

Συσχετιστικοί τελεστές (*relational operators*)

<u>Τελεστής</u>	<u>Δράση</u>
<	Μικρότερο από
>	Μεγαλύτερο από
<=	Μικρότερο ή ίσον από
>=	Μεγαλύτερο ή ίσον από
==	Ίσο
!=	διάφορο

Το αποτέλεσμα είναι πάντοτε είτε **ΑΛΗΘΕΣ (TRUE)** είτε **ΨΕΥΔΕΣ (FALSE)**

Συσχετιστικοί τελεστές

- Παράδειγμα:

- Η τιμή της έκφρασης $(3 < 2)$ είναι **ΨΕΥΔΗΣ**
- Η τιμή της έκφρασης $(2 == 2)$ είναι **ΑΛΗΘΗΣ**

- Στη C (και σε πολλές άλλες γλώσσες),
 - Η τιμή **ΑΛΗΘΗΣ** αντιστοιχεί στον ακέραιο 1
 - Η τιμή **ΨΕΥΔΗΣ** αντιστοιχεί στον ακέραιο 0

Συσχετιστικοί – Αριθμητικοί Τελεστές:

- ΚΑΙ ΟΙ ΔΥΟ χρησιμοποιούν αριθμητικές εισόδους:

Παράδειγμα: $(num < 10)$ και $(num + 10)$
(όπου num είναι μία ακέραια μεταβλητή)

- Ωστόσο, οι συσχετιστικές έξοδοι είναι μόνο TRUE/FALSE.
 - $(number < 10)$ δίνει TRUE ή FALSE (0/1)
 - $(number + 10)$ δίνει οποιοδήποτε αριθμό

Λογικοί τελεστές

Διαφορετικοί από τους συσχετιστικούς τελεστές, καθώς έχουν εισόδους **True/False** και εξόδους **True/False**.

Τελεστής

Δράση

Πίνακας αληθείας

&&

AND

||

OR

!

NOT

p

q

p&&q
(and)

p||q
(or)

!p
(not)

T

T

T

T

F

T

F

F

T

F

F

T

F

T

T

F

F

F

F

T

Λογικοί τελεστές

Παραδείγματα:

```
int x,y;
```

```
x=10;
```

```
y=-8;
```

Υπολογισμός των παρακάτω εκφράσεων:

```
(x+5) < (12-y) (10+5) < (12- (-8))  
15 < 20 → TRUE
```

```
(x>5) || (y>10) (10>5) || (-8>10)  
(TRUE) || (FALSE) → TRUE
```

Τελεστές διαχείρισης δυαδικών ψηφίων

Στη γλώσσα C υπάρχουν έξι τελεστές διαχείρισης δυαδικών ψηφίων (bitwise operators), οι οποίοι επιτρέπουν την εκτέλεση πράξεων σε επίπεδο bit. Οι τελεστές αυτοί βρίσκουν εφαρμογή στα πεδία της κωδικοποίησης και της συμπίεσης δεδομένων. Οι τελεστές της κατηγορίας αυτής παρατίθενται στον κατωτέρω πίνακα, ενώ στον πίνακα της επόμενης διαφάνειας περιγράφεται ο τρόπος λειτουργίας τους (πίνακας αληθείας):

Τελεστής	Δράση
&	λογικό AND
	λογικό OR
~	συμπλήρωμα ως προς 1
^	eXclusive OR
<<	ολίσθηση προς τα αριστερά
>>	ολίσθηση προς τα δεξιά



Τελεστές διαχείρισης δυαδικών ψηφίων

x	y	$x \& y$	$x y$	$x \wedge y$	$\sim x$
1	1	1	1	0	0
1	0	0	1	1	
0	1	0	1	1	1
0	0	0	0	0	

- Η ολίσθηση προς τα αριστερά (bit left shift) μετατοπίζει όλα τα bits του αριστερού τελεστέου κατά τόσες θέσεις προς τα αριστερά, όση είναι η τιμή του δεξιού τελεστέου. Τα bits που απομένουν άνευ περιεχομένου στο δεξί τμήμα του τελεσταίου, συμπληρώνονται με μηδενικά. Η ολίσθηση αυτή πολλαπλασιάζει την υφιστάμενη τιμή του αριστερού τελεστέου επί 2^n , όπου n είναι ο αριθμός των θέσεων ολίσθησης.
- Η ολίσθηση προς τα δεξιά (bit right shift) μετατοπίζει όλα τα bits του αριστερού τελεστέου κατά τόσες θέσεις προς τα δεξιά, όση είναι η τιμή του δεξιού τελεστέου. Τα bits που απομένουν άνευ περιεχομένου στο αριστερό τμήμα του τελεσταίου, συμπληρώνονται με μηδενικά. Η ολίσθηση αυτή διαιρεί την υφιστάμενη τιμή του αριστερού τελεστέου διά 2^n , όπου n είναι ο αριθμός των θέσεων ολίσθησης.
- Όταν χρησιμοποιούνται τελεστές ολίσθησης, είναι προτιμότερο να εφαρμόζεται σε μη προσημασμένες μεταβλητές, καθώς η εφαρμογή τους σε αρνητικούς αριθμούς εξαρτάται από τον εκάστοτε μεταγλωττιστή.

Παράδειγμα *bitwise operators*

Να υπολογιστούν οι εκφράσεις:

(α) $10101010 \ \& \ 00001111$

(β) $010101010 \ | \ 11110000$

(γ) $11001100 \ \wedge \ 00111100$

(δ) ~ 11010110

(ε) $9 \ll 4$

(στ) $36 \gg 2$



Παράδειγμα *bitwise operators*

(α) $10101010 \ \& \ 00001111 \rightarrow (1\&0) (0\&0) (1\&0) (0\&0) (1\&1) (0\&1) (1\&1) (0\&1) \rightarrow 00001010$

(β) $01010101 \ | \ 11110000 \rightarrow (0|1) (1|1) (0|1) (1|1) (0|0) (1|0) (0|0) (1|0) \rightarrow 11110101$

(γ) $11001100 \ ^\wedge \ 00111100 \rightarrow (1^\wedge 0) (1^\wedge 0) (0^\wedge 1) (0^\wedge 1) (1^\wedge 1) (1^\wedge 1) (0^\wedge 0) (0^\wedge 0) \rightarrow 11110000$

(δ) $\sim 11010110 \rightarrow 00101001$

(ε) Ο δεκαδικός αριθμός **9** αντιστοιχεί στον δυαδικό αριθμό **00001001** $\rightarrow b_7b_6b_5b_4b_3b_2b_1b_0$. Μετακίνηση κατά 4 θέσεις προς τα αριστερά σημαίνει ότι θα γίνουν οι ακόλουθες αντικαταστάσεις: $b_7=b_3$, $b_6=b_2$, $b_5=b_1$, $b_4=b_0$, $b_3=b_2=b_1=b_0=0$ και ο αριστερός τελεσταίος γίνεται **10010000**, ο οποίος αντιστοιχεί στον δεκαδικό αριθμό **144**.

(στ) Ο δεκαδικός αριθμός **36** αντιστοιχεί στον δυαδικό αριθμό **00100100** $\rightarrow b_7b_6b_5b_4b_3b_2b_1b_0$. Μετακίνηση κατά 2 θέσεις προς τα δεξιά σημαίνει ότι θα γίνουν οι ακόλουθες αντικαταστάσεις: $b_0=b_2$, $b_1=b_3$, $b_2=b_4$, $b_3=b_5$, $b_4=b_6$, $b_5=b_7$, $b_6=b_7=0$ και ο αριστερός τελεσταίος γίνεται **00001001**, ο οποίος αντιστοιχεί στον δεκαδικό αριθμό **9**.

Ρητή μετατροπή τύπου (typecasting)

Ο τελεστής μετατροπής τύπου ή **cast** τελεστής, όπως αποκαλείται, είναι μοναδιαίος κι έχει τη μορφή (τύπος δεδομένων), π.χ. (float). Τοποθετείται μπροστά από μία έκφραση για να μετατρέψει την τιμή της στον περικλειόμενο σε παρενθέσεις τύπο. Η μετατροπή ισχύει αποκλειστικά στο σημείο εφαρμογής της, όπως φαίνεται στο ακόλουθο παράδειγμα:

Παράδειγμα:

```
int i,j;
```

```
float f1,f2,f3;
```

```
i=5; j=2;
```

```
f1 = i/j + 0.5;
```

```
f2 = (float)i/(float)j + 0.5;
```

```
f3 = i/j + 0.5;
```

μετατροπή των i και j σε float

/* αποτέλεσμα: 2.5 */


/* αποτέλεσμα : 3.0 */

/* αποτέλεσμα : 2.5 */

Παράδειγμα: Στον κώδικα που ακολουθεί αποδεικνύεται ότι η μετατροπή τύπου ισχύει για όλους τους τύπους δεδομένων.

```
#include <stdio.h>
int main(){
    char x='A',y;
    int i=78;
    float f1;
    y=(char)i;
    printf( "\ni=%d y=%c\n",i,y );
    f1=(float)x;
    printf( "x=%c f1=%f\n",x,f1 );
    return 0;
}
```

*Ο ASCII χαρακτήρας με
δεκαδικό ισοδύναμο 78*



```
i=78 y=N
x=A f1=65.000000
—
```

Έμμεση μετατροπή τύπου

Οι έμμεσες μετατροπές διευκολύνουν την εργασία του προγραμματιστή, ο οποίος όμως θα πρέπει σε κάθε περίπτωση να γνωρίζει τις συνέπειες μίας μετατροπής. Για παράδειγμα, η έκφραση **3.0+1/2** δεν δίνει τιμή **3.5**, όπως πιθανόν να ήταν αναμενόμενο, αλλά **3.0**.

Η διαδικασία της αυτόματης μετατροπής στηρίζεται στους ακόλουθους κανόνες:

- Σε κάθε πράξη που υπάρχουν δύο τύποι δεδομένων, ο τύπος χαμηλότερης ιεραρχίας μετατρέπεται στον υψηλότερης ιεραρχίας χωρίς να υπάρχει απώλεια πληροφορίας.
- Οι τύποι της γλώσσας κατατάσσονται ιεραρχικά ανάλογα με το μέγεθος της μνήμης που απαιτούν για αποθήκευση, όπως παρακάτω:

char < int < long < float < double

Ο τύπος **unsigned** ακολουθεί τον αντίστοιχο προσημασμένο τύπο.

- Όλοι οι μεταγλωττιστές της C, όταν υπολογίζουν αριθμητικές εκφράσεις, μετατρέπουν αυτόματα τον τύπο **char** σε **int** και τον **float** σε **double**.



Παράδειγμα έμμεσης μετατροπής τύπου

```
#include <stdio.h>
int main()
{
    char ch;
    int i;
    float fl;
    fl=i=ch='A'; // (1)
    printf( "ch=%c, i=%d, fl=%2.2f, \n", ch, i, fl );
    ch=ch+1;      // (2)
    i=fl+2*ch;    // (3)
    fl=2.0*ch+i;  // (4)
    printf( "ch=%c, i=%d, fl=%2.2f, \n", ch, i, fl );

    return 0;
}
```

ch=A, i=65, fl=65.00,
ch=B, i=197, fl=329.00



Παράδειγμα έμμεσης μετατροπής τύπου

- (1) Ο χαρακτήρας '**A**' αποθηκεύεται ως χαρακτήρας στη μεταβλητή **ch**. Η μεταβλητή **i** λαμβάνει την τιμή του ακέραιου από τη μετατροπή του '**A**' (**65**), ενώ η μεταβλητή **fl** λαμβάνει την τιμή του αριθμού κινητής υποδιαστολής που προέρχεται από το **65** (**65.00**).
- (2) Η πρόσθεση της μονάδας γίνεται στην ακέραια τιμή του '**A**'. Το αποτέλεσμα, **66**, αντιστοιχεί στον χαρακτήρα '**B**', ο οποίος αποθηκεύεται στη μεταβλητή **ch**.
- (3) Η πράξη δίνει **2*66+65.00=197.00**. Το αποτέλεσμα μετατρέπεται σε **int**, **197**, και αποθηκεύεται στη μεταβλητή **i**.
- (4) Η πράξη δίνει **2.0*66+197=329.00** (οι αριθμοί **int** μετατρέπονται σε **float**). Το αποτέλεσμα αποθηκεύεται στη μεταβλητή **fl**.

Τελεστής *sizeof*

Ο τελεστής *sizeof* είναι μοναδιαίος και δρα:

- α) σε έκφραση, π.χ. *sizeof(x+y)* και
- β) σε τύπο δεδομένων, π.χ. *sizeof(int)*

Σε κάθε περίπτωση, επιστρέφει τον αριθμό των bytes που η τιμή της έκφρασης ή ο τύπος των δεδομένων καταλαμβάνει στη μνήμη. Προσοχή θα πρέπει να δοθεί στο γεγονός ότι το σύστημα δεν υπολογίζει την τιμή της έκφρασης κι έτσι πιθανή ύπαρξη παρενεργειών τελεστών δε δημιουργεί παρενέργειες. Μπορεί να βρεθεί το μέγεθος σε bytes ενός πίνακα χρησιμοποιώντας τον τελεστή *sizeof*. Για παράδειγμα, αν θεωρηθεί ο πίνακας *int ar[5]*; η έκφραση *sizeof(ar)* δίνει τιμή 20 επειδή ο πίνακας αποτελείται από 5 ακεραίους των 4 bytes.

Τελεστής *sizeof* (συνέχεια)

Στη *sizeof* θα πρέπει να περιλαμβάνεται μόνο το όνομα του πίνακα. Αν περιληφθεί δείκτης ενός στοιχείου, τότε θα εξαχθεί το μέγεθος του στοιχείου. Για παράδειγμα, η έκφραση *sizeof(ar[0])* δίνει τιμή 4. Χρησιμοποιώντας ένα συνδυασμό των παραπάνω, μπορεί να βρεθεί ο αριθμός των στοιχείων του πίνακα. Η έκφραση *sizeof(ar)/sizeof(ar[0])* δίνει 5, τον αριθμό δηλαδή των στοιχείων του πίνακα *ar*.

Παράδειγμα: Ο κώδικας που ακολουθεί δίνει το μέγεθος των 4 βασικών τύπων δεδομένων της C.

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

```
int void main() {
```

```
    printf( "\nsize of char = %d",sizeof(char) );
```

```
    printf( "\nsize of int = %d",sizeof(int) );
```

```
    printf( "\nsize of float = %d",sizeof(float) );
```

```
    printf( "\nsize of double = %d",sizeof(double) );
```

```
    return 0;
```

```
}
```

```
size of char = 1
size of int = 4
size of float = 4
size of double = 8
```

Θεματική ενότητα 4: **Έλεγχος ροής – Προτάσεις υπό** **συνθήκη διακλάδωσης**

Προτάσεις ελέγχου ροής

Ο πιο συνηθισμένος τρόπος εκτέλεσης είναι ο **ακολουθιακός**:

δύο ή περισσότερες προτάσεις βρίσκονται διατεταγμένες η μία μετά την άλλη και εκτελούνται διαδοχικά.



Προτάσεις ελέγχου ροής

- Ωστόσο, ορισμένες φορές επιβάλλεται να γίνουν λογικές επιλογές (με **χρήση** λογικών τελεστών και τελεστών συσχέτισης).

- Παράδειγμα:

ΕΑΝ στον σηματοδότη βρίσκεται ο **ΓΡΗΓΟΡΗΣ**

ΤΟΤΕ μπορείς να διασχίσεις την οδό

ΑΛΛΙΩΣ περίμενε αλλαγή του σηματοδότη

- Για να επιτευχθεί οποιαδήποτε διαφοροποίηση από την ακολουθιακή εκτέλεση, απαιτούνται ειδικές κατασκευές. Ορισμένες από αυτές τις κατασκευές διασφαλίζουν ταυτόχρονα τη δόμηση του προγράμματος, με κύριο στόχο: **η δομή του πηγαίου κώδικα να μας βοηθά να κατανοήσουμε τι κάνει το πρόγραμμα.** Οι κατασκευές διακρίνονται σε δύο βασικές κατηγορίες:

- 1) την **επανάληψη** (looping)
- 2) την **υπό συνθήκη διακλάδωση** (conditional branching)

Προτάσεις ελέγχου ροής στη C

- Προτάσεις διακλάδωσης υπό συνθήκη

if – else

switch case

- Προτάσεις επανάληψης

while

do while

for

-

- Προτάσεις διακλάδωσης χωρίς συνθήκη

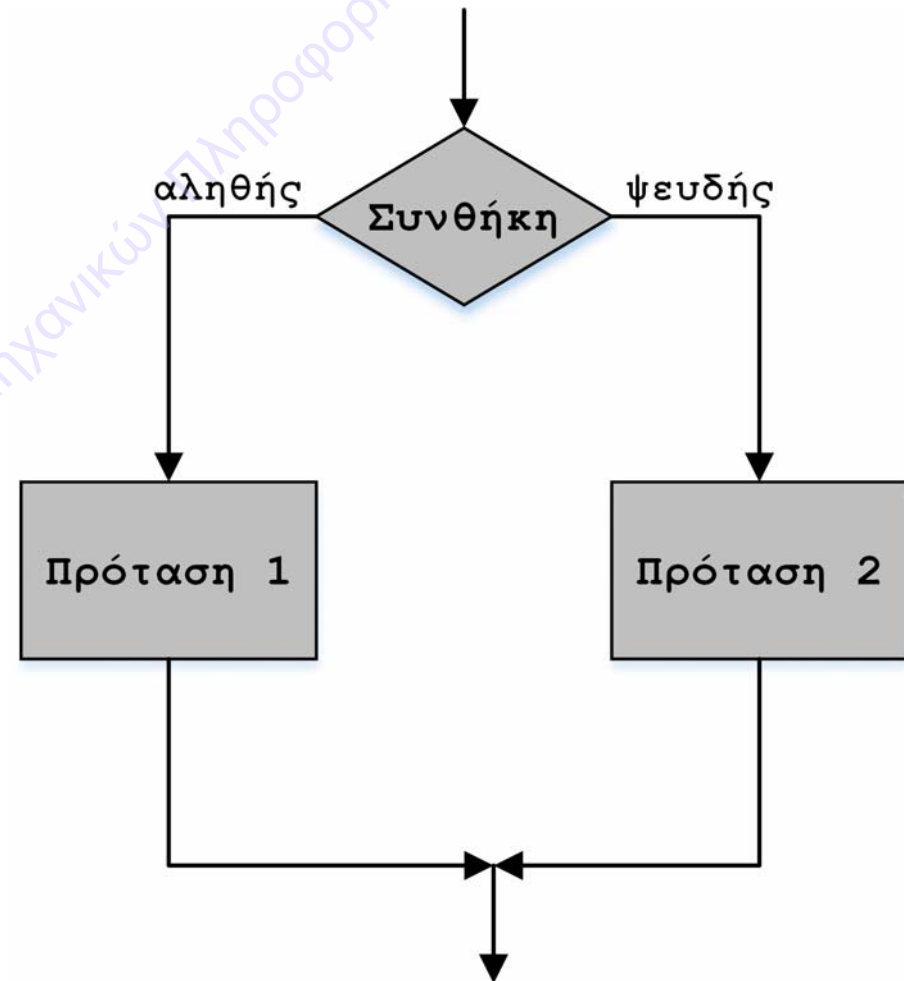
break

continue

goto

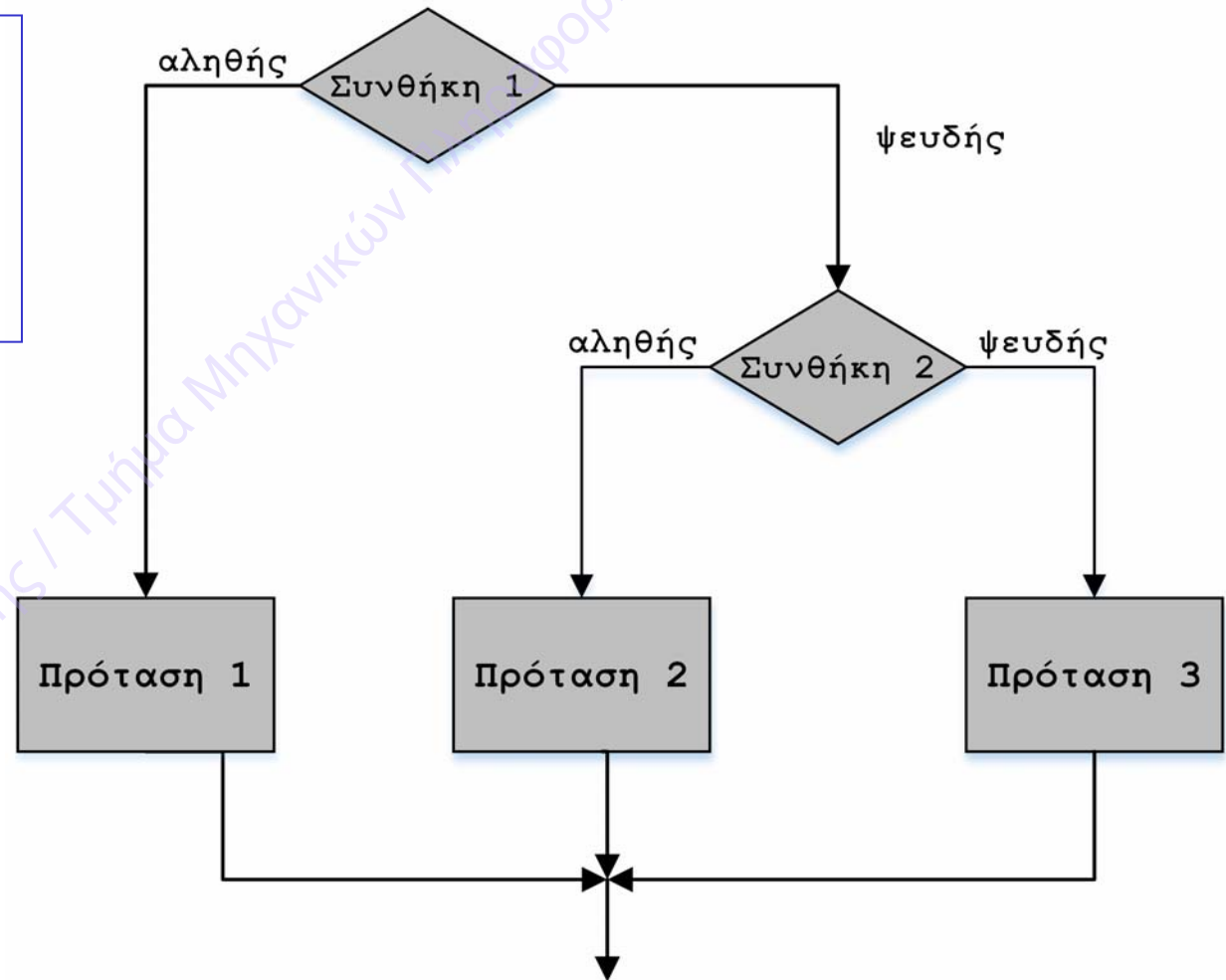
Επιλεκτική εκτέλεση δύο προτάσεων

if Συνθήκη then Π1 else Π2



Επιλεκτική εκτέλεση πολλών προτάσεων με ένθεση

```
if      Συνθήκη 1 then Π1  
else if Συνθήκη 2 then Π2  
else    Π3
```



Παράδειγμα: Να περιγραφεί με ψευδοκώδικα η διεργασία που πρέπει να ακολουθήσει ο υπολογιστής για να διαπιστώσει κατά πόσο ένα δεδομένο έτος είναι δίσεκτο ή όχι.

Να χρησιμοποιηθεί η κατασκευή *if – else*.

Λύση:

Εάν αναπαρασταθεί το έτος με την ακέραια μεταβλητή *year* και ο **τελεστής υπολοίπου** (*modulo*) με το σύμβολο **%**, η περιγραφή μπορεί να γίνει ως ακολούθως:

IF ((year % 400) == 0) THEN το έτος είναι δίσεκτο

ELSE IF ((year % 100) == 0) THEN το έτος δεν είναι δίσεκτο

ELSE IF ((year % 4) == 0) THEN το έτος είναι δίσεκτο

ELSE το έτος δεν είναι δίσεκτο



Εναλλακτικά, ξεκινώντας από την περίπτωση το έτος να μην είναι δίσεκτο, η οποία καλύπτει την πλειοψηφία των ετών, έχουμε επιτάχυνση της εκτέλεσης του κώδικα, καθώς – εν γένει – διενεργούνται λιγότεροι έλεγχοι:

```
IF ((year % 4) != 0) THEN το έτος δεν είναι δίσεκτο  
ELSE IF ((year % 400) == 0) THEN το έτος είναι δίσεκτο  
ELSE IF ((year % 100) == 0) THEN το έτος δεν είναι δίσεκτο  
ELSE το έτος είναι δίσεκτο
```

Πλειοψηφία των ετών

Υπό συνθήκη διακλάδωση *if - else*

Μία δήλωση, τρία τμήματα:

if (light_color==green)

{

cross_the_street();

}

else

{

wait_for_light_to_change();

}

συνθήκη

Σε απλή πρόταση τα άγκιστρα περιττεύουν

Υπό συνθήκη διακλάδωση *if - else*

Μία δήλωση, τρία τμήματα:

```
if (light_color==green)
```

```
{
```

```
    cross_the_street();
```

```
}
```

```
else
```

```
{
```

```
    wait_for_light_to_change();
```

```
}
```

συνθήκη

ΑΛΗΘΕΣ (TRUE) ΤΜΗΜΑ : μία πρόταση ή ένα μπλοκ προτάσεων

Υπό συνθήκη διακλάδωση *if - else*

Μία δήλωση, τρία τμήματα:

```
if (light_color==green)
```

συνθήκη

```
{
```

```
    cross_the_street();
```

```
}
```

```
else
```

```
{
```

```
    wait_for_light_to_change();
```

```
}
```

ΑΛΗΘΕΣ ΤΜΗΜΑ : μία πρόταση ή ένα μπλοκ προτάσεων

ΨΕΥΔΕΣ (FALSE) ΤΜΗΜΑ : μία πρόταση ή ένα μπλοκ προτάσεων

Υπό συνθήκη διακλάδωση *if - else*

- Μερικές φορές δεν υπάρχει *else*, δηλαδή δεν υπάρχει ΨΕΥΔΕΣ τμήμα:

- Παράδειγμα:

```
if (gas_tank_empty == TRUE) fill_up_tank();
```

Εάν η συνθήκη είναι ψευδής (π.χ. το ντεπόζιτο της βενζίνης είναι άδειο) δε γίνεται καμία ενέργεια.

Υπό συνθήκη διακλάδωση *if - else*

- Όταν υπάρχουν περισσότερα από δύο τμήματα και απαιτούνται ένθετες (φωλιασμένες) προτάσεις *if/else*.

- Μπορεί να αντικατασταθεί το ζεύγος

```
else {  
    if (συνθήκη) {  
        προτάσεις; }  
}
```

Π.χ.

```
if (people<5) <get a car>;  
else if (people<15) <get a van>;  
else if (people<50) <get a bus>;  
else <cancel_event>;
```

με την περισσότερο ευανάγνωστη μορφή:

```
else if (συνθήκη) {  
    προτάσεις;}
```

Προτάσεις υπό συνθήκη διακλάδωσης

- `if(συνθήκη) πρόταση;`

- `if (συνθήκη)`
`{`
 προτάσεις;
 προτάσεις; ...
`}`

- `if (συνθήκη)`
`{`
 προτάσεις;
 προτάσεις;...
`}`
`else`
`{`
 προτάσεις;
 προτάσεις;...
`}`
`};`

3 μορφές

Σημειώστε ότι η **συνθήκη** βρίσκεται πάντοτε ανάμεσα σε παρενθέσεις.

Όλα τα ΑΛΗΘΗ τμήματα και όλα τα ΨΕΥΔΗ τμήματα είναι μία μόνο πρόταση ή ένα μπλοκ προτάσεων `{}`.

Παράδειγμα: Να γραφεί πρόγραμμα που υπολογίζει το μέγιστο ανάμεσα σε τρεις ακέραιους

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

```
void main() {
```

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

```
    scanf("%d\n",&a); scanf("%d\n",&b); scanf("%d\n",&c);
```

```
    if (a>b) {
```

```
        if (a>c) printf( "max(%d,%d,%d) = %d\n",a,b,c,a );
```

```
        else printf( "max(%d,%d,%d) = %d\n",a,b,c,c );
```

```
    }
```

```
    else if (b>c) printf( "max(%d,%d,%d) = %d\n",a,b,c,b );
```

```
    else printf( "max(%d,%d,%d) = %d\n",a,b,c,c );
```

```
}
```


Υποθετικός τελεστής

Ο υποθετικός τελεστής (**$?:$**) αποτελείται από δύο σύμβολα. Ανήκει στην κατηγορία των τελεστών που αποτελούνται από συνδυασμό συμβόλων και δεν ακολουθούν καμία από τις postfix, prefix ή infix σημειογραφίες. Όταν τα σύμβολα ή οι λέξεις του τελεστή είναι διάσπαρτα στους τελεστέους στους οποίους εφαρμόζεται ο τελεστής, λέμε ότι ο τελεστής είναι σε **μεικτή σημειογραφία** (mixfix notation).

Υποθετικός τελεστής (συνέχεια)

- Η έκφραση που σχηματίζει ο υποθετικός τελεστής έχει τη μορφή:

$\text{εκφρ1} \text{ ? εκφρ2 : εκφρ3}$

- Η τιμή της παραπάνω έκφρασης είναι η τιμή της εκφρ2 , εάν η εκφρ1 είναι αληθής, αλλιώς είναι η τιμή της εκφρ3 .

- Η εκφρ1 αποτελεί τη συνθήκη ελέγχου. Έτσι η έκφραση
 $x > z \text{ ? } x : z$

έχει τιμή x , εάν το $x > z$ είναι αληθές, διαφορετικά έχει τιμή z .

Παράδειγμα: Να γραφεί κώδικας που βρίσκει το μεγαλύτερο δύο και τριών ακεραίων αριθμών και τον τυπώνει.

Δύο αριθμοί:

```
printf( "maximum is %d\n", (num1>num2)?num1:num2 );
```

Τρεις αριθμοί:

```
max=(num1>num2?num1:num2)> num3 ? (num1>num2?num1:num2):num3;  
printf( "maximum is %d\n", max );
```

Προκύπτει ένας αριθμός και συγκρίνεται με τον **num3**. Εάν ισχύει η ανισότητα αυτός ο αριθμός θα τυπωθεί ως μέγιστος, αλλιώς ο **num3**.

Παράδειγμα: Να γραφεί πρόγραμμα που επιλύει δευτεροβάθμιες εξισώσεις. Να δέχεται ως είσοδο τους συντελεστές της εξίσωσης και να ελέγχει το είδος των ριζών (απλές πραγματικές, συζυγείς μιγαδικές ή διπλή πραγματική).

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

```
#include<math.h>          // για sqrt(), fabs()
```

```
int main() {
```

```
    float a,b,c, D, r1,r2, r,im;
```

```
    printf( "This program provides the roots of the second order equation\n" );
```

```
    printf( "\t\t\tax^2+bx+c=0\n" );
```

```
    printf( "\nGive parameter a:");   scanf("%f",&a );
```

```
    while (fabs(a)<1e-10) { // Έλεγχος για α=0, οπότε η εξίσωση γίνεται α/θμια
```

```
        printf( "For a 2nd order equation, a!=0. Try again\n" );
```

```
        printf( "\nGive parameter a:");   scanf("%f",&a );
```

```
    } // τέλος της if
```



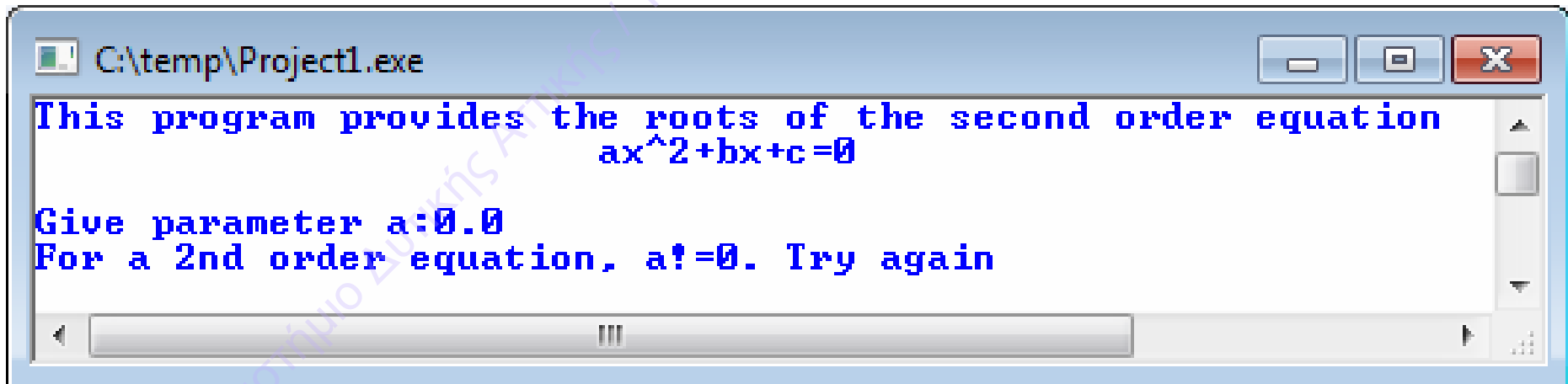
```
printf( "\nGive parameter b:"); scanf("%f",&b );  
printf( "\nGive parameter c:"); scanf("%f",&c );  
printf( "\t\t%.3f*x^2 + %.3f*x + %.3f = 0\n",a,b,c ) ;
```

```
D = b*b-4*a*c; // Διακρίνουσα  
if(D<0) // Εάν  $\Delta < 0$ , οι ρίζες είναι συζυγείς μιγαδικές  
{  
    printf( "There exist two conjugate complex roots:\n" );  
    r=-b/(2*a);  
    im=sqrt(-D)/(2*a);  
    printf( "r1 = %.3f + j%.3f\n",r,im );  
    printf( "r2 = %.3f - j%.3f",r,im );  
} // τέλος της if  
else if (fabs(D)<1e-10) // fabs -> float, abs -> integer  
{  
    // Εάν  $\Delta = 0$ , υπάρχει διπλή ρίζα  
    printf( "There exists a double root:\n" );  
    printf( "r1 = r2 = %.3f\n", -b/(2*a) );  
} // τέλος της else if
```



```
else // Σε κάθε άλλη περίπτωση υπάρχουν δύο πραγματικές ρίζες
{
    printf( "There exist two real roots:\n" );
    r1=(-b+sqrt(D))/(2*a);
    r2=(-b-sqrt(D))/(2*a);
    printf("r1=%.3f\nr2=%.3f\n",r1,r2 );
} // τέλος της else
return 0;
} // τέλος της main
```

Αποτέλεσμα για $a=0$:

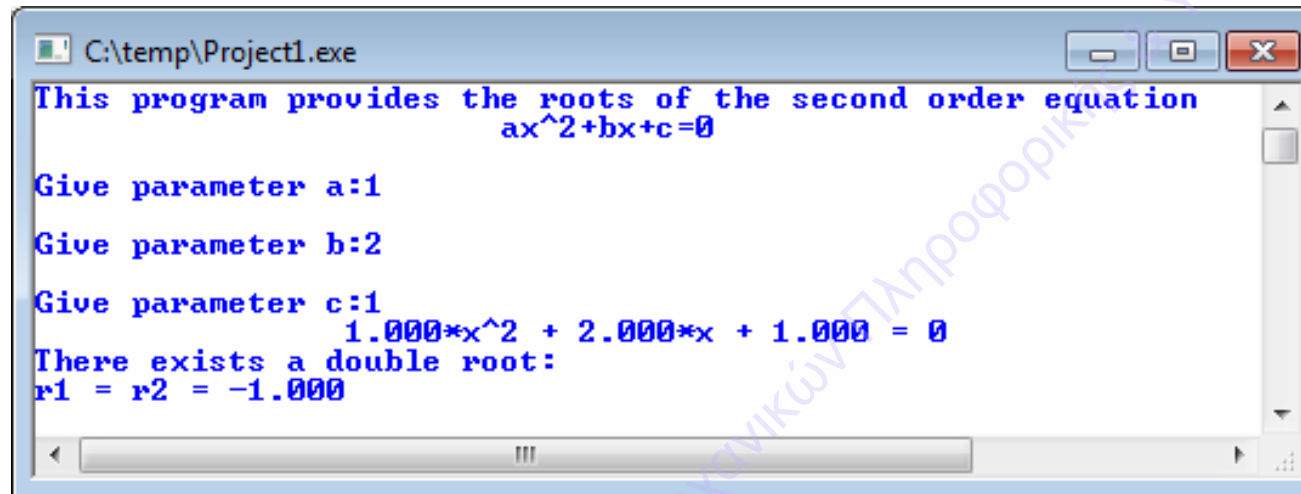


```
C:\temp\Project1.exe

This program provides the roots of the second order equation
          ax^2+bx+c=0

Give parameter a:0.0
For a 2nd order equation, a!=0. Try again
```

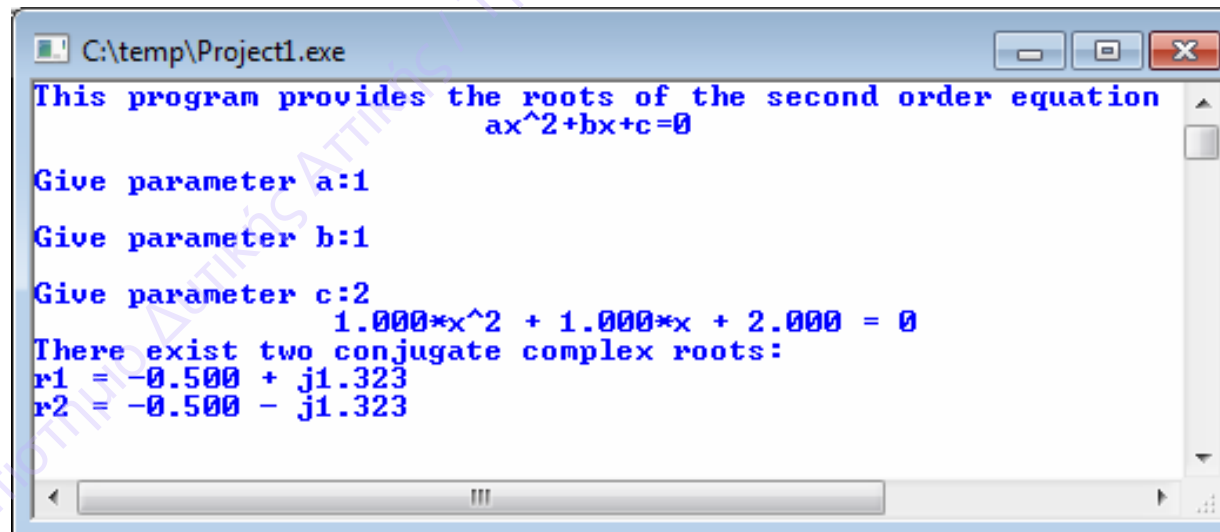
Αποτέλεσμα για διπλή πραγματική ρίζα:



```
C:\temp\Project1.exe
This program provides the roots of the second order equation
      ax^2+bx+c=0

Give parameter a:1
Give parameter b:2
Give parameter c:1
      1.000*x^2 + 2.000*x + 1.000 = 0
There exists a double root:
r1 = r2 = -1.000
```

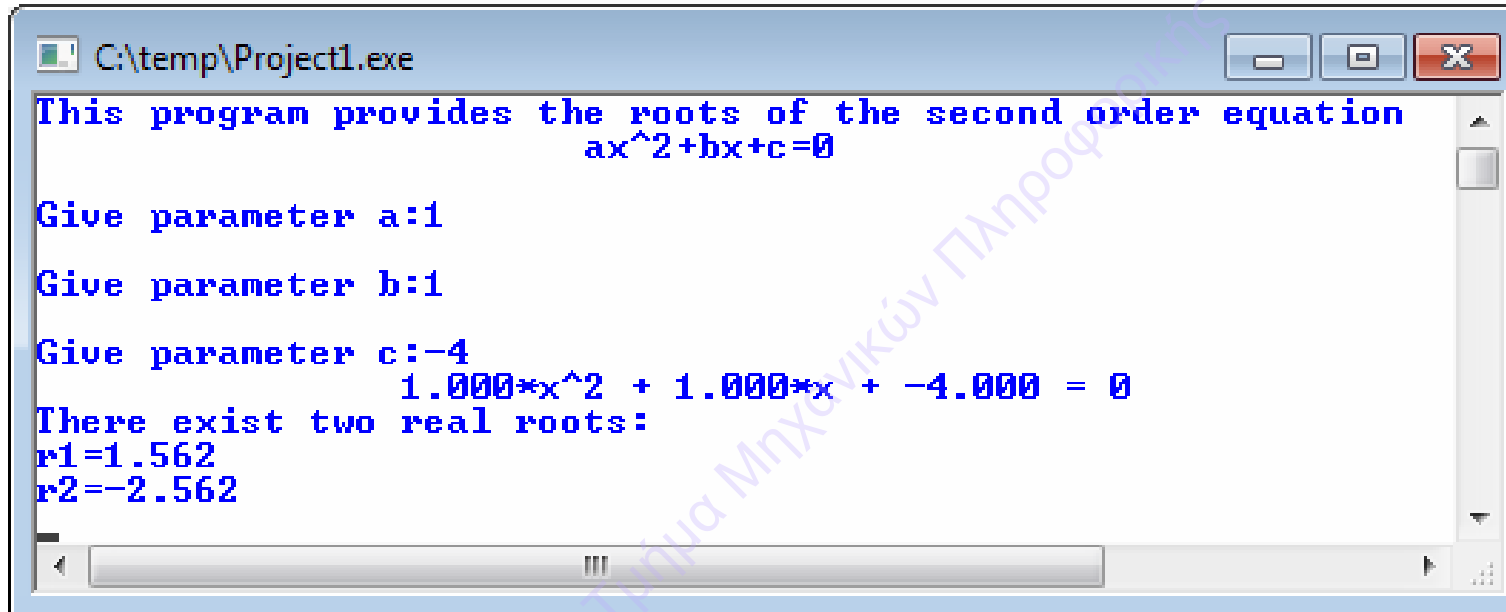
Αποτέλεσμα για συζυγείς μιγαδικές ρίζες:



```
C:\temp\Project1.exe
This program provides the roots of the second order equation
      ax^2+bx+c=0

Give parameter a:1
Give parameter b:1
Give parameter c:2
      1.000*x^2 + 1.000*x + 2.000 = 0
There exist two conjugate complex roots:
r1 = -0.500 + j1.323
r2 = -0.500 - j1.323
```

Αποτέλεσμα για απλές πραγματικές ρίζες:



```
C:\temp\Project1.exe  
This program provides the roots of the second order equation  
ax^2+bx+c=0  
  
Give parameter a:1  
Give parameter b:1  
Give parameter c:-4  
1.000*x^2 + 1.000*x + -4.000 = 0  
There exist two real roots:  
r1=1.562  
r2=-2.562
```

Τροποποίηση του προηγούμενου προγράμματος. Επιτρέπει να εισαχθεί μηδενικός συντελεστής για τον μεγιστοβάθμιο όρο. Στην περίπτωση αυτή επιλύει πρωτοβάθμια εξίσωση:

trinomial_ENHANCED.c