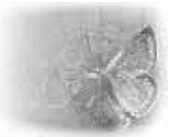
ערכת לימוד

C++

הוצאת פוקוס מחשבים

que®



פרק דוגמה מתוך הספר

ערכת לימוד ל- C++

Authorized translation from the English language edition of:
C++ From Screatch
By Jesse Liberty
Published by Que Publishing
Copyright © 1999 by Que Publishing

מהדורה ראשונה בעברית - 2000
הדפסה: 1 2 3

הוצאת פוקוס-מחשבים
ת"ד 863 ר"ג, 52108
טל': 03-6773898, פקס: 03-5746513
אתר אינטרנט: http://www.focus.co.il
דואר אלקטרוני: focus@focus.co.il

הפצה לחנויות: ליאור שרף שיווק והפצה בע"מ,
טל': 03-9021902, פקס: 03-9025758

בספר זה השתמשנו בפנייה למשתמש בלשון זכר לשם נוחיות בלבד, ואתכן הסליחה

כל הזכויות למהדורה העברית שמורות לפוקוס הוצאת ספרי מחשב בע"מ
© 2000

אין להעתיק, לשכפל ולצלם ספר זה ו/או את התקליטון המצורף לו, או קטעים מהם בשום צורה ובשום אמצעי אלקטרוני, אופטי או מכני לכל מטרה שהיא, ללא אישור בכתב מפוקוס, הוצאת ספרי מחשב בע"מ.

תרגום ולכידת מסכים: ענת סילמן
בדיקה מקצועית: שמעון פרוידנברגר
סדר ועריכה: ורדה אחירון-זנזורי וגיא משה
עטיפה: בלום עיצוב, הפקה וכתיבה
לוחות: מ. אביב בע"מ
הדפסה: דפוס טופSפרינט בע"מ
כריכה: כריכיית אהרון בע"מ
דנאקוד 239-298

הודעת זכויות

הספרים מוגנים בזכויות יוצרים. הוצאת פוקוס-מחשבים מרשה את הורדתם כשירות לציבור, לשימוש אישי של המוריד בלבד. אין להפיץ או להעביר לאחרים, בין בתמורה ובין ללא תמורה, חלק כלשהו מן החומר המורד. אין להפיץ תדפיס או העתק של החומר בכל צורה שהיא ללא רשות מפורשת, בכתב ומראש מהוצאת פוקוס-מחשבים.

תוכן מקוצר

הקדמה 13

פרק 1: מבוא לשפת C++ 17

פרק 2: מתחילים..... 39

פרק 3: זרימת התוכנית 67

פרק 4: יצירת מחלקות..... 85

פרק 5: משחקים את המשחק 109

פרק 6: שימוש ברשימות מקושרות..... 159

פרק 7: שיטות עבודה "לפי הספר" 201

פרק 8: שימוש ברב-צורתיות (POLYMORPHISM)..... 237

פרק 9: מימוש תבניות 269

פרק 10: שימוש בספריית התבניות הסטנדרטית 299

פרק 11: המחשב מנחש 321

פרק 12: האצלת אחריות 359

פרק 13: הימשכות (PERSISTENCE) 399

פרק 14: הצגת חריגים 453

נספח א': בינארי והקסדצימלי 475

נספח ב': קדימות אופרטורים 485

מפתח העניינים..... 487

תוכן העניינים

הקדמה 13

לימוד C++ אינו חייב להיות קשה 13
על התקליטון המצורף 14
הידור הקוד 14
מוסכמות בספר זה 14
שבירת הקוד 15

פרק 1: מבוא לשפת C++ 17

תוכניות 18
פתרון בעיות 19
תכנות פרוצדורלי, מובנה ומונחה עצמים 19
מדוע C++? 21
מדוע תכנות מונחה אובייקטים? 21
התמודדות עם מורכבות 22
אובייקטים 22
כימוס 22
האצלה 23
התמחות 24
הכללה ורב צורתיות 25
שלושת העמודים 26
כיצד מתבצעים ניתוח ועיצוב מונחי אובייקטים? 26
ניתוח ועיצוב מונחי עצמים לפרויקטים קטנים 28
החזון 28
ניתוח דרישות 30
עיצוב מהיר ומלוכלך 31
מימוש 31
תוכניות וקוד מקור 32
מהדרים 33
סביבת הפיתוח שלך 33
עורך טקסט 33
הידור קוד המקור 34
יצירת קובץ בר ביצוע עם המקשר 34
מחזור הפיתוח 35
DECRYPTIX.CPP: תוכנית C++ הראשונה שלך 36

שגיאות הידור 37
הפצה 38
השלבים הבאים 38

פרק 2: מתחילים 39

מה גודלו של פרויקט קטן? 39
מדוע ללמד תהליך שטוב רק לפרויקטים קטנים? 40
חילוץ הידע שלך 40
יצירת הפרויקט 40
בדיקת הקוד 42
ניתוח הקוד 44
תחומי השם 45
חיפוש שגיאות בקוד 48
החזרת ערך 49
main() שווה יותר מאחרות 49
שימוש בפקודה cout להדפסה למסך 50
משתנים 52
קבועים 62

פרק 3: זרימת התוכנית 67

בניית תוכנית חסונה 67
מה אתה מנסה להשיג? 70
פתרון הבעיה עם לולאות 71
אופרטורים יחסיים 72
בלוקים ומשפטים מורכבים 72
אופרטורים לוגיים 73
המשפט IF 75
הערכה מקוצרת 77
קדימות יחסית 77
לחבר את הכל יחד 78
קבועי מנייה 79
חזרה לקוד 80
קבלת תשובה בוליאנית מהמשתמש 81
אופרטור שוויון == 82
else 82
האופרטור המותנה (או המשולש) 83
לחבר את הכל יחד 84

פרק 4: יצירת מחלקות 85

מדוע מחלקות? 85
יצירת טיפוסים חדשים: מחלקה 86
ממשק לעומת מימוש 86
לקוחות 87

מביטים בקוד 87
הצהרה על המחלקה 88
מחלקות ואובייקטים 88
משתנים חברים 88
שיטות חברות או פונקציות חברות 89
גודל אובייקטים 89
קבצים 90
בנאים 90
מפרקים 91
מימוש השיטות 91
הכללת קובץ הכותרת 94
מימוש הבנאי 94
אתחול 94
השימוש במנפה השגיאות 97
בדיקת הבנאי 97
השיטות האחרות 98
אחסון התבנית 98
מהו מערך? 99
אתחול מערכים 99
איברי מערך 101
כתיבה מעבר לסוף מערך 102
יצירת הפתרון 103
בדיקת קובץ הערכים המוגדרים 107

פרק 5: משחקים את המשחק 109

מימוש כלול 111
שיטות חברות קבועות 112
החתימה 113
העברה לפי הפניה והעברה לפי ערך 114
הפניות והעברה לפי הפניה 119
מצביעים 120
מהו מצביע? 120
כתובות זיכרון 121
ביטול ההפניה (Dereferencing) 126
עושים סדר באופרטורים 127
מערכים 127
מערכים כמצביעים 129
העברת המערך כמצביע 132
השימוש ב-ASSERT 136
כיצד ASSERT עובד? 136
לעבור דרך התוכנית פעם אחת, לפי המספרים 143

פרק 6: שימוש ברשימות מקושרות 159
מבני נתונים דינאמיים 160
ספריית התבניות הסטנדרטית 161
רשימות מקושרות 162
מהן רשימות מקושרות? 162
תוכנית הרצה פשוטה 166
השיטה HowMany() 168
Insert() בפירוט 171
שימוש באופרטור new 176
שימוש ברשימה המקושרת הפשוטה שלנו בתוכנית Decryptix! 185
הרץ! 189
לשחק את המשחק 194
פתרון הבעיה באמצעות שיטה חברה 195
העמסת אופרטורים 196
כיצד תבצע העמסת אופרטורים? 197
העברת אובייקטים לפי ערך 198
מדוע זו הפניה? 199

פרק 7: שיטות עבודה "לפי הספר" 201
העמסת שיטות 201
מחלקות Shape 202
העמסת בנאים 202
שיטות ברירת מחדל 203
בנאי ברירת המחדל 204
מתי אתה מקבל בנאי שמסופק על ידי המהדר? 207
מפרק ברירת המחדל 208
בנאי ההעתקה 209
מדוע בנאי ברירת מחדל להעתקה אינו מספיק? 218
כתיבת בנאי העתקה משלך 225
אופרטור ההקצאה 226
כשזה נראה כמו הקצאה, אבל איננו הקצאה 235
חזרה לרשימות מקושרות 235

פרק 8: שימוש ברב-צורתיות (Polymorphism) 237
התמחות (specialization) 238
יתרונות של התמחות 239
רב-צורתיות 239
טיפוסי נתונים מופשטים 240
כיצד זה ממומש בשפת C++? 240
התחביר של הורשה 240
פונקציות אוכפות 241
שיטות וירטואליות 241

כיצד עובדות פונקציות וירטואליות? 242
מפרקים וירטואליים 243
מימוש רב-צורתיות..... 248
הוספת אות שנייה 265
בדיקת האופרטור [] 266

פרק 9: מימוש תבניות 269

יצירת ההיסטוריה..... 269
אופרטורים מועמסים (overloaded operators) 271
כתיבת נתוני מחלקה אל cout..... 272
הגדרה בקובץ הכותרת..... 281
צור תחילה את הטיפוס ללא פרמטריזציה 285
יצירת מופע של התבנית 285
השימוש באובייקט HISTORY..... 287
שימוש בהיסטוריה 298
לעשות את זה כמו שצריך 298

פרק 10: שימוש בספריית התבניות הסטנדרטית 299

המרת !DECRYPTIX ל-STL..... 299
מחלקות אוסף 300
עבודה עם וקטורים..... 300
בדיקת הפלט 315
אתחול המשחק 315
השיטה ()Play..... 316
הצגת תוכן וקטור..... 317
דירוג הניחוש..... 317
יצירת היסטוריה 318
בדיקת הרמזים..... 320

פרק 11: המחשב מנחש 321

מלמדים את המחשב לנחש..... 321
לכידת הכללים 322
מחלקות DECRYPTER..... 323
מימוש Game..... 327
ניחושי Human 335
המחשב מנחש 347
יצירת אובייקט Computer 350

פרק 12: האצלת אחריות 359

הקצאת אחריות..... 359
מהו פרמטר ברירת המחדל?..... 371
SmartChar בפירוט..... 372
להתבונן בזה בפעולה 372

המחשב משחק 373
יצירת מחרוזת חכמה 374
חילול ניחושים: סקירה כללית 378
יצירת ניחוש, פרטים 380
ההיגיון של CanEliminateCharacters 381
CanEliminateCharacters בפירוט 382
IsConsistent בפירוט 388
ההשפעה של סטאטי 392
חזרה לניתוח 393
ביצוע אלימינציה על תווים במקום 395

פרק 13: הימשכות (PERSISTENCE) 399

הימשכות אובייקטים 399
טעמים של הימשכות 400
תכנון הימשכות 400
הקורא והכותב 402
נושאי ממשק משתמש 405
כתיבת המשחק 428
כתיבת המשחק, צעד אחר צעד 428
כתיבת אובייקטים רב-צורתיים 429

פרק 14: הצגת חריגים 453

כשמשהו חריג קורה 453
שחרור המחסנית 461
קריאת אובייקט מהדיסק 464
שחזור HUMAN 473

נספח א': בינארי והקסדצימלי 475

בסיסים אחרים 476
מסביב לבסיסים 477
בינארי 478
מדוע בסיס 2? 479
סיביות ובתים 479
מהו KB? 479
מספרים בינאריים 480
הקסדצימלי 480

נספח ב': קדימות אופרטורים 485

מפתח העניינים 487

הקדמה

בפרק זה

- לימוד C++ אינו חייב להיות קשה
- הידור הקוד
- מוסכמות בספר זה

ספר זה שונה מכל ספר למתחילים בשפת C++ שנכתב **אי פעם**. וזהו ההבדל: כל ספרי התכנות האחרים מתחילים בכך שהם מלמדים אותך מיומנויות פשוטות ההולכות והופכות לקשות, ומוסיפים מיומנות על מיומנות בעודך מתקדם. לאחר שלמדת את כל המיומנויות, הספרים מדגימים לך מה אתה יכול לעשות: תוכנית דוגמה.

ספר **זה** אינו מתחיל בטכניקות של תכנות - הוא מתחיל בפרויקט. נתחיל בניתוח ותכנון הפרויקט, ולאחר מכן נממש את אותו התכנון. מיומנויות תכנות נלמדות בהקשר של המימוש: תחילה עליך להבין את מה שתנסה להשלים, ולאחר מכן תלמד את המיומנויות הנדרשות לביצוע העבודה.

לימוד C++ אינו חייב להיות קשה

C++ נראית כשפה קשה מאוד. אני מאמין שזה משום שרוב המתכנתים לומדים כאשר ההתמקדות היא ב**תחביר** (syntax) של השפה (באילו מלים תשתמש?) במקום **בסמנטיקה** (semantics) (מה אתה מנסה לומר?)

מונחון

תחביר (Syntax) – השימוש הנאות במונחים ובפיסוק.
סמנטיקה (Semantics) – המשמעות של הקוד ומטרתו.

יש שתי דרכים ללמוד לדבר שפה זרה: דרך אחת היא לשנן עשרות מלים של אוצר מלים ולתרגל הטיית פעלים, והדרך השנייה היא לנסוע לארץ ולתקשר עם דוברי השפה. אנשים שונים לומדים בדרכים שונות, אבל אני יכול לומר לך שלפי ניסיוני, שבוע בצרפת שווה שנתיים בכיתה.

אם אני הייתי עומד ללמד אותך C++ והיינו עובדים יחד, בכלל לא הייתי נותן לך ספר. הייתי יושב איתך והיינו כותבים ביחד תוכנית. בדרך, הייתי מלמד אותך את מה שאתה צריך לדעת, ומידי פעם הייתי נותן לך קטעים קצרים לקריאה, כדי שתעמיק את הבנתך.

כך בדיוק עובד ספר זה: אנחנו נשב ביחד ונכתוב תוכנית, ובדרך אני אלמד אותך את מה שאתה צריך לדעת. מהעמוד הראשון, נתמקד בהבנת הבעיה שאנחנו מנסים לפתור, ובתכנון הפתרון, במקום להתמקד בתחביר של השפה.

על התקליטון המצורף

התקליטון המצורף מכיל את קטעי הקוד הכתובים בספר. שמות הקבצים כפי שהם מופיעים בתקליטון כתובים בסוגריים ליד מספרי התדפיס.

כדי להתקין את התקליטון הכנס אותו לכונן התקליטונים והקלד A:\setup. תוכנית ההתקנה תתקין את הקבצים לספריה C:\CPP-Practice. קובצי הקוד נמצאים מתחת לספריה הנ"ל בספריות משנה הנושאות את מספרי הפרקים.

הידור הקוד

הקוד בספר זה צריך לעבוד היטב עם כל מהדר (compiler) תואם ANSI/ISO. כתבתי את כל תוכניות הדוגמה ב-Microsoft Visual C++ 6.0 במחשב פנטיום II 300 Mhz עם 256 MB של RAM . אני ממליץ **מאוד** שתקנה את המהדורה החדשה ביותר של מהדר C++ 32 ביט טוב כלשהו לפני שתנסה להריץ את הקוד בספר זה, כיוון שהשפה השתנתה משמעותית בשנים האחרונות.

לרוע המזל, אינני יכול לתמוך במהדר האינדיבידואלי שלך, אבל אם הוא תואם את מפרט השפה, לא תהיה לך שום בעיה.

מוסכמות בספר זה

כמה מהמאפיינים הייחודיים בסדרה זו כוללים

מונחון

מונחון – סמל בשוליים המציין שימוש במונח חדש. מונחים חדשים יופיעו בפסקה באותיות מודגשות.

כיצד מבטאים?

כיצד מבטאים – תראה סמל בשוליים שליד תיבה המכילה מונח טכני וכיצד מבטאים אותו. לדוגמה, "cin מבוטא **סי-אין**, ו-cout מבוטא **סי-אאוט**"

סמל כזה בשוליים מציין קוד שאפשר להקליד, להדר ולהריץ אותו.

סיורים

סיורים הם סטיות מהנושא העיקרי הנדון, והם מהווים הזדמנות להעמיק את הבנתך בנושא.

בספר מסוג זה, ייתכן שאתייחס לנושא בכמה מקומות, על פי הזמן והמיקום שבו נוסיף פונקציונליות במהלך פיתוח היישום. כדי לעזור ולהבהיר, הוספנו רשת מושגים המספקת ייצוג גרפי לאופן שבו כל מושגי התכנות מתקשרים זה לזה. תמצא אותה בכריכה הפנימית הקדמית של ספר זה.

הערות מציעות הראות והערות צדדיות על הנושא הנדון, כמו גם הסבר מלא על תפיסות מסוימות.

טיפים מספקים קיצורי דרך נהדרים ורמזים כיצד לתכנת בשפת C++ ביתר יעילות.

אזהרות מסייעות לך להימנע ממלכודות של תכנות, ובכך מונעות ממך לעשות טעויות שתמררנה את חייך.

בנוסף, תמצא כמה מוסכמות טיפוגרפיות בספר זה:

- פקודות, משתנים וקוד אחר יופיעו בטקסט בגופן מיוחד (computer).
- בספר זה, אני בונה על תדפיסים קיימים בעודנו ממשיכים לבחון את הקוד. כשאוסיף קטעים חדשים לקוד קיים, תבחין בו באמצעות גופן מודגש. (computer)
- פקודות ודברים שעליך להקליד יופיעו באותיות מודגשות.
- מצייני מיקום בתיאורי תחביר יופיעו בגופן נטוי (computer). זה מציין שתחליף את מציין המיקום בשם הקובץ, בפרמטר או בכל אלמנט אחר אותו הוא מייצג.

שבירת הקוד

בכמה מקרים, כאשר תתבונן בקוד, תבחין שכמה שורות קוד התפצלו לשתיים, ושליד מספרי השורות יש אותיות. לדוגמה, ראה שורות 10 ו-10a:

```
10:     if ( ( someCondition || someSecondCondition ) &&
10a:        ( someAlternative || someOtherAlternative ) )
```

מה שקרה כאן זה שפיצלתי שורה יחידה של קוד כי היא היתה ארוכה מכדי שתיכנס בשורה אחת בספר זה. השכתוב עדיין חוקי בשפת C++, וניתן להקליד אותו בדיוק כמו שהוא כתוב (ללא מספרי השורות, כמובן.)

האות a מסמנת לך שבדרך כלל הייתי כותב את שתי השורות הללו בשורה אחת. אם זו היתה שורה אחת ארוכה, היא היתה נראית כך:

```
10: if ( (someCondition || someSecondCondition ) && ( someAlternative || someOtherAlternative ) )
```

במקרים רבים, עלי לבצע התאמות קוד כדי לפצל את השורה כך שתמשיך להיות בשפת C++ חוקית. לדוגמה

```
cout << "This is line 10 and you would think this would be one long line of
code" << endl;
```

כדי לפצל את שורת הקוד הזאת, אני חייב לסיים את משפט ה-cout ולהוסיף הצהרה חדשה בשורה שבאה אחריה:

```
10: cout << "This is line 10 and you would think this would be ";
10a:cout << "one long line of code" << endl;
```

ושוב, הקוד שנוצר הוא חוקי, אך לא כפי שהייתי כותב אותו כרגיל.

פרק 2: מתחילים

בפרק זה

- מה גודלו של פרויקט קטן?
- חילוץ הידע שלך
- יצירת הפרויקט
- בדיקת הקוד
- חיפוש שגיאות בקוד
- ניתוח הקוד
- שימוש במשתנה מטיפוס integer

בתוכנית פשוטה יחסית כמו !Decryptix, המטרה הראשונה שלי היא ליצור גרסה שתוכל לרוץ - ושתמשיך לרוץ. לאחר שתעבוד, אוסיף לה תכונות, ואעצב אותה מחדש באופן שוטף ככל שאתקדם.

בפרויקט גדול, זה יכול להיות תהליך בלתי יעיל בצורה קטלנית. כשמוסיפים תכונות, המורכבות של הפרויקט הכללי גדלה, וללא עיצוב טוב תקבל בסוף קוד שקשה לתחזק אותו.

עם זאת, בפרויקט קטן יותר כמו !Decryptix הסיכון מינימלי. אם תכונה חדשה דורשת עיצוב מחדש ושכתוב כוללים של הקוד, אין בעיה. מלכתחילה נדרשו ימים ספורים בלבד כדי לכתוב את הקוד.

מה גודלו של פרויקט קטן?

עד כמה קטנה צריכה להיות תוכנית כדי שתוכל לעצב אותה בעודך מתקדם? אני טוען שכל תוכנית שלוקחת לאדם אחד יותר מכמה שבועות לתכנת דורשת תהליך עיצוב קפדני יותר. והנה הסיבה: תוכניות שמתפתחות באופן אורגני, ולא לפי עיצוב, בדרך כלל צריכות לעבור שכתוב מחדש לפחות פעם אחת. אם השכתוב מיגע, משתלם לבצע את העיצוב מראש. עם זאת, אם השכתוב טריוויאלי, לא נאבד דבר אם נצלול ישר פנימה.

מדוע ללמד תהליך שטוב רק לפרויקטים קטנים?

"מדוע," אני שומע אותך שואל, "להראות דוגמה של עיצוב אורגני אם כל הפרויקטים הגדולים דורשים עיצוב פורמלי?" התשובה ברורה למדי. יש הרבה מה ללמוד הן בתכנות והן בעיצוב. מטרתו של ספר זה היא ללמד תכנות. תמצא ספרים רבים (ביניהם כמה שאני כתבתי) שעוסקים בניתוח ועיצוב מונחי עצמים. לא תוכל ללמוד את הכל בבת אחת.

אין דבר בספר זה שאינו עקבי עם עיצוב טוב, פשוט לא נשקיע זמן בעיצוב כל דבר מראש. אני לא יודע מה אתך, אבל אני משתוקק לצלול לתוך הקוד.

חילוץ הידע שלך

בספר קלאסי למתחילים בשפת C++, הייתי מתחיל במבנה של התוכנית, מציג הצהרות וביטויים, מוסיף משתנים וקבועים, ואז עובר למחלקות. הייתי מפתח מיומנות על גבי מיומנות, ורק אחרי בערך 600 עמודים בספר היית יכול להתחיל לכתוב את תוכנית Decryptix! שלך.

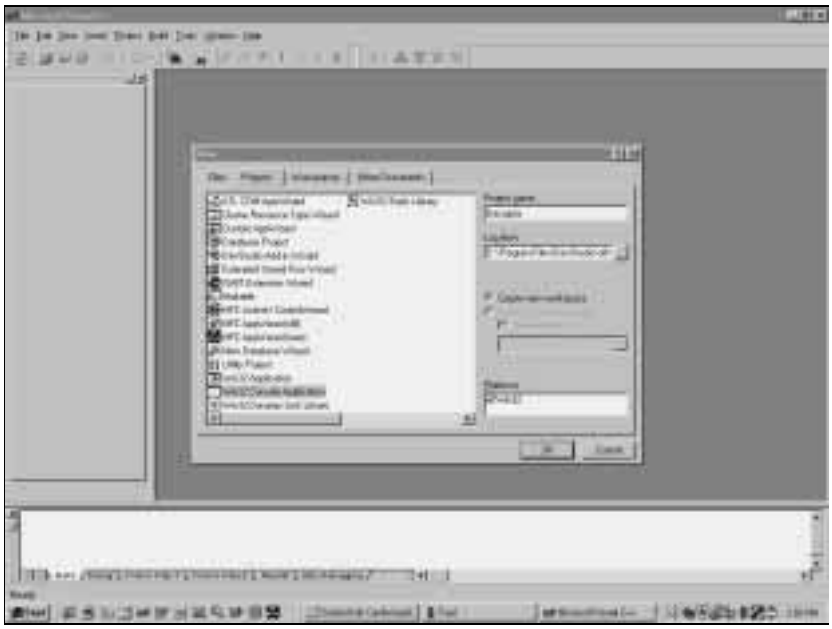
ספר זה שונה. אתה תקפוץ ישר פנימה ותשכשך מעט במים. לא הכל יהיה מובן בהתחלה, ואני אעבור במהירות על פרטים רבים רק כדי לחזור אליהם מאוחר יותר בספר, אבל הזרימה הנחוצה של התוכנית תוסבר במהירות. מדי פעם תצא ל"סיור" לתחום C++ קשור, אך לא לחלוטין רלבנטי.

יצירת הפרויקט

ספר זה תוכנן לשמש אותך, ללא התחשבות בסוג המהדר בו תשתמש או בפלטפורמה (לדוגמה, **חלונות** או **מקינטוש**) עבורה אתה מפתח. עם זאת, מדי פעם אדגים כיצד תוכל לבצע משימה מסוימת בכלי Microsoft Visual C++ 6.0. המהדר שלך עשוי להיות שונה במקצת, אך העקרונות זהים. בעזרת הידע שמסופק לך כאן תוכל בקלות לקרוא את התיעוד של המהדר שלך ולבצע את ההתאמות הנחוצות.

אתחיל ביצירת פרויקט. בכוונתי שבו התקנתי את המהדר שלי יצרתי ספרייה שנקראת Decryptix Projects. ספרייה זו תאכסן את כל הגרסאות של התוכנית שאצור.

ראשית אפעיל את Visual C++ ואומר לה ליצור יישום מסוף Win32 (Win32 Console Application) חדש שנקרא Decryptix, בדומה למוצג באיור 2.1.



איור 2.1

פרויקט חדש של Microsoft Visual C++.

אם המהדר שלך מציע **אשף** (wizard) (סדרה של תיבות דו-שיח שעוזרות לך לבצע את ההחלטות האלה), בחר באפשרות כלשהי שמספקת לך סביבת ISO סטנדרטית הפשוטה ביותר, מבוססת על טקסט וחסרת חלונות. במקרה זה, בחרתי באפשרות empty application (יישום ריק).

לאחר יצירת הפרויקט, Visual C++ מורידה אותי בסביבת הפיתוח המשולבת (Integrated Development Environment) (IDE). מתוך התפריט File (קובץ) אני בוחר בפקודה New (חדש) ונכנס לקובץ מקור חדש של C++ שנקרא Decryptix.cpp.

בסביבות אחרות, לדוגמה, בעורך הטקסט ב-Unix, אני פשוט אפתח קובץ חדש ואשמור אותו תחת השם Decryptix.cpp. לעתים קרובות, בסביבות פיתוח משולבות, שמירת הקובץ עם סיומת CPP מסמנת שזהו קוד מקור בשפת C++ ומפעילה תמיכה בכניסות (הזחות) קוד מקור (ולפעמים גם טקסט מקודד בצבעים!). תמיכת כניסות קוד מקור גורמת לכך שכאשר תקליד את קוד המקור, העורך יערוך את כניסות הטקסט עבורך. וכך, אם תקליד

```
if ( someValue > thisValue)
```

ולאחר מכן תקיש Enter, העורך יכניס אוטומטית את השורה הבאה. (אל תדאג מה עושה הקוד הזה, הכל יוסבר בהמשך).

כדי ללמוד איזו תמיכה מספק העורך שלך, בדוק נא את התיעוד שמגיע עם המהדר שלך.

בדיקת הקוד

כעת נתבונן בגרסה מקדמית של !Decryptix, בתדפיס 2.1. תוכל לפתוח קובץ בפרויקט שלך, לשמור אותו בשם Decryptix.cpp ואז להקליד את הקוד הזה, בדיוק כפי שהוא מוצג.

אני ממליץ מאוד שתקליד את כל קוד המקור בעצמך כי זאת הדרך הטובה ביותר ללמוד. עם זאת, אם אתה לא מסוגל לשאת את המחשבה על ההקלדה המרובה, תוכל למצוא את הקוד בתקליטור שמלווה ספר זה, או שתוכל להוריד קוד זה – ואת כל הקודים שבספר – מאתר האינטרנט שלי (גלוש לכתובת www.libertyassociates.com ולחץ על Books & Resources).

תוכנית זו מתקדמת למדי, וכמובן שלא תבין את רוב מה שתקרא. אל תפחד; פרק זה ופרק 3, **זרימת התוכנית** , יסבירו אותה שורה אחר שורה. עם זאת, ייתכן שתקבל רעיון לא רע על מה התוכנית עושה אם פשוט תקרא אותה כפרוזה.

נסה להריץ אותה ולבחון את פעולתה, ונסה להתאים את הקוד לפלט.

תדפיס 2.1 הצצה ראשונה בתוכנית !Decryptix

```
0: #include <iostream>
1:
2: int main()
3: {
4:    std::cout << "Decryptix. (c)Copyright 1999 Liberty ";
5:    std::cout << "Associates, Inc. Version 0.2\n " << std::endl;
6:    std::cout << "There are two ways to play Decryptix: ";
7:    std::cout << "either you can guess a pattern I create, ";
8:    std::cout << "or I can guess your pattern.\n\n";
9:
10:   std::cout << "If you are guessing, I will think of a\n ";
11:   std::cout << "pattern of letters (e.g., abcde).\n\n";
12:
13:   std::cout << "On each turn, you guess the pattern and\n";
14:   std::cout << " I will tell you how many letters you \n";
15:   std::cout << "got right, and how many of the correct\n";
```

```
16:   std::cout << "letters were in the correct position.\n\n";
17:
18:   std::cout << "The goal is to decode the puzzle as quickly\n";
19:   std::cout << "as possible. You control how many letters \n";
20:   std::cout << "can be used and how many positions\n";
21:   std::cout << " (e.g., 5 possible letters in 4 positions) \n";
22:   std::cout << "as well as whether or not the pattern might\n";
23:   std::cout << " contain duplicate letters (e.g., aabcd).\n\n";
24:
25:   std::cout << "If I'm guessing, you think of a pattern \n";
26:   std::cout << "and score each of my answers.\n\n" << std::endl;
27:
28:   const int minLetters = 2;
29:   const int maxLetters = 10;
30:   const int minPositions = 3;
31:   const int maxPositions = 10;
32:
33:   int     howManyLetters = 0, howManyPositions = 0;
34:   bool        duplicatesAllowed = false;
35:   int     round = 1;
36:
37:   std::cout << "How many letters? (";
38:   std::cout << minLetters << "-" << maxLetters << "): ";
39:   std::cin >> howManyLetters;
40:
41:   std::cout << "How many positions? (";
42:   std::cout << minPositions <<"-"<< maxPositions << "): ";
43:   std::cin >> howManyPositions;
44:
45:   char    choice;
46:   std::cout << "Allow duplicates (y/n)? ";
47:   std::cin >> choice;
48:
49:   return 0;
50:}
```

הדר, קשר והרץ את התוכנית. במערכת Visual C++ תוכל לעשות את כל הפעולות האלה בבת אחת על ידי הקשה על Ctrl+F5. הנה הפלט:

פלט

```
Decryptix. (C)Copyright 1999 Liberty Associates, Inc. Version 0.2
 There are two ways to play Decryptix:
   either you can guess a pattern I create,
 or I can guess your pattern.

If you are guessing, I will think of a
pattern of letters (e.g., abcde).

On each turn, you guess the pattern and
  I will tell you how many letters you
got right, and how many of the correct
 letters were in the correct position.

The goal is to decode the puzzle as quickly
as possible. You control how many letters
can be used and how many positions
 (e.g., 5 possible letters in 4 positions)
as well as whether or not the pattern might
 contain duplicate letters (e.g., aabcd).

If I'm guessing, you think of a pattern
and score each of my answers.

How many letters? (2-10):
```

ניתוח הקוד

השורה הראשונה בתוכנית זו (שורה 0) היא

```
#include <iostream>
```

מטרת שורה זו היא להוסיף לקובץ הנוכחי שלך את המידע הדרוש לו כדי לתמוך **ברצף קלט פלט** (Input and Output Streaming): היכולת לקרוא מהמקלדת (קלט) (input) ולכתוב על המסך (פלט) (output).

מונחון

רצף קלט – האופן בו הנתונים נכנסים לתוכנית שלך; בדרך כלל מהמקלדת.
רצף פלט – האופן בו הנתונים עוזבים את התוכנית שלך; בדרך כלל לתצוגה.

וכך זה עובד: C++ מכילה כעת קבוצה של קוד תומך שנקראת ספרייה סטנדרטית, (standard library) שמאפשרת לאובייקטים לטפל בקלט ופלט. cin הוא אובייקט שמטפל בפלט מהמקלדת, ו-cout הוא אובייקט שמטפל בפלט למסך. פרטי אופן פעולתם אינם חשובים בשלב זה, אבל כדי להשתמש בהם עליך לכלול בתוכנית שלך את הקובץ iostream שמספק את ההגדרות שלהם. הגדרה של אובייקט אומרת למהדר מה שהוא צריך לדעת כדי שניתן יהיה להשתמש באובייקט.

כיצד מבטאים? cin מבוטא "סִי-אִין", ו-cout מבוטא "סִי-אָאוּט".

אתה כולל את הקובץ הזה בתוכנית שלך עם הביטוי #include . כאשר יופעל המהדר שלך, הקדם-מהדר (precompiler) ירוץ, יקרא את התוכנית שלך ויחפש שורות שמתחילות בסמל #. כאשר יראה #include הוא ידע שעליו לקרוא בתוך קובץ. הסוגרים הזוויתיים (< ו->) אומרים "הסתכל במקום הרגיל." כשהתקנת את המהדר שלך, הוא אמור היה להתקין את ה"מקום הרגיל" שבו יחפש קבצים אלה.

כיצד מבטאים? אני קורא לסימן # סולמית, ומבטא את שורת הקוד "סולמית כלול אי-אוֹ-סטרים".

האפקט נטו הוא שהקובץ iostream נקרא בשלב זה לתוך התוכנית, וזה בדיוק מה שאתה רוצה. תוכל עכשיו להשתמש באובייקט cout, כפי שתראה בעוד מספר רגעים.

השימוש בסוגריים הזוויתיים, <iostream> מציין שהקדם-מהדר המקדים צריך "לחפש במקום הרגיל". לחילופין אפשר להשתמש בגרשיים כפולים - לדוגמה "myfile.h" - שאומר "חפש בספריית הפרויקט הנוכחית, ואם לא תמצא, חפש במקום הרגיל."

תחומי השם

בשונה מהקוד בפרק 1, **מבוא לשפת C++**, גרסה זו משתמשת בקובץ הכותרת החדש של הספרייה הסטנדרטית ANSI/ISO <iostream>, במקום ב-<isotream.h> (שים לב שהכותרת החדשה אינה משתמשת ב-h.).

כותרות אלו תומכות בפרוטוקולי תחומי השם (namespace) החדשים, שמאפשרים להימנע מהתנגשויות בשמות אובייקטים ושיטות כאשר תעבוד עם קוד של ספקים אחרים. לדוגמה, ייתכן שיהיו שני אובייקטים שנקראים cout. אנו פותרים בעיה זו על ידי "הכשרת" השם ב-std::, כמוצג בשורות 4-26. הכשרה זו מכוונת את המהדר להשתמש באובייקט cout שמוגדר בספרייה הסטנדרטית (std), שמגיעה עם המהדר. לרוע המזל, זה הופך את הקוד למורכב יותר וקשה לקריאה.

השימוש ב-namespace std

כדי לפשט את הקוד הזה וכדי להקל עלינו להתמקד בנושאים שמעניינים אותנו, אשכתב את הדוגמה הקודמת על ידי הוספת מילות המפתח

```
using namespace std;
```

זה מסמן למהדר שהקוד שאני כותב נמצא בתחום השמות של הספרייה הסטנדרטית. למעשה, זה אומר למהדר שכאשר יראה cout עליו להתייחס אליו כאל std::cout.

יתר הקוד בספר זה משתמש בתכסיס זה, שהופך את הקוד לקל יותר לקריאה ולמעקב, אך המחיר הוא ערעור ההגנה שמספקים תחומי השמות.
כשתכתוב יישומים מסחריים משלך ייתכן שתרצה להימנע משימוש בצירוף המילים using namespace std כי ייתכן שתהיה מעוניין לוודא שתהיה הגנה של תחום השם.

תדפיס 2.1a הוא העתק מדויק של תדפיס 2.1, אך הוא משתמש בצירוף המילים using namespace.

תדפיס 2.1a using namespace std (List 0201)

```
0: #include <iostream>
1: using namespace std;
2: int main()
3: {
4:    cout << "Decryptix. (c)Copyright 1999 Liberty ";
5:    cout << "Associates, Inc. Version 0.2\n " << endl;
6:
7:    cout << "There are two ways to play Decryptix: ";
8:    cout << " either you can guess a pattern I create, ";
9:    cout << "or I can guess your pattern.\n\n";
10:
11:   cout << "If you are guessing, I will think of a\n ";
12:   cout << "pattern of letters (e.g., abcde).\n\n";
13:
14:   cout << "On each turn, you guess the pattern and\n";
15:   cout << " I will tell you how many letters you \n";
16:   cout << "got right, and how many of the correct\n";
17:   cout << " letters were in the correct position.\n\n";
18:
19:   cout << "The goal is to decode the puzzle as quickly\n";
```

```
20:   cout << "as possible. You control how many letters \n";
21:   cout << "can be used and how many positions\n";
22:   cout << " (e.g., 5 possible letters in 4 positions) \n";
23:   cout << "as well as whether or not the pattern might\n";
24:   cout << " contain duplicate letters (e.g., aabcd).\n\n";
25:
26:   cout << "If I'm guessing, you think of a pattern \n";
27:   cout << "and score each of my answers.\n\n" << endl;
28:
29:   const int minLetters = 2;
30:   const int maxLetters = 10;
31:   const int minPositions = 3;
32:   const int maxPositions = 10;
33:
34:   int     howManyLetters = 0, howManyPositions = 0;
35:   bool        duplicatesAllowed = false;
36:   int     round = 1;
37:
38:   cout << "How many letters? (";
39:   cout << minLetters << "-" << maxLetters << "): ";
40:   cin >> howManyLetters;
41:
42:   cout << "How many positions? (";
43:   cout << minPositions << "-" << maxPositions << "): ";
44:   cin >> howManyPositions;
45:
46:   char    choice;
47:   cout << "Allow duplicates (y/n)? ";
48:   cin >> choice;
49:
50:   return 0;
51: }
```

חיפוש שגיאות בקוד

אחת מהדרכים רבות העוצמה ביותר ללמוד C++ היא להשתמש ב**מנפה השגיאות** (debugger) שלך. אני ממליץ מאוד שמיד לאחר שתקליד את הקוד הזה לתוך הפרויקט שלך (או אחרי שתוריד אותו מהאתר שלי), תהדר, תקשר ותריץ אותו. תצטרך לבדוק את התיעוד שלך כדי לראות איך עושים את זה, אבל ברוב סביבות הפיתוח המשולב המודרניות יש אפשרות תפריט ל"בניית הפרויקט כולו" (Build the entire project).

אם אתה משתמש במערכת Visual C++ תוכל פשוט להצביע עם הסמן שלך על הלחצנים שבסרגל הכלים עד שתמצא את הלחצנים שמהדרים ומקשרים, או את הלחצן שבונה את הפרויקט כולו.

לאחר שזה עובד, שים את הספר הזה בצד וקרא את התיעוד של מנפה השגיאות שלך, אותו תמצא בתיעוד המהדר שלך. הכנס **נקודת עצירה** (break point) בשורה הראשונה של הקוד ב-`main()` (ראה שורה 5 בתדפיס 2.1). במערכת Visual C++ פשוט הצב את הסמן באותה שורה והקש F9, או לחץ על הלחצן break point (נקודת עצירה) שעל סרגל הכלים. לאחר שהצבת את נקודת העצירה, הרץ עד לנקודת העצירה (במערכת Visual C++, הקש F5). צעד מעבר לכל שורת קוד ונסה לנחש מה קורה. שוב, תצטרך לבדוק את התיעוד שלך כדי לראות איך צועדים מעבר לכל שורת קוד (במערכת Visual C++ מקישים על F10.)

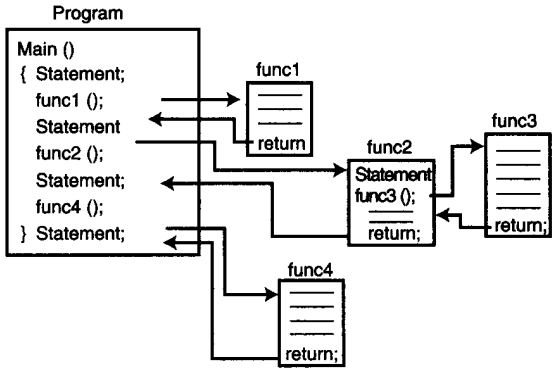
מנפה השגיאות הוא אחד הדברים האחרונים שרוב הספרים למתחילים מציגים. לדעתי הוא צריך להיות אחד הדברים **הראשונים** שעליך ללמוד. אם תתקע, עיין בהסבר על ניפוי שגיאות שמופיע בסוף פרק זה.

בכל תוכנית C++ יש פונקציה `main()` (תדפיס 2.1, שורה 2). המטרה הכללית של פונקציה היא להריץ קטע קוד קצר ולחזור למי שקרא לה.

מונחון

כל הפונקציות מתחילות ומסתיימות בסוגריים מסולסלים, כפי שתראה בשורות 3 ו-51. **פונקציה** מכילה סדרת משפטים, שהם כל השורות שמופיעות בין הסוגריים.

זוהי התמצית של תוכנית מובנית. זרימת התוכנית ממשיכה בסדר שבו מופיע הקוד בקובץ עד שקוראים לפונקציה. הזרימה מתפצלת לפונקציה ועוקבת אחריה שורה אחר שורה, עד שקוראים לפונקציה אחרת או עד שהפונקציה חוזרת (ראה איור 2.2).



איור 2.2

כאשר תוכנית קוראת לפונקציה, הביצוע עובר לפונקציה וממשיך בשורה שלאחר הקריאה לפונקציה.

במובן מסוים, פונקציה היא תת-תוכנית. בכמה שפות היא נקראת **סברוטינה** (subroutine) או **פרוצדורה** (procedure). תפקידה של הפונקציה לבצע עבודה מסוימת, ולהחזיר את השליטה למי שקרא לפונקציה.

כאשר main() מתבצעת, אנחנו מבצעים את Statement1. לאחר מכן אנחנו מתפצלים לשורה הראשונה של Func1(). שלוש השורות של Func1 מתבצעות, והבקרה חוזרת ל-main(), שם אנחנו מבצעים את Statement2. מתבצעת קריאה לפונקציה Func2, שבתורה קוראת לפונקציה Func3(). כאשר Func3 מסיימת לפעול היא חוזרת לפונקציה Func2(), שממשיכה לרוץ עד שמגיעה למשפט return שלה, ואז אנחנו חוזרים ל-main() ומבצעים את Statement3. לאחר מכן אנחנו קוראים לפונקציה Func4(), אשר מבצעת את הקוד שלה וחוזרת ל-main(), שם מתבצע Statement4.

החזרת ערך

כאשר פונקציה חוזרת למי שקרא לה, היא יכולה להחזיר ערך. בהמשך תראה מה הפונקציה הקוראת יכולה לעשות עם אותו ערך.

כל פונקציה מוכרחה להצהיר איזה סוג של ערך היא מחזירה: לדוגמה, האם היא מחזירה מספר שלם או תו? אם הפונקציה אינה מחזירה ערך, היא מצהירה על עצמה כמחזירה void, כלומר, שאינה מחזירה דבר.

main() שווה יותר מאחרות

main() היא פונקציה חשובה בשפת C++. כל התוכניות בשפת C++ מתחילות ב-main(); כאשר main מסתיימת, התוכנית מסתיימת. במובן מסוים, מערכת ההפעלה (חלונות, דוס וכו') קוראת ל-main().

main() תמיד מחזירה int (מספר שלם) (integer). אדון בטיפוסים שונים של ערכים מאוחר יותר בספר. לעת עתה מספיק לדעת שאתה תמיד חייב להצהיר ש-main() מחזירה מספר שלם.

בחלק מהמהדרים הישנים יותר, main() יכולה להחזיר void, אך זה לא חוקי לפי סטנדרט ISO החדש. מומלץ להתרגל לכך ש-main() תחזיר תמיד int.

שים לב ש-main() מחזירה מספר שלם (במקרה הזה, 0) בשורה 50. כשמריצים תוכניות מ**קובצי אצווה** (batch files) או **סקריפטים** (scripts), אפשר לבחון את הערכים האלה. בתוכניות בספר זה (וכנראה ברוב התוכניות שתכתוב), לא נעשה שימוש בערך זה. לפי המוסכמות, תחזיר 0 כדי לציין שהתוכנית רצה ללא תקלות.

שימוש בפקודה cout להדפסה למסך

רוב המשפטים בתוכנית ראשונה זו תוכננו להיכתב על המסך. השתמש באובייקט הפלט הסטנדרטי cout. אתה שולח מחרוזת של תווים לאובייקט cout על ידי הכנסתם בין גרשיים ושימוש באופרטור של הכוונה מחדש של הפלט (output redirection operator) (<<), אותו תיצור על ידי החזקת המקש shift לחוץ תוך כדי שתי הקשות על המקש **ת** (או פסיק באנגלית).

פעולה זו למעשה מנצלת את התכונה המתקדמת מאוד בשפת C++ שנקראת **העמסת אופרטורים** (operator overloading), בה נדון בפירוט בפרק 6, **שימוש ברשימות מקושרות**. למרבה המזל, לעת עתה תוכל להשתמש בתכונה זו מבלי להבין אותה לגמרי. התוצאה הסופית היא שהמילים

```
Decryptix. (C)Copyright 1999 Liberty
```

נשלחות למסך.

העמסת אופרטורים (operator overloading) - היכולת של טיפוסים שנוצרו על ידי המשתמש להשתמש באופרטורים שטיפוסים מובנים משתמשים בהם, כגון +, = או ==. אסביר כיצד לעשות זאת בפרק 6.

תווי הדפסה מיוחדים

שורה 5 מדפיסה את המילים

```
Associates, Inc. Version 0.2
```

למסך. שים לב שלפני הגרשיים הסוגרים, שורה 5 מכילה \n. אלה שני סימנים מיוחדים בתוך מחרוזות נתונות בגרשיים. הקו הנטוי (slash) נקרא **תו חילוף** (escape character), וכאשר הוא נמצא במחרוזת עם גרשיים, משמעותו "מה שיבוא אחרי הוא הוראה מיוחדת למהדר." כשהאות n מופיעה אחרי תו חילוף, משמעותה "שורה חדשה" (new line). וכך, מה שקורה הוא הדפסה של שורה חדשה, אל הקלט.

מונחון

תו חילוף (escape character) – (תו שמשמש כסימן למהדר או לקדם-מהדר, שהאות שבאה אחריו דורשת טיפול מיוחד. לדוגמה, הקדם-מהדר מתייחס לתו n כאל אות, אך כאשר יש לפניו תו חילוף (\n), הוא מציין שורה חדשה.

כמו כן, שים לב ששורה זו מסתיימת במחרוזת

```
<< endl;
```

cout יכול לקבל יותר מאשר סתם מחרוזות. במקרה זה, אופרטור ההכוונה (redirection operator) משמש לשליחת endl.

כיצד מבטאים?

Endl מבטאים אנד-אל ומשמעותו "סוף שורה" (end line).

זה שולח שורה חדשה נוספת לפלט ומנקה את **המאגרים** (buffers). נסביר מאגרים מאוחר יותר, כשאדבר על ערוצים, אך ההשפעה הכללית מוודאת שכל הטקסט נכתב למסך באופן מידי.

שורה 7 מתחילה להדפיס שורה נוספת, שממשיכה בשורה 8 ומסתיימת בשורה 9.

ביחד, שורות אלה מדפיסות את הפלט הבא:

```
Decryptix,(c)Copyright 1999 Liberty Associates, Inc. Version 0.2

There are two ways to play Decryptix: either you can guess a pattern I
create, or I can guess your pattern.
```

שים לב שאין שורה חדשה אחרי Liberty ולפני Associates. לא היתה הוראה ל-cout להדפיס שורה חדשה, ולכן לא הודפסה שורה חדשה. שתי השורות החדשות מופיעות אחרי 0.2. הראשונה, שנוצרה על ידי התו \n, מסיימת את השורה; השנייה שנוצרה על ידי endl, מדלגת שורה.

תוכל להשיג את האפקט של דילוג שורה על ידי הכנסת שני תווי \n, בדומה למוצג בשורה 9.

טבלה 2.1 מתארת את תווי ההדפסה המיוחדים האחרים.

טבלה 2.1 תווי הדפסה מיוחדים

תו	משמעותו
\n	שורה חדשה
\t	טאב
\b	מצלצל בפעמון
*	מדפיס גרשיים כפולים
\'	מדפיס גרש יחיד
\?	מדפיס סימן שאלה
\\	מדפיס לוכסן אחורי (backslash)

משתנים

משתנה (variable) הוא מקום לאכסון ערך במהלך התקדמות התוכנית שלך.

מונחון

משתנה – מקום לאכסון ערך.

במקרה זה, בשורה 36, תרצה לשמור מעקב על מספר המשחק שאליו הגעת. אחסן מידע זה במשתנה ששמו round:

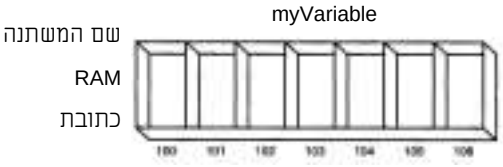
```
int round = 1;
```

דרך אחת לחשוב על זיכרון המחשב שלך הוא כעל סדרת תאים. כל תא בגודל **בית** (byte) אחד, וכל בית ממוספר ברצף: המספר הוא הכתובת של אותו תא זיכרון. כל משתנה מקצה בית אחד או יותר שבהם תוכל לאחסן ערך.

שמו של המשתנה שלך (round) הוא תווית על אחד מאותם תאים, שמאפשר לך למצוא אותו בקלות מבלי לדעת את כתובת הזיכרון הממשית שלו.

חשוב על זה כך: כשאתה קופץ לתוך מונית בוושינגטון, תוכל לבקש לרדת בשדרות פנסילבניה מס' 1600, או שתוכל לבקש לרדת בבית הלבן. **המזהה** (identifier) "הבית הלבן" הוא השם של אותה כתובת.

איור 2.3 הוא ייצוג סכימטי של רעיון זה. כפי שתראה באיור, round מתחיל בכתובת הזיכרון 103. round יכול לתפוס כתובת זיכרון אחת או יותר, בהתאם לגודלו.



איור 2.3

ייצוג סכימטי של הזיכרון.

מונחון

RAM – **זיכרון לגישה אקראית** (Random Access Memory). כשאתה מריץ את התוכנית שלך, היא נטענת מקובץ הדיסק ל-RAM. גם כל המשתנים נוצרים ב-RAM. כאשר מתכנתים מדברים על זיכרון, הם בדרך כלל מתייחסים ל-RAM.

הקצאת זיכרון

כשאתה מגדיר משתנה בשפת C++, עליך לומר למהדר על איזה סוג משתנה אתה מצהיר: משתנה מטיפוס int, char וכו'. **הטיפוס** (type) אומר למהדר את גודל המשתנה. לדוגמה, char הוא בגודל בית אחד, ובמחשבים מודרניים int הוא 4 בתים. לכן המשתנה round צורך ארבעה בתים (תאים) של זיכרון.

הגדרת משתנה

אתה מגדיר משתנה על ידי הצהרה על הטיפוס שלו, רווח אחד או יותר, שם המשתנה, ונקודה-פסיק:

```
int round;
```

למעשה, שם המשתנה יכול להיות כל צירוף של אותיות, אבל הוא אינו יכול להכיל רווחים. שמות משתנים חוקיים הם x, J23qrsnf, ו-myAge. זהו הרגל תכנות טוב להשתמש בשם משתנה שאומר לך למה משמש המשתנה . זה הופך אותם לקלים יותר להבנה, וזה מקל עליך לתחזק את התוכנית שלך.

רגישות לרישיות אות

C++ רגישה לרישיות אות, ולכן משתנה ששמו round שונה מ-Round, שהוא שונה מ-ROUND. הימנע מלהשתמש במשתנים מרובים ששמותיהם שונים רק ברישיות האותיות - זה יכול להיות מבלבל מאוד.

כמה מהדרים מאפשרים לך לכבות את הרגישות לרישיות אות. אל תתפתה לעשות זאת. התוכניות שלך לא תעבודנה עם מהדרים אחרים, והקוד שלך יבלבל מאוד מתכנתי C++ אחרים.

מילות מפתח

C++ משמרת כמה מילים, ולא תוכל להשתמש בהן כשמות משתנים. אלה הם מילות מפתח המשמשות את המהדר לבקרה על התוכנית שלך. מילות מפתח כוללות if, while, for ו-main. מדריך המהדר שלך מספק כנראה רשימה מלאה, אבל בדרך כלל, סביר שכל שם הגיוני למשתנה, אינו מילת מפתח.

יצירת כמה משתנים בעת ובעונה אחת

תוכל ליצור כמה משתנים מאותו טיפוס במשפט אחד על ידי כתיבת הטיפוס ושמות המשתנים, מופרדים בפסיקים. לדוגמה

```
int howManyLetters, howManyPositions;
bool valid, duplicatesAllowed;
```

הקצאת ערכים למשתנים שלך

בתדפיס 2.1, בשורה 35, משתנה מקומי מוגדר על ידי הצהרת הטיפוס (int) ושם המשתנה (round).

זה למעשה מקצה זיכרון עבור המשתנה. כיוון שמשתנה מטיפוס int הוא בן ארבעה בתים, מוקצים ארבעה בתים של זיכרון. כאשר המהדר מקצה זיכרון, הוא שומר את הזיכרון לשימוש המשתנה שלך ומקצה את השם שסיפקת (במקרה הזה, round).

תחום הכרה

תחום הכרה (scope) מתייחס לאזור של התוכנית שבה ה**מזהה** (identifier) - משהו שיש לו שם, כמו אובייקט, משתנה, פונקציה או קבוע - תקף. כשאני אומר שמשתנה הוא בעל **תחום הכרה מקומי** (local scope), אני מתכוון שהוא תקף בתוך פונקציה מסוימת.

מונחון

תחום הכרה (scope) – אזור של התוכנית שבה מזהה (כלומר, השם של משהו) תקף.
תחום הכרה מקומי (local scope) – כשלמזהה יש תחום הכרה מקומי, הוא תקף בתוך פונקציה מסוימת.

ישנן רמות תחום הכרה נוספות (גלובלי, חבר סטטי וכו') שבהן אדון כשאתקדם בתוכנית.

הערך של משתנים

משתנים מקומיים, כמו round, הם בעלי ערך כאשר הם נוצרים, בין אם אתחלת אותם ובין אם לאו. אם לא אתחלת אותם (כמוצג כאן), כל מה שנמצא במקרה בפיסת הזיכרון יוקצה להם - כלומר, ערך **זבל** (garbage) אקראי.

הרגל תכנות טוב הוא **לאתחל** (initialize) את המשתנים שלך. כאשר אתה מאתחל משתנה, אתה יוצר אותו וגם נותן לו ערך ספציפי, בשלב אחד:

```
int round =1;
```

זה יוצר את המשתנה round ומאתחל אותו עם ערך 1.

כפי שתוכל ליצור מספר משתנים בבת אחת, כך גם תוכל לאתחל מספר משתנים. לדוגמה,

```
int howManyLetters = 0, howManyPositions = 0;
```

מאתחל שני משתנים howManyLetters ו-howManyPositions עם ערך 0. תוכל גם לערבב הגדרות ואתחולים:

```
int howManyLetters = 0,round, howmanyPositions = 2;
```

דוגמה זו מגדירה שלושה משתנים מטיפוס int, ומאתחלת את הראשון והשלישי.

סיור

תווים

בשורה 46 בתדפיס 2.1 יצרת משתנה תו (טיפוס char) ששמו choice. ברוב המחשבים, משתני תו הם בני בית אחד, וזה מספיק להחזיק 256 ערכים. Char יכול להיות מפורש כמספר קטן (255–0) או כחבר במערכת התווים של ASCII. ASCII משמעותו קוד אמריקאי סטנדרטי לחילופי מידע (American Standard Code for Information Interchange). מערכת התווים של ASCII ושווה הערך של ISO, ארגון התקנים הבינלאומי (International Standards Organization) הם שיטות לקידוד כל האותיות, הספרות וסימני הפיסוק.

מונחון

ASCII – קוד אמריקאי סטנדרטי לחילופי מידע (American Standard Code for Information Interchange)

ISO – ארגון התקנים הבינלאומי (International Standards Organization).

אתה יוצר תו על ידי מיקום האות בין תווי גרש. לכן 'a' יוצר את התו a.
בקוד ASCII לאות הקטנה a מוקצה הערך 97. לכל האותיות הרישיות והאותיות הקטנות, לכל הספרות ולכל סימני הפיסוק מוקצים ערכים בין 1 ו–128. 128 סימנים וסמלים נוספים שמורים לשימוש על ידי יצרן המחשב.
הערכים שמעל 127 משמשים לייצוג תווים ספציפיים לשפות המדוברות במדינות שונות. כך למשל הערכים שבין 128 לבין 154 מייצגים את אותיות הא–ב העברי בשיטה המשמשת במדינת ישראל.

תווים ומספרים

כשתכניס תו – 'a', לדוגמה – לתוך משתנה char, מה שלמעשה יימצא שם זה מספר בין 0 ו–255. עם זאת, המהדר יודע כיצד לתרגם הלוך ושוב בין תווים ובין ערכי ASCII.
הקשר ערך/אות הוא שרירותי, אין סיבה מיוחדת לכך של–a קטנה מוקצה הערך 97. כל עוד כולם (המקלדת, המהדר והמסך שלך) מסכימים, אין בעיה. עם זאת, חשוב להבין שיש הבדל גדול בין הערך 5 והתו '5'. ערכו של התו 5 הוא למעשה 53, כפי שערכה של האות 'a' הוא 97.
תדפיס 2.2 הוא תוכנית פשוטה שמדפיסה את ערכי התווים של המספרים השלמים 127–32. אל תשים לב לפרטי תוכנית זאת – אנחנו נפרט כיצד היא עובדת מאוחר יותר בספר זה.

תדפיס 2.2 הדפסת התווים

```
  #include <iostream>
using namespace std;
int main()
    {
    for (int i = 32; i<128; i++)
        cout << (char) i;
    return 0;
    }
```

פלט

```
!"#$%G'()*+,./0123456789:;<>?@ABCDEFGHIJKLMNOP
_QRSTUVWXYZ[\]^'abcdefghijklmnopqrstuvwxyz<¦>~s
```

המחשב שלך עשוי להדפיס רשימה שונה במקצת.

טיפוסים מובנים

C++ מגיעה מהקופסה עם ידע על מספר **טיפוסים מובנים** (built-in types) פרימיטיביים. הטיפוס של המשתנה או האובייקט מגדיר את גודלו, את תכונותיו ואת יכולותיו.

לדוגמה, גודלו של `int` במהדרים מודרניים הוא 4 בתים. הוא מחזיק ערכים בין 2,147,483,648- לבין 2,147,483,647. למידע נוסף אודות בתים ומדוע 2,147,483,648 הוא מספר עגול, ראה נספח א', **בינארי והקסדצימלי**.

אתה עשוי לחשוב ש-`integer` (שלם) הוא `integer` (שלם), אבל זה לא מדויק. מילת המפתח `integer` מתייחסת לערך בן ארבעה בתים, אך רק אם אתה משתמש במהדר מודרני במחשב 32 סיביות מודרני. עם זאת, אם התוכנה או המחשב שלך הם 16 סיביות, `integer` עשוי להיות בגודל שני בתים בלבד. מילת המפתח `short` בדרך כלל מתייחסת ל-`integer` בן שני בתים, ומילת המפתח `long` בדרך כלל מתייחסת ל-`integer` בן ארבעה בתים, אך אף אחד משני אלה אינו וודאי. השפה דורשת רק ש-`short` יהיה קצר או שווה ל-`integer`, וש-`integer` יהיה קצר או שווה ל-`long`. במחשב שלי, `short` הוא בן 2 בתים, ו-`intger` ו-`long` הם בני 4 בתים.

תקן ISO של C++ מספק את הטיפוסים הרשומים בטבלה 2.2.

טבלה 2.2 טיפוסי משתנים

טיפוס	גודל	ערכים
unsigned short int	2 בתים	0 עד 65,535
short int	2 בתים	-32,768 עד 32,767
unsigned long int	4 בתים	0 עד 4,294,967,295
long int	4 בתים	-2,147,483,648 עד 2,147,483,647
int (16-bit)	2 בתים	-32,768 עד 32,767
int (32-bit)	4 בתים	-2,147,483,648 עד 2,147,483,647
unsigned int (16-bit)	2 בתים	0 עד 65,535
unsigned int (32-bit)	4 בתים	0 עד 4,294,967,295
char	1 בית	256 ערכי תווים
float	4 בתים	1.2e-38 עד 3.4e38
double	8 בתים	2.2e-308 עד 1.8e308
bool	1 בית	true או false

תקן ISO של C++ הוסיף לאחרונה טיפוס חדש, bool, שהוא ערך אמת (true) או שקר (false). Bool נקרא על שם המתמטיקאי הבריטי ג'ורג' בול (1815-1864), שהמציא את האלגברה הבוליאנית, מערכת לוגיקה סימבולית.

גדלים של שלמים

ספר זה מניח שאתה משתמש במחשב 32 סיביות (לדוגמה, פנטיום) ושאתה מתכנת עם מהדר 32 סיביות. בסביבת פיתוח כזאת, גודלו של integer תמיד 4 בתים. תדפיס 2.3 יכול לעזור לך לקבוע את גודל הטיפוסים המובנים במחשב שלך, על ידי שימוש במהדר שלך.

תדפיס 2.3 מציאת גודל של טיפוסים מובנים

```
1:   #include <iostream>
2:   using namespace std;
3:   int main()
4:   {
5:   cout << "The size of an int is:\t\t";
5a:  cout << sizeof(int)    << " bytes.\n";
6:   cout << "The size of a short int is:\t";
6a:  cout << sizeof(short)  << " bytes.\n";
7:   cout << "The size of a long int is:\t";
7a:  cout << sizeof(long)   << " bytes.\n";
8:   cout << "The size of a char is:\t\t";
```

תדפיס 2.3 מציאת גודל של טיפוסים מובנים (המשך)

```
8a:  cout << sizeof(char)   << " bytes.\n";
9:   cout << "The size of a float is:\t\t";
9a:  cout << sizeof(float)  << " bytes.\n";
10:  cout << "The size of a double is:\t";
10a: cout << sizeof(double) << " bytes.\n";
11:  cout <<"The Size of a bool is:\t\t";
11a: cout << sizeof(bool) << " bytes. \n";
12:     return 0;
13:  }
```

פלט

```
The size of an int is:          4 bytes.
The size of a short int is:     2 bytes.
The size of a long int is:      4 bytes.
The size of a char is:          1 bytes.
The size of a float is:         4 bytes.
The size of a double is:        8 bytes.
The size of a bool is:          1 bytes.
```

במחשב שלך, מספר הבתים המוצגים עשוי להיות שונה. אם מספר הבתים שדווחו לגבי int (השורה הראשונה בפלט) הוא 2 בתים, אתה משתמש במהדר 16-סיביות ישן יותר (וכנראה גם מיושן).

אם אתה משתמש במהדר Visual C++, תוכל להריץ את התוכנית שלך על ידי הקשה על Ctrl+F5. הפלט שלך יוצג, ואחריו תופענה המילים

```
Press any key to continue
```

זה נותן לך זמן להביט בפלט. לאחר מכן, כשתקיש על מקש, החלון ייסגר.

אם המהדר שלך אינו מספק שירות זה, אתה עשוי לגלות שהטקסט נגלל מהר מאוד, ואתה חוזר לסביבת הפיתוח המשולבת שלך. במקרה כזה, הוסף את השורה הבאה לראש הקובץ:

```
#include <conio.h>
```

הוסף את השורה הבאה מיד לפני `return 0`:

```
_getch();
```

זה גורם לתוכנית שלך לעצור לאחר שכל הפלט הושלם; היא מחכה שתקיש על מקש הרווח או מקש תו אחר. הוסף שורות אלה לכל תוכנית דוגמה.

ניתוח

רובו של תדפיס 2.3 כנראה מוכר לך למדי. התכונה החדשה היחידה היא השימוש באופרטור sizeof(), המסופק על ידי המהדר שלך ואומר לך את גודלו של האובייקט שאתה מעביר אליו כפרמטר. לדוגמה, בשורה 5, מילת המפתח int מועברת אל sizeof(). על ידי שימוש באופרטור sizeof(), אני קובע שבמחשב שלי int שווה ל-long int, שהוא ארבעה בתים.

שימוש במשתנה integer

במשחק !Decryptix אנו רוצים לעקוב אחר מספר האותיות השונות שהקוד יכול להכיל. כיוון שזהו מספר (בין 1 ל-26), תוכל לשמור מעקב על ערך זה עם `int`. למעשה, כיוון שזהו מספר מאוד קטן, הוא יכול להיות מטיפוס `short int`, ומכיוון שהוא חייב להיות מספר חיובי, הוא יכול להיות מטיפוס `unsigned short int`.

השימוש בטיפוס `unsigned short int` יכול לחסוך לי שני בתים. היתה תקופה שחיסכון כזה היה משמעותי. עם זאת, כיום, אני טורח רק לעתים נדירות. במקום `short int` אפשר לכתוב רק `short`, ולכן זהו שיא הפזרנות לבזבז בתים בגלל שאני עצלן מכדי לכתוב `short` במקום `int`.

כשהייתי ילד, לבתים היה ערך, ושמרתי על כל אחד מהם. היום, עבור רוב היישומים, הבתים זולים. הם האגורות של התכנות, וכיום רוב המתכנתים שמים כוס קטנה של בתים על הדלפק ונותנים ללקוחות לקחת אותם כשהם צריכים כמה, ומידי פעם זורקים בתים מיותרים לכוס, לשימושו של האדם הבא.

בשנות ה-60, מתכנתים רבים עבדו עם שפות פרימיטיביות, אשר ייצגו תאריכים כתווים. כל מספר בתאריך צרך בית אחד (וכך 1999 צרך 4 בתים). דאגה כפייתית לחיסכון בבית פה ושם גרמה למתכנתים רבים לקצר תאריכים מ-4 ספרות (1999) ל-2 ספרות (99), וכך הם חסכו שני בתים ויצרו את בעיית באג 2000.

עבור רוב התוכניות, הטיפוסים המובנים היחידים שצריך להתייחס אליהם הם `int`, `char`, ו-`bool`. פה ושם תשתמש בטיפוסים `unsigned int`, `float` ו-`double`. כמובן, רוב הזמן תשתמש בטיפוסים שנוצרו על ידי המתכנת משל עצמך.

מסומנים ובלתי מסומנים

אחרי שקבעת את גודלו של מספר שלם, עדיין לא סיימת לגמרי. משתנה מטיפוס `integer` (`short` או `long`) יכול להיות **מסומן** (signed) או **בלתי מסומן** (unsigned). אם הוא מסומן, הוא יכול לאחסן מספרים שליליים וחיובים. אם הוא בלתי מסומן, הוא יכול לאחסן רק מספרים חיוביים.

כיוון שמספרים מסומנים יכולים לאחסן מספרים שליליים כמו גם מספרים חיוביים, הערך המוחלט שהם יכולים לאחסן הוא רק חצי בגודלו.

גלישת unsigned integer (שלם לא מסומן)

העובדה שלמשתנים מטיפוס unsigned long integer יש גבול לערך שהם יכולים לאחסן מהווה בעיה רק לעתים נדירות – אבל מה קורה כשבאמת אוזל המקום?

כאשר משתנה מטיפוס unsigned integer מגיע לערך המקסימלי שלו, הוא גולש ומתחיל מהתחלה, כמו מד-מרחק (odometer) במכונית. תדפיס 2.4 מראה מה קורה כשמנסים לשים ערך גדול מדי במשתנה מטיפוס short integer.

תדפיס 2.4 גלישת unsigned integer

```
1: #include <iostream>
2: using namespace std;
3:  int main()
4:  {
5:    unsigned short int smallNumber;
6:    smallNumber = 65535;
7:    cout << "small number:" << smallNumber << endl;
8:    smallNumber++;
9:    cout << "small number:" << smallNumber << endl;
10:  smallNumber++;
11:  cout << "small number:" << smallNumber << endl;
12:  return 0;
13: }

small number:65535
small number:0
small number:1
```

בשורה 4, smallNumber מוצהר כטיפוס unsigned short int, שבמהדר שלי הוא משתנה בן 2 בתים שיכול לאחסן ערך בין 0 ו-65,535. בשורה 5 מוקצה הערך המקסימלי למשתנה smallNumber, והוא מודפס בשורה 6 תוך שימוש בפונקציית הפלט של הספרייה הסטנדרטית.

בשורה 7, smallNumber עובר **הגדלה** (incremented), כלומר, הוא גדל ב-1. הסמל להגדלה עצמית הוא ++ (כמו בשם C++, שיפור על C). וכך, הערך במשתנה smallNumber הוא 65,536. עם זאת, משתנים מטיפוס unsigned short integer אינם יכולים להחזיק מספר גדול מ-65,535, ולכן הערך גולש ל-0, שמודפס בשורה 8.

מונחון

הגדלה (incremented) – כאשר ערך עובר הגדלה, הוא גדל ב-1.

גלישת signed Integer (שלם מסומן)

signed integer שונה מ-unsigned integer בכך שחצי מהערכים שתוכל לייצג הם שליליים. במקום לדמיין מד-מרחק רגיל של מכונית, תוכל לדמיין מד-מרחק שמסתובב למעלה עבור מספרים חיוביים ולמטה עבור מספרים שליליים. קילומטר אחד מ-0 הוא 1 או 1-. כאשר ייגמרו לך המספרים החיוביים, תגיע ישירות למספרים השליליים הגדולים ותספור אחורה חזרה ל-0. תדפיס 2.5 מראה מה קורה כשתוסיף 1 למספר החיובי המקסימלי במשתנה מטיפוס short integer.

תדפיס 2.5 גלישת signed integer

```
1: #include <iostream>
2: using namespace std;
3:  int main()
4:  {
5:    short int smallNumber;
6:    smallNumber = 32767;
7:    cout << "small number:" << smallNumber << endl;
8:    smallNumber++;
9:    cout << "small number:" << smallNumber << endl;
10:  smallNumber++;
11:  cout << "small number:" << smallNumber << endl;
12:  return 0;
13: }
small number:32767
small number:-32768
small number:-32767
```

בשורה 5 smallNumber מוצהר כמשתנה מטיפוס signed short integer. (אם לא תאמר בפרוש שהוא אינו מסומן, ההנחה היא שהוא מסומן.) התוכנית ממשיכה בדומה לתוכנית הקודמת, אך הפלט שונה למדי.

השורה התחתונה היא שבדומה למשתנה מטיפוס unsigned integer, signed integer גולש מהערך החיובי הגבוה ביותר לערך השלילי הגבוה ביותר.

קבועים

מטרתו של **משתנה** (variable) היא לאחסן ערך. אני קורא לו משתנה בגלל שהוא עשוי להשתנות, כלומר, הערך עשוי להשתנות במהלך התוכנית. howManyLetters מתחיל כ-0, אך מוקצה לו ערך חדש המבוסס על הקלט של המשתמש.

עם זאת, לפעמים תצטרך ערך קבוע - ערך שלא ישתנה במהלך התוכנית. המספר המינימלי של אותיות שמותר למשתמש לבחור הוא דוגמה לערך קבוע; המתכנת קובע אותו הרבה לפני שהתוכנית רצה.

ערך קבוע מראש נקרא **קבוע** (constant). יש שני סוגים של קבועים: **קבוע מילולי** (literal) ו**קבוע סמלי** (symbolic).

קבועים מילוליים

קבוע מילולי (literal constant) הוא ערך המוקלד ישירות לתוכנית שלך בכל פעם שיש בו צורך. לדוגמה

```
int howManyLetters = 7;
```

howManyLetters הוא משתנה מטיפוס int; 7 הוא קבוע מילולי. לא תוכל להקצות ערך ל-7, ולא ניתן לשנות את ערכו.

קבועים סמליים

בדומה למשתנים, **קבועים סמליים** (symbolic constants) הם מיקומי אחסון, אך התוכן שלהם לעולם לא משתנה במהלך התוכנית שלך. כאשר אתה מצהיר על קבוע, מה שאתה באמת עושה זה אומר למהדר "התייחס אליו כמו למשתנה, אבל אם אי פעם אשנה את הערך המאוחסן כאן, הודע לי על כך." המהדר מודיע לך על ידי שגיאת מהדר.

תגדיר את minLetter כקבוע סמלי - ובאופן ספציפי, integer קבוע שערכו 2. שים לב שמספר שורות קוד משתמשות בקבוע minLetter. אם תחליט מאוחר יותר לשנות את הערך ל-3, תצטרך לשנות אותו רק במקום אחד - השינוי משפיע על שורות קוד רבות. זה עוזר לך להימנע מבאגים בקוד שלך. השינויים הם מקומיים, ולכן אין סיכוי ששורה אחת תניח שמספר האותיות המינימלי הוא 2, ושורה אחרת תניח שהוא 3.

בפועל, ישנן שתי דרכים להצהיר על קבוע סמלי בשפת C++. הדרך הישנה והמסורתית שהיא כיום מיושנת, משתמשת בהוראת קדם-מעבד (preprocessor directive): #define.

הגדרת קבועים עם #define

כדי להגדיר קבוע בדרך המסורתית, הקלד את הקוד הבא:

```
#define minLetters 2
```

שים לב שכאשר הוא מוצהר בצורה זו, minLetters אינו מטיפוס כלשהו (int, char וכו'). #define מבצע החלפת טקסט פשוטה. בכל פעם שהקדם-מעבד רואה את המילה minLetters, הוא מכניס את הטקסט 2.

כיוון שהקדם-מעבד מופעל לפני המהדר, המהדר שלך לעולם לא רואה את הקבוע שלך; הוא רואה את המספר 2. מאוחר יותר, כשאדבר על ניפוי שגיאות, תגלה שקבועים של #define אינם מופיעים כקבועים סימבוליים במנפה השגיאות; תראה רק את הערך המילולי (2).

הגדרת קבועים עם const

למרות ש-#define עובד, ישנה דרך חדשה וטובה בהרבה להגדיר קבועים בשפת C++:

```
const int minLetters = 2;
```

זה יוצר את הקבוע הסמלי minLetters אך מוודא שיש לו טיפוס מסוים - integer. אם תנסה לכתוב

```
const int minLetters = 3.2;
```

המהדר יתלונן שהצהרת עליו כטיפוס int אך אתה מנסה לאתחל אותו בטיפוס float.

סיור

מטרת **מנפה השגיאות** (debugger) היא לאפשר לך להציץ לתוך המכונה ולהביט בשינויים שעוברים המשתנים שלך במהלך ריצת התוכנית. תוכל להשיג הרבה עם מנפה שגיאות, ובצד עורך הטקסט שלך, זהו הכלי החשוב ביותר שלך.

לרוע המזל, רוב הטירונים אינם חשים בנוח עם מנפה השגיאות שלהם עד לשלב מאוחר בניסיונם בשפת C++. זה חבל כי מנפה השגיאות הוא כלי לימודי נהדר.

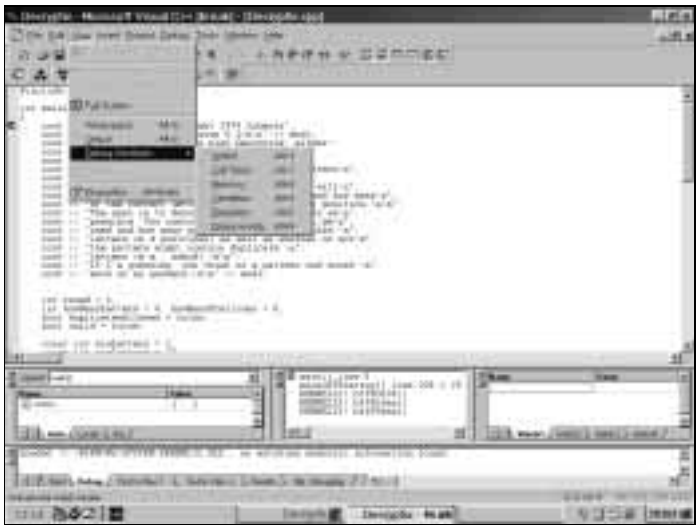
למרות שכל מנפה שגיאות הוא שונה ואתה לבטח תרצה לעיין בתיעוד שלך, סיור זה מדגים כיצד תוכל לנפות שגיאות בתדפיס 2.1 תוך שימוש במנפה השגיאות של Microsoft Visual C++ Enterprise Edition. הניסיון המדויק שלך עשוי להיות שונה, אך העקרונות זהים.

בצע את הצעדים הבאים:

1. צור פרויקט בשם Decryptix.
2. צור קובץ חדש בשם decryptix.cpp.
3. הכנס את התוכנית כפי שהיא כתובה או הורד אותה מאתר האינטרנט שלי.
4. הצב את הסמן בשורה הראשונה אחרי הסוגר הפותח והקש F9. תראה נקודה אדומה בשוליים שליד אותה שורה, המציינת נקודת עצירה (ראה איור 2.4).
5. הקש Go (F5). מנפה השגיאות מתחיל לפעול, והקוד שלך רץ.
6. מתוך התפריט View (תצוגה) בחר בפקודה Debug Windows (חלונות ניפוי שגיאות) וודא שהחלונות Watch (התבונן) ו-Variables (משתנים) פתוחים (ראה איור 2.5)

7. הקש על Step Over (F10) (צעד מעבר) כדי לעבור על הקוד שורה אחר שורה.
8. גלול מטה לשורה שבה מוגדר `duplicatesAllowed`, והצב שם את נקודת העצירה. הקש על Go (F5) כדי להריץ עד נקודת עצירה שנייה זו.
9. שים לב שבחלון המשתנים `howManyLetters` ו- `howManyPositions` הם `0`, אבל למשתנה `duplicatesAllowed` יש ערך אקראי (ראה איור 2.6).
10. הקש על StepOver (F10) כדי לצעוד מעבר לשורת קוד זו. זה גורם להיווצרותו ואתחולו של `duplicatesAllowed`. שים לב לכך שהערך המוצג בחלון המשתנים הוא כעת נכון (`false` מיוצג כ-`0` ו-`true` מוצג כ-1). שים לב גם לכך שלמשתנה `round` יש ערך אקראי. הקש שוב על F10, `round` מאותחל ב-1.
11. סייר במנפה השגיאות וקרא את התיעוד ואת קובצי העזרה. ככל שתבלה זמן רב יותר במנפה השגיאות, כך תעריך את ערכו העצום, הן לחיפוש באגים והן בעזרה שהוא נותן לך כדי להבין כיצד תוכניות עובדות.

איור 2.4
נקודת עצירה במערכת Visual C++.



איור 2.5
בדיקה שהחלונות Watch ו-Variable פתוחים.

איור 2.6
התבוננות בערכים במנפה השגיאות.