבוע 114, 381
קלט 391, 392
שליחה לפונקציה 417
מילת מפתח keyword 114
מיקרו-מעבד 419
מספרים
אי-רציונליים irrational 20
טבעיים natural 20
פלינדרומיים 275
ראשוניים 290
שליליים negative 20
מערך 57, 337 array
איבר element 338
אתחול
דו-מימדי 362
חד-מימדי 360
רב-מימדי 367
דו-מימדי 347
חד-מימדי 338
פנייה 348, 366
רב-מימדי 365
שליחה לפונקציה 371
דו-מימדי 377
חד-מימדי 371
מציאת אורך מחרוזת 397
משפט
break 114, 207, 283
case 114, 206
continue 114, 287
default 114, 212
else 114, 185
for 114
אופרטור הפסיק 262

אינסופית 279
ביטויים 251, 259, 265, 280
goto 114, 280
תווית 280
if 114, 180
ביטוי תנאי 177
switch 114
break 207
case 206
default 212
גלישה 207
while 114
השמה 122
תווית 280
משתנה 32, 121 variable
אתחול 158 initialization
הגדרה 128
הצבת ערכים 122
מונה 49
מערך 338
סוגים 125, 172
עשרוני 126, 128 float
פונקציות 327
קליטת ערכים 137
שלם 125, 130, 137 integer
שם 33, 122
תווי 143
מתכנת 27 programmer

נ

ניידות 425
ניפוי שגיאות 109 debugging
נקודה-פסיק 115, 122, 251

ס

סביבת פיתוח משולבת 91, 92
סוגריים
מסולסלים 98, 115, 184, 360, 363
מרובעים 115, 338, 348, 366
רגילים 115, 117, 123, 124
סימני פיסוק 115

ע
עצרת 81, 238, 239, 265, 268
ערך
הצבה 122
מספרי 145
קליטה 137

פ
פונקציה ראשית 102
פונקציות 116, 293 functions
getch 151
gets 392
putch 150
printf 153, 389
scanf 151, 391
stdio.h 104, 139
strcat 405
strcmp 408
strcpy 400
string.h 397, 399
strlen 397
אבטיפוס 323
הגדרה 294, 333
החזרת ערך 313
הצהרה 322, 333
משתנים 327
סדר ביצוע 300
ספריה 119
פרמטרים 303
פשוטות 296
קריאה 117
ארגומנטים 117, 264
פסיק 115, 117, 264
שם 294, 295
פיבונאצ'י 83 fibonacci
פלט 34 output
מחרוזות 389
פונקציות ספריה
printf 389
פעולת פלט אלגוריתמית 64
קבועים 137, 390

תווים 146
פלינדרום
מחרוזת 393
מספר 275
פסאודו-קוד pseudo code 32
פסיק 115
פעולות אלגוריתמיות
הצבה ועיבוד 34, 66
פלט 64
קלט 32, 34, 63
תנאי 40

ק

קבוע constant / literal 114
מחרוזתי 114
עשרוני 114, 126, 128
שלם 114, 125
תווי 110, 114
קדם-מעבד 88
קובץ file
אובייקטים object 88
אסמבלי assembly 88
הרצה executable 88, 419
יצירה 90
מקור source 87
ספריה library 119
קוד מקור source code 87
קינון nesting
הערות 110
לולאות 283
תנאי 47
קישור link 89, 93, 95
קלט input 32
מחרוזות 391, 392
פונקציות ספריה
scanf 391
פעולת קלט אלגוריתמית 32, 34, 63
תווים 148
קשר 89

ר

ריבוע קסם 353, 354

ש
שארית 123, 124, 127
שגיאה 97
שגיאת תחביר 29 syntax error
שורה חדשה 106, 110, 124, 167, 178, 179, 222
שורש 37, 42, 72, 123
שורת הגדרה 128
שורת פקודה 91, 94
שפות תכנות 28
C 422
התפתחות 422
תקן K&R 423
C++
מחלקה 426
שפה עילית 420
שפת מכונה 419
שפת סף 420

ת
תו null 382
תווים characters
בקרה 110, 133
המרה 131, 133
d% 137
לבנים 103
ערך מספרי 145
קבועים 110, 114
ריקים 99, 169
תווית 280
תוכנית שלד 101 skeleton application
תכנות
מונחה עצמים 426
מושגים 27
תנאי 38, 177 condition
if 114, 180
קינון 191
if-else 114, 185
קינון 194
או 45
אם..אז 45
לולאה 53

קינון 47
~~תקליטור מצורף 21, 445~~
תקן
ANCI C 423
K&R 423
תרשימי זרימה flowchart 61
אליפסת התחלה 66
אליפסת סוף 66
לולאות 77
פעולת השמה 66
פעולת פלט 64
פעולת קלט 63
תנאים 69
קינון 75

ת.ד. 6108 הרצליה 46160 **הזמנות באתר**
info@hod-ami.co.il www.**hod-ami**.co.il

קטלוג ינואר 2017

	כולל	עמ'	מחיר*
אינטרנט - מפתחי אתרים/גרפיקה			
אמא, אבא - בניתי אתר באינטרנט (HTML)		256	29
הגדל את הכנסות העסק שלך באמצעות **פרסום בגוגל Google AdWords**		300	249
HTML5 המדריך לבניית אתרים ולמערכות WEB, הדור הבא – מהד' 2		390	159
The **Java** Tutorial סדנת לימוד	CD	768	179
JavaScript סדנת לימוד		586	159
מדריך **ASP.NET MVC 4**		514	199
ASP.NET 3.5 סדנת לימוד בשפות #C ו-VB		824	99
תכנות			
Code Complete – מדריך מעשי לפיתוח תוכנה		288	139
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
Visual C# 3.0 סדנת לימוד		656	99
ללמוד **C** - מהד' 3		480	139
שפת **אסמבלי** למחשב האישי, מהד' 2		314	89
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 4		152	95
PC - חומרה, תוכנה ורשתות			
מדריך Hacking ואבטחת מידע, מהד' 2		428	169
מדריך **חומרה ותוכנה** לטכנאי PC - מהד' 5 (כולל חלונות 7/8)		752	189
מדריך **רשתות** לטכנאי PC ולמנהלי רשת - מהד' 4		608	219
Windows			
Windows 8.1 מדריך למשתמש		544	149
Windows 8 מדריך למשתמש		438	19
Windows 7 צעד-אחר-צעד		272	19
LINUX			
LINUX למתקדמים, טיפים, טריקים ותכנות ב-BASH		226	189

* מחיר מומלץ לצרכן כולל מע"מ

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il

	כולל	עמ'	מחיר*
גרפיקה			
Flash – ספר הדרכה ותרגילים		122	64
אינדיזיין – ספר הדרכה ותרגילים		132	72
Illusatrator – ספר הדרכה ותרגילים		120	64
Photoshop צעד אחר צעד (צבע מלא, למתחילים), מהד' 3 כריכה קשה		200	159
Photoshop צעד אחר צעד (ש/ל, למתחילים), מהד' 3		200	89
מדריך לתוכנת העיצוב והאנימציה **3ds max** (2 כרכים)	CD	1400	289
OFFICE			
יישומי סטטיסטיקה בגיליון אלקטרוני Excel		86	69
סטטיסטיקה יישומית		150	97
טבלאות ציר – ניתוח נתונים חכם		116	87
יסודות התכנות ב-**VBA** לתוכנת **Excel**, מהד' 4		152	95
Access 2016 צעד אחר צעד		384	169
Word 2016 צעד אחר צעד		150	59
Excel 2016 צעד אחר צעד		336	129
PowerPoint 2010 צעד אחר צעד		202	69
Outlook 2010 צעד אחר צעד		190	69
Access 2010 סדנת לימוד		360	179
עוד ספרים בגרסאות קודמות (2007 ו-2003) ניתן למצוא באתר הוד-עמי			
ניהול, כלכלה ושונות			
לחפש באגים, מדריך מעשי לבודק תוכנה, מהד' 3		350	169
ניהול ממוקד לעשות יותר עם מה שיש (כריכה קשה) - מהד' 4		358	133
לי זה עולה יותר (תמחיר) (כריכה קשה) - מהד' 3		368	129
מערכות מידע			
Oracle SQL יכולות מתקדמות		626	249
SAS (**Statistical Analysis System**) – ספר לימוד		256	169
בסיסי נתונים ושפת **SQL** – עקרונות ועיצוב		648	149
ניתוח מערכות מידע כולל מתודולוגיית ה- **UML**		818	229
המדריך העברי השלם **UML**		346	329
ספרים דיגיטליים			
לרכישת ספרים לצפייה בפורמט PDF היכנס לאתר לקטגוריה "ספרים דיגיטליים"			
קבצי תרגול לספרים			
קבצי תרגול לספרים שונים תמצא באתר בקטגוריה "קבצי תרגול לספרים"			

* מחיר מומלץ לצרכן כולל מע"מ. קטלוג 1/2017

היכנס לאתר להתעדכן בספרים החדשים ובמחירי המבצע בהוצאה

תוכן עניינים ופרקים לדוגמה www.**hod-ami**.co.il