

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

Hydra.c

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

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

```
int main() {  
    int heads;  
    int eyes;
```

```
    heads = 3;          /* ανάθεση τιμής */  
    eyes = heads * 2;    /* υπολογισμός τιμής */
```

```
    printf("It has %d heads and %d eyes! \n", heads, eyes);
```

```
    return 0;
```

```
}
```

Έξοδος:

> It has 3 heads and 6 eyes!

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

Hydra.c

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

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

```
int main() {  
    int heads;  
    int eyes;
```

Δήλωσε (declare) κάθε μεταβλητή.
Γράψε μία πρόταση για να τεθεί το
όνομά της και το είδος της.

```
heads = 3;
```

```
eyes = heads * 2;
```

```
/* ανάθεση τιμής */
```

```
/* υπολογισμός τιμής */
```

```
printf("It has %d heads and %d eyes! \n", heads, eyes);
```

```
return 0;}
```

Έξοδος:

> It has 3 heads and 6 eyes!

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

Hydra.c

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

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

```
int main() {  
    int heads;  
    int eyes;
```

```
    heads = 3;  
    eyes = heads * 2;
```

```
    /* ανάθεση τιμής */  
    /* υπολογισμός τιμής */
```

```
    printf("It has %d heads and %d eyes! \n", heads, eyes);
```

```
    return 0;
```

```
}
```

Μετά τον ορισμό, θέσε
τιμές στις μεταβλητές.

Έξοδος:

> It has 3 heads and 6 eyes!

Ονόματα μεταβλητών

- Έγκυρα ονόματα μεταβλητών:

totalArea *max_amount* *counter1*
Counter1 *_temp_in_F*

- Μη έγκυρα ονόματα μεταβλητών:

\$product *total%* *3rd*

- Απαράδεκτα ονόματα μεταβλητών:

/ *x2* *maximum_number_of_students_in_my_class*

Τύποι μεταβλητών

Υπάρχουν 4 βασικοί τύποι μεταβλητών στη γλώσσα C:

<u>Τύπος</u>	<u>Λέξη κλειδί στη C:</u>
Integer	<i>int</i>
Floating point	<i>float</i>
Double	<i>double</i>
Character	<i>char</i>

Ο τύπος του χαρακτήρα (char)

- Παριστάνει απλούς χαρακτήρες του αλφάβητου της γλώσσας. Βρίσκεται ανάμεσα σε απλά εισαγωγικά (π.χ. **'C', '2', '*', ')**).
- Δήλωση: **char choice;**
- Δήλωση με αρχική τιμή: **char choice='A';**
- Εκτύπωση: με χρήση της συνάρτησης **printf()** και του προσδιοριστή (specifier) **%c**.

printf("The character is %c\n", choice);

- Εάν αντί του **%c** χρησιμοποιηθεί ο **%d**, η **printf()** θα εμφανίσει τον ASCII κωδικό του χαρακτήρα. Η πρόταση

printf("The ASCII Code of %c is %d\n", choice,choice);

θα τυπώσει

The ASCII Code of A is 65

Ο τύπος του χαρακτήρα

• Εισαγωγή: με χρήση της συνάρτησης `scanf()` και του προσδιοριστή (specifier) `%c`. Η πρόταση

`scanf("%c", &ch);`

διαβάζει από την κύρια είσοδο (πληκτρολόγιο) ένα χαρακτήρα και τον αποδίδει στη μεταβλητή `ch`. Θα πρέπει να προσεχθεί η χρήση του `&` πριν από τη μεταβλητή. Ονομάζεται **τελεστής διεύθυνσης** και προηγείται πάντοτε των μεταβλητών στην εντολή `scanf()`.

• Αποθήκευση και ανάκληση ASCII χαρακτήρα: Ο μεταγλωττιστής απαιτεί 1 byte μνήμης για την αποθήκευση της τιμής μίας μεταβλητής χαρακτήρα. Η αποθήκευση και ανάκληση παρουσιάζεται στην επόμενη διαφάνεια:

Αποθήκευση και ανάκληση ASCII χαρακτήρα



Μη εκτυπούμενοι χαρακτήρες

Οι σταθερές τύπου χαρακτήρα **‘νέα γραμμή (new-line)’** και **‘στηλοθέτης (tab)’** ανήκουν στην κατηγορία των μη εκτυπούμενων χαρακτήρων, τους οποίους η C αναπαριστά με τις **ακολουθίες διαφυγής (escape sequences)** **‘\n’** **‘\t’**, αντίστοιχα. Η παρακάτω πρόταση δίνεται ως παράδειγμα χρήσης χαρακτήρων διαφυγής:

```
printf( "Write, \"a \\ is a backslash. \\\"\\n\" );
```

Η πρόταση θα εμφανίσει στην κύρια έξοδο (οθόνη):

Write, "a \ is a backslash."

Διπλό εισαγωγικό

Διπλό εισαγωγικό

Νέα γραμμή

Μη εκτυπούμενοι χαρακτήρες και αντίστοιχες ακολουθίες διαφυγής

Χαρακτήρας	ακολουθία διαφυγής	Χαρακτήρας	ακολουθία διαφυγής
συναγερμός (κουδούνι)	\a	πλάγια	\\
οπισθοχώρηση	\b	λατινικό ερωτηματικό	\?
αλλαγή σελίδας	\f	μονό εισαγωγικό	\'
νέα γραμμή	\n	διπλό εισαγωγικό	\"
επαναφορά κεφαλής	\r	οκταδικός αριθμός	\ooo
οριζόντιος στηλοθέτης	\t	16-δικος αριθμός	\xhhh
κατακόρυφος στηλοθέτης	\v		

Παράδειγμα: Να καταστρωθεί πρόγραμμα που να επιτελεί τα παρακάτω:

- Ζήτησε από τον χρήστη ένα χαρακτήρα
- Πάρε από τον χρήστη τον χαρακτήρα
- Τύπωσε τον χαρακτήρα και τον ASCII κωδικό του
- Βρες τον επόμενο χαρακτήρα
- Τύπωσέ τον μαζί με τον κωδικό του

/******

Το πρόγραμμα διαβάζει ένα χαρακτήρα και τυπώνει τον χαρακτήρα και τον επόμενο του καθώς και τους ASCII κωδικούς τους.

*****/

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

```
int main()
```

```
{
```

```
char ch,next_ch;
```

```
printf( "Write a character:\t" );
```

```
scanf( "%c",&ch );
```

```
printf( "The ASCII code of char %c is %d\n", ch, ch );
```

```
next_ch=ch+1;    /* βρίσκει τον επόμενο χαρακτήρα */
```

```
printf( "The ASCII code of char %c is %d\n", next_ch, next_ch );
```

```
return 0
```

```
}
```

Για να τυπωθεί ο χαρακτήρας ένα στηλοθέτη δεξιότερα

Ο τύπος του ακεραίου

Δήλωση: δηλώνεται με τη λέξη κλειδί *int* και χρησιμοποιείται για να παραστήσει ακέραιους αριθμούς, αρνητικούς ή θετικούς. Η περιοχή τιμών εξαρτάται από την αρχιτεκτονική του μηχανήματος. Για έναν υπολογιστή με λέξη (word) 16 bits η περιοχή τιμών του τύπου *int* είναι από -32767 έως +32767.

Εάν πριν από τη λέξη *int* τοποθετηθεί ο προσδιοριστής *long*, τότε οι ακέραιοι *long int* εξασφαλίζουν αποθηκευτικό χώρο 32 bits. Αντίστοιχα, ο προσδιοριστής *unsigned* χρησιμοποιείται πριν από τη λέξη *int* για να χαρακτηρίσει τη μεταβλητή χωρίς πρόσημο, η οποία λαμβάνει τιμές από 0 έως 65535 για λέξη 16 bits.

Σε περιβάλλοντα 32 και 64 bits οι ακέραιοι αποθηκεύουν τιμές στο διάστημα από -2.147.483.648 έως +2.147.483.648.

Ένας ακέραιος *short int* είναι τουλάχιστον 16 bits και ο *int* είναι τουλάχιστον τόσο μεγάλος όσο ο *short int*.

Εκτύπωση: με χρήση της συνάρτησης *printf* και των προσδιοριστών *%d, %o, %x* για την εμφάνιση σε δεκαδική, οκταδική και δεκαεξαδική μορφή, αντίστοιχα. Οι προσδιοριστές *l* (long), *h* (short), και *u* (unsigned) τοποθετούνται πριν από τους *d, o, x*.

```
printf( "dec=%d, octal=%o, hex=%x", num,num,num );
```

Εισαγωγή: με χρήση της συνάρτησης *scanf* και του προσδιοριστή (specifier) *%d*. Η πρόταση

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

διαβάζει από την κύρια είσοδο (πληκτρολόγιο) σε δεκαδική μορφή και αποδίδει την τιμή στην ακέραια μεταβλητή *num*.

Ακέραια σταθερά: Όταν στον πηγαίο κώδικα γράφουμε έναν αριθμό χωρίς δεκαδικό ή εκθετικό μέρος, ο compiler τον χειρίζεται ως ακέραια σταθερά. Η σταθερά 245 αποθηκεύεται ως *int*, ενώ η σταθερά 100.000 αποθηκεύεται ως *long int*. Εάν ορίσουμε τη σταθερά 8965 ως 8965L, ο compiler δεσμεύει χώρο για *long int*.

Παρατήρηση: Υπάρχει η δυνατότητα να καθορισθεί ο αριθμός των ψηφίων που θα εκτυπωθούν, τοποθετώντας τον επιθυμητό αριθμό ανάμεσα στο **%** και το **d**. Εάν ο αριθμός είναι μικρότερος από τον απαιτούμενο αριθμό ψηφίων του ακέραιου, η επιλογή δε θα ληφθεί υπόψη. Στην αντίθετη περίπτωση, στις πλεονάζουσες θέσεις θα τοποθετηθούν κενά.

Ο αριθμός που τοποθετείται στον προσδιοριστή **%d** ονομάζεται **καθοριστικό ελάχιστου πλάτους πεδίου**. Με αυτόν τον τρόπο, σε διαδοχικές **printf()** θα υπάρξει ευθυγράμμιση των αποτελεσμάτων κατά στήλες. Οι προτάσεις

```
printf( "dec=%1d, octal=%4o, hex=%4x", num,num,num );
```

```
printf( "dec=%4d, octal=%4o, hex=%4x", num,num,num );
```

θα τυπώσουν αντίστοιχα:

```
dec=46, octal= 56, hex= 2e
```

```
dec= 46, octal= 56, hex= 2e
```



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

Το πρόγραμμα εξετάζει το μήκος του τύπου ακεραίου.

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

```
#include <stdio.h>    // για την printf
```

```
#include <limits.h>   // climits.h αλλού
```

```
int main() {
```

```
    int number_int=INT_MAX; // Μέγιστος ακέραιος, οριζόμενος στο limits.h
```

```
    short int number_short=SHRT_MAX; // Μέγιστος short int
```

```
    long int number_long=LONG_MAX; // Μέγιστος long integer
```

```
// ο τελεστής sizeof δίνει το μέγεθος ενός τύπου δεδομένου ή μίας μεταβλητής
```

```
    printf( "int is %d bytes\n",sizeof(int) );
```

```
    printf( "short is %d bytes\n",sizeof(short) );
```

```
    printf( "long is %d bytes\n",sizeof(long) );
```

```
    printf( "\nmax int:%d  min int:%d\n",number_int,INT_MIN );
```

```
    printf( "\nmax short:%d  min short:%d\n",SHRT_MAX,SHRT_MIN );
```

```
    printf( "\nmax long:%d  min long:%d\n",number_long,LONG_MIN );
```

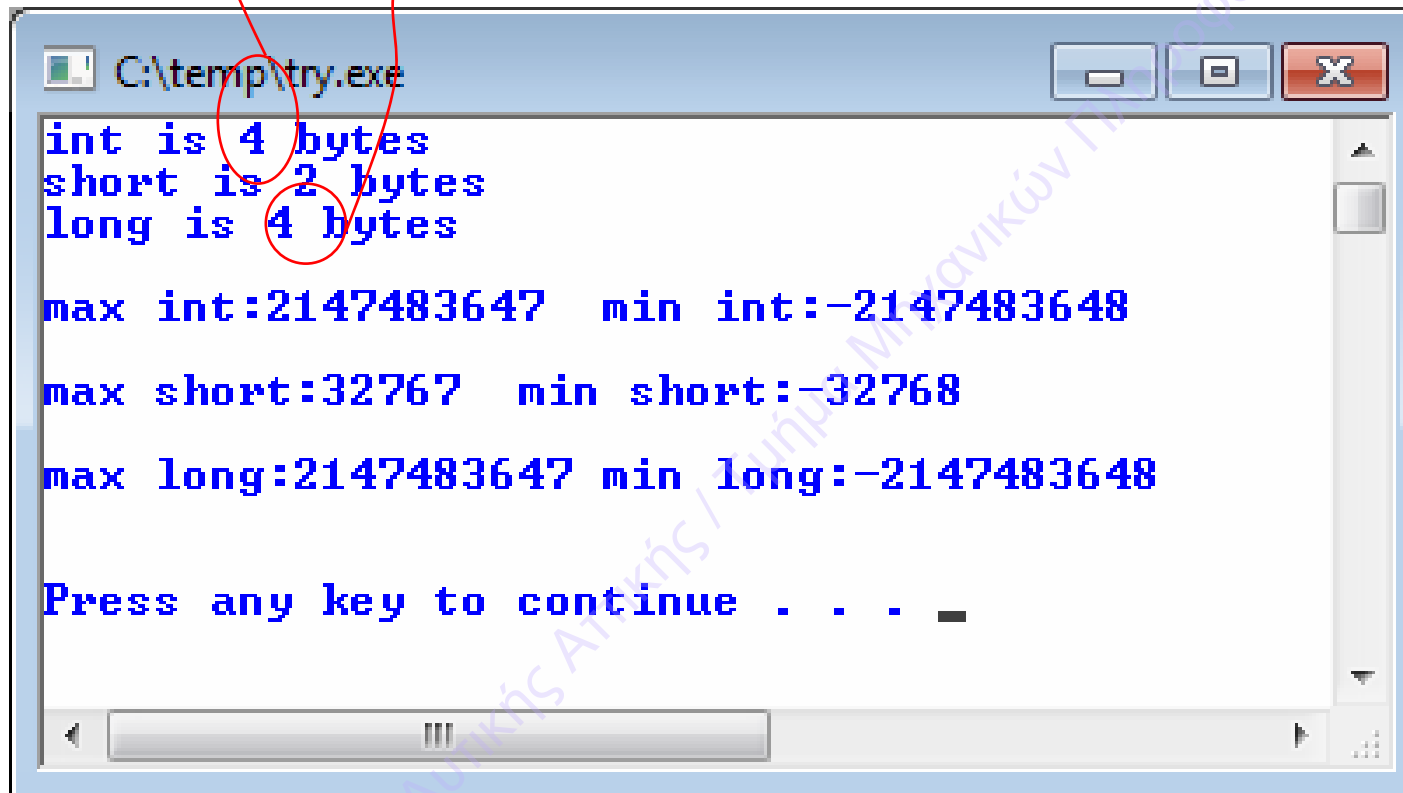
```
    return 0;
```

```
} // τέλος της main
```



Αποτελέσματα:

Ταυτίζονται τα bytes για int και long γιατί ο int καταλαμβάνει 4 bytes.



```
C:\temp\try.exe  
int is 4 bytes  
short is 2 bytes  
long is 4 bytes  
  
max int:2147483647 min int:-2147483648  
max short:32767 min short:-32768  
max long:2147483647 min long:-2147483648  
  
Press any key to continue . . .
```

Τύποι πραγματικών αριθμών

Δήλωση: πραγματικοί είναι οι αριθμοί που διαθέτουν κλασματικό μέρος και εκφράζονται συνήθως στις ακόλουθες μορφές:

<i>Fixed-point number</i>	<i>Scientific notation</i>	<i>Exponential notation</i>
123.456	1.23456×10^2	$1.23456e+02$
0.00002	2.0×10^{-5}	$2.0e-5$
50000.0	5.0×10^4	$5.0e+04$

Η γλώσσα C διαθέτει δύο τύπους για αναπαράσταση πραγματικών αριθμών. Τον τύπο **float** για αριθμούς κινητής υποδιαστολής απλής ακρίβειας και τον τύπο **double** για αριθμούς κινητής υποδιαστολής διπλής ακρίβειας. Η πρόταση **float plank=6.63e-34;** δηλώνει τη μεταβλητή *plank* ως απλής ακρίβειας και της δίνει την τιμή $6.63e-34$.

Η χρήση του προσδιοριστή **long** πριν από τον τύπο **double** χρησιμοποιείται για δήλωση μεταβλητής κινητής υποδιαστολής εκτεταμένης ακρίβειας, π.χ. **long double plank;**

Εκτύπωση: Με χρήση της `printf()` και των προσδιοριστών `%f` για εμφάνιση σε fixed point μορφή, `%e` για εμφάνιση σε εκθετική μορφή, και `%g` για να ανατεθεί στο σύστημα να επιλέξει μεταξύ των δύο προηγούμενων, με προτεραιότητα στη μορφή με το μικρότερο μέγεθος.

Αποθήκευση: Ως συνηθισμένα μεγέθη αναφέρονται για τους μεν `float` τα 32 bits, 8 για εκθέτη και πρόσημο και 24 για τη βάση, για τους δε `double` τα 64 bits, με τα επιπλέον 32 να χρησιμοποιούνται για αύξηση της ακρίβειας της βάσης (3 χρησιμοποιούνται στον εκθέτη).

Πραγματικές σταθερές: Πραγματικοί αριθμοί όπως οι

0.12 45.68 9e-5 24e09 0.0034e-08

όταν εμφανίζονται στον πηγαίο κώδικα αποτελούν τις πραγματικές σταθερές. Θεωρούνται από τον μεταγλωττιστή ως `double` και δεσμεύουν τον αντίστοιχο χώρο.

Entire Data types in c:

Data type	Size(bytes)	Range	Format string
Char	1	128 to 127	%c
Unsigned char	1	0 to 255	%c
Short or int	2	-32,768 to 32,767	%i or %d
Unsigned int	2	0 to 65535	%u
Long	4	-2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Float	4	3.4 e-38 to 3.4 e+38	%f or %g
Double	8	1.7 e-308 to 1.7 e+308	%lf
Long Double	10	3.4 e-4932 to 1.1 e+4932	%Lf

long long int (8 bytes, **%lld**) -9223372036854775808 έως 9223372036854775808

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

Το πρόγραμμα εξετάζει το μήκος του τύπου κινητής υποδιαστολής.

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

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

```
#include <float.h> // για το μέγιστο και τον ελάχιστο float, double
```

```
int main()
```

```
{
```

```
float num_float=FLT_MAX; // Μέγιστος float
```

```
double num_double=DBL_MAX; // Μέγιστος double
```

```
printf("float is %d bytes\n",sizeof(float));
```

```
printf("double is %d bytes\n",sizeof(double));
```

```
printf("\nmax float:%e min float:%e\n",num_float,FLT_MIN);
```

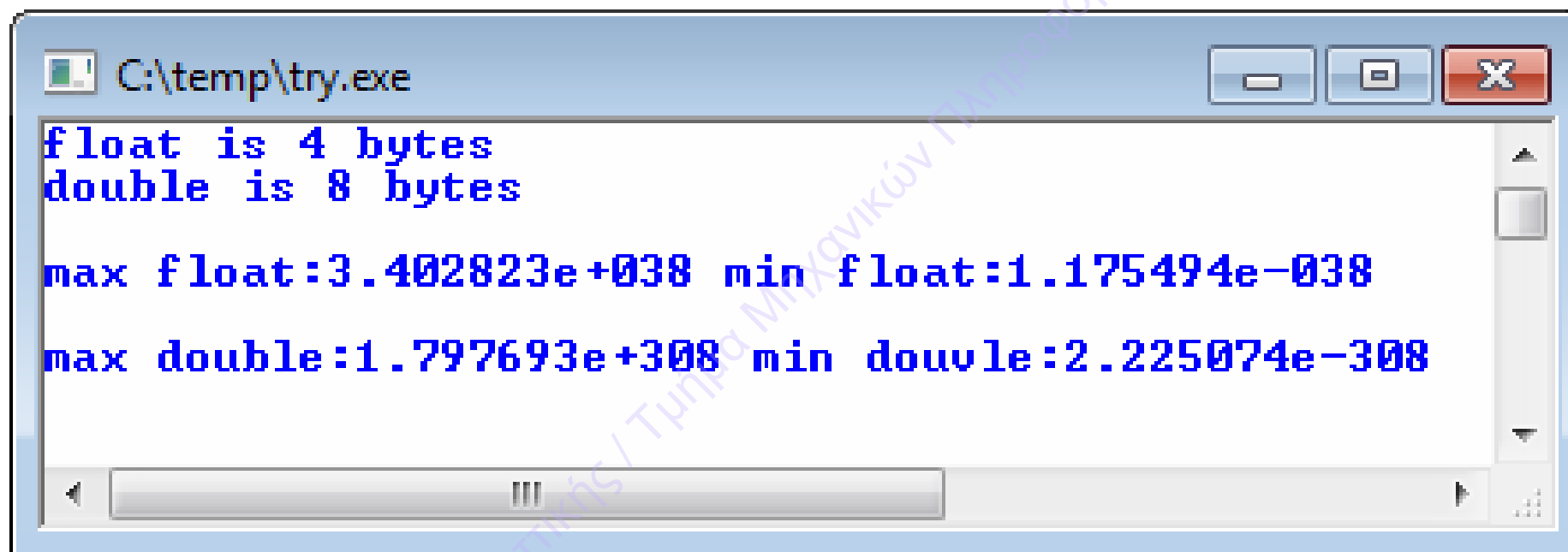
```
printf("\nmax double:%e,min double:%e\n", num_double,DBL_MIN);
```

```
return 0;
```

```
}
```



Αποτελέσματα:



```
C:\temp\try.exe  
float is 4 bytes  
double is 8 bytes  
  
max float:3.402823e+038 min float:1.175494e-038  
max double:1.797693e+308 min double:2.225074e-308
```

Ο προσδιοριστής *const*

Στη γλώσσα C μπορούν να οριστούν και να αρχικοποιηθούν μεταβλητές, για τις οποίες δεν επιτρέπεται η μεταβολή της τιμής τους. Τέτοιες μεταβλητές δηλώνονται, όπως και οι υπόλοιπες, με την προσθήκη του προσδιοριστή **const** στα αριστερά του τύπου τους:

```
const double numDouble=46.358767;  
const int numInt=46;  
const int numChar='G';
```

Στις παραπάνω προτάσεις δηλώνονται και αρχικοποιούνται οι μεταβλητές τριών διαφορετικών τύπων **numDouble**, **numInt** και **numChar**. Οι μεταβλητές αυτές δεν μπορούν να λάβουν άλλη τιμή καθ'όλη τη διάρκεια εκτέλεσης του προγράμματος.

Σημαίες

Ένα επιπλέον εργαλείο μορφοποίησης των εκτυπούμενων δεδομένων αποτελούν οι **σημαίες** (flags), που παρατίθενται στον Πίνακα. Αριστερά από κάθε σημαία απαιτείται η προσθήκη του συμβόλου **%**.

Σημαία	Λειτουργία
-	Αριστερή στοίχιση της εξόδου στο πεδίο πλάτους (η προκαθορισμένη στοίχιση είναι στα δεξιά)
+	Προσθήκη του προσήμου στις θετικές τιμές
#o	Προσθήκη του 0 μπροστά από οκταδικούς
#x	Προσθήκη του 0x μπροστά από δεκαεξαδικούς αριθμούς
#X	Προσθήκη του 0X μπροστά από δεκαεξαδικούς αριθμούς
0	Προσθήκη των απαιτούμενων μηδενικών μπροστά από την τιμή, ώστε να καλυφθεί το πεδίο πλάτους.
κενός χαρακτήρας	Προσθήκη του κενού χαρακτήρα μπροστά από τις μηδενικές τιμές



Παράδειγμα με σημαίες

```
#include <stdio.h>

int main()
{
    int x=323;

    printf( "1:\t%-5d\n",x );
    printf( "2:\t%+5d\n",x );
    printf( "3:\t% d\n",x );
    printf( "4:\t%#o\n",x );
    printf( "5:\t%#x\n",x );
    printf( "6:\t%#X\n",x );
    printf( "7:\t%05d\n",x );

    return 0;
}
```

1:	323
2:	+323
3:	323
4:	0503
5:	0x143
6:	0X143
7:	00323

Παράδειγμα με σημαίες

Κάθε `printf()` αρχικά εμφανίζει στην οθόνη τους ακέραιους αριθμούς 1,2,...,7, ακολουθούμενους από τον χαρακτήρα ' : '. Στη συνέχεια, εμφανίζεται το περιεχόμενο της ακεραίας μεταβλητής `x`, όπως αυτό μορφοποιείται σύμφωνα με την εκάστοτε σημαία. Συγκεκριμένα:

- Η πρώτη `printf()` εμφανίζει την τιμή 323 στοιχισμένη στα αριστερά.
- Στη δεύτερη `printf()` το πλάτος πεδίου είναι 5 και γίνεται χρήση της σημαίας `+`. Εφόσον το 323 καταλαμβάνει 3 θέσεις, στην οθόνη εμφανίζεται ένας κενός χαρακτήρας και ακολούθως το +323.
- Στην τρίτη `printf()` η χρήση της σημαίας κενού χαρακτήρα οδηγεί στην εμφάνιση ενός κενού αριστερά της τιμής 323.
- Στην τέταρτη `printf()` το 323 εμφανίζεται σε οκταδική μορφή.
- Στην πέμπτη και έκτη `printf()` το 323 εμφανίζεται σε δεκαεξαδική μορφή.
- Στην τελευταία `printf()` προστίθενται δύο μηδενικά πριν το 323.

Παρατηρήσεις για την `scanf`

- Η κλήση της συνάρτησης `scanf()` διακόπτει τη ροή εκτέλεσης του προγράμματος, έως ότου ο χρήστης πληκτρολογήσει τα δεδομένα και πατήσει το πλήκτρο επαναφοράς (ENTER).
- Εάν στο τέλος του αλφαριθμητικού μορφοποίησης προστεθεί η ακολουθία διαφυγής αλλαγής γραμμής (`'\n'`), τότε η συνάρτηση `scanf()` υποχρεώνεται να προχωρήσει στην ανάγνωση του επόμενου μη κενού χαρακτήρα. Κατά συνέπεια, η ακόλουθη γραμμή κώδικα

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

οδηγεί στην ανάγνωση ενός ακεραίου και στην παύση της εκτέλεσης του προγράμματος, καθώς αναμένεται η εισαγωγή από τον χρήστη ενός μη κενού χαρακτήρα.



Παρατηρήσεις για την `scanf`

- Όταν η συνάρτηση `scanf()` χρησιμοποιείται για την ανάγνωση αριθμητικών τιμών, οι «λευκοί χαρακτήρες» που πιθανόν υπάρχουν πριν την αριθμητική τιμή (οι χαρακτήρας του κενού, της αλλαγής γραμμής και του στηλοθέτη), αγνοούνται.
- Επιτρέπεται η χρήση προσδιοριστή πλάτους μέσα στο αλφαριθμητικό μορφοποίησης. Κατά συνέπεια, στην ακόλουθη γραμμή κώδικα

```
scanf( "%4d", &x );
```

εάν ο χρήστης πληκτρολογήσει **57937425**, θα αποθηκευτούν στην ακέραια μεταβλητή **x** οι τιμές των πρώτων τεσσάρων ψηφίων, δηλαδή το περιεχόμενο της **x** θα γίνει **5793**.



Παρατηρήσεις για την `scanf`

- Η συνάρτηση `scanf()` μπορεί να λάβει προδιαγραφές μετατροπής, δηλαδή να θέσει περιορισμούς στους χαρακτήρες που μπορεί να αναγνώσει, μέσω του προσδιοριστή `[]`. Επιπλέον, με χρήση του συμβόλου `-` μπορούν να προσδιοριστούν ομάδες αποδεκτών χαρακτήρων, ενώ με χρήση του συμβόλου `^` μπορούν να αποκλειστούν χαρακτήρες. Για παράδειγμα, ο προσδιοριστής μορφοποίησης `%[K-W]` επιτρέπει την καταχώρηση μόνο των χαρακτήρων `'K', 'L', ..., 'W'`, ενώ ο προσδιοριστής `%[^K-W]` δεν επιτρέπει την ανάγνωση των χαρακτήρων αυτών.

Παράδειγμα χρήσης printf, scanf

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

int main()
{
    int v1,v2;
    char c1,c2;
    /* 1η ομάδα προτάσεων */
    printf( "\n\tGive two integers: " );
    scanf( " \t%d,%d",&v1,&v2 );
    printf( " \n\tv1=%d, v2=%d",v1,v2 );
    /* 2η ομάδα προτάσεων */
    printf( "\n\n\tGive a character: " );
    scanf( "\t%c",&c1 );
    printf( "\n\tc1=%c",c1 );
    /* 3η ομάδα προτάσεων */
    printf( "\n\n\tGive another integer: " );
    scanf( "\t%d",&v1 );
    printf( "\n\tv1=%d",v1 );
    /* 4η ομάδα προτάσεων */
    printf( "\n\n\tGive two new characters: " );
    scanf( "%c",&c1 );
    scanf( "%c",&c2 );
    printf( "\n\tc1=%c\tc2=%c",c1,c2 );

    return 0;
}
```

Give two integers: 13,27

v1=13, v2=27

Give a character: F

c1=F

Give another integer: -438

Give two new characters: CJ

c1=

c2=C



Παράδειγμα χρήσης `printf`, `scanf`

- Στην πρώτη ομάδα προτάσεων η συνάρτηση `scanf()` διαβάζει δύο ακεραίους, τις τιμές των οποίων αποδίδει στις ακέραιες μεταβλητές `v1` και `v2`.
- Στη δεύτερη ομάδα προτάσεων αναγιγνώσκεται ένας χαρακτήρας και αποθηκεύεται στη μεταβλητή τύπου χαρακτήρα `c1`.
- Στην τρίτη ομάδα προτάσεων γίνεται η ανάγνωση και αποθήκευση ενός ακεραίου.
- Στην τέταρτη ομάδα προτάσεων αναδεικνύεται το πρόβλημα στη λειτουργία της `scanf()` που διατυπώθηκε προηγουμένως: Ενώ επιχειρείται να αναγνωσθούν οι χαρακτήρες '`C`', ..., '`J`', στη μεταβλητή `c1` αποθηκεύεται ένας κενός χαρακτήρας και στη μεταβλητή `c2` αποθηκεύεται το '`C`'. Το παράδοξο αποτέλεσμα εξηγείται από το γεγονός ότι στο τέλος της τρίτης σειράς προτάσεων, η πληκτρολόγηση του **ENTER** μετά την τιμή **-438** οδήγησε την αποθήκευση του χαρακτήρα αλλαγής γραμμής στο `stdin`. Όταν ακολούθως πληκτρολογήθηκαν οι χαρακτήρες '`C`', ..., '`J`', αυτοί τοποθετήθηκαν με τη σειρά στο `stdin`. Έτσι, όταν έγιναν οι εκχωρήσεις τιμής, πρώτα αποδόθηκε στη μεταβλητή `c1` ο αποθηκευθείς λευκός χαρακτήρας της αλλαγής γραμμής, μετά αποθηκεύτηκε στη μεταβλητή `c2` ο χαρακτήρας '`C`', ενώ ο χαρακτήρας '`J`' παρέμεινε στο `stdin`.

I/O κονσόλας

Η I/O κονσόλας αναφέρεται στις λειτουργίες που γίνονται στο πληκτρολόγιο και στην οθόνη του υπολογιστή. Εκτός από τις `printf()` και `scanf()` που χρησιμοποιήθηκαν προγουμένως (και αποτελούν τη **φορμαρισμένη/μορφοποιούμενη** I/O κονσόλας γιατί μπορούν να διαβάζουν δεδομένα σε διάφορες φόρμες), υπάρχει μία σειρά απλούστερων συναρτήσεων που αναπτύσσεται ακολούθως.

Οι συναρτήσεις *getche*, *getch*

- Η συνάρτηση *getche()* διαβάζει ένα χαρακτήρα από την κύρια είσοδο (πληκτρολόγιο). Αναμένει έως ότου πατηθεί ένα πλήκτρο και στη συνέχεια επιστρέφει την τιμή του, εμφανίζοντας στην οθόνη το πλήκτρο που πατήθηκε.
- Το πρωτότυπο της *getche()* είναι το ακόλουθο:
`int getche(void);`
- Η *getche()* επιστρέφει μεν έναν ακέραιο αλλά το byte χαμηλής τάξης περιέχει τον χαρακτήρα. Η χρήση ακεραίων γίνεται για λόγους συμβατότητας με τον αρχικό μεταγλωττιστή της UNIX C.

Οι συναρτήσεις *getche*, *getch* (συνέχεια)

- Στην Dev C++ το αρχείο κεφαλίδας της συνάρτησης *getche()* βρίσκεται στο *stdio.h*.
- Η συνάρτηση *getch()* αποτελεί παραλλαγή της *getche()* και βρίσκεται στο *stdio.h*. Λειτουργεί όπως ακριβώς η *getche* με τη διαφορά ότι η *getch()* **δεν εμφανίζει τον πληκτρολογηθέντα χαρακτήρα στην οθόνη**.
- Το πρωτότυπο της *getch()* είναι το ακόλουθο:
`int getch(void);`

Η συνάρτηση *getchar*

- Η συνάρτηση *getchar()* διαβάζει ένα χαρακτήρα από την κύρια είσοδο και τον επιστρέφει στο πρόγραμμα. Αποτελεί παραλλαγή της *getche()*. Είναι η αρχική συνάρτηση εισόδου χαρακτήρων που βασίζεται στο UNIX. Το πρόβλημα με τη συνάρτηση αυτή είναι ότι κρατά την είσοδο στην περιοχή προσωρινής αποθήκευσης μέχρι να δώσουμε επαναφορά κεφαλής. Έτσι, μετά την επιστροφή της *getchar()* περιμένουν ένας ή περισσότεροι χαρακτήρες στην ουρά εισόδου.
- Το πρωτότυπο της *getchar()* είναι το ακόλουθο:
`int getchar(void);`
- Το αρχείο κεφαλίδας της συνάρτησης *getchar()* βρίσκεται στο *stdio.h*.

Η συνάρτηση `putchar`

- Η συνάρτηση `putchar()` εμφανίζει στην οθόνη τον χαρακτήρα που έχει ως όρισμα (π.χ. `c`), στην τρέχουσα θέση του δρομέα.
- Το πρωτότυπο της `putchar()` είναι το ακόλουθο:
`int putchar(int c);`
- Η `putchar()` επιστρέφει μεν έναν ακέραιο αλλά το byte χαμηλής τάξης περιέχει τον χαρακτήρα. Η χρήση ακεραίων γίνεται για λόγους συμβατότητας με τον αρχικό μεταγλωττιστή της UNIX C.
- Το αρχείο κεφαλίδας της συνάρτησης `putchar()` βρίσκεται στο `stdio.h`.

Η συνάρτηση *kbhit*

- Η συνάρτηση *kbhit()* (keyboard hit) ελέγχει κατά πόσον ο χρήστης έχει πατήσει κάποιο πλήκτρο. Εφόσον έχει πατήσει κάποιο πλήκτρο η συνάρτηση επιστρέφει ως αληθής, σε αντίθετη περίπτωση επιστρέφει ως ψευδής. Η συνάρτηση *kbhit()* χρησιμοποιείται κυρίως για να διακόπτει ο χρήστης το πρόγραμμα κατά το δοκούν.
- Το πρωτότυπο της *kbhit()* είναι το ακόλουθο:
`int kbhit(void);`
- Στη Dev C++ το αρχείο κεφαλίδας της συνάρτησης *kbhit()* βρίσκεται στο *stdio.h*.

Παράδειγμα: Το ακόλουθο πρόγραμμα παίρνει χαρακτήρες από το πληκτρολόγιο και μετατρέπει τα κεφαλαία σε μικρά και τούμπαλιν. Το πρόγραμμα σταματά μόλις πληκτρολογηθεί μία τελεία. Το αρχείο-κεφαλίδα **ctype.h** απαιτείται για τη συνάρτηση **islower()**, που αληθεύει αν το όρισμά της είναι σε μικρά γράμματα, και τις συναρτήσεις **toupper()**, **tolower()**, που μετασχηματίζουν τα γράμματα.

(να μη γίνει ιδιαίτερη μνεία για την επαναληπτική πρόταση και την υπό συνθήκη διακλάδωση. Θα μελετηθούν εκτενώς αργότερα.)

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

```
#include <ctype.h>
```

```
int main() {
```

```
    char ch;
```

```
    printf( "Start writing letters without ENTER\n\n");
```

```
    do {
```

```
        ch=getche();    printf( " -> " );
```

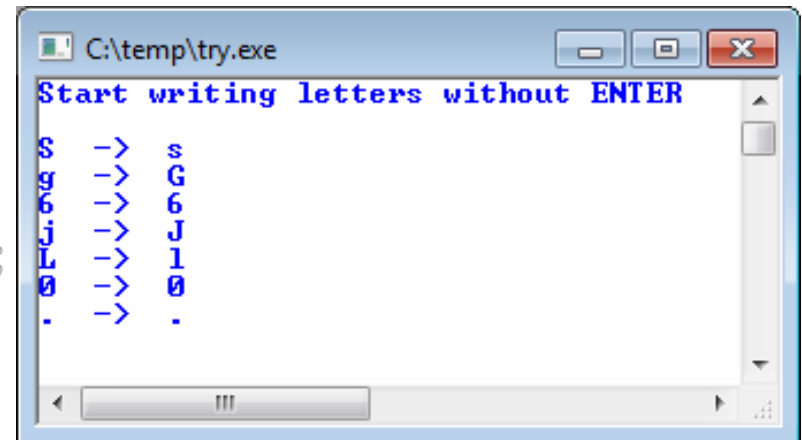
```
        if (islower(ch)) putchar(toupper(ch));
```

```
        else putchar(tolower(ch));    printf( "\n" );
```

```
    } while (ch!='.');
```

```
    return 0;
```

```
}
```



Θεματική ενότητα 3:

Τελεστές – εκφράσεις

Τελεστές (operators) – Εκφράσεις (expressions)

- Σύμβολα ή λέξεις που αναπαριστούν συγκεκριμένες διεργασίες, οι οποίες εκτελούνται πάνω σε ένα ή περισσότερα δεδομένα.
- Τα δεδομένα καλούνται **τελεστέοι** (operands) και μπορούν να είναι μεταβλητές, σταθερές ή ακόμη και κλήσεις συναρτήσεων.
- Οι τελεστές χρησιμοποιούνται για τον σχηματισμό εκφράσεων. Μία έκφραση, εν γένει, αποτελείται από έναν ή περισσότερους τελεστέους και προαιρετικά από έναν ή περισσότερους τελεστές.
- Κάθε έκφραση έχει μία τιμή, η οποία υπολογίζεται με ορισμένους κανόνες.

Σύμβολα συνηθισμένων τελεστών

Δυαδικός τελεστής	Μαθηματικό σύμβολο	C	Pascal
μικρότερο	<	<	<
μικρότερο ή ίσο	≤	<=	<=
ίσο	=	==	=
διάφορο	≠	!=	<>
μεγαλύτερο	>	>	>
μεγαλύτερο ή ίσο	≥	>=	>=
πρόσθεση	+	+	+
αφαίρεση	-	-	-
πολλαπλασιασμός	*	*	*
διαίρεση πραγματικών	/	/	/
διαίρεση ακεραίων	div	/	div
υπόλοιπο διαίρ. ακερ.	Mod	%	mod

Συμβολισμοί στο σχηματισμό εκφράσεων

Ένας **δυαδικός** (binary) τελεστής μπορεί να τοποθετηθεί:

- Μεταξύ των δεδομένων στα οποία ενεργεί, όπως στην έκφραση $x+y$, οπότε έχουμε τη σημειογραφία **ένθεσης** ή ένθετου τελεστή (infix notation).
- Πριν από τους τελεστέους, όπως στην έκφραση $+x y$, οπότε έχουμε τη σημειογραφία **πρόθεσης** ή **προπορευόμενου τελεστή** (prefix notation).
- Μετά από τους τελεστέους, όπως στην έκφραση $x y+$, οπότε έχουμε τη σημειογραφία **επίθεσης** ή **παρελκόμενου τελεστή** (postfix notation).

Προσοχή:

- Καλύτερη πρακτική: να μη χρησιμοποιούνται τελεστές σε μεικτούς τύπους:

```
out_int = my1_int + my2_int;    // καλό  
out_float = my_double / my_int; // κακό
```

- !!ΚΙΝΔΥΝΟΣ!!**: Όταν γίνεται διαίρεση ακεραίων το αποτέλεσμα είναι το πηλίκο, δηλαδή $5 / 2 = 2$ κι όχι 2.5
- Για να ληφθεί ως αποτέλεσμα αριθμός κινητής υποδιαστολής, τουλάχιστον ένας από τους τελεστέους πρέπει να είναι αριθμός κινητής υποδιαστολής:

$5.0 / 2$ υπολογίζεται ως 2.5

Οι εκφράσεις είναι συχνά **φωλιασμένες** (nested):

$((n+5) \leq a) \ \&\& \ q$

– Η έκφραση αυτή

Υπολογίζεται και γίνεται ένας όρος:

$((n+5) \leq a) \ \&\& \ q$

στη συνέχεια η έκφραση υπολογίζεται και γίνεται ένας όρος:

$((n+5) \leq a) \ \&\& \ q$

Προτεραιότητα (*precedence*) και προσεταιριστικότητα (*associativity*)

- Πώς θα υπολογισθεί η έκφραση $17 * 8 - 2$;

Είναι $17 * (8 - 2)$ ή $(17 * 8) - 2$;

- Οι ανωτέρω εκφράσεις οδηγούν σε διαφορετικά αποτελέσματα. Κατά συνέπεια απαιτούνται κανόνες.
- Η σειρά εφαρμογής των τελεστών ονομάζεται **εφαρμοστική σειρά** (applicative order).

Προτεραιότητα και προσεταιριστικότητα τελεστών στη C

ΤΕΛΕΣΤΕΣ

() [] ->
! ~ ++ -- + - * & (τύπος)
size of
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
?:
= += -= *= /= %= &= ^= |=
<<= >>=
,

ΠΡΟΣΕΤΑΙΡΙΣΤΙΚΟΤΗΤΑ

Από αριστερά προς τα δεξιά

Από δεξιά προς τα αριστερά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από αριστερά προς τα δεξιά

Από δεξιά προς τα αριστερά

Από δεξιά προς τα αριστερά

Από αριστερά προς τα δεξιά

Προτεραιότητα και προσεταιριστικότητα

Παραδείγματα:

$$X = 17 - 2 * 8$$

Απάντηση: $X = 17 - (2 * 8)$, $X = 1$

$$Y = 17 - 2 - 8$$

Απάντηση: $Y = (17 - 2) - 8$, $Y = 7$

$$Z = 10 + 9 * ((8 + 7) \% 6) + 5 * 4 \% 3 * 2 + 1 (;$$

Μπερδευτήκατε; Τότε χρησιμοποιείτε παρενθέσεις στον κώδικά σας.

Τελεστές αύξησης και μείωσης

- **Τελεστής αύξησης** (increment operator): **++**

Αντί για `num = num + 1;`

γράφουμε `num++;`

- **Τελεστής μείωσης** (decrement operator): **--**

Αντί για `num = num - 1;`

γράφουμε `num--;`

Μήπως γίνεται πλέον φανερό από πού έλαβε η C++ το όνομά της;

Παράδειγμα: Στην ακολουθία εκφράσεων που παρατίθεται ακολούθως παρουσιάζεται ενδεικτικά η λειτουργία των προπορευόμενων και παρελκόμενων τελεστών μοναδιαίας αύξησης και μείωσης.

πρόταση	τιμή x	τιμή y
int x = 10, y = 20;	10	20
++x;	11	20
y = --x;	10	10
y = x-- + y;	9	20
y = y - x++;	10	11

Παράδειγμα: Να προσδιορισθεί η τιμή των x και z μετά την εκτέλεση κάθε μίας από τις παρακάτω προτάσεις, θεωρώντας ότι, πριν την εκτέλεση της κάθε πρότασης, οι τιμές των x και y είναι το 10 και 20 αντίστοιχα.

α) $z = ++x + y;$

β) $z = --x + y;$

γ) $z = x++ + y;$

δ) $z = x-- + y;$

Λύση: Στην περίπτωση του προπορευόμενου τελεστή, το σύστημα πρώτα εκτελεί την αύξηση ή μείωση και μετά χρησιμοποιεί τη νέα τιμή της μεταβλητής στον υπολογισμό της τιμής της έκφρασης (προτάσεις α και β). Αντίθετα, στην περίπτωση του παρελκόμενου τελεστή το σύστημα πρώτα χρησιμοποιεί την τιμή της μεταβλητής για τον υπολογισμό της τιμής της έκφρασης και μετά εκτελεί την αύξηση ή μείωση της τιμής της μεταβλητής (προτάσεις γ και δ).



Αποτελέσματα:

Πρόταση	Τιμή x	Τιμή z
$z = ++x + y;$	11	31
$z = --x + y;$	9	29
$z = x++ + y;$	11	30
$z = x-- + y;$	9	30

Τελεστές ανάθεσης (assignment)

- $x^* = 10;$ εκτελεί την πράξη του πολλαπλασιασμού μεταξύ των x και 10 και εκχωρεί το αποτέλεσμα στο x . Αντιστοιχεί στην πρόταση $x = x * 10;$
- $x^* = y + 1;$ Αντιστοιχεί στην πρόταση $x = x * (y + 1);$ κι ΟΧΙ στην πρόταση $x = x * y + 1;$
- Τελεστές ανάθεσης δημιουργούν κι οι τελεστές διαχείρισης δυαδικών ψηφίων (bitwise operators). Οι τελεστές αυτοί είναι: $>>=$ $<<=$ $\&=$ $\wedge=$ $|=$. Περισσότερες λεπτομέρειες σε επόμενη διαφάνεια.

Τελεστές ανάθεσης (συνέχεια)

- Οι τελεστές ανάθεσης μαζί με τους τελεστές αύξησης/μείωσης γίνονται αιτία δημιουργίας παρενεργειών (side effects), για το λόγο αυτό αναφέρονται και ως **παρενεργοί τελεστές** (side effect operators).
- Οι παρενέργειες αυτές έχουν ως αποτέλεσμα την απροσδιόριστη συμπεριφορά του συστήματος ως προς τον τρόπο υπολογισμού της τιμής της μεταβλητής **i** σε εκφράσεις όπως: **i = n[i++]**; ή **i = ++i + 1**;