Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

3α

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #3

9 & 10 Μαρτίου 2022

Παναγιώτης Παύλου

c-programming-23@allos.gr

Εμβάθυνση στις συναρτήσεις

Ο μηχανισμός της κλήσης και οι μεταβλητές τους

Εμβέλεια τοπικών μεταβλητών

Τοπική μεταβλητή, όπως ήδη αναφέρθηκε, είναι η κάθε μεταβλητή που δηλώνεται στα πλαίσια ενός block εντολών. Η μεταβλητή δημιουργείται στη μνήμη με τη δήλωσή της και συνεχίζει να υπάρχει μέχρι η ροή εκτέλεσης να βγει από το συγκεκριμένο block. Η έκταση στην οποία μπορεί να χρησιμοποιηθεί αυτή η μεταβλητή ονομάζεται **εμβέλεια της μεταβλητής**. Στις τοπικές μεταβλητές ταυτίζεται με την έκταση του block στο οποίο ορίζονται.

Μέσα στο ίδιο block εντολών δεν επιτρέπεται να δηλωθεί δεύτερη μεταβλητή με το ίδιο όνομα.


```c
#include <stdio.h>

int main() {
    int A = 1;
    printf("Program starting...\n");
    {// Μπορώ να έχω ένα block χωρίς λόγο!
        printf("Inside the block\n");
        int x;
        x = 10;
        x = x + A;
        printf("Block completing...\n");
    }

    printf("Out of block!\n");
    return 0;
}
```

Επισκίαση μεταβλητών

Κάθε **τοπική μεταβλητή** πρέπει να έχει μοναδικό όνομα στο block της, όμως σε άλλα block – ακόμα και ένθετα/εμφωλευμένα (nested) – δεν υπάρχει περιορισμός. Έτσι ο δίπλα κώδικας είναι απόλυτα ορθός.

Η μεταβλητή Z στο εξωτερικό block είναι διαφορετική από την Z στο εσωτερικό. Για όσο υπάρχει η Z του εσωτερικού block ο κώδικας δεν έχει τρόπο να αναφερθεί στην Z του εξωτερικού block. Αυτό ονομάζεται επισκίαση (shadowing) της μεταβλητής Z.

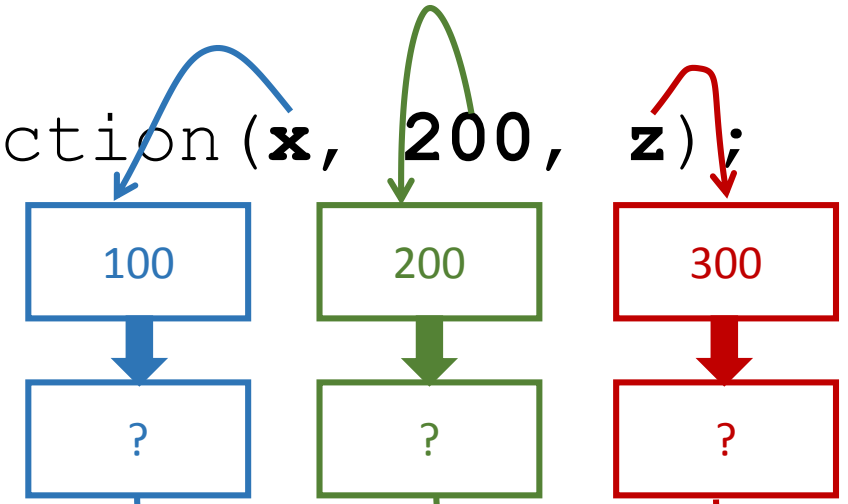
```
int main() {
    int Z = 0;
    printf("Program starting... %d\n",Z);
    {
        printf("Block start: %d\n",Z);
        int Z = 10;
        printf("After definition: %d\n",Z);
        Z = Z + 1;
        printf("Block end: %d\n",Z);
    }

    printf("Out of block! %d\n", Z);
    return 0;
}
```

Call by value – Κλήση με τιμή

Μία κλήση συνάρτησης:

```
some_function(x, 200, z);
```



Και ο ορισμός της:

```
void some_function(int a, int b, int z)
```

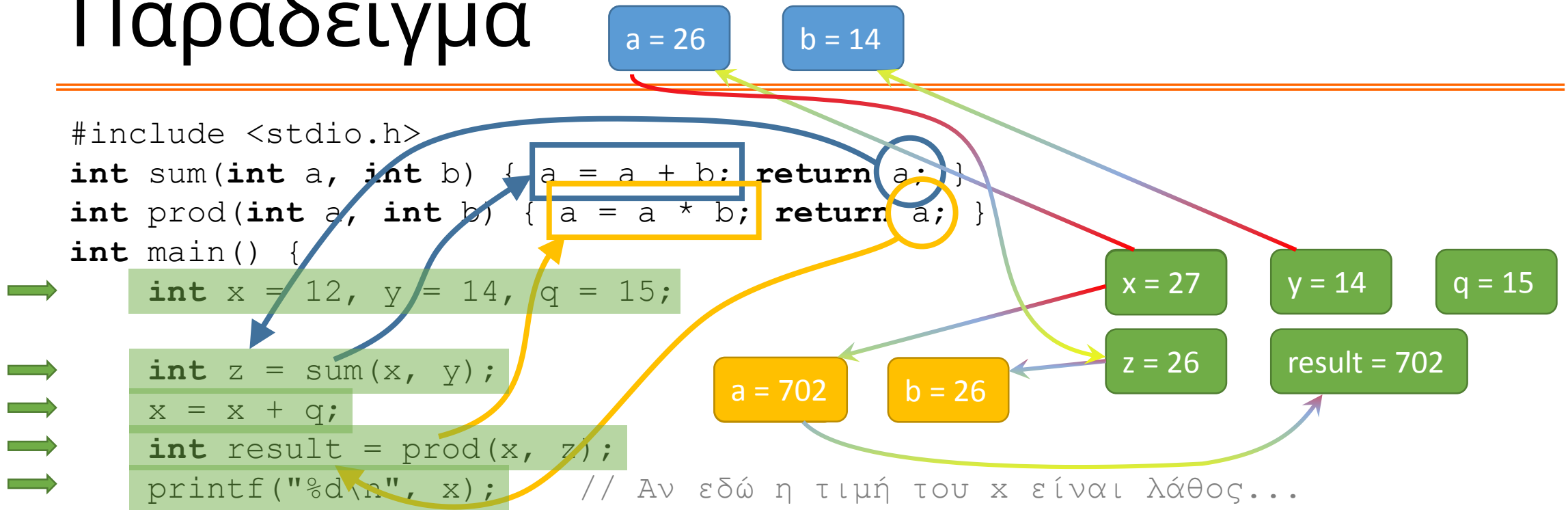
Για να γίνει η κλήση οι τιμές των **ορισμάτων** x,y,z **αντιγράφονται** στις **παραμέτρους** a,b,z και προσέξτε ότι οι a,b,z είναι άλλες μεταβλητές από τις x,y,z !

Συνέπειες του Call-by-value

Η κλήση με τιμή (call-by-value) είναι ο μοναδικός τρόπος που υποστηρίζει η C για την κλήση συναρτήσεων. Αυτό έχει κάποιες συνέπειες:

- Αφού δημιουργούνται αντίγραφα των τιμών, οι όποιες αλλαγές στις παραμέτρους μέσα στο σώμα της συνάρτησης δεν επιδρούν στις τιμές των ορισμάτων. Οι παράμετροι δρουν ως τοπικές μεταβλητές μέσα στη συνάρτηση.
- Καμία κλήση συνάρτησης δεν μπορεί να αλλάξει τις τιμές των ορισμάτων που χρησιμοποιούνται σε αυτή. Αυτό μειώνει τον χρόνο του debugging αισθητά!

Παράδειγμα




```
#include <stdio.h>
int sum(int a, int b) { a = a + b; return a; }
int prod(int a, int b) { a = a * b; return a; }
int main() {
    int x = 12, y = 14, q = 15;

    int z = sum(x, y);
    x = x + q;
    int result = prod(x, z);
    printf("%d\n", x);      // Αν εδώ η τιμή του x είναι λάθος...

    return 0;
}
```

Αν η τιμή του `x` βρεθεί να είναι λανθασμένη στη γραμμή με το σχετικό σχόλιο, τότε δεν μπορεί να οφείλεται στη συνάρτηση `sum`.

Ο προγραμματιστής χρειάζεται να κοιτάξει μόνο τις γραμμές όπου γράφεται `x=` άρα κερδίζει πολύ χρόνο κατά το debugging!

Λογικές Παραστάσεις

Προτάσεις που μπορούν να χαρακτηριστούν αληθείς ή ψευδείς

Λογικές ποσότητες

Μία λογική ή boolean ποσότητα είναι η απλούστερη πληροφορία που μπορεί να παρασταθεί και ταιριάζει και με τη δυαδική λογική του Η/Υ. Η λογική ποσότητα έχει δύο τιμές **αληθή (true)** ή **ψευδή (false)** όπως και ένα δυαδικό ψηφίο.

Δεν υπάρχει σχετικός εγγενής τύπος δεδομένων γι'αυτό παραδοσιακά χρησιμοποιούνται αντίστοιχοι αριθμητικοί τύποι και μάλιστα ακέραιοι. Τυπικά το αληθές αντιστοιχεί στον αριθμό **1** και το ψευδές στο **0**

Γι'αυτό οι όροι **αληθές**, **true** και **1** θα χρησιμοποιούνται εναλλάξιμα και ομοίως οι όροι **ψευδές**, **false** και **0**.

Στην πράξη όμως ο επεξεργαστής θεωρεί το `0` ψευδές και κάθε τι μη μηδενικό αληθές. Έτσι π.χ. και το `-4` αλλά και το `12.34e1` θεωρούνται αληθείς ποσότητες.

Υπάρχουν δύο προσεγγίσεις στην υποστήριξη λογικών ποσοτήτων στη C:

1. Χρήση του τύπου δεδομένων `bool` και τις τιμές `true` ή `false` βάσει του νεότερου προτύπου της C (το C99) το οποίο απαιτεί να γίνει include το αρχείο stdbool.h
2. Ορισμός των `true` και `false` σε `1` και `0` αντίστοιχα με `#define` και χρήση ακέραιου τύπου δεδομένων πχ του unsinged char που γίνεται `#define` ως `bool` (λειτουργεί ανεξαρτήτως έκδοσης της C)

Ανεξαρτήτως της προσέγγισης το αποτέλεσμα είναι το ίδιο. Υπάρχουν λογικές μεταβλητές διαθέσιμες για τον προγραμματιστή.

Παράδειγμα

```c
#include <stdbool.h>

int main() {
    bool isRed = true;
    bool isBad = false;
    return 0;
}
```

```c
#define false 0
#define true 1
#define bool unsigned char

int main() {
    bool isRed = true;
    bool isBad = false;
    return 0;
}
```

Τελεστές σύγκρισης

Οι κατεξοχήν παραστάσεις που δίνουν λογικό αποτέλεσμα είναι οι συγκρίσεις μεταξύ τιμών. Οι συγκρίσεις γίνονται μεταξύ αριθμητικών τιμών με τους ίδιους κανόνες όπως και οι αριθμητικές πράξεις.

Τελεστής	Αντίστοιχο λεκτικό
<	Μικρότερο
<=	Μικρότερο ή ίσο
>	Μεγαλύτερο
>=	Μεγαλύτερο ή ίσο
==	Ίσο
!=	Διάφορο

Δηλαδή συγκρίνουν αριθμητικές ποσότητες μεταξύ τους, αφού πρώτα τις μετατρέψουν στον ίδιο τύπο δεδομένων.

Π.χ.

```
a < 100
c >= d
1000 == x
bool Q = a != b;
```

Τελεστές bool (1/2)

Άλγεβρα bool ονομάζεται η άλγεβρα μεταξύ λογικών ποσοτήτων. Οι βασικές πράξεις της που υποστηρίζονται και σε επίπεδο επεξεργαστή και παράγουν λογικές ποσότητες είναι:

- η άρνηση (NOT) που αντιστοιχεί στον μοναδιαίο τελεστή **!**
  Το αποτέλεσμα της άρνησης είναι η αντίθετη λογική τιμή
- η σύζευξη (AND) που αντιστοιχεί στον τελεστή **&&**
  Το αποτέλεσμα του AND είναι αληθές μόνο όταν και οι δύο τελεσταίοι είναι true, όπως ο πολλαπλασιασμός μεταξύ 0 και 1.
- η διάζευξη (OR) που αντιστοιχεί στον τελεστή **||**
  Το αποτέλεσμα του OR είναι ψευδές μόνο όταν και οι δύο τελεσταίοι είναι false, όπως η πρόσθεση μεταξύ 0 και 1.

x	!x
0	1
1	0

a	b	a && b
0	0	0
1	0	0
0	**1**	0
1	**1**	**1**

a	b	a || b
0	0	0
1	0	**1**
0	**1**	**1**
1	**1**	**1**

Τελεστές bool (2/2)

Τέλος πρέπει να δοθεί προσοχή στα εξής δύο σημεία.

Το ένα είναι η προτεραιότητά των τελεστών, η οποία ταυτίζεται με τη σειρά της προηγούμενης λίστας (NOT , AND , OR).

Επίσης σε σχέση με τους τελεστές σύγκρισης οι AND και OR έχουν χαμηλότερη προτεραιότητα, ενώ ο NOT υψηλότερη.

Δείτε τον πίνακα προτεραιότητας στις σημειώσεις (σελ. 10 στην έκδοση 2020.1)

Το άλλο αφορά τον τρόπο υπολογισμού των λογικών παραστάσεων. Στα OR και AND, εφόσον από τον υπολογισμό του 1ου τελεσταίου προκύπτει το αποτέλεσμα της παράστασης (δηλαδή στο OR αν είναι true και στο AND αν είναι false), τότε δεν υπολογίζεται το 2ο μέρος της παράστασης (ο 2ος τελεσταίος). Π.χ. στο παρακάτω δεν εκτελείται ποτέ η συνάρτηση `test(x)` για `x<5`

```
x < 5 || test(x) == a
```

Παραδείγματα

Οι παρακάτω παραστάσεις είναι αληθείς όταν...

... η μεταβλητή `x` βρίσκεται στο διάστημα [`a b`)

```
a <= x && x < b
```

... μόνο ένα από τα `p` και `q` είναι αληθές

```
p && !q || !p && q
```

... το `x` δεν συμπίπτει με τα `A` και `B`

```
x!=A && x!=B
```

... οι `x, y, z` είναι διάφορες μεταξύ τους

```
x!=y && y!=z && z!=x
```

Τυπικά λάθη

Οι παρακάτω παραστάσεις είναι λανθασμένες επειδή...

... το = είναι ο τελεστής ανάθεσης και όχι σύγκρισης, αυτό σαν λογική παράσταση δίνει πάντα αληθές (αφού το 5 είναι μη μηδενικό)

```
x = 5
```

... οι συγκρίσεις δεν μπορεί να είναι πολλαπλές

```
A < x < B
```

... τα `|` , `^` και `&` δεν είναι λογικοί τελεστές αλλά αφορούν bits γι'αυτό δεν τα χρησιμοποιούμε για κανένα λόγο σε λογικές παραστάσεις. Αν x = 5 και y = 10 το παρακάτω επιστρέφει false παρότι είναι μη μηδενικά ενώ το `x && y` επιστρέφει true

```
x & y
```

... χρειάζεται πολύ προσοχή στην προτεραιότητα. Π.χ. Είναι τα x και y εκτός ορίων;

Εδώ θα εκτελεστεί πρώτα άρα θα χρειαστούν παρενθέσεις

```
( x < 0 || x > A )&&( y < 0 || y > A )
```

Έλεγχος ροής εκτέλεσης

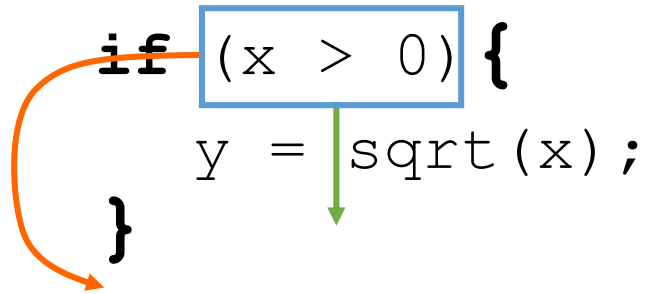
Πως εκτελούνται επιλεκτικά ή επαναλαμβάνονται τμήματα κώδικα

Επιλεκτική εκτέλεση – Εντολή if

Με την μέχρι αυτό το σημείο σειριακή εκτέλεση των εντολών οι δυνατότητες του κώδικα είναι περιορισμένες. Με την εντολή **if** μπορεί ο προγραμματιστής να επιλέξει εάν θα εκτελεστεί κάποιο block κώδικα ή όχι, βάσει της αλήθειας μιας συνθήκης (=μιας λογικής παράστασης).

Αληθής συνθήκη συνεπάγεται εκτέλεση του block. Ψευδής συνθήκη συνεπάγεται παράκαμψη του block.

π.χ.



Η σύνταξη της **if** έχει ως εξής:

```
if ( συνθήκη ) {
    // εντολές που εκτελούνται
    // μόνο αν η συνθήκη ισχύει
}
```

Παρατηρήσεις:

- Ο κώδικας της if είναι στοιχισμένος «πιο μέσα» ώστε να είναι άμεσα εμφανές το ποιες εντολές αφορά.
- Επειδή κάθε μπλοκ εντολών αντιστοιχεί συντακτικά σε μία εντολή, όταν η if καθορίζει την εκτέλεση μίας μόνο εντολής τα άγκιστρα μπορούν να παραληφθούν.

ΠΡΟΣΟΧΗ! Αυτό αποτελεί κακή πρακτική και πρέπει να αποφεύγεται.

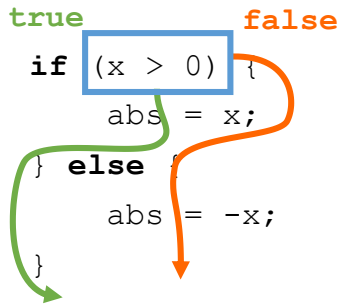
Εναλλακτική εκτέλεση – Εντολή if/else

Κάποιες φορές είναι χρήσιμο να εκτελείται είτε ένα block εντολών (όταν η συνθήκη είναι αληθής), είτε ένα άλλο (όταν η συνθήκη είναι ψευδής). Αυτό γίνεται με τη χρήση της `else` η οποία ακολουθεί το block της `if`. Η `else` συντάσσεται με δικό της block εντολών.

Η σύνταξη της **`if/else`** έχει ως εξής:

```
if ( συνθήκη ) {
    // εντολές για αληθή συνθ.
} else {
    // εντολές για ψευδή συνθ.
}
```

Μόνο ένα από τα δύο block εντολών εκτελείται, αλλά οπωσδήποτε εκτελείται κάποιο



Επειδή τα block εντολών αυτά δεν έχουν κάποια ιδιαιτερότητα, μπορούν να περιέχουν και άλλες εντολές `if/else`. Π.χ.

```
if (age >= 16) {
    if (gender == MALE) {
        // είσαι κύριος
    } else {
        // είσαι κυρία
    }
} else {
    if (gender == FEMALE) {
        // είσαι κορίτσι
    } else {
        // είσαι αγόρι
    }
}
```

Πολλαπλές εναλλακτικές (1/2)

Συχνά χρειάζεται να εξεταστούν πολλαπλές περιπτώσεις εναλλακτικές μεταξύ τους.

Οπότε, σε κάθε `else` υπάρχει εμφωλευμένο ένα και μόνο `if`. Κατά συνέπεια δεν χρειάζονται και τα άγκιστρα στο `else` αφού το `if` είναι μία εντολή.

Για να είναι πιο ξεκάθαρος ο κώδικας και οι εσοχές να μην προκαλούν οπτικό χάος, το ίδιο γράφεται όπως δεξιά.

```
if (x < 0) {
        // x < 0
} else {


}
```

```
if (x < 0) {
        // x < 0
} else
```

Πολλαπλές εναλλακτικές (2/2)

Ο κώδικας αυτό εκτελείται ως εξής: Ξεκινώντας από πάνω προς τα κάτω, ελέγχονται μία προς μία οι συνθήκες μέχρι να βρεθεί η πρώτη αληθής.

Τότε εκτελείται ο κώδικας που αντιστοιχεί στη συνθήκη που επαληθεύθηκε.

Αν δεν επαληθεύεται καμία συνθήκη τότε (εφόσον υπάρχει η τελική else) εκτελούνται οι εντολές της, αλλιώς καμία.

```
if (x < 0) {
        // x < 0
} else if (x < 50){{
        // x >= 0 && x <10
} else if (x < 50){{
        // x >= 50&&&x<10
} else {
        // x >= 10
}
```

Εάν γίνει η εναλλαγή προκύπτει λογικό σφάλμα!

Η παράσταση αυτή είναι πάντα **ψευδής**!

Πρέπει να δοθεί πολύ προσοχή στο εξής: Εάν γραφούν οι συνθήκες με σειρά ώστε μία γενική να προηγείται μίας ειδικής, τότε προκύπτει λογικό σφάλμα, καθώς θα επαληθεύεται η γενική και δε θα δίνεται η ευκαιρία να ελεγχθεί η ειδική συνθήκη.

Αυτό στον δίπλα κώδικα είναι προφανές, αλλά δεν είναι πάντα.

Δίσεκτα έτη

Τα δίσεκτα έτη ξέρουμε όλοι ότι είναι όσα είναι ακέραια πολλαπλάσια του 4.

Υπάρχει όμως μία εξαίρεση: Όσα είναι ακέραια πολλαπλάσια του 100 δεν είναι δίσεκτα (θα περίσσευε μία ημέρα)

Και υπάρχει και η εξαίρεση της εξαίρεσης: Όσα έτη είναι ακέραια πολλαπλάσια του 400 είναι δίσεκτα (θα έλειπε μία ημέρα)

Σημειώστε το προφανές (?) ότι τα ακέραια πολλαπλάσια του 400 είναι ακέραια πολλαπλάσια και του 100 και του 4. Γι'αυτό η σειρά που γράφονται οι συνθήκες χρειάζεται προσοχή!

Αυτό σε κώδικα γράφεται...

```
if (year % 400==0) {
    isLeap = true;  // e.g. 2000
} else if (year % 100==0) {
    isLeap = false; // e.g. 1900
} else if (year % 4==0) {
    isLeap = true;  // e.g. 1984
} else {
    isLeap = false; // e.g. 2019
}
```

Ειδικά Θέματα

Ο τριαδικός τελεστής – Η πολλαπλή ανάθεση

Ο τριαδικός τελεστής ?:

Κάποιες φορές χρειάζεται απόφαση για μία τιμή που θα χρησιμοποιηθεί σε κάποια παράσταση, οπότε δεν συμφέρει να γραφτεί μία if και θα βόλευε κάπως το if να γραφεί «μέσα» στην παράσταση. Π.χ.

```
if (x > 100) {
    hlp = 2.1;
} else {
    hlp = 3.85;
}
a = hlp * sqrt(x);
```

```
a = (x>100?2.1:3.85)
    * sqrt(x);
```

```
if (x > 0) {
    hlp = x;
} else {
    hlp = -x;
}
a = hlp * M_PI;
```

```
a = (x>0?x:-x)*M_PI;
```

```
if (x > 10) {
    hlp = x;
} else {
    hlp = 0;
}
a = hlp + x;
```

```
a = (x>10?x:0)+x;
```

Ο τριαδικός τελεστής δέχεται μία συνθήκη, την τιμή που θα αντικαταστήσει την παράσταση εάν η συνθήκη είναι αληθής και την τιμή που την αντικαταστήσει εάν η συνθήκη είναι ψευδής.

Πολλαπλή ανάθεση τιμής

Μέχρι τώρα είναι ξεκάθαρο ότι:

```
int a=1, b=2;
int c = a + b; // τώρα το c είναι 3
```

```
int c = 3;
```

Όπως η παράσταση `a + b` δίνει 3 στο παραπάνω παράδειγμα, έτσι και παράσταση `k = 12` δίνει ως αποτέλεσμα την τιμή 12. Άρα εάν γράψει κάποιος:

```
q = k = 12;
```

```
int q = 12;
```

Όπως το `c` έπαιρνε την τιμή του `a + b`, έτσι και τώρα το `q` παίρνει την τιμή του `k=12` (που είναι 12). Άρα τόσο το `q` όσο και το `k` έχουν την τιμή 12. Με αυτό τον τρόπο μπορεί να γίνει ανάθεση της ίδιας τιμής σε πολλές μεταβλητές ταυτόχρονα.

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

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  c-programming-23@allos.gr

- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε μαζί τον κώδικα και τα μηνύματα λάθους από το CLion ως κείμενα με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Τονίζουμε : Μην στείλετε **ποτέ κώδικα ως εικόνα**, είναι παντελώς άχρηστος!