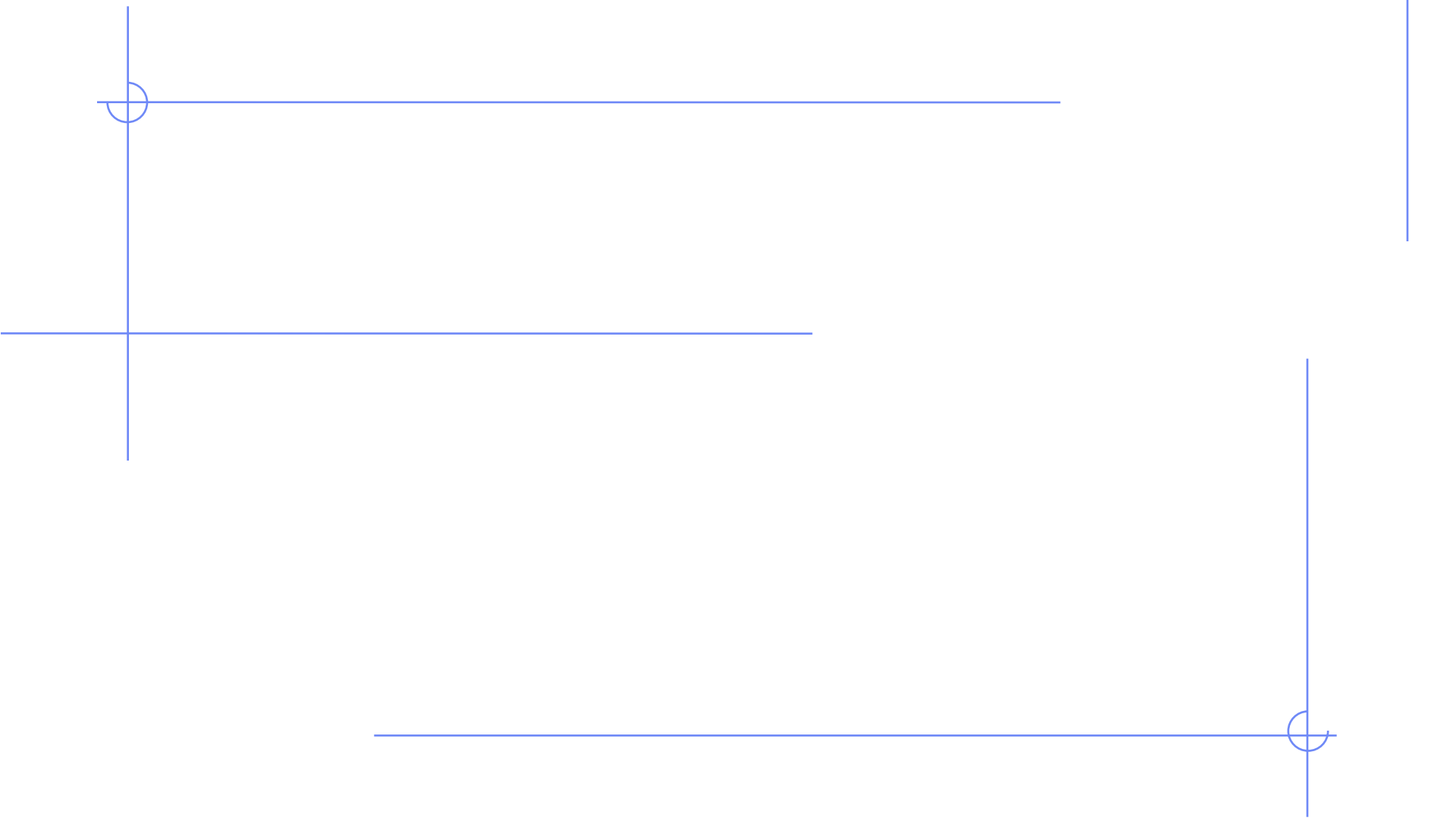
«Προγραμματισμός Ι»

2- Τύποι Δεδομένων – Δήλωση Μεταβλητών – Έξοδος Δεδομένων - Είσοδος Δεδομένων

Δρ. Χαράλαμπος Καρανίκας
Π.Σ. Μηχανικών Πληροφορικής



Οι διαφάνειες βασίζονται στο βιβλίο: C: Από τη Θεωρία στην Εφαρμογή Γ. Σ. Τσελίκης – Ν. Δ. Τσελίκας

Μνήμη και Μεταβλητές

- Σχέση Μνήμης Υπολογιστή και Μεταβλητών
  - Η μνήμη (RAM) ενός υπολογιστή αποτελείται από πολλά εκατομμύρια θέσεις αποθήκευσης δεδομένων που έχουν διαδοχική αρίθμηση
  - Το μέγεθος κάθε θέσης μνήμης είναι μία οκτάδα (byte)
  - Π.χ. σκεφτείτε ότι ένας παλιός υπολογιστής με μόνο 16MB μνήμης έχει μνήμη:

    16 * 1.024 = 16.384 kbytes

    16.384 * 1.024 = 16.777.216 θέσεις μνήμης (bytes)
  - Κάθε θέση μνήμης μπορεί να έχει ένα όνομα και ένα περιεχόμενο
  - Μεταβλητή ονομάζεται μία θέση μνήμης που έχει ένα συγκεκριμένο όνομα
  - Η τιμή μίας μεταβλητής είναι το περιεχόμενο αυτής της θέσης μνήμης (ή των θέσεων μνήμης, όπως θα δούμε) και μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης του προγράμματος

Ονόματα Μεταβλητών

- **Απαράβατοι κανόνες κατά τη δήλωση του ονόματος μίας μεταβλητής**
  - Μπορεί να αποτελείται από **πεζά και κεφαλαία** γράμματα του **λατινικού αλφαβήτου** και **ψηφία**
  - Αποδεκτός είναι επίσης και ο **χαρακτήρας υπογράμμισης** '_' (underscore)
  - Ο **πρώτος χαρακτήρας** **πρέπει** να είναι γράμμα ή ο χαρακτήρας υπογράμμισης '_'
  - Η γλώσσα C είναι **case sensitive** (δηλ. κάνει διάκριση μεταξύ των πεζών και κεφαλαίων γραμμάτων)
    - Συνεπώς, η μεταβλητή με το όνομα `nick` είναι διαφορετική από τη μεταβλητή με το όνομα `Nick`
  - Οι **δεσμευμένες λέξεις** της C **απαγορεύεται να χρησιμοποιηθούν** ως ονόματα μεταβλητών

Δεσμευμένες Λέξεις της C

auto	do	goto	signed	unsigned
break	double	if	sizeof	void
case	else	int	static	volatile
char	enum	long	struct	while
const	extern	register	switch	
continue	for	return	typedef	
default	float	short	union	

Παρατηρήσεις

- Το όνομα που επιλέγετε να δώσετε σε μία μεταβλητή είναι χρήσιμο να περιγράφει όσο το δυνατόν καλύτερα τον σκοπό της μεταβλητής μέσα στο πρόγραμμα, ώστε ο κώδικας να είναι πιο ευανάγνωστος
  - Π.χ. το όνομα μίας μεταβλητής που υπολογίζει το άθροισμα κάποιων αριθμών είναι προτιμότερο να είναι **`sum`** αντί για **`var`**

- Αν το όνομα που επιλέξατε για μία μεταβλητή αποτελείται από δύο ή και περισσότερες λέξεις, τότε προτείνεται να τις διαχωρίζετε μεταξύ τους με τον χαρακτήρα υπογράμμισης '_', έτσι ώστε να διευκολύνεται η ερμηνεία τους
  - Π.χ. το όνομα μίας μεταβλητής που υπολογίζει τον αριθμό των βιβλίων σε μία βιβλιοθήκη είναι προτιμότερο να είναι **`books_number`** αντί για **`booksnumber`**

- Προτείνεται, τα ονόματα μεταβλητών να αποτελούνται μόνο από πεζά γράμματα

Δήλωση Μεταβλητών

- Για να χρησιμοποιήσετε μία μεταβλητή μέσα σε ένα πρόγραμμα πρέπει πρώτα να τη δηλώσετε

- Η δήλωση μίας μεταβλητής γίνεται με τον ακόλουθο τρόπο:

```
τύπος_δεδομένων όνομα_μεταβλητής;
```

- Το `όνομα_μεταβλητής` είναι το τυχαίο όνομα που επιλέγει ο προγραμματιστής σύμφωνα με τους κανόνες και τις παρατηρήσεις που είπαμε προηγουμένως

- Ο `τύπος_δεδομένων` είναι ένας από τους αριθμητικούς τύπους δεδομένων που υποστηρίζει η γλώσσα C
  - Π.χ. η δεσμευμένη λέξη `int` χρησιμοποιείται για τη δήλωση ακέραιων μεταβλητών, δηλαδή μεταβλητών που μπορούν να έχουν μόνο ακέραιες τιμές ενώ η δεσμευμένη λέξη `float` χρησιμοποιείται για τη δήλωση πραγματικών μεταβλητών, δηλαδή μεταβλητών που μπορούν να έχουν τιμές με κλασματικό μέρος

Τύποι Μεταβλητών

Τύπος	Συνηθισμένο μέγεθος (bytes)	Εύρος τιμών (min-max)	Ψηφία ακρίβειας
char	1	-128 … 127	
short	2	-32.768 … 32.767	
int	4	-2.147.483.648…2.147.483.647	
long	4	-2.147.483.648…2.147.483.647	
float	4	Χαμηλότερη θετική τιμή: $1.17*10^{-38}$ Υψηλότερη θετική τιμή: $3.4*10^{38}$	6
double	8	Χαμηλότερη θετική τιμή: $2.2*10^{-308}$ Υψηλότερη θετική τιμή: $1.8*10^{308}$	15
unsigned char	1	0 … 255	
unsigned short	2	0 … 65535	
unsigned int	4	0 … 4.294.967.295	
unsigned long	4	0 … 4.294.967.295	

- Π.χ.

```
int a; /* Δήλωση ακέραιας μεταβλητής με όνομα a. */
float b; /* Δήλωση πραγματικής μεταβλητής με όνομα b. */
```

Παρατηρήσεις

- Πολλές μεταβλητές του ίδιου τύπου μπορούν να δηλωθούν στην ίδια γραμμή, αρκεί να διαχωρίζονται μεταξύ τους με κόμμα (,)
  - Δηλαδή, αντί να δηλώσετε τις μεταβλητές `a`, `b` και `c` σε τρεις ξεχωριστές γραμμές:

```
int a;
int b;
int c;
```

    μπορείτε να τις δηλώσετε σε μία γραμμή ως εξής:

```
int a, b, c;
```

- Το μέγεθος της μνήμης που δεσμεύει ένας τύπος δεδομένων μπορεί να διαφέρει από υπολογιστή σε υπολογιστή
  - Δηλαδή, ο τύπος `int` μπορεί να δεσμεύει 2 bytes σε κάποιον υπολογιστή και όχι 4 bytes
  - Για να μάθετε πόσες οκτάδες δεσμεύει ένας τύπος δεδομένων στον υπολογιστή που εργάζεστε πρέπει να χρησιμοποιήσετε τον τελεστή `sizeof` → Θα τον δούμε παρακάτω

- Να χρησιμοποιείτε τον τύπο `float` μόνο όταν η ακρίβεια των δεκαδικών ψηφίων δεν είναι τόσο σημαντική στο πρόγραμμά σας
- Σε περίπτωση που χρειάζεστε υψηλή ακρίβεια των δεκαδικών ψηφίων, να χρησιμοποιείτε τον τύπο `double`

Εκχώρηση τιμών σε Μεταβλητές (Ι)

- Η εκχώρηση μίας τιμής σε μία μεταβλητή γίνεται είτε μαζί με τη δήλωση της μεταβλητής είτε αργότερα

- Π.χ. με την πρώτη εντολή δηλώνεται μία ακέραια μεταβλητή (**int**) με όνομα **a** και μετά της εκχωρείται η τιμή 100

```
int a;
a = 100;
```

- Εναλλακτικά, θα μπορούσαμε να γράψουμε την εκχώρηση τιμής μαζί με τη δήλωση:

```
int a = 100;
```

Εκχώρηση τιμών σε Μεταβλητές (ΙΙ)

- Επίσης, επιτρέπεται η απόδοση αρχικών τιμών σε περισσότερες από μία μεταβλητές ίδιου τύπου μαζί με τη δήλωσή τους, π.χ.

```
int a = 100, b = 200, c = 300;
```

- Για την εκχώρηση μίας πραγματικής τιμής σε μία μεταβλητή τύπου `float` χρησιμοποιείται η τελεία (.) για το δεκαδικό μέρος και όχι το κόμμα (,) π.χ.

```
float a = 1.24;
```

- Αν μπροστά από μία ακέραια τιμή υπάρχει το ψηφίο **0**, τότε αυτή η τιμή ερμηνεύεται σαν οκταδικός αριθμός
  - Π.χ. με την παρακάτω εντολή η τιμή που εκχωρείται στη μεταβλητή **a** δεν είναι **100**, αλλά **64**

```
int a = 0100;
```

- Παρομοίως, αν μπροστά από μία ακέραια τιμή υπάρχει το **0x** ή το **0X**, τότε αυτή η τιμή ερμηνεύεται σαν δεκαεξαδικός αριθμός
  - Π.χ. με την παρακάτω εντολή η τιμή της μεταβλητής **a** γίνεται **16**.

```
int a = 0x10;
```

Παρατηρήσεις (Ι)

- Η τιμή μίας μεταβλητής μπορεί (προφανώς) να αλλάζει μέσα στο πρόγραμμα
- Όταν γίνεται χρήση μίας μεταβλητής στο πρόγραμμα χρησιμοποιείται πάντα η τελευταία τιμή της και όχι κάποια από τις προηγούμενες τιμές της

```
#include <stdio.h>
int main()
{
        int a;
        a = 1;
        a = 2;
        a = 3;
        printf("Val = %d\n",a);
        return 0;
}
```

- Ποια τιμή εκτυπώνεται στην οθόνη???

Παρατηρήσεις (ΙΙ)

- Η τιμή που εκχωρείται σε μία μεταβλητή πρέπει να συμβαδίζει με τον τύπο της μεταβλητής
  - Π.χ. με την εντολή:

```
int a = 10.9;
```

η τιμή της μεταβλητής **a** γίνεται **10**, γιατί η μεταβλητή **a** δηλώνεται σαν ακέραια μεταβλητή και όχι σαν πραγματική και το δεκαδικό μέρος αποκόπτεται (Προσοχή!! Δεν στρογγυλοποιείται)

- Η τιμή που εκχωρείται σε μία μεταβλητή πρέπει να είναι μέσα στο επιτρεπτό εύρος τιμών
  - Π.χ. με την εντολή:

```
char ch = 130;
```

η τιμή της μεταβλητής **ch** δεν γίνεται **130**, γιατί το εύρος τιμών μίας μεταβλητής τύπου char είναι από **-128** έως **127**. Άρα, η τιμή **130** είναι μία τιμή εκτός των επιτρεπτών ορίων

Παρατηρήσεις (III)

- Η τιμή μίας πραγματικής μεταβλητής μπορεί να είναι και ακέραια
  - Π.χ. επιτρέπεται να γράψουμε:

```
float a = 50;
```

  γιατί είναι ισοδύναμο με:

```
float a = 50.0;
```

- Η τιμή μίας πραγματικής μεταβλητής μπορεί να γραφεί και με επιστημονική σημειογραφία (συνήθως χρησιμοποιείται όταν η τιμή είναι πολύ μικρή ή πολύ μεγάλη)
  - Π.χ. αντί για

```
a = 0.085;
```

  μπορούμε να γράψουμε

```
a = 85E-3;
```

- Το γράμμα **E** ή **e** αναπαριστά το 10, ενώ ο αριθμός που το ακολουθεί είναι η θετική ή αρνητική δύναμη του 10.

- Δηλαδή, η έκφραση 85E-3 αντιστοιχεί στον αριθμό $85*10^{-3}$

Σταθερές (Ι)

- Σταθερά ονομάζεται μία μεταβλητή που η τιμή της δεν μπορεί να αλλάξει μέσα στο πρόγραμμα
- Για να δηλωθεί μία μεταβλητή σαν σταθερά, πρέπει να προηγηθεί η λέξη `const` πριν από τον τύπο της μεταβλητής
- Επίσης, μαζί με τη δήλωση της σταθεράς, πρέπει να της εκχωρηθεί και μία αρχική τιμή, η οποία δεν θα μπορεί να αλλάξει μέσα στο πρόγραμμα

  - Π.χ. με την επόμενη εντολή η ακέραια μεταβλητή `a` δηλώνεται σαν σταθερά και της εκχωρείται (μόνιμα) η τιμή `10`

    ```
    const int a = 10;
    ```

  - Αν σε κάποιο σημείο του προγράμματος επιχειρήσουμε να της αλλάξουμε τιμή, π.χ. να γράψουμε:

    ```
    a = 100;
    ```

    τότε ο μεταγλωττιστής θα εμφανίσει μήνυμα λάθους για μη επιτρεπτή ενέργεια

Σταθερές (ΙΙ)

- Εναλλακτικός τρόπος για τη δήλωση μίας σταθεράς είναι η χρήση της οδηγίας **#define**, η οποία χρησιμοποιείται για τη δήλωση μακροεντολών
- Συνήθως, μία μακροεντολή αντιστοιχίζει ένα συμβολικό όνομα με κάποια αριθμητική τιμή
- Για τη δήλωση μακροεντολών, η οδηγία **#define** χρησιμοποιείται ως εξής:

```
#define όνομα_μακροεντολής τιμή
```

- Π.χ. η εντολή:

```
#define NUM 100
```

δηλώνει τη μακροεντολή με όνομα **NUM** και τιμή 100

- Η **NUM** μπορεί να χρησιμοποιηθεί οπουδήποτε μέσα στο πρόγραμμα
- Ο μεταγλωττιστής όταν συναντάει τη **NUM** μέσα στο πρόγραμμα την αντικαθιστά με την τιμή 100

Παρατηρήσεις

- Οι δηλώσεις των μακροεντολών με την οδηγία **#define** είναι προτιμότερο να γίνονται πριν από τη συνάρτηση **main()**
- Τα ονόματα των μακροεντολών με την οδηγία **#define** είναι προτιμότερο να δηλώνονται με κεφαλαία γράμματα
- Στο τέλος της οδηγίας **#define** δεν μπαίνει ελληνικό ερωτηματικό (;)
- Π.χ.

```
#include <stdio.h>

#define NUM 100

int main()
{
        int a,b,c;
        a = 20 - NUM;
        b = 20 + NUM;
        c = 3*NUM;
        return 0;
}
```

Η συνάρτηση `printf()`

- Η συνάρτηση `printf()` χρησιμοποιείται για την εμφάνιση δεδομένων στο αρχείο εξόδου `stdout` *(standard output stream)* , το οποίο εξ' ορισμού συνδέεται με την οθόνη

- Η συνάρτηση `printf()` δέχεται μία μεταβλητή λίστα παραμέτρων

  - Η πρώτη παράμετρος είναι ένα **αλφαριθμητικό μορφοποίησης** *(format string)*, δηλαδή **μία ακολουθία χαρακτήρων μέσα σε διπλά εισαγωγικά** (" ") η οποία καθορίζει τον τρόπο με τον οποίο θα εμφανιστούν τα δεδομένα στην οθόνη

  - Οι επόμενες παράμετροι είναι **προαιρετικές** και, αν υπάρχουν, η `printf()` εμφανίζει τις τιμές τους στην οθόνη

- Το αλφαριθμητικό μορφοποίησης (format string) μπορεί να περιέχει:
  - **Απλούς χαρακτήρες** (οι οποίοι εμφανίζονται όπως είναι στην οθόνη)
  - **Ακολουθίες Διαφυγής**
  - **Προσδιοριστικά Μετατροπής**

Ακολουθία Διαφυγής

- Μία ακολουθία διαφυγής (escape sequence) χρησιμοποιείται είτε για να μετακινηθεί ο δρομέας (cursor) σε κάποια θέση της οθόνης είτε για την εμφάνιση κάποιων ειδικών χαρακτήρων
- Μία ακολουθία διαφυγής αποτελείται από μία ανάστροφη κεκλιμένη (\) (backslash) και έναν ειδικό χαρακτήρα

Ακολουθία διαφυγής	Σημασία
\a	Χρησιμοποιείται για τη δημιουργία ηχητικού σήματος.
\b	Χρησιμοποιείται για τη διαγραφή του τελευταίου χαρακτήρα, όπως το πλήκτρο backspace.
\n	Χρησιμοποιείται για την αλλαγή γραμμής, όπως το πλήκτρο Enter.
\r	Χρησιμοποιείται για την επαναφορά του δρομέα στην αρχή της τρέχουσας γραμμής.
\t	Χρησιμοποιείται για τη μετακίνηση του δρομέα σε μία απόσταση ίση με το μήκος του tab, όπως το πλήκτρο tab.
\\	Χρησιμοποιείται για την εμφάνιση της ανάστροφης κεκλιμένης (\).
\"	Χρησιμοποιείται για την εμφάνιση των διπλών εισαγωγικών (").

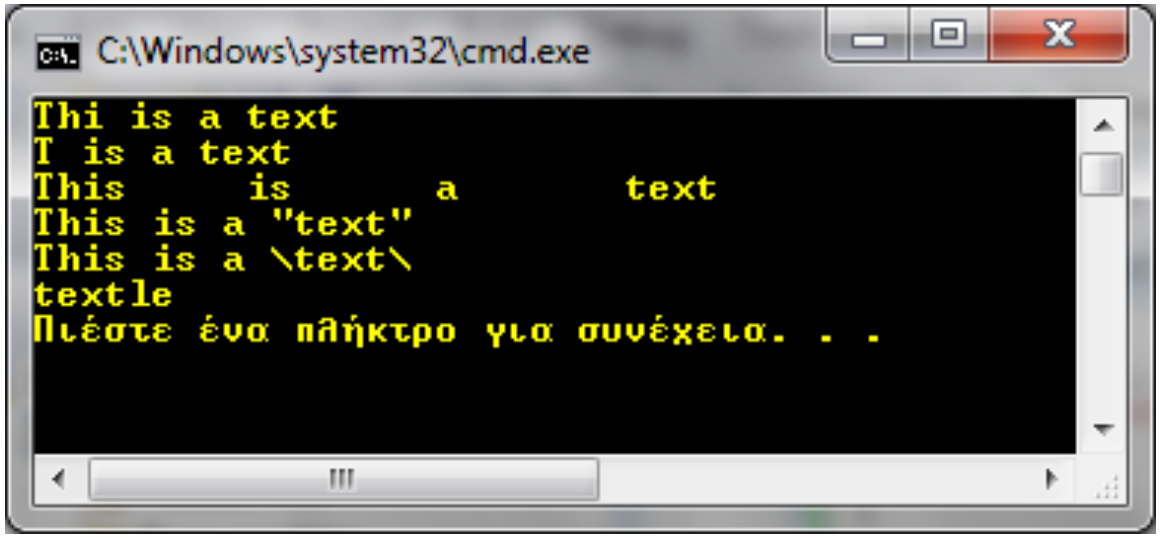
Προσδιοριστικό Μετατροπής

- Ένα προσδιοριστικό μετατροπής (conversion specification) αρχίζει με τον χαρακτήρα % και ακολουθείται από έναν ή περισσότερους χαρακτήρες μετατροπής που έχουν ειδική σημασία

Χαρακτήρας μετατροπής	Σημασία
c	Χρησιμοποιείται για την εμφάνιση του χαρακτήρα που αντιστοιχεί σε μία ακέραια τιμή.
d ή i	Χρησιμοποιείται για την εμφάνιση ενός ακεραίου.
u	Χρησιμοποιείται για την εμφάνιση ενός μη-προσημασμένου ακεραίου.
f	Χρησιμοποιείται για την εμφάνιση ενός πραγματικού αριθμού. Η εξ' ορισμού ακρίβεια είναι έξι δεκαδικά ψηφία.
s	Χρησιμοποιείται για την εμφάνιση των χαρακτήρων ενός αλφαριθμητικού.
e ή E	Χρησιμοποιείται για την εμφάνιση ενός πραγματικού αριθμού σε επιστημονική μορφή. Ανάλογα με την επιλογή, εμφανίζεται το γράμμα e ή E πριν από τον εκθέτη.
g ή G	Χρησιμοποιείται για την εμφάνιση ενός πραγματικού αριθμού σε κανονική ή επιστημονική μορφή.
p	Χρησιμοποιείται για την εμφάνιση μίας διεύθυνσης μνήμης σε δεκαεξαδική μορφή.
x ή X	Χρησιμοποιείται για την εμφάνιση ενός μη-προσημασμένου ακεραίου σε δεκαεξαδική μορφή. Με το %x τα δεκαεξαδικά ψηφία (a-f) εμφανίζονται πεζά, ενώ με το %X εμφανίζονται ως κεφαλαία (A-F).
o	Χρησιμοποιείται για την εμφάνιση ενός μη-προσημασμένου ακεραίου σε οκταδική μορφή.
%	Χρησιμοποιείται για την εμφάνιση του χαρακτήρα %.

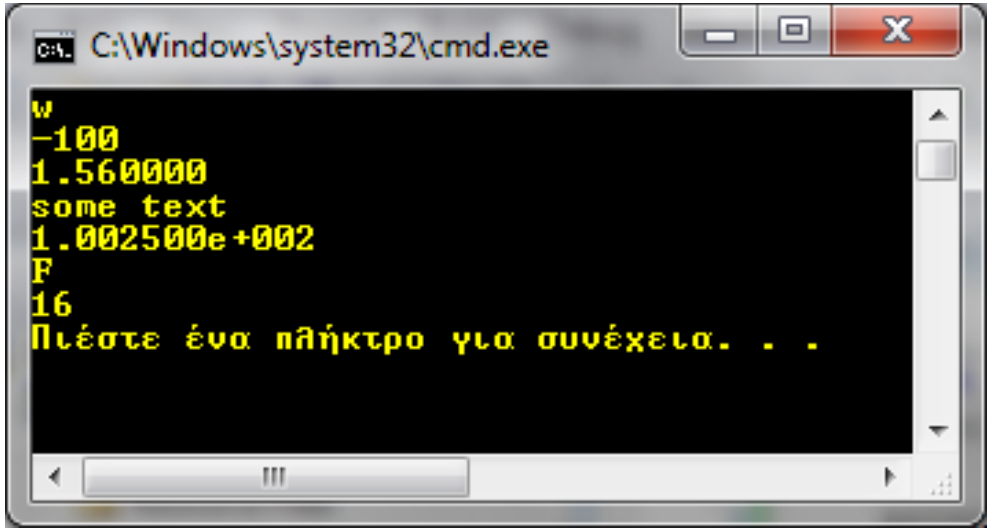
Παραδείγματα (I)

```c
#include <stdio.h>
int main()
{
	printf("\a");
	printf("This\b is a text\n");
	printf("This\b\b\b is a text\n");
	printf("This\t is\t a\t text\n");
	printf("This is a \"text\"\n");
	printf("This is a \\text\\\n");
	printf("Sample\rtext\n");
	return 0;
}
```



Παραδείγματα (II)

```c
#include <stdio.h>
int main()
{
	printf("%c\n",'w');
	printf("%d\n",-100);
	printf("%f\n",1.56);
	printf("%s\n","some text");
	printf("%e\n",100.25);
	printf("%X\n",15);
	printf("%o\n",14);
	return 0;
}
```



Εμφάνιση μεταβλητών

- Τα ονόματα των μεταβλητών εισάγονται στην `printf()` μετά τα διπλά εισαγωγικά (" ") με τη χρήση κόμματος (,) και αν οι μεταβλητές είναι περισσότερες από μία πρέπει και αυτές να διαχωρίζονται μεταξύ τους με κόμμα (,)

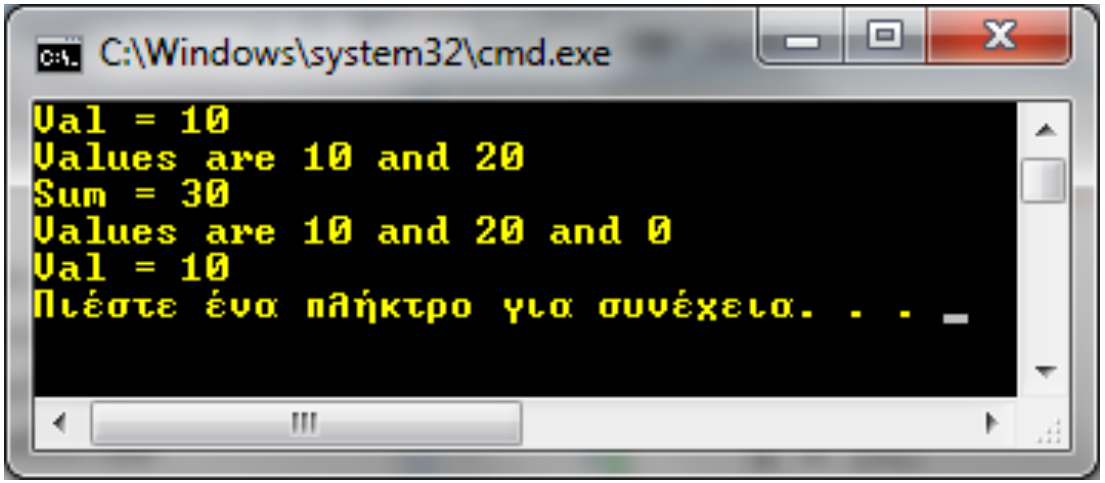
- Ο μεταγλωττιστής αντιστοιχίζει ένα-προς-ένα, από αριστερά προς τα δεξιά, τα ονόματα των μεταβλητών με τα προσδιοριστικά μετατροπής

- Αν τα προσδιοριστικά μετατροπής είναι περισσότερα από τις μεταβλητές, τότε για τα πρόσθετα προσδιοριστικά εμφανίζονται τυχαίες τιμές (σκουπίδια)

- Αντίστοιχα, αν τα προσδιοριστικά μετατροπής είναι λιγότερα από τις μεταβλητές, τότε δεν εμφανίζονται οι τιμές των πρόσθετων μεταβλητών

Παράδειγμα

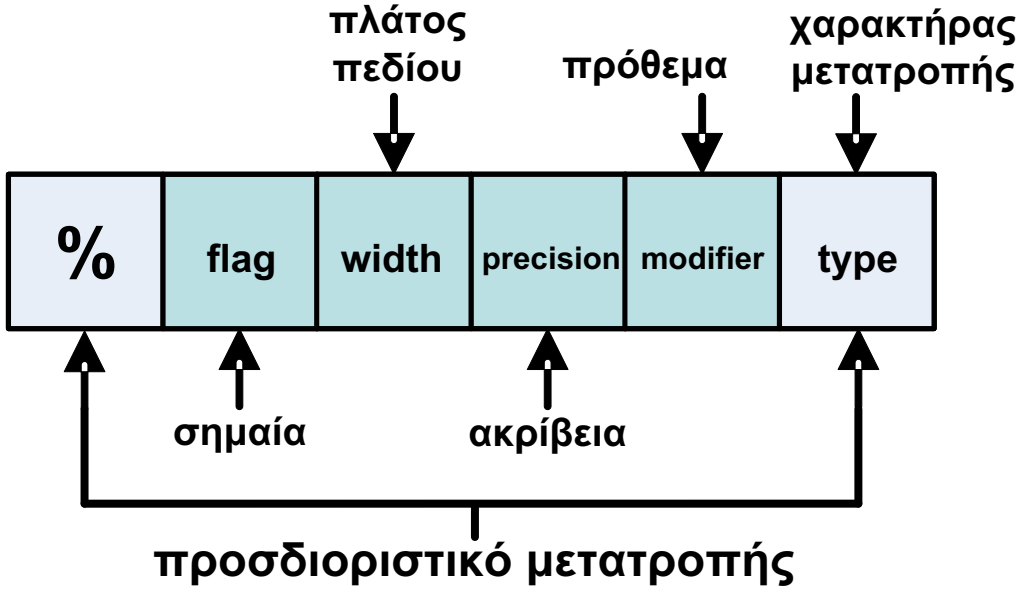
```c
#include <stdio.h>
int main()
{
	int a,b;
	a = 10;
	b = 20;
	printf("Val = %d\n",a);
	printf("Values are %d and %d\n",a,b);
	printf("Sum = %d\n",a+b);
	printf("Values are %d and %d and %d\n",a,b);
	printf("Val = %d\n",a,b);
	return 0;
}
```



Προαιρετικά Πεδία

- Ένα προσδιοριστικό μετατροπής, στην απλή μορφή του, αρχίζει με τον χαρακτήρα % και ακολουθείται από τον κατάλληλο χαρακτήρα μετατροπής

- Όμως, ανάμεσά τους, μπορεί να περιέχονται έως και τέσσερα επιπλέον προαιρετικά πεδία, όπως φαίνεται στο σχήμα



Ακρίβεια

- Όταν εμφανίζουμε την τιμή μίας πραγματικής μεταβλητής (τύπου `float` ή `double`) μπορούμε να καθορίσουμε πόσα ψηφία ακρίβειας θα εμφανιστούν στην οθόνη

- Εξ' ορισμού (by default) εμφανίζονται έξι δεκαδικά ψηφία

- Αν δεν επιθυμούμε να εμφανιστούν έξι ψηφία, τότε μετά τον χαρακτήρα `%` πρέπει να προσθέσουμε την τελεία (`.`) και
  - είτε έναν ακέραιο αριθμό που να δηλώνει το επιθυμητό πλήθος των δεκαδικών ψηφίων
  - είτε τον χαρακτήρα `*` και έναν ακέραιο στη λίστα των μεταβλητών που να δηλώνει το επιθυμητό πλήθος των δεκαδικών ψηφίων

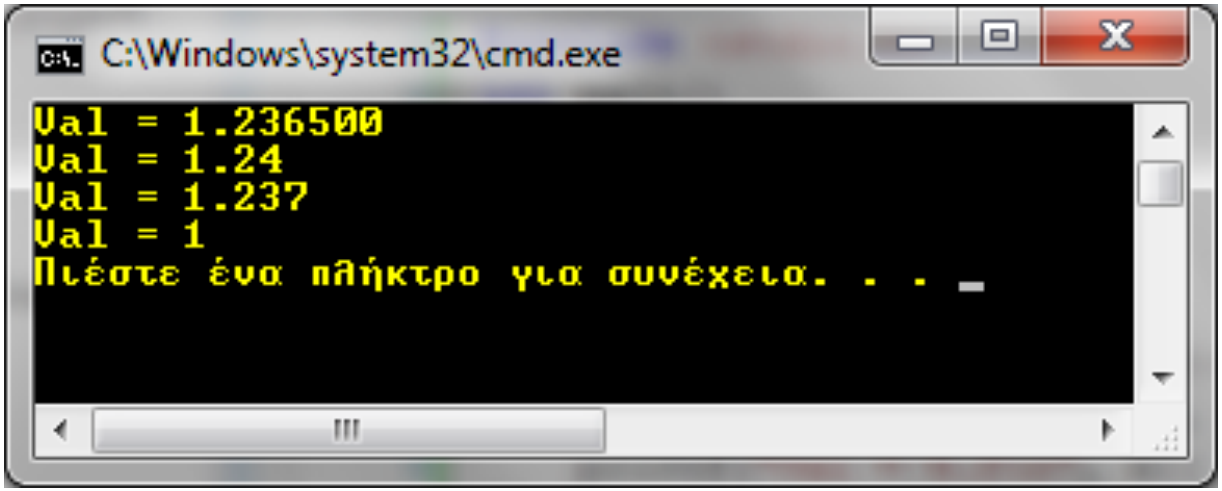
- Η τελική τιμή του πραγματικού αριθμού που εμφανίζεται με την `printf()` στρογγυλοποιείται προς τα πάνω ή προς τα κάτω, ανάλογα με το αν η τιμή του πρώτου ψηφίου που αποκόπτεται είναι μεγαλύτερη ή όχι από το 4, αντίστοιχα

- Αν δεν θέλουμε να εμφανιστούν δεκαδικά ψηφία, τότε προσθέτουμε μόνο την τελεία (`.`), χωρίς αυτή να ακολουθείται από κάποιον αριθμό

Παράδειγμα

```c
#include <stdio.h>
int main()
{
        float a = 1.2365;

        printf("Val = %f\n", a);
        printf("Val = %.2f\n", a);
        printf("Val = %.*f\n", 3, a);
        printf("Val = %.f\n", a);
        return 0;
}
```

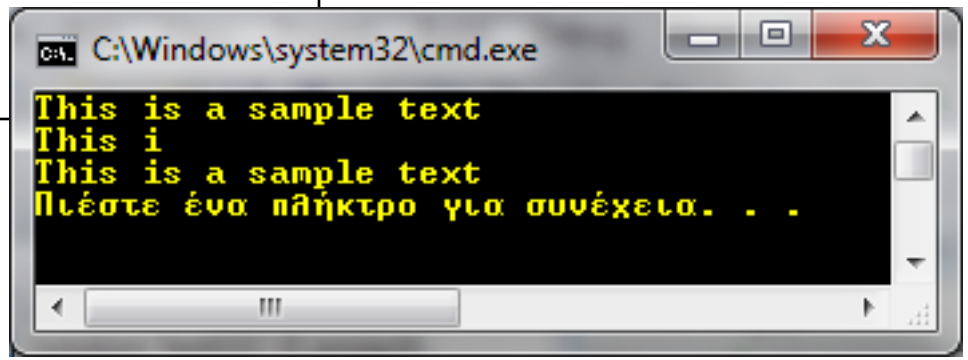


Εμφάνιση χαρακτήρων σε Αλφαριθμητικό

- Όταν θέλουμε να εμφανίσουμε ένα αλφαριθμητικό, δηλαδή μία ακολουθία χαρακτήρων, μπορούμε να καθορίσουμε πόσοι χαρακτήρες του θα εμφανιστούν, ακολουθώντας την ίδια τεχνική με προηγουμένως

- Αν ο αριθμός που θα δηλωθεί υπερβαίνει το πλήθος των χαρακτήρων, τότε εμφανίζονται όλοι οι χαρακτήρες του αλφαριθμητικού και δεν προστίθεται κανένας άλλος

```
#include <stdio.h>
int main()
{
      char msg[] = "This is a sample text";
      printf("%s\n",msg);
      printf("%.6s\n",msg);
      printf("%.30s\n",msg);
      return 0;
}
```



Πλάτος Πεδίου

- Όταν εμφανίζουμε την τιμή μίας ακέραιας ή πραγματικής μεταβλητής μπορούμε να καθορίσουμε το συνολικό πλήθος των χαρακτήρων που θα εμφανιστούν στην οθόνη, μαζί με τα ψηφία ακρίβειας και την υποδιαστολή

- Για να καθορίσουμε το συνολικό πλήθος εισάγουμε:
  - είτε έναν ακέραιο αριθμό αμέσως μετά από τον χαρακτήρα %, ο οποίος ονομάζεται πλάτος πεδίου
  - είτε τον χαρακτήρα * και έναν ακέραιο στη λίστα μεταβλητών, ο οποίος δηλώνει το πλάτος πεδίου

- Αν η τιμή της μεταβλητής χρειάζεται λιγότερους χαρακτήρες από το δηλωμένο πλάτος, τότε στην έξοδο προστίθενται κενοί χαρακτήρες από αριστερά προς τα δεξιά μέχρι να συμπληρωθεί το συνολικό πλήθος των χαρακτήρων

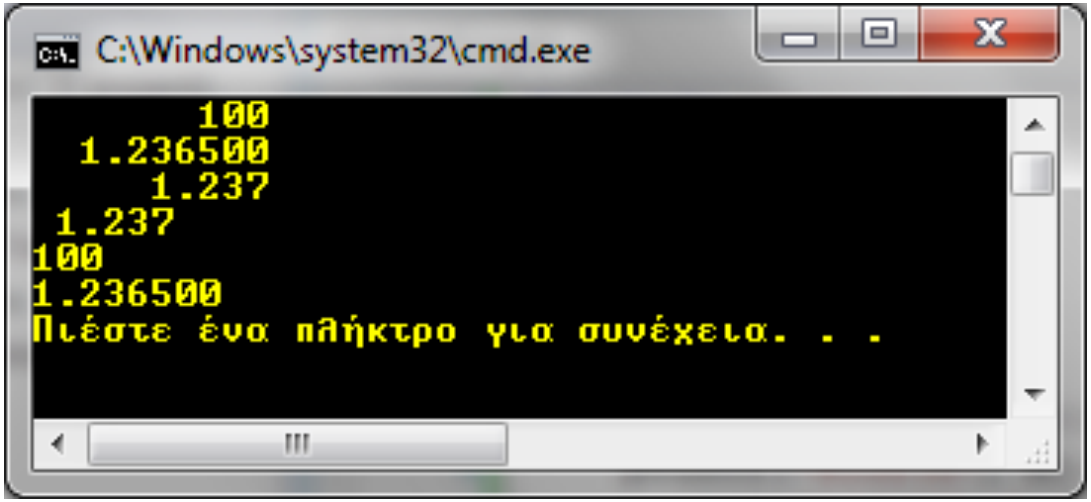
- Αν η τιμή της μεταβλητής χρειάζεται περισσότερους χαρακτήρες από το δηλωμένο πλάτος, τότε ο αριθμός αυτός δεν λαμβάνεται υπόψη και η τιμή της μεταβλητής εμφανίζεται με όσους χαρακτήρες απαιτείται

ΣΗΜΕΙΩΣΗ: Προφανώς, σε περίπτωση πραγματικού αριθμού, μετά το πλάτος του πεδίου μπορεί να καθοριστεί και η ακρίβεια του πραγματικού αριθμού (με χρήση της τελείας και ενός αριθμού που έπεται αυτής και δηλώνει το πλήθος των δεκαδικών ψηφίων, όπως δείξαμε προηγουμένως)

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int a = 100;
        float b = 1.2365;

        printf("%10d\n", a);
        printf("%10f\n", b);
        printf("%10.3f\n", b);
        printf("%*.3f\n", 6, b);
        printf("%2d\n", a);
        printf("%6f\n", b);
        return 0;
}
```

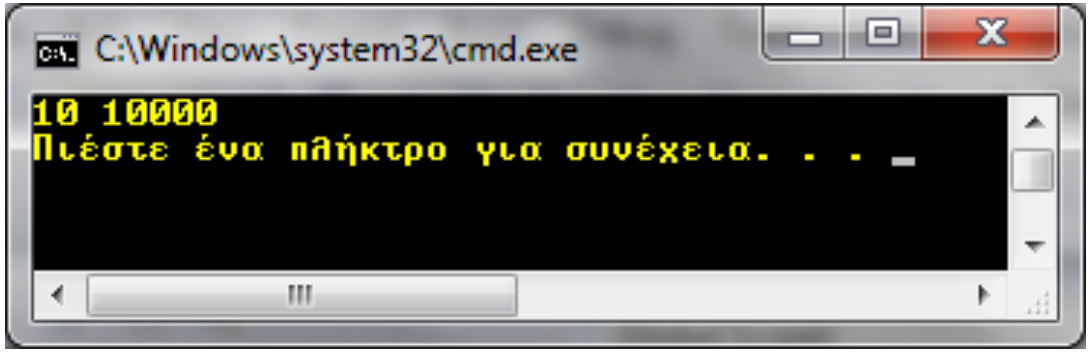


Πρόθεμα

- Για την εμφάνιση ενός **short** ακεραίου μπορεί να χρησιμοποιηθεί προαιρετικά το γράμμα **h**, ενώ για την εμφάνιση ενός **long** ακεραίου μπορεί να χρησιμοποιηθεί το γράμμα **l** ή **L**, όπως φαίνεται στο παρακάτω παράδειγμα

```
#include <stdio.h>
int main()
{
        short a = 10;
        long b = 10000;

        printf("%hd %ld\n", a, b);
        return 0;
}
```



Σημαίες

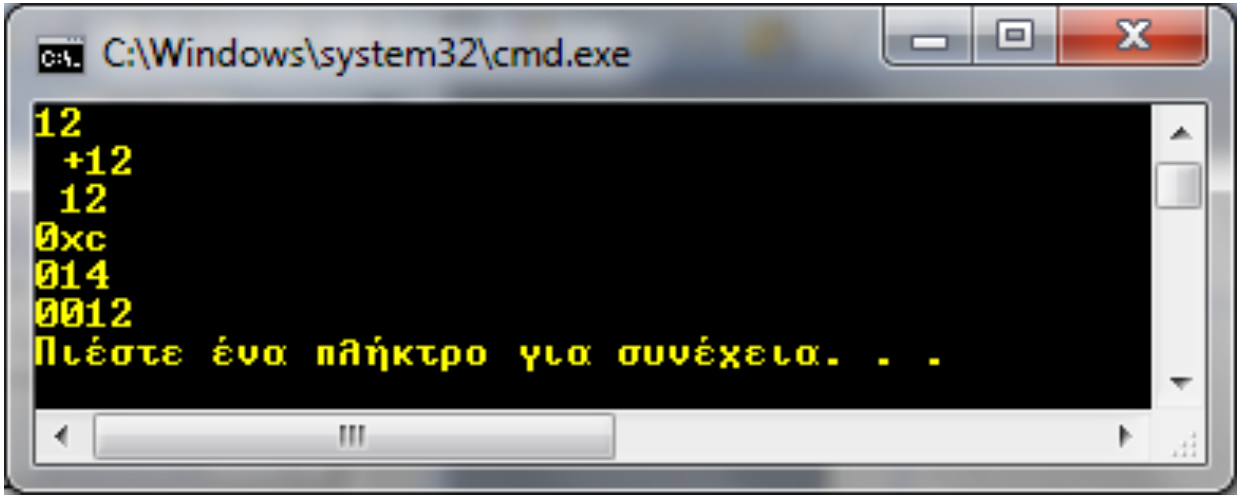
- Οι σημαίες χρησιμοποιούνται για περαιτέρω μορφοποίηση των εμφανιζόμενων τιμών, όπως φαίνεται στον παρακάτω πίνακα

Σημαία	Σημασία
-	Η έξοδος στο πεδίο πλάτους στοιχίζεται αριστερά (εξ' ορισμού η έξοδος στοιχίζεται δεξιά).
+	Προσθέτει το πρόσημο + μπροστά από τις θετικές τιμές.
κενός χαρακτήρας	Προσθέτει τον κενό χαρακτήρα μπροστά από τις θετικές τιμές.
#	Προσθέτει το 0 μπροστά από τις οκταδικές τιμές και το 0x ή το 0X μπροστά από δεκαεξαδικές τιμές.
0	Προσθέτει όσα μηδενικά χρειάζονται μπροστά από την εμφανιζόμενη τιμή, ώστε να καλυφθεί το πεδίο πλάτους.

Παράδειγμα

```c
#include <stdio.h>
int main()
{
	int a = 12;

	printf("%-4d\n", a);
	printf("%+4d\n", a);
	printf("% d\n", a);
	printf("%#0x\n", a);
	printf("%#o\n", a);
	printf("%04d\n", a);
	return 0;
}
```

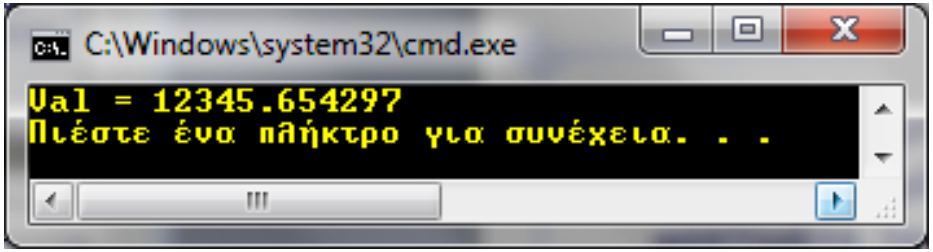


Παρατηρήσεις (Ι)

- Οταν χρησιμοποιείτε συχνά μία πραγματική μεταβλητή σε διάφορες εκφράσεις μέσα στο πρόγραμμα (π.χ. συγκρίσεις, πράξεις, ...), τότε να προτιμάτε τον τύπο **double** και όχι τον τύπο **float**, γιατί είναι πιθανό να μην γίνει η διαχείριση των δεκαδικών ψηφίων με τον τρόπο που θα αναμένατε

- Π.χ. το επόμενο πρόγραμμα μπορεί να μην εμφανίσει την τιμή **12345.65432**, αλλά μία τιμή παραπλήσια σε αυτή.

```
#include <stdio.h>
int main()
{
        float a;
        a = 12345.65432;
        printf("Val = %f\n",a);
        return 0;
}
```



Παρατηρήσεις (ΙΙ)

- Αν θέλετε η ακολουθία χαρακτήρων της **printf()**, για λόγους εμφάνισης, να εκτείνεται σε περισσότερες από μία γραμμές στον κώδικά σας, τότε να χρησιμοποιείτε τον χαρακτήρα της ανάστροφης κεκλιμένης **'\'** (backslash)

- Π.χ. ο κώδικας της παρακάτω **printf()** εκτείνεται σε δύο γραμμές

```
printf("This printf takes two lines. However, the \
message is shown in the same line ");
```

- Όμως, σαν αποτέλεσμα στην οθόνη, όλοι οι χαρακτήρες του μηνύματος θα εμφανίζονται στην ίδια γραμμή

- Λόγω της ειδικής σημασίας του χαρακτήρα **%**, για την εμφάνιση του χαρακτήρα **'%'** πρέπει να γραφούν δύο χαρακτήρες **%**

- Π.χ. η επόμενη **printf()** εμφανίζει το μήνυμα **100%** στην οθόνη

```
printf("%d%%\n",100);
```

Μετατροπή Τύπου (type cast)

- Υπάρχουν περιπτώσεις όπου ένας τύπος δεδομένων πρέπει να μετατραπεί προσωρινά σε κάποιον άλλο τύπο δεδομένων

- Π.χ. είναι πιθανό σε ένα σημείο του προγράμματος μία ακέραια μεταβλητή που έχει δηλωθεί σαν `int` να πρέπει να μετατραπεί προσωρινά σε μία πραγματική τύπου `float`, ή και το αντίστροφο

- Η γενική μορφή μίας τέτοιας μετατροπής είναι:

```
(τύπος_δεδομένων) (παράσταση)
```

- Π.χ. αν η μεταβλητή `a` έχει δηλωθεί:

```
int a;
```

τότε η έκφραση:

```
(float)a;
```

προσαρμόζει προσωρινά τον τύπο της `a` από `int` σε `float`

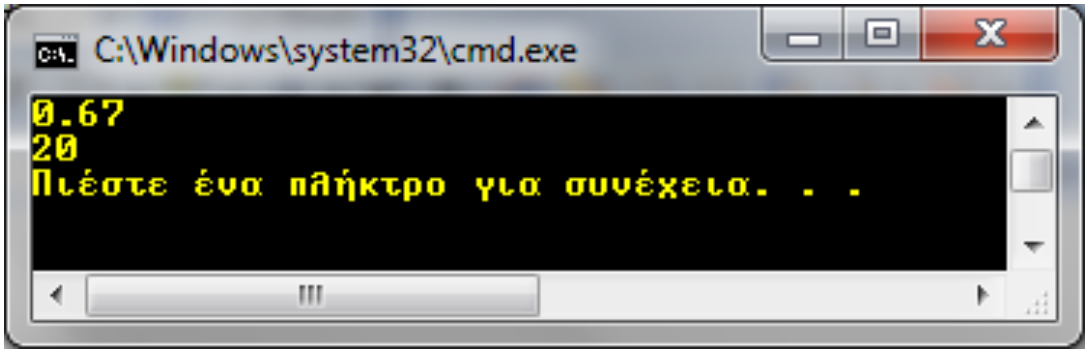
- Λέγοντας προσωρινά, εννοούμε ότι στη συνέχεια του προγράμματος ο τύπος της μεταβλητής `a` συνεχίζει να είναι `int`

Παράδειγμα

```
#include <stdio.h>
int main()
{
        int i = 20,j = 30;
        float k;

        k = (float)i/j;

        printf("%.2f\n",k);
        printf("%d\n",i);
        return 0;
}
```



- Η έκφραση `(float)i` προσαρμόζει προσωρινά τον τύπο της μεταβλητής `i` από `int` σε `float`, έτσι ώστε το αποτέλεσμα της διαίρεσης να είναι πραγματικός αριθμός
- Αν γράφαμε `k = i/j`, τότε η τιμή του `k` θα ισούνταν με το αποτέλεσμα της ακέραιας διαίρεσης `20/30`, άρα υπό μορφή `float` με ακρίβεια 2 δεκαδικών ψηφίων (λόγω της σύγκεκριμένης `printf()`) η τιμή του `k` θα ήταν `0.00`

Η συνάρτηση `scanf()`

- Η συνάρτηση **`scanf()`** χρησιμοποιείται για την είσοδο δεδομένων από ένα αρχείο εισόδου, το οποίο ονομάζεται **`stdin`** *(standard input stream)* και εξ' ορισμού συνδέεται με το πληκτρολόγιο

- Η **`scanf()`** δέχεται μία μεταβλητή λίστα παραμέτρων, παρόμοια με την **`printf()`**, δηλαδή:

  - Η πρώτη παράμετρος είναι ένα **αλφαριθμητικό μορφοποίησης** *(format string)*, το οποίο, συνήθως, περιέχει μόνο απλά προσδιοριστικά μετατροπής (π.χ. **`%d`** για μεταβλητές τύπου **`int`**, **`%f`** για μεταβλητές τύπου **`float`** κτλ...)

  - Οι επόμενες προαιρετικές παράμετροι είναι οι **διευθύνσεις μνήμης** των μεταβλητών στις οποίες θα εκχωρηθούν τα δεδομένα που θα εισάγει ο χρήστης από το πληκτρολόγιο

Κάθε προσδιοριστικό μετατροπής πρέπει να αντιστοιχεί σε μία διεύθυνση μεταβλητής και η αντιστοίχηση γίνεται ένα προς ένα

Για μεταβλητές τύπου **`double`**, χρησιμοποιείται το προσδιοριστικό μετατροπής **`%lf`** και όχι το **`%f`**, το οποίο χρησιμοποιείται μόνο για μεταβλητές τύπου **`float`**

Παράδειγμα 1

- Π.χ.

```
int i;
scanf("%d", &i);
```

- Ο χαρακτήρας &, που μπαίνει πριν από το όνομα της μεταβλητής, ονομάζεται τελεστής διεύθυνσης και χρησιμοποιείται για να αποθηκευτεί ο αριθμός που θα εισάγει ο χρήστης στη διεύθυνση μνήμης της μεταβλητής i

- Περισσότερες λεπτομέρειες για τη σημασία του τελεστή διεύθυνσης & θα δούμε αναλυτικά στους "δείκτες της C"

Παράδειγμα 2 (Ι)

- Μπορούμε να χρησιμοποιήσουμε τη συνάρτηση **scanf()** για να διαβάσουμε περισσότερες από μία τιμές από το πληκτρολόγιο και να τις αποθηκεύσουμε ως τιμές σε κάποιες μεταβλητές του προγράμματος
- Π.χ.

```
int i;
float j;
scanf("%d%f", &i, &j);
```

- Η πρώτη παράμετρος της **scanf()** είναι το αλφαριθμητικό μορφοποίησης **%d%f**, ενώ οι επόμενες παράμετροι είναι οι διευθύνσεις μνήμης των μεταβλητών **i** και **j** αντίστοιχα
  - Το **%d** αντιστοιχεί στη διεύθυνση της μεταβλητής **i**
  - Το **%f** αντιστοιχεί στη διεύθυνση της μεταβλητής **j**
  - Δηλαδή η αντιστοίχιση γίνεται ένα προς ένα και από αριστερά προς τα δεξιά
- Για την είσοδο των δεδομένων χρησιμοποιείται συνήθως το «κενό διάστημα» (space) μεταξύ των διαφορετικών τιμών που εισάγονται, δεδομένου ότι κατά το διάβασμα αριθμητικών τιμών, η **scanf()** αγνοεί όλα τα λευκά διαστήματα (π.χ. κενά διαστήματα, tab, χαρακτήρα νέας γραμμής) που μπορεί να υπάρχουν πριν από κάθε αριθμητική τιμή

Παράδειγμα 2 (ΙΙ)

- Στο προηγούμενο παράδειγμα, αν ο χρήστης εισάγει π.χ. τις τιμές `10` και `4.65`, αυτές θα πρέπει να απέχουν μεταξύ τους ένα ή περισσότερα κενά
- Για να ληφθούν από τη συνάρτηση `scanf()` πρέπει μετά ο χρήστης να πατήσει Enter
- Τότε, η τιμή `10` αποθηκεύεται στη μεταβλητή `i` και η τιμή `4.65` στη μεταβλητή `j`, αντίστοιχα

Παραδείγματα 3 & 4

- Στο επόμενο παράδειγμα, η `scanf()` διαβάζει έναν χαρακτήρα και τον αποθηκεύει στη μεταβλητή `ch`

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

- Στο επόμενο παράδειγμα, η `scanf()` διαβάζει ένα αλφαριθμητικό και το αποθηκεύει στον πίνακα χαρακτήρων `str`
- Παρατηρήστε, ότι πριν από τη μεταβλητή `str` δεν χρησιμοποιείται ο τελεστής `&`, γιατί - όπως θα δούμε στο κεφάλαιο των "Πινάκων της C"- το όνομα ενός πίνακα ισοδυναμεί με τη διεύθυνση του πρώτου στοιχείου του

```
char str[100];
scanf("%s", str);
```

- Αν ο χρήστης εισάγει το αλφαριθμητικό `sample` και πατήσει Enter, τότε οι χαρακτήρες του θα αποθηκευτούν στις αντίστοιχες θέσεις του πίνακα `str`
- Δηλαδή, η τιμή του `str[0]` θα γίνει `'s'`, του `str[1]` θα γίνει `'a'`, του `str[2]` θα γίνει `'m'`, κ.ο.κ.
- Το παράδειγμα αυτό θα το κατανοήσετε καλύτερα αργότερα, αφού θα μιλήσουμε για πίνακες, χαρακτήρες και αλφαριθμητικά

Παρατηρήσεις

Να θυμάστε ότι η `scanf()` απαιτεί τον τελεστή διεύθυνσης `&` πριν από το όνομα κάθε αριθμητικής μεταβλητής (π.χ. `int`, `double`, `char`, `float`, …)

Αν τον ξεχάσετε, το πρόγραμμά σας δεν θα εκτελεστεί σωστά

Αντίθετα, όταν ο τύπος της μεταβλητής είναι δείκτης, ο τελεστής διεύθυνσης δεν χρειάζεται

Για να διαβάσετε με τη `scanf()` ένα αλφαριθμητικό που μπορεί να αποτελείται από πολλές λέξεις (π.χ. `"Text with multiple words"`), πρέπει να χρησιμοποιήσετε μία πιο σύνθετη μορφή της,

π.χ.: `scanf("%[^\n]",str);`

γιατί η `scanf()` εξ' ορισμού σταματάει το διάβασμα του αλφαριθμητικού όταν συναντήσει έναν κενό χαρακτήρα

Τί επιστρέφει η συνάρτηση `scanf()` ???

- Η συνάρτηση `scanf()` επιστρέφει έναν ακέραιο αριθμό που δηλώνει πόσα από τα δεδομένα εισόδου διαβάστηκαν και εκχωρήθηκαν στις μεταβλητές του προγράμματος, ενώ οι τιμές που δεν διαβάστηκαν παραμένουν στο `stdin`

- Π.χ. στο παράδειγμα:

```
int i;
scanf("%d", &i);
```

αν ο χρήστης εισάγει έναν ακέραιο, η συνάρτηση `scanf()` επιστρέφει την τιμή `1`

- Ενώ στο παράδειγμα:

```
int i;
float j;
scanf("%d%f", &i, &j);
```

αν ο χρήστης εισάγει έναν ακέραιο και έναν πραγματικό αριθμό, η συνάρτηση `scanf()` επιστρέφει την τιμή `2`

Παρατηρήσεις

- Η συνάρτηση **scanf()** δεν είναι μία ασφαλής συνάρτηση και πρέπει να τη χρησιμοποιείτε με μεγάλη προσοχή
  - Σε επαγγελματικές εφαρμογές πρέπει να ελέγχετε την τιμή επιστροφής της

```
#include <stdio.h>
int main()
{
	int num;
	printf("Enter number: ");
	while(scanf("%d", &num) != 1)
	{
		printf("Enter number: ");
		while(getchar() != '\n'); /* Καθαρισμός μνήμης
πληκτρολογίου. */
	}
	printf("Inserted value: %d\n", num);
	return 0;
}
```

  - Σχεδόν σε όλα τα προγράμματα που θα υλοποιήσουμε, όταν χρησιμοποιούμε τη συνάρτηση **scanf()**, για λόγους απλότητας, δεν θα ελέγχουμε την τιμή επιστροφής της και θα θεωρούμε ότι οι τιμές που εισάγει ο χρήστης θα είναι σε συμφωνία με τα προσδιοριστικά μετατροπής

Παρεμβολή απλών χαρακτήρων στη `scanf()` (I)

- Στην πιο συνηθισμένη χρήση της, το αλφαριθμητικό μορφοποίησης της `scanf()` δεν περιέχει απλούς χαρακτήρες παρά μόνο τα προσδιοριστικά μετατροπής (π.χ. `%d`, `%f`,…)

- Ωστόσο, αν παρεμβληθούν κάποιοι απλοί χαρακτήρες, τότε πρέπει οι αντίστοιχοι χαρακτήρες να εισαχθούν και από το πληκτρολόγιο

- Π.χ. στην επόμενη `scanf()` παρεμβάλλεται ο χαρακτήρας κόμμα (,)

```
#include <stdio.h>
int main()
{
        int a, b;
        scanf("%d , %d", &a, &b);
        printf("%d %d\n", a, b);
        return 0;
}
```

- Για να λειτουργήσει σωστά αυτό το πρόγραμμα πρέπει οι ακέραιες τιμές που θα εισάγει ο χρήστης να διαχωρίζονται μεταξύ τους με κόμμα (,)

Παρεμβολή απλών χαρακτήρων στη `scanf()` (II)

- Αν αντί για κόμμα (όπως είδαμε στο προηγούμενο παράδειγμα), μεταξύ των ειδικών χαρακτήρων μέσα στην `scanf()` χρησιμοποιούνταν ο χαρακτήρας `'m'`, (όπως φαίνεται παρακάτω), θα έπρεπε να εισαχθεί ο χαρακτήρας `'m'` μεταξύ των τιμών που θα έδινε ο χρήστης από το πληκτρολόγιο

```
#include <stdio.h>
int main()
{
        int a, b;
        scanf("%dm%d", &a, &b);
        printf("%d %d\n", a, b);
        return 0;
}
```

- Αν δηλαδή ο χρήστης επιθυμούσε να εισάγει τις τιμές `12` και `43`, θα έπρεπε να πληκτρολογήσει: `12m43`

- Γενικά, προτείνουμε <u>να μην παρεμβάλλεται</u> κάποιος χαρακτήρας μεταξύ των προσδιοριστικών μετατροπής, έτσι ώστε να μην χρειάζεται κάποια επιπλέον εισαγωγή χαρακτήρων από τον χρήστη (σκεφτείτε επίσης ότι ο τελικός χρήστης του προγράμματος δεν είναι απαραίτητα και ο δημιουργός του προγράμματος)

Καθαρισμός Μνήμης

- Η συνάρτηση `scanf()` δεν θα λειτουργήσει σωστά, αν ο χρήστης δεν εισάγει τα δεδομένα σύμφωνα με την ακολουθία των προσδιοριστικών μετατροπής που ορίζονται σε αυτήν

- Στο επόμενο παράδειγμα η `scanf()` αναμένει μία ακέραια και μία πραγματική τιμή

```
#include <stdio.h>
int main()
{
        int i;
        double j;

        printf("Enter numbers: ");

        scanf("%d", &i);
        scanf("%lf", &j);

        printf("Num1 = %d, Num2 = %f\n", i, j);
        return 0;
}
```

- Τι θα συμβεί αν ο χρήστης εισαγάγει – έστω κατά λάθος – ως ακέραια τιμή την τιμή `5.65` ?

Παράδειγμα (Ι)

- Η συνάρτηση **scanf()** δεν διαβάζει τον χαρακτήρα νέας γραμμής που πληκτρολογεί ο χρήστης στο τέλος της εισαγωγής δεδομένων
- Αυτός ο χαρακτήρας θα διαβαστεί στην επόμενη κλήση της **scanf()**
- Αν όμως, στην επόμενη κλήση της, η **scanf()** χρησιμοποιείται για το διάβασμα χαρακτήρων, τότε θα διαβαστεί μόνο αυτός ο χαρακτήρας (της νέας γραμμής) και οι υπόλοιποι θα αγνοηθούν
- Π.χ. το επόμενο πρόγραμμα δεν θα εκτελεστεί σωστά

```
#include <stdio.h>
int main()
{
        char ch;
        int i;

        printf("Enter number: ");
        scanf("%d",&i);

        printf("Enter character: ");
        scanf("%c",&ch);

        printf("Int = %d and Char = %c\n",i,ch);
        return 0;
}
```

Παράδειγμα (ΙΙ)

- Αν αντιστρέψουμε στο προηγούμενο παράδειγμα τη σειρά του διαβάσματος, τότε το πρόγραμμα θα εκτελεστεί σωστά, αφού – σύμφωνα με προηγούμενη παρατήρηση – ο χαρακτήρας νέας γραμμής που υπάρχει πριν από την αριθμητική τιμή αγνοείται

```
#include <stdio.h>
int main()
{
        char ch;
        int i;

        printf("Enter character: ");
        scanf("%c",&ch);

        printf("Enter number: ");
        scanf("%d",&i);

        printf("Int = %d and Char = %c\n",i,ch);
        return 0;
}
```

Τρόποι Καθαρισμού Μνήμης

- Ένας τρόπος για να αδειάσουμε τη μνήμη του πληκτρολογίου από τα δεδομένα που έχουν παραμείνει είναι με τη χρήση της συνάρτησης `getchar()` χρησιμοποιώντας τον παρακάτω επαναληπτικό βρόχο

```
while(getchar() != '\n');
```

- Ωστόσο, πολλοί προγραμματιστές χρησιμοποιούν τη συνάρτηση `fflush()`

```
fflush(stdin);
```

- Προσέξτε όμως, σύμφωνα με το πρότυπο της C, η συμπεριφορά της `fflush()` όταν χρησιμοποιείται με όρισμα το `stdin` είναι ακαθόριστη