תוכן עניינים

פלט קלט 3
printf 3
scanf 5
קליטת ערכים בעזרת מציינים 5
getchar / putchar 7
משתנים 7
משתנים גלובליים 7
משתנים לוקליים: 8
משתנים קבועיים: 8
typdef 8
Casting 8
אופרטורים 10
האופרטור sizeof 11
הוראות בקרה 13
if 13
אופרטור ההתניה :? 14
switch 14
while 15
for loop 15
do while 16
break continue 16
פונקציות מתמטיות 16
מערכים 17
גודל המערך 19
מערך סטטי 19
שימוש ב typdef 20
מחרוזות 21
strcat 22
טעויות נפוצות בעבודה עם מחרוזות: 22
strtok 23
מצביעים 24

אריתמטיקה של מצביעים: 25
מצביעים קבועים ומצביעים לערכים קבועים 27
מצביעים לפונקציות 28
הפעלת פונקציה שחזרה כתשובה לפונקציה אחרת 29
מיון גנרי מתוך שיעורי בית 30
החלפה מספר מסוים של בתים 33
מבנים 33
מערך של מבנים 35
הקצאות דינמיות 39
דוגמא הקצאה של מערך חד ודו מימדי 40
שינוי הקצאה 43
Memcpy 43
מבני נתונים 44
רשימה מקושרת כדי לבדוק אם משפט הוא פלינדרום 44
עץ בינארי 49
דוגמא תוכנית לספירת חזרות של מילים בעזרת עץ בינארי 51
פעולות על בתים 54
^ xor 54
| or 55
& AND 55
~ NOT 55
Shift 56
פונקציות עזר 56
get 56
printbit 56
set 56
החלפת משתנים בעזרת XOR 56
עבודה עם קבצים 57
פתיחת קובץ 57
פלט קבצים 57
קלט קבצים 58
דוגמא מסכמת משיעורי בית reverse: 59
קריאת קובץ שורה אחר שורה 61
קדם מהדר 62

מאקרו לשרשור מחרוזת ושימוש במילות מפתח..... 65
מאקרו לביצוע פקודה מספר מסויים של פעמים..... 65
סכנות..... 66
IEEE754..... 67
Ellipsis..... 69
Set/Reset bit positions using elpsis..... 69
Command Line Arguments..... 70
Complicated declaration explained with examples..... 72
טיפול בחריגות..... 75
Extren and Static..... 76
ASCII TABLE..... 77

פלט קלט

printf

specifier	**Output**	**Example**
`d` *or* `i`	Signed decimal integer	`392`
`u`	Unsigned decimal integer	`7235`
`o`	Unsigned octal	`610`
`x`	Unsigned hexadecimal integer	`7fa`
`X`	Unsigned hexadecimal integer (uppercase)	`7FA`
`f`	Decimal floating point, lowercase	`392.65`
`F`	Decimal floating point, uppercase	`392.65`
`e`	Scientific notation (mantissa/exponent), lowercase	`3.9265e+2`
`E`	Scientific notation (mantissa/exponent), uppercase	`3.9265E+2`
`g`	Use the shortest representation: `%e` or `%f`	`392.65`
`G`	Use the shortest representation: `%E` or `%F`	`392.65`
`a`	Hexadecimal floating point, lowercase	`-0xc.90fep-2`
`A`	Hexadecimal floating point, uppercase	`-0XC.90FEP-2`
`c`	Character	`a`
`s`	String of characters	`sample`
`p`	Pointer address	`b8000000`

קובעי רוחב

לפני המציין, אפשר אופציונאלית לכתוב גם קובע רוחב, כך:

```
%[width]<specifier>
```

כאשר specifier הוא המציין, ו widthהוא קובע הרוחב.

אם width הוא מספר, אז הדפסת הערך תיקח לכל הפחות width תווים. לדוגמה:

```
printf("%10d", 3);
```

תדפיס:

```
         3
```

כך מאפיינים גם רוחב הדפסה למחרוזות. לדוגמה:

```
printf("%10s", "Hello");
```

תדפיס:

```
     Hello
```

קובעי דיוק

לפני המציין, אפשר אופציונאלית לכתוב גם קובע דיוק, כך:

```
%[.precision]<specifier>
```

כאשר specifier הוא המציין, ו precisionהוא קובע הדיוק.

אם precision הוא מספר, אז דיוק המספר יהיה בדיוק precision תווים. לדוגמה:

```
printf("%.3f\n", 3.14159265);
printf("%.3f\n", 3.1);
```

ידפיסו:

```
3.142
3.100
```

אפשר להפעיל זאת גם על מחרוזות. לדוגמה:

```
printf("%.3s\n", "Hello");
```

תדפיס:

Hel

אפשר להשתמש בו זמנית בקובעי רוחב ודיוק, בצורה:

```
%[width][.precision]<specifier>
```

במקרה כזה, קודם יופעל קובע הדיוק, ולאחריו קובע הרוחב. לדוגמה:

```
printf("%10.3f\n", 3.14159265);
printf("%10.3f\n", 3.1);
printf("%10.3s\n", "Hello");
```

ידפיסו:

```
    3.142
    3.100
      Hel
```

דגלים

לפני המציין, אפשר אופציונאלית לכתוב גם דגלים, כך:

```
%[flags]<specifier>
```

כאשר flags הם דגלים, ו specifierהוא מציין.

הדגלים הם:

דגל	משמעות
-	במקרה שצויין קובע רוחב ,והערך המודפס צר יותר, הצמד לצד שמאל (ברירת המחדל הוא צד ימין).
+	הצמד לפני מספרים חיוביים את התו '+' (ברירת המחדל היא לכתוב מספרים חיוביים ללא סימן).
(רווח)	הצמד לפני מספרים חיוביים את התו.' '
#	הוסף עוד תווים לאחידות פלט, לדוגמא התו '.' לאחר מספרי נקודה צפה שערכם שלם.
0	במקרה שצויין קובע רוחב, והערך המודפס צר יותר, כתוב אפסים משמאל למספר.

אפשר להשתמש בו זמנית בדגלים, קובעים, ומציינים, בצורה הבאה:

```
%[flags][width][.precision]<specifier>
```

scanf

קליטת ערכים בעזרת מציינים
לאחר שמצהירים על משתנה ,אפשר לקלוט ערכים לתוכו. לדוגמה:

```
int x;

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

כאשר התוכנית תגיע לשורה זו, היא תמתין עד שהמשתמש/ת יקליד מספר וילחץ .Enter המספר ייקלט למשתנה.x

באותו האופן, נוכל לקלוט מספר משתנים מסוגים שונים:

```
int number;
char first_letter;
```

```
int phone;

printf("Please enter a number, first character of your name, and your
phone number:\n");
scanf("%d %c %d", &number, &first_letter, &phone);
```

בדוגמה זו יקבל המשתמש/ת בקשה לכתוב מספר, תו משמו ואת מספר הטלפון שלו. לאחר שיזין פרטים אלה, הם ייקלטו במשתנים.

לפני המציין, אפשר אופציונאלית לכתוב גם קובע רוחב, כך:

```
%[width]<specifier>
```

כאשר specifier הוא המציין, ו width-הוא קובע הרוחב.

אם width הוא מספר, אז ייקלטו לכל היותר width תווים. לדוגמה:

```
scanf("%3d", &num)
```

תקלוט לכל היותר 3 תווים. כך מאפיינים גם רוחב קליטה למחרוזות. לדוגמה:

```
  char a[5];

  scanf("%4s", a);
```

תקלוט לכל היותר 4 תווים למחרוזת.a

קליטת מחרוזות קבועות

נניח שאנו רוצים לקלוט שני מספרים, כל אחד בעל 5 ספרות. נוכל לכתוב זאת כך:

```
int x, y;

scanf("%5d %5d", &x, &y);
```

נשים לב שיקלטו שני מספרים שביניהם רווח אחד או יותר.

עתה נניח שאנו רוצים לקלוט שני מספרים, כל אחד בעל 5 ספרות, מופרדים על ידי פסיק. נוכל לכתוב זאת כך:

```
int x, y;

scanf("%5d,%5d", &x, &y);
```

כלומר, צריך לשים פסיק במחרוזת התבנית. אם המשתמש אכן יקליד שני מספרים מופרדים על ידי פסיקים, ייקלטו המספרים ב x-ו ,y-והפונקציה תחזיר 2. אם לא, הפוקנציה תחזיר מספר קטן מ-2.

נוכל להכליל זאת:

- אם מופיע תו שאינו רווח במחרוזת התבנית, הקלט ימשיך רק אם המשתמש יקליד תו זה בדיוק.

- הפונקציה scanf מחזירה את מספר המשתנים שהצליחה לקלוט.

getchar / putchar

הפונקציה getchar משמשת לקליטת תו בודד מהמשתמש. לדוגמא:

```
char ch;

ch = getchar;( )
```

הפוקנציה putchar משמשת להדפסת תו בודד על המסך, לדוגמא:

```
putchar('g');
```

ניתן להשתמש תמיד ב-printf/scanf ולא בפונקציות אלו, אולם אם נרצה להדפיס רק תו בודד, נעדיף

לעיתים להשתמש ב-putchar/getchar

משתנים

טיפוס שלם –

העיקריים:

טיפוס בסיסי	גודל (בבתים)	ערך מינימלי (signed)	ערך מקסימלי (signed)	ערך מקסימלי (unsigned)
char	1	-128	127	255
short int	2 לפחות	-32,768	32,767	65,535
int	2 לפחות	-32,768	32,767	65,535
long int	4 לפחות	-2,147,483,648	2,147,483,647	4,294,967,295

במהדר שאנו משתמשים int תופס 4 בתים.

טיפוסי נקודה צפה

טיפוסים אלה נועדו לאחסן מספרים רציונאליים .הטיפוסים נקראים מספרי נקודה צפה על שם השיטה רבת-הדיוק בה משתמש המחשב כדי לאחסן אותם.

- טיפוס רציונאלי בעל יכולת דיוק בינונית – *float*
- טיפוסים בעלי יכולת דיוק גבוהה וגבוהה במיוחד – *long double* ו-*double*.

משתנים גלובליים

להגדיר משתנים גם מחוץ לכל בלוק שהוא. משתנים כאלו יהיו *גלובליים* ,ויוכרו ע"י כל קטע קוד המכיר את השם שלהם. בקוד הבא, לדוגמה x ,הוא משתנה גלובלי:

```
char x;

int main()
```

```
{
  int n;

  n = 3;

  x = 'f';

  return 0;
}
```

משתנים לוקליים:

משתנה לוקליים נוצר ומת בתוך אותו בלוק שבוא הוא הוגדר.

משתנים קבועיים:

לפעמים מגדירים משתנים כמשתנים *קבועים*, או משתנים שאסור לשנות את ערכם. עושים זאת בצורה הבאה:

```
const <type> <name>
```

typdef

C מאפשרת להגדיר "שם נרדף" לטיפוסים. עושים זאת בצורה:

```
typedef <known_type> <alias>;
```

כאשר known_type הוא טיפוס משתנה ידוע, ו alias-הוא "שם נרדף" לו.

לדוגמה, אפשר לתת "שם נרדף" לשלם, ולהשתמש בו להצהרה על משתנים:

```
typedef int my_new_name_for_int;

my_new_name_for_int x = 3;
```

Casting

כדי להסב ערך כותבים בסוגריים, לפני הערך עצמו, את סוג הטיפוס אליו רוצים להסב. דוגמה: נניח שיש לנו מספר שלם ונרצה להכניס אותו למשתנה מסוג float, אז נכתוב את הקוד הבא:

```
float x = (float) 1;
```

ניתן לבצע זאת גם עם משתנים:

```
int i = 121;
char *x = (char *) i;
```

אלו הן דוגמאות חסרות תועלת במרבית המקרים, אך חוקיות לשימוש.

הסבה אוטומטית וסכנת אובדן המידע

נשים לב שבמרבית המקרים הסבות כמו שראינו קודם הן מיותרות: כמעט בכל מצב בו נזדקק להסבה של סוגי משתנים פשוטים, המהדר ידע לבצע זאת באופן עצמאי. למשל, חוקי (ונפוץ) להמיר int ל float-בצורה הבאה:

```
int a = 1;
float b = a;
```

שימו לב: לעיתים קרובות הפעולה הזו כרוכה באובדן מידע. אם, לדוגמה, נמיר את המספר 1.234 (שהוא עשרוני) ל-int, נקבל 1, ולא נוכל לשחזר את המספר המקורי. הרעיון פשוט: אם מסבים משתנה אל סוג אחר שיכול להכיל פחות מידע - המידע הנוסף יקוצץ. ניתן לראות זאת בדוגמה הבאה:

```
#include <stdio.h>

int main() {
        double a = 7.654321;
        int b;
        printf("Before: %lf\n", a);
        b = (int) a;
        a = (double) b;
        printf("After: %lf\n", a);
        return 0;
}
```

כל מה שהיה במשתנה a אחרי הנקודה אבד כתוצאה מההסבה, אפילו שהסבנו את אותו הערך בדיוק חזרה ל-double.

קריאה לפונקציה עם הסבה

מקרה נפוץ בו נזדקק להסבה הוא קריאה לפונקציות שדורשות סוג מסויים של משתנים. אם, למשל, נרצה לקרוא לפונקציה sqrt עם מספרים שלמים, נצטרך להסב אותם קודם. צורת הכתיבה היא כזאת:

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

int main() {
        int a = 16;
        double b;
        b = sqrt((double) a);
        printf("Square root of %d: %d\n", a, (int) b);
        return 0;
}
```

הסבה לא חוקית

הסבה היא חוקית בין משתנים פשוטים, אך בדרך כלל איננה חוקית עם סוגים אחרים. הדוגמה הבאה מראה שתי הסבות לא חוקיות:

```
struct my_struct {
        int x;
};
```

```c
int main() {
        int a = 16;
        struct my_struct b;
        b = (struct my_struct) a;     // Illegal
        a = (int) b;                  // Illegal
        return 0;
}
```

אופרטורים

אופרטורים ++/--

אופרטורים אלו פועלים על משתנה בודד. האופרטור ++ מגדיל את ערכו של המשתנה ב-1, ואילו האופרטור -- מקטין את ערכו של המשתנה ב-1.

אם האופרטור ייכתב משמאל למשתנה עלו הוא פועל, הערך של המשתנה ישתנה לפני ביצוע הפקודה הנוכחת. אם האופרטור ייכתב מימין, ערך המשתנה ישתנה לאחר ביצוע הפקודה הנוכחית.

לדוגמא:

```
int a=2, b=2, c, d;

c = a;++

d = ++b;
```

אופרטורים של השוואה

כאמור, לכל ביטוי בשפת C יש ערך. ניתן להתייחס אל כל ביטוי כאל ביטוי לוגי, כאשר מוגדר:

אם ערך הביטוי הוא 0, אזי הערך הלוגי המתאים לו הוא שקר (FALSE).

עבור כל ביטוי השונה מ-0, הערך הלוגי המתאים לו הוא אמת (TRUE).

למשל, ערכו של הביטוי 5 הוא אמת. אם נגדיר משתנה מסוג int בשם x, ונשים בו 0, אזי ערכו של הביטוי x הוא שקר.

שפת C מכילה מגוון של אופרטורים של השוואה. ניתן ליצור ביטויים לוגיים, הכוללים אופרטורים אלו.

למשל: ערכו של הביטוי (3>2) הוא אמת (שווה ל-1), בעוד שערכו של הביטוי (3=<2) הוא שקר (שווה ל-0).

לדוגמא, נביט בקטע הקוד הבא:

```
int x = 3 > 2;

printf("%d\n", x);
```

על המסך יודפס המספר 1.

נשים לב כי למרות שאמרנו כי כל מספר שונה מ-0 הוא אמת, כאשר בשפה אנו משתמשים בביטוי לוגי,

ערכו 1 אם הוא אמת ו-0 אם הוא שקר.

אופרטורים של השוואה בשפת C:

> קטן

< גדול

=> קטן שווה

=< גדול שווה

== שווה

=! לא שווה

אופרטורים לוגיים

אופרטור משמעות

! NOT (אופרטור אונרי)

&& AND לוגי (אופרטור בינארי)

|| OR לוגי (אופרטור בינארי)

אופרטור אונרי הוא אופרטור שפועל על ביטוי אחד. אופרטור בינארי הוא אופרטור שפועל בין שני ביטויים.

כאמור, בשפת C כל תוצאה שהיא שונה מ-0 מתפרשת כאמת ו-0 הוא שקר.

בטבלה הבאה, 1 מציין בטבלה זו כל אחד השונה מ-0.

x	!x
1	0
0	1

האופרטור sizeof

אופרטור זה מחזיר את מספר הבתים של הביטוי, טיפוס או משתנה הנמצא מימינו.

אופן השימוש באופרטור:

sizeof(expression)

sizeof(type)

לדוגמא:

```
#include <stdio.h>

int main()

{

 double j = 5.5;
```

```
 printf("In my computer: \n");
 printf("The size of char variable is %d\n,"
 sizeof(char);(
 printf("The size of int variable is %d\n", sizeof(int));
 printf("The size of j is %d\n", sizeof(j));
 return 0;
}
```

הפלט של תוכנית זו משתנה ממחשב למחשב, מכיוון שלא מוגדר ב-C גודל קבוע של בתים עבור כל סוג משתנה. פלט אפשרי של התוכנית יכול להיות:

```
In my computer:
The size of char variable is 1
The size of int variable is 2
The size of j is 8
```

הערך המוחזר על ידי האופרטור sizeof שונה בין מערכת הפעלה אחת לשניה ובין קומפליר לקומפילר.

עם זאת, שפת C מבטיחה לנו את קיום התנאים הבאים:

```
sizeof(short) < sizeof(int) < sizeof(long)
sizeof(float) <  sizeof(double) < sizeof(long double)
sizeof(signed) < sizeof(unsigned) < sizeof(int)
```

במקרה שלנו:

```
byte 1
char 1
short 2
int 4
long 8
float 4
double 8
```

קדימות	פעולה
1	. -> [] () ++(prefix) --(prefix)
2	sizeof - + ~ ! & * (cast)
3	* / %
4	+ -

<< >>	5
< <= > >=	6
== !=	7
&	8
|	9
&&	10
||	11
? :	12
= *= /= %= += -= <<= >>= &= |= ^=	13
,	14
--(postfix) ++(postfix)	15

הוראות בקרה

if

if (expression)

{

...

}

else

{

...

}

```
if(boolean_expression 1)
{
   /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
   /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
   /* Executes when the boolean expression 3 is true */
}
else
{
   /* executes when the none of the above condition is true */
}
```

אופרטור ההתניה :?

אופרטור זה הוא למעשה דרך מקוצרת ליצירת הסתעפות. בכל מקום בו אנו משתמשים באופרטור זה, ניתן

להחליפו במשפט if מקביל. עם זאת, ישנם מקרים בהם נוח להשתמש באופרטור זה במקום ב-if.

תחביר השימוש באופרטור:

```
(expression) ? expression1 : expression2;
```

משמעות האופרטור: הביטוי expression ייבדק. במידה והוא אמת, יבוצע expression1. במידה והביטוי הוא שקר, יבוצע expression2.

דוגמא:

x, y, z הם משתנים שהוגדרו קודם לכן בתוכנית.

```
z = x > y ? x : y;
```

z מקבל את הערך של הגדול מבין x ו-y.

switch

```
switch (expression)
{
 case VALUE1:
 ...
 break;
 case VALUE2:
 ...
 break;
...
 case VALUEn:
 ...
 break;
 default:
 break;
}
```

לדוגמא:

```
#include <stdio.h>
int main()
{
```

```
int i;
printf("Please enter number between 1 to 3: ");
scanf("%d", &i);
switch(i)
{
case 1:
printf("You entered 1.\n");
break;
case 2:
case 3:
printf("You entered 2 or 3.\n");
break;
default:
printf("ERROR: Illegal input.\n");
}
return 0;
}
```

כאשר נריץ את התוכנית, היא תחכה לקלט מהמשתמש.

במידה והמשתמש יקליד את המספר 1, יודפס "You entered 1". במידה והמשתמש יקליד את המספרים 2 או 3, יודפס "You entered 2 or 3". אם המשתמש יקליד מספר אחר תודפס הודעת שגיאה.

נשים לב שאחרי "case 2:" לא השתמשנו במילה השמורה break, ולכן הביצוע במקרה שהמשתמש מקיש 2, ממשיך אל "case 3:", ונעצר רק כאשר מגיעים אל הפקודה break הנמצאת שם

while

```
while ( expression )
{
   Single statement
   or
   Block of statements;
}
```

for loop

```
for( expression1; expression2; expression3)
{
   Single statement
   or
   Block of statements;
```

```
}
```

do while

```
do
{
   Single statement
   or
   Block of statements;
}while(expression);
```

break continue

break גרום ליציאה מהלולאה

continue מורה להתקדם לאיטרציה הבאה.

פונקציות מתמטיות

טיפוס מוחזר	פעולה מתמטית	פונקציה ב C
Double	שורש	Sqrt(x)
Double	חזקה(בסיס a, מאריך b)	Pow(a,b)
Double	e^x	exp(x)
Double	ln(x)	log(x)
Double	logx	log10(x)
Double	Sin(x)	sin(x)
Double	cos x	cos(x)
Double	tan x	tan(x)
Int	|x|	abs(x)
Float	|x|	fabs(x)

`sin(x)`	sine of x
`cos(x)`	cosine of x
`tan(x)`	tangent of x
`asin(x)`	$\sin^{-1}(x)$ in range $[-\pi/2, \pi/2]$, $x \in [-1, 1]$.
`acos(x)`	$\cos^{-1}(x)$ in range $[0, \pi]$, $x \in [-1, 1]$.
`atan(x)`	$\tan^{-1}(x)$ in range $[-\pi/2, \pi/2]$.
`atan2(y,x)`	$\tan^{-1}(y/x)$ in range $[-\pi, \pi]$.
`sinh(x)`	hyperbolic sine of x
`cosh(x)`	hyperbolic cosine of x
`tanh(x)`	hyperbolic tangent of x
`exp(x)`	exponential function e^x
`log(x)`	natural logarithm $\ln(x)$, $x>0$.
`log10(x)`	base 10 logarithm $\log_{10}(x)$, $x>0$.
`pow(x,y)`	x^y. A domain error occurs if `x`=0 and `y`$\leqslant$0, or if `x`$<$0 and `y` is not an integer.
`sqrt(x)`	$\sqrt{x}$, $x \geqslant 0$.
`ceil(x)`	smallest integer not less than `x`, as a `double`.
`floor(x)`	largest integer not greater than `x`, as a `double`.
`fabs(x)`	absolute value $\lvert x \rvert$
`ldexp(x,n)`	$x \cdot 2^n$
`frexp(x, int *exp)`	splits `x` into a normalized fraction in the interval [1/2, 1), which is returned, and a power of 2, which is stored in `*exp`. If `x` is zero, both parts of the result are zero.
`modf(x, double *ip)`	splits `x` into integral and fractional parts, each with the same sign as `x`. It stores the integral part in `*ip`, and returns the fractional part.
`fmod(x,y)`	floating-point remainder of `x/y`, with the same sign as `x`. If `y` is zero, the result is implementation-defined.

מערכים

הגדרת מערך צריכה להיות מהצורה:

```
<type> <name>[<size>];
```

בהגדרה מהצורה הזאת הערך size חייב להיות ידוע בזמן קומפילציה.

לדוגמה, נתבונן בקטע הקוד הבא:

```
int array1[30];
```

```
char array2[50];
double array3[1];
```

די לגשת לאיבר במערך, אפשר להשתמש ב*אינדקס* :יש לכתוב את שם המערך ואחריו בתוך סוגריים מרובעים את המספר הסידורי של האיבר במערך:

```
array1[2] = 5;
```

לעתים, כאשר מייצרים מערך, יש לתת ערך התחלתי לאיבריו. לדוגמה, נניח שמייצרים מערך של שלמים בגודל 3, ורוצים להכניס לו את האיברים 12, 22, ו-33.

כפי שראינו ב[גישה לאברי מערך](), אפשר לגשת לכל אחד מאיברי המערך. נוכל, לכן, להשתמש בהשמה לכל אחד מאיבריו:

```
int nums[3];

nums[0] = 12;
nums[1] = 22;
nums[2] = 33;
```

עם זאת, לצורך אתחול המערך בלבד, השפה מאפשרת צורה מקוצרת יותר:

```
int nums[3] = {12, 22, 33};
```

אפשר לאתחל את המערך גם בלי ציון גודל המערך בסוגריים המרובעות:

```
char nums[] ={"Goodbye"};     ==>    nums[7]

int nums[] ={3, 5, 65, 189, 54};   ==> nums[5]
```

מערכים רב מימדיים:

הלן דוגמה למערך דו-מימדי:

```
int matrix_2d[10][3];
```

זהו מערך של 10 מערכים, כל אחד בעלי 3 איברים. כדי לגשת לאיבר השני של המערך השלישי, נרשום

```
/* Access to the second element of the third array. */
matrix_2d[3][2] = 2;
```

גם כאן, כמובן, יש להיזהר מגלישה:

```
/* Error! out of range */
matrix_2d[2][10] = 2;
```

אין מניעה לעצור בהכרח בשני מימדים. אפשר להגדיר מערך בעל מספר שרירותי של מימדים. להלן דוגמה למערך בעל שישה מימדים:

```
/* A 6 dimensional array. */
```

```
int matrix_6d[5][8][2][9][3][1];
```

גודל המערך

יש לשים לב שרק בתוך הבלוק שבו הוגדר מערך מותר להשתמש בקוד הבא כדי להגדיר את גודל המערך

```
int length = sizeof(arr)/sizeof(arr[0])
```

מערך סטטי

מערך גלובאלי (סטטי) חייב לקבל ערכי אתחול קבועים.

```
1. #include <stdio.h>
2. #define N 8;
3. int  a[5] = {2,1,4+6,N,N-1};
4. void  func(int  n) {
5. …
6. }
7. int  main() {
8.  func(3);
9.  func(7);
10.       }
```

```
1.  #define  N  10
2.   typedef  int  Arr[N];
3.  void printArray(const Arr  a, int n) {
4.   int k;
5.   for (k=0; k<n ;k++)
6.    printf("%d ",a[k]);
7.   }
8.   void readArray(Arr  a, int n) {
9.    int  k;
10.  for (k=0; k<n; k++)
11.    scanf("%d",&a[k]);
12. }
13. int  main () {
14.  Arr  a;
15. int k;
16. readArray(a, sizeof(a)/sizeof(a[0]));
17. for (k=0; k< sizeof(a)/sizeof(a[0]);k++)
18.    printArray(a+k , sizeof(a)/sizeof(int) -
    k);
19. return 0;
20.}
```

ועבור מערך דו מימדי

```
#define M  3
#define N 4
typedef  int   Row[N];
typedef  int   Mat[M][N];
typedef  Row Matrix[M];
int  main(){
 Row  mat1 [m];
```

```
 Mat   mat2;

Matrix  mat3;

}
```

מחרוזות

מחרוזת ב C-מוגדרת על ידי מערך של משתנים מסוג .char לדוגמה:

```
char my_string[5];
```

היא הכרזה על מערך בשם my_string בעל מקום ל-5 תווים.

היות שמחרוזת היא מערך, ניתן לאתחל אותה כפי שמאתחלים כל מערך, או, לחלופין, אפשר להשים ערכים לאיבריה ממש כמו לכל מערך אחר. בשני המקרים יש לאתחל או לשים גם את התו הריק.

לדוגמה, אם נרצה לאתחל את המחרוזת my_string ל "wiki"-אפשר לאתחל אותה כך:

```
char my_string[5] = {'w', 'i', 'k', 'i', 0};
```

יש לשים לב לכך שבאתחול מהצורה הזאת חייב להגיע בסוף אפס או '0\'

או, לחלופין, אפשר להשים ערכים לאיבריה כך:

```
my_string[0] = 'w';
my_string[1] = 'i';
my_string[2] = 'k';
my_string[3] = 'i';
my_string[4] = 0;
```

השפה גם מאפשרת צורת אתחול מיוחדת למחרוזות, הקריאה יותר משתי האפשרויות הקודמות:

```
char my_string[] = "wiki";
```

כאשר עובדים עם מחרוזת חשוב לזכור לייבא את הספרייה של C לעבודה עם מחרוזות

```
#include <string.h>
```

strlen

```
char str[15] = "Boker Tov!";
int length = strlen( str );
printf("%d\n", length);
```

ידפיס את המספר 10, שכן ישנם עשרה תווים עד לתו הריק הראשון.

strcpy

היות שמחרוזת היא מערך, אי אפשר להעתיק מחרוזת על ידי השמה פשוטה כמו במשתנים רגילים. קטע הקוד הבא, לדוגמה, אינו חוקי:

```
char source[] = "Shalom";
char dest[] = source; /* ERROR! */
```

כדי להעתיק מחרוזת, יש להעתיק תו אחר תו. הפונקציה strcpy עושה זאת. לדוגמה:

```
char source[] = "Shalom";
char dest[15];

strcpy( dest, source );

printf("%s\n%s\n", source, dest);
```

ידפיס

```
Shalom
Shalom
```

strcpy מקבלת שתי מחרוזות, ומעתיקה את השנייה לתוך הראשונה, כולל תו סיום המחרוזת.

strcat

הפונקציה strcat משרשרת מחרוזת אחת לסוף של מחרוזת אחרת:

```
char source[] = "olam";
char dest[30] = "Shalom ";
strcat( dest, source );
printf("%s\n%s\n", source, dest);
```

טעויות נפוצות בעבודה עם מחרוזות:

כבר ראינו שבשפת C, אם מערך הוא בגודל 20, אז גישה לאיבר מעבר לכך, נניח 33, גולשת מתחום המערך, ויכולה לגרום לבעיות. חשוב להבין שקל מאד ליפול לבעיות אלו דווקא בעבודה עם מחרוזות. נתבונן, לדוגמה, בקטע הקוד הבא:

```
char src[] = "Hello, world!";
char dest[4];

strcpy(dest, src);
```

קטע קוד זה מכיל שגיאה, מפני שבמערך dest אין מספיק מקום להעתיק את תווי "Hello, world!".

המסקנה, על כן, היא שבעבודה עם מחרוזות, ובעיקר עם פונקציות המעתיקות דברים למחרוזות לדוגמה strcpy, strcat, או פונקציות הקלט שראינו - יש לנהוג בזהירות רבה.

עם תשומת לב מספיקה, אפשר להבטיח שמרבית הקריאות המטפלות במחרוזות יעבדו בצורה בטוחה. כך, לדוגמה, לפני כל קריאה ל ,strcpy-אפשר לבדוק את אורכי המחרוזות, ולבצע את הפעולה רק אם אין סכנת גלישה. ישנה בעייה מיוחדת בקלט, מפני שאין דרך לדעת מראש כמה תווים יקליד המשתמש. קטע הקוד הבא, לדוגמה, משתמש ב scanf-ללא קובע רוחב:

```
char name[10];
printf("Please enter your name:\n");
scanf("%s", name);
```

חשוב להבין שזהו קטע קוד בעייתי ביותר. בכלל, כשעוסקים בקלט מחרוזות (אולי על ידי שימוש בפונקציות ספריה אחרות), חשוב לשים לב לנקודה. פתרון אפשרי:

```
  char a[5];

  scanf("%4s", a);
```

תקלוט לכל היותר 4 תווים למחרוזת.a

strtok

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

int main()
{
   const char str[80] = "This is - www.tutorialspoint.com - website";
   const char s[2] = "-";
   char *token;

   /* get the first token */
   token = strtok(str, s);

   /* walk through other tokens */
   while( token != NULL )
   {
      printf( " %s\n", token );

      token = strtok(NULL, s);
   }

   return(0);
}
```

קטע הקוד יפיק את הפלט הבא:

```
This is
www.tutorialspoint.com
website
```

`char *strcpy(s,ct)`	copy string `ct` to string `s`, including `'\0'`; return `s`.
`char *strncpy(s,ct,n)`	copy at most `n` characters of string `ct` to `s`; return `s`. Pad with `'\0'`s if `t` has fewer than `n` characters.
`char *strcat(s,ct)`	concatenate string `ct` to end of string `s`; return `s`.
`char *strncat(s,ct,n)`	concatenate at most `n` characters of string `ct` to string `s`, terminate `s` with `'\0'`; return `s`.
`int strcmp(cs,ct)`	compare string `cs` to string `ct`; return <0 if `cs<ct`, 0 if `cs==ct`, or >0 if `cs>ct`.
`int strncmp(cs,ct,n)`	compare at most `n` characters of string `cs` to string `ct`; return <0 if `cs<ct`, 0 if `cs==ct`, or >0 if `cs>ct`.
`char *strchr(cs,c)`	return pointer to first occurrence of `c` in `cs` or `NULL` if not present.
`char *strrchr(cs,c)`	return pointer to last occurrence of `c` in `cs` or `NULL` if not present.

`char *strcpy(s,ct)`	copy string `ct` to string `s`, including `'\0'`; return `s`.
`char *strncpy(s,ct,n)`	copy at most `n` characters of string `ct` to `s`; return `s`. Pad with `'\0'`s if `t` has fewer than `n` characters.
`char *strcat(s,ct)`	concatenate string `ct` to end of string `s`; return `s`.
`char *strncat(s,ct,n)`	concatenate at most `n` characters of string `ct` to string `s`, terminate `s` with `'\0'`; return `s`.
`int strcmp(cs,ct)`	compare string `cs` to string `ct`; return <0 if `cs<ct`, 0 if `cs==ct`, or >0 if `cs>ct`.
`int strncmp(cs,ct,n)`	compare at most `n` characters of string `cs` to string `ct`; return <0 if `cs<ct`, 0 if `cs==ct`, or >0 if `cs>ct`.
`char *strchr(cs,c)`	return pointer to first occurrence of `c` in `cs` or `NULL` if not present.
`char *strrchr(cs,c)`	return pointer to last occurrence of `c` in `cs` or `NULL` if not present.

`void *memcpy(s,ct,n)`	copy `n` characters from `ct` to `s`, and return `s`.
`void *memmove(s,ct,n)`	same as `memcpy` except that it works even if the objects overlap.
`int memcmp(cs,ct,n)`	compare the first `n` characters of `cs` with `ct`; return as with `strcmp`.
`void *memchr(cs,c,n)`	return pointer to first occurrence of character `c` in `cs`, or `NULL` if not present among the first `n` characters.
`void *memset(s,c,n)`	place character `c` into first `n` characters of `s`, return `s`.

מצביעים

אמנם, איננו יכולים לקבוע היכן יישבו משתנים בזיכרון, אך יש באפשרותנו לברר את כתובתם, לפי הצורה:

```
&<var>
```

מצהירים על מצביע כך:

```
<type> *<name>;
```

כפי שראינו, מצביע הוא פשוט משתנה שערכו כתובת בזיכרון. כמו כל משתנה אחר, אפשר לאתחל מצביע, ואפשר לשים לו ערך. כל פעולה כזו תשנה להיכן הוא מצביע.

לדוגמה:

```
int m0, m1;
int *p;

p = &m0;
p = &m1;
```

מצביע לא מאותחל מצביע למקום שרירותי בזיכרון. לפעמים אי אפשר לאתחל מצביע לכתובת חוקית. כדי לסמן שהמצביע אינו מצביע למקום חוקי בזיכרון, נהוג לאתחל אותו לכתובת 0. לשם הקריאות, יש המשתמשים בקבוע NULL שמוגדר כ0. כך, לדוגמה, שכיח לראות קוד כזה:

```
int *p = NULL;
```

כאשר מצביע מצביע לכתובת חוקית, אפשר לגשת לערך באותה כתובת. עושים כך בצורה הבאה:

```
*<ptr>
```

העברת נתונים לפונקציה by address:

```
void swap(int *px, int *py)
{
  char temp = *px;
  *px = *py;
  *py = temp;
}
```

כדי לקרוא לפונקציה כעת מעבירים שתי **כתובות למצביעים**.

```
swap(&a,&b);
```

אריתמטיקה של מצביעים:

p + 1 ו p + 2 הם מצביעים לתו אחד קדימה, ולשני תווים קדימה. אם p הוא מצביע לתו, והוא מצביע לכתובת 2000, אז p + 1 מצביע לכתובת 2001, ו p + 2 מצביע לכתובת 2002.

עד כאן המצב זהה לחיבור רגיל, אולם כעת נראה משהו שונה. נתבונן בקטע הקוד הבא:

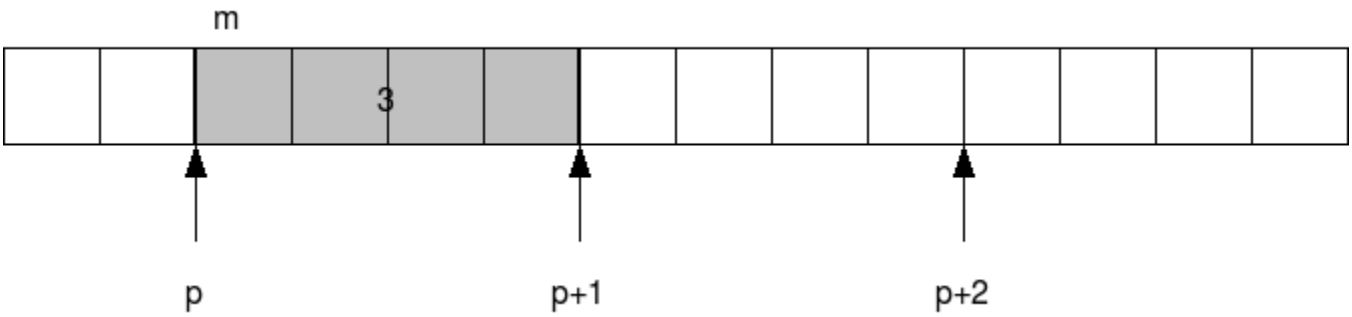
```
int m = 3;

int *p = &m;
```

```
int *p_plus_1, *p_plus_2;

p_plus_1 = p + 1;
p_plus_2 = p + 2;
```

מה משמעות p + 1 ו p + 2 כעת? התרשים הבא מראה את התשובה לכך:



p + 1 ו p + 2 הם מצביעים לשלם אחד קדימה, ולשני שלמים קדימה. אם p הוא מצביע לשלם, והוא מצביע לכתובת 2000, אז במחשב שבו שלם תופס 4 בתים p + 1 יערכו 2004, ו p + 2 יערכו 2008.

נוכל לסכם זאת כך. אם p הוא מצביע לטיפוס t כלשהו, אז ערך p + i הוא ערך p ועוד i * sizeof(t). במילים אחרות p + i מצביע ל i פעמים t קדימה.

באותו אופן גם חיסור שלם ממשתנה. אם p הוא מצביע לטיפוס t כלשהו, אז ערך p - i הוא ערך p פחות i * sizeof(t). במילים אחרות p - i מצביע ל i פעמים t אחורה..

לאחר שראינו מה משמעות חיבור וחיסור שלם למצביע, לא קשה לנחש את משמעותו של קוד כזה:

```
int *p = &m;

p++;
```

ראינו בהגדלה עצמית והקטנה עצמית בפעולות חשבוניות שזוהי כתיבה מקוצרת לקוד:

```
int *p = &m;

p = p + 1;
```

המשמעות של ארבע השורות הבאות, לכן:

```
p++;
++p;
p--;
--p;
```

היא, בהתאמה:

- קידום בדיעבד של המצביע p לאיבר הבא

- קידום לכתחילה של המצביע p לאיבר הבא
- הסגה בדיעבד של המצביע p לאיבר הקודם
- הסגה לכתחילה של המצביע p לאיבר הקודם

הפרש בין מצביעים:

ההפרש בין שני מצביעים מאותו סוג, הוא מספר האיברים ביניהם בזיכרון.

לדוגמה, נניח ש p0-מצביע לכתובת 2008 p1 ,מצביע לכתובת 2000, אלה מצביעים לשלמים, ובמחשב זה תופס שלם 4 בתים. אז:

```
p0 - p1
```

הוא 2.

ההפרש בין שני מצביעים מסוגים שונים אינו ניתן לחישוב ב - Cהמהדר יתלונן על הניסיון לעשות זאת.

אין אפשרות לבצע שום פעולה חשבונית אחרת על מצביעים. לדוגמה, אין אפשרות לחלק מצביע בקבוע, להכפיל מצביע במצביע, וכולי.

יש לקחת מקרה מסוים כאשר שאנחנו מתעסקים עם מצביע מסוג *void , בסוג מצביעים לא מוגדר בצורה מפורשת איך מתבצעים חישוביים חשבוניים ולכן אנו נבצע המרה ל *char ולבצע את החישובים בהתאם לכך שגודל של char הוא בית אחד.

מצביעים קבועים ומצביעים לערכים קבועים

מה משמעות מצביעים קבועים, אם .והערך המוצבע, יעד ההצבעה :מצביעים הם משתנים ששני ערכים קשורים אליהם כן? אפשר לחשוב על שתי אפשרויות:

- מצביעים שאין אפשרות לשנות את יעד ההצבעה שלהם.
- מצביעים שאין אפשרות לשנות את הערך המוצבע שלהם.

לדוגמה) הוא טיפוס tמאפשרת להגדיר מצביעים כקבועים לכל אחת מארבעת השילובים האפשריים. נניח ש C שפת int), iname אז שם. הוא:

```
<t> *<name>
```

מגדיר משתנה שמותר לשנות את יעד ההצבעה שלו, ומותר לשנות את הערך המוצבע שלו.

```
<t> *const <name>
```

מגדיר משתנה ש**אסור** לשנות את יעד ההצבעה שלו, ומותר לשנות את הערך המוצבע שלו.

```
const <t> *<name>
```

מגדיר משתנה שמותר לשנות את יעד ההצבעה שלו, **ואסור** לשנות את הערך המוצבע שלו.

```
const <t> *const <name>
```

מגדיר משתנה ש**אסור** לשנות את יעד ההצבעה שלו, **ואסור** לשנות את הערך המוצבע שלו.

התוכנית הבאה, לדוגמה, חוקית:

```
char a;
const char b = 'b';

char *p = &a;

*p = 'c';
```

התוכנית הבאה אינה חוקית:

```
char a;
const char b = 'b';

char *const p = &a;

p = &b; /* Error: can't change the destination of p! */
```

גם התוכנית הבאה אינה חוקית:

```
char a;
const char b = 'b';

const char *p = &b;

*p = 'c'; /* Error: can't change the value pointed by p! */
```

מצביעים לפונקציות

נשתמש במצביעים לפונקציות כדי ליצור פונקציות גנריות, מוטיביציה ? ניתן לחשוב על דוגמאת המיון גנרי , לאחר שכתבו את לוגיקת המיון עבור מערך מסוג אחד אין טעם לכתוב אותו שוב ושוב. עבור סוגי נתונים שונים.

```
#include <stdio.h>

enum {DECIMAL = 0, OCTAL, HEXADECIMAL};

void printDecilal(int n)

{

printf("%d ", n);

}
```

```
void printOctal(int n)
{
printf("%o ", n);
}
void printHexa(int n)
{
printf("%x ", n);
}
void printNum(int n, int nMode, void (*print[])(int))
{
(*print[nMode])(n);
}
void main()
{
void (*ptrprint[])(int) = {printDecilal,
printOctal,
printHexa };
printNum(30, DECIMAL, ptrprint);
printNum(30, OCTAL, ptrprint);
printNum(30, HEXADECIMAL, ptrprint);
}
/***** Output *****
30 36 1e
*******************/
```

הפעלת פונקציה שחזרה כתשובה לפונקציה אחרת

```
int square(int x){
        return x*x;
}
int (*f())(int) {
```

```
   return square;
}

int main(){
    int   x=5;
      printf("%d", f()(x));
      return 0;

}
```

מיון גנרי מתוך שיעורי בית

```
#include <stdio.h>
#include <string.h>
#include <math.h>
int genericStrCmp(const void*  P1,const void*  P2){
        return strcmpi((char*)P1,(char*)P2);
}
int genericStrPointerCmp(const void*   P1,const void*  P2){
        return strcmpi(*(char**)P1,*(char**)P2);
}
void strPointerSwap(void* P1,void* P2){
        char* P =*(char**)P1;
        *(char**)P1=*(char**)P2;
        *(char**)P2=P;
}
void strSwap(void* P1,void* P2){
        char* ptr1=(char*)P1;
        char* ptr2=(char*)P2;
        int l=strlen(ptr1);
        if(strlen(ptr2)>l)
```

```
                l=strlen(ptr2);
        for(int i=0;i<l;i++){
        char temp = '\0';
        temp = *(ptr1+i);
        *(ptr1+i) = *(ptr2+i);
        *(ptr2+i)=temp;
        }
}

int intCmp(const void* P1,const void* P2){

        return *(int*)P1 - *(int*)P2;
}
int doubleCmp(const void* P1,const void* P2){
        if( *(double*)P1 - *(double*)P2 == 0)
                return 0;
        else
                if( *(double*)P1 - *(double*)P2 < 0 ) return -1;
                else
                return 1;
}
void doubleSwap(void* P1,void* P2){
        double temp = *(double*)P1;
        *(double*)P1 = *(double*)P2;
        *(double*)P2 = temp;
}
void intSwap(void* P1,void* P2){
        int* a1 = (int*)P1;
        int* a2 = (int*)P2;
```

```
	int temp = *a1;
	*a1=*a2;
	*a2=temp;
}

int genericFindingIndexOFMaxElement(void* A,int size,int typeSize, int (*fP)(const void*, const void*)){
	int max=0;
	for(int i=1;i<size;i++){
	if(fP((char*)A+max*typeSize,(char*)A+typeSize*i)<0)
		max = i ;
	}
	return max;
}
void genericSort(void* A,int size,int typeSize, int (*cmp)(const void*,const void*),void
(*swap)(void*,void*)){
	for (int i=size-1;i>=0;i--){
		int m = genericFindingIndexOFMaxElement(A,i+1,typeSize,cmp);
				swap((char*)A+typeSize*i,(char*)A+typeSize*m);
	}
}
int main(){
	int A[] = {2,3,1,4,5,3,2,1,9};
	int max = genericFindingIndexOFMaxElement(A,sizeof(A)/sizeof(A[0]),sizeof(A[0]),intCmp);
	genericSort(A,sizeof(A)/sizeof(A[0]),sizeof(A[0]),intCmp,intSwap);
	double B[] = {2.5,3.2,4,1.5,7,10.4,0.2,0.3,0.1};
	genericSort(B,sizeof(B)/sizeof(B[0]),sizeof(B[0]),doubleCmp,doubleSwap);
	printf("%d ",A[max]);
	char   c[][5]={"abc","xy","ac"};
	char  *d[]={"abc","xy","ac"};
```

```
    genericSort(d,sizeof(d)/sizeof(d[0]),sizeof(d[0]),genericStrPointerCmp,strPointerSwap);
    genericSort(c,sizeof(c)/sizeof(c[0]),sizeof(c[0]),genericStrCmp,strSwap);
    return 0;
}
```

החלפה מספר מסוים של בתים

```c
#include <stdio.h>

void MySwap(void *a, void *b, size_t size)
{
    int i;
    unsigned char *tmpA = (unsigned char *) a, *tmpB = (unsigned char *)
b;
    for(i=0; i<size; i++)
    {
        unsigned char tmp = *tmpA;
        *tmpA = *tmpB;
        *tmpB = tmp;
        ++tmpA;
        ++tmpB;
    }
}

int main()
{
    int i = 2, j = 4;
    printf("A: %d B: %d\n", i, j);
    MySwap(&i, &j, sizeof(int));
    printf("A: %d B: %d\n", i, j);
    float d1 = 1.5, d2 = 3.2;
    printf("A: %g B: %g\n", d1, d2);
    MySwap(&d1, &d2, sizeof(float));
    printf("A: %g B: %g\n", d1, d2);
    printf("Messy swap: A: %g B: %d\n", d1, i);
    MySwap(&d1, &i, sizeof(int));
    printf("Messy swap: A: %d B: %g\n", i, d1);
    return 0;
}
```

מבנים

מבנה מאפשר להגדיר טיפוס נתונים חדש, כאשר הוא מוגדר משדות של מבני נתונים שהוגדרו קודם או משתנים פרימיטיביים .

ניגשים לשדות משתנה בצורה:

```
<name>.<field_name>
```

כאשר name הוא שם המשתנה field_name, הוא שם השדה.

לדוגמה, נניח ש shokoהוא משתנה מסוג struct item. כדי לקבוע את מחירו ל12.90, נכתוב:

```
shoko.price = 12.90;
```

כדי לקבוע את שמו כ "shoko", נכתוב:

```
strcpy(shoko.name, "shoko");
```

כדי להדפיס את שמו ואת מחירו, נכתוב:

```
printf("The price of %s is %f", shoko.name, shoko.price);
```

נניח ש p הוא מצביע למבנה. נוכל לגשת לאיבר שלו בצורה:

```
(*p).<field_name>
```

לדוגמה, אם p הוא מצביע ל struct itemאז את שלוש הדוגמאות הקודמות אפשר לכתוב כך:

```
(*p).price = 12.90;
strcpy((*p).name, "shoko");
printf("The price of %s is %f", *(p.name), *(p.price));
```

גישה למבנה על ידי מצביע היא מהפעולות השכיחות בשפת C. השפה לכן כוללת את הצורה המקוצרת:

```
p-><field_name>
```

שמשמעותה זהה. נוכל, לכן, לכתוב את שלוש הדוגמאות בצורה קצרה יותר:

```
p->price = 12.90;
strcpy(p->name, "shoko");
printf("The price of %s is %f", p->name, p->price);
```

דוגמא מבנה המייצג חשבון בנק:

```
#include <stdio.h>

#include <string.h>

struct Account

{

long lId;
```

```
double dBalance;
char strOwnerName[30];
}Account;
void print(struct Account ac)
{
printf("Accout. Number:%d, Balance:%.2lf, Owner:%s\n",
ac.lId, ac.dBalance, ac.strOwnerName);
}
void main()
{
struct Account ac1={111, -2000, "Jack"}, ac2, ac3=ac1;
ac2.lId = 222;
ac2.dBalance = 330;
strcpy(ac2.strOwnerName, "John");
print(ac1);
print(ac2);
print(ac3);
}
/***** Output *****
Accout. nunber: 111, Balance: -2000.00, Owner: Jack
Accout. nunber: 222, Balance: 330.00, Owner: John
Accout. nunber: 111, Balance: -2000.00, Owner: Jack
******************/
```

מערך של מבנים

```
#include <stdio.h>
#define CAPACITY 100
#define TRUE 1
#define FALSE 0
typedef struct
```

```
{
char strName[30];
long lPhone;
}Friend;
typedef struct
{
Friend friendsArray[CAPACITY];
int nNumberOfFriends;
}PhoneBook;
short addFriend(PhoneBook* ptrPB)
{
if(ptrPB->nNumberOfFriends == CAPACITY)
{
printf("Too many friends...\n");
return FALSE;
}
printf("\nEnter Name: ");
scanf("%s",
&(ptrPB->friendsArray[ptrPB->nNumberOfFriends].strName));
printf("Enter Phone Number: ");
scanf("%d",
&(ptrPB->friendsArray[ptrPB->nNumberOfFriends].lPhone));
ptrPB->nNumberOfFriends++;
return TRUE;
}
short deleteFriend(PhoneBook* ptrPB, int index)
{
char chAnswer;
int i;
```

```
if(index < 1 || index > ptrPB->nNumberOfFriends)
{
return FALSE;
}
printf("\nAre you sure you wonna delete %s, (Y / N)? ",
ptrPB->friendsArray[index-1].strName);
do
{
scanf("%c", &chAnswer);
}while(chAnswer != 'Y' && chAnswer != 'y' &&
chAnswer != 'N' && chAnswer != 'n');
if(chAnswer != 'Y' && chAnswer != 'y')
{
return FALSE;
}
for(i=index-1; i<ptrPB->nNumberOfFriends-1; i++)
{
ptrPB->friendsArray[i] = ptrPB->friendsArray[i+1];
}
ptrPB->nNumberOfFriends--;
return TRUE;
}
void printBook(PhoneBook PB)
{
int i;
printf("\nYour Phone Book: \n");
for(i = 0; i < PB.nNumberOfFriends; i++)
{
printf("%d. %10s %9d \n", i+1,
```

```
PB.friendsArray[i].strName,
PB.friendsArray[i].lPhone);
}
}
```

```
void main()
{
PhoneBook pbMyBook;
pbMyBook.nNumberOfFriends = 0;
addFriend(&pbMyBook);
addFriend(&pbMyBook);
addFriend(&pbMyBook);
printBook(pbMyBook);
deleteFriend(&pbMyBook, 2);
printBook(pbMyBook);
}
/***** Output *****
Enter Name: Yoheved
Enter Phone Number: 7521111
Enter Name: Zoshinqua
Enter Phone Number: 8476622
Enter Name: Zeldka
Enter Phone Number: 6423355
Your Phone Book:
1. Yoheved 7521111
2. Zoshinqua 8476622
3. Zeldka 6423355
Are you sure you wonna delete Zoshinqua, (Y / N)? y
```

Your Phone Book:

1. Yoheved 7521111

2. Zeldka 6423355

**********/

הקצאות דינמיות

כדי להקצות זיכרון, אפשר להשתמש בפונקציה malloc, בקריאה מהצורה הבאה:

```
malloc(<total_size>)
```

כאשר total_size הוא הגודל (בבתים) שאותו רוצים להקצות. לדוגמה:

```
malloc(sizeof(int))
```

היא בקשה להקצות מספיק בתים לשלם יחיד, ואילו:

```
malloc(sizeof(int) * 80)
```

היא בקשה להקצות מספיק בתים ל-80 שלמים רצופים.

לחלופין, אפשר להשתמש בפונקציה calloc, בקריאה מהצורה הבאה:

```
calloc(<num>, <size>)
```

כאשר num * size הוא הגודל (בבתים) שאותו רוצים להקצות. בפרט, אם num הוא מספר משתנים, ו sizeהוא גודל כל אחד מהם, אז הקריאה הנ"ל תקצה מקום רציף ל numמשתנים. לדוגמה:

```
calloc(80, sizeof(int));
```

היא בקשה להקצות מספיק בתים ל80 שלמים רצופים.

שתי הפונקציות נבדלות בעיקר בכך ש calloc,מעבר להקצאת זיכרון, גם מאפסת את קטע הזיכרון אותו היא מקצה. למעט זאת, אין הבדל בין הקריאות הבאות:

```
malloc(80 * sizeof(int));
calloc(80, sizeof(int));
calloc(sizeof(int), 80);
```

הן malloc והן calloc מחזירות את כתובת הזיכרון שאותו היקצו. טיפוס הערך המוחזר הוא) * voidשהוסבר כאן .(אם הפונקציות נכשלו בהקצאה, הכתובת שיחזירו תהיה .NULLלרוב, לכן, יש שתי פעולות שיש לבצע על הערך המוחזר:

1. לשים אותו למשתנה מצביע.
2. לבדוק האם הערך המוחזר הוא NULL, ובמקרה שכן, לטפל בכישלון ההקצאה.

דוגמא הקצאה של מערך חד ודו מימדי

```
1.  void  printArray ( int arr [ ], int arrSize )
2.  {
3.    int  i;
4.    for ( i = 0; i < arrSize ; i ++ )
5.      printf ("%d  ", arr [ i ]);
6.   putchar('\n');
7.  }
8.  int * AllocationArr ( int * pArrSize ) {
9.    int * pArr = 0, *pTmp;
10.   int newNumber , arrSize = 0;
11.  while ( scanf ("%d", & newNumber) == 1 ) {
12.   arrSize ++;
13.   pTmp = (int *) realloc( pArr, arrSize * sizeof (int));
14.  if ( !pArr ) { arrSize-- ; break ;}
15.  pArr = pTmp;
16.  pArr [ arrSize - 1 ] = newNumber ;
17. }
18. pArrSize = arrSize ;
19. return pArr ;
20. }
21. int main()
22. {
23.   int * pArr;
24.   int arrSize;
25.   pArr = AllocationArr( &arrSize );
26.   if ( pArr == 0 )
27.     return 0;
28.   printArray ( pArr, arrSize);
29.   free ( pArr );
30.  return 0;
31. }
```

```
 void matrix_Print ( int ** pp , int  row, int  col )
{
  int i, j;
  for ( i = 0; i < row ; i++ ) {
   for ( j = 0 ; j < col ; j++ )
     printf ( "%d  ", pp [ i ][ j ]);
   printf ( "\n" );
  }
}
int ** matAlloc(int row, int col )
{
  int ** pp;
  int  i, j, counter = 1;
  ppTemp = (int ** ) malloc ( sizeof ( int * ) * row);
  if (!pp) return NULL;
  for ( i = 0; i < row ; i ++ ) {
    pp [ i ] = (int * ) malloc ( sizeof ( int ) * col );
    if (!pp[i]) {
      for (k=0; k < i ; k++)  free(pp[k]); free(pp);
      return NULL;
     }
   }
 return pp;
}
void matrix_Free ( int ** pp , int row ) {
  int i;
   for ( i = 0; i < row ; i++ )
    if ( pp [ i ] )
      free ( pp [ i ] );
    if ( pp )
      free ( pp );
}
void matRead( int ** pp , int  row, int  col ) {
  int i, j;
  for ( i = 0; i < row ; i++ ) {
   for ( j = 0 ; j < col ; j++ )
     printf ( "%d  ", & pp [ i ][ j ]);
  }
}
int main() {
   int ** ppMatr, rows, cols;
   scanf ("%d %d", & rows, & cols );
   ppMatr = matrAlloc (rows,cols);
   matRead(ppMatr, rows, cols);
  matPrint (ppMatr, rows, cols);
  matFree (ppMatr, rows);
 return 0;
}
```

```
int **2d_array; // an array of int arrays (a pointer to pointers to ints)

// declaration and allocation:

// allocate an array of N pointers to ints
// malloc returns the address of this array (a pointer to (int *)'s)
2d_array = (int **)malloc(sizeof(int *)*N);

// for each row, malloc space for its buckets and add it to
// the array of arrays
for(i=0; i < N; i++) {
  2d_array[i] = (int *)malloc(sizeof(int)*M);
}

// in memory:
// 2d_array -----> | *-|-------> [ 0, 0, 0, ... , 0]  row 0
//                 | *-|-------> [ 0, 0, 0, ... , 0]  row 1
//                 |...|                    ...
//                 | *-|-------> [ 0, 0, 0, ... , 0]

// access using [] notation:
//  2d_array[i] is ith bucket in 2d_array, which is the address of
// a 1d array, on which you can use indexing to access its bucket value
for(i=0; i < N; i++) {
  for(j=0; j < M; j++) {
     2d_array[i][j] = 0;
  }
}
```

```
// access using pointer arithmetic (NOT all N*M buckets are contigous)
// but each row's buckets are
int *ptr;
for(i=0; i < N; i++) {
  *ptr = 2d_array[i];  // set pointer to address of bucket 0 in row i
  for(j=0; j < M; j++) {
    *ptr = 0;
    ptr++;
  }
```

שינוי הקצאה

שינויי הקצאות אפשר לעשות בעזרת הפונקציה realloc, על ידי קריאות מהצורה הבאה:

```
realloc(<old_ptr>, <total_size>)
```

כאשר old_ptr היא כתובת הזיכרון המוקצאת הנוכחית, ו total_sizeהוא הגודל החדש המבוקש (בבתים).

מערכת ההפעלה תנסה לראות האם אפשר לשנות את רצף הזיכרון הנוכחי לגודל המבוקש. אם הדבר אפשרי, הפונקציה תחזיר את כתובת הזיכרון של הרצף הנוכחי כ) * voidבדיוק כפי שראינו מקודם בהקצאה .(אם הדבר אינו אפשרי, היא תבדוק האם יש רצף אחר מתאים בזיכרון. אם היא הצליחה, היא תעתיק את תוכן הרצף הנוכחי לרצף החדש, תשחרר את הרצף הנוכחי, ותחזיר את כתובת הרצף החדש. אם אין רצף אחר מתאים בזיכרון, היא לא תשנה כלום בזיכרון (ובפרט, לא תשחרר את הרצף הנוכחי), ותחזיר NULL כדי לסמן שלא הצליחה.

Memcpy

The following example shows the usage of memcpy() function.

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

int main ()
{
   const char src[50] = "http://www.tutorialspoint.com";
   char dest[50];

   printf("Before memcpy dest = %s\n", dest);
   memcpy(dest, src, strlen(src)+1);
   printf("After memcpy dest = %s\n", dest);

   return(0);
}
```

Let us compile and run the above program, this will produce the following result:

```
Before memcpy dest =
After memcpy dest = http://www.tutorialspoint.com
```

מבני נתונים

ב C נשתמש במבנים והקצאות דינמניות כדי לממש מבני נתונים שונים נדגים כיצד אנו ממשים רשימה מקושרת ועץ בינארי.

רשימה מקושרת כדי לבדוק אם משפט הוא פלינדרום

```
#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

typedef struct node {

char ch;

struct node * next;

} Node, *NodePtr;

int main() {

NodePtr getPhrase();

NodePtr reverseLetters(NodePtr);

int compare(NodePtr, NodePtr);

NodePtr phrase, s1, s2;

printf("Type a phrase. (To stop, press 'Enter' only): ");

phrase = getPhrase();

while (phrase != NULL) {

s1 = reverseLetters(phrase);

s2 = reverseLetters(s1);

if (compare(s1, s2) == 0) printf("is a palindrome\n");

else printf("is not a palindrome\n");

printf("Type a word. (To stop, press 'Enter' only): ");

phrase = getPhrase();

}
```

```
} //end main
NodePtr getPhrase() {
NodePtr top = NULL, last, np;
char c = getchar();
while (c != '\n') {
np = (NodePtr) malloc(sizeof(Node));
np -> ch = c;
np -> next = NULL;
if (top == NULL) top = np;
else last -> next = np;
last = np;
c = getchar();
}
return top;
} //end getPhrase
NodePtr reverseLetters(NodePtr top) {
NodePtr rev = NULL, np;
char c;
while (top != NULL) {
c = top -> ch;
if (isalpha(c)) { // add to new list
np = (NodePtr) malloc(sizeof(Node));
np -> ch = tolower(c);
np -> next = rev;
rev = np;
}
top = top -> next; //go to next character of phrase
}
```

```
return rev;
} //end reverseLetter
int compare(NodePtr s1, NodePtr s2) {
//return -1 if s1 < s2, +1 if s1 > s2 and 0 if s1 = s2
while (s1 != NULL) {
if (s1 -> ch < s2 -> ch) return -1;
else if (s1 -> ch > s2 -> ch) return 1;
s1 = s1 -> next;
s2 = s2 -> next;
}
return 0;
}
```

דוגמא לרשימה מקושרת דו כיוונית גנרית עם זנב

```
typedef struct _node {
void *data;
struct _node *next;
} Node;
typedef struct _linkedList {
Node *head;
Node *tail;
Node *current;
} LinkedList;
```

We will develop several functions that use these structures to support linked list

functionality:

void initializeList(LinkedList*) Initializes the linked list

void addHead(LinkedList*, void*) Adds data to the linked list's head

void addTail(LinkedList*, void*) Adds data to the linked list's tail

void delete(LinkedList*, Node*) Removes a node from the linked list

Node *getNode(LinkedList*, COMPARE, void*) Returns a pointer to the node containing a specific data item

void displayLinkedList(LinkedList*, DISPLAY) Displays the linked list

```
void initializeList(LinkedList *list) {
list->head = NULL;
list->tail = NULL;
list->current = NULL;
}
void addHead(LinkedList *list, void* data) {
Node *node = (Node*) malloc(sizeof(Node));
node->data = data;
if (list->head == NULL) {
list->tail = node;
node->next = NULL;
} else {
node->next = list->head;
}
list->head = node;
}
void addTail(LinkedList *list, void* data) {
Node *node = (Node*) malloc(sizeof(Node));
node->data = data;
node->next = NULL;
if (list->head == NULL) {
list->head = node;
} else {
list->tail->next = node;
}
list->tail = node;
}
```

```
Node *getNode(LinkedList *list, COMPARE compare , void* data) {
Node *node = list->head;
while (node != NULL) {
if (compare(node->data, data) == 0) {
return node;
}
node = node->next;
}
return NULL;
}
void delete(LinkedList *list, Node *node) {
if (node == list->head) {
if (list->head->next == NULL) {
list->head = list->tail = NULL;
} else {
list->head = list->head->next;
}
} else {
Node *tmp = list->head;
while (tmp != NULL && tmp->next != node) {
tmp = tmp->next;
}
if (tmp != NULL) {
tmp->next = node->next;
}
}
free(node);
}
void displayLinkedList(LinkedList *list, DISPLAY display) {
```

```
printf("\nLinked List\n");
Node *current = list->head;
while (current != NULL) {
display(current->data);
current = current->next;
}
}
```

עץ בינארי

```
int numNodes(TreeNodePtr root) {
if (root == NULL) return 0;
return 1 + numNodes(root -> left) + numNodes(root -> right);
}
int numLeaves(TreeNodePtr root) {
if (root == NULL) return 0;
if (root-> left == NULL && root-> right == NULL) return 1;
return numLeaves(root-> left) + numLeaves(root-> right);
}
int height(TreeNodePtr root) {
if (root == NULL) return 0;
return 1 + max(height (root-> left), height (root-> right));
}
```

where max can be defined as follows:

```
#define max(a, b) ((a) > (b) ? (a) : (b))
deleteNode(TreeNode T) {
if (T == null) return null
if (right(T) == null) return left(T) //cases 1 and 2b
R = right(T)
if (left(T) == null) return R //case 2a
if (left(R) == null) {
```

```
left(R) == left(T)
return R
}
while (left(R) != null) { //will be executed at least once
P = R
R = left(R)
}
//R is pointing to the in-order successor of T;
//P is its parent
left(R) = left(T)
left(P) = right(R)
right(R) = right(T)
return R
} //end deleteNode
TreeNodePtr findOrInsert(BinaryTree bt, NodeData d) {
//searches for d; if found, return pointer to node containing d
//else insert a node containing d and return pointer to new node
TreeNodePtr newTreeNode(NodeData);
if (bt.root == NULL) return newTreeNode(d);
TreeNodePtr curr = bt.root;
int cmp;
while ((cmp = strcmp(d.word, curr -> data.word)) != 0) {
if (cmp < 0) { //try left
if (curr -> left == NULL) return curr -> left = newTreeNode(d);
curr = curr -> left;
}
else { //try right
if (curr -> right == NULL) return curr -> right = newTreeNode(d);
curr = curr -> right;
```

```
}
}
//d is in the tree; return pointer to the node
return curr;
} //end findOrInsert
```

The creation of a new node is delegated to newTreeNode.

```
TreeNodePtr newTreeNode(NodeData d) {
TreeNodePtr p = (TreeNodePtr) malloc(sizeof(TreeNode));
p -> data = d;
p -> left = p -> right = NULL;
return p;
} //end newTreeNode
```

This allocates storage for the node, stores d in the data field, and sets the left and right pointers to NULL. It returns

a pointer to the node created

דוגמא תוכנית לספירת חזרות של מילים בעזרת עץ בינארי

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#define MaxWordSize 20
typedef struct {
char word[MaxWordSize+1];
int freq;
} NodeData;
typedef struct treeNode {
NodeData data;
```

```
struct treeNode *left, *right;
} TreeNode, *TreeNodePtr;
typedef struct {
TreeNodePtr root;
} BinaryTree;
int main() {
int getWord(FILE *, char[]);
TreeNodePtr newTreeNode(NodeData);
NodeData newNodeData(char [], int);
TreeNodePtr findOrInsert(BinaryTree, NodeData);
void inOrder(FILE *, TreeNodePtr);
char word[MaxWordSize+1];
FILE * in = fopen("wordFreq.in", "r");
FILE * out = fopen("wordFreq.out", "w");
BinaryTree bst;
bst.root = NULL;
while (getWord(in, word) != 0) {
if (bst.root == NULL)
bst.root = newTreeNode(newNodeData(word, 1));
else {
TreeNodePtr node = findOrInsert(bst, newNodeData(word, 0));
node -> data.freq++;
}
}
fprintf(out, "\nWords Frequency\n\n");
inOrder(out, bst.root); fprintf(out, "\n\n");
fclose(in);
fclose(out);
} // end main
```

```
int getWord(FILE * in, char str[]) {
// stores the next word, if any, in str; word is converted to lowercase
// returns 1 if a word is found; 0, otherwise
char ch;
int n = 0;
// read over non-letters
while (!isalpha(ch = getc(in)) && ch != EOF) ; //empty while body
if (ch == EOF) return 0;
str[n++] = tolower(ch);
while (isalpha(ch = getc(in)) && ch != EOF)
if (n < MaxWordSize) str[n++] = tolower(ch);
str[n] = '\0';
return 1;
} // end getWord
TreeNodePtr findOrInsert(BinaryTree bt, NodeData d) {
//searches for d; if found, return pointer to node containing d
//else insert a node containing d and return pointer to new node
TreeNodePtr newTreeNode(NodeData);
if (bt.root == NULL) return newTreeNode(d);
TreeNodePtr curr = bt.root;
int cmp;
while ((cmp = strcmp(d.word, curr -> data.word)) != 0) {
if (cmp < 0) { //try left
if (curr -> left == NULL) return curr -> left = newTreeNode(d);
curr = curr -> left;
}
else { //try right
if (curr -> right == NULL) return curr -> right = newTreeNode(d);
curr = curr -> right;
```

```
}
}
//d is in the tree; return pointer to the node
return curr;
} //end findOrInsert
TreeNodePtr newTreeNode(NodeData d) {
TreeNodePtr p = (TreeNodePtr) malloc(sizeof(TreeNode));
p -> data = d;
p -> left = p -> right = NULL;
return p;
} //end newTreeNode
void inOrder(FILE * out, TreeNodePtr node) {
if (node!= NULL) {
inOrder(out, node -> left);
fprintf(out, "%-15s %2d\n", node -> data.word, node -> data.freq);
inOrder(out, node -> right);
}
} //end inOrder
NodeData newNodeData(char str[], int n) {
NodeData temp;
strcpy(temp.word, str);
temp.freq = n;
return temp;
} //end newNodeData
```

פעולות על בתים

xor ^

x	y	x ^ y
0	0	0

0	1	1
1	0	1
1	1	0

or |

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

AND &

x	y	x& y
0	0	0
0	1	0
1	0	0
1	1	1

NOT ~

x_i	~ x_i
0	1
1	0

למעט האופטור NOT ניתן להשתמש באופרטור XOR OR AND כמו שאנו משתמשים באופרטורים ארטימטיים והוראות מהצורה הבאה חוקיים

x&=

x|=

x^=

Shift

יש לשים לב שבשימוש ב shift ימינה או שמאלה חייב להגיע מספר לאחר האופרנד.

left- <<

הזזת ביטים שמאלה שקול לכפל ב 2

right - >>

הזזת ביטים ימינה לא בדיוק שקול לחילוק ב 2.

פונקציות עזר

get

```
int getBit(int x,int p){
unsigned int i = 1<<p;
return x&i? 1:0;
}
```

printbit

```
void printBit(int x){
for(unsigned int i=1<<31;i;i>>=1)
   if(x&i)
   printf("%d",1);
   else
      printf("%d",0);
   printf("\n");

}
```

set

```
int setBit(int x,int p){
   int i = 1<<p;
   x|=i;
return x;
}
```

החלפת משתנים בעזרת XOR

```
#define SWAP(a, b) (((a) ^= (b)), ((b) ^= (a)), ((a) ^= (b)))
```

עבודה עם קבצים

פתיחת קובץ

```
FILE *f = fopen("try.txt" , "rt");
if(!f){
  /* Handle case where couldn't open file. */
}
```

תווים	משמעות
r	פתיחת קובץ קיים לקריאה. אם לא קיים קובץ בשם האמור, הפתיחה תכשל ותחזיר NULL.
w	פתיחת קובץ חדש לכתיבה. אם קיים כבר קובץ בשם האמור, תוכנו ימחק.
a	פתיחת קובץ קיים לכתיבת נתונים *בסופו*. התוכן החדש ידרוס את תו ה-EOF, וייכתב אחריו. אם הקובץ לא קיים, הוא יווצר תחילה.
r+	פתיחת קובץ קיים לקריאה וכתיבה. אם לא קיים קובץ בשם האמור, הפתיחה תכשל.
w+	פתיחת קובץ חדש לקריאה וכתיבה. אם קיים כבר קובץ בשם האמור, תוכנו ימחק.
a+	פתיחת קובץ קיים לכתיבת נתונים החדשים *בסופו*. תו סיום הקובץ יוזז לסוף הנתונים החדשים שנכתוב, כך שבהצגת תוכן הקובץ, יוצג כל התוכן. אם הקובץ לא קיים, הוא יווצר תחילה.

תוכן הקובץ

תווים	משמעות
t	קובץ טקסט. זוהי ברירת מחדל ולכן במידה ולא נכתוב שום סוג תוכן לאחר סוג הפתיחה המחשב יחשיב את הסוג כ-t.
b	קובץ בינרי

הפונקציה fclose סוגרת קובץ שנפתח על ידי fopen. משתמשים בה בצורה:

```
fclose(<f>);
```

כאשר f הוא מצביע ל FILE שהוחזר מקריאה מוצלחת מ-fopen.

פלט קבצים

ש לשים לב לשינויים הבאים:

- הארגומנט הראשון הוא מצביע ל-FILE.
- הפונקציה מדפיסה לקובץ שאותו מתאר ה-FILE.

- רצוי לבדוק את הערך המוחזר מהפונקציה, שכן כתיבה לקובץ עלולה להיכשל.

הפונקציה מחזירה מספר שלילי במקרה כישלון פלט. להלן דוגמה לבדיקה:

```
if( fprintf(f,"Hello world") < 0)
  ... /* Handle error. */
```

להלן דוגמה לשימוש ב:fprintf

```
#include <stdio.h>


int main()
{
  FILE *const f = fopen("new_file.txt", "wt");

  if(!f)
  {
    printf("Error: could not open file!\n");

    return -1;
  }

  fprintf(f, "This line will be written to the file.\n");
  fprintf(f, "Here is another line: %d + %d = %d", 2, 2, 2 + 2);

  fclose(f);

  return 0;
}
```

קלט קבצים

הפונקציה fscanf דומה מאד לפונקציה scanf

```
fscanf(f, "%d", &count);
```

יש לשים לב למעט השינויים הבאים:

- הארגומנט הראשון הוא מצביע ל.FILE
- הפונקציה קולטת מהקובץ שאותו מתאר ה.FILE
- חשוב במיוחד לבדוק את הערכים המוחזרים מהקריאות, מפני שאין להניח את הצלחתן של קריאות מקובץ.

נניח ש fscanfביקשה לקרוא ל n משתנים. אז הקריאה מוצלחת אם הפונקציה מחזירה .n להלן דוגמה לבדיקה:

```
int d;
```

```c
char c;
FILE* fp;
/*...open file code...*/

if( fscanf(fp, "%d %c", &d, &c) != 2)
  ... /* Handle error. */
```

size_t **fread** (void *buffer, size_t size, size_t count, FILE*stream);	קוראת עד countבלוקים בגודל sizeביית מהקובץ למקום בזיכרון-buffer. מחזירה מספר בלוקים **שלמים** שנקראו בהצלחה . המספר יכול להיות קטן מ- count. סוג unsined int ≡size_t
int **fscanf** (FILE *stream, const char *format...);	כמו scanf
int **fseek**(FILE *stream, long offset, int origin);	מזיזה "מצביע בתוך קובץ" ל- offsetביית יחסית למקום originבקובץ: SEEK_CUR- מקום נוכחי בתוך קובץ, SEEK_SET, SEEK_ENDסוף או תחילת קובץ. סיום הפעולה בהצלחה מוחזר 0 , אם ארעה שגיאה 0=!
long **ftell** (FILE *stream);	מחזירה מיקום נוכחי של מצביע בתוך הקובץ - מספר בייתים יחסי לתחילת הקובץ או 1L- בשגיאה
size_t **fwrite** (const void *buffer, size_t size, size_t count, FILE *stream);	כותבת עד count חלקים בגודל sizeביית ממקום buffer בזיכרון לזרם. מחזירה מספר חלקים **שלמים** שנכתבו. אם המספר שהוחזר קטן מ- count ארעה שגיאה. סוג unsined int ≡ size_t
void **rewind** (FILE *stream);	מזיזה "מצביע בתוך קובץ" לתחילת הקובץ

דוגמא מסכמת משיעורי בית reverse:

```c
#include <stdio.h>

#include <stdlib.h>

int sizeOfFile(FILE* f1){

   fseek(f1,0,SEEK_END);

   //rewind(f1);

   int size = ftell(f1);

   return size;

}

void createFile(FILE* f1){
```

```
   rewind(f1);
   for(int i=0;i<50;i++)
      fwrite(&i, sizeof(i), 1, f1);
}
void readIntFile(FILE* f1){
   int x;
   int n=sizeOfFile(f1)/sizeof(int);
   rewind(f1);

 //  while(!feof(f1)){
     for(int i=0;i<n;i++){
      fread(&x,sizeof(x),1,f1);
      printf("%d ",x);
      }
}
void rev(FILE* f1){
   int n1,n2;
   int n= sizeOfFile(f1)/sizeof(int);
//   printf("%d ",n);
   for(int i=0;i<n/2;i++){
      fseek(f1,i*(sizeof(int)),SEEK_SET);
      fread(&n1,sizeof(int),1,f1);
      //fseek(f1,(n-i)*(sizeof(int)),SEEK_SET);
      fseek(f1,-(i+1)*(sizeof(int)),SEEK_END);
      fread(&n2,sizeof(int),1,f1);
      //fseek(f1,(n-i)*(sizeof(int)),SEEK_SET);
      fseek(f1,-(i+1)*(sizeof(int)),SEEK_END);
      fwrite(&n1, sizeof(i), 1, f1);
      fseek(f1,i*(sizeof(int)),SEEK_SET);
```

```
        fwrite(&n2, sizeof(i), 1, f1);
        //printf("%d %d \n",n1,n2);
    }
}
int main()
{
    FILE* f2 = fopen("ex.bin","wb");
    if(!f2){
        fprintf(stderr,"FATAL ERROR CANNOT OPEN FILE");
        return -1;
    }
    createFile(f2);
    fclose(f2);
    FILE* f1 = fopen("ex.bin","r+b");
    if(!f1){
        fprintf(stderr,"FATAL ERROR CANNOT OPEN FILE");
    }
    //printf("%d",sizeOfFile(f1));
    readIntFile(f1);
    printf("\n");
    rev(f1);
    readIntFile(f1);
    fclose(f1);
    return 0;
}
```

קריאת קובץ שורה אחר שורה

```c
#include <stdio.h>

int main()
{
   FILE *fp;
   char str[128];
```

```
   /* opening file for reading */
   fp = fopen("file.txt" , "r");
   if(fp == NULL) {
      perror("Error opening file");
      return(-1);
   }
   if( fgets (str, 128, fp)!=NULL ) {
      /* writing content to stdout */
      puts(str);
   }
   fclose(fp);

   return(0);
}
```

קדם מהדר

בהתניה. קבצי כותרת לעתים קרובות נראים מהצורה היא סימון [קבצי כותרת](#) C שיטה מקובלת בתכנות:

```
#ifndef <const>
#define <const>
...
#endif
```

`#ifndef-#define` הוא קבוע כלשהו. כלומר, כל תוכן הקובץ נמצא בין רצף const כאשר `#endif`-ל.

לדוגמה - תוכנו עשוי מאד להראות כך ,file_1.hאם נתבונן בקובץ ששמו:

```
#ifndef FILE_1_H
#define FILE_1_H
...
#endif /* ifndef FILE_1_H */
```

אוסף הפקודות הזה מביא לכך שתוכן הקובץ מוכלל אך ורק פעם אחת:

1. בפעם הראשונה בה הקובץ נקרא, מוגדר הקבוע (const כאן, לדוגמה.(FILE_1_H ,
2. בפעמים הבאות בהן הקובץ נקרא, הקבוע כבר מוגדר. היות שתוכן הקובץ עטוף בזוג `#ifndef-`
   - `#endif` המהדר יתעלם ממנו.

הסיבה לשימוש בסימון זה היא שפעמים רבות ייקראו קבצי כותרת פעמים רבות (לפעמים מקבצי כותרת אחרים), מה שעלול לגרום לבעיות בהידור הקובץ (אם, למשל, יוגדר struct זהה יותר מפעם אחת - התוכנית תיכשל בשלב ההידור.)

ש המוסיפים הערה ל `#endif`-המציינת את תוכן השורה שהתחילה את הקטע, לדוגמה כך:

```
#ifdef DEBUG
...
#endif /* ifdef DEBUG */
```

או כך:

```
#ifndef FILE_1_H
#define FILE_1_H
...
#endif /* ifndef FILE_1_H */
```

- #if

  #if expression // evaluates to true (non zero) or false (zero)

  …

  #endif

- #if defined (identifier) // true if identifier was defined by #define

  …

  #endif

- #ifdef

  #ifdef identifier // true if identifier was defined by #define

  …

  #endif

- #ifndef

  #ifndef identifier // true if identifier was not defined by #define

  …

  #endif

- && , || , ! are allowed.

הדבר עשוי לעזור למתכנתת להבין היכן החל קטע הקוד המותנה.

```
#include <stdio.h>

#define DENOMINATOR 4.4

#define DEBUG

void main()

{

float f1 = 1.1, f2 = 2.2, f3 = 3.3, f4;

#if DENOMINATOR != 0

f4 = f3 / DENOMINATOR;

printf("f4 = %f \n", f4);

#else

printf("f4 can not be computed \n");

#endif

f4 = ++f1 - f2 / f3 + --f2;

#ifdef DEBUG

printf("f4 = %f \n", f4);

#endif

}

/**** Output ****

f4 = 0.750000

f4 = 2.936363
```

*****************/

מאקרו לשרשור מחרוזת ושימוש במילות מפתח

```
#include <stdio.h>

#define ADD(a, b) a##b

void main()

{

int n = ADD(1,1);

char str[] = ADD("Shenkar", " college");

printf("%d\n %s\n", n, str);

printf("File: %s\n", __FILE__);

printf("Line: %d\n", __LINE__);

}
```

```
/**** Output ****

11

Shenkar college

File: C:\WINDOWS\Desktop\BOOK\8.2__pre_defined_macro.c

Line: 14

*****************/
```

מאקרו לביצוע פקודה מספר מסויים של פעמים

```
#define do2(x,y) {for (int i=0;i<y;i++) x;}

int main(){

do2(printf("Hello\n"),5);

    return 0;
```

```
}
```

פלט:

```
Hello
Hello
Hello
Hello
Hello
```

סכנות

שימוש ביכולות הקדם-מעבד מספק כמה יכולות רבות עוצמה, אך מצד שני - טומן בחובו לא מעט סכנות. ראו לדוגמה את המאקרו התמים הבא:

```
#define MY_FOO(x) x * x
```

כעת חשבו על קטע הקוד הבא, אשר עושה שימוש במאקרו:

```
int foo(int x, int y) {
   return MY_FOO(x + y);
}
```

אחרי מעבר הקדם מעבד, תהייה התוצאה:

```
int foo(int x, int y) {
   return x + y * x + y;
}
```

תוצאה בלתי מתוכננת!

על בעייה זו אפשר להתגבר בעזרת שימוש בסוגריים, כלומר:

```
#define MY_FOO(x) ((x) * (x))
```

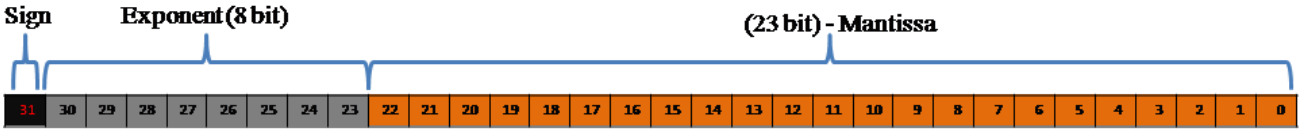
האמנם הצלחנו לכתוב קוד בטוח? מה יקרה במקרה הבא?

```
int x = 3;
MY_FOO(++x)
```

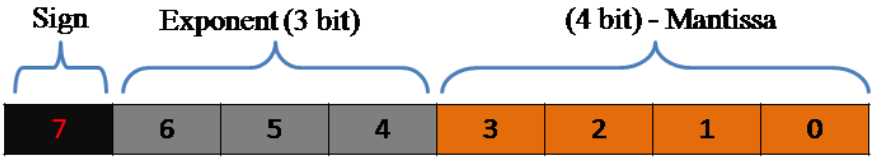
האופרטור ++ יתבצע פעמיים (במקום, כפי שרצינו, פעם אחת), וגם כאן יביא לתוצאה שאינה מתוכננת (ולפעמים קשה מאוד למציאה!).

המסקנה משתי דוגמאות אלו היא שחובה לנהוג זהירות רבה לפני שימוש במאקרו. בפרט, הקפדה על שימוש בסוגריים והמנעות ממאקרו שכולל בתוכו את אותו המשתנה יותר מפעם אחת יפחיתו את מספר הטעויות האפשריות.

IEEE754



1. ניתן לראות שביטים 0 עד 22 מייצגים את המנטיסה, וביטים 23 עד 30 את המערך. ביט 31 הוא הסימן (חיובי שלילי).
2. <u>איך זה בדיוק עובד ואיפה בדיוק נכנסת ה"נקודה הצפה"?</u>
3. כדי להסביר זאת טוב יותר נסתכל בנוסחה המציגה את הסטנדרט IEEE 754 ב 32 ביט–
4. $$v = (-1)^S \quad x \quad M \quad x \quad 2^{E-127}$$
5. הסבר –
6. S = סימן המיוצג על ידי ביט אחד (0 עבור מספר חיובי, 1 עבור מספר שלילי)
   M = מנטיסה, ערכו של המספר בשבר בינארי.
   E = מערך, קובע את גודלו של המספר.
7. שימו לב שבנוסחה המערך הוא E – 127, E(המערך) הוא מספר בין 0 ל- 255 שהם 8 הביטים של המערך, חיסור של Offset מהמערך מאפשר לנוע מ עד, וכך לייצג ערכים מאוד גדולים ומאוד קטנים וגם לייצג מספרים חיוביים ושליליים.
   המערך מאוכסן עם *offset* של 127, לכן המערך יכול לנוע מ- 127- (127- = 127 – 0) עד 128 (128 = 127 – 255).
   ניתן לומר שייצוג מספר בנקודה צפה הוא מוטה עם *offset* מסיום. לרוב משתמשים בערכים של –
8. $$2^{E-1} = \textit{offset} - 1$$
9. $$2^{E-1} = \textit{offset}$$
10. המנטיסה, M, מורכבת מ-23 ביט בצורת שבר בינארי. לדוגמה, את השבר העשרוני – 3.141 ניתן לפרק ל- 3 + 1/10 + 4/100 + 1/1000. והשבר הבינארי 1.0101 ניתן לפרק ל- 1 + 0/2 + 1/4 + 0/8 + 1/16.
    מספרים בנקודה צפה מנורמלים כמו מספרים בכתיב מדעי, כלומר יש רק מספר אחד שהוא לא 0 מצדה השמאלי של הנקודה העשרונית. מאחר והמספר היחיד שקיים בבסיס בינארי שהוא לא 0 זה 1, הספרה המובילה במנטיסה תמיד תהיה 1, לכן אין צורך לאכסן אותה. כך שבהסרת הספרה הזו מהאכסון אנו מאפשרים לשימוש ביט נוסף.
    לכן $M = 1.\ m_{22}m_{21}m_{20}m_{19}.....m_0$
11. באופן זה ערכו של מספר הוא למעשה $N = (-1)^S \times 1.M \times 2^{E-offset}$
12. כדי להבין יותר לעומק את השיטה נציג כמה דוגמאות ב -8 ביט בסטנדרט של IEEE 754 -



13.
14. חישוב ה- offset :
15. $$1 - \text{Offset} = 2^{E-1} = 2^{\text{מספר הביטים} - 1} - 1$$
16. $$\text{Offset} = 2^{3-1} - 1 = 3$$

17. לדוגמה המספר 15 בנקודה צפה יוצג כך –

עשרוני	ייצוג בינארי								הנוסחה
15	S	E			M				$(-1)^0 * 2^{110-011} * 1.1110$
	0	1	1	0	1	1	1	0	

18. $2^{110-011} = 2^{6-3} = 2^3 = 8$

19. $1.1110 = 1 + 1/2 + 1/4 + 1/8 = 1 + 0.5 + 0.25 + 0.125 = 1.875$

20. $8 * 1.875 = 15$

21. **<u>איך הגענו למספר הבינארי "בנקודה צפה"?</u>**

22. אם נתון לנו מספר עשרוני כדוגמת 5.5, קודם נמיר אותו לשבר בינארי – 101.1
כעת ננרמל את השבר על ידי כך שנזיז את הנקודה העשרונית 2 צעדים קדימה – 1.011
מכיוון שהזזנו את הנקודה העשרונית ב- 2 צעדים קדימה אז המעריך יהיה 2, לא לשכוח שיש Offset של 3(011 בבינארי) לכן ה- E יהיה 5, כי 3 = 5 - 2, כלומר E = **101** .

23. $2^{101-011} = 2^{5-3} = 2^2 = 4$

24. כעת נחשב את המנטיסה –

25. $1.011 = 1 + 0/2 + 1/4 + 1/8 = 1 + 0 + 0.25 + 0.125 = 1.375$

26. לא לשכוח, את הספרה 1 מצידה השמאלי של הנקודה העשרונית במנטיסה לא מאכסנים, לכן המנטיסה שנאכסן תהיה – **0110** ולא 1011 כמו בחישוב מעל(1.011).

27. ועכשיו נכפיל –

28. $4 * 1.375 = 5.5$

29. ה – sign יהיה **0** כדי לציין מספר חיובי.

30. **<u>מכאן יוצא</u>**

31.

S	E			M			
0	1	0	1	0	1	1	0

32. **<u>דוגמה נוספת</u>**

33. 12.5 בעשרוני מה הייצוג בנקודה צפה ?

34. נמיר את המספר העשרוני לבינארי - 12.5 = 1100.1

35. ננרמל את המספר הבינארי – $1.1001 * 2^3$ = 1100.1

36. M = 1001

37. E = 110(שזה 6 בעשרוני, צריך ש – E יהיה שווה ל-6 כדי לקבל 3 במעריך, שזה מספר הצעדים שהזזנו את הנקודה העשרונית, כי ישנו Offset של 3 ו 3 = 3 - 6) .

38. S = 0 (מייצג מספר חיובי)

39. תוצאה - 01101001

12.125

12=1100

0.125*2=0.25(0)

0.25*2=0.5(0)

0.5*2=0(1)

100

1100.001=1.100001*2^3

127+3=(10000010)2

(01000001010000100000000000000000)IEEE754

Ellipsis

איך משתמשים ב Ellipsis:

1. מגדירים את מצביע

```
va_list list_ptr;
```

2. מקשרים את המצביע לרשימה שמועברת דרך כותרת הפונקציה.

```
va_start(list_ptr,last-named);
```

יש לשים לב ש last-named מתייחס לשם של המשתנה שמועבר לפני ה elpsis.

3. מבנה המעבר על הרשימה

```
while(val = va_arg(list_ptr,datatype \* for example int*\)!=NULL){..}
```

דרך זו מתייחסת כאשר הפרמטר האחרון שמועבר הוא מסוג null, ניתן לעבור ב for על הרשימה בצורה דומה כאשר מעבירים כפרמטר לפני הelpsis את כמות הפרמטרים ב elpsis .

4. אסור לשכוח לשחרר את הרשימה בסוף הפונקציה

```
va_end(list_ptr)
```

Set/Reset bit positions using elpsis

```
/* 1. usage format: x = bits(x,<sr>,<list>); */

/* 2. <sr>='s' sets to 1 <sr>='r' resets to 0 */

/* 3. bit positions listed in order from bit 0 */

/* 4. a trailing 0 is needed to delimit */

/*************************************************/

int bits(unsigned int x, char sr, ...)

{

int a, b=0;

va_list listptr; /* declare list pointer */

va_start(listptr, sr); /* start after var sr */

/* step through list: cover case of initial 0 */

b+=1<<va_arg(listptr,int);

while ((a=va_arg(listptr,int)) != 0)
```

```
b+=1<<a; /* use shift to get 2^a */

va_end(listptr); /* clean up for return */

if (sr == 's') return(x|b);

if (sr == 'r') return(x&~b);

else return(x);

}
```

As an example of typical usage, the function call

```
x = bits(x,'s',0,3,17,0);
```

to set bit positions 0, 3, and 17 of variable x to 1, accomplishes

this by calculating b=20

+23

+217 and then ORing x with b [x|b].

Command Line Arguments

כאשר רוצים לאפשר לתוכנית שלנו לקבל פרמטרים דרך ה command line צריך לשנות את הכותרת של ה main לתצורה הבאה:

```
int main(int argc, char *argv[]){..}
```

כאשר argc יחזיק את מספר הפרמטרים שהועברו(פרמטרים מופרדים ע"י whitespace) agrv מערך של הפרמטרים, תמיד argv[0] יהיה השם של התוכנית שלנו, argv[argc] יהיה NULL

דוגמא תוכנית שמקבל כפרמטר קובץ מקור של תוכנית C ומפריד אותו לקובץ מקור חדש ללא הערות וקובץ נוסף המאפשר שחזור של הערות

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void deleteComments(FILE* c,FILE* x,FILE* y){
	int n;
	int count=0;
	while((n=getc(c))!=EOF){
		int flag=0;
		if(n=='/')
		{
			int n2 = getc(c);
			if(n2==EOF) return;
			if(n2=='/'){
				flag=1;
				fwrite(&count,sizeof(int),1,y);
				count=0;
				putc('/',y);
				putc('/',y);
```

```
                        while((n=getc(c))!=EOF && n!='\n'){
                                putc(n,y);
                        }
                }
                else if(n2=='*'){
                        flag=1;
                        fwrite(&count,sizeof(int),1,y);
                        count=0;
                        putc('/',y);
                        putc('*',y);
                        while((n2=getc(c))!=EOF){
                                if(n2 == '*'){
                                        n=getc(c);
                                        if(n=='/'){
                                                putc('*',y);
                                                putc('/',y);
                                                break;
                                        }
                                        else{
                                        putc(n2,y);
                                        putc(n,y);
                                        }
                                }
                                putc(n2,y);
                        }
                }
                else{
                        putc(n,x);
                        count++;
                        putc(n2,x);
                        count++;
                        continue;
                }
        }
        if(!flag){
        count++;
        putc(n,x);
        }
    }
}
int main(int argc,char** argv){
    if(argc!=2){
            fprintf(stderr,"ERROR");
            return 1;
    }
    char * file = argv[1];
    int len = strlen(file);
    if(file[len-1]!='c' && file[len-2]!='.'){
            fprintf(stderr,"ERROR");
            return 2;
    }
    FILE* c = fopen(file,"r+");
    if(!c){
            fprintf(stderr,"ERROR");
            return 3;
    }
    file[len-2]=0;
    char* file2=(char*)malloc(len*sizeof(char));
```

```
        char* file3=(char*)malloc(len*sizeof(char));
        strcpy(file2,file);
        strcat(file2,".x");
        strcpy(file3,file);
        strcat(file3,".y");
        FILE* x = fopen(file2,"w+");
        FILE* y = fopen(file3,"w+");
        deleteComments(c,x,y);
        fclose(c);
        fclose(x);
        fclose(y);
}
```

Complicated declaration explained with examples

Most of the times declarations are simple to read, but it is hard to read some declarations which involve pointer to functions. For example, consider the following declaration from “signal.h”.

```
void (*bsd_signal(int, void (*)(int)))(int);
```

Let us see the steps to read complicated declarations.

1) Convert C declaration to postfix format and read from left to right.
2) To convert experssion to postfix, start from innermost parenthesis, If innermost parenthesis is not present then start from declarations name and go right first. When first ending parenthesis encounters then go left. Once whole parenthesis is parsed then come out from parenthesis.
3) Continue until complete declaration has been parsed.

Let us start with simple example. Below examples are from “K & R” book.

```
1)  int (*fp) ();
```

Let us convert above expression to postfix format. For the above example, there is no innermost parenthesis, that’s why, we will print declaration name i.e. “fp”. Next step is, go to right side of expression, but there is nothing on right side of “fp” to parse, that’s why go to left side. On left side we found “*”, now print “*” and come out of parenthesis. We will get postfix expression as below.

```
  fp  *  ()  int
```

Now read postfix expression from left to right. e.g. fp is pointer to function returning int

Let us see some more examples.

```
2) int (*daytab)[13]
```

Postfix : daytab *[13] int
Meaning : daytab is pointer to array of 13 integers.

```
3) void (*f[10]) (int, int)
```

Postfix : f[10] * (int, int) void
Meaning : f is an array of 10 of pointer to function(which takes 2 arguments of type int) returning void

```
4) char (*(*x())[]) ()
```

Postfix : x () * [] * () char
Meaning : x is a function returning pointer to array of pointers to function returnging char

```
5) char (*(*x[3])())[5]
```

Postfix : x[3] * () * [5] char
Meaning : x is an array of 3 pointers to function returning pointer to array of 5 char's

```
6) int *(*(*arr[5])()) ()
```

Postfix : arr[5] * () * () * int
Meaning : arr is an array of 5 pointers to functions returning pointer to function returning pointer to integer

```
7) void (*bsd_signal(int sig, void (*func)(int)))(int);
```

Postfix : bsd_signal(int sig, void(*func)(int)) * (int) void
Meaning : bsd_signal is a function that takes integer & a pointer to a function(that takes integer as argument and returns void) and returns pointer to a function(that take integer as argument and returns void)

more examples:

```
char **argv
    argv:  pointer to pointer to char
int (*daytab)[13]
    daytab:  pointer to array[13] of int
int *daytab[13]
    daytab:  array[13] of pointer to int
void *comp()
    comp:  function returning pointer to void
void (*comp)()
    comp:  pointer to function returning void
char (*(*x())[])()
    x: function returning pointer to array[] of
    pointer to function returning char
char (*(*x[3])())[5]
    x: array[3] of pointer to function returning
    pointer to array[5] of char
```

description like "`x` is a function returning a pointer to an array of pointers to functions returning `char`," which we will express as

```
x () * [] * () char
```

to

```
char (*(*x())[])()
```

```
#include <stdio.h>

int mat[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

int   (*g())[4]{
 return mat;
}
int ( * ( * f() ) ()  ) [4] {
        return g;
}
void main(){
  int i,j;
  for (i=0; i<3; i++) {
          for (j=0; j<4; j++)
            printf("%d ",f()()[i][j]);
          printf("\n");
  }
  getchar();
}
```

טיפול בחריגות

ב C ישנו משתנה גלובלי errno ששומר קוד שגיאה שחוזר מפונקציה, כדי להדפיס את הודעת השגיאה המתאימה ניתן להשתמש בשתי הפקודות הבאות- strerror perror כמו בדוגמא למטה

```c
#include <stdio.h>
#include <errno.h>
#include <string.h>

extern int errno ;

int main ()
{
   FILE * pf;
   int errnum;
   pf = fopen ("unexist.txt", "rb");
   if (pf == NULL)
   {
      errnum = errno;
      fprintf(stderr, "Value of errno: %d\n", errno);
      perror("Error printed by perror");
      fprintf(stderr, "Error opening file: %s\n", strerror( errnum ));
   }
   else
   {
      fclose (pf);
   }
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of errno: 2
Error printed by perror: No such file or directory
Error opening file: No such file or directory
```

או שניתן להשתמש במערך הגלובלי sys_errlist.

Extren and Static

ב C ראינו את שאפשר לפרק תוכנית למודולים , מה קורה שרוצים להשתמש בפונקציות שהוגדרו במודל אחר?
כל מה שאנו צריכים לעשות זה להצהיר עליהם, אך כאשר אנו רוצים להשתמש במשתנים גלובליים שהוגדרו במודלים אחרים כעת אנו צריכים להרחיב את הנגישות שלהם לכלל התוכנית ע"י הכרזה שהוא משתנה extern.

המשמעות של static היא ההפך מ extern ומאפשרת להגביל שימוש בפונקציות ומשתנים למודול שבו הוגדרו בלבד.

דוגמא:

file3.h

```
extern int global_variable;  /* Declaration of the variable */
```

file1.c

```
#include "file3.h"  /* Declaration made available here */
#include "prog1.h"  /* Function declarations */

/* Variable defined here */
int global_variable = 37;    /* Definition checked against declaration */

int increment(void) { return global_variable++; }
```

file2.c

```
#include "file3.h"
#include "prog1.h"
#include <stdio.h>

void use_it(void)
{
    printf("Global variable: %d\n", global_variable++);
}
```

That's the best way to use them.

אפשר לשים לב שהצהרה על משתנה כ extern יכול לעשות מספר מקומות ומספר פעמים אבל הגדרתו יכולה לקרות פעם אחת. הגדרה של משתנה כמה פעמים תגרור שגיאת קומפילציה .

ASCII TABLE

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com