Δομημένος προγραμματισμός

Συναρτήσεις εισόδου/εξόδου, τελεστές

Περίγραμμα ύλης που θα καλυφθεί

- Η συνάρτηση scanf()
- Οι συναρτήσεις getchar(), putchar()
- Συμβολοσειρές
- Οι συναρτήσεις puts() & gets()
- Τελεστές
  - Τελεστές συσχετισμού
  - Λογικοί τελεστές
  - Τελεστές επιπέδου bit
  - Οι τελεστές αύξησης/μείωσης
  - Συνδυαστικοί τελεστές
- Προτεραιότητες τελεστών

Η συνάρτηση scanf() – είσοδος δεδομένων

- Διαβάζει από το πληκτρολόγιο (ροή *stdin*) μορφοποιημένες τιμές μεταβλητών.

scanf (ΣΕΙΡΑ_ΕΛΕΓΧΟΥ, δείκτης_μεταβλητής-1, δείκτης_μεταβλητής-2,... , δείκτης_μεταβλητής-ν)

- Στη πιο γνωστή της μορφή χρησιμοποιείται για να καταχωρίσουμε δεδομένα σε μεταβλητές από το πληκτρολόγιο.
- Οι προσδιοριστές στη *ΣΕΙΡΑ_ΕΛΕΓΧΟΥ* αντιστοιχούν σε διευθύνσεις μεταβλητών (δείκτες).
- Η συνάρτηση *scanf()* μοιάζει πολύ με την *printf()* με μια μεγάλη διαφορά:

δεν αναφέρεται σε ονομασίες μεταβλητών, αλλά σε δείκτες μεταβλητών.

- Άλλη διαφορά:

η ΣΕΙΡΑ_ΕΛΕΓΧΟΥ στη scanf() περιέχει κυρίως προσδιοριστές και όχι επεξηγηματικό κείμενο. Χαρακτήρες ελέγχου μπορείτε να βάλετε.

- Για μεταβλητές τύπου double, ο προσδιοριστής είναι: **%lf**

Λίγα για τους δείκτες

Δήλωση στη C

```
int   NUM = 5 ;
```

Η διεύθυνση του NUM (&NUM)

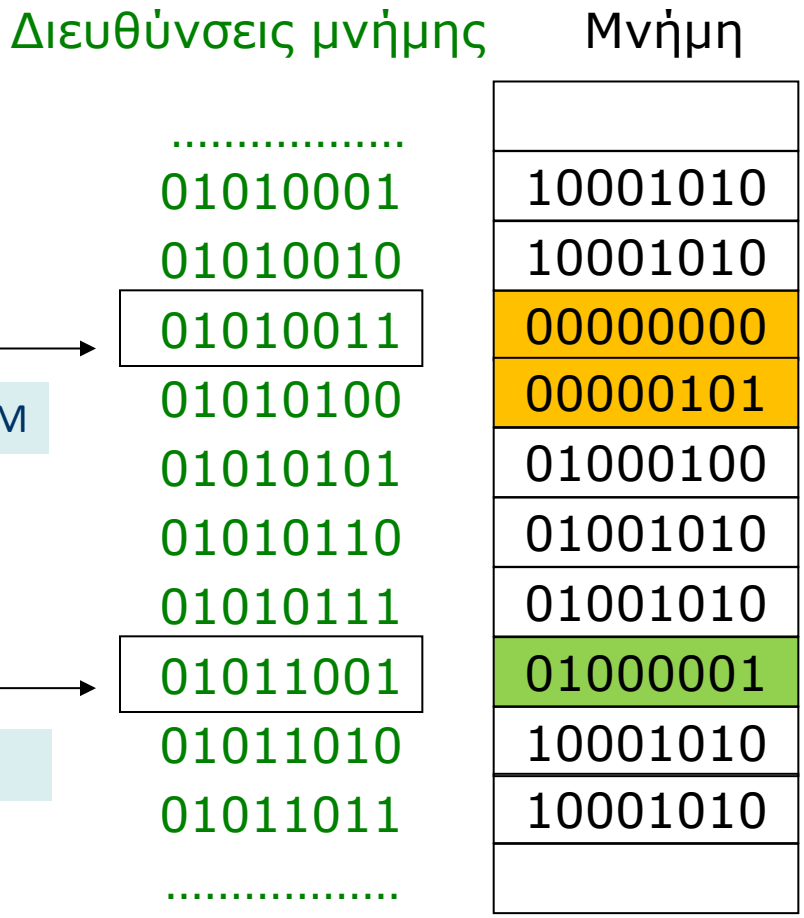
Δείκτης της NUM

Δήλωση στη C

```
char LET = 'A' ;
```

Η διεύθυνση του LET (&LET)

Δείκτης της LET



Η έκφραση &NUM εννοεί την διεύθυνση της μεταβλητής NUM ($=01010011_2=83_{10}$).

Η scanf() λοιπόν αντίθετα με την printf() χρησιμοποιεί δείκτες μεταβλητών.

Παραδείγματα εντολών με τη scanf()

```
/* Διαβάζει από το πληκτρολόγιο την ακέραια μεταβλητή x */
printf("Enter first number: ");
scanf("%d", &x);
```

Μετά την είσοδο δεδομένων, πατάμε <ENTER>

```
/* Διαβάζει από το πληκτρολόγιο μια μεταβλητή κινητής υποδιαστολής (fnum)
   και μια  ακέραια μεταβλητή  (inum) */
printf ("Give me 1 float and 1 integer number:");
scanf("%f %d", &fnum, &inum);
```

Με 1 κενό μεταξύ των τιμών που δίνουμε

- Καλό είναι κάθε κλήση της *scanf()* να συνοδεύεται από μια κλήση της *printf()* - η printf() να μπαίνει πριν την scanf() ώστε ο χρήστης γνωρίζει τι δεδομένο πρέπει να πληκτρολογήσει.
- Όταν εισάγονται με μια κλήση της *scanf()* πάνω από 2 δεδομένα αυτά θα πρέπει να έχουν μεταξύ τους τουλάχιστον 1 κενό.
- Δίνουμε *<enter>* μετά την πληκτρολόγηση των δεδομένων.
- Σύνηθες λάθος: *scanf("%d",x)* αντί για *scanf("%d", &x)*.

Τι επιστρέφει η scanf()

- η scanf() έχει κάποια τιμή επιστροφής που ανάλογα είναι:
  - ο αριθμός των όρων που διάβασε με επιτυχία,
  - το μηδέν (0) αν δεν μπορέσει να διαβάσει με επιτυχία ή
  - η τιμή του EOF (End Of File) όταν φτάσει απροσδόκητα στο τέλος αρχείου.
- Η κλήση:

n = scanf("%d%f", &i, &x);

όταν η γραμμή εισαγωγής (πληκτρολόγιο) είναι *25 54.32*

και οι δηλώσεις των μεταβλητών είναι:

int i, n; float x;

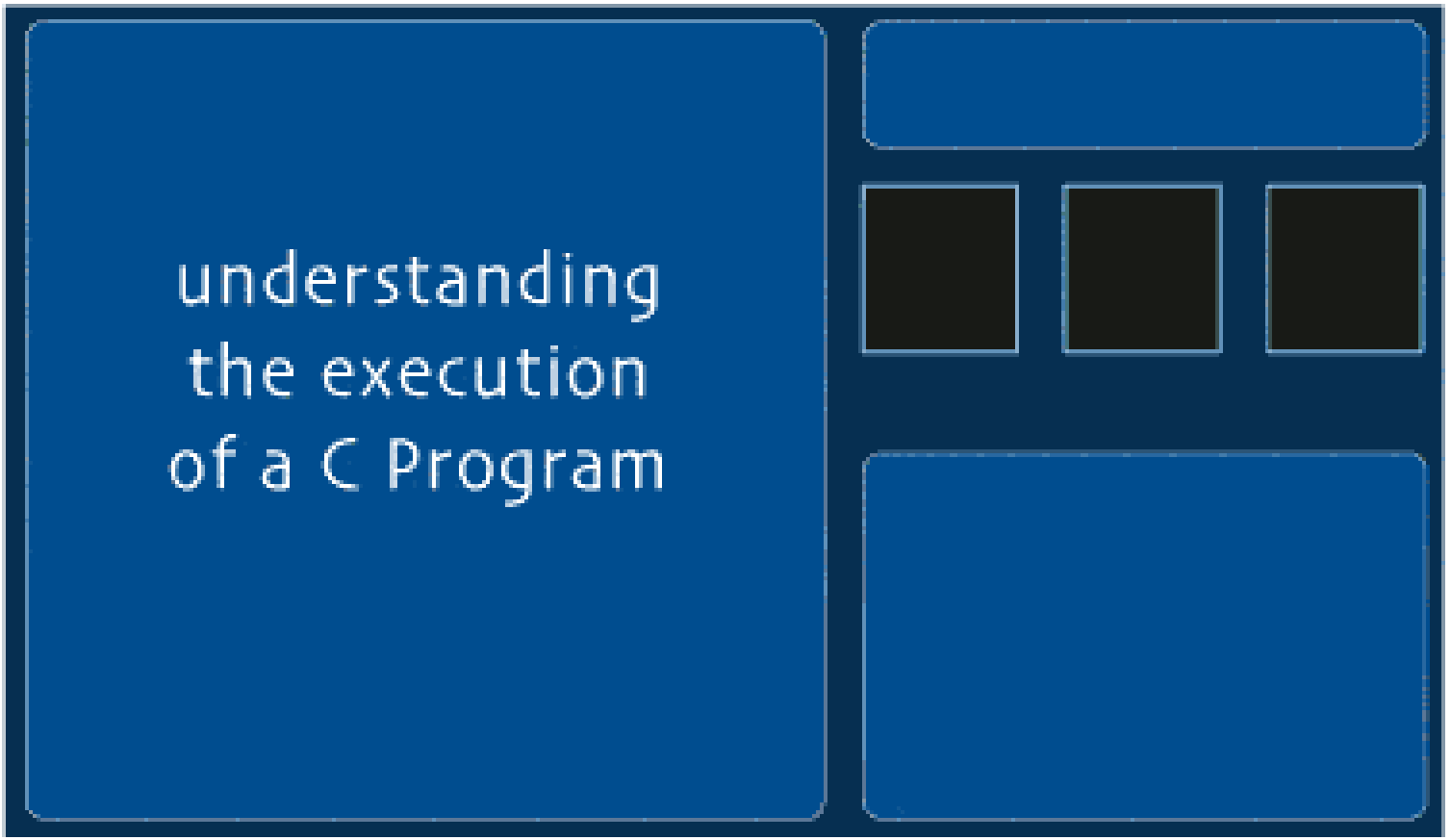
θα έχει σαν αποτέλεσμα:

η μεταβλητή *n* να πάρει την τιμή *2* η μεταβλητή *i* να πάρει την τιμή *25* και η μεταβλητή *x* να πάρει την τιμή *54.32*.

Παραδείγματα προγραμμάτων με την scanf()

```
#include <stdio.h>
#include <stdlib.h>
main()
{
      int index;
      float num;
      printf (" Give me the integer index: ");
      scanf("%d", &index);
      printf (" Give me the float num: ");
      scanf("%f", &num);
      printf("The value of index is %d\n",index);
      printf("The value of num is %f\n",num);
      num = 27.567;
      printf("The new value of num is %f\n",num);
      system("PAUSE");
  }
```

```
#include <stdio.h>
 #define FPA 0.23
main()
{
      float timi_DVD, poso;
      int posotita;
      printf ("\nTimi DVD:");
      scanf ("%f", &timi_DVD);
      printf ("Dwste posotita:\n");
      scanf ("%d", &posotita);
      poso=timi_DVD*posotita;
      printf ("\n\n");
      printf("Poso plirwmis: %5.2f\n", poso);
      poso = poso + FPA * poso;
      printf ("\n\n");
      printf("Poso plirwmis: %6.3f\n", poso);
      getchar(); getchar();
}
```


understanding
the execution
of a C Program
©2004 HowStuffWorks

Οι συνάρτησεις getchar(), putchar()

- Η *getchar()* διαβάζει έναν χαρακτήρα από το πληκτρολόγιο με το πλήκτρο <*enter*> (περιμένει τη πληκτρολόγηση του <enter>).
- Η συνάρτηση getchar() δεν δέχεται παραμέτρους και υπάρχει στη βιβλιοθήκη <*stdio.h*>.
- Όταν εκτελείται, επιστρέφει τον χαρακτήρα που διάβασε:

```
char alpha;
.....
alpha = getchar();
```
⟷ scanf("%c", &alpha);

- Η getchar() είναι μια παραλλαγή της συνάρτησης *getche()* -διαβάζει από το πληκτρολόγιο έναν χαρακτήρα και τον εμφανίζει στη οθόνη.
  - Η getche() υπάρχει στην <*conio.h*> και δεν δέχεται παραμέτρους.
- Η *putchar()* εμφανίζει στην οθόνη ένα μόνο χαρακτήρα:

```
putchar(alpha); //εμφανίζει στην οθόνη το περιεχόμενο της alpha
putchar(getchar()); //εμφανίζει τον χαρακτήρα που πληκτρολογήθηκε
```

- Η putchar() υπάρχει στην βιβλιοθήκη <*stdio.h*>.

Συμβολοσειρές

```
printf("What's your name:"); gets(name);
printf("Hello "); puts(name);
```

```
#include <stdio.h>
main()
{
   char aaa[20];
   char bbb[10] = "Ritsa";
   printf("Hello my name is "); puts(bbb);
   printf("What's your name:"); gets(aaa);
   printf("\n");
   printf("%s you 're welcome",aaa);
}
```

Στη C για την αποθήκευση και χειρισμό των συμβολοσειρών χρησιμοποιείται την δομή του πίνακα τύπου char. Ο τελευταίος χαρακτήρας του πίνακα είναι ο \0 (μηδενικός χαρακτήρας).

```
char minima1[20] = "hello";
char minima2[ ] = {'h', 'e','l','l','o','\0'};
……..
gets (minima1);
scanf("%s",minima2);
……..
puts(minima1);
printf("To minina einai:%s\n", minima);
```

Χωρίς &

Οι συναρτήσεις puts() & gets()

- Η *puts()* εμφανίζει στην οθόνη μια σειρά χαρακτήρων (συμβολοσειρά):

puts(<μεταβλητή_συμβολοσειράς>);

- Η *puts()* μετά την εμφάνιση της συμβολοσειράς, τυπώνει τον χαρακτήρα *\n*, δηλαδή επιστρέφει στην επόμενη σειρά.
- Ο χαρακτήρας *\0* που υποδεικνύει το τέλος της συμβολοσειράς (μηδενικός χαρακτήρας) χρησιμοποιείται από την puts() και καλό θα είναι να προσέχετε, ώστε η συμβολοσειρά που έχετε δώσει για εκτύπωση να περιέχει στο τέλος τον χαρακτήρα *\0*.
- Η *gets()* διαβάζει από το πληκτρολόγιο (είσοδος -stdin) μια σειρά χαρακτήρων και την αναθέτει σε μια μεταβλητή συμβολοσειράς:

gets(<μεταβλητή_συμβολοσειράς>);

- Η *gets()* διαβάζει συνεχώς από το πληκτρολόγιο μέχρι να συναντήσει τον χαρακτήρα *\n* (πράγμα που συμβαίνει όταν πατήσουμε το *<enter>*). Ο χαρακτήρας *\0* μπαίνει αυτόματα στο τέλος της συμβολοσειράς που πληκτρολογείτε.

Οι τελεστές συσχετισμού

- Πρόκειται για τελεστές που τους χρησιμοποιούμε για την δημιουργία απλών λογικών εκφράσεων συσχετισμού (σύγκρισης).

<	*Μικρότερο από*
>	*Μεγαλύτερο από*
<=	*Μικρότερο από ή ίσο*
>=	*Μεγαλύτερο από ή ίσο*
==	*Λογικό ίσο*
!=	*Διάφορο (όχι ίσο)*

- Το αποτέλεσμα μιας λογικής έκφρασης είναι:

  είτε *1* (αληθές), είτε *0* (ψευδές).

- Οι τελεστές σύγκρισης χρησιμοποιούνται σαν συνθήκες, κυρίως στις εντολές ελέγχου και επανάληψης.

Παρατηρήσεις στους τελεστές συσχετισμού

- Αν έχω *x=2*; και *y=3*; τότε η λογική έκφραση

  (x > y)

  επιστρέφει *0* (ψευδής έκφραση).

- Προσοχή: ο τελεστής για το *λογικό ίσο* που είναι ένα διπλό ίσον (==).

- Στην C η προτεραιότητα των τελεστών συσχετισμού είναι μικρότερη από αυτή των αριθμητικών τελεστών, ενώ είναι μεγαλύτερη από αυτή του τελεστή καταχώρησης.

  Άρα η μεταβλητή *i* στη εκτέλεση του κώδικα που ακολουθεί,

  ```
  j = k =13;
  i = j == k;
  ```

  θα πάρει την τιμή *1*, διότι ο τελεστής της ισότητας (==) έχει μεγαλύτερη προτεραιότητα από τον τελεστή καταχώρησης (ένα μονό ίσο =). Καθώς λοιπόν οι μεταβλητές *j* και *k* είναι ίσες το αποτέλεσμα της σύγκρισης αυτών (*j == k*) θα είναι αληθές, δηλαδή *1*.

Λογικοί τελεστές

- Πρόκειται για τελεστές που μας επιτρέπουν να συνδυάσουμε απλές εκφράσεις συσχετισμού και να δημιουργήσουμε πιο πολύπλοκες λογικές προτάσεις.

&&	Λογικό "**και**" (AND)
||	Λογικό διαζευκτικό "**ή**" (είτε - OR)
!	Λογική άρνηση (NOT)

- Το αποτέλεσμα μιας πρότασης με λογικούς τελεστές είναι

  είτε *μηδέν* (0 -ψευδές), είτε *όχι μηδέν* (αληθές).

- Με τους τελεστές && και ||, απαιτείται η χρήση *2* όρων, ενώ με τον τελεστή *!* ενός (μοναδιαίος).

$$x > 10 \;\&\&\; x < 20$$

$$(x > 10 \;\&\&\; x < 20) \;||\; x > 50$$

Παράδειγμα με λογικούς τελεστές

- Η προτεραιότητα των τελεστών συσχετισμού είναι μεγαλύτερη από αυτή των λογικών τελεστών, ενώ η προτεραιότητα και των δύο (συσχετισμού και λογικών) είναι μικρότερη από αυτή των αριθμητικών τελεστών!
- Η λογική έκφραση που ακολουθεί μας βρίσκει αν ένα έτος είναι δίσεκτο ή όχι:

  (etos % 4 = = 0 && etos % 100 != 0 || etos % 400 = = 0)

- Δίσεκτα έτη θεωρούνται όσα διαιρούνται ακριβώς με το *4* αλλά όχι με το *100*. Εξαίρεση αποτελούν τα έτη που διαιρούνται ακριβώς με το *400*, που θεωρούνται επίσης δίσεκτα. Αν το έτος λοιπόν είναι δεν είναι δίσεκτο τότε η λογική έκφραση επιστρέφει την τιμή *0* (ψευδής).
- Άλλος τρόπος:

  (! (etos % 4) && (etos % 100) || ! (etos % 400))

Οι τελεστές ολίσθησης “<<“ και “>>”

- Πρόκειται για λογικούς τελεστές *που χειρίζονται bits*.
- Με τη εφαρμογή των τελεστών ολίσθησης τα bits της μεταβλητής μετατοπίζονται προς τα αριστερά ή δεξιά.

<<	Ολίσθηση προς αριστερά
>>	Ολίσθηση προς δεξιά.

Η έκφραση *alpha << 2* μετατοπίζει τα bits της μεταβλητής alpha κατά 2 θέσεις αριστερά. Στα κενά που δημιουργούνται τοποθετούνται μηδενικά.

Η έκφραση *alpha >> 2* μετατοπίζει τα bits της μεταβλητής alpha κατά 2 θέσεις δεξιά. Αν η μεταβλητή δεν έχει πρόσημο στα κενά που δημιουργούνται τοποθετούνται μηδενικά. Αν η μεταβλητή έχει πρόσημο τότε τα bits που αδειάζουν, είτε παίρνουν το bit του πρόσημου, είτε "γεμίζουν" με 0.

Παραδείγματα με τους τελεστές ολίσθησης

```
/* Ολίσθηση αριστερά
unsigned short K, J;
J=12;                  // J = 0000000000001100₂
 K=J << 2;             // K = 0000000000110000₂ (48₁₀)
```

```
/* Ολίσθηση δεξιά
unsigned short K, J;
J=12;                  // J = 0000000000001100₂
 K=J >> 2;             // K = 0000000000000011₂ (3₁₀)
```

- Η ολίσθηση μιας μεταβλητής προς τα αριστερά κατά *n* θέσεις ισοδυναμεί με πολλαπλασιασμό της με 2^n ενώ προς τα δεξιά με διαίρεση της με 2^n.

Άλλοι τελεστές χειρισμού bits

Τελεστής	Περιγραφή
~	*Δυαδική άρνηση (όπου 1 μπαίνει 0 και όπου 0 μπαίνει 1 - one's complement). Υπάρχει ένας μόνο όρος.*
&	*Δυαδικό "**και**" σε επίπεδο bit (σύζευξη / AND). Ο τελεστής απαιτεί δύο όρους. Υπάρχει και ο τελεστής της διεύθυνσης (&) με τον ίδιο συμβολισμό που όταν μπαίνει μπροστά σε μία μεταβλητή αναφέρεται στην διεύθυνση της.*
|	*Δυαδικό διαζευκτικό "**ή**" σε επίπεδο bit (διάζευξη / OR). Ο τελεστής απαιτεί δύο όρους.*
^	*Αποκλειστική διάζευξη "**ή**" (exclusive OR) σε επίπεδο bit. Ο τελεστής απαιτεί δύο όρους.*

Έλεγχος αν ένας αριθμός είναι άρτιος ή περιττός

- Έστω *L* ακέραια μεταβλητή χωρίς πρόσημο (*unsigned int L;*). Θέλουμε να διαπιστώσουμε αν η τιμή που έχει είναι άρτιος ή περιττός αριθμός.
- Αυτό γίνεται αν δούμε τη τιμή της μεταβλητής *K* (ίδιος τύπος με την *L*) όπου:

K= L &1;

Αν K είναι ίσος με 0 είναι τότε ο L είναι άρτιος. Αν K είναι ίσος με 1 τότε ο L είναι περιττός. Γιατί?

- Όταν ένας ακέραιος εκφράζεται σε δυαδικό σύστημα τότε αν είναι άρτιος το τελευταίο bit δεξιά είναι *0*. Αν είναι περιττός τότε είναι *1*.
- Κάνοντας *L&1* ουσιαστικά μηδενίζουμε όλα τα bits του L εκτός του τελευταίου δεξιά (*Least Significant Bit - LSB*) που παραμένει ως έχει.

	00101011	(L = 43)
&	00000001	(1)
	00000001	(K)

	00101010	(L = 42)
&	00000001	(1)
	00000000	(K)

Βασικά ελέγχουμε το LSB

Κάνοντας 1 το i bit από δεξιά

- Πως μπορώ να κάνω το *1* το *i* bit (*set on*) μιας μεταβλητής *L*?
- Έστω ότι έχω μια μεταβλητή με περιεχόμενο *01111000* (πρόκειται για τον αριθμό *120*). Το *3ο* bit είναι *0*.
- Αν θέλω να κάνω *1* το *3ο* bit από δεξιά θα πρέπει να δώσω:

L | 4 ;

- Το *4* στο δυαδικό είναι *00000100* (το *3ο* bit είναι *1* και υπόλοιπα *0*).
- Εκτελώντας *L=L|4;* η *L* παίρνει τη δυαδική τιμή *01111100* όπου το 3ο bit είναι 1 (τα υπόλοιπα παραμένουν ως έχουν).

```
  01111000  (120)
| 00000100  (4)
  01111100  (124)
```

Οι τελεστές αύξησης/μείωσης

Τελεστής	Περιγραφή
++	*κάνει αύξηση μιας μεταβλητής κατά 1 μονάδα π.χ i++;*
- -	*κάνει μείωση μιας μεταβλητής κατά 1 π.χ i-- ;*

- Έχει σημασία αν οι τελεστές αύξησης/μείωσης τεθούν πριν (++i) ή μετά (i--) την μεταβλητή.

```
int a=3, b=3;
int syna, synb;
syna=a++;
synb=++b;
printf ("a= %d, syna= %d, b= %d, synb=%d\n", a, syna, b, synb);
```

Αποτέλεσμα: a=4, syna=3, b=4, synb=4

Οι συνδιαστικοί τελεστές εκχώρησης

+=	*Πρόσθεση σε μεταβλητή. Παράδειγμα: x+=1; είναι το ίδιο όπως x=x+1*
-=	*Αφαίρεση από μεταβλητή. Παράδειγμα: x-=1; (όπως x=x-1)*
***=**	*Πολλαπλασιασμός μεταβλητής. Παράδειγμα: x*=3; (όπως x=x*3)*
/=	*Διαίρεση μεταβλητής. Παράδειγμα: x/=2; (όπως x=x/2)*
%=	*Το υπόλοιπο. Παράδειγμα: alpha %=2 (οπως alpha=alpha %2)*
>>=	*Ολίσθηση δεξιά. Παράδειγμα: beta >>= 3 (τα bits της beta ολίσθησαν δεξιά κατα 3 θέσεις)*
<<=	*Ολίσθηση αριστερά. Παράδειγμα: beta <<= 3 (τα bits της beta ολίσθησαν αριστερά κατά 3 θέσεις)*
&=	*Σύζευξη (AND). Παράδειγμα: var &= 0177 (όπως var = var & 0177)*
|=	*Διάζευξη (OR). Παράδειγμα: var | = 0177*
^=	*Αποκλειστική διάζευξη (exclusive OR). Παράδειγμα: var ^ = 0177*

Ο τελεστής συνθήκης “?”

- Η γενική μορφή χρήσης του τελεστή «?» είναι:

  Λογική έκφραση/ πρόταση ? εντολή-1 : εντολή-2

- Αν η λογική έκφραση/πρόταση είναι αληθής τότε εκτελείται η εντολή-1, διαφορετικά εκτελείται η εντολή-2.

  (alpha > 0) ? printf(“θετικός”) : printf(“αρνητικό ή μηδέν»);

- Ουσιαστικά πρόκειται για μια συμπαγής μορφή της εντολής *if-else* (που θα δούμε αργότερα)

- Η τιμή μιας έκφρασης με τον τελεστή *?* είναι ίση με την τιμή της έκφρασης που εκτελείται τελευταία.

```
num = (x<=5) ? 1 : 0;
```

ισοδύναμα

```
if (x<=5)
        num=1;
else
        num=0;
```

Προτεραιότητες μεταξύ των τελεστών της C

Ιεραρχία	Τελεστές	Φορά
1	( ) [ ] . ->	Από αριστερά πρός δεξιά
2	! ~ ++ -- &(διεύθυνση) *(indirection) (type) sizeof	Από δεξιά πρός αριστερά
3	*(πολλαπλασιασμός) / %	Από αριστερά πρός δεξιά
4	+ -	Από αριστερά πρός δεξιά
5	<< >>	Από αριστερά πρός δεξιά
6	< <= > >=	Από αριστερά πρός δεξιά
7	= = !=	Από αριστερά πρός δεξιά
8	& (AND)	Από αριστερά πρός δεξιά
9	^	Από αριστερά πρός δεξιά
10	|	Από αριστερά προς δεξιά
11	&&	Από αριστερά πρός δεξιά
12	| |	Από αριστερά πρός δεξιά
13	? :	Από δεξιά πρός αριστερά
14	= += -= *= /= %= &= ^= |= <<= >>=	Από δεξιά πρός αριστερά
15	,	Από αριστερά προς δεξιά