examples c++

Global vs local variables

1.
arguments. These variables are called the formal parameters of the function
It is Mean void f(char c, char b)

even though it looks correct.
Storage for global variables is in a fixed region of memory set aside for this purpose by the compiler. Global variables are helpful when many functions in your program
use the same data. You should avoid using unnecessary global variables, however. They take up memory the entire time your program is executing, not just when they are needed. In addition, using a global where a local variable would do makes a function less general because it relies on something that must be defined outside itself. Finally, using a large number of global variables can lead to program errors because of unknown

למרות שזה נראה נכון.
אחסון למשתנים גלובליים נמצא באזור קבוע של זיכרון שהוקצה למטרה זו על ידי המהדר. משתנים גלובליים מועילים כאשר פונקציות רבות בתוכנית שלך
להשתמש באותם נתונים. עם זאת, עליך להימנע משימוש במשתנים גלובליים מיותרים. הם תופסים זיכרון כל הזמן שהתוכנית שלך מופעלת, לא רק כאשר הם נחוצים. בנוסף, שימוש בגלובל שבו משתנה מקומי יתאים הופך פונקציה לפחות כללית מכיוון שהיא מסתמכת על משהו שחייב להיות מוגדר מחוץ לעצמה. לבסוף, שימוש במספר רב של משתנים גלובליים יכול להוביל לשגיאות תוכנית בגלל לא ידוע

2.
to understand the difference between a declaration and a definition. A declaration declares the name and

type of an object. A definition causes storage to be allocated for the object. While there can be many
declarations of the same object, there can be only one definition for the object.

3.
Constants:
Constants refer to fixed values that the program cannot alter
Both C and C++ define wide characters (used mostly in non-English language environments), which are 16 bits long. To specify a wide character constant, precede the character with an **L**. For example,

```
wchar_t wc;
wc = L'A';
```

Here, **wc** is assigned the wide-character constant equivalent of A. The type of wide characters is **wchar_t**. In C, this type is defined in a header file and is not a built-in type. In C++, **wchar_t** is built in.
סוג זה מוגדר בקובץ כותרת ואינו סוג מובנה. ב c/c++,
so wchar_t is built in,

4.
string vs single character : char ch='a' vs char ch[]="a"
"this is a test" is a string. You have seen examples of strings in some of the printf( ) statements in the sample programs. Although C allows you to define string constants, it does not formally have a string data type. (C++ does define a string class, however.)
You must not confuse strings with characters. A single character constant is enclosed in single quotes, as in 'a'. However, "a" is a string containing only one letter.
'a'- קבוע תו בודד מוקף במרכאות בודדות
"a"-הוא מחרוזת המכילה רק אות אחת

זהו מבחן" הוא מחרוזת. ראית דוגמאות למחרוזות בחלק מהמשפטים מאפשר לך להגדיר קבועי מחרוזת, C-בתוכניות לדוגמה. למרות ש printf( ) מגדיר מחלקת C++, עם זאת). אין לו באופן רשמי סוג נתוני מחרוזת מחרוזת.)

5.
Type Conversion in Assignments
When variables of one type are mixed with variables of another type, a *type conversion* will occur. In an assignment statement, the type

conversion rule is easy: The value of the right side (expression side) of the assignment is converted to the type of the left side (target variable), as illustrated here:

```
int x;
char ch;
float  f;
void func(void)
{
ch = x;
x = f;
f = ch;
f = x;
}
```

6.

wchar_t vs string

wchar_t ca='a' // it print the value string but we have to %c

wchar_t ca[]= L"s";//it print the value string but we have to %s because the-> ""

wprintf(L"%s \n",ca);

wchar_t ca= L'a';

wprintf(L"%c \n",ca); ans printf("%c just c \n",ca);

char c=''//until 8 bit

char c[]="אבגabc" // string char can get any char print c[index] or c print all string

wchar_t ch='א';// . bit 16 any char

wchar_t ch[]="אבג";

wchar_t ci[]= L"דגכ" /L'א'; // end with variable if !=0 and print hex of code point utf-8 or printf("%4x",ci) print hex unicode

//wcout<<L"test- "<< ci<<" \n";

string test="גל";// print value

string *ptest=&test;

//ptest++;

cout<<ci<<"\n";

ofstream f("./readmine.txt",ios::out);

f << ci;//write string

//f.write((char *)ptest,2);// write one char also 16 bits

f.close();

Type character	Argument	Output format
%c	Character	When used with printf functions, specifies a single-byte character; when used with wprintf functions, specifies a wide character.
%C	Character	When used with printf functions, specifies a wide character; when used with wprintf functions, specifies a single-byte character.
%s	String	When used with printf functions, specifies a single-byte or multi-byte character string; when used with wprintf functions, specifies a wide-character string. Characters are displayed up to the first null character or until the *precision* value is reached.

%S	String	When used with printf functions, specifies a wide-character string; when used with wprintf functions, specifies a single-byte or multi-byte character string. Characters are displayed up to the first null character or until the *precision* value is reached.

7.
x = (y=3, y+1);
first assigns y the value 3 and then assigns x the value 4. The parentheses are necessary because the comma operator has a lower precedence than the assignment operator. Essentially, the comma causes a sequence of operations. When you use it on the
right side of an assignment statement, the value assigned is the value of the last expression of the comma-separated list.
The comma operator has somewhat the same meaning as the word "and" in normal English as used in the phrase "do this and this and this."

תחילה מקצה ל y את הערך 3 ולאחר מכן מקצה לx את הערך 4 הסוגרים נחוצים כי לאופרטור הפסיק יש קידומת נמוכה יותר מהאופרטור ההקצאה

The [ ] and ( ) Operators המפעילים:
Parentheses are operators that increase the precedence of the operations inside them. Square brackets perform array indexing

סוגריים הם אופרטורים המגדילים את הקידומת בתוכם סוגרים מרובעים מבצעים אינדקסים מערכים

char arr[70]- the expression within square
Brackets provides an index into that array

הביטוי בתוך הסוגרים המרובעים מספר אינדקס למערך זה,

8.
The Dot (.) and Arrow (->) Operators:
In C, the . (dot) and the >(arrow) operators access individual elements of structures and unions. Structures and unions are compound (also called aggregate) data types that may be referenced under a single name (see Chapter 7). In C++, the dot and arrow operators are also used to access the members of a class.
The dot operator is used when working with a structure or union directly. The arrow operator is used when a pointer to a structure or union is used. For example, given the fragment
מבנים ואיגודים הם סוגי נתונים מורכבים נקראים גם מצטברים,
אופרטור הנקודה משמש כאשר עובדים ישירות עם מבנה או האיגוד
אופרטור החץ משמש כאשר נעשה שימוש במצביע
למבנה או איחוד

example:

```
struct employee
    {
      char name[80];
      int age;
      float wage;
    } emp;
    struct employee *p = &emp; /* address of emp into p */
```

you would write the following code to assign
כדי להקצות את הערך
the value 123.23 to the **wage** member of structure variable **emp**:

```
    emp.wage = 123.23;
```

However, the same assignment using a
עם זאת אותה הקצאה באמצעות מצביע
pointer to **emp** would be p->wage = 123.23;

9.
Type Conversion in Expressions:
The compiler converts all operands up to the type of the largest operand, which is called type promotion. First, all char and short int values are automatically elevated to int. (This process is

called integral promotion.) Once this step has been completed, all other conversions are done operation by operation, as described in the following type conversion algorithm:

כאשר קבועים ומשתנים מסוגים שונים מעורבים בביטויים כולם מומרים לאותו סוג המהדר ממיר את כל האופרנדים מהקטן עד לסוג האופרנד הגדול ביותר

תהליך זה נקרא קידום אינטגרלי,

cast to int or some num to get the remander:

```
int i;
        for(i=1; i<=100; ++i)
          printf("%d / 2 is: %f\n", i, (float) i /2);
```

10.

Redundant or additional parentheses do not cause errors or slow down the execution of an expression. You should use parentheses to clarify the exact order of evaluation,
both for yourself and for others. For example, which of the following two expressions is easier to read?

סוגריים מיותרים או נוספים אינם גורמים לשגיאות או מאטים את הביצוע של ביטוי. עליך להשתמש בסוגריים כדי להבהיר את סדר ההערכה המדויק, גם עבור עצמך וגם עבור אחרים. לדוגמה, איזה משני הביטויים הבאים קל יותר לקריאה?

```
// get same result with () or without:
 x = y/3-1*temp+127;
 cout<<x<<"\n";
 x = (y/3) - (1*temp) + 127;
 cout<<x;
```

11.

The if-else-if Ladder:

```
if (expression) // ביטוי
     statement; // הצהרות כמו משתנים פרמטיבים
else if (expression)
      statement;
else if (expression)
      statement; .
else
     statement;
```

```
x = 10;
  y = x>9 ? 100 : 200;
```

In this example, **y** is assigned the value 100. If **x** had been less than 9, **y** would have received the value 200. The same code written with the **if-else** statement would be

בדוגמא זו y מקבל את הערך 100 אם x קטן מ9
אחרת אם x גדול מ9 y מקבל את הערך 200

same code that wrote like that:

```
 x = 10;
  if(x>9) y = 100;
  else y = 200;
```

12.

The Conditional Expression הביטוי המותנה

13.

exit()

void exit(int *return_code*);

The value of *return_code* is returned to the calling process, which is usually the operating system. Zero is generally used as a return code to indicate normal program termination. Other arguments are used to indicate some sort of error.

C++-style header **<cstdlib>**.

```
      int main(void)
      {
        if(!virtual_graphics()) exit(1);
        play();
        /* ... */
}
```

14.

Passing Single-Dimension Arrays to Functions

העברת מערכים חד ממדים לפונקציה

In C/C++, you cannot pass an entire array as an argument to a

function. You can, however, pass to the function a pointer to an array by specifying the array's name without an index. For example, the following program fragment passes the address of i to func1( ):

לא ניתן להעביר מערך שלם כארגומנט לפונקציה עם זאת ניתן להעביר לפונקציה מצביע למערך ע״י ציון שם המערך ללא אינדקס לדוגמא התוכנית מעבירה את הכתובת של i לפונקציה func1

```
 int main(void)
   {
int i[10];
     func1(i);
.
.
}
```

If a function receives a single-dimension array, you may declare its formal parameter in one of three ways: as a pointer, as a sized array, or as an unsized array. For example, to receive **i**, a function called **func1( )** can be declared as

אם הפונקציה מקבלת יחיד גודל מערך, אתה יכול להכריז פורמלי פרמטר ב 1 מ 3 הדרכים,
כמצביע, כגודל מערך, כמערך ללא גודל,
לדוגמא כדי לקבל את i בפונקציה שנקראת func1 ניתן להכריז:

E
1

```
void func1(int *x) /* pointer */
   {...}
```

2

```
void func1(int x[10]) /* sized array */
{...}
```

3

```
 void func1(int x[]) /* unsized array */
   {...}
```

All three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. The first declaration actually uses a pointer. The second employs the standard array declaration. In the final version, a modified version of an array declaration simply specifies that an array of type int of some length is to be received. As you can see, the length of the array doesn't matter as far as the function is concerned because C/C++ performs no bounds checking. In fact, as far as the compiler is

concerned,

כל 3 שיטות הכרזה מייצרות תוצאות דומות שכל אחת אומרת למהדר שהתקבל מצביע של מספר שלם

ההכרזה הראשונה משתמשת למעשה במצביע

ההכרזה השניה בהצהרת מערך הסטנדרטי

הכרזה השלישית גרסה שונה של הצהרת מערך פשוט היא אומרת למהדר שיש לקבל מערך סטנדרטי מסוג int באורך מסויים כפי שאתה רואה אורך המערך המערך הוא לא משנה מבחינת הפונקציה מכיוון ש c/c++ לא מבצעים בדיקת גבולות, מבחינת המהדר,

5

```
 void func1(int x[32])
   {...}
```

also works because the compiler generates code that instructs func1( ) to receive a pointer—it does not actually create a 32-element array.

זה גם עובד כי המהדר מייצר קוד המורה ל func1() לקבל מצביע, והוא לא מייצר 32 אלמנטים במערך

Null-Terminated Strings

By far the most common use of the one-dimensional array is as a character string.

C++ supports two types of strings. The first is the null-terminated string, which is a null-terminated character array. (A null is zero.) Thus a null-terminated string contains the characters that comprise the string followed by a null. This is the only type of string defined by C, and it is still the most widely used. Sometimes null-terminated strings are called C-strings. C++ also defines a string class, called string, which provides an object-oriented approach to string handling. It is described later in this book. Here, null-terminated strings are examined.

When declaring a character array that will hold a null-terminated string, you need to declare it to be one character longer than the largest string that it is to hold.

ללא ספק השימוש השימוש הנפוץ ביותר במערך החד מימדי הוא כמחרוזת תווים, c++ תומך בשתי סוגי מחרוזות, 1 מחרוזת עם סיומת אפס שהיא מערך תווים עם סיומת אפס בסוף התווים null זה אפס

וזהו סוג המחרוזת היחיד שמוגדר על ידי c והוא עדיין הנפוץ ביותר לפעמיים מחרוזת עם סיומת אפס נקראות גם
S-String
2 מגדיר גם מחלקת מחרוזת הנקראת string המספקת גישה מונחה עצמים לטיפול במחרוזת,
כאשר מחריזים על מערך תווים שיכיל מחרוזת אתה צריך להחריש על תו אחד יותר כדי שיכיל גם את ה null אפס בסוף מערך תווים שהוא צריך להחזיק
לדוגמא:

```
char str[11];
“hello there” // 0-9=10 plus 1 null= 11 indexs
```

אינך צריך להוסיף null אפס לסוף קבועי המחרוזת באופן ידני המהדר עושה זאת עבורך באופן אוטומטי

C/C++ supports a wide range of functions that manipulate null-terminated strings. The most common are
תומך טווח רחב מגוון של פונקציות המבצעות מניפולציות null אפס עם סיומת במחרוזת, הנפוץ ביותר:

Name
strcpy(*s1*, *s2*), strcat(*s1*, *s2*), strlen(*s1*), strcmp(*s1*, *s2*), strchr(*s1*, *ch*), strstr(*s1*, *s2*),
Function explain
Copies *s2* into *s1*.
מעתיק 2 ל 1
Concatenates *s2* onto the end of *s1*.
משרשר 2 לקצה של 1
Returns the length of *s1*.
מחזיר אורך של 1
Returns 0 if *s1* and *s2* are the same; less than 0 if *s1*<*s2*; greater than 0 if *s1*>*s2*.
מחזיר אפס שתי המחרוזות דומות באורך, קטן מאפס אם מחרוזת אחת קטנה מ 2, גדול מאפס אם מחרוזת 1 גדולה ממחרוזת 2
Returns a pointer to the first occurrence of *ch* in *s1*.
מחזיר מצביע למופע הראשון של תווים ch בs1
Returns a pointer to the first occurrence of *s2* in *s1*.
דומה רק ב s2
These functions use the standard header file string.h. (C++ programs can also use the C++-style header <cstring>)

Two-Dimensional Arrays

C/C++ supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, essentially, an array of one-dimensional arrays. To declare a two-dimensional integer array d of size 10,20, you would write

תומך במערכים רב ממדים הצורה הפשוטה ביותר של מערך הרב מימדי היא מערך הדו מימדי, מערך דו מימדי הוא בעצם מערך של מערך חד מימדיים כדי להכריז על מערך שלם דו מימדי d בגודל 10,20

```
int d[10][20];
```

When a two-dimensional array is used as an argument to a function, only a pointer to the first element is actually passed. However, the parameter receiving a two-dimensional array must define at least the size of the rightmost dimension. (You can specify the left dimension if you like, but it is not necessary.) The rightmost dimension is needed because the compiler must know the length of each row if it is to index the array correctly. For example, a function that receives a two-dimensional integer array with dimensions 10,10 is declared like this:

כאשר מערך דו מימדי משמש כארגומנט לפונקציה רק המצביע לאלמנט הראשון מעובר או נשלח בפועל, עם זאת הפרמטר המקבל מערך דו מימדי חייב להגדיר לפחות את גודל של המימד הימני ביותר כלומר השורות מכוון שהמהדר חייב לדעת את כל אורך של כל שורה כדי לאינדקס את המערך בצורה הנכונה,
לדוגמא מקבלת מערך 10,10 שמקבל מספרים שלמים:

```
int x[3][4];
func1(x);
 void func1(int x[][10])
   {...}
```

The compiler needs to know the size of the right dimension in order to correctly execute expressions such as

inside the function. If the length of the rows is not known, the compiler cannot determine where the third row begins.

בתוך הפונקציה אם אורך השורה אינו ידוע (הפונקציה למעלה אורך הדו

מימד מוגדר בצד ימין של הפרמטר המקבל אך לא בתוך הפונקציה) המהדר לא יכול לקבוע היכן מתחילה השורה השלישית

int x[2][4];//0-1,0-3

Arrays of Strings

It is not uncommon in programming to use an array of strings. For example, the input processor to a database may verify user commands against an array of valid commands. To create an array of null-terminated strings, use a two-dimensional character array. The size of the left index determines the number of strings and the size of the right index specifies the maximum length of each string. The following code declares an array of 30 strings, each with a maximum length of 79 characters, plus the null terminator.

char str_array[5][10];//0-4,0-9

It is easy to access an individual string: You simply specify only the left index. For example, the following statement calls gets( ) with the third string in str_array.

gets(str_array[2]);"yoni" ->n char of index

The preceding statement is functionally equivalent to

gets(&str_array[2][0]);//"yoni" ->n

but the first of the two forms is much more common in professionally written C/C++ code.

To better understand how string arrays work, study the following short program, which uses a string array as the basis for a very simple text editor:

```
/* A very simple text editor. */
#include <stdio.h>
#define MAX 100
#define LEN 80
char text[MAX][LEN];
int main(void)
{
  register int t, i, j;
  printf("Enter an empty line to quit.\n");
```

```
    for(t=0; t<MAX; t++) {
      printf("%d: ", t);
      gets(text[t]);
      if(!*text[t]) break; /* quit on blank line */
}
    for(i=0; i<t; i++) {
      for(j=0; text[i][j]; j++) putchar(text[i][j]);
      putchar('\n');
}
return 0; }
```

This program inputs lines of text until a blank line is entered. Then it redisplays each line one character at a time.

התוכנית מכניסה שורות טקסט עד להזמנת שורה ריקה לאחר מכאן הוא מציג מחדש כל שורה בתו אחד כל פעם,

Multidimensional Arrays

מערכים רב מימדיים

Arrays of more than three dimensions are not often used because of the amount of memory they require. For example, a four-dimensional character array with dimensions 10,6,9,4 requires

10 * 6 * 9 * 4

or 2,160 bytes. If the array held 2-byte integers, 4,320 bytes would be needed. If the array held doubles (assuming 8 bytes per double), 17,280 bytes would be required. The storage required increases exponentially with the number of dimensions. For example, if a fifth dimension of size 10 was added to the preceding array, then 172, 800 bytes would be required.

In multidimensional arrays, it takes the computer time to compute each index. This means that accessing an element in a multidimensional array can be slower than accessing an element in a single-dimension array.

When passing multidimensional arrays into functions, you must declare all but the leftmost dimension. For example, if you declare array m as

מערכים של יוצר מתלת מימד אינם משמשים לעתים קרובות הם גם דורשים יוצר זיכרון

במערכים רב מימדיים לוקח למחשב זמן לחשב כל ינדקס משמעות שגישה

לאלמנט במערך רב מימד יכולה להיות איטית מאשר חד מימדי, כאשר מעבירים מערכים רב מימדיים לפונקציות עליך להכריז על כולם מלבד הממד השמאלי ביותר לדוגמא:

```
int m[4][3][6][5];
```

a function, func1( ), that receives m, would look like this:

```
void func1(int d[][3][6][5])
{}
```

Of course, you can include the first dimension if you like.

Indexing Pointers

In C/C++, pointers and arrays are closely related. As you know, an array name without an index is a pointer to the first element in the array. For example, consider the following array.

```
char p[10];
```

The following statements are identical:

```
p &p[0]
```

Put another way,

```
p == &p[0]
```

evaluates to true because the address of the first element of an array is the same as the address of the array.

As stated, an array name without an index generates a pointer. Conversely, a pointer can be indexed as if it were declared to be an array. For example, consider this program fragment:

מוערך true כי הכתובת של האלמנט הראשון זהה לכתובת המערך, כאמור שם מערך ללא אינדקס יוצר מצביע, לעומת זאת מצביע ניתן לאינדקס כאילו הוא הוכרז כמערך לדוגמא שקול את הקטע הבא:

```
int *p, i[10];
p = i;
p[5] = 100;  /* assign using index */
*(p+5) = 100; /* assign using pointer arithmetic */
```

Both assignment statements place the value 100 in the sixth element of i. The first statement indexes p; the second uses pointer arithmetic. Either way, the result is the same.

This same concept also applies to arrays of two or more dimensions. For example, assuming that a is a 10-by-10 integer array, these two statements are equivalent:

שתי ההצהרות ההקצאה מציבות את הערך 100 באלמנט השישי של i

ההצהרה הראשונה מדגישה את p השניה משתמשת באריתמטיקה מצביע
כך או כך התוצאה זהה, אותו מושג חל גם על מערכים שתי מימדיים או יותר,
לדוגמא שתי ההצהרות הללו שקולות:

a
&a[0][0]

Pointer arithmetic is often faster than array indexing

לעתים משתמשים במצביעים כדי לגשת למערכים מכוון מכיוון שאריתמטיקה
של מצביעים היא לרוב מהירה יוצר מאינדקס מערך

to access elements within a row of a two-dimensional array. The following function illustrates this technique. It will print the contents of the specified row for the global integer array num:

כדי לגשת לאלמנטים בתוך שורה של. מערך דו מימדי
הפונקציה הבאה ממחישה טכניקה זו, זה ידפיס את התוכן של השורה
שצוינה, עבור מערך שלם הגלובלי,

```
void  pr_row(int j)
  {
int *p, t;
    p = (int *) &num[j][0]; /* get address of first
                               element in row j */
    for(t=0; t<10; ++t) printf("%d ", *(p+t));
  }
```

You can generalize this routine by making the calling arguments be the row, the row length, and a pointer to the first array element, as shown here:

אתה יכול להכליל שיגרה זו ע״י הפיכת ארגומנטים הקוראים להיות שורה,
אורך השורה ומצביע לרכיב המערך הראשון, כפי שמוצג כאן:

```
  void pr_row(int j, int row_dimension, int *p)
  {
int t;
    p = p + (j * row_dimension);
    for(t=0; t<row_dimension; ++t)
      printf("%d ", *(p+t));
}

  void f(void)
  {
```

```
  int num[10][10];
  pr_row(0, 10, (int *) num); /* print first row */
}
```

Arrays of greater than two dimensions may be reduced in a similar way. For example, a three-dimensional array can be reduced to a pointer to a two-dimensional array, which can be reduced to a pointer to a single-dimension array. Generally, an *n*-dimensional array can be reduced to a pointer and an (*n*-1)-dimensional array. This new array can be reduced again with the same method. The process ends when a single-dimension array is produced.
ניתן להקטין מערכים של יותר משני מימדים באופן דומה לדוגמה ניתן לצמצם מערך תלת מימדי למצביע למערך דו מימד,אותו ניתן לצמצם למצביע למערך חד מימדי, מערך n לצמצם n-1 ולהמשיך אם רוצים עד מערך חד מימדי,

Array Initialization

אתחול מערך

In the following example, a 10-element integer array is initialized with the numbers 1 through 10:

```
int i[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

For example, this code fragment initializes **str** to the phrase "I like C++".

```
char str[11] = "I like C++";
```

This is the same as writing

```
char str[11] = {'I', ' ', 'l', 'i', 'k', 'e',' ', 'C',
                '+', '+', '\0'};
```

Multidimensional arrays are initialized the same as single-dimension one
example, the following initializes **sqrs** with the numbers 1 through 10 and their squares.
מערכים רב מימדיים מאותחלים כמו מערכים חד מימדיים

```
int sqrs[10][2] = {
  1, 1,
 2,3,
```

```
  5, 7
};
```

When initializing a multidimensional array, you may add braces around the initializers for each dimension. This is called subaggregate grouping. For example, here is another way to write the preceding declaration.

אתחול מערך רב מימדי

```
int sqrs[10][2] = {
        {1, 1},
        {2, 4},
        {3, 9},
        {10, 100}
};
```

When using subaggregate grouping, if you don't supply enough initializers for a given group, the remaining members will be set to zero automatically.

בעט שימוש בקיבוץ משנה מצטבר אם אינך מספק מספיק מאתחלים עבור קבוצה נתונה, החברים הנוצרים יוגדרו לאפס באופן אוטומטי,

Unsized Array Initializations

אתחול מערך ללא גודל

Imagine that you are using array initialization to build a table of error messages, as shown here:

```
      char e1[12] = "Read error\n";
      char e2[13] = "Write error\n";
      char e3[18] = "Cannot open file\n";
```

As you might guess, it is tedious to count the characters in each message manually to determine the correct array dimension. Fortunately, you can let the compiler automatically calculate the dimensions of the arrays. If, in an array initialization statement, the size of the array is not specified, the C/C++ compiler automatically creates an array big enough to hold all the initializers present. This is called an unsized array. Using this approach, the message table becomes

```
  char e1[] = "Read error\n";
   char e2[] = "Write error\n";
   char e3[] = "Cannot open file\n";
```

Given these initializations, this statement

```
printf("%s has length %d\n",  e2,  sizeof e2);
```

will print

```
Write error has length 13
```

Besides being less tedious, unsized array initialization allows you to change any of the messages without fear of using incorrect array dimensions.

Unsized array initializations are not restricted to one-dimensional arrays. For multidimensional arrays, you must specify all but the leftmost dimension.

אם בהצהרת האתחול גודל המערך לא צוין המהדר
c/c++ יוצר אוטומטית מערך גדול מספיק כדי לאכיל את כל ה מאתחלים הקיימים, נקרא מערך ללא גודל
אתחול מערך ללא גודל מאפשר לך לשנות כל אחת מההודעות. אלה ללא חשש משימוש של מימדים שגויים של במערך, אתחולי מערך ללא גודל אינם מוגבלים למערכים חד מימדיים, עבור מערכים רב מימדיים עליך לציין את כולם מלבד המימד השמאלי ביותר, בדרך זו אפשר לבנות טבלאות בגדלים שונים והמבדר מקצה באופן אוטומטי מספיק אחסון עבורם,

```
int sqrs[][2] = {
    {1, 1},
    {2, 4},
    {3, 9},
    {10, 100}
};
```

The advantage of this declaration over the sized version is that you may lengthen or shorten the table without changing the array dimensions.

היתרון שאתה יכול להאריך או לקצר את הטבלה מבלי לשנות את מידות המערך,

Chapter 5.

Multiple Indirection

עקיפה מרובה

You can have a pointer point to another pointer that points to the target value. This situation is called *multiple indirection*,

pointers

A variable that is a pointer to a pointer must be declared as such. You

do this by placing an additional asterisk in front of the variable name. For example, the following declaration tells the compiler that **newbalance** is a pointer to a pointer of type **float**:

```
float **newbalance;
```

You should understand that **newbalance** is not a pointer to a floating-point number but rather a pointer to a **float** pointer.
To access the target value indirectly pointed to by a pointer to a pointer, you must apply the asterisk operator twice, as in this example:

משתנה שהוא מצביע למצביע חייב להיות מוכרז ככזה
אתה עושה זאת ע״י הצבת כוכבית נוספת לפני שם המשתנה, הוא אינו מצביע למספר של נקודה צפה אלא מצביע למצביע של צף, כדי לגשת לערך היעד שאליו מצביע למצביע. אופן עקיף עליך להחיל את האופרטור הכוכבית פעמיים כמו בדוגמא

```
#include <stdio.h>
int main(void)
{
  int x, *p, **q;
  x = 10;
  p = &x;
  q = &p;
  printf("%d", **q); /* print the value of x */
return 0; }
```

Here, **p** is declared as a pointer to an integer and **q** as a pointer to a pointer to an integer. The call to **printf( )** prints the number **10** on the screen.

Arrays of Pointers

Pointers may be arrayed like any other data type. The declaration for an **int** pointer array
of size 10 is

מצביעים עשויים להיות מערכים כמו כל סוג נתונים אחר הצהרה עבור מערך בגודל 10 נראת כך

```
int *x[10];
```

To assign the address of an integer variable called **var** to the third element of the pointer array, write

כדי להקצות את הכתובת של משתנה שהוא מספר שלם שנקרא var לאלמנט 3 של מערך המצביע

x[2] = &var;

To find the value of **var**,
כדי למצוא את הערך של var במערך המצביעים תכתוב כך
write: printf("%d ",*x[2]);

If you want to pass an array of pointers into a function, you can use the same method that you use to pass other arrays—simply call the function with the array name without any indexes. For example, a function that can receive array **x** looks like this:
אם תרצה לעביר מערך של מצביעים לפונקציה אפשר להשתמש באותה שיטה שבה אתה משתמש כדי להעביר מערכים אחרים, פשוט לקרוא לפונקציה עם שם המערך ללא כל אינדקסים לדוגמא:

```
void display_array(int *q[])
{
  int t;
  for(t=0; t<10; t++)
      printf("%d ", *q[t]);
 }
```

זכור q אינו מצביע למספרים שלמים אחא מצביע למערך של מצביעים למספרים שלמים לכן עליך להכריז על הפרמטר q כמערך של מצביעים שלמים
כפי שמוצג בשיטה, אתה לא יכול להכריז על q פשוט כמצביע שלם כי זה לא מה שזה,

Pointer arrays are often used to hold pointers to strings. You can create a function that outputs an error message given its code number, as shown here:
מעריכי מצביע לעתים קרובות משמשים להחזיק מצביעים למחרוזת, אתה יכול ליצור פונקציה שמוציאהאת השגיאה בהודעה כשנשלח לשיטה מספר קוד שגיאה כפי שמוצג כאן:

```
void syntax_error(int num)
{
  static char *err[] = {
    "Cannot Open File\n",
    "Read Error\n",
    "Write Error\n",
```

```
      "Media Failure\n"
};
    printf("%s", err[num]);
  }
```

15.

Two-dimensional arrays are stored in a row-column matrix, where the first index indicates the row and the second indicates the column. This means that the rightmost index changes faster than the leftmost when accessing the elements in the array in the order in which they are actually stored in memory. See Figure 4-2 for a graphic representation of a two-dimensional array in memory.
In the case of a two-dimensional array, the following formula yields the number of bytes of memory needed to hold it:
bytes = size of 1st index x size of 2nd index x sizeof(base type)
Therefore, assuming 4-byte integers, an integer array with dimensions 3,4 would have
3x 4 x 4 bytes
or 12*4 bytes allocated

The following example loads a two-dimensional array with the numbers 1 through 12 and prints them row by row.
Array row colm

```
 int main(void)
  {
    int t, i, num[3][4];//t is column and, i is the row
    for(t=0; t<3; ++t)
      for(i=0; i<4; ++i)
        num[t][i] = (t*4)+i+1;
    /* now print them out */
    for(t=0; t<3; ++t) {
      for(i=0; i<4; ++i)
        printf("%3d ", num[t][i]);
      printf("\n");
}
return 0; }
```

Chapter 6.

The General Form of a Function

functions are the building blocks of C and C++ and the place where all program activity occurs. This chapter examines their C-like features, including passing arguments, returning values, prototypes, and recursion. Part Two discusses the C++-specific features of functions, such as function overloading and reference parameters. The general form of a function is

פונקציות הם אבני דרך הבניין של c/c++ המקום בו מתרחשת כל הפעילות התוכנית, פרק זה בוחן את התכונות הדומות ל c שלהם כמו: including , העברת ארגומנטים, ערכים חוזרים, אבות טיפוס, רקורסיה, חלק שני דן בתכונות ספציפיות ל c++
של פונקציות, כגון עומס יתר של פונקציות ופרמטרי התייחסות,
הצורה הכללית של הפונקציה היא:

ret-type function-name(*parameter list*) {
body of the function
}

Scope Rules of Functions

היקף כללי פונקציות

Function Arguments
פונקציות ארגומנטים

Call by Value, Call by Reference
קריאה לפונקציה ע״י ערך או ע״י הפניה

Creating a Call by Reference
ליצור שיחה לפי הפניה

Calling Functions with Arrays
קריאה לפונקציה עם מערכים

Arrays are covered in detail in Chapter 4. However, this section discusses passing arrays as arguments to functions because it is an exception to the normal call-by-value parameter passing.
When an array is used as a function argument, its address is passed to a function. This is an exception to the call-by-value parameter passing convention. In this case, the code inside the function is operating on, and potentially altering, the actual contents of the array used to call the function. For example, consider the function print_upper( ), which prints its string argument in uppercase:

After the call to print_upper( ), the contents of array s in main( ) have also been changed to uppercase. If this is not what you want, you could write the program like this:
זה חריג לעברת פרמטר רגיל להתקשר לפי ערך
כאשר מערך משמש כארגומנט לפונקציה הכתובת שלו מעוברת לפונקציה
זהו חריג למוסכמה של העברת פרמטר במקרה זה הקוד של הפונקציה פועל
על ועלול לשנות את התוכן בפועל של המערך המשמש לקריאה לפונקציה
לאחר הקריאה לפונקציה התוכן של s שונה גם הוא לאותיות רשויות אם זה
לא מה שאתה רוצה אפשר לכתוב בדרך אחרת שכתבתי בהמשך or
main()

```
char s[80];
   gets(s);// read line text from keyboard
    print_upper(s)

 /* Print a string in uppercase. */
      void print_upper(char *string)
      {
        register int t;
        for(t=0; string[t]; ++t)  {
          string[t] = toupper(string[t]);
          putchar(string[t]);



}



   //or one line
```

בדרך הזאת s ב main לא משתנה

```
   putchar(toupper(string[t]));
```

In this version, the contents of array **s** remain unchanged because its values are not altered inside **print_upper( )**.
The standard library function **gets( )** is a classic example of passing arrays into functions. Although the **gets( )** in your standard library is more sophisticated,

בגירסה זו התוכן של s נשאר ללא שינוי מכוון שהערכים שלו בפונקציה print_upper נשאר ללא שינוי

the following simpler version, called **xgets( )**, will give you an idea of how it works.

הגרסה הבאה הפשוטה שנקראת xgets() תיתן לך מושג איך זה עובד,

```
   /* A simple version of the standard
      gets() library function. */
   char *xgets(char *s)
   {
     char ch, *p;
     int t;
     p = s;  /* gets() returns a pointer to s */
     for(t=0; t<80; ++t){
       ch = getchar();
switch(ch) {
case '\n':
```

```
  s[t] = '\0'; /* terminate the string */
  return p;
case '\b':
  if(t>0) t--;
  break;
default:
  s[t] = ch;
} }
s[79] = '\0';
return p; }
```

The **xgets( )** function must be called with a character pointer. This, of course, can
be the name of a character array, which by definition is a character pointer. Upon entry, **xgets( )** establishes a **for** loop from 0 to 79. This prevents larger strings from being entered at the keyboard. If more than 80 characters are entered, the function returns. (The real **gets( )** function does not have this restriction.) Because C/C++ has no built-in bounds checking, you should make sure that any array used to call **xgets( )** can accept at least 80 characters. As you type characters on the keyboard, they are placed in the string. If you type a backspace, the counter **t** is reduced by 1, effectively removing the previous character from the array. When you press ENTER, a null is placed at the end of the string, signaling its termination. Because the actual array used to call **xgets( )** is modified, upon return it contains the characters that you type.

יש לקרוא לפונקציה() xgets עם מצביע תו זה כמובן יכול להיות שם של מערך תווים שבהגדרה הוא מצביע תווים, אם כניסה ל()xgets קובע לולאה מאפס עד 79 תווים זה מונע הזנת מחרוזות גדולות יותר במקלדת, אם מזנים יותר מ 80 תווים הפונקציה חוזרת,
לפונקציה ()gets אין הגבלה כזאת, בגלל c ++c
אין בדיקת גבולות מובנית,
עליך לוודא שכל מערך משמש לקריאה לxgets שיכול לקבל לפחות 80 תווים, בזמן שאתה מקליד תווים במקלדת הם ממוקמים במחרוזת, אם תקליד רווח אחורי המונה t מצטמצם ב 1, ולמעשה מסיר את התו הקודם מהמערך,
כאשר אתה לוחץ inter ריק ממוקם בסוף המחרוזת, המסמן את סיומו, מכוון שהמערך בפועל המשמש לקריאה ל-()xgets שונה, עם החזרה הוא מכיל את התווים שאתה מקליד,

argc and argv—Arguments to main( )

Returning from a Function

There are two ways that a function terminates execution and returns to the caller. The first occurs when the last statement in the function has executed and, conceptually, the function's ending curly brace (}) is encountered. (Of course, the curly brace isn't actually present in the object code, but you can think of it in this way.) For example, the pr_reverse( ) function in this program simply prints the string "I like C++" backwards on the screen and then returns.

```
pr_reverse("I like C++");
void pr_reverse(char *s)
    {
      register int t;
      for(t=strlen(s)-1; t>=0; t--) putchar(s[t]);
    }
```

Once the string has been displayed, there is nothing left for pr_reverse( ) to do, so it returns to the place from which it was called. Actually, not many functions use this default method of terminating their execution. Most functions rely on the return statement to stop execution either because a value must be returned or to make a function's code simpler and more efficient.
A function may contain several return statements. For example, the find_substr( ) function in the following program returns the starting position of a substring within a string, or returns –1 if no match is found.

בתוכנית הבאה מחזירה את מיקום ההתחלה של מחרוזת משנה בתוך מחרוזת, או מחזיר 1- אם לא נמצאה התאמה.

לאחר שהמחרוזת הוצגה, לא נותר דבר לעשות אז היא חוזרת למקום שמימנו היא נקראה,ל-pr_reverse( )
למעשה, לא הרבה פונקציות משתמשות בשיטת ברירת המחדל הזו כדי לעצור את return להפסקת ביצוען. רוב הפונקציות מסתמכות על הצהרת

הביצוע או בגלל שיש להחזיר ערך או כדי להפוך את הקוד של פונקציה לפשוט ויעיל יותר.
לדוגמה, הפונקציה find_substr( ) בתוכנית הבאה מחזירה את מיקום ההתחלה של מחרוזת משנה בתוך מחרוזת, או מחזירה –1 אם לא נמצאה התאמה. פונקציה עשויה להכיל מספר הצהרות return.

heb

```
 int find_substr(char *s1, char *s2);
  int main(void)
      {

if(find_substr("C++ is fun", "is") != -1)
          printf("substring is found");
return 0; }
      /* Return index of first match of s2 in s1. */
      int find_substr(char *s1, char *s2)
      {
        register int t;
        char *p, *p2;
        for(t=0; s1[t]; t++) {
          p = &s1[t];
          p2 = s2;
          while(*p2 && *p2==*p) {
            p++;
p2++; }
          if(!*p2) return t; /* 1st return */
        }
         return -1; /* 2nd return */
      }
```

Returning Values

150 page

As long as a function is not declared as void, you may use it as an operand in an expression

כל עוד הפונקציה היא לא מוצהרת כ void

אתה יכול להשתמש בה כביטוי, ראה בנוסף את הדוגמא של find_substr() למטה,

x = power(y);// expression ביטוי

```
if(max(x,y) > 100) printf()  //expression in if ביטוי
for(ch=getchar(); isdigit(ch); ) ... ; //expression in for loop
```

In C++ (and C99), a non-void function must contain a return statement that returns a value. That is, in C++, if a function is specified as returning a value, any return statement within it must have a value associated with it.

However if execution reaches to end of non- void function, than garbage value is returned,
עם זאת אם הביצוע מגיע לסוף הפונקציה שאינה ביטלה אז זבל ערך מוחזר

Although this condition is not syntax error, it is still fundamental flaw and should be avoided
למרות שהמצב אינו תחביר שגיאה הוא עדיין יסודי פגום ויש/וצריך להימנע מיזה,
הפונקציה הבאה תקינה:

```
int find_substr(char *s1, char *s2);

int find_substr(char *s1, char *s2){
 int t;
 char *p, *p2;
 for(t=0; s1[t]; t++ ){

   p= &s1[t];//p pointer to first address and execution loop to second address
   p2= s2; //p2 pointer to first address

    cout<<"index *p IsSpase/Null= "<<s1[t] <<(s1[t]==' ')<< " " <<"substring IsSpase/Empty= "<< (!*p2)<<"    "<<"Chars "<<*p2<<" and "<<*p<<":" <<"are equal""?= "<< (*p2==*p) << "\n";

  while(*p2 && *p2==*p){//if p2 not empty and equal to first char of word than
    // check if word of p2 equal to word in string of p

    p++;//move next address
    p2++;
    cout<< "check eq words by next index of char->" << (*p2==*p) << "\n";
```

```
  }

   if(!*p2) return t;

 //pointer p2 that pointer to word we want search at string    but  if
p2==' '  we get to end word  so break loop and return index of value.
   // while break execution's reaches  if one char of word not equal,
   // other side this function search just one word so if we not return
// the pointer-> p2= s2; pointer again to first char of word and it could
be
// stack overflow with-> p= &s1[t] becouse this loop has just one
condition to return



 }// closing curly bracket of loop
 return -1;//if loop done there are no substring so -1.
}

int main(){

  **call the function find_substr()**
  As long as a function is not declared as void, you may use it as an
operand in an expression
            כל עוד הפונקציה לא מוצהרת כ void אתה יכול להשתמש בה כביטוי
 if(find_substr((char *)"the color's car is green and the bike is black",
(char *) "green")!= -1){
    cout<<"substring is found"<<"\n";
 }else{
    cout<<"substring is not found"<<"\n";
 }
}
```