

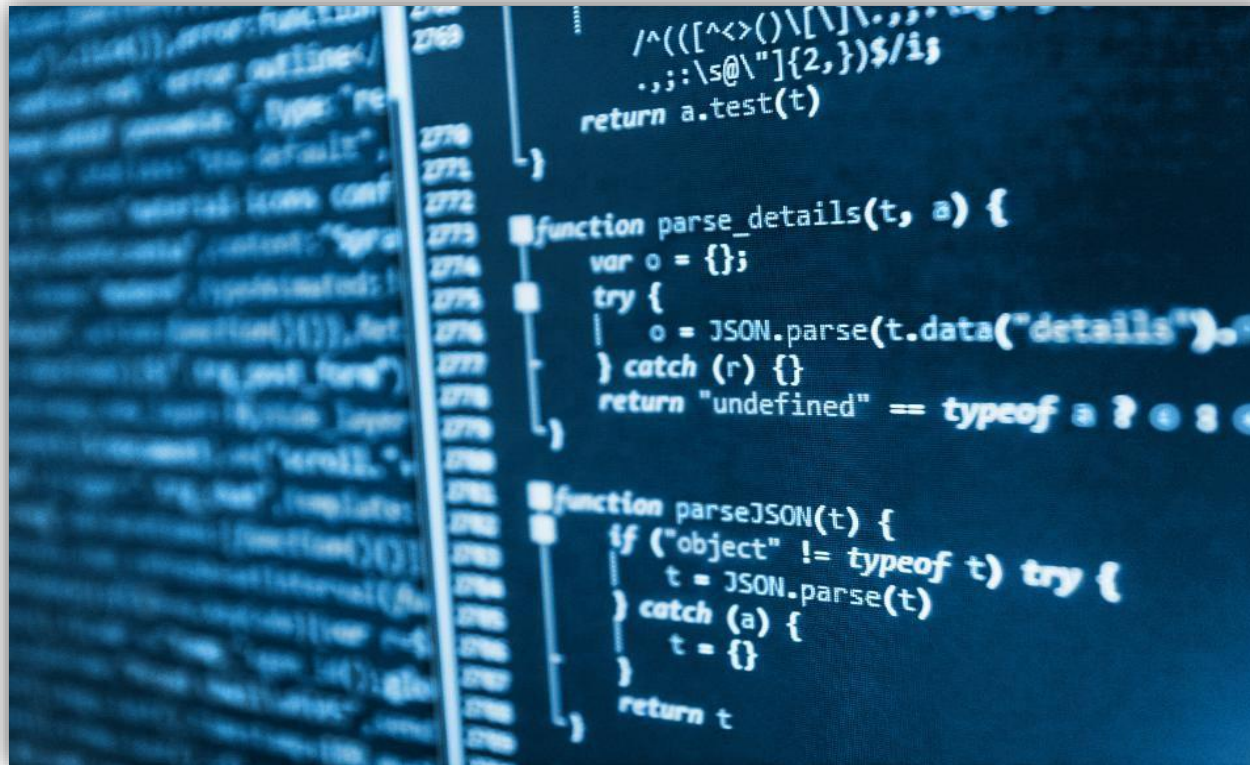
**ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΙΩΑΝΝΙΝΩΝ**



**ΤΜΗΜΑ  
ΠΛΗΡΟΦΟΡΙΚΗΣ &  
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

# ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Ι

## Τελεστές ΙΙ



```
.../^\s*([<>()\\[\]{}.,:;@\"']{2,})$/i;
...return a.test(t)

2770
2771
2772
2773 function parse_details(t, a) {
2774   var o = {};
2775   try {
2776     o = JSON.parse(t.data("details"));
2777   } catch (r) {}
2778   return "undefined" == typeof a ? o : a
2779 }
2780
2781 function parseJSON(t) {
2782   if ("object" != typeof t) try {
2783     t = JSON.parse(t)
2784   } catch (a) {
2785     t = {}
2786   }
2787   return t
2788 }
```

# Ο τελεστής !

- Ο τελεστής ! είναι μοναδιαίος, δηλαδή εφαρμόζεται σε έναν μόνο τελεστέο
- Αν μία έκφραση  $exp$  είναι **αληθής** (δηλαδή έχει **μη μηδενική τιμή**), τότε το αποτέλεσμα της πράξης  $!exp$  είναι μηδέν (0)
- Αν μία έκφραση  $exp$  είναι **ψευδής** (δηλαδή έχει **μηδενική τιμή**), τότε το αποτέλεσμα της πράξης  $!exp$  είναι ένα (1)

# Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος:

```
#include <stdio.h>
int main()
{
    int a = 4;
    printf("Num = %d\n", !a);
    return 0;
}
```

Έξοδος: Num = 0

- Ποια είναι η έξοδος του παρακάτω προγράμματος:

```
#include <stdio.h>
int main()
{
    int a = 4;
    printf("Num = %d\n", !!a);
    return 0;
}
```

Έξοδος: Num = 1

# Παρατηρήσεις

- Συνήθως, ο τελεστής `!` χρησιμοποιείται σε συνθήκες ελέγχου στην εντολή `if`

- Π.χ.  
η εντολή:

```
if (!a)
```

είναι ισοδύναμη με

```
if (a == 0)
```

και η εντολή:

```
if (a)
```

είναι ισοδύναμη με

```
if (a != 0)
```

# Συνδυαστικοί τελεστές

- Οι συνδυαστικοί τελεστές χρησιμοποιούνται για να γραφούν μαθηματικές εκφράσεις με πιο σύντομο τρόπο, βάσει του παρακάτω τύπου:

$$\text{exp1 } \text{op} = \text{exp2};$$

όπου συνήθως ο τελεστής **op** είναι κάποιος από τους αριθμητικούς τελεστές  $+$ ,  $-$ ,  $*$ ,  $\%$ ,  $/$  ή κάποιος από τους τελεστές bit που θα δούμε παρακάτω ( $\&$ ,  $\wedge$ ,  $|$ ,  $\ll$ ,  $\gg$ ).  
Η παραπάνω έκφραση είναι ισοδύναμη με:

$$\text{exp1} = \text{exp1 } \text{op} (\text{exp2});$$

- Π.χ. η έκφραση:

$$a \ += \ b;$$

είναι ισοδύναμη με:

$$a = a + b;$$

ενώ η έκφραση:

$$a \ *= \ b;$$

είναι ισοδύναμη με:

$$a = a * b;$$



# Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος:

```
#include <stdio.h>
int main()
{
    int a = 4, b = 2;

    a += 6;
    a *= b+3;
    a -= b+8;
    a /= b;
    a %= b+1;

    printf("Num = %d\n", a);
    return 0;
}
```

Έξοδος: Num = 2

# Λογικοί τελεστές

- Ο τελεστής  $\&\&$ 
  - Η τιμή μίας έκφρασης που περιέχει τον τελεστή  $\&\&$  είναι ένα (1), δηλαδή **αληθής**, μόνο αν όλοι οι όροι της έκφρασης είναι αληθείς
  - η τιμή της έκφρασης που περιέχει τον τελεστή  $\&\&$  είναι μηδέν (0), δηλαδή **ψευδής**, αν έστω και ένας όρος έχει ψευδή τιμή
  - Ο τελεστής  $\&\&$  εφαρμόζει δηλαδή τη λογική πράξη ΚΑΙ (λογική πράξη AND) μεταξύ των όρων στους οποίους εφαρμόζεται
- Ο τελεστής  $||$ 
  - Μία έκφραση που περιέχει τον τελεστή  $||$  είναι ένα (1), δηλαδή **αληθής**, αν έστω και ένας όρος της έκφρασης είναι αληθής
  - Μία έκφραση που περιέχει τον τελεστή  $||$  είναι μηδέν (0), δηλαδή **ψευδής**, αν κανένας όρος της έκφρασης δεν είναι αληθής
  - Ο τελεστής  $||$  εφαρμόζει δηλαδή τη λογική πράξη Ή (λογική πράξη OR) μεταξύ των όρων στους οποίους εφαρμόζεται



# Παραδείγματα

- Η έκφραση  $(10 == 10) \ \&\& \ (5 > 3)$  είναι **αληθής**, γιατί και οι δύο όροι της έκφρασης είναι αληθείς
- Αν γράψουμε:  
 $a = (10 == 10) \ \&\& \ (5 > 3);$  τότε η τιμή της μεταβλητής  $a$  θα γινόταν ίση με 1
- Η έκφραση  $(10 == 10) \ \&\& \ (5 > 3) \ \&\& \ (13 < 8)$  είναι **ψευδής**, γιατί υπάρχει ένας όρος που έχει ψευδή τιμή
- Αν γράψουμε:  
 $a = (10 == 10) \ \&\& \ (5 > 3) \ \&\& \ (13 < 8);$  τότε η τιμή της μεταβλητής  $a$  θα γινόταν ίση με 0

# Παραδείγματα

- Η έκφραση  $(10 == 10) \ || \ (3 > 5)$  είναι **αληθής**, γιατί ένας όρος της έκφρασης είναι αληθής
- Αν γράψουμε:  
 $a = (10 == 10) \ || \ (3 > 5);$  τότε η τιμή της μεταβλητής  $a$  θα γινόταν ίση με 1
- Η έκφραση  $(10 != 10) \ || \ (3 > 5)$  είναι **ψευδής**, γιατί δεν υπάρχει κάποιος όρος που να είναι αληθής
- Αν γράψουμε:  
 $a = (10 != 10) \ || \ (3 > 5);$  τότε η τιμή της μεταβλητής  $a$  θα γινόταν ίση με 0

# Παρατηρήσεις

- Αν ο όρος που ελέγχεται σε μία έκφραση με τον τελεστή && έχει ψευδή τιμή, τότε ο μεταγλωττιστής δεν ελέγχει τους υπόλοιπους όρους και θέτει κατευθείαν την τιμή της συνολικής έκφρασης ίση με 0
- Αντίστοιχα, αν ο όρος που ελέγχεται σε μία έκφραση με τον τελεστή || είναι αληθής, τότε ο μεταγλωττιστής δεν ελέγχει τους υπόλοιπους όρους και θέτει κατευθείαν την τιμή της συνολικής έκφρασης ίση με 1

## Ασκήσεις

9. Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει τρεις ακεραίους (διαφορετικούς μεταξύ τους) και να εμφανίζει τον μεγαλύτερο και τον μικρότερο.

# Ο τελεστής ,

- Ο τελεστής κόμμα ( , ) διαχωρίζει δευτερεύουσες εκφράσεις οι οποίες εκτελούνται διαδοχικά από αριστερά προς τα δεξιά

- Π.χ.

```
#include <stdio.h>
int main()
{
    int b;
    b = 20, b = b + 30, printf("Num = %d\n", b);
    return 0;
}
```

- Ο τελεστής κόμμα ( , ) – όπως βλέπετε – οδηγεί σε δυσανάγνωστο κώδικα και γι' αυτό δεν χρησιμοποιείται
- Όπως θα δούμε στην συνέχεια, η συνηθέστερη χρήση του είναι στα τμήματα της εντολής `for`

# Ο τελεστής sizeof

- Ο τελεστής `sizeof` υπολογίζει τις οκτάδες που δεσμεύει στη μνήμη του υπολογιστή ο τύπος δεδομένων ή η μεταβλητή που δηλώνεται στις παρενθέσεις του

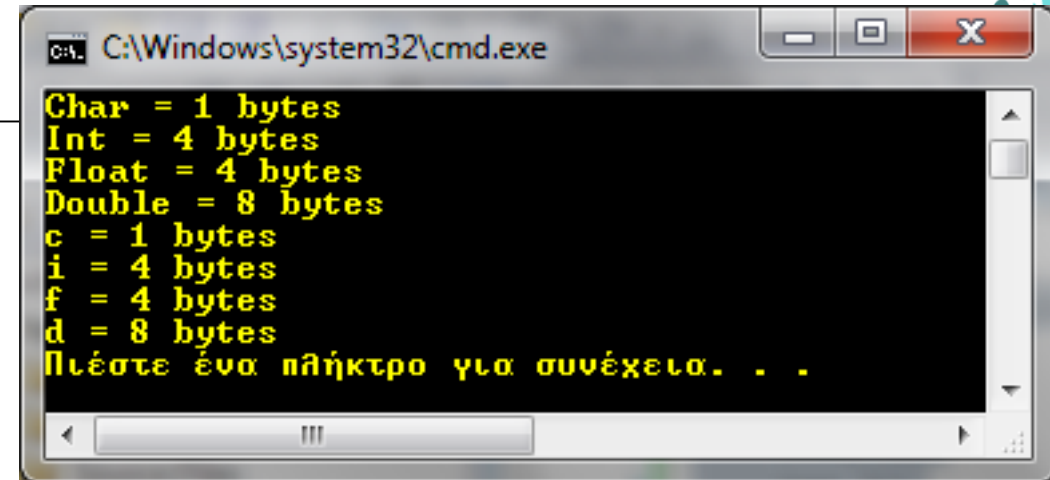
Π.χ.

```
#include <stdio.h>
int main()
{
    char c;
    int i;
    float f;
    double d;

    printf("Char = %d bytes\n", sizeof(char));
    printf("Int = %d bytes\n", sizeof(int));
    printf("Float = %d bytes\n", sizeof(float));
    printf("Double = %d bytes\n", sizeof(double));

    printf("c = %d bytes\n", sizeof(c));
    printf("i = %d bytes\n", sizeof(i));
    printf("f = %d bytes\n", sizeof(f));
    printf("d = %d bytes\n", sizeof(d));

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
Char = 1 bytes
Int = 4 bytes
Float = 4 bytes
Double = 8 bytes
c = 1 bytes
i = 4 bytes
f = 4 bytes
d = 8 bytes
Πιέστε ένα πλήκτρο για συνέχεια. . .
```



# Ασκήσεις

10. Επεκτείνεται το προηγούμενο πρόγραμμα ώστε να περιλαμβάνει και τους τύπους:  
*short, long,*  
*unsigned char, unsigned short,*  
*unsigned int, unsigned long*

# Ο τύπος `enum` (I)

- Ο τύπος απαρίθμησης `enum` (*enumeration type*) χρησιμοποιείται για να οριστεί ένα σύνολο ακεραίων με συγκεκριμένα ονόματα και σταθερές τιμές
- Στην απλή περίπτωση δηλώνεται ως εξής:

```
enum όνομα {λίστα απαρίθμησης};
```

- Το αναγνωριστικό `όνομα` είναι προαιρετικό και δηλώνει το όνομα της απαρίθμησης, π.χ. η εντολή:

```
enum seasons {AUTUMN, WINTER, SPRING, SUMMER};
```

δηλώνει τον τύπο απαρίθμησης `seasons` και τις ακέραιες σταθερές `AUTUMN`, `WINTER`, `SPRING` και `SUMMER`

## Ο τύπος `enum` (II)

- Εξ'ορισμού, κατά τη δήλωση ενός τύπου απαρίθμησης, η τιμή της πρώτης σταθεράς αρχικοποιείται με 0
- Αν σε κάποια σταθερά δεν αποδίδεται τιμή, η τιμή της γίνεται ίση με την τιμή της προηγούμενης σταθεράς αυξημένη κατά ένα.
- Επομένως, στο προηγούμενο παράδειγμα που δεν αποδίδονται τιμές στις σταθερές, οι τιμές των σταθερών `AUTUMN`, `WINTER`, `SPRING` και `SUMMER` γίνονται 0, 1, 2 και 3, αντίστοιχα.
- Σε περίπτωση που στο προηγούμενο παράδειγμα θα θέλαμε να αποδώσουμε συγκεκριμένες τιμές στις σταθερές, θα μπορούσαμε να δηλώσουμε τον τύπο απαρίθμησης π.χ. ως εξής:

```
enum seasons {AUTUMN=10, WINTER=20, SPRING=30, SUMMER=40};
```

# Δήλωση μεταβλητής τύπου `enum`

- Για να δηλώσουμε μία μεταβλητή σύμφωνα με έναν ήδη δηλωμένο τύπο απαρίθμησης γράφουμε:

```
enum όνομα_τύπου λίστα_μεταβλητών;
```

- Π.χ. με την εντολή:

```
enum seasons s1, s2;
```

δηλώνουμε τις `s1` και `s2` σαν μεταβλητές απαρίθμησης του τύπου `seasons` του προηγούμενου παραδείγματος

- Εναλλακτικά, μπορούμε να δηλώσουμε τις μεταβλητές μαζί με τη δήλωση του τύπου απαρίθμησης, π.χ.:

```
enum seasons {AUTUMN, WINTER, SPRING, SUMMER} s1, s2;
```

# Παρατηρήσεις

- Θυμηθείτε ότι η οδηγία `#define` αποτελεί έναν εναλλακτικό τρόπο δήλωσης συμβολικών ονομάτων που αντιστοιχούν σε συγκεκριμένες τιμές
- Η κύρια διαφορά τους είναι ότι ο τύπος `enum` ομαδοποιεί τις σταθερές, ώστε να φαίνεται ότι χαρακτηρίζουν ένα σύνολο τιμών
- Αυτό που πρέπει να θυμάστε με τις μεταβλητές τύπου απαρίθμησης (`enum`) είναι ότι η γλώσσα προγραμματισμού C τις χειρίζεται σαν ακέραιες μεταβλητές ενώ τα ονόματα της λίστας απαρίθμησης σαν ακέραιες σταθερές

```
enum seasons {AUTUMN, WINTER, SPRING, SUMMER} s1, s2;
```

# Οι τελεστές bit

- Οι τελεστές bit χρησιμοποιούνται για το χειρισμό των bits μίας ακέραιας μεταβλητής ή σταθεράς
- Η τιμή ενός bit – ως γνωστόν – μπορεί να είναι 0 ή 1
- Ο υπολογισμός της τιμής μίας έκφρασης που περιέχει τελεστές bit γίνεται με την εφαρμογή τους στα αντίστοιχα bits των τελεστών
- Οι τελεστές bit είναι οι εξής:
  - Ο τελεστής **AND** &
  - Ο τελεστής **OR** |
  - Ο τελεστής **XOR** ^
  - Ο τελεστής **NOT** ~



Όταν εκτελείτε πράξεις με τελεστές bit είναι ασφαλέστερο να δηλώνετε ως **unsigned** τις αντίστοιχες μεταβλητές αλλιώς, να λαμβάνετε υπόψη σας το bit προσήμου, όταν κάνετε τους υπολογισμούς σας



# Ο τελεστής &

- Ο τελεστής & εφαρμόζει τη λογική πράξη **AND** (λογική πράξη **ΚΑΙ**) στα bits των δύο τελεστών και θέτει το bit εξόδου στο 1 μόνο αν τα αντίστοιχα bits και στους δύο τελεστές είναι 1, αλλιώς, το bit εξόδου τίθεται στο 0
- Π.χ. το αποτέλεσμα της πράξης  $19 \ \& \ 2$  είναι 2

	00010011	(19)
&	00000010	(2)
	-----	
	00000010	(2)

# Ο τελεστής |

- Ο τελεστής | εφαρμόζει τη λογική πράξη **OR** (λογική πράξη **'H**) στα bits των δύο τελεστών και θέτει το bit εξόδου στο 0 μόνο αν τα αντίστοιχα bits και στους δύο τελεστές είναι 0, αλλιώς, το bit εξόδου τίθεται στο 1
- Π.χ. το αποτέλεσμα της πράξης  $19 \mid 6$  είναι 23

	00010011	(19)
	00000110	(6)
	-----	
	00010111	(23)

# Ο τελεστής $\wedge$

- Ο τελεστής  $\wedge$  εφαρμόζει τη λογική πράξη **XOR** (e**X**clusive **O**R, αποκλειστικό **'H**) στα bits των δύο τελεστών και θέτει το bit εξόδου στο 1 μόνο αν τα αντίστοιχα bits και στους δύο τελεστές είναι διαφορετικά μεταξύ των, αλλιώς, το bit τίθεται στο 0
- Π.χ. το αποτέλεσμα της πράξης  $19 \wedge 6$  είναι 21

	00010011	(19)
$\wedge$	00000110	(6)
	-----	
	00010101	(21)

# Ο τελεστής ~

- Ο τελεστής συμπληρώματος ~ είναι μοναδιαίος, δηλαδή εφαρμόζεται σε έναν τελεστέο και εφαρμόζει τη λογική πράξη **NOT** (λογική πράξη **ΔΕΝ**)
- Συγκεκριμένα, αντιστρέφει κάθε bit στον τελεστέο του, αλλάζοντας όλα τα 0 σε 1, και αντιστρόφως
- Π.χ. σε ένα 32-bit σύστημα το αποτέλεσμα της πράξης ~19 είναι  $(2^{32} - 1) - 19$

~	00000000	00000000	00000000	00010011	(19)	=
	11111111	11111111	11111111	11101100		

# Οι τελεστές ολίσθησης

- Οι τελεστές ολίσθησης (>> και <<) μετατοπίζουν τα bits μίας ακέραιας μεταβλητής ή σταθεράς κατά ένα συγκεκριμένο αριθμό θέσεων, όπως δείχνουν τα «νοητά βέλη»
- Ο τελεστής >> μετατοπίζει τα bits της μεταβλητής προς τα δεξιά, όπως δηλαδή δείχνουν τα «νοητά βέλη»
- Ο τελεστής << μετατοπίζει τα bits της μεταβλητής προς τα αριστερά, όπως δηλαδή δείχνουν τα «νοητά βέλη»



Επειδή η εφαρμογή των τελεστών ολίσθησης σε αρνητικούς αριθμούς εξαρτάται από τον μεταγλωττιστή, είναι ασφαλέστερο να τους χρησιμοποιείτε σε θετικούς ακεραίους ή `unsigned` μεταβλητές

# Ο τελεστής >>

- Η έκφραση  $i \gg n$  μετατοπίζει τα bits της μεταβλητής  $i$  κατά  $n$  θέσεις δεξιά και τοποθετεί μηδενικά στα  $n$  υψηλότερης τάξης bits της μεταβλητής
- Π.χ. ποια θα είναι η τιμή της μεταβλητής  $a$  κατά την εκτέλεση του παρακάτω κώδικα;

```
unsigned int a, b = 35;  
a = b >> 2;
```

$a = 8$  , διότι:  $00100011 \gg 2 = 00001000$

- Συγκεκριμένα, χάθηκαν τα τελευταία bits 1 και 1 του αρχικού αριθμού (35) και τοποθετήθηκαν τα bits 0 και 0 στην έβδομη και όγδοη θέση, αντίστοιχα
- Και ποια είναι η τιμή της μεταβλητής  $b$  ???



# Ο τελεστής <<

- Η έκφραση  $i \ll n$  μετατοπίζει τα bits της μεταβλητής  $i$  κατά  $n$  θέσεις αριστερά και τοποθετεί μηδενικά στα  $n$  χαμηλότερης τάξης bits της μεταβλητής
- Π.χ. ποια θα είναι η τιμή της μεταβλητής  $a$  κατά την εκτέλεση του παρακάτω κώδικα;

```
unsigned int a, b = 35;  
a = b << 2;
```

$a = 140$  , διότι:  $00100011 \ll 2 = 0010001100$

- Συγκεκριμένα, τα bits του αρχικού αριθμού (35) ολίσθησαν δύο θέσεις αριστερά και τοποθετήθηκαν τα bits 0 και 0 στην πρώτη και στη δεύτερη θέση, αντίστοιχα
  - Και ποια είναι η τιμή της μεταβλητής  $b$  ???

# Παρατηρήσεις (I)

- Όταν χρησιμοποιείται ο τελεστής << και το αποτέλεσμα αποθηκεύεται σε μία μεταβλητή, ο τύπος δεδομένων της μεταβλητής πρέπει να είναι τέτοιος ώστε να μπορεί να αποθηκευτεί η τελική τιμή
- Π.χ. Ποια είναι η έξοδος του προγράμματος ???

```
#include <stdio.h>
int main()
{
    char a = 1;

    a <<= 8; /* Ισοδύναμη με a = a << 8; */
    printf("Value = %d\n", a);
    return 0;
}
```

Περιμένετε να τυπωθεί: Value = 256  
αλλά τυπώθηκε Value = 0  
Γιατί???

## Παρατηρήσεις (II)

- Επειδή η θέση ενός bit αντιστοιχεί σε μία δύναμη του 2:
  - η ολίσθηση ενός θετικού ακεραίου κατά  $n$  θέσεις δεξιά ( $>>$ ) ισοδυναμεί με τη **διαίρεση** της τιμής του με  $2^n$ 
    - Π.χ. θυμηθείτε  $35 >> 2 = 8$
  - η ολίσθηση ενός θετικού ακεραίου  $n$  θέσεις αριστερά ( $<<$ ) ισοδυναμεί με τον **πολλαπλασιασμό** της τιμής του με  $2^n$ 
    - Π.χ. θυμηθείτε  $35 << 2 = 140$

# Προτεραιότητα Τελεστών

- Κάθε τελεστής χαρακτηρίζεται από μία **προτεραιότητα**
- Σε μία έκφραση που περιέχονται περισσότεροι του ενός τελεστές, οι πράξεις εκτελούνται σύμφωνα **με τη σειρά προτεραιότητας** του κάθε τελεστή
- Π.χ. το αποτέλεσμα της πράξης:  
$$7 + 5 * 3 - 1 \text{ είναι } 21,$$
γιατί ο τελεστής  $*$  έχει μεγαλύτερη προτεραιότητα από τους τελεστές  $+$  και  $-$ , οπότε πρώτα εκτελείται η πράξη  $5 * 3 = 15$  και όχι οι πράξεις  $7 + 5$  ή  $3 - 1$
- Αν μία έκφραση περιέχει διαδοχικούς τελεστές με την **ίδια** προτεραιότητα, τότε οι πράξεις εκτελούνται σύμφωνα **με τη συσχέτισή τους** (associativity), δηλαδή από αριστερά προς τα δεξιά ή αντίστροφα
- Π.χ. το αποτέλεσμα της πράξης:  
$$7 * 4 / 2 * 5 \text{ είναι } 70,$$
γιατί, αφού οι τελεστές  $*$  και  $/$  έχουν την ίδια προτεραιότητα και συσχέτιση από αριστερά προς τα δεξιά, τότε:  
πρώτα εκτελείται ο πολλαπλασιασμός  $7 * 4 = 28$ , μετά η διαίρεση  $28 / 2 = 14$  και  
μετά ο πολλαπλασιασμός  $14 * 5 = 70$

# Πίνακας Προτεραιοτήτων

Θέση	Τελεστές	Συσχέτιση
1	( ) [ ] -> .	αριστερά προς δεξιά
2	! ~ ++ -- * (περιεχόμενο) & (διεύθυνση) <b>sizeof()</b>	δεξιά προς αριστερά
3	* (πολλαπλασιασμός) / %	αριστερά προς δεξιά
4	+ -	αριστερά προς δεξιά
5	<< >>	αριστερά προς δεξιά
6	< <= > >=	αριστερά προς δεξιά
7	== !=	αριστερά προς δεξιά
8	&	αριστερά προς δεξιά
9	^	αριστερά προς δεξιά
10		αριστερά προς δεξιά
11	&&	αριστερά προς δεξιά
12		αριστερά προς δεξιά
13	?:	δεξιά προς αριστερά
14	= += -= *= /= %= &= ^=  = <<= >>=	δεξιά προς αριστερά
15	,	αριστερά προς δεξιά

# Παρατηρήσεις

- Όπως φαίνεται στον πίνακα προτεραιοτήτων, κάθε τελεστής χαρακτηρίζεται από μία προτεραιότητα
- Αν μία έκφραση περιέχει διαδοχικούς τελεστές με την ίδια προτεραιότητα, τότε οι πράξεις εκτελούνται σύμφωνα με την συσχέτισή τους
- Προτείνεται η χρήση παρενθέσεων ( ), ακόμα και όταν δεν χρειάζονται, έτσι ώστε ο κώδικας να είναι πιο ευανάγνωστος και να γίνεται σαφέστερη η σειρά εκτέλεσης των πράξεων
- Είναι πιο σαφές να γράψουμε:  
π.χ.1)  $a = 7 + (5 * 3) - 1;$  αντί  $a = 7 + 5 * 3 - 1;$   
π.χ.2) `if ( (a >> 2) == 10)` αντί `if (a >> 2 == 10)`
- Οι τελεστές `:?`, `[]`, `->`, `.`, `&`, `*` θα παρουσιαστούν σε επόμενες διαλέξεις (έλεγχος προγράμματος/πίνακες/δείκτες/δομές)



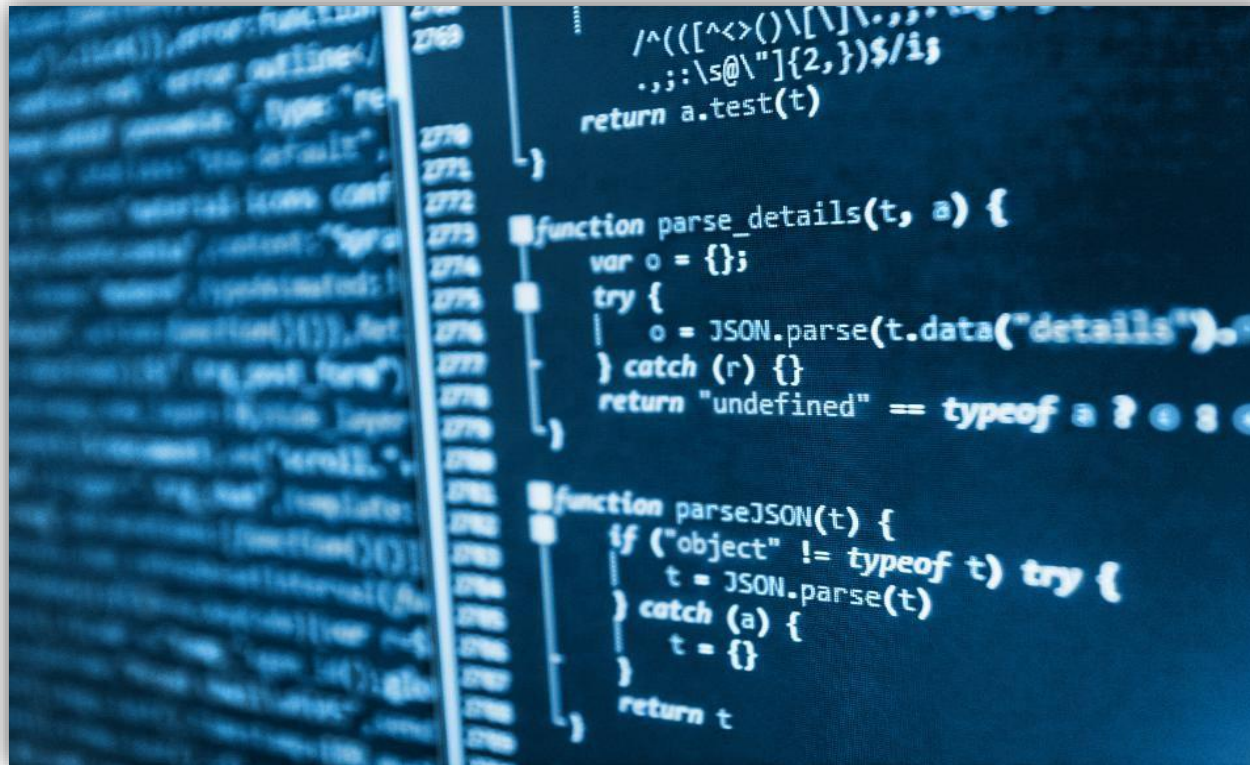
**ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΙΩΑΝΝΙΝΩΝ**



**ΤΜΗΜΑ  
ΠΛΗΡΟΦΟΡΙΚΗΣ &  
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

# ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Ι

## Εντολές Επιλογής



```
.../^\s*([<>()\\[\]{}.,:;@\"']{2,})$/i;
...return a.test(t)

2770
2771
2772
2773 function parse_details(t, a) {
2774     var o = {};
2775     try {
2776         o = JSON.parse(t.data("details"));
2777     } catch (r) {}
2778     return "undefined" == typeof a ? o : a;
2779 }
2780
2781 function parseJSON(t) {
2782     if ("object" != typeof t) try {
2783         t = JSON.parse(t);
2784     } catch (a) {
2785         t = {};
2786     }
2787     return t;
2788 }
```

# Η εντολή `if` (I)

- Η εντολή `if` είναι μία από τις βασικότερες δομές ελέγχου ροής στη C, αλλά και στις περισσότερες γλώσσες προγραμματισμού
- Με την εντολή `if` γίνεται δυνατή η επιλεκτική εκτέλεση ενός τμήματος κώδικα, ανάλογα με την τιμή μίας συνθήκης
- Γενική σύνταξη της εντολής `if` (στην πιο απλή της μορφή):

```
if (συνθήκη)
{
    ... // ομάδα εντολών
}
```

# Η εντολή if (II)

- Αν η συνθήκη είναι **αληθής** (**true**), τότε εκτελούνται οι εντολές που περιλαμβάνονται στα άγκιστρα {...}

```
int x = 3;
if(x != 0)
{
    printf("x isn't zero\n");
}
```

- Αν η συνθήκη **δεν είναι αληθής**, δηλαδή αν η συνθήκη είναι **ψευδής** (**false**), τότε το μπλοκ των εντολών που περιλαμβάνεται στα άγκιστρα παρακάμπτεται και συνεπώς δεν εκτελείται

```
int x = -3;
if(x == 0)
{
    printf("x is zero\n");
}
```

# Παρατηρήσεις (I)

- Αν το μπλοκ εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν

- Π.χ.

```
int x = 3;  
if(x > 0)  
    printf("x is positive\n");
```

- Αν, βέβαια, το μπλοκ εντολών περιέχει περισσότερες από μία εντολές, τότε τα άγκιστρα είναι απαραίτητα

```
int x = 3;  
if(x > 0)  
{  
    printf("x is positive\n");  
    printf("In other words, x is greater than zero\n");  
}
```

# Παρατηρήσεις (II)

## ! ΠΡΟΣΟΧΗ!!!

- Μην βάζετε το ελληνικό ερωτηματικό ; στο τέλος της `if` εντολής, γιατί ουσιαστικά το ερωτηματικό τερματίζει στο σημείο εκείνο την εντολή `if`
- Π.χ. τί εμφανίζει το παρακάτω παράδειγμα ???

```
int x = -3;  
if(x > 0);  
    printf("x is positive\n");
```

- και τί αυτό ???

```
int x = 3;  
if(x > 0);  
    printf("x is positive\n");
```

- Στην οθόνη εμφανίζεται το μήνυμα `x is positive` ανεξάρτητα από την τιμή της μεταβλητής `x`



# Παρατηρήσεις (III)

## ! ΠΡΟΣΟΧΗ!!!

- Μην συγχέετε τον τελεστή ελέγχου ισότητας == (διπλό ίσον) με τον τελεστή εκχώρησης = (μονό ίσον)
- Το παρακάτω πρόγραμμα εμφανίζει στην οθόνη `x equals 2`, αν και η αρχική τιμή της μεταβλητής `x` είναι 3

```
int x = 3;  
if (x = 2)  
    printf("x equals 2\n");
```

- Για να είχαμε «σωστό χειρισμό» στη συνθήκη `if`, η συνθήκη θα έπρεπε να γραφεί ως `if (x == 2)`, δηλαδή με διπλό ίσον και όχι με μονό

# Παρατηρήσεις (IV)



ΠΡΟΣΟΧΗ!!!

- Η εκχώρηση μίας μη μηδενικής τιμής σε μία μεταβλητή ισοδυναμεί με **αληθή συνθήκη**, ενώ η εκχώρηση μηδενικής τιμής ισοδυναμεί με **ψευδή συνθήκη**
- Π.χ. τι εμφανίζει το παρακάτω κομμάτι κώδικα;

```
int x = -3;  
if(x = -2)  
    printf("x equals -2\n");
```

- Και τι αυτό;

```
int x = 0;  
if(x = 0)  
    printf("x equals zero\n");
```



# Παρατηρήσεις (V)

- Η έκφραση:

`if(x)` είναι ισοδύναμη με `if(x != 0)`

- Η έκφραση:

`if(!x)` είναι ισοδύναμη με `if(x == 0)`

- Η εντολή `if` μπορεί προαιρετικά να συμπληρώνεται με την εντολή `else`, όπως θα δούμε στη συνέχεια

# Ασκήσεις

- Σε τρεις διαφορετικούς αγώνες πρόκρισης για την Ολυμπιάδα του Σίδνεϋ στο άλμα εις μήκος ένας αθλητής πέτυχε τις επιδόσεις  $a, b, c$ . Να γραφεί πρόγραμμα που:
  - a. να διαβάσει τις τιμές των επιδόσεων  $a, b, c$
  - b. να υπολογίζει και να εμφανίζει τη μέση τιμή των παραπάνω τιμών
  - c. να εμφανίζει μήνυμα «ΠΡΟΚΡΙΘΗΚΕ» αν η παραπάνω μέση τιμή είναι μεγαλύτερη των 8 μέτρων.

**ΛΥΣΗ:** Solution

# Η εντολή `if...else` (I)

- Όταν θέλουμε να προσδιορίσουμε μία ομάδα εντολών που θα εκτελεστεί όταν μία συνθήκη είναι **αληθής** (**true**) και μία άλλη ομάδα εντολών που θα εκτελεστεί όταν η συνθήκη αυτή είναι **ψευδής** (**false**), τότε χρησιμοποιούμε την εντολή ελέγχου **`if...else`**
- Γενική σύνταξη της εντολής **`if...else`**:

```
if(συνθήκη)
{
    ... // ομάδα εντολών A
}
else
{
    ... // ομάδα εντολών B
}
```

# Η εντολή `if...else` (II)

- Όταν η συνθήκη είναι **αληθής** (**true**), τότε εκτελείται η ομάδα εντολών A (δηλ. οι εντολές που περιέχονται ανάμεσα στα άγκιστρα του **if**), ενώ όταν η συνθήκη είναι **ψευδής** (**false**), τότε εκτελείται η ομάδα εντολών B (δηλ. οι εντολές που περιέχονται ανάμεσα στα άγκιστρα του **else**)
- Π.χ.

```
int x = -3;
if(x > 0)
{
    printf("x is positive\n");
}
else
{
    printf("x is negative or zero\n");
}
```

# Παρατηρήσεις

- Θυμηθείτε ότι στην περίπτωση της εντολής `if`, αν η ομάδα εντολών περιέχει μόνο μία εντολή, τότε τα άγκιστρα μπορούν να παραλειφθούν.
- Το ίδιο ισχύει και στην περίπτωση της εντολής `if...else`
- Δηλαδή, το προηγούμενο παράδειγμα θα μπορούσε να γραφεί και ως εξής:

```
int x = -3;  
if(x > 0)  
    printf("x is positive\n");  
else  
    printf("x is negative or zero\n");
```

- Αν, βέβαια, κάποια από τις ομάδες εντολών περιέχει περισσότερες από μία εντολές, τότε τα άγκιστρα είναι απαραίτητα στο συγκεκριμένο μπλοκ

# Ασκήσεις

- Να γραφεί πρόγραμμα το οποίο θα διαβάσει από το πληκτρολόγιο έναν ακέραιο  $a$  και θα εμφανίζει στην οθόνη το μήνυμα «ΑΡΤΙΟΣ» ή «ΠΕΡΙΤΤΟΣ» ανάλογα με το αν ο αριθμός είναι άρτιος ή περιττός.

**ΛΥΣΗ:** [Solution](#)

# Ασκήσεις

- Τί τιμή έχει η μεταβλητή x μετά την ολοκλήρωση εκτέλεσης καθενός από τα παρακάτω τμήματα προγράμματος;

**ΛΥΣΗ:** Solution

A. <pre>int x=5; if(x&gt;5) {     x-=4; } else {     x+=2; }</pre>	B. <pre>int x=7; if(x&gt;5) {     x-=4; } else {     x+=2; }</pre>	Γ. <pre>int x=5; if (x&gt;5) {     x-=4; } if (x&lt;=5) {     x+=2; }</pre>	Δ. <pre>int x=7; if (x&gt;5) {     x-=4; } if (x&lt;=5) {     x+=2; }</pre>
---	---	--	--

# Ένθετες `if` εντολές (I)

- Στη γενικότερη περίπτωση, τα μπλοκ εντολών των `if` και `else` εντολών επιτρέπεται να περιέχουν και άλλες `if` και `else` εντολές, οι οποίες με τη σειρά τους μπορεί να περιέχουν και άλλες, κ.ο.κ.
- Όταν υπάρχει μία `if` εντολή μέσα σε μία άλλη, τότε αυτή η `if` εντολή ονομάζεται **ένθετη** ή **φωλιασμένη** (*nested*)
- Παράδειγμα με δύο ένθετες `if` εντολές

```
#include <stdio.h>
int main()
{
    int a = 10, b = 20, c = 30;

    if(a > 5)
    {
        if(b == 20)
            printf("1\n");

        if(c == 40)
            printf("2\n");
        else
            printf("3\n");
    }
    else
        printf("4\n");

    return 0;
}
```



- Στην περίπτωση που ένα πρόγραμμα περιέχει **ένθετες if** εντολές, ο κανόνας είναι ότι κάθε **else** εντολή συνδέεται με την αμέσως προηγούμενη **if** εντολή που υπάρχει στην **ίδια ομάδα εντολών** (δηλ. ανάμεσα στα ίδια άγκιστρα), αρκεί αυτή να μη σχετίζεται με άλλη **else** εντολή

## Ένθετες if εντολές (II)

```
#include <stdio.h>
int main()
{
    int a = 5;
    ? if(a != 5) X
      ? if(a-2 > 5) ✓
        printf("One\n");
    else
        printf("Two\n");
    return 0;
}
```

- Όταν γίνεται χρήση ένθετων εντολών **if** προτείνεται η χρήση των αγκίστρων, για να είναι πιο ξεκάθαρη η σχέση μεταξύ των εντολών **else** και **if** (ιδιαίτερα στην περίπτωση που στο πρόγραμμά σας χρησιμοποιείτε μεγάλο αριθμό από **if** και **else** εντολές)

# Ένθετες `if` εντολές (III)

- Στο διπλανό πρόγραμμα, η εντολή `else printf("3\n");` αντιστοιχεί στην πλησιέστερη `if` εντολή, που είναι η `if(c == 40)`
- Όμως, η τελική εντολή `else printf("4\n");` δεν αντιστοιχίζεται με την πλησιέστερη `if` εντολή, που είναι η `if(b == 20)`, γιατί δεν ανήκουν στο ίδιο μπλοκ
- Η εντολή αυτή συνδέεται με την εντολή `if(a > 5)`
- Άρα, η ποια είναι η έξοδος του προγράμματος ???

```
#include <stdio.h>
int main()
{
    int a = 10, b = 20, c = 30;

    if(a > 5)
    {
        if(b == 20)
            printf("1\n");

        if(c == 40)
            printf("2\n");
        else
            printf("3\n");
    }
    else
        printf("4\n");

    return 0;
}
```

Έξοδος: 1 3

## Προτεινόμενη σύνταξη ένθετων `if` εντολών

- Μία πολύ συνηθισμένη χρήση των ένθετων εντολών `if` στηρίζεται στην ακόλουθη σύνταξη:

```
if(συνθήκη_A)
{
    ... /* ομάδα εντολών A */
}
else if(συνθήκη_B)
{
    ... /* ομάδα εντολών B */
}
else if(συνθήκη_C)
{
    ... /* ομάδα εντολών C */
}
.
.
.
else
{
    ... /* ομάδα εντολών N */
}
... /* επόμενες εντολές του προγράμματος. */
```

- Βάσει αυτής της σύνταξης, όταν βρεθεί μία συνθήκη που να είναι αληθής, τότε εκτελείται η ομάδα εντολών που σχετίζεται με αυτή και οι υπόλοιπες `else if` συνθήκες αγνοούνται
- Δηλαδή, η εκτέλεση του κώδικα συνεχίζει με την πρώτη εντολή που υπάρχει μετά την τελευταία `else` εντολή

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

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

    if(a == 1)
        printf("One\n");
    else if(a == 2)
        printf("Two\n");
    else
        printf("Something else\n");

    printf("End\n");
    return 0;
}
```

- Σημειώστε ότι η τελική `else` εντολή δεν είναι υποχρεωτικό να υπάρχει
- Αν δεν υπάρχει, και καμία συνθήκη δεν είναι αληθής, τότε – πολύ απλά – το πρόγραμμα δεν κάνει τίποτα
- Ποια θα ήταν η έξοδος του προηγούμενου παραδείγματος αν δεν υπήρχε η τελική `else` εντολή (βλ. δίπλα) ενώ ο χρήστης εισήγαγε την τιμή 3 ???

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

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

    if(a == 1)
        printf("One\n");
    else if(a == 2)
        printf("Two\n");

    printf("End\n");
    return 0;
}
```

Έξοδος: End

- Να γραφεί πρόγραμμα το οποίο θα δέχεται έναν ακέραιο αριθμό από το πληκτρολόγιο και θα τυπώνει στην οθόνη σχετικό μήνυμα ανάλογα με την κατάσταση του αριθμού, «Θετικός», «Μηδέν» ή «Αρνητικός».

**ΛΥΣΗ:**[Solution](#)

- Να γραφεί πρόγραμμα θα διαβάζει ένα έτος και να εμφανίζει σχετικό μήνυμα αν είναι δίσεκτο ή όχι. Για τον υπολογισμό ισχύουν τα ακόλουθα:

**ΠΡΟΣΟΧΗ!!!!!!**

- Αν διαιρείται με το 400 είναι δίσεκτο (το 2000 είναι δίσεκτο).
- Αν διαιρείται με το 4 αλλά όχι με το 100 είναι δίσεκτο. (το 1980 είναι δίσεκτο, το 1900 δεν είναι).
- Αν ένα έτος δεν διαιρείται με το 4 δεν είναι δίσεκτο

**ΛΥΣΗ:**[Solution](#)



- Ένας υπάλληλος παίρνει επίδομα που είναι ανάλογο με τον αριθμό παιδιών που έχει και τα χρόνια εργασίας στην εταιρία του σύμφωνα με τον παρακάτω πίνακα. Να αναπτύξετε πρόγραμμα το οποίο να δέχεται τα χρόνια εργασίας ενός υπαλλήλου και τον αριθμό των παιδιών του και να υπολογίζει και εμφανίζει το επίδομα που δικαιούται να πάρει.

Αρ. χρόνων εργασίας	Αρ. παιδιών	Επίδομα (ευρώ)
0 - 10	0,1,2	70
	3,4	90
	πάνω από 4	30 για κάθε παιδί
Πάνω από 10	0,1,2	100
	3,4	150
	πάνω από 4	50 για κάθε παιδί

ΛΥΣΗ: [Solution](#)

- Ο Δείκτης Μάζας του ανθρώπινου Σώματος (ΔΜΣ) υπολογίζεται από το βάρος (B) σε χλγ. και το ύψος (Υ) σε μέτρα με τον τύπο  $\Delta\text{ΜΣ} = B/Y^2$ . Ο ανωτέρω τύπος ισχύει για άτομα άνω των 18 ετών. Το άτομο ανάλογα με την τιμή του ΔΜΣ χαρακτηρίζεται σύμφωνα με τον παρακάτω πίνακα:

$\Delta\text{ΜΣ} < 18,5$	“αδύνατο άτομο”
$18,5 \leq \Delta\text{ΜΣ} < 25$	“κανονικό άτομο”
$25 \leq \Delta\text{ΜΣ} < 30$	“βαρύ άτομο”
$30 \leq \Delta\text{ΜΣ}$	“υπέρβαρο άτομο”

Να γράψετε αλγόριθμο ο οποίος:

- να διαβάζει την ηλικία, το βάρος και το ύψος του ατόμου.
- εάν η ηλικία είναι μεγαλύτερη των 18 ετών, τότε:
  - να υπολογίζει το ΔΜΣ
  - να ελέγχει την τιμή του ΔΜΣ από τον ανωτέρω πίνακα και να εμφανίζει τον αντίστοιχο χαρακτηρισμό
- εάν η ηλικία είναι μικρότερη ή ίση των 18 ετών, τότε να εμφανίζει το μήνυμα “δεν ισχύει ο δείκτης ΔΜΣ”.



- Ο τελεστής ? : επιτρέπει την εκτέλεση **μίας** από δύο ενέργειες, σύμφωνα με την τιμή μίας έκφρασης και η σύνταξή του είναι:

```
exp1 ? exp2 : exp3;
```

- Σε μία εντολή με τον τελεστή ? : αν η έκφραση exp1 είναι αληθής, τότε θα εκτελεστεί η έκφραση που ακολουθεί το ερωτηματικό ? (δηλαδή η exp2), αλλιώς θα εκτελεστεί η έκφραση που ακολουθεί την άνω-κάτω τελεία : (δηλαδή η exp3)

- Π.χ.

```
#include <stdio.h>
int main()
{
    int b = 20;
    (b > 10) ? printf("One\n") : printf("Two\n");
    return 0;
}
```

- Ο τελεστής ? : χρησιμοποιείται συνήθως για να υποκαταστήσει την εντολή `if`, όταν αυτή έχει απλή μορφή

- Η τιμή μίας έκφρασης με τον τελεστή ? : είναι ίση με την τιμή της έκφρασης που εκτελείται **τελευταία**
- Ποια είναι η τιμή της μεταβλητής max στην παρακάτω έκφραση ;

```
max = (a > b) ? a : b;
```

- Η παραπάνω έκφραση είναι ισοδύναμη με:

```
if (a > b)
    max = a;
else
    max = b;
```

- Η έκφραση μετά την την άνω-κάτω τελεία : (δηλαδή η `exp3`) μπορεί να αντικατασταθεί από άλλη έκφραση που χρησιμοποιεί τον τελεστή ? :
- Π.χ.

```
k = exp1 ? exp2 : add1 ? add2 : add3;
```

Η παραπάνω έκφραση είναι ισοδύναμη με:

```
if (exp1)
    k = exp2;
else if (add1)
    k = add2;
else
    k = add3;
```

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάσει τρεις ακραίους (διαφορετικούς μεταξύ τους), να υπολογίζει τον μεγαλύτερο και τον μικρότερο με χρήση του τελεστή  $?:$ , και να εμφανίζει τα αποτελέσματα στην οθόνη.

- Η εντολή ελέγχου `switch` χρησιμοποιείται εναλλακτικά έναντι της `if-else-if` δομής, όταν επιθυμούμε να ελέγξουμε μία έκφραση για όλες τις δυνατές τιμές που αυτή η έκφραση μπορεί να πάρει και να χειριστούμε τη κάθε περίπτωση με διαφορετικό τρόπο
- Γενική σύνταξη της εντολής `switch`:

```
switch (έκφραση)
{
    case σταθερά_1:
        /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
        έκφρασης είναι ίση με τη σταθερά_1. */
        break;

    case σταθερά_2:
        /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
        έκφρασης είναι ίση με τη σταθερά_2. */
        break;

    ...

    case σταθερά_n:
        /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
        έκφρασης είναι ίση με τη σταθερά_n. */
        break;

    default:
        /* ομάδα εντολών που θα εκτελεστεί αν η τιμή της
        έκφρασης δεν είναι ίση με καμία από τις προηγούμενες
        σταθερές. */
        break;
}
```

- Η έκφραση που ελέγχεται πρέπει να είναι ακέραιη μεταβλητή ή έκφραση
- Οι τιμές των σταθερά\_1, σταθερά\_2, ..., σταθερά\_n πρέπει και αυτές να είναι ακέραιες σταθερές με διαφορετικές τιμές μεταξύ των
- Τα «βήματα» κατά την εκτέλεση της εντολής `switch`:
  1. Η τιμή της έκφρασης συγκρίνεται διαδοχικά με κάθε μία από τις σταθερά\_1, σταθερά\_2, ..., σταθερά\_n
    - Αν βρεθεί μία ίδια τιμή, τότε εκτελούνται οι εντολές που ακολουθούν το αντίστοιχο `case` και στη συνέχεια γίνεται τερματισμός της εντολής `switch` μέσω της εντολής `break` (λεπτομέρειες για την εντολή `break` σε επόμενο μάθημα...)
    - Αν δεν βρεθεί ίδια τιμή, τότε εκτελούνται οι εντολές που ακολουθούν το `default` και στη συνέχεια γίνεται τερματισμός της εντολής `switch` μέσω της εντολής `break`
  2. Και στις δύο περιπτώσεις, η εκτέλεση του κώδικα συνεχίζει με την πρώτη εντολή που υπάρχει μετά το άγκιστρο κλεισίματος της `switch` εντολής

- Η ύπαρξη της `default` περίπτωσης στην εντολή `switch` δεν είναι υποχρεωτική (όπως δεν ήταν υποχρεωτική και η ύπαρξη της εντολής `else` στην εντολή `if`)
- Σε περίπτωση που δεν υπάρχει η `default` περίπτωση και η τιμή της έκφρασης δεν είναι ίση με κάποια από τις τιμές των `σταθερά_1`, `σταθερά_2`, ..., `σταθερά_n`, τότε γίνεται τερματισμός της εντολής `switch`, χωρίς να γίνει κάποια άλλη ενέργεια
- Δηλαδή, η ροή του προγράμματος συνεχίζει με την εκτέλεση της πρώτης εντολής μετά το `switch`

- Αν τα μπλοκ εντολών που αντιστοιχούν σε δύο ή περισσότερες `case` περιπτώσεις **είναι κοινά**, τότε μπορεί να γίνει συνένωση των αντίστοιχων `case`
- Π.χ. αν τα μπλοκ εντολών για τις περιπτώσεις των σταθερά\_1, σταθερά\_2 και σταθερά\_3 είναι κοινά, τότε τα αντίστοιχα `case` συνενώνονται ως εξής (έχουν, όπως βλέπουμε, κοινή `break`)

```
case σταθερά_1:  
case σταθερά_2:  
case σταθερά_3:  
/* μπλοκ εντολών που θα εκτελεστεί αν η τιμή της έκφρασης  
είναι ίση με σταθερά_1 ή σταθερά_2 ή σταθερά_3. */  
break;
```



- Κάθε `switch` εντολή μπορεί να γραφτεί ισοδύναμα με χρήση πολλαπλών εντολών `if-else-if`
- ΜΕΙΟΝΕΚΤΗΜΑΤΑ ΤΗΣ `switch` έναντι της `if`:
  1. Η εντολή `switch` διαφέρει από την εντολή `if` στο ότι η `switch` κάνει έλεγχο μόνο για ισότητα (δηλαδή, για τιμές της έκφρασης που να είναι **ίσες** με σταθερές `case`), ενώ η συνθήκη σε μία `if` εντολή μπορεί να είναι οποιουδήποτε τύπου
  2. Οι τιμές της έκφρασης της `switch` και των συγκρινόμενων σταθερών πρέπει υποχρεωτικά να είναι ακέραιες

Ποια είναι η έξοδος του προγράμματος, αν ο χρήστης πληκτρολογήσει:

- A) 2
- B) 1
- Γ) 0

Έξοδος:

- A) Two  
End
- B) One  
Two  
End
- Γ) Something else  
End

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

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

    switch(a)
    {
        case 1:
            printf("One\n");

        case 2:
            printf("Two\n");
            break;

        default:
            printf("Something else\n");
            break;
    }
    printf("End\n");
    return 0;
}
```