ללמוד
C

מהדורה שלישית

פנה לאתר האינטרנט של הספר כדי להוריד את התוכניות שבספר והפתרונות:
www.hod-ami.co.il/59315.html

עורך ראשי: **יצחק עמיהוד**
עריכה לשונית ועיצוב: **שרה עמיהוד, טליה טופז**
עיצוב עטיפה: **סטודיו מצגר**

תודה לחן קורן ולשאר הקוראים על הערותיהם

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

ללמוד
C

מהדורה שלישית

יואב נתיב

הוצאת
הוד-עמי
ספרי מחשבים

Learn C - Third Edition

By **Yoav Nativ**

Editor: **I. Amihud**

הוצאת הוד-עמי
לספרי מחשבים בע"מ
ת.ד. 6108 הרצליה 46160
טלפון: 09-9564716 פקס: 09-9571582

הודפס בישראל
מהדורה 1 - 1996
מהדורה 2 - 1999, 2000
מהדורה 3 - 2/2001, 8/2001

HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, December 1996
Third Edition - 2001

מסת"ב 0-284-361-965 ISBN

ספר זה מוקדש למשפחתי

אשתי יעל ובתי רומי

ולנכדים שי

ומי אהרון ברומברג ע"ז

ומתי שרה ברומברג ע"ז

ומי אברהם בנו נעוב ע"ז

תוכן עניינים מקוצר

חלק 2: מבני נתונים

חלק 3: דוגמאות יישומיות

תוכן העניינים

חלק 3: דוגמאות יישומיות

הקדמה

היסטוריה של השפה

שפת C פותחה בראשית שנות ה- 70 על ידי דניס ריצ'י (Dennis M. Ritchie) במעבדות חברת Bell. במקור, נועדה השפה למערכת ההפעלה UNIX, אך מהר מאוד היא התפשטה ויצאו מספר גרסאות שונות לשפת C, אשר נבדלו זו מזו בהבדלים קטנים. בשנת 1983 הקים ANSI, מכון התקנים האמריקאי (American National Standard Institute) ועדה, כדי לקבוע תקן מוסכם לשפה ולמנוע את ההבדלים השונים בין הגרסאות. התקן שנקבע נקרא **ANSI C**, וכל המהדרים הקיימים כיום בשוק בנויים לפי תקן זה.

תכונות השפה

- תכנות פרוצדורלי.
- תכנות מודולרי.
- מיגוון פקודות לשליטה על זרימת התוכנית.
- מיגוון רחב של סוגי משתנים.
- מספר מצומצם מאוד של פקודות.
- אפשרות לגשת ל"לב המחשב" ולשילוב קטעי קוד בשפת Assembly.

תכנות פרוצדורלי

התוכנית מחולקת למספר קטעי קוד קטנים הנקראים **פונקציות** (functions). כל קטע קוד יטפל בבעיה מסוימת, ויהיה לו תפקיד קבוע וידוע, כמו חישוב עצרת, הדפסת ממוצע, או כתיבה וקריאה מקבצים.

תכנות מודולרי

זאת האפשרות ליצור פרויקט שלם בשפת C, המורכב ממספר קבצי מקור שונים. כך ניתן לחלק עבודה בין מספר אנשים.

מיגוון פקודות לשליטה על זרימת התוכנית

המחשב מבצע את פקודות התוכנית לפי הסדר, זו אחר זו. בעזרת פקודות השולטות על זרימת התוכנית ניתן לגרום לביצוע מותנה של חלקים בתוכנית, או ביצוע קטעים מסוימים מספר רב של פעמים על ידי שימוש בלולאות.

מיגוון רחב של סוגי משתנים

בשפת C יש מיגוון רחב של סוגי משתנים בהם ניתן לשמור נתונים בזיכרון. יש סוגי משתנים עבור תווים, מספרים שלמים, מספרים שלמים גדולים מאוד ומספרים ממשיים.

מספר קטן של פקודות

בשפת C יש מספר קטן של פקודות, ורק הפקודות ההכרחיות נמצאות בה. בזכות צמצום הפקודות בשפה, המהדר (compiler) של השפה קטן ופשוט, ותוצאות ההידור יעילות. כמו כן כושר הניידות (portability) של השפה בין מערכות שונות גבוה מאוד. ספריית פונקציות ענפה משלימה את הפקודות החסרות, כמו למשל פונקציות קלט ופלט.

אפשרות לגשת ל"לב המחשב" ולשילוב קטעי קוד בשפת Assembly

בשפת C ניתן להפעיל פסיקות (interrupts) בשפת מכונה, ואף לגשת ישירות לכתובות בזיכרון. עובדה זו נותנת עוצמה רבה לשפה ותורמת לפופולריות שלה בקרב מתכנתים רבים.

הנחיות להורדת קוד מקור מהאתר

פנה לאתר האינטרנט של הספר כדי להוריד את התוכניות והפתרונות של הספר:

www.hod-ami.co.il/59315.html

דבר המחבר

כמרצה לשפת C נוכחתי בהיעדר ספר לימוד בעברית המתייחס לנושאים הבסיסיים הנדרשים ללימוד השפה בצורה תמציתית ומדויקת. על כן הוספתי בסוף כל פרק נקודות לסיכום, דוגמה מסכמת ומיגוון תרגילים. בפרקי הספר השונים משולבות דוגמאות ענייניות להבהרת הנושאים הנלמדים בו.

הפרקים בנויים בסדר דידקטי, ולכן רצוי ללמוד כל פרק לפי הסדר, ורק אחר כך לעבור לפרק הבא. אם תפעל בהתאם להנחיות אלו, סבורני שבסופו של דבר תרכוש את הידע והניסיון הדרושים לך בעבודה כמתכנת מתחיל בשפת C.

לאחר ניסיון המהדורה הראשונה והשנייה, ספר זה במהדורה 3 חולק ל- 3 חלקים:

חלק 1: לימוד בסיסי.

חלק 2: מבני נתונים.

חלק 3: דוגמאות יישומיות. כאן ריכזתי מספר תרגילים שמכילים מספר נושאים והבאתי את פתרונם המלא.

נספח ב' מכיל תרגילים נוספים בכל הנושאים לחזרה ולהטמעה של החומר. כמו כן, צירפתי פתרונות נבחרים כולל הסברים אשר מכילים את התרגילים החשובים ביותר בכל פרק. בנוסף, באתר הוד-עמי תמצא **פתרונות מלאים** לכל התרגילים עד פרק 6.

רוב הדוגמאות אשר בספר מצויות באתר הוד-עמי ומחולקות בתיקיות הנושאות את שם פרקי הספר, כמו למשל CHAP01, CHAP02 וכד'. כך למשל הדוגמאות אשר בפרק 10 מופיעות בתיקיה CHAP10. אני ממליץ בחום לעיין בדוגמאות בתשומת לב וכמובן - להריץ אותן ולבחון את הפלט על פי קלטים שונים.

תקוותי, כי ספר זה יהיה לך לעזר ותפיק ממנו את המירב.

ברצוני להודות להוצאת הוד-עמי ולקוראים הנאמנים.

וכמובן, איך אוכל לא להודות לאשתי יעל אשר תמכה, עזרה, עודדה, וסבלה בשקט בעוד ביליתי שעות מול המחשב.

לימוד פורה

יואב נתיב

למשתמשי Visual C++

בהקצאות זיכרון דינמיות יש לכלול את הקובץ malloc.h ולא alloc.h עבור משתמשי Borland. כמו כן ישנה אפשרות לחפש את הקובץ malloc.h, להעתיקו וליצור קובץ זהה נוסף בשם alloc.h.

void main() או int main()

ספר זה משתמש בפונקציה main(), ללא ערך מוחזר: void main(). כאשר מתכנתים לומדים שפת תכנות חדשה, הגישה שלי היא ללמד את הדברים פשוט ככל שניתן, ולכן גם כתיבת main() ללא ערך מוחזר. עם זאת, למרות שהדבר עובד, ולא גורע מלימוד השפה לטעמי, יש מספר סיבות טובות מאוד מדוע יש להשתמש בערך מוחזר int ולכתוב את הפונקציה main() כמחזירה int ולא void. כמרצה, מפתח תוכנות, ומנהל מחלקת תוכנה אני חייב לציין סיבות אלו, בצד הסיבות שהכריעו בצד בחירה ב- void.

מדוע לא להשתמש בערך מוחזר void ולהשתמש במבנה int main()

1. על פי הסטנדרט יש להחזיר int.
2. ייתכן שפונקציית התיחול והעלייה – ה-startup, עשויה להניח שהערך המוחזר של הפונקציה main(), יידחף למחסנית. כאשר הפונקציה main() לא עושה כך, ייתכן שתיווצר בעיה על המחסנית, דבר אשר יגרום ליציאה לא סדירה מהתוכנית, ואפילו אף לקריסה.
3. כנראה שכאשר כותבים ערך מוחזר מסוג void חוזר ערך מוחזר אקראי למערכת ההפעלה, דבר אשר אנו מעדיפים להימנע ממנו.

מדוע אפשר להשתמש בערך מוחזר void

1. התלמידים, רובם ככולם, משתמשים במערכות הפעלה של Microsoft. על פי Microsoft, ניתן לכתוב את הפונקציה main עם ערך מוחזר void (למעשה ללא ערך מוחזר), ניתן לחפש את מילת המפתח main ב-msdn יולי 2001:

   "Alternatively, the **main** and **wmain** functions can be declared as returning **void** (no return value). If you declare **main** or **wmain** as returning **void**, you cannot return an exit code to the parent process or operating system using a **return** statement; to return an exit code when **main** or **wmain** are declared as **void**, you must use the **exit** function."

2. קל ופשוט יותר ללמוד את שפת התכנות C כאשר כותבים את הפונקציה main() עם ערך מוחזר void.
3. הסיבה העיקרית לשימוש בערך מוחזר מסוג int היא מטעמי ניידות של התוכניות, פורטביליות (portable). אם נהיה ריאליים, רובנו, בעיקר הלומדים, לא כותבים תוכניות אשר ישמשו אותנו באין ספור מערכות הפעלה וסביבות עבודה. כאשר ביום מן הימים נכתוב תוכנית ונרצה להפיץ אותה, נוכל לכתוב את ה-main() עם ערך מוחזר void.

חלק 1

תכנות בסיסי

1
היכרות ראשונית

אין דרך טובה יותר ללמוד שפת תכנות, מאשר להתחיל בדוגמה פשוטה. בדוגמה הבאה נראה תוכנית פשוטה בשפת C, אשר קולטת שני מספרים ומדפיסה את סכומם.

(שם התוכנית: firstexm.c).

הערה: המספרים בצד שמאל **אינם** חלק מהתוכנית, ונועדו להסברים בלבד. שים לב לשורות הגולשות. במסך תראה את הכתוב בשורה אחת. שים לב גם לרווחים שבין השורות. אלה נועדו לשפר את הקריאה, ואין לכך כל משמעות עבור שפת C. ועוד, בכל מקום שצריך, או אפשר לכתוב תו רווח, אפשר לכתוב יותר מרווח אחד.

```
1   #include <stdio.h>

2   void main()
    {
3       int first,second;
4       printf("please enter the first number: ");
5       scanf("%d", &first);
6       printf("please enter the second number: ");
7       scanf("%d", &second);
8       printf("The sum of the numbers is %d\n",
                                        first+second);
    }
```

כעת, לאחר שראינו דוגמה פשוטה, אפשר להתחיל בהסברים:

בראש התוכנית (בשורה 1) מופיעה השורה

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

בשפת C יש פקודות בסיסיות בלבד ולכן, כאשר ברצוננו לכלול פקודות נוספות, עלינו להשתמש בפקודה #include. המשמעות של פקודה זו, צירוף קובץ header - הכולל מספר פונקציות. בדוגמה זו, stdio.h הוא קובץ אשר מכיל פקודות קלט פלט סטנדרטיות (STDIO - standard input & output). בתוכנית זו ברצוננו לקלוט ולהדפיס

נתונים, ולכן צירפנו לתוכנית שכתבנו את הקובץ הזה. ראוי לציין שיש קבצי header רבים, כמו conio.h לטיפול במסך, או graphics.h הכולל פונקציות גרפיות.

```
void main()
```

(בשורה 2) היא הפונקציה הראשונה שהמחשב מריץ. זוהי למעשה **נקודת ההתחלה** של התוכנית. פונקציה זו **חייבת** להיות בכל תוכנית.

מייד לאחר שם הפונקציה ()main ניתן לראות סוגריים מסולסלים. הפקודות בתוכנית יבוצעו זו אחר זו מנקודת פתיחת הסוגריים ועד סגירתם. כל קטע קוד הנכתב בתוך סוגריים מסולסלים נקרא **block**. נשים לב שכל הוראה/פקודה מסתיימת בתו נקודה-פסיק (;). התבנית הכללית של הפונקציה ()main היא:

```
void main()
{
        הוראה ראשונה;
        הוראה שנייה;
        ....
        ....
        הוראה אחרונה;
}
```

שורה 3 היא:

```
int first,second;
```

הגדרת שני משתנים מסוג 'int'. משתנים מסוג זה משמשים לאחסון ערכים שלמים. בדוגמה זו הגדרנו שני משתנים, ואף קראנו לכל אחד מהם בשמות: לראשון - first, ולשני - second.

נביט על שורה 4:

```
printf("please enter the first number: ");
```

הפונקציה printf משמשת להדפסת טקסט על המסך. בדוגמה זו הדפסנו את הטקסט:

```
please enter the first number:
```

בשורה 5 נמצא את ההוראה

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

הפונקציה scanf משמשת לקליטת משתנים. המחרוזת "%d" מציינת שהמספר הנקלט הוא מספר מסוג int ואילו &first מציין שאת המספר יש לאחסן במשתנה first.

שורה 6 דומה מאוד לשורה 4, והמטרה זהה: **פלט**. הפעם מוצגת הודעה למשתמש להכניס את המספר השני.

שורה 7,

```
scanf("%d", &second)
```

מיועדת לקליטת מספר נוסף, הפעם לתוך המשתנה second.

בשורה 8 שוב יש שימוש בפונקציה printf, אך הפעם בשילוב של תוצאת ביטוי חשבוני:

```
printf("The sum of the numbers is %d\n", first+second);
```

הוראה זו מדפיסה את סכום שני המספרים. התווים '%d' מציינים שכאן יודפס ערך מספרי שלם. התווים '\n' מציינים מעבר לשורה חדשה.

בשלב זה כדאי להריץ את התוכנית ולאחר מכן להמשיך בלימוד.

משתנים

משתנה (variable) הינו תא בזיכרון, אשר בתוכו ניתן לאחסן ערך. במשתנים שונים אפשר לאחסן ערכים מסוגים שונים. לכל משתנה יש שם ייחודי וערך אחד מוגדר. בשפת C יש מספר סוגים של משתנים. לכל **סוג משתנה** (variable type) דרושה הקצאת זיכרון בגודל שונה. תפוסת הזיכרון נמדדת בבתים (bytes).

סוגי המשתנים הבסיסיים בשפת C הם: char, int, long, float, double.

char

המשתנה char תופס בית אחד בזיכרון. גם ניתן לאחסן בו ערכים מספריים שלמים מ-128- ועד 127+. יחוד משתנה זה בכך, שהוא משמש בדרך כלל לאחסון תווים או סימנים ולאו דווקא מספרים.

> **הערה:** ערכים שלמים הם מספרים ללא נקודה עשרונית, כמו 10, 452, 23- וכדומה.

תווים וסימנים הם למעשה כל האותיות, סימני הפיסוק או סימונים אחרים שניתן להפיק על ידי המחשב, למשל: a, r, T, Q, !, ?, ., *, { ועוד.

int

המשתנה int תופס שני בתים בזיכרון, וניתן לאחסן בו ערכים שלמים (int הינו קיצור של integer - שלם) בתחום 32768- עד 32767+.

> **הערה:** גודלו בבתים של משתנה מסוג int תלוי במערכת ההפעלה. במערכת הפעלה DOS, גודלו של int הוא 2 בתים. במערכות הפעלה Windows 95/NT, גודל משתנה מסוג int הוא 4 בתים, ממש כמו long (ראה בהמשך). בספר זה נתייחס לגודל int כשני בתים.

long

המשתנה long תופס ארבעה בתים בזיכרון, וניתן לאחסן בו ערכים שלמים בטווח גדול יותר מאשר ב-int. תחום המספרים המותר הוא 2147483648- עד 2147483647+ (הערה: ניתן גם לרשום 'long int' - הדבר זהה לרישום 'long').

float

המשתנה float תופס ארבעה בתים בזיכרון וניתן לאחסן בו ערכים ממשים עד 6 ספרות דיוק. השיטה בה מוחזקים המספרים נקראת 'נקודה צפה' (floating point, ומכאן שם המשתנה).

הערה: ערכים ממשיים הם ערכים לא שלמים ושלמים אשר ניתן לייצג אותם באמצעות נקודה עשרונית כמו 100.4, 3.1415, 55.1-, 100, 200, 4.0.

double

המשתנה double תופס שמונה בתים בזיכרון. משתנה זה משמש אף הוא לאחסון ערכים ממשיים בתחומים גדולים יותר, עד 10 ספרות דיוק.

הערה: קיים גם משתנה long double לאחסון ערכים ממשים יותר גדולים.

signed, unsigned

כל משנה מספרי שלם ניתן להגדיר כ- signed (עם סימן +/-) או unsigned (ללא סימן). signed פירושו אחסון מספרים שליליים וחיוביים, ואילו unsigned פירושו אחסון מספרים לא-שליליים בלבד (אפס, ומספרים חיוביים). כאשר מגדירים משתנה כ- unsigned, טווח הערכים החיוביים שלו גדל. לדוגמה, במשתנה מסוג **signed char** ניתן לאחסן ערכים חיוביים ושליליים בתחום 128- עד 127+ (סה"כ 256 ערכים שונים). במשתנה מסוג **unsigned char** ניתן לאחסן ערכים מ- 0 ועד 255 (סה"כ 256 ערכים שונים). כאשר לא נציין אם המשתנה הוא signed או unsigned, הוא יוגדר אוטומטית כ- signed. זוהי המשמעות של הביטוי **ברירת מחדל** (default option).

הערה: ברירת מחדל היא למעשה האפשרות שתבחר, אם לא בחרנו באחת מהאפשרויות. לדוגמה: במסעדה מסוימת, כאשר מזמינים סלט ולא מציינים איזה רוטב רוצים עם הסלט, נקבל רוטב אלף האיים. כאן ברירת המחדל ברטבים היא רוטב אלף האיים. דוגמה נוספת: כאשר מדליקים את הטלוויזיה, ולא בוחרים מספר תחנה מסוימת (מדליקים על ידי מקש on/off), ברירת המחדל היא תחנה 1, כלומר אם לא נציין תחנה מסוימת, הטלוויזיה תידלק על תחנה 1.

טבלת סיכום של סוגי המשתנים

סוג המשתנה	גודל בבתים	
char או signed char	1	128- עד 127+
unsigned char	1	0 עד 255
int או signed int	2	32768- עד 32767+
unsigned int	2	0 עד 65535
long או signed long	4	2147483648- עד 2147483647+
unsigned long	4	0 עד 4294967295+
float	4	3.4E-38 עד 3.4E+38
double	8	1.7E-308 עד 1.7+308
long double	16	3.4E-4932 עד 3.4E+4932

שמות חוקיים של משתנים

לכל משתנה חייב להיות שם ייחודי. כך נוכל לאחסן בתא זיכרון מסוים ערך, וגם נוכל לפנות אליו כדי לקרוא את הערך מתוך התא. בשתי פעולות אלו עלינו לגשת אל התא על ידי קריאה בשמו.

שם חוקי של משתנה מתחיל **באות** או **בקו תחתי**, אך מומלץ להתחיל באותיות בלבד. שאר תווי השם יכולים להיות להכיל אותיות, מספרים, וקו תחתי. אין לכלול סימנים נוספים כמו נקודה, פסיק, או רווח.

דוגמה לשמות חוקיים:

First_Num, Total_Grades, City95, This_name_is_fine

דוגמה לשמות לא חוקיים:

95Year	לא חוקי, כי מתחיל במספר
City!	לא חוקי, כי מכיל סימן קריאה
$I	מכיל תו אסור: '$'

בהגדרת המשתנים מומלץ להשתמש בשמות משמעותיים, כדי שהתוכנית תהיה קריאה ומובנת. למשל, כאשר נתייחס למשתנה בשם salary, סביר להניח שהוא מייצג משכורת של עובד, המשתנה grade עשוי להכיל ציון, והמשתנה City_name ישמש כנראה לאחסון שם של עיר. לעומת זאת, כאשר נקרא שמות משתנים כמו b1, holala, stam, aa15n או A_7, קשה יהיה לנחש למה הם משמשים.

שפת C רגישה לגודל אות - Case Sensitive - כלומר, שפה המבדילה בין אותיות גדולות וקטנות. על כן, המשתנה A שונה מהמשתנה a, והמשתנים TOTAL, total, TOtal, ו- TotaL הם למעשה **ארבעה** משתנים שונים.

מומלץ **לא** להשתמש במספר משתנים, אשר ההבדל ביניהם הוא אותיות גדולות או קטנות.

שם המשתנה מוגבל באורך מסוים, בדרך כלל עד 32 תווים. שם ארוך אינו שגיאה, אך אם אורך משתנים מוגבל ל- 32 תווים ונרשום משתנה באורך 40 תווים, המהדר לא יתייחס ל- 8 התווים האחרונים.

מקום הגדרת המשתנים

משתנים ניתן להגדיר מחוץ לפונקציות, או בתחילת block (לאחר פתיחת הסוגריים המסולסלים). בשלב ראשון נראה כיצד מגדירים משתנים בתוך block, ואחר כך נלמד על הגדרת משתנים מחוץ לפונקציות.

הגדרת משתנים בתוך בלוק (block)

הגדרה של משתנים בתוך בלוק יכולה להתבצע בתחילת הבלוק, **לפני** כל הוראה אחרת. משתנים אלה יוכרו אך ורק בתוך אותו בלוק. כאשר נגדיר משתנה, נרשום תחילה את **סוגו** (type) ולאחר מכן את **שמו** (name). בין סוג המשתנה לבין שמו חייב להופיע **לפחות רווח אחד:**

```
char ch;
int num;
float Worker_Salary;
```

שים לב לעניין **הרווחים**: בכל מקום שצריך, או אפשר לכתוב **רווח אחד**, אפשר לכתוב **מספר רווחים**.

כאשר יש מספר משתנים מאותו סוג, ניתן לרשום אותם בשורה אחת בצורה זו:

```
float Salary, Average_of_95, PI, Width;
int Items, Counter, Number_of_students;
```

כאשר נרצה לציין אם משתנה הוא signed או unsigned, נעשה זאת לפני סוג המשתנה. כמובן שבמקרה של signed הדבר מיותר, משום שזו ברירת המחדל.

```
unsigned int Age, I_am_not_negative, My_money;
unsigned char Ascii, code;
```

משתנים המוגדרים מחוץ לפונקציות נקראים גלובליים, ועליהם נלמד בפרק העוסק בפונקציות.

השמת ערכים ואתחול משתנים

כאשר מכריזים על משתנה חדש, ערכו **אינו** ידוע. הוא יכול להכיל ערך שנקבע לו בתוכנית קודמת, את הערך אפס, או כל ערך חוקי אחר. כדי לבצע **השמת/הקצאת** ערכים למשתנים (assignment), נשתמש בסימן = (שווה).

בכל מקום בתוכנית שבו המשתנים מוכָרים, נוכל לבצע השמת ערכים, ואף נוכל לאתחל משתנים בשורת ההגדרה. הנה דוגמה:

```
void main()
{
  int first=1, second=2, third;
  float salary, price=10.5;
  char nine='9';
  int a,b;
  a=5;
  b=17;
  salary=3050;
  third=55;
  ....
  ....
}
```

בדוגמה זו הגדרנו מספר משתנים מסוגים שונים. מתוך שלושת המשתנים מסוג int שהוגדרו בתחילת התוכנית אתחלנו רק את first ואת second. למשתנה third ביצענו השמה בשלב מאוחר יותר. בשורה השנייה הגדרנו שני משתנים מסוג float ורק ל-price קבענו ערך התחלתי.

ניתן לראות מדוגמה זו שבאפשרותנו לתת ערך למשתנה (השמה), גם בשורת ההגדרה וגם במהלך התוכנית. אנו גם יכולים לשנות את ערכי המשתנים שוב ושוב במהלך התוכנית.

הערה: מקור שמם של המשתנים הוא בכך שערכם יכול **להשתנות** במהלך התוכנית.

ביטויים - Expressions

ביטוי הינו סדרה של **אופרנדים** המחוברים על ידי **אופרטורים**, למשל:

```
Total_For_Today = amount * price * (100 + VAT) / 100;
```

אופרנדים (operands) הינם **משתנים** - variables (כמו למשל price, first, או salary), או **קבועים** - constants (כמו 927, 2.12135 או "Good Morning"). הקבועים הם ערכים שאינם משתנים במהלך ביצוע התוכנית.

אופרטורים (operators) יכולים להיות סימני פעולות חשבוניות (כמו כפל וחיבור), תנאים, פעולות על סיביות, ועוד.

כעת נעסוק באופרנדים ובאופרטורים. אז ניקח נשימה ארוכה וקדימה. . .

אופרנדים - Operands

אנו כבר יודעים שאופרנדים הינם משתנים או קבועים. אנו כבר מכירים את המשתנים, וכעת נעסוק בקבועים.

קבועים - Constants

בשפת C, כמו בשפות אחרות יש קבועים, דהיינו ערכים ידועים מראש. כולנו יודעים, למשל, מה ערך המספר 10. כל **קבוע** (constant) מיוחס לסוג משתנה מסוים. מספר יהיה int, אם הוא מספר שלם, long אם הוא מספר שלם גדול, double אם הוא ממשי (למשל 3.14), ו**מחרוזת** - אם הוא טקסט (תווים מסוגים שונים). נתבונן בדוגמאות:

דוגמה	סוג	הסבר לסוג הקבוע
age = 65;	int	מספר שלם
phone = 9613513;	long	גדול מדי עבור int
PI = 3.1415;	double	אינו מספר שלם
My_Name = "Yoav";	מחרוזת	אלה הם תווים שונים

הערה: בנושא המחרוזות עוד נדון בהמשך.

אפשר לאלץ קבוע להיות int, float, או כל סוג אחר, על ידי הוספת סיומת מתאימה:
L עבור long, F עבור float, D עבור double, U עבור unsigned.

כך למשל,

age = 65L	יהיה מסוג	Long	עקב הסיומת L
PI = 3.1415F	יהיה מסוג	float	עקב הסיומת F
tmp=100D	יהיה מסוג	double	עקב הסיומת D
a=30UL	יהיה מסוג	unsigned long	עקב הסיומת UL
b=50U	יהיה מסוג	unsigned int	עקב הסיומת U

הערה: לא כל המהדרים תומכים בסיומות אלה, התמיכה היא במהדרים החדשים.

מקרה מיוחד הינו קבוע מסוג char. קבוע מסוג char ייכתב בין שני תווי גרש בודדים, למשל:

```
ch_exmp = 'A';
```

משתנה זה הוא מסוג char וערכו יהיה 65, כמו ערך התו 'A' בטבלת ASCII (עיין בערכי ASCII שבנספח. שים לב לערכי אותיות רישיות וקטנות באנגלית).

טבלת ASCII וקוד ASCII: עבור כל תו או סימן מוגדר קוד מוסכם בינלאומי. ASCII - ראשי תיבות של American Standard for Coded Information Interchange - הקוד האמריקאי הסטנדרטי להחלפת מידע, הוא כינוי להגדרה בינלאומית עבור התווים במחשב. על פי ההגדרה, כל תו מיוצג על ידי בית בודד (8 סיביות) - 256 ערכים אפשריים, ולכן יש 256 תווים שונים בטבלת ASCII. ערך ASCII של התו 'A' הוא 65, ערך התו 'B' הוא 66 וערך התו '6' הוא 54.
הסיבה לשימוש בקוד ASCII היא בכך שהמחשב מאחסן תווים כערכים מספריים. לא ניתן לאחסן תווים ישירות בזיכרון המחשב, אלא מספרים בלבד.

אפשר לבצע השמה של משתנים מסוג אחד לסוג אחר, אך יש לשים לב שכאשר עושים זאת, עלולים לאבד לפעמים חלק מהמידע שנמצא בהם. למשל, כאשר נבצע השמה מ- float ל- int נאבד את כל החלק הלא-שלם של המספר.

long width = 30;	זוהי השמה מ- int ל- long.
int ASCII_char = 'C';	זוהי השמה מ- char ל- int.

קבוע מספרי הינו בדרך כלל בבסיס עשרוני, אך ניתן לרשום קבוע גם בבסיס אוקטלי (בסיס 8) או בבסיס הקסדצימלי (בסיס 16).

בסיסים: מספר הספרות בהן משתמשים לייצוג מספרים. בחיי היום-יום אנו משתמשים בייצוג מספרים בבסיס 10 (עשרוני, דצימלי) - סה"כ משתמשים ב- 10 ספרות (0 עד 9). ניתן לייצג גם מספרים בבסיס בינארי - שתי ספרות 0, ו- 1, בבסיס 8 (אוקטלי) - הספרות 0 עד 7, בבסיס 16 (הקסדצימלי) - הספרות 0 עד 9 והאותיות A, B, C, D, E ו-F, ובבסיסים נוספים.

מספרים שנכתבים כך 10, 232, 9782 הם מספרים בבסיס עשרוני; כאשר נוסיף לפני המספר 0 (הספרה אפס) הוא ייחשב כמספר בבסיס אוקטלי, וכאשר נוסיף לפני המספר 0x (אפס ו- x קטן) - לפנינו מספר בבסיס הקסדצימלי.

הנה כמה דוגמאות:

age = 0x41;	מספר לפי בסיס 16; ערכו בבסיס 10 הינו 65.
teen_age = 0x11;	מספר בבסיס 16; הערך לפי בסיס 10 הינו 17.
her_age = 025;	בסיס אוקטלי; ערכו בבסיס 10 הינו 21.
weight = 0120;	בסיס אוקטלי; ערכו בבסיס 10 הינו 80.

קבועים מסוג מחרוזת, או קבוע מסוג char המכיל תו יכול להיות מורכב משני תווים, כך שיצרו "תו מיוחד". התו '\' מציין שאחריו מופיע תו, והצירוף שלהם יחדיו מצביע על משמעות מיוחדת:

הצירוף	משמעות
\n	מציין מעבר לשורה חדשה (new line)
\r	בדיוק כמו הקשה על Enter (ערך 13 בטבלת ASCII)
\t	tab (התקדמות במספר תווים)
\v	tab אנכי
\b	BackSpace
\f	Form Feed
\0	Null Terminator (נלמד עליו בהמשך בנוגע למחרוזות)
\xN	מספר הקסדצימלי
\"	גרשיים
\'	גרש
\\	לוכסן הפוך אחד (\ )

והנה דוגמה לשימוש:

כאשר נרצה להדפיס את המילה 'hello' ומייד אחר כך לעבור לשורה חדשה, נשתמש בצירוף התווים \n או \r. בתוכנית נראה זאת כך:

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

אופרטורים - Operators

אופרטורים מאפשרים לבצע פעולות חיבור (+), חיסור (-), כפל (*), חילוק (/), ואף חישוב שארית (%). נכתוב למשל,

```
Sum = product_a + product_b;
```

כאן השתמשנו באופרטור החיבור: המשתנה Sum יכיל את סכום שני המשתנים product_a ו- product_b. ועוד דוגמה,

```
Total = Money - price;
```

חיסור: Total יכיל את ערך המשתנה Money פחות ערכו של price.

```
To_Pay = Price * 1.17;
```

כפל: המשתנה To_Pay יכיל את מכפלת המשתנה price ב- 1.17.

```
Average = sum / total;
```

חילוק: Average יכיל את תוצאת החילוק של sum ב- total.

```
Mdl = Num % 10;
```

שארית: Mdl יכיל את השארית של חלוקת ערך Num ב- 10.

ניתן לבנות ביטויים ארוכים ומורכבים ככל שנרצה, ואפשר אף לשלב סוגריים:

```
Total=(car_price+tape_price)*(100+tax)/100;
```

משתנה מסוים יכול להופיע מספר פעמים:

```
Price = prd + prd * 2
```

וניתן אף לבצע את הפעולה הבאה:

```
Counter = Counter + 10;
```

כאן פשוט "קידמנו" את המשתנה Counter ב- 10.

במקרים רבים נרצה לקדם משתנה על ידי הוספה של ערך כלשהו. בדרך כלל נרצה לעשות זאת בלולאות, בעת קידום מונה ב- 1, או בעת ספירת פריטים למשל. עקב ריבוי של פעולות אלו, קיימים בשפה קיצורים.

קיצורים

כאשר נרצה להוסיף ערך למשתנה מסוים, נוכל לעשות זאת בצורה הבאה:

```
Total = Total + 5;
```

אבל, כאשר ניתקל במצב בו שם המשתנה ארוך:

```
Total_Benefits_For_1996 = Total_Benefits_For_1996+1005;
```

ניתן לראות שמצב זה אינו נוח לכתיבה ולכן יש דרך קיצור: כדי להוסיף למשתנה ערך ניתן להשתמש בקיצורים אלה: =-, =+, =*, =/, =%.

וכך, במקום הביטוי הקודם יכולנו לכתוב:

```
Total_Benefits_For_1996 += 1005;
```

באותה דרך ניתן להשתמש בשאר הקיצורים. במקום הביטוי: `price = price * 1.17;` ניתן לכתוב:

```
price *= 1.17;
```

מקרה מיוחד הינו **הגדלה או הקטנה ב- 1**. ניתן להשתמש בקיצור לפעולת החיבור כמו שכבר ראינו, אך קיימים שני אופרטורים נפוצים מאוד לשם כך:

++ יקדם את ערך המשתנה ב- 1.

-- יחסר מהמשתנה את הערך 1.

על כן, כל הביטויים הבאים שקולים:

```
Total = Total + 1;
Total += 1;
Total++;
++Total;
```

עוד מספר מילים על ++ ועל --

את האופרטור ++ או את -- ניתן לכתוב **לפני** המשתנה, או **אחריו**, בהתאמה. יש הבדל משמעותי בין הפעולות. הדבר מבלבל מעט, אך ראוי לשים לב היטב למיקום האופרטור. כאשר נשתמש באופרטור **לפני** המשתנה, המשתנה יקודם תחילה, ואחר כך נוכל להשתמש בערכו החדש. וכאשר האופרטור יופיע **לאחר** המשתנה, קודם נשתמש בערך הקודם של המשתנה, ורק אחר כך ערכו יקודם או יוקטן.

הדוגמאות הבאות יבהירו זאת:

```
void main()
{
  int a =5 , b = 10, q = 8, p = 13;
  int c, d, e, f;
  c = a++;                /*
                          כאן האופרטור מופיע אחרי המשתנה a ולכן, קודם
                          תבוצע ההשמה ורק לאחר מכן יקודם המשתנה a.
                          בסוף הוראה זו ערך c יהיה 5 וערך a יהיה 6.
                                                  */
```

```
  d = ++b;                    /*
                                האופרטור מופיע לפני המשתנה b ולכן תחילה
                                יקודם המשתנה b ורק לאחר מכן תבוצע ההשמה.
                                בסיום הוראה זו ערך b יהיה 11 וכך גם ערך d.
                              */
  e = p--;                    /*
                                תחילה תבוצע ההשמה ולאחר מכן יחוסר 1 מ- p.
                                בסיום הוראה זו, ערך p יהיה 12 וערך e יהיה 13
                                (זהו בדיוק ערך p לפני ההשמה).
                              */
  f = --q;                    /*  /* ערכו של 'q' 7  וכך גם ערכו של 'f'
}
```

נסכם את הקיצורים בטבלה:

ביטוי	זהה ל-	ביטוי	זהה ל-
a = a + 10;	a += 10;	x = x * 1.5;	x *= 1.5;
a = a + 1;	a++;	t = t / 10;	t /= 10;
a = a - 10;	a -= 10;	f = f % 3;	f %= 3;
a = a - 1;	a--;		

אופרטורים נוספים

בשפת C קיימים אופרטורים נוספים. הסבר ודוגמאות מפורטות נראה בהמשך, אך הנה מספר דוגמאות פשוטות:

אופרטורי יחס:

גדול מ-	>	קטן או שווה ל-	=<
גדול או שווה ל-	>=	שווה ל-	==
קטן מ-	<	שונה מ-	=!

דוגמאות:

```
if (Age > 60)
        printf("Not accepted\n");

if (code != 1425)
        exit(1);
```

אופרטורים לוגיים

או	||
וגם	&&

דוגמאות:

```
if (age>10 && age<20)
        printf("Teenage\n");

if (height == 20 || age >30)
        accepted();
```

יש גם **אופרטורים הפועלים על סיביות**, כמו בדוגמה זו:

```
Right_Bit = ByteRead & 0x01;
```

הערה: אם אינך יודע מהו מספר בינארי ומהם פעולות על סיביות הינך יכול לדלג על חלק זה.

בעזרת האופרטורים הפועלים על סיביות ניתן לבצע את הפעולות הבאות:

- פעולת AND (וגם) באמצעות האופרטור &
- פעולת OR (או) באמצעות האופרטור |
- פעולת NOT (לא) באמצעות האופרטור ^

פעולות אלו יתבצעו על כל הסיביות של המשתנה. לדוגמה:

```
int a = 0x0F;
int b = 0x01;
int c = a & 0x01;
```

a	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
c	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

כאן ביצענו פעולת AND על כל הסיביות, סיבית אחר סיבית:

- פעולת AND על סיבית 0 של משתנה a וסיבית 0 של משתנה b
- פעולת AND על סיבית 0 של משתנה a וסיבית 1 של משתנה b
- פעולת AND על סיבית 0 של משתנה a וסיבית 2 של משתנה b

...

- פעולת AND על סיבית 0 של משתנה a וסיבית 15 של משתנה b

התוצאה אוחסנה במשתנה C.

דוגמה נוספת:

```
int a = 0xF1;
int b = ^a;
```

a	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1
b	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0

כאן פעולת NOT בוצעה על כל הסיביות של a והתוצאה אוחסנה במשתנה b. סיבית אשר ערכה היה 0 במשתנה a הפכה ל- 1 במשתנה b. סיבית אשר ערכה היה 1 במשתנה 1 הפכה ל- 0 במשתנה b. שים לב: ניתן גם להפעיל פעולת סיביות ולאחסן את התוצאה באותו משתנה:

```
x = ^x;
a = a|0x80;
```

ויש גם **ערכים בוליאניים**: אמת (true), שקר (false).

בשפת C אין סוג משתנה מיוחד לאחסון ערכים בוליאניים - ערכי אמת או שקר. משתמשים למטרה זו במשתנים או מספרים שלמים. הערך אפס (0) הוא שקר, וערך אחר כלשהו, פרט לערך אפס הינו אמת. לכן, נוכל לרשום את הביטוי הבא:

```
int fast = (speed > 100);
```

כאן, המשתנה fast יקבל ערך אמת (שונה מאפס) אם המשתנה speed גדול מ- 100. אחרת, ערכו שקר (אפס). נוכל להשתמש בערך בוליאני גם במשפטי תנאי (אשר נלמד מאוחר יותר), למשל:

```
if (fast) . . . . .
```

אם ערכו של fast 'אמת', אז . . .

דוגמה נוספת:

```
void main()
{
  int flg = 10;
  flg = !flg;              // המשתנה יקבל את הערך אפס (שקר)
  flg = !flg;              // המשתנה יקבל ערך אמת
}
```

הסבר:

בשורה ראשונה הכרזנו על משתנה בשם 'flg' ואתחלנו אותו בערך 10. כידוע, ערך זה שונה מאפס ולכן נקבל ערך "אמת". בשורה השנייה ביצענו השמה ל- 'flg'. האופרטור '!' מציין שלילה (not), ומשמעותו: "ההיפך". כלומר, אם ערך משתנה a הוא "אמת", אז 'a!' פירושו "שקר", ולכן 'flg' יקבל ערך "שקר" - כלומר אפס. בשורה האחרונה יש שוב שלילה של 'flg', והפעם נקבל ערך "אמת" (בדרך כלל זהו הערך 1).

כתיבת הערות

אנו יכולים להוסיף הערות בתוכנית, כפי שכבר ראינו בדוגמאות קודמות. הערות אלו ישמשו אותנו בלבד, ואת כל מי שיקרא את התוכנית שכתבנו. המהדר לא יתייחס להערות וידלג עליהם, אולם אנחנו נוכל להבין טוב יותר תוכניות, ולהיזכר מה עשינו בעת שנחזור לתוכניות שכתבנו, כדי לערוך בה שינויים.

הערה מתחילה ב- '*/' ומסתיימת ב- '/*'. לדוגמה:

```
printf("Hello\n");          /* printing Hello */
```

הערות יכולות להתפרס על מספר שורות:

```
/* The following program prints
   "Good Morning" and skips to the next line */
printf("Good Morning\n");
```

בשפת C זו הדרך היחידה לכתוב הערות, אך בשפת C++ קיימת דרך נוספת לרשום הערה המתפרסת על שורה אחת - על ידי צמד לוכסנים (//). הערה זו תימשך עד סוף השורה בלבד, והיא נתמכת גם על ידי רוב המהדרים של שפת C. לדוגמה:

```
printf("Shalom\n");        // printing "Shalom"
                           // this is a one line remark.
```

הגדרות מאקרו

ניתן להגדיר מאקרו על ידי שימוש בהוראה #define באופן הבא:

```
#define שם_המאקרו הגדרה
```

לדוגמה: `#define SIZE 50`

דבר זה יגרום להחליף כל מקום בתוכנית שבו מופיע SIZE בערך 50.

לדוגמה: `int a = SIZE;`

המהדר, רגע לפני הידור התוכנית, יחליף את SIZE בערך 50 ואז יהדר את התוכנית.

ניתן לכתוב הגדרות מאקרו הרבה יותר מורכבות, אך איני מוצא טעם להיכנס לנושא זה. הינך מוזמן לקרוא עוד על מאקרו בקבצי העזרה של המהדר.

מספר נקודות לסיכום הפרק

בפרק זה למדנו את הבסיס של השפה ואת סוגי המשתנים השונים.

1. על ידי הפקודה #include נכלול קבצי כותר (header files) אשר מכילים הצהרות על פונקציות. בין קבצים אלה נמצא את stdio.h, conio.h, graphics.h, dos.h ועוד.
2. void main() היא הפונקציה הראשונה אשר מופעלת בתוכנית.
3. כל הוראה/פקודה (כל statement) מסתיימת בתו נקודה-פסיק (;).
4. משתנה הינו תא בזיכרון אשר בתוכו ניתן לאחסן ערכים מסוגים שונים (ערך זה יכול להשתנות מספר פעמים במהלך התוכנית). ישנם משתנים מסוגים שונים עבור סוגי נתונים שונים.
5. משתנים מסוג char, int, ו- long מיועדים לאחסון ערכים שלמים. במשתנה char נאחסן בדרך כלל תווים.
6. משתנים מסוג float ו- double מיועדים לאחסון ערכים ממשיים. כל משתנה שלם ניתן להגדיר כ- signed או unsigned. המשמעות הינה - שימוש בערכים חיוביים ושליליים (signed), או שימוש בערכים גדולים או שווים לאפס בלבד (unsigned). ברירת המחדל הינה signed.

7. שם משתנה חוקי מתחיל באות או בקו תחתי. שאר התווים יכולים להיות אותיות, מספרים או קו תחתי.
8. הגדרת משתנה תתבצע על ידי רישום סוג המשתנה ומייד אחריו את שמו.
9. השמת ערכים למשתנים תתבצע על ידי הסימן שווה (=).
10. ביטוי הינו סדרה של אופרנדים המחוברים על ידי אופרטורים.
11. תווים מיוחדים מתחילים בתו לוכסן הפוך - '\'.
12. קיימים בשפה אופרטורים לקיצור פעולות, כמו =+, =-, =*, =/, =%, ++, --.
13. האופרטור ++ או -- יכולים להיכתב לפני או אחרי שם משתנה. במקרה הראשון, מבוצעת תחילה פעולת הקידום (או החיסור), ורק אז מחושב הביטוי כולו (עם המשתנה לאחר השינוי). במקרה השני, קודם מחושב הביטוי (עיין הערך הנוכחי של המשתנה) ורק אחר כך מבוצעת פעולת הקידום (או החיסור) שלו.
14. בשפת C אין משתנה להחזקת ערכים בוליאניים (אמת או שקר). לצורך כך משתמשים במשתנים המאחסנים ערכים שלמים. "אפס" מייצג מצב "שקר", וכל ערך שאינו אפס הוא "אמת".
15. כדי לכתוב הערה בגוף התוכנית, כותבים משני צידיה את התווים האלה: צמד התווים '*/' משמאל וצמד התווים '/*' מימין, או בשורה אחת מייד לאחר צמד הלוכסנים '//'.

תרגילים

1. איזה מבין שמות המשתנים הבאים חוקיים, איזה לא, ומדוע?

```
myname
my_name
who?
95's_total
no-problemo
this_is_test_#1
Total_for_january
_how?
no1
r5
```

2. עבור כל ערך קבע את סוג המשתנה המומלץ להשתמש בו:

   לדוגמה: עבור המספר 15555 נגדיר משתנה מסוג int.

   - מספר פריטים במחסן (0 עד 20000)
   - אחוז ריבית שנתית. לדוגמה, 0.33
   - התווים 'A' עד 'Z'
   - סה"כ סטודנטים באזור תל-אביב (0 עד 40000)
   - מס' ת"ז (לדוגמה, 2019572)
   - גיל של אדם (0 עד 130)
   - מונה אוכלוסיה (0 ומעלה)

3. מה ערכי המשתנים האלה price, amount, leftover, to_pay, x בסיום התוכנית:

```
void main()
{
  float p = 3.14, tax = 0.17;
  int price = 5, amount = 11, leftover;
  int to_pay, x;
  leftover = amount % 10;
  price++;
  to_pay = --price * amount;
  x = price * amount--;
}
```

מה ערכו של x בסיום התוכנית:

```
void main()
}
  int x = 100;
  x--;
  x = !x;
}
```

4. מה שגוי בתוכנית הבאה:

```
void main()
{
  float price = 10.5, total;
  total = price * amount;
}
```

מה צריכה להיות השורה האחרונה, כדי שבסיום התוכנית, יכיל הערך sum את סכום המשתנים a, b, c?

```
void main()
{
   int a,b;
   int c = 5  sum;
   a = 17;
   b = 32;
   ? ?  ?  ?  ? ?  ?
}
```

5. נתונה התוכנית הבאה:

```
#include <stdio.h>

void main()
{
   printf("hello ");
   printf("world\n");
}
```

האם הפלט יהיה

```
hello world
```

או שהפלט יהיה

```
hello
world
```