סדנת לימוד
C++

הוראות התקנה והפעלה של התקליטור
בנספח "התקליטור המצורף" ובקובץ ONCD בתקליטור

עורך ראשי: **זהר עמיהוד**

תרגום: **ערן מיק, אורי יאנובר, גל וינר - "יזמים צעירים ישראל"**

גלית איטקין

ייעוץ מקצועי: **יואב נתיב, שביט נחמן**

עריכה מקצועית: **תומר אחרק**

עריכה לשונית ועיצוב: **ענבל אילני, שרה עמיהוד**

עיצוב עטיפה: **ישראל מצגר**

שמות מסחריים

שמות המוצרים והשירותים המוזכרים בספר הינם שמות מסחריים רשומים של החברות שלהם. הוצאת הוד-עמי והוצאת SAMS עשו כמיטב יכולתן 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 והוצאת SAMS אינן אחראיות כלפי יחיד או ארגון עבור כל אובדן או נזק אשר ייגרם, אם ייגרם, מהמידע שבספר זה, או מהתקליטור המצורף.

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**

סדנת לימוד
C++

Jesse Liberty

Designed for Microsoft® Windows NT® Windows® 98

SAMS

הוצאת הוד-עמי
ספרי מחשבים

Teach Yourself C++, Third Ed.

By **Jesse Liberty**

Editor: **Z. Amihud**

Authorized translation from the English language edition, by Sams Publishing Copyright ©.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Hebrew language edition published by Hod-Ami Ltd. Copyright © 2001

©

כל הזכויות שמורות

הוצאת הוד-עמי

לספרי מחשבים בע"מ

ת.ד. 6108 הרצליה 46160

טלפון: 09-9564716 פקס: 09-9571582

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 2001

All Rights Reserved
HOD-AMI Ltd.
P.O.B. 6108, Herzliya
ISRAEL, 2001

מסת"ב 965-361-271-9 ISBN

תוכן עניינים מקוצר

הקדמה 25

פרק 1: הצעד הראשון 27

פרק 2: חלקי תוכנית C++ 45

פרק 3: משתנים וקבועים 57

פרק 4: משפטים וביטויים 79

פרק 5: פונקציות 110

פרק 6: מחלקות בסיסיות 147

פרק 7: רצף הפעולות בתוכניות 181

סיכום פרקים 1-7 213

פרק 8: מצביעים 221

פרק 9: ייחוסים (References) 255

פרק 10: פונקציות מתקדמות 289

פרק 11: הורשה 327

פרק 12: מערכים ורשימות מקושרות 361

פרק 13: פולימורפיזם 404

פרק 14: מחלקות ופונקציות מיוחדות 445

סיכום פרקים 8-14 475

פרק 15: הורשה מתקדמת 487

פרק 16: זרמים (Streams) 542

פרק 17: מרחבי שמות 585

פרק 18: ניתוח ועיצוב מוכווני עצמים 603

פרק 19: תבניות 643

פרק 20: חריגים וטיפול בשגיאות 693

פרק 21: מה הלאה 725

סיכום פרקים 15-21 766

נספח: התקליטור המצורף 779

נספח א: קדימות אופרטורים 784

נספח ב: מילות מפתח ב- C++ 786

נספח ג: בינארי והקסדצימלי 787

שים לב: נספח ד' והאינדקס הם באנגלית ולכן הם מופיעים משמאל לימין.

נספח ד: תשובות 1

אינדקס 73

תוכן העניינים

הקדמה 25

למי מיועד הספר 25
מוסכמות בספר 25
התקליטור המצורף 26

פרק 1: הצעד הראשון 27

ההיסטוריה של C++ בקצרה 27
תוכניות 28
פתרון בעיות 29
תכנות נוהלי, תכנות מובנה ותכנות מוכוון עצמים 29
C++ ותכנות מוכוון עצמים 31
כימוס 31
ירושה ומחזור 32
פולימורפיזם 32
כיצד התפתחה C++? 33
האם עלי ללמוד תחילה את שפת C? 33
C++ ו- Java 33
תקן ANSI 33
הכנות לתכנות 34
סביבת הפיתוח 35
הידור קובץ המקור 36
יצירת קובץ שניתן להפעלה עם מקשר 36
מחזור הפיתוח 36
HELLO.cpp – תוכנית C++ הראשונה שלך 37
TCLite ו- Visual C++ 6 40
בניית הפרויקט Hello World 40
שגיאות הידור 40
סיכום 41
שאלות ותשובות 42
סדנה 42
שאלון 42
תרגילים 43

פרק 2: חלקי תוכנית C++ 45

תוכנית פשוטה 45
מבט קצר על cout 47
הערות 50
סוגי הערות 50
השימוש בהערות 50
מילת אזהרה בנוגע להערות 51
פונקציות 52
השימוש בפונקציות 53
סיכום 55
שאלות ותשובות 55
סדנה 55
שאלון 56
תרגילים 56

פרק 3: משתנים וקבועים 57

מהו משתנה? 57
הקצאת זיכרון 58
גודל משתנים שלמים 58
signed ו-unsigned 60
סוגי משתנים בסיסיים 60
הגדרת משתנה 61
רגישות לגודל האותיות 62
מילות מפתח 63
יצירת יותר ממשתנה אחד באותה ההצהרה 64
הצבת ערכים במשתנים 64
typedef 66
מתי להשתמש ב-short ומתי להשתמש ב-long 67
גלישה במשתנה שלם ללא סימן 67
גלישה במשתנה שלם עם סימן 68
תווים 69
בין תווים למספרים 70
סימני הדפסה מיוחדים 70
קבועים 71
קבועים עובדתיים 71
קבועים מייצגים 71
הגדרת קבועים בעזרת #define 72
הגדרת קבועים בעזרת const 72
קבועים ממוספרים 73
סיכום 76
שאלות ותשובות 76

סדנה 77
שאלון 77
תרגילים 78

פרק 4: ביטויים ומשפטים 79

משפטים 80
שטחים ריקים (whitespaces) 80
בלוקים ומשפטים מורכבים 80
ביטויים 81
אופרטורים 82
אופרטור השמה 82
אופרטורים מתמטיים 83
חילוק של שלמים ושארית 84
שילוב של אופרטורי השמה ואופרטורים מתמטיים 85
הוספה והפחתה 86
תחילי וסופי 87
קדימות 88
קינון סוגריים 89
האמת לאמיתה 90
אופרטורי יחס 91
משפט if 92
סגנונות הזחה (Indentation) 95
הפסוקית else 96
משפטי if מתקדמים 98
שימוש בסוגריים מסולסלים במשפטי if מקוננים 99
אופרטורים לוגיים 102
AND לוגי – גם 102
OR לוגי – או 102
NOT לוגי – לא 103
קיצור דרך להערכת משפט לוגי 103
קדימות של אופרטורי יחס 103
עוד על "אמת" ועל "שקר" 104
אופרטור התניה (טרינארי) 105
סיכום 106
שאלות ותשובות 107
סדנה 107
שאלון 107
תרגילים 108

פרק 5: פונקציות 110

מהי פונקציה 110
ערכים מוחזרים, פרמטרים וארגומנטים 111
הכרזה על פונקציות והגדרתן 112
הכרזה על הפונקציה 112
אבטיפוסים של פונקציות 112
הגדרת פונקציה 114
ביצוע של פונקציות 116
משתנים מקומיים 116
משתנים גלובליים 119
משתנים גלובליים : מילת אזהרה 120
עוד על משתנים מקומיים 121
משפטי פונקציות 122
עוד על ארגומנטים של פונקציה 123
שימוש בפונקציות כפרמטרים של פונקציות 123
פרמטרים הם משתנים מקומיים 123
עוד על ערכים מוחזרים 125
פרמטרים של ברירת המחדל 127
העמסת פונקציות 130
נושאים מיוחדים בנושא פונקציות 132
פונקציות כלולות (Inline) 133
רקורסיה (Recursion) 135
כיצד פועלות הפונקציות - הצצה אל "מתחת למכסה המנוע" 139
רמות של הפשטה 140
מחיצות זיכרון 140
המחסנית והפונקציות 143
סיכום 144
שאלות ותשובות 144
סדנה 145
שאלון 145
תרגילים 146

פרק 6: מחלקות בסיסיות 147

יצירת טיפוסים חדשים 147
מדוע ליצור טיפוס חדש? 148
מחלקות ואיברים 148
הכרזה על מחלקה 149
מוסכמות לקביעת שמות 149
הגדרת אובייקטים 150
מחלקות לעומת אובייקטים 150

גישה לאיברי מחלקה 150
הצבה לאובייקטים, ולא למחלקות 151
אם לא תכריז, לא תמצא דבר במחלקה 151
פרטי כנגד ציבורי 152
הפוך את משתני המחלקה לפרטיים 154
פרטיות לעומת אבטחה 156
יישום פונקציות מחלקה 157
בנאים ומפרקים 160
בנאים ומפרקים – ברירת המחדל 161
כיצד להשתמש בבנאי ברירת המחדל 161
פונקציות מחלקה מסוג const 164
ממשק לעומת מימוש 165
היכן למקם את הכרזות המחלקה ואת הגדרות פונקציות המחלקה? 168
מימוש של פונקציה מובנית 169
מחלקות עם מחלקות אחרות בתור משתנים 172
מבנים (structures) 176
מדוע שתי מילות מפתח מבצעות את אותה הפעולה? 176
סיכום 176
שאלות ותשובות 177
סדנה 178
שאלון 178
תרגילים 179

פרק 7: רצף הפעולות בתוכניות 181

לולאות (Loops) 181
ההיסטוריה של הלולאה – goto 181
מדוע ראוי להימנע משימוש ב-goto 182
לולאות while 183
משפטי while מורכבים 185
המשפטים break ו-continue 186
לולאות while (true) 189
לולאות do...while 190
ביצוע do...while 191
לולאות for 193
לולאות for מתקדמות 195
ריבוי פעולות אתחול וקידום 196
משפטים ריקים (null statements) בלולאות for 196
לולאות for ריקות 198
לולאות מקוננות (Nested loops) 199
טווח הכרה (scoping) בלולאות for 201
סיכום לולאות 202

משפטי switch 204
שימוש במשפט switch עם תפריט 207
סיכום 210
שאלות ותשובות 211
סדנה 211
שאלון 211
תרגילים 212

סיכום פרקים 1-7 213

חזרה 213
פרקים 1-7 חלפו עברו 219

פרק 8: מצביעים 221

מהו מצביע? 221
אחסון הכתובת בתוך מצביע 225
שמות מצביעים 226
האופרטור "מצביע לכתובת זיכרון" 226
מצביעים, כתובות ומשתנים 227
פעולות על נתונים בעזרת מצביעים 228
בחינת הכתובת 229
מדוע להשתמש במצביעים? 231
המחסנית ומאגר הזיכרון החופשי 231
new – חדש 233
delete – מחיקה 233
דליפות זיכרון 235
יצירת אובייקטים במאגר הזיכרון החופשי 236
מחיקת אובייקטים 236
גישה לאיברי נתונים 238
איברי נתונים במאגר הזיכרון החופשי 239
המצביע this 242
מצביעים תועים, פראיים או מתנדנדים 244
מצביעי const 247
מצביעי const ופונקציות const 247
מצביעי this קבועים (const) 249
אריתמטיקה במצביעים 249
סיכום 252
שאלות ותשובות 253
סדנה 253
שאלון 253
תרגילים 254

פרק 9: ייחוסים (References) 255

מהו ייחוס (Reference)? 255
ייחוסים והאופרטור כתובת של (&) 257
לא ניתן להגדיר ייחוסים מחדש 258
עבור מה יוצרים ייחוס? 260
מצביעים ריקים וייחוסים ריקים 262
העברת ארגומנטים של פונקציות בעזרת ייחוס 262
כיצד לגרום ל-()swap לפעול עם מצביעים 264
מימוש ()swap בעזרת ייחוסים 265
כותרי פונקציות ואבטיפוסים - הרחבה 267
החזרת ערכים מרובים 268
החזרת ערכים בעזרת ייחוס 269
העברת ערכים בייחוס לשם היעילות 271
העברת מצביע const 274
ייחוסים כחלופה 277
מתי להשתמש בייחוסים ומתי במצביעים 279
שימוש בייחוסים ובמצביעים יחד 280
אל תחזיר ייחוס אל אובייקט שאינו בטווח ההכרה! 281
החזרת ייחוס לאובייקט שנמצא במאגר הזיכרון החופשי 283
בעלות על מצביעים 285
סיכום 286
שאלות ותשובות 287
סדנה 287
שאלון 287
תרגילים 288

פרק 10: פונקציות מתקדמות 289

העמסה של פונקציות חברות 289
ערכי ברירת מחדל 292
בחירה בין ערכי ברירת מחדל לבין העמסת יתר של פונקציות 294
בנאי ברירת המחדל 294
העמסה של בנאים 295
אתחול אובייקטים 296
בנאי העתקים 297
העמסה של אופרטורים 302
כתיבת פונקציית הוספה (Increment) 303
העמסה של האופרטור התחילי (prefix operator) 304
החזרת טיפוסים בפונקציות אופרטורים מועמסות 306
החזרת אובייקטים זמניים חסרי שם 307
שימוש במצביע this 309
העמסה של אופרטור ההוספה הסופי 310

ההבדל בין הוספה סופית לבין הוספה תחילית 310
אופרטור החיבור..... 313
העמסה של אופרטור החיבור..... 314
סוגיות שונות בהעמסה של אופרטורים..... 316
מגבלות על העמסת יתר של אופרטורים..... 316
מה להעמיס?..... 317
אופרטור ההצבה..... 317
אופרטורים ממירים (conversion operators)..... 319
אופרטורים ממירים..... 323
סיכום..... 324
שאלות ותשובות..... 324
סדנה..... 325
שאלון..... 325
תרגילים..... 326

פרק 11: הורשה 327

מהי הורשה?..... 327
הורשה וגזירה..... 328
ממלכת החי..... 329
התחביר של גזירה..... 329
פרטי או ציבורי..... 331
בנאים ומפרקים..... 333
העברת ארגומנטים לבנאי מחלקות הבסיס..... 336
דריסה של פונקציות (overriding)..... 340
הסתרת הפונקציה של מחלקת הבסיס..... 342
קריאה לפונקציה במחלקת הבסיס..... 344
פונקציות וירטואליות..... 345
כיצד פועלות הפונקציות הווירטואליות..... 350
אי אפשר להגיע מכאן לשם..... 351
פריסה (Slicing)..... 351
מפרקים וירטואליים..... 354
בנאי העתקים וירטואלי..... 354
מחיר הפונקציות הווירטואליות..... 357
סיכום..... 358
שאלות ותשובות..... 358
סדנה..... 359
שאלון..... 359
תרגילים..... 360

פרק 12: מערכים ורשימות מקושרות 361

מהו מערך? 361
איברי מערך 362
כתיבה מעבר לגבולות המערך 363
שגיאות עמוד הגדר 366
אתחול מערכים 367
הגדרת מערכים 368
מערכים של אובייקטים 369
מערכים רב-מימדיים 370
אתחול מערכים רב-מימדיים 371
על אודות הזיכרון 373
מערכים של מצביעים 374
הגדרת מערכים בזיכרון הפנוי 375
מצביע למערך לעומת מערך של מצביעים 376
מצביעים ושמות מערכים 376
מחיקת מערכים מהזיכרון הפנוי 378
מערכי תווים (char) 379
strcpy() ו- strncpy() 381
מחלקות מחרוזת 382
רשימות מקושרות ומבנים אחרים 389
בחינה של רשימה מקושרת 390
האצלת סמכויות 390
הרכיבים השונים במערכת 391
מה למדת עד כה? 400
מחלקות מערך 400
סיכום 401
שאלות ותשובות 402
סדנה 402
שאלון 402
תרגילים 403

פרק 13: פולימורפיזם 404

בעיות עם הורשה יחידה 404
פעפוע כלפי מעלה 407
המרה כלפי מטה 407
הוספה לשתי רשימות 410
הורשה מרובה 411
רכיבי אובייקט שנוצר בהורשה מרובה 414
פונקציות בנייה באובייקטים מרובי הורשה 414
פתרון דו-משמעויות 417
הורשה ממחלקת בסיס משותפת 418

הורשה וירטואלית..... 422
בעיות בהורשה מרובה..... 426
מחלקות מסוג Mixin או Capability..... 427
סוגי נתונים מופשטים..... 427
פונקציות וירטואליות טהורות..... 431
מימוש פונקציות וירטואליות טהורות..... 432
היררכיות מורכבות של מופשטות..... 436
איזה סוגים הם מופשטים?..... 440
תבנית המשקיף..... 440
הורשה מרובה, סוגי נתונים מופשטים ו-Java..... 441
סיכום..... 442
שאלות ותשובות..... 442
סדנה..... 443
שאלון..... 443
תרגילים..... 444

פרק 14: מחלקות ופונקציות מיוחדות 445

נתוני מחלקה סטטיים..... 445
פונקציות מחלקה סטטיות..... 451
מצביעים אל פונקציות..... 453
למה להשתמש במצביעים לפונקציות?..... 457
מערכים של מצביעים לפונקציות..... 460
העברת מצביעים לפונקציות אל פונקציות אחרות..... 462
שימוש ב-typedef עם מצביעים לפונקציות..... 465
מצביעים לפונקציות מחלקה..... 467
מערכים של מצביעים לפונקציות מחלקה..... 470
סיכום..... 472
שאלות ותשובות..... 473
סדנה..... 473
שאלון..... 473
תרגילים..... 474

סיכום פרקים 8-14 475

חזרה..... 475
פרקים 8-14 חלפו עברו..... 486

פרק 15: הורשה מתקדמת..... 487

הכלה (Containment)..... 487
גישה אל רכיבים במחלקה המוכלת..... 494
סינון גישה לרכיבים מוכלים..... 494
עלות ההכלה..... 494

העתקה לפי ערך 497
מימוש במונחים של הורשה / הכלה לעומת האצלה 501
האצלה (Delegation) 502
הורשה פרטית 511
מחלקות חברות 520
פונקציות חברות 529
פונקציות חברות והעמסת אופרטורים 529
העמסת אופרטור ההוספה 534
סיכום 539
שאלות ותשובות 539
סדנה 540
שאלון 540
תרגילים 540

פרק 16: זרמים (Streams) 542

סקירת הנושא 542
כימוס (Encapsulation) 543
חציצה (Buffering) 543
זרמים וחוצצים 547
אובייקטי I/O תקניים 547
ניתוב מחדש 547
קלט באמצעות cin 548
מחרוזות 550
בעיות במחרוזות 550
מחזיר ייחוס אל אובייקט istream operator>> 553
פונקציות מחלקה אחרות של cin 553
קלט של תו יחיד 553
שימוש ב-()get ללא פרמטרים 553
שימוש ב-()get עם פרמטר ייחוס אל תו 555
קבלת מחרוזות מקלט סטנדרטי 556
שימוש ב- ()cin.ignore 558
()peek ו- ()putback 560
פלט עם cout 561
שטיפת הפלט 561
פונקציות מיוחסות 561
מניפולטורים, דגלים והוראות עיצוב 563
שימוש ב- ()cout.width 564
הגדרת תווי המילוי 565
הגדרת דגלים 565
זרמים לעומת הפונקציה ()printf 567
קלט ופלט לקובץ 571

ofstream 571
מצבי תנאי 572
פתיחת קבצים לקלט ופלט 572
שינוי התנהגות ברירת המחדל של ofstream בפתיחת קובץ 574
בינארי לעומת קבצי טקסט 576
עיבוד שורת פקודה 578
סיכום 582
שאלות ותשובות 582
סדנה 583
שאלון 583
תרגילים 584

פרק 17: מרחבי שמות 585

התחלת העבודה 585
פונקציות ומחלקות מזוהות לפי שם 586
יצירת מרחב שמות 589
הכרזה והגדרת סוגים 590
הגדרת פונקציות מחוץ למרחב שמות 591
הוספת רכיבים חדשים 591
קינון מרחבי שמות 592
שימוש במרחב שמות 592
מילת המפתח using 594
ההנחיה using 594
ההכרזה using 596
כינוי של מרחב שמות 598
מרחב שמות ללא שם 598
מרחב השמות הסטנדרטי std 599
סיכום 600
שאלות ותשובות 601
סדנה 602
שאלון 602
תרגילים 602

פרק 18: ניתוח ועיצוב מוכווני עצמים 603

האם C++ היא שפה מוכוונת עצמים? 603
בניית מודלים 604
תכנון תוכנה: Modeling Langauge 605
תכנון תוכנה: התהליך 606
חזון (Vision) 609
ניתוח דרישות 609

use-cases..... 609
זהה את השחקנים..... 610
זיהוי ה-use-cases הראשונים..... 611
יצירת דגם התחום..... 612
זיהוי תרחישים..... 616
זיהוי קווים מנחים..... 617
יצירת חבילות..... 620
ניתוח יישום..... 620
ניתוח מערכות..... 620
מסמכי תכנון..... 621
Visualizations..... 622
מסמכים וארטיפקטים..... 622
תכנון..... 623
מהן המחלקות?..... 623
טרנספורמציות..... 625
טרנספורמציות אחרות..... 626
דגם סטטי (Static Model)..... 627
כרטיסי CRC..... 627
קשרים בין מחלקות..... 630
דגם דינמי..... 637
שרטוטי מעבר מצב..... 638
סיכום..... 640
שאלות ותשובות..... 641
סדנה..... 641
שאלון..... 641
תרגילים..... 642

פרק 19: תבניות..... 643

מהן תבניות?..... 643
סוגים עם פרמטרים..... 644
מימוש מופע של תבנית..... 644
הגדרת תבנית..... 644
שימוש בשם..... 646
מימוש התבנית..... 647
פונקציות תבנית..... 650
תבניות וחברים..... 651
מחלקות ופונקציות חברות שאינן תבנית..... 651
מחלקה או פונקציה חברה שהיא תבנית כללית..... 655
שימוש בפריטי תבנית..... 658
פונקציות מיוחדות..... 663
תבניות ורכיבים סטטיים..... 668

ספריית התבניות הסטנדרטית STL..... 672
מכולות..... 673
הבנת מחלקות רצף..... 673
המכולה וקטור..... 673
המכולה רשימה..... 680
המכולה Deque (תור דו קצווי)..... 681
מחסניות..... 681
הבנת תורים..... 682
הבנת מכולות אסוציאטיביות..... 683
המכולה מפה..... 683
מכולות אסוציאטיביות אחרות..... 686
מחלקות אלגוריתם..... 687
פעולות רצף לא-משתנה..... 688
אלגוריתמים לשינוי רצף..... 689
סיכום..... 690
שאלות ותשובות..... 690
סדנה..... 691
שאלון..... 691
תרגילים..... 691

פרק 20: חריגים וטיפול בשגיאות 693

שגיאות, תקלות וטעויות..... 694
חריגים..... 695
מילה אודות ריקבון קוד (Code Rot)..... 695
חריגים..... 696
כיצד משתמשים בחריגים..... 696
שימוש בבלוקים מסוג try ו-catch..... 701
תפיסת חריגים..... 702
מספר משפטי catch..... 702
היררכיות חריגים..... 705
נתונים בחריגים ומתן שמות לאובייקטי חריגים..... 708
חריגים ותבניות..... 715
חריגים ללא שגיאות..... 718
שגיאות ותוכניות ניפוי..... 719
נקודות שבירה..... 720
נקודות תצפית..... 720
התבוננות בזיכרון..... 720
אסמבלר..... 720
סיכום..... 720
שאלות ותשובות..... 721

סדנה 722
מבחן 722
תרגילים 722

פרק 21: מה הלאה 725

קדם המעבד והמהדר 726
הצגת קובץ הביניים 726
שימוש ב- define# 726
שימוש ב- define# עבור קבועים 727
שימוש ב- define# לבדיקות 727
פקודת else# בקדם המעבד 727
הוספה ומגיני הוספה 729
פונקציות מאקרו 730
בשביל מה כל הסוגריים? 731
מאקרו לעומת פונקציות ותבניות 733
פונקציות בשורה 733
טיפול במחרוזות 734
החרזה (Stringizing) 735
שרשור (Concatenation) 735
פונקציות מאקרו מוגדרות מראש 736
assert() 736
ניפוי עם assert() 738
assert() לעומת חריגים 738
תופעות לוואי 739
Class Invariants 739
הדפסת ערכים זמניים 744
רמות ניפוי 746
תמרון סיביות (Bit Twiddling) 752
אופרטור AND 753
אופרטור OR 753
אופרטור Exclusive OR 753
האופרטור המשלים (Complement) 753
הגדרת סיביות 753
ניקוי סיביות 754
הפיכת סיביות 754
שדות סיביות 754
סגנון 758
כניסות 758
סוגריים 758
שורות ארוכות 759
משפטי switch 759

טקסט תוכנית..... 759
שמות מזהים..... 760
איות ואותיות רישיות של שמות 760
הערות..... 761
גישה 762
הגדרות מחלקה..... 762
קבצי include..... 762
assert()..... 763
const..... 763
סיכום..... 763
שאלות ותשובות..... 764
סדנה..... 765
שאלון..... 765
תרגילים 765

סיכום פרקים 15-21..... 766

חזרה..... 766

נספח: התקליטור המצורף 779
הפעלת התקליטור 780
קטלוג ספרים 780
קוד מקור 781
התקנת תוכנות 781
התוכנות המומלצות להתקנה 781
Adobe Acrobat Reader 781
שאלות נפוצות 782
הערה חשובה! 782
צור קשר 782

נספח א: קדימות אופרטורים 784

נספח ב: מילות מפתח ב- C++ 786

נספח ג: בינארי והקסדצימלי 787
בסיסים אחרים 788
טיול בין הבסיסים 788
בינארי 790
מדוע בסיס 2? 790
סיביות, בתים ו-Nybbles 791
מהו KB? 791
מספרים בינאריים 791
הקסדצימלי 792

שים לב: נספח ד' והאינדקס הם באנגלית ולכן הם מופיעים משמאל לימין.

נספח ד: תשובות 1

אינדקס 73

הקדמה

למד בעצמך כיצד לתכנת ב- C++ בעזרת ספר זה. ב- 21 פרקים תלמד את היסודות כמו קלט/פלט (I/O), לולאות (loops) ומערכים (arrays), תכנות מוכוון עצמים (Object Oriented Programming), תבניות (templates), ובניית יישומי C++ - כל זאת במבנה נכון וקל למעקב של פרקים.

בכל פרק תמצא קטעי קוד לדוגמה, פלט והסבר לקוד העוסקים בנושא המדובר.

כדי להפוך אותך ל"מומחה" - בסוף כל פרק תמצא שאלות ותשובות, תרגילים ושאלון. תוכל לבחון את עצמך בעזרת התשובות לתרגילים ולשאלון שנמצאים בנספח ד'.

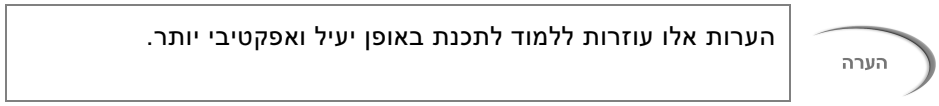
למי מיועד הספר

הספר מיועד לכל מי שרוצה להתחיל לתכנת בשפת C++, ואין לו ניסיון קודם.

ספר זה מלמד את השפה ואת עקרונות התכנות בשפת C++. בעזרת מספר גדול של דוגמאות תחביר והסברים מפורטים לקוד תוכל ללמוד את השפה ביתר קלות.

בין אם אתה מתחיל ובין אם יש לך ניסיון כלשהו בתכנות - תמצא את הספר קל ונוח ומהיר ללימוד.

מוסכמות בספר

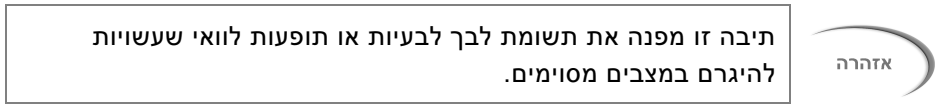


הערות אלו עוזרות ללמוד לתכנת באופן יעיל ואפקטיבי יותר.

FAQ

מה זה?

תשובה: שאלות אלו מספקות מבט מעמיק יותר בשימוש השפה ומספקות הבהרה לדברים שיכולים לגרום בלבול. FAQ = שאלה נפוצה.



תיבה זו מפנה את תשומת לבך לבעיות או תופעות לוואי שעשויות להיגרם במצבים מסוימים.

תיבה זו מספקת הגדרות מדויקות למונחים חיוניים.

עשה	אל תעשה
ראה טבלה זו לסיכום קצר של עיקרון בסיסי בפרק.	אל תדלג על טבלה שימושית זו.

תדפיסי הקוד מלווים במספרי שורות. אם אין מספר שורה - זוהי המשך שורה קודמת ממוספרת (יש שורות רחבות יותר מרוחב הספר). אתה צריך להקליד את שתי השורות כשורה אחת ולא לחלק ביניהן.

התקליטור המצורף

התיקיות הרלוונטיות לספר זה הן:

TCLite\Software שמכילה את תוכנת TCLite שבעזרתה תוכל להריץ את רוב התוכניות שבספר. תוכניות שרצות רק ב- Visual C++ סומנו בספר בתחילת התדפיס.

59302\Books שמכילה את התוכניות שבספר בתיקיות לפי מספרי הפרקים.

פנה לנספח **התקליטור המצורף** ולקובץ **ONCD** שבתקליטור, לפירוט על התוכנות בתקליטור וכיצד להתקין אותן.

הערה לגבי תרגום מונחים

"עברית שפה קשה" ובמחשבים עוד יותר. אנשים שונים מתרגמים מונחים באנגלית בצורה שונה, במקומות שונים מקובל מונח שונה ולעיתים מונח כלשהו באנגלית יכול לקבל 2 משמעויות ויותר בעברית. למשל, תמצא שתרגמנו את המילה method לשיטה וגם לפונקציה, OOP הוא תכנות מוכוון עצמים אבל object הוא אובייקט וכך הלאה. תמיד בדוק כיצד כתוב המונח באנגלית.

לא נחסך כל מאמץ להבטחת הדיוק של ספר זה והתקליטור המצורף אליו. אם יש לך הערות, שאלות או רעיונות הנוגעים לספר זה של ההוצאה ולתקליטור המצורף, אנא שלח אותם להוצאת הוד-עמי באחת השיטות הבאות:

דואר אלקטרוני: support@hod-ami.co.il בשורת Subject ציין את המספר 59302

דואר רגיל:

הוצאת הוד-עמי לספרי מחשבים

ת.ד. 6108

הרצליה 46160

הצעד הראשון

ברוך הבא ל- "סדנת לימוד ++C"! היום תתחיל בדרכך להיות מתכנת ++C מיומן. בפרק זה תלמד:

- מדוע ++C היא התקן המקובל לפיתוח תוכנה.
- השלבים לפיתוח תוכנית ++C.
- כיצד להיכנס לתוכנית ++C הראשונה שלך, כיצד ל**הדר** (compile) ול**קשר** (link) אותה.

ההיסטוריה של ++C בקצרה

שפות מחשבים עברו התפתחות דרמטית מאז שנבנו המחשבים האלקטרוניים הראשונים, כדי לסייע בחישוב מסלולים של קליעי תותח במהלך מלחמת העולם השנייה. בתחילה השתמשו המתכנתים בהוראות המחשב הבסיסיות ביותר, בשפת מכונה (machine language). הוראות אלו יוצגו על ידי מחרוזות ארוכות של ספרות 'אחת' ו-'אפס'. במהרה, פותחו שפות סף, או אסמבלי (assemblers) כדי למפות את הוראות המכונה לביטויים שקל יהיה לזכור ולקרוא אותם, וגם יהיו ניתנים לניהול, כמו ADD ו- MOV.

עם הזמן פותחו שפות ברמה יותר גבוהה, כמו BASIC ו- COBOL. שפות אלו אפשרו לכתוב את פקודות המחשב במילים ובמשפטים שיש להם דמיון לשפה מדוברת, או לנוסחאות מתמטיות, כמו LET I=100. הוראות אלו תורגמו לשפת מכונה על ידי **מפרשים** (interpreters) ו**מהדרים** (compilers). 'מפרש' מתרגם את התוכנית במהלך קריאתה והופך את הוראות התוכנית, או את הקוד שלה, לפעולות שהמחשב מבצע מייד. 'מהדר' מתרגם את הקוד לפקודות במבנה ביניים בתהליך **הידור** (compiling) שיוצר קובץ **יעד** (object). המהדר קורא ל**מקשר** (linker) והופך את קובץ היעד לתוכנית **בת-ביצוע** (executable).

מכיון שמפרשים קוראים את הקוד כפי שהוא כתוב ומבצעים אותו מייד, קל למתכנת לעבוד עם מפרשים. מהדרים מציגים את הצעדים הנוספים הלא נוחים של הידור וקישור הקוד. עם זאת, מהדרים מייצרים תוכנית מוכנה שאין צורך לפענח ולפרש אותה מחדש בכל הרצה (Run), ולכן הפעלתה מהירה יותר. תרגום **קוד המקור** (source code) לשפת מכונה כבר בוצע, ואין צורך לחזור עליו.

יתרון נוסף של שפות מהודרות רבות כמו C++ הוא, שניתן להפיץ את התוכנית בת-הביצוע לאנשים שאין להם מהדר. בשפה של מפרש, חייב המפרש להיות מותקן במחשב, כדי שניתן יהיה להריץ בו את התוכנית.

שפות מסוימות, כמו Visual Basic, קוראות למפרש Run Time Library (ספריית זמן ריצה). Java קוראת למפרש זמן הריצה שלה Virtual Machine (VM), אבל במקרה זה המפרש הוא חלק מהדפדפן (browser), כמו Internet Explorer או Netscape.

במשך שנים רבות המטרה העיקרית של מתכנתי מחשבים היתה לכתוב קטעי קוד קצרים שיבוצעו במהירות. התוכנית היתה צריכה להיות קטנה, מכיון שזיכרון היה יקר, והיא היתה צריכה להיות מהירה, מכיון שהמעבדים היו איטיים וגם יקרים. ככל שהמחשבים נעשו קטנים יותר, זולים יותר ומהירים יותר, וככל שמחיר הזיכרון והמעבדים ירד, סדרי העדיפויות הללו השתנו. כיום מחיר הזמן של המתכנת יקר בהרבה ממחירם של רוב המחשבים שבשימוש עסקי. הביקוש כיום הוא לקוד כתוב היטב וקל לתחזוקה. 'קל לתחזוקה', משמעו שכאשר הדרישות של העסק משתנות, ניתן לשנות את התוכנית ללא הוצאה גדולה.

תוכניות

משתמשים במילה **תוכנית** (program) בשתי משמעויות: כדי לתאר הוראות פרטניות (או קוד מקור) שנוצרו על ידי המתכנת, וכדי לתאר קטע קוד שלם של תוכנית בת-ביצוע. הבחנה זו יכולה לגרום לבלבול עצום. על כן, ננסה להבדיל בין קוד מקור מצד אחד, ותוכנית בת-ביצוע מצד שני.

תוכנית יכולה להיות מוגדרת הן כסדרה של הוראות כתובות שנוצרות על ידי המתכנת והן כקטע של תוכנית בת-ביצוע.

קוד מקור יכול להיות מוסב לתוכנית בת-ביצוע בשתי דרכים: **מפרשים** (interpreters) מתרגמים את קוד המקור להוראות מחשב, והמחשב פועל בהתאם להוראות אלו מייד. לחילופין, **מהדרים** (compilers) מתרגמים את קוד המקור לתוכנית, שניתן להריץ אותה במועד מאוחר יותר. למרות שקל יותר לעבוד עם מפרשים, רוב התכנות מתבצע עם מהדרים, מכיון שקוד מהודר רץ הרבה יותר מהר ואפשר להפעיל עליו תהליכי מיטוב (אופטימיזציה). C++ היא שפה שעוברת הידור.

פתרון בעיות

הבעיות שמתכנתים התבקשו לפתור השתנו. לפני עשרים שנה, תוכניות נוצרו כדי לנהל כמויות גדולות של נתונים גולמיים. האנשים שכתבו את הקוד והאנשים שהשתמשו בתוכנית היו כולם מקצועני מחשבים. כיום, רבים יותר משתמשים במחשבים ורובם יודעים מעט מאוד על אופן פעולתם של מחשבים ותוכניות, וגם אינם מתעניינים בכך. מחשבים הם כלים בשימושם של אנשים שיותר מתעניינים ב**פתרון הבעיות העסקיות** שלהם מאשר במאבק עם המחשב.

למרבה האירוניה, כדי להפוך את התוכניות לקלות יותר לשימוש עבור הקהל החדש הזה, התוכניות הפכו הרבה יותר מתוחכמות. חלפו הימים שבהם המשתמשים הקלידו פקודות בקודים שרק הם יכלו לפענח. התוכניות של היום משתמשות ב"ממשק משתמש גרפי" (GUI) מתוחכם הכולל חלונות מרובים, תפריטים, תיבות דו-שיח (dialog boxes) ומספר רב של אמצעים שרבים כבר התוודעו אליהם. התוכניות שנכתבו כדי לתמוך בגישה חדשה זו הרבה יותר מורכבות מאלו שנכתבו לפני עשר שנים ויותר.

עם ההתפתחות של רשת האינטרנט (Web), המחשבים נכנסו לעידן חדש של חדירה לשוק; יותר אנשים משתמשים במחשבים מאי פעם והציפיות שלהם מאוד גבוהות. התוכניות הפכו גדולות ומורכבות יותר, והצורך בשיטות **תכנות מוכוון עצמים** (object oriented programming) כדי לפצח מורכבות זו הפך לברור וגלוי.

כאשר דרישות התכנות השתנו, השפות והשיטות לכתיבת התוכניות התפתחו במקביל. למרות שההיסטוריה השלמה היא מרתקת כשלעצמה, ספר זה יתמקד בעיקרי השוני בין **תכנות נוהלי** (procedural programming) ל**תכנות מוכוון עצמים.**

תכנות נוהלי, תכנות מובנה ותכנות מוכוון עצמים

עד לא מזמן, המתכנתים ראו את התוכנית כסדרה של הליכים שמבוצעים על נתונים. **הליך** (procedure), או פונקציה, הוא סדרה של הוראות מפורטות המבוצעות זו אחר זו. הנתונים היו נפרדים מההליכים, ובמהלך התכנות צריך היה לעקוב אחרי הקריאות של פונקציה אחת לאחרת, על איזה נתונים היא פועלת ואיזה נתונים עודכנו.

כדי להבין את המצב שיכול לבלבל, נוצר **תכנות מובנה** (structured programming).

הרעיון העיקרי שמאחורי תכנות מובנה פשוט כמו הרעיון של "הפרד ומשול". תוכנית מחשב יכולה להיחשב כמכילה סדרה של משימות. כל משימה אשר מורכבת מדי מכדי להיות מתוארת בפשטות, תפורק לסדרה של רכיבי משימות קטנים יותר, עד שהמשימות יהיו מספיק קטנות ומאורגנות כיחידות נפרדות, ושלמות כך שהם יובנו בקלות.

דוגמה: חישוב של משכורת ממוצעת של כל עובד בחברה הוא משימה די מורכבת. אולם, ניתן לפרק אותה לתת המשימות הבאות:

1. מצא כמה מרוויח כל עובד.
2. ספור כמה אנשים ישנם.
3. חשב את סכום כל המשכורות.
4. חלק את הסכום הכללי במספר האנשים שישנם.

סיכום המשכורות לכל העובדים יכול להיות מפורק לשלבים אלה:

1. השג את הרישום על העובד.
2. גש למשכורת.
3. הוסף את המשכורת לסכום הכללי המצטבר.
4. השג את הרישום על העובד הבא.

קבלת נתוני כל עובד יכולה להתפרק לשלבים אלה:

1. פתח את הקובץ של העובדים.
2. גש לרשומה המתאימה.
3. קרא את הנתונים מהדיסק.

תכנות מובנה הינו גישה טובה לטיפול בבעיות מורכבות. אולם, בשנות השמונים המאוחרות התבררו כמה מהחסרונות של שיטות התכנות המובנה.

ראשית, נטייה טבעית היא לחשוב על נתונים (לדוגמה, רישומים על כל עובד) ומה שניתן לעשות אתם (למיין, לערוך וכו') כרעיון אחד. תכנות מובנה נוגד את הנטייה הזו בכך שהוא מפריד את מבני הנתונים מהפונקציות שמטפלות בהם.

שנית, תוכניתנים נוכחו לדעת שהם ממציאים מחדש פתרונות חדשים לבעיות ישנות בקביעות. מה שנקרא "להמציא את הגלגל מחדש", שהוא ההפך מִמְחָזור עבודות קודמות לשימושים חדשים. הרעיון שמאחורי מִחָזור הוא לבנות רכיבים שיש להם תכונות ידועות, שניתן אחר כך לחבר אותם לתוכנית ככל הנדרש. הרעיון נלקח מעולם החומרה – כאשר מהנדס זקוק לטרנזיסטור חדש, הוא אינו ממציא או מפתח אותו, אלא הולך לתיבה הגדולה של הטרנזיסטורים ומוצא את המתאים לו ביותר לפי הצרכים שלו, או אולי מתאים אותו לצרכיו. עבור מהנדס תוכנה לא היתה קיימת אפשרות כזו.

הדרך בה אנו משתמשים כיום במחשבים – עם תפריטים,לחצנים וחלונות – מאמצת גישה יותר אינטראקטיבית ומונחית אירועים (event-driven) לתכנות מחשבים. "מונחית אירועים", משמע שכאשר מתרחש אירוע, המשתמש לוחץ על לחצן או בוחר מתפריט – והתוכנית חייבת להגיב. תוכניות נעשות אינטראקטיביות יותר ויותר ונעשה חשוב לתכנן עבור סוג כזה של תפקודיות.

תוכניות מיושנות אילצו את המשתמש להתקדם צעד אחר צעד דרך סדרה של מסכים. תוכניות מונחות אירועים מציגות את כל האפשרויות בבת אחת ומגיבות לפעולות המשתמש.

תכנות מוכוון עצמים (object oriented programming) מנסה לענות על צרכים אלה. הוא מספק שיטות לניהול, משיג מחזור של רכיבי תוכנה ומשלב בין נתונים לבין משימות שמתפעלות אותם.

התמצית של תכנות מוכוון עצמים היא ההתייחסות לנתונים ולהליכים שפועלים על הנתונים כ"**עצם**", "**אובייקט**" (object) יחיד – ישות שלמה שיש לה זהות ומאפיינים ייחודיים.

C++ ותכנות מוכוון עצמים

C++ תומכת בצורה מלאה בתכנות מוכוון עצמים (object oriented), כולל שלושת אבני היסוד של פיתוח מוכוון עצמים: כימוס (encapsulation), ירושה (inheritance) ופולימורפיזם (polymorphism).

כימוס

כאשר מהנדס צריך להוסיף נָגָד למכשיר שהוא בונה, הוא משתמש בדרך כלל ברכיב קיים שהוא מוצא בתיבה. הוא בודק את הפסים הצבעוניים שמציינים את התכונות ובוחר באחד שדרוש לו. הנגד הוא "קופסה שחורה" ככל שזה נוגע למהנדס – לא אכפת לו כיצד הוא פועל, כל עוד הוא מתאים לדרישותיו; הוא אינו צריך ל'הסתכל' בתוך הרכיב כדי להשתמש בו.

התכונה של היות הרכיב יחידה שלמה ומוכללת מבחינה פונקציונלית נקראת **כימוס** (encapsulation). הכימוס מאפשר **הסתרת נתונים** (data hiding). הסתרת נתונים היא מאפיין חשוב ומהותי, המצביע על כך שאובייקט יכול להיות בשימוש בלי שהמשתמש ידע, או שיהיה אכפת לו, כיצד הוא מורכב ופועל בתוכו. כפי שניתן להשתמש במקרר מבלי לדעת כיצד המדחס פועל, כך ניתן להשתמש באובייקט מתוכנן היטב מבלי לדעת על הרכבו ועל הנתונים הפנימיים שלו.

בדומה, כאשר המהנדס משתמש בנגד, הוא אינו צריך לדעת דבר על מבנהו הפנימי. כל תכונות הנגד **כמוסות** (encapsulated), או מוכללות, באובייקט הקרוי "נגד"; הן אינן מפוזרות בין כל המעגלים האלקטרוניים. אין צורך להבין כיצד הנגד פועל כדי להשתמש בו ביעילות. הנתונים שלו חבויים בתוך המארז ואינם גלויים למשתמש.

++C תומכת בתכונות הכימוס על ידי יצירת **טיפוסים המוגדרים על ידי המשתמש** (user-defined types), הנקראים **מחלקות** (classes). בפרק 6 תלמד איך ליצור מחלקות. ברגע שנוצרה מחלקה מוגדרת היטב, היא מתפקדת כישות כמוסה (encapsulated) ומשתמשים בה כיחידה שלמה מבלי לחדור לרכיביה הפנימיים. הפעולות הפנימיות של המחלקה צריכות להיות מוסתרות. המשתמשים במחלקה מוגדרת היטב אינם צריכים לדעת כיצד היא פועלת; הם רק צריכים לדעת כיצד להשתמש בה.

ירושה ומחזור

כאשר המהנדסים של "סוסיתא תעשיות רכב" רצו לבנות מכונית חדשה, עמדו בפניהם שתי אפשרויות: הם יכלו להתחיל מאפס, או שיכלו לשנות מודל קיים ולהתאימו לדרישותיהם החדשות. מודל "כרמל" שלהם נראה מתאים, אבל בכל זאת רצו להוסיף רכיבים אחרים, כמו למשל תיבת הילוכים אוטומטית, או הזרקת דלק במקום קרבורטור רגיל. המהנדס הראשי העדיף לא להתחיל מהבסיס, אלא לומר "הבה נבנה מכונית כרמל חדשה אשר נוסיף לה את היכולות החדשות ונכנה אותה בשם כרמל דוכס". כלומר, "כרמל דוכס" היא סוג של "כרמל", אך בעלת ייחוד כלשהו, בעלת מאפיינים חדשים.

++C תומכת ב**הורשה** (inheritance). ניתן להכריז על **טיפוס** (type) חדש, שהוא הרחבה של טיפוס קיים. תת-המחלקה החדשה נגזרת מהטיפוס הקיים, ולכן היא קרויה גם טיפוס נגזר (derived type). בדוגמה שלנו, המכונית "כרמל דוכס" נגזרה מ"כרמל", ולכן יורשת את כל תכונותיה, אך יכולה להוסיף או לשנות ככל הנדרש. כך למשל, אם נגדיר מחלקה בשם "דוח" נוכל לגזור ממנה דוחות ייחודיים שונים ורבים, אך לכולם יהיו המאפיינים המבדילים בין דוח לבין מסד נתונים, למשל. על ירישה והורשה (שני צדדים של אותו מטבע) ועל היישום ב- ++C תלמד בפרק 11 ובפרק 15.

פולימורפיזם

בשעה שמעבירים הילוך, סביר שמכונית כרמל דוכס החדשה תגיב אחרת מאשר מכונית כרמל הקיימת. לצורך מעבר הילוך בכרמל דוכס צריך להפעיל את ידית ההילוכים בלבד, בשעה שבמכונית כרמל גם צריך לשחרר את המצמד שבין המנוע לתיבת ההילוכים. כאן המשתמש מודע להבדלים. אך אם נתבונן בשוני שבין הזרקת דלק לבין שימוש בקרבורטור רגיל, ניווכח שהמשתמשים אינם חייבים לדעת על ההבדל כלל וכלל. הם רק צריכים ללחוץ "פול גז" והדבר הנכון יקרה, בהתאם למכונית שבה הם נוהגים.

++C תומכת בתפיסה שאובייקטים שונים יבצעו את "הדבר הנכון" על ידי מנגנון הקרוי **פולימורפיזם** (polymorphism), אשר מתייחס לפונקציות ולמחלקות. פולימורפיזם מתייחס לאותו שם של פונקציה או של מחלקה, אשר מקבל צורות רבות. בנושא זה נעסוק בפרק 10 ובפרק 13.

כיצד התפתחה ++C?

כאשר התפיסה **מוכוונת האובייקטים** (object oriented) החלה להיות מקובלת בתחומי ניתוח, תכנון ותכנות של יישומי מחשבים, לקח ביורן סטראוסטרופ (Bjarne Straustrup) את השפה המקובלת ביותר עבור פיתוח מסחרי של תוכנה, את שפת C, והרחיב אותה כדי לספק את המאפיינים הדרושים ליישום תכנות מוכוון עצמים.

למרות ש- ++C היא הרחבה של C, ולמעשה כל תוכנית חוקית של C היא תוכנית חוקית של ++C, הקפיצה מ-C אל ++C היא מאוד משמעותית. הזיקה שבין ++C אל C, איפשרה למתכנתי C לעבור בקלות יחסית לשימוש ב- ++C. אולם, כדי לקבל את היתרון המלא של ++C, היה צורך לעבור תהליך של שינוי חשיבתי וללמוד מושגים חדשים ודרך חדשה לפתרון בעיות תכנות.

האם עלי ללמוד תחילה את שפת C?

השאלה העולה בסופו של דבר: "מכיון ש- ++C היא הרחבה של C, האם צריך ללמוד תחילה את C?". סטראוסטרופ ומתכנתי ++C אחרים מסכימים שלא רק שאין צורך ללמוד תחילה את C, אלא לעיתים רצוי שלא לעשות זאת.

ספר זה אינו מניח שלקוראיו יש ניסיון תכנות קודם כלשהו. אולם, אם אתה מתכנת בשפת C, חמשת הפרקים הראשונים יהיו בעיקר חזרה עבורך. החל מפרק 6 אנו מתחילים את העבודה האמיתית ללימוד פיתוח תוכנה מוכוונת אובייקטים.

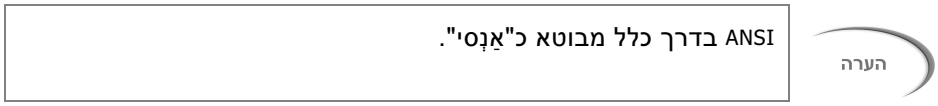
++C ו- Java

++C הינה השפה הדומיננטית והמקובלת ביותר לפיתוח יישומי תוכנה מסחריים. בשנים האחרונות, Java קראה תיגר על שליטה זו, אולם הגלגל שוב התהפך ורבים מהמתכנתים שעזבו את ++C לטובת Java החלו לאחרונה לחזור. בכל מקרה, שתי השפות מאוד דומות, וניתן לומר שהחפיפה ביניהן היא בשיעור של 90% בערך. במהלך השנים, התפתחה Java לשפת התכנות של יישומי אינטרנט, בשעה ש- ++C מתמקדת ביישומים מסחריים רגילים, גרפיקה מחשבית ועוד.

תקן ANSI

ועדת התקנים הפועלת במסגרת מכון התקנים האמריקני ANSI (**A**merican **N**ational **S**tandards **I**nstitute), הגדירה תקן בינלאומי עבור ++C.

תקן ++C גם הוגדר במסגרת תקני ISO (**I**nternational **S**tandards **O**rganization – ארגון התקנים הבינלאומי), תקני NCITS (**N**ational **C**ommittee for **I**nformation **T**echnology **S**tandards – הוועדה הלאומית לתקני טכנולוגיית מידע), תקני X3 (השם הישן של NCITS) ובמסגרת תקני ANSI/ISO. ספר זה יתייחס בעיקר לתקן ANSI, שהוא המקובל והנפוץ ביותר.



הכפיפות לתקן ANSI הוא ניסיון להבטיח ש- C++ תהיא ניידת. כלומר, להבטיח שקוד תואם תקן ANSI הנכתב עבור מהדר של Microsoft, יעבור הידור ללא שגיאות גם במהדר של ספק אחר. בנוסף, מכיון שהקוד בספר זה תואם את תקן ANSI הוא צריך לעבור הידור ללא שגיאות גם בכל אחת מהמערכות Windows, Mac, או Alpha.

עבור רוב לומדי C++ תקן ANSI יהיה שקוף. התקן הינו יציב, וכל היצרנים הגדולים תומכים בו. כאמור, השתדלנו לשמור על כך שכל הקוד המוצג בספר ובקבצי התקליטור יהיה תואם ANSI.

הכנות לתכנות

C++, אולי יותר משפות אחרות, דורשת שהמתכנתים יתכננו את התוכנית לפני שהם כותבים אותה. בעיות פשוטות לכאורה, כמו אלו שנדונות בפרקים הראשונים אינן דורשות תכנון רב. בעיות מורכבות, לעומת זאת, כמו אלו שמתכנתים מקצועיים מתמודדים אתן בכל יום, דורשות תכנון. ככל שהתכנון יסודי יותר, כך סביר יותר שהתוכנית תפתור את הבעיה בזמן ובתקציב הנתונים. תכנון טוב תורם לכך שהתוכנית תהיה נקיה משגיאות וקלה לתחזוקה. מעריכים כי 90% מעלות התוכנה מוקדשת ל**ניפוי שגיאות** (debugging) ולתחזוקה. במידה שתכנון טוב יכול להפחית עלויות אלו, יכולה להיות לו השפעה משמעותית על העלות הסופית של הפרויקט.

השאלה הראשונה שצריכה להישאל בעת עיצוב תוכנית כלשהי היא: "מה הבעיה שמנסים לפתור?". לכל תוכנית צריכה להיות מטרה ברורה ומנוסחת היטב. במהרה תראו שגם לתוכנית הפשוטה ביותר בספר זה יש צורך להגדיר מטרה.

השאלה השנייה שכל המתכנתים הטובים שואלים היא: "האם ניתן להשיג זאת בלי להזדקק לכתיבת תוכנה לפי הזמנה?". מְחָזור של תוכנית ישנה על ידי שימוש בעט ונייר, או על ידי קניית תוכנה מהמדף, הוא בדרך כלל פתרון טוב יותר לבעיה מאשר כתיבת משהו חדש. המתכנת שיכול להציע חלופות אלו לעולם לא יסבול מחוסר עבודה; מציאת פתרון פחות יקר לבעיות של היום תמיד ייצור הזדמנויות חדשות מחר.

בהנחה שמובן וברור מהי הבעיה, ואם היא מחייבת כתיבת תוכנית חדשה, צריך להתחיל בתכנון.

התהליך של הבנת הבעיה במלואה (ניתוח – analysis) ויצירת פתרון (עיצוב – design) הוא הבסיס ההכרחי כדי לכתוב יישומים מסחריים ברמה גבוהה. למרות שצעדים אלה צריכים להיות לפני הקידוד – כלומר, יש להבין את הבעיה, לתכנן ולעצב את הפתרון לפני היישום שלו – עדיף ללמוד את התחביר הבסיסי והסמנטיקה של C++ לפני שלומדים שיטות ניתוח ותכנון תקניות.

סביבת הפיתוח

ספר זה מניח כי ל**מהדר** (compiler) שברשותך יש **עורך** (editor) המאפשר להקליד ישירות למסך, בלי לדאוג לסביבה הגרפית, כמו שקיימת ב-Windows או ב-Mac. רצוי לבדוק את תיעוד המהדר לפני תחילת העבודה, או לחפש תוכנית שיכולה לעשות זאת.

לרבים מהמהדרים יש עורך טקסט מובנה, או שניתן להשתמש בעורך טקסט או מעבד תמלילים מסחרי שיכול ליצור קבצי טקסט. אין כל חשיבות לדרך שבה כותבים את התוכנית; הדבר החשוב הוא שתהיה אפשרות לשמור קבצי טקסט (plain-text, text only) פשוטים ונקיים, ללא פקודות עיבוד תמלילים המוטבעות לעיתים בטקסט. דוגמאות לעורכים בטוחים הם Windows Notepad, DOS Edit, Brief, Epsilon, EMACS, או vi. למעבדי תמלילים מסחריים רבים, כמו Word ורבים אחרים, יש אפשרות לשמור קבצי טקסט פשוטים.

הקבצים שנוצרים על ידי העורך נקראים **קבצי מקור** (source files), ועבור C++ הם בדרך כלל מקבלים את הסיומות cpp., cp., או c.. בספר זה ניתן לכל קבצי קוד המקור את הסיומת **cpp.**, אך ראוי לבדוק אם למהדר שמשתמשים בו אין דרישות אחרות.

הערה

לרוב מהדרי C++ לא אכפת איזו סיומת ניתנת לקבצי המקור. אם לא נפרט אחרת, נשתמש בסיומת **cpp.** כברירת מחדל. אולם יש להיזהר, כי יש מהדרים המתייחסים לקבצי c. כקוד C ולקבצי cpp. – כקוד C++. חשוב לבדוק את התיעוד של המהדר שמשתמשים בו.

עשה	אל תעשה
השתמש בעורך טקסט פשוט כדי ליצור את קוד המקור שלך, או השתמש בעורך מובנה שמגיע עם המהדר.	אל תשתמש במעבד תמלילים השומר תווי עיצוב מיוחדים. אם הינך משתמש במעבד תמלילים, שמור את הקובץ במבנה ASCII text.
שמור את קבצי התוכנית עם סיומת c., cp., או cpp. (רצוי – cpp.).	
בדוק בתיעוד שלך את תיעוד המהדר והמקשר (linker) שלך, כדי לוודא שאתה יודע כיצד להדר ולקשר את התוכניות.	

הידור קובץ המקור

למרות שקוד המקור בקובץ עשוי להיראות מוצפן, ולמי שאינו יודע C++ יהיה קשה להבין למה הוא משמש, הוא עדיין נחשב כצורה שניתנת לקריאה על ידי אדם. קובץ קוד המקור אינו תוכנית בת-ביצוע, ולא ניתן להפעיל אותו או להריץ אותו כפי שניתן לעשות עם תוכנית.

כדי להפוך קוד המקור לתוכנית יש להשתמש במהדר. כיצד קוראים למהדר וכיצד אומרים לו היכן למצוא את קוד המקור? הדבר תלוי במהדר שבו משתמשים; זאת יש ללמוד מתוך התיעוד.

לאחר שקוד המקור עבר **הידור** (compilation), נוצר **קובץ יעד** (object file). קובץ זה נושא בדרך כלל את הסיומת **obj.**, אולם זו עדיין אינה תוכנית שניתנת להפעלה. כדי להפוך את הקובץ לתוכנית שניתנת להפעלה, חייבים להפעיל **מקשר** (linker).

יצירת קובץ שניתן להפעלה עם מקשר

תוכניות C++ נוצרות על ידי קישור של קובץ obj. אחד או יותר עם ספריה אחת או יותר. **ספריה** (library) היא אוסף של קבצים הניתנים לקישור, ואשר סופקו עם המהדר, שנרכשו בנפרד, או שנכתבו וצורפו לספריה. לכל מהדרי C++ מצורפת ספריה של פונקציות שימושיות (או הליכים, procedures) ומחלקות שניתן לכלול אותם בתוכנית. **פונקציה** (function) היא גוש, או בלוק, של קוד שמבצע שירות אחד בדרך כלל, כמו חיבור שני מספרים, הדפסה למסך, עריכת כותרת דוח ועוד. **מחלקה** (class) היא אוסף של נתונים ופונקציות הקשורות בהם. החל מפרק 5 נעסוק רבות בפונקציות ובמחלקות.

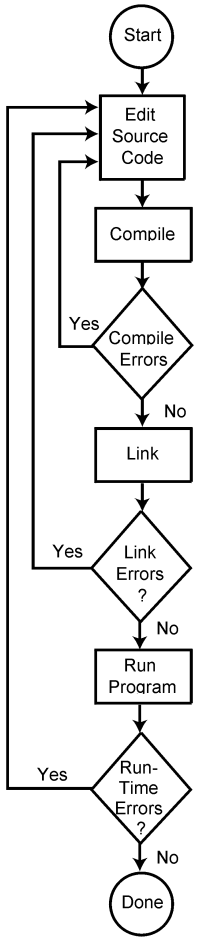
השלבים ליצירת קובץ שניתן להפעלה (excecutable file), או קובץ הרצה:

1. צור קובץ קוד מקור עם סיומת cpp.
2. הדר את קוד המקור לקובץ עם סיומת obj.
3. קשר את קובץ obj. עם הספריות הנחוצות ליצירת תוכנית שניתנת להרצה.

מחזור הפיתוח

אם כל תוכנית היתה פועלת בפעם הראשונה שמנסים אותה, מחזור הפיתוח השלם היה: כתוב את התוכנית, הדר את קוד המקור, קַשֵּר את התוכנית, הרץ אותה. לצערנו, כמעט בכל תוכנית, ללא תלות בפשטות שלה, יכולות להיות ותהיינה שגיאות או באגים (bugs - תקלות). יהיו באגים שיגרמו להידור להיכשל, אחרים יגרמו לקישור להיכשל ואחרים יופיעו רק כאשר מריצים את התוכנית.

כל סוג של באג שנמצא חייב להיות מתוקן, והדבר כולל עריכת קוד המקור, הידור מחדש וקישור מחדש והרצת התוכנית מחדש. מחזוריות זו מוצגת בתרשים 1.1 אשר ממחיש את השלבים במחזור הפיתוח.

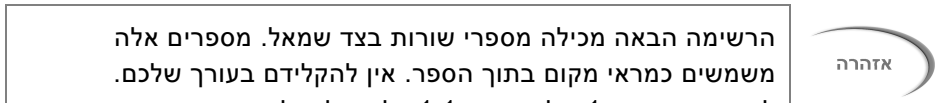


תרשים 1.1 *השלבים בהתפתחותה של תוכנית C++.*

HELLO.cpp – תוכנית C++ הראשונה שלך

ספרי תכנות מסורתיים מתחילים בכתיבת המילים "Hello World" על המסך, או נוסחים שונים של הצהרה זו. מסורת מקודשת זו ממשיכה גם כאן.

כתוב, או הקלד, את התוכנית הראשונה אל העורך (editor), בדיוק כפי שהיא נראית. לאחר שהינך בטוח שהיא נכונה, שמור את הקובץ, הדר אותו (compile), קשר (link) והרץ אותו. התוכנית תדפיס את המילים Hello World על המסך. אין צורך להיות מוטרד מהאופן בו התוכנית פועלת; מטרתה לערוך היכרות עם מחזור הפיתוח בלבד. כל היבט של תוכנית זו יוסבר במהלך שני הפרקים הבאים.



הרשימה הבאה מכילה מספרי שורות בצד שמאל. מספרים אלה משמשים כמראי מקום בתוך הספר. אין להקלידם בעורך שלכם. לדוגמה, בשורה 1 של רשימה 1.1 עליכם להקליד:

```
#include <iostream.h>
```

תדפיס 1.1 HELLO.cpp – התוכנית Hello World

```
1:    #include <iostream.h>
2:
3:    int main()
4:    {
5:          cout<<"Hello World!\n";
6:          return 0;
7:    }
```

עליך להקליד את הקוד בדיוק כפי שהוא כתוב, ושים לב במיוחד לפיסוק. הסימן >> בשורה 5 הוא סימן שינוי כיוון, שנוצר ברוב המקלדות על ידי החזקת מקש shift והקלדת פסיק (או האות 'ת') פעמיים. שורה 5 מסתיימת בנקודה-פסיק (; semicolon). אין להשמיט אותו! סימן זה מציין את סיום ההוראה.

בנוסף, עליך לעקוב אחר הוראות המהדר שברשותך. רוב המהדרים יבצעו קישור אוטומטי, אך יש לבדוק את התיעוד, כי ייתכן שיש צורך לתת למהדר הוראה מפורשת לעשות זאת. אם קיבלת הודעה על שגיאות במהלך ההידור, בדוק אם אלו הן התרעות בלבד, או שגיאות שאינן מאפשרות להתקדם. בדוק היטב את הקוד, וראה אם הוא שונה מהקוד שמוצג בספר. אם אתה רואה שגיאה בשורה 1, כמו למשל cannot find file iostream.h, בדוק את התיעוד של המהדר ולמד כיצד עליך להגדיר את נתיב הספריה (include path), או את משתני הסביבה. אם קיבלת שגיאה האומרת שלא קיים אב-טיפוס (prototype) ל-main, עליך להוסיף את השורה ; **int main()** לפני שורה 3. במקרה זה עליך לזכור להוסיף שורה זו לפני כל פונקציה main בכל תוכנית בספר זה. רוב המהדרים אינם דורשים זאת, אולם ייתכן שהמהדר שברשותך ידרוש זאת. התוכנית הגמורה שלך תיראה כך:

```
1:    #include <iostream.h>
2:    int main(); // רוב המהדרים אינם צריכים שורה זו
3:    int main()
4:    {
5:          cout<<"Hello World!\n";
6:          return 0;
7:    }
```

קשה לקרוא ולהבין תוכנית אם אינך יודע כיצד לבטא את התווים המיוחדים ואת מילות המפתח. קוראים את השורה הראשונה כך: "סולמית (pound / hash) אינקלוד איי-או-סטרים נקודה h". את שורה 5 קוראים: "סי-אאוט הלו וורלד".

כשתנסה להריץ את התוכנית HELLO.exe, היא צריכה לכתוב על המסך:

```
Hello World!
```

אם זה אכן קרה – כל הכבוד! ברגע זה הקלדת, הידרת והרצת את תוכנית C++ הראשונה שלך. אולי אין זה שיא שאיפותיך בתכנות, אבל כל מתכנת C++ מקצועי חייב להתחיל בצעד הראשון.

בתקליטור המצורף לספר זה תמצא את כל התוכניות המודפסות תחת הכותרת "תדפיס". כל הקבצים נמצאים בספריה ומסומנים, כפי שהם מסומנים בספר. קרא בעיון את ההוראות בנספח **התקליטור המצורף**.

אם כך, בוודאי תבחר לקרוא את הקובץ, ולא להקליד אותו. ראוי שאת התוכניות הראשונות תקליד, כדי להכיר את תהליך כתיבת התוכניות. אחר כך, השתמש בקבצים הקיימים ורק ערוך בהם שינויים כנדרש.

שימוש בספריות סטנדרטיות

כדי לוודא שלקוראים המשתמשים במהדרים ישנים יותר לא תהיינה תקלות עם הקוד שבספר זה, בחרנו להשתמש בסגנון הכתיבה הישן עבור ציון קבצי include:

```
#include <iostream.h>
```

צורת הכתיבה החדשה להכללת קבצי הכותר (header) החדשים של הספריה הסטנדרטית (standard library) היא:

```
#include <iostream>
```

הקוד הישן אמור לפעול בכל המהדרים, אך יש לו כמה חסרונות. אם אתה מעדיף להשתמש בספריות הסטנדרטיות החדשות, עליך להשמיט את הציון h. משם הקובץ, ולכתוב כך:

```
#include <iostream>
```

בנוסף, עליך להוסיף את השורה הזו לאחר רשימת הקבצים הכלולים:

```
using namespace std;
```

השימוש במרחב שמות (namespace) מוסבר בפירוט בפרק 17.

גם אם תשתמש בקבצי header סטנדרטיים וגם אם לא, הקוד בספר זה צריך לרוץ ללא שינוי. ההבדל העיקרי בין הספריות הישנות לבין הספריה הסטנדרטית החדשה היא הספריה iostream (המתוארת בפרק 16). אפילו שינויים אלה לא ישפיעו על הקוד בספר זה, כי במידה שהם קיימים הם עוסקים בנושאים שמעבר לדיון בספר לימוד זה.

TCLite ו- Visual C++ 6

כל התוכניות בספר זה נבדקו עם המהדר TCLite במהדורה הנמצאת בתקליטור המצורף. יש תוכניות מתקדמות שאינן יכולות לרוץ בעזרת מהדר זה והן הורצו ב- Visual C++ 6.0. התוכניות יכולות לעבור הידור, קישור ולרוץ בצורה נקיה עם כל מהדר של Microsoft Visual C++, החל מגירסה 4.0 ומעלה. מכיון שהקוד תואם את תקן ANSI, כל התוכניות בספר זה צריכות לרוץ היטב על מהדר תואם ANSI של כל ספק. כך בתיאוריה, אך כידוע עולם המציאות אינו תואם במלואו לתיאוריה, ולכן – ייתכנו הפתעות.

כדי שתוכל להתחיל, עליך ללמוד כיצד לערוך, להדר, לקשר ולהריץ תוכנית על ידי מהדר של Microsoft. אם אתה משתמש במהדר אחר, ייתכן שפעולות מסוימות תהיינה שונות ולכן – **תמיד קרא בעיון את התיעוד של המהדר שברשותך**. גם אם הינך משתמש במהדר Microsoft Visual C++ 6.0, ראוי שתעיין בתיעוד שלו בתשומת לב.

בניית הפרויקט Hello World

כדי ליצור ולבדוק את התוכנית Hello World, עקוב אחר שלבים אלה:

1. הפעל את המהדר.
2. בחר File ← New מהתפריטים.
3. בחר Win32 Console Application, כתוב את שם הפרויקט, כמו למשל Example 1, ולחץ OK.
4. בחר Empty Project מתפריט האפשרויות, ולחץ OK.
5. בחר File ← New מהתפריטים.
6. בחר C++ Source File וקרא לו ex1.
7. הקלד את קוד התוכנית כפי שהוא מופיע בספר.
8. בחר Build ← Build Example 1.exe.
9. בדוק שאין שגיאות במהלך העבודה עד כה.
10. לחץ Ctrl+F5 כדי להריץ את התוכנית.
11. לחץ על מקש הרווח (spacebar) כדי לסיים את התוכנית.

שגיאות הידור

שגיאות בזמן הידור יכולות להתרחש בשל מיגוון סיבות. בדרך כלל הן נובעות משגיאות הקלדה או שגיאות קלות לא מכוונות אחרות. מהדרים טובים יאמרו לך לא רק במה טעית, אלא גם מהו המקום המדויק בקוד שבו התגלתה הטעות או השגיאה. הטובים מאוד אפילו יציעו לך פתרון!

תוכל לראות זאת אם תכניס במכוון שגיאה בתוכנית ותנסה להדר ולהריץ אותה. אם HELLO.cpp רצה ללא בעיות, ערוך אותה כעת והסר את הסוגר המסולסל שבשורה 7. התוכנית תיראה כעת כמו בתדפיס 1.2.

תדפיס 1.2 הדגמה של שגיאת הידור

```
1:    #include <iostream.h>
2:
3:    int main()
4:    {
5:          cout<<"Hello World!\n";
6:          return 0;
```

הדר מחדש את התוכנית ואז תופיע הודעת שגיאה, כמו זו, למשל:

```
Hello.cpp, line 5: Compound statement missing terminating; in function main().
```

או שתוצג הודעת שגיאה שנראית כך:

```
F:\Mcp\Tycpp21d\Testing\List0101.cpp(8) : fatal error C1004:
unexpected end of file found
Error executing c1.exe.
 }
```

הודעת שגיאה זו מציינת את הקובץ ומספר השורה של השגיאה, וגם מהי התקלה (למרות שלעיתים אין זה מובן למשתמש מתחיל). שים לב שהודעת השגיאה מצביעה על שורה 5. המהדר לא היה בטוח אם התכוונת לכתוב סוגר מסולסל לפני או אחרי הפקודה cout בשורה 5. לעיתים הודעת השגיאות רק מצביעה על התקלה ועל מקום מקורב, ולא מדויק. אם המהדר היה יכול לזהות באופן מושלם כל שגיאה, הוא גם יכול היה לתקן את הקוד עצמו.

סיכום

לאחר קריאת פרק זה צריכה להיות לך הבנה טובה כיצד C++ התפתחה ואיזה בעיות היא תוכננה לפתור. עליך להשתכנע שלימוד C++ הוא הבחירה הנכונה לכל מי שמתעניין בתכנות בעשור הבא. C++ מספקת את הכלים של תכנות מוכוון עצמים ואת הביצועים של שפה ברמת מערכת, וכך היא הופכת לשפת הפיתוח הנבחרת.

למדת כיצד להקליד, להדר ולהריץ את תוכנית C++ הראשונה שלך, וגם למדת מהו מחזור הפיתוח התקין. למדת מעט מהו תכנות מוכוון עצמים (object oriented). בנושאים אלה נעסוק במהלך הפרקים הבאים.

שאלות ותשובות

ש: מהו ההבדל בין עורך טקסט לבין מעבד תמלילים?

ת: עורך טקסט יוצר קבצים כשבהם טקסט פשוט, תווים בלבד. פקודות עיצוב או סימנים מיוחדים אחרים אינם כלולים בו, כמו שקיימים במעבד תמלילים. בקבצי טקסט אין גלישת מילים אוטומטית, הדפסה **מודגשת** (Bold), *נטויה* (Italic) וכד׳, תכונות המאפיינות כל מעבד תמלילים.

ש: אם למהדר שלי יש עורך מובנה, האם עלי להשתמש בו?

ת: כמעט כל המהדרים יהדרו קוד שנוצר על ידי עורך טקסט כלשהו. אולם, היתרונות שבשימוש בעורך טקסט מובנה יכולים לכלול את היכולת לנוע במהירות קדימה ואחורה בין שלבי העריכה וההידור של מחזור הפיתוח. מהדרים מתוחכמים כוללים סביבת פיתוח משולבת ומאפשרים למתכנת גישה לקבצי עזרה, לערוך ולהדר את הקוד במקום, וגם לפתור שגיאות הידור וקישור בלי לעזוב את סביבת הפיתוח.

ש: האם ניתן להתעלם מהודעות אזהרה של המהדר?

ת: ספרים רבים מתחמקים מתשובה ישירה בנושא זה. אני אסתכן ואומר: לא! התרגל מההתחלה לטפל בכל הודעות האזהרה כמו בשגיאות. C++ משתמשת במהדר כדי להזהיר אותך על פעולה שאינה נכונה, פעולה שאולי לא התכוונת לה. שים לב לאזהרות אלו ועשה מה שדרוש כדי להעלים אותן.

ש: מהו זמן הידור?

ת: זמן הידור (compile time) הוא הזמן בו אתה מריץ את המהדר; בזמן קישור (link time) אתה מריץ את המקשר, ובזמן ריצה (run time) אתה מריץ את התוכנית. אלה רק ציונים לזיהוי שלושת הזמנים שבהם יכול להיות דיווח על שגיאות ותקלות של התוכנית.

סדנה

הסדנה מציעה שאלות המסייעות לחזור על החומר ולבסס את הידע וההבנה; היא מכילה גם תרגילים שמאפשרים לצבור ניסיון ביישום החומר. נסה לענות על השאלות והתרגילים לפני שתבדוק את התשובות בנספח ד׳. ודא שהבנת את כל התשובות לפני שתעבור לפרק הבא.

שאלון

1. מהו ההבדל בין מפרש (interpreter) לבין מהדר (compiler)?
2. כיצד להדר קוד מקור במהדר שברשותך?
3. מה עושה המקשר (linker)?
4. מהם השלבים במחזור פיתוח תקין?

תרגילים

1. מה מבצעת התוכנית הזו? (נסה לענות מבלי להריץ אותה)

```
1:    #include <iostream.h>
2:    int main()
3:    {
4:          int x = 5;
5:          int y = 7;
6:          cout << "\n";
7:          cout << x + y << " " << x * y;
8:          cout << "\n";
9:          return 0;
10:   }
```

2. הקלד את התוכנית מתרגיל 1, הדר (compile) וקשר (link) אותה. מה היא עושה? האם זהו מה שניחשת?

3. הקלד את קטע הקוד הבא והדר אותו. מהי השגיאה המתקבלת?

```
1:    include <iostream.h>
2:    int main()
3:    {
4:          cout << "Hello World!\n";
5:          return 0;
6:    }
```

4. תקן את השגיאה בתוכנית שבתרגיל 3 והדר אותה מחדש, קשר והרץ אותה. מה היא עושה?