ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαδικαστικός Προγραμματισμός

Α' Εξάμηνο

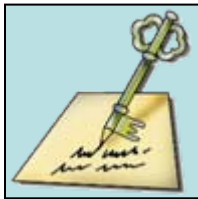
Μάθημα 1ο: Βασικές έννοιες της γλώσσας προγραμματισμού C

Στόχοι μαθήματος

- Να κατανοήσετε τις έννοιες του *πηγαίου*, *αντικειμενικού* και *εκτελέσιμου αρχείου*.
- Να κατανοήσετε το ρόλο των διαδικασιών *μεταγλώττισης* και *σύνδεσης*.
- Να αποκτήσετε μια αίσθηση της *δομής* των προγραμμάτων C.
- Να εκτιμήσετε τη σπουδαιότητα των *βιβλιοθηκών*.
- Να κατανοήσετε ότι πολλά απλά προγράμματα αποτελούνται από τρεις φάσεις: *είσοδο*, *υπολογισμό* και *έξοδο*.
- Να κατανοήσετε το ρόλο των *μεταβλητών* σε ένα πρόγραμμα ως δεσμευτικών χώρου για τιμές δεδομένων που μπορεί να αλλάζουν κατά την εκτέλεση ενός προγράμματος.
- Να αναγνωρίζετε την ύπαρξη διαφορετικών *τύπων δεδομένων*, συμπεριλαμβανομένων των: **`int`**, **`long`**, **`double`**, **`string`**.
- Να χρησιμοποιείτε τις *συναρτήσεις* **`GetInteger()`**, **`GetLong()`**, **`GetReal()`**, **`GetLine()`** της βιβλιοθήκης **`simpio.h`** *για την είσοδο/διάβασμα δεδομένων*.
- Να μπορείτε να κάνετε απλούς υπολογισμούς χρησιμοποιώντας *αριθμητικές παραστάσεις*.
- Να κατανοήσετε τη διεργασία της *μετατροπής αριθμητικών τύπων*.

Αλγόριθμοι

Επιστήμη των υπολογιστών ⇒ επιστήμη που αφορά στην **επίλυση προβλημάτων** (problem solving) με τη χρήση υπολογιστών



Οι στρατηγικές για την επίλυση προβλημάτων με τη βοήθεια του υπολογιστή είναι γνωστές ως **αλγόριθμοι** (algorithms).

Ένας αλγόριθμος πρέπει να είναι:

- Σαφώς και απερίφραστα ορισμένος
- Αποτελεσματικός: τα βήματα του να είναι εκτελέσιμα
- Πεπερασμένος: τερματίζεται μετά από ένα ορισμένο πλήθος βημάτων

Τα παραπάνω κριτήρια είναι γνωστά ως κριτήρια πληρότητας ενός αλγορίθμου.

Προγράμματα

Για να εκτελεστεί ένας αλγόριθμος στον υπολογιστή πρέπει να γραφτεί ένα **πρόγραμμα** (program) που να τον υλοποιεί, διαδικασία γνωστή και ως **κωδικοποίηση** (coding).

Τα προγράμματα γράφονται συνήθως σε μια **γλώσσα υψηλού επιπέδου** (higher-level language)

στη συνέχεια, ο **μεταγλωττιστής** (compiler) που ενσωματώνεται στο χρησιμοποιούμενο προγραμματιστικό περιβάλλον μεταφράζει το πρόγραμμα στη **γλώσσα μηχανής χαμηλού επιπέδου** (lower-level machine language) ενός συγκεκριμένου υπολογιστικού συστήματος

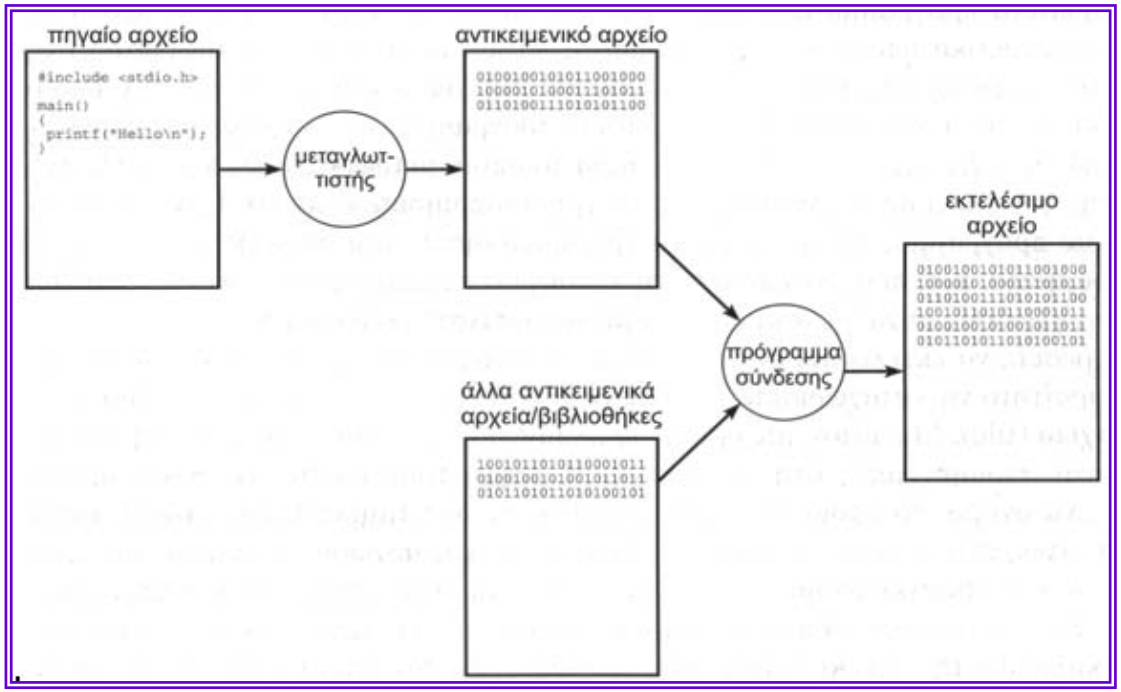
Η διαδικασία της μεταγλώττισης

Δημιουργία **αρχείου πηγαίου κώδικα** (source file), το οποίο περιλαμβάνει το κείμενο του προγράμματος

Ο μεταγλωττιστής μεταφράζει το πηγαίο αρχείο σε ένα **αντικειμενικό αρχείο** (object file) που περιέχει τις εντολές που είναι κατάλληλες για το συγκεκριμένο υπολογιστικό σύστημα

Το αντικειμενικό αρχείο συνδέεται με άλλα αντικειμενικά αρχεία – **βιβλιοθήκες** (libraries), τα οποία περιέχουν τις εντολές γλώσσας μηχανής για διάφορες λειτουργίες που απαιτούνται από τα περισσότερα προγράμματα, διαδικασία γνωστή ως **σύνδεση** (linking)

Το αποτέλεσμα της σύνδεσης είναι η δημιουργία του **εκτελέσιμου αρχείου/προγράμματος** (executable file/program)



Προγραμματιστικά λάθη – Αποσφαλμάτωση (1)

Συντακτικά λάθη:

- Οι γλώσσες προγραμματισμού διαθέτουν ένα σύνολο **συντακτικών κανόνων** (syntax rules), οι οποίοι καθορίζουν αν ένα πρόγραμμα είναι σωστά δομημένο.

- Ο μεταγλωττιστής ελέγχει το πρόγραμμά σας και αν κάποιος ή κάποιοι από αυτούς τους συντακτικούς κανόνες παραβιάζονται τότε αναφέρει ένα **συντακτικό λάθος** (syntax error).

- Σε αυτή την περίπτωση τα λάθη θα πρέπει να διορθωθούν και το πρόγραμμα να μεταγλωττιστεί ξανά.

Προγραμματιστικά λάθη – Αποσφαλμάτωση (2)

Σφάλματα:

- Ο σημαντικότερος τύπος προγραμματιστικού λάθους δεν είναι τα συντακτικά, αλλά τα λάθη που έχουν ως αποτέλεσμα το πρόγραμμα να παράγει λάθος αποτελέσματα ή καθόλου αποτελέσματα.

- Τα λάθη αυτά, τα οποία εμποδίζουν το πρόγραμμά σας να επιλύσει ένα πρόβλημα εξαιτίας ενός λάθους στη λογική σας, ονομάζονται **σφάλματα** (bugs) ή **λογικά λάθη**.

- Η διαδικασία εύρεσης και διόρθωσης των σφαλμάτων ονομάζεται **αποσφαλμάτωση** (debugging).

Τα περισσότερα προγραμματιστικά περιβάλλοντα ενσωματώνουν εργαλεία αποσφαλμάτωσης που στηρίζουν ουσιαστικά τον προγραμματιστή στον εντοπισμό των σφαλμάτων των προγραμμάτων του.

Συντήρηση λογισμικού

Τα περισσότερα προγράμματα πρέπει να ενημερώνονται σε τακτά χρονικά διαστήματα ώστε να διορθώνονται τα σφάλματα ή προκειμένου να ανταποκρίνονται σε αλλαγές στις απαιτήσεις της εφαρμογής, διεργασία γνωστή ως **συντήρηση λογισμικού** (software maintenance).

Η συντήρηση λογισμικού είναι – τις περισσότερες φορές – δύσκολη, γιατί οι προγραμματιστές συνήθως γράφουν ένα πρόγραμμα ώστε να λειτουργεί και να επιλύει το υπάρχον πρόβλημα χωρίς να βλέπουν μακροπρόθεσμα.

Το πεδίο της συγγραφής προγραμμάτων με τέτοιο τρόπο ώστε να μπορούν να κατανοηθούν και να συντηρηθούν από άλλους ονομάζεται **τεχνολογία λογισμικού** (software engineering).

Στόχος μας είναι να μάθετε να γράφετε προγράμματα σε C σύμφωνα με τις αρχές της τεχνολογίας λογισμικού, ή με απλά λόγια με σωστό στυλ.

Το πρόγραμμα "Hello world"

```
/* * Αρχείο: hello.c
    * -------------
    * Το πρόγραμμα αυτό εμφανίζει στην οθόνη το μήνυμα
    * "Hello, world." (γεια σου κόσμε).
    * Το συγκεκριμένο πρόγραμμα περιλαμβάνεται στο
    * κλασικό βιβλίο αναφοράς της C "The C Programming
    * Language", των Brian Kernighan και Dennis Ritchie.
    */
```

σχόλιο προγράμματος

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

συμπεριλήψεις βιβλιοθηκών

```
main()
{
    printf("Hello, world.\n");
}
```

Κυρίως πρόγραμμα

Σχόλια

Σχόλιο (comment) θεωρείται οποιοδήποτε κείμενο περικλείεται μεταξύ των σημειωτών **/*** και ***/**

- **Σκοπός**: τα σχόλια περιγράφουν το έργο/λειτουργία ενός προγράμματος ή γενικότερα ενός τμήματος κώδικα και γράφονται για να βοηθήσουν τους ανθρώπους και όχι τον υπολογιστή στην κατανόηση και συντήρηση ενός προγράμματος.
- Ένα σχόλιο μπορεί να εκτείνεται σε περισσότερες από μία γραμμές.
- Όταν ένα πρόγραμμα μεταγλωττίζεται τα σχόλια αγνοούνται.
- Στο παράδειγμα "Hello world." χρησιμοποιείται ένα ειδικό σχόλιο που ονομάζεται **σχόλιο προγράμματος** και περιλαμβάνει το όνομα του αρχείου του προγράμματος και μια σύντομη περιγραφή της λειτουργίας του.

Τα σχόλια που περιγράφουν τη λειτουργία ενός προγράμματος, το ρόλο των μεταβλητών και τη λειτουργία πολύπλοκων τμημάτων κώδικα αποτελούν παράδειγμα καλού προγραμματιστικού στυλ και θα πρέπει να περιλαμβάνονται στα προγράμματά σας.

Συμπεριλήψεις βιβλιοθηκών (1)

Η **βιβλιοθήκη** (library) είναι μια συλλογή εργαλείων που έχουν γραφτεί από άλλους προγραμματιστές προκειμένου να εκτελούν συγκεκριμένες λειτουργίες.

- **Σκοπός**: οι βιβλιοθήκες είναι πολύ σημαντικές για τον προγραμματισμό, γιατί μας δίνουν τη δυνατότητα να χρησιμοποιούμε τα εργαλεία που περιλαμβάνονται σε αυτές και μας απαλλάσσουν από τον κόπο να τα γράφουμε μόνοι μας.

- Για να χρησιμοποιήσουμε μια βιβλιοθήκη, ο μεταγλωττιστής της C θα πρέπει να γνωρίζει ποια εργαλεία είναι διαθέσιμα σε αυτή τη βιβλιοθήκη. Στις περισσότερες περιπτώσεις, αυτές οι πληροφορίες παρέχονται με τη μορφή ενός **αρχείου κεφαλίδας** (header file), το οποίο περιέχει μια περιγραφή των εργαλείων που παρέχονται από τη συγκεκριμένη βιβλιοθήκη.

Συμπεριλήψεις βιβλιοθηκών (2)

Για κάθε βιβλιοθήκη που θέλουμε να συμπεριλάβουμε σε ένα πρόγραμμα θα πρέπει να γράψουμε - μετά το σχόλιο προγράμματος - μια γραμμή include:

```
# include <όνομα-αρχείου-κεφαλίδας> ή
# include "όνομα-αρχείου-κεφαλίδας"
```

- Οι γωνιακές αγκύλες χρησιμοποιούνται όταν θέλουμε να συμπεριλάβουμε στο πρόγραμμά μας μια **πρότυπη βιβλιοθήκη**, δηλαδή μια βιβλιοθήκη που είναι πάντα διαθέσιμη όταν χρησιμοποιούμε την ANSI C.

- Τα εισαγωγικά χρησιμοποιούνται όταν θέλουμε να συμπεριλάβουμε στο πρόγραμμά μας μια **προσωπική βιβλιοθήκη**, δηλαδή μια βιβλιοθήκη που έχουμε γράψει εμείς.

Συμπεριλήψεις βιβλιοθηκών (3)

Παραδείγματα:

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

συμπερίληψη της **πρότυπης βιβλιοθήκης εισόδου/εξόδου stdio** (standard input/output library) που παρέχεται μαζί με την ANSI C

```
# include "genlib.h"
```

συμπερίληψη της **γενικής βιβλιοθήκης genlib** (general library) που είναι μία από τις εκτεταμένες βιβλιοθήκες που συνοδεύουν το βιβλίο του μαθήματος

Συναρτήσεις: το κυρίως πρόγραμμα

Συνάρτηση (function) είναι μια ακολουθία μεμονωμένων βημάτων προγράμματος που έχουν ομαδοποιηθεί, και στην οποία έχει δοθεί ένα όνομα.

Η συνάρτηση **main** που παρουσιάζεται στο πρόγραμμα "Hello world" είναι το πρώτο παράδειγμα συνάρτησης της C.

Κάθε φορά που εκτελείτε ένα πρόγραμμα C, ο υπολογιστής εκτελεί τις **εντολές** (statements) που περικλείονται στο **σώμα** (body) της συνάρτησης **main**, η οποία θα πρέπει να υπάρχει σε κάθε ολοκληρωμένο πρόγραμμα C.

```
main()
{
        printf("Hello, world.\n");
}
```

Σώμα συνάρτησης

εντολή

Συναρτήσεις: η συνάρτηση printf

Στη συνάρτηση **main** του παραδείγματος μας υπάρχει μία μόνο εντολή:

```
printf("Hello, world.\n");
```

Η εντολή **printf** είναι και η ίδια μια συνάρτηση της βιβλιοθήκης εισόδου/εξόδου που είναι διαθέσιμη στο πρόγραμμά μας, εφόσον την έχουμε συμπεριλάβει με την γραμμή:

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

*Τι κάνει όμως η **printf**;*

- Όπως και η **main**, η **printf** είναι και αυτή μια συνάρτηση, κάτι που σημαίνει ότι αντιστοιχεί σε μια ακολουθία λειτουργιών.

- Όταν θέλετε να χρησιμοποιήσετε αυτές τις λειτουργίες, μπορείτε να αναφερθείτε σε αυτές συνολικά με το όνομα της συνάρτησης.

Συναρτήσεις: κλήση

Στον προγραμματισμό, η επίκληση μιας συνάρτησης με τη χρήση του ονόματός της ονομάζεται **κλήση** (calling) της συνάρτησης.

Για παράδειγμα, η εντολή:

ως **όρισμα** χρησιμοποιείται μια ακολουθία χαρακτήρων που περικλείεται σε εισαγωγικά, ή αλλιώς ένα **αλφαριθμητικό** (string)

```
printf("Hello, world.\n");
```

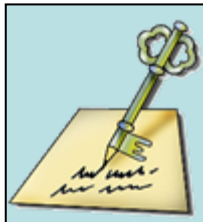
αντιπροσωπεύει μια κλήση της συνάρτησης **printf**.

Κατά την κλήση μιας συνάρτησης, συχνά πρέπει να παρέχουμε επιπλέον πληροφορίες.

Για παράδειγμα, η συνάρτηση **printf** εμφανίζει δεδομένα στην οθόνη. Αυτά τα δεδομένα ονομάζονται ορίσματα:

Όρισμα (argument) είναι μια πληροφορία την οποία θέτει αυτός που έχει κάνει την κλήση μιας συγκεκριμένης συνάρτησης (ο καλών) στη διάθεση της ίδιας της συνάρτησης.

Δεδομένα



Δεδομένα (data) θεωρούμε τις πληροφορίες που χειρίζεται το πρόγραμμα: οποιαδήποτε μηνύματα εμφανίζονται, η εισαγωγή δεδομένων (είσοδος) που ζητείται από τον χρήστη, τιμές που παράγονται ως αποτέλεσμα υπολογισμών, ή τα ενδιάμεσα αποτελέσματα που δημιουργούνται στην πορεία.

Το πρώτο παράδειγμα δεδομένων είναι η ακολουθία χαρακτήρων που περικλείεται σε εισαγωγικά και ονομάζεται **αλφαριθμητικό** (string):

```
"Hello, world.\n"
```

Ο τελευταίος χαρακτήρας του αλφαριθμητικού είναι ένας ειδικός χαρακτήρας που ονομάζεται **χαρακτήρας αλλαγής γραμμής** (newline), υποδεικνύεται με την ακολουθία **\n** και έχει ως αποτέλεσμα την μετατόπιση του δρομέα στην αρχή της επόμενης γραμμής της οθόνης.

Ένα πρόγραμμα άθροισης δύο αριθμών

```
/*
 * File: add2.c
 * ------------
 * Αυτό το πρόγραμμα «διαβάζει» δύο αριθμούς, τους
 * προσθέτει και εμφανίζει το άθροισμά τους.
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
   int n1, n2, total;

   printf("This program adds two numbers.\n");
   printf("1st number? ");
   n1 = GetInteger();
   printf("2nd number? ");
   n2 = GetInteger();
   total = n1 + n2;
   printf("The total is %d.\n", total);
}
```

βιβλιοθήκη απλοποιημένης εισόδου/εξόδου (simplified input/output)

δήλωση μεταβλητών

συνάρτηση της βιβλιοθήκης simpio για την ανάγνωση ακέραιων τιμών που εισάγει ο χρήστης

εντολές ανάθεσης

Οι φάσεις ενός προγράμματος

Τα προγράμματα, στην πλειονότητά τους, χωρίζονται σε τρεις **φάσεις**:

Στο πρόγραμμα άθροισης των 2 αριθμών έχουμε:

- Τη φάση **εισόδου** (input), κατά την οποία το πρόγραμμα ζητά από τον χρήστη να εισαγάγει τους δύο αριθμούς που πρόκειται να προστεθούν.
- Τη φάση **υπολογισμού** (computation), κατά την οποία το πρόγραμμα υπολογίζει το άθροισμα των δύο αριθμών.
- Τη φάση **εξόδου** (output), κατά την οποία το πρόγραμμα εμφανίζει το αποτέλεσμα (το άθροισμα των 2 αριθμών).

Σε κάθε πρόγραμμα πριν από τις παραπάνω φάσεις θα πρέπει να δηλωθούν οι απαραίτητες μεταβλητές.

Δήλωση μεταβλητών (1)

Μια **μεταβλητή** (variable) αποτελεί δεσμευτικό θέσης για μια τιμή (που είναι άγνωστη τη στιγμή που γράφεται το πρόγραμμα) και έχει τρεις σημαντικές ιδιότητες: ένα *όνομα*, μια *τιμή* και έναν *τύπο*.

- Πριν να χρησιμοποιηθεί μια μεταβλητή, πρέπει να δηλωθεί. Η **δήλωση μιας μεταβλητής** (variable declaration) πληροφορεί τον μεταγλωττιστή για το **όνομα** της νέας μεταβλητής και τον **τύπο των δεδομένων** που θα μπορεί να φιλοξενήσει η μεταβλητή κατά την εκτέλεση του προγράμματος:

π.χ.
```
<τύπος-δεδομένων>   <όνομα-μεταβλητής>;
int                 n1;
```

- Κανόνες **ονομασίας** μεταβλητών:
  - ξεκινάει με γράμμα ή _
  - τα κεφαλαία και πεζά γράμματα είναι διαφορετικά (π.χ. AB, Ab είναι διαφορετικά ονόματα)
  - δεν μπορεί να είναι λέξη-κλειδί (keywords) (π.χ. int)
  - μπορούν να έχουν οποιοδήποτε μήκος, αλλά μόνο οι 31 πρώτοι χαρακτήρες είναι σημαντικοί
  - το όνομα θα πρέπει να κάνει σαφές στον αναγνώστη ποια τιμή περιέχει
- Η αρχική τιμή μιας μεταβλητής δεν είναι ορισμένη.

Δήλωση μεταβλητών (2)

```
int n1, n2, total;
```

Αποτέλεσμα εκτέλεσης της εντολής

Μνήμη (μεταβλητές)

n1	n2	total
?	?	?

Η αρχική τιμή κάθε μεταβλητής είναι απροσδιόριστη, και δεν θα πρέπει να κάνετε καμία υπόθεση σχετικά με τις τιμές που μπορεί να περιέχει όταν ξεκινά το πρόγραμμα.

Η φάση εισόδου (1)

```
1.  printf("1st number? ");
2.  n1 = GetInteger();
3.  printf("2nd number? ");
4.  n2 = GetInteger();
```

Γραμμές 1, 3: εμφάνιση **προτρεπτικών μηνυμάτων** (prompt) μέσω των οποίων ο χρήστης ενημερώνεται για το τι απαιτείται από αυτόν.

Γραμμές 2, 4: **εντολές ανάθεσης** (assignment statements), οι οποίες αποθηκεύουν την τιμή που βρίσκεται στα δεξιά του συμβόλου ισότητας στη μεταβλητή που βρίσκεται στα αριστερά του.

- Το δεξιό μέλος είναι μια κλήση της συνάρτησης **GetInteger** της βιβλιοθήκης **simpio**, η οποία:
  - (1) περιμένει από τον χρήστη να εισαγάγει έναν ακέραιο αριθμό χρησιμοποιώντας το πληκτρολόγιο και
  - (2) επιστρέφει την τιμή που πληκτρολόγησε ο χρήστης πίσω στο κυρίως πρόγραμμα και την αναθέτει στη μεταβλητή **n1** ή **n2** αντίστοιχα.

Η φάση εισόδου (2)

```
printf("1st number? ");
n1 = GetInteger();
printf("2nd number? ");
n2 = GetInteger();
```

Αποτέλεσμα εκτέλεσης των εντολών

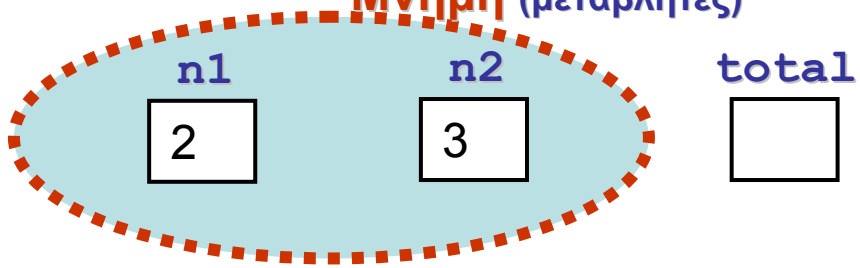
(με πράσινο χρώμα παρουσιάζονται τα δεδομένα που εισάγει ο χρήστης)

Οθόνη

```
This program adds two numbers
1st number? 2 ↵
2nd number? 3 ↵
```

↵ = enter

Μνήμη (μεταβλητές)



Η φάση υπολογισμού

```
total = n1 + n2;
```

- Ο υπολογισμός καθορίζεται γράφοντας μια **παράσταση** (expression) που υποδεικνύει τις απαραίτητες πράξεις:

  n1 + n2

- Το αποτέλεσμα της παράστασης αποθηκεύεται με μια εντολή ανάθεσης στη μεταβλητή **total**, έτσι ώστε να χρησιμοποιηθεί στη συνέχεια του προγράμματος.

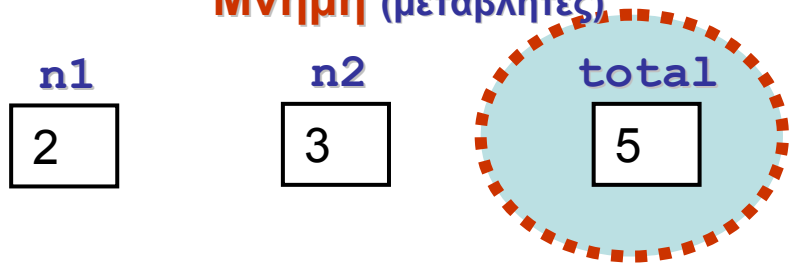
Αποτέλεσμα εκτέλεσης των εντολών

(με πράσινο χρώμα παρουσιάζονται τα δεδομένα που έχει εισάγει ο χρήστης)

Οθόνη

```
This program adds two numbers
1st number? 2
2nd number? 3
```

Μνήμη (μεταβλητές)



Η φάση εξόδου

Η φάση εξόδου συνίσταται στην εμφάνιση του υπολογισμένου αποτελέσματος χρησιμοποιώντας τη συνάρτηση **printf**

```
printf("The total is %d.\n", total);
```

- Η **printf** εμφανίζει στην οθόνη καθέναν από τους χαρακτήρες του αλφαριθμητικού της ορίσματος, μέχρι να φτάσει στο σύμβολο του ποσοστού, οπότε κάνει κάτι ιδιαίτερο.
- Το σύμβολο **%** και το γράμμα που το ακολουθεί ονομάζεται **κωδικός μορφοποίησης** (format code).
- Στο παράδειγμα μας ο κωδικός **%d** καθορίζει ότι η έξοδος θα πρέπει να εμφανιστεί ως **δεκαδικός ακέραιος** (decimal integer).

Αποτέλεσμα εκτέλεσης των εντολών

(με πράσινο χρώμα παρουσιάζονται τα δεδομένα που έχει εισάγει ο χρήστης)

Οθόνη

```
This program adds two numbers
1st number? 2
2nd number? 3
The total is 5.
```

Μνήμη (μεταβλητές)

n1	n2	total
2	3	5

Περισσότερα για τους κωδικούς μορφοποίησης

- Ο κωδικός μορφοποίησης ενεργεί ως δεσμευτικό θέσης για μια τιμή, η οποία εισάγεται σε εκείνο το σημείο κατά την έξοδο.
- Το γράμμα του κωδικού μορφοποίησης καθορίζει τη μορφοποίηση της εξόδου.
- Η συνάρτηση **printf** μπορεί να εμφανίσει οποιοδήποτε πλήθος τιμών δεδομένων ως τμήμα της εξόδου.
- Για κάθε ακέραιη τιμή που θέλετε να εμφανιστεί ως τμήμα της εξόδου, θα πρέπει να συμπεριλάβετε τον κωδικό **%d** στο αλφαριθμητικό που χρησιμοποιείται ως πρώτο όρισμα κατά την κλήση της **printf**.

```
printf(" %d + %d = %d.\n", n1, n2, total);
```

Αποτέλεσμα εκτέλεσης των εντολών

(με πράσινο χρώμα παρουσιάζονται τα δεδομένα που εισάγει ο χρήστης)

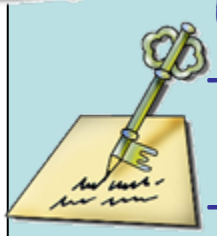
Οθόνη

```
This program adds two numbers
1st number? 2
2nd number? 3
2 + 3 = 5.
```

Μνήμη (μεταβλητές)

n1	n2	total
2	3	5

Τύποι δεδομένων



Ένας **τύπος δεδομένων** (data type) ορίζεται από δύο χαρακτηριστικά:

- ένα σύνολο τιμών ή **πεδίο ορισμού** (domain), δηλ. το σύνολο των τιμών που αποτελούν στοιχεία αυτού του τύπου
- ένα **σύνολο πράξεων**, το οποίο περιλαμβάνει τα εργαλεία που έχετε στη διάθεσή σας προκειμένου να χειριστείτε τις τιμές του συγκεκριμένου τύπου.

Τύπος δεδομένων	Πεδίο ορισμού	Σύνολο πράξεων
`int`	όλοι οι ακέραιοι (…, -2, -1, 0, 1, 2, …) μέχρι τα όρια που καθορίζονται από το υλικό της μηχανής (συνήθως -32767…32767)	πρόσθεση, αφαίρεση, πολ/μός, διαίρεση
`long`	"μεγαλύτερος" ακέραιος – Είσοδος: `GetLong()` Κωδικός μορφοποίησης: `%ld`	≠
`χαρακτήρας`	το σύνολο των συμβόλων που εμφανίζονται στο πληκτρολόγιο ή που μπορούν να εμφανιστούν στην οθόνη	π.χ. σύγκριση χαρακτήρων

Δεδομένα κινητής υποδιαστολής

Αριθμοί κινητής υποδιαστολής (floating-point numbers), στις περισσότερες γλώσσες προγραμματισμού, ονομάζονται οι αριθμοί που περιλαμβάνουν δεκαδικό κλασματικό μέρος (*και χρησιμοποιούνται στα μαθηματικά για την προσέγγιση των πραγματικών αριθμών*).

- **Τύπος δεδομένων**: **`double`**
- **Είσοδος/διάβασμα δεδομένων** τύπου **`double`**: καλείται η συνάρτηση **`GetReal`** της βιβλιοθήκης **`simpio`** που επιστρέφει μια τιμή τύπου **`double`**
- **Εμφάνιση τιμής** τύπου **`double`** σε μια **`printf`**: χρησιμοποιείται ο κωδικός μορφοποίησης **`%g`**

Άθροισμα αριθμών κινητής υποδιαστολής

```
/*
 * Αρχείο: add2f.c
 * --------------
 * Αυτό το πρόγραμμα διαβάζει δύο αριθμούς κινητής υποδιαστολής,
 * τους προσθέτει, και εμφανίζει το άθροισμά τους.
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
    double n1, n2, total;

    printf("This program adds two floating-point numbers.\n");
    printf("1st number? ");
    n1 = GetReal();
    printf("2nd number? ");
    n2 = GetReal();
    total = n1 + n2;
    printf("The total is %g\n", total);
}
```

```
This program adds two floating-point numbers
1st number? 2.0 ↵
2nd number? 3.6 ↵
The total is 5.6.
```

Αλφαριθμητικά δεδομένα

Ο τύπος του αλφαριθμητικού (ακολουθία χαρακτήρων) δεν έχει οριστεί από τους σχεδιαστές της C. Ωστόσο, έχει οριστεί στη βιβλιοθήκη **genlib** και μπορούμε να τον χρησιμοποιούμε όπως τους τύπους **int** και **double**.

- **Τύπος δεδομένων**: **string**

- **Είσοδος/διάβασμα δεδομένων** τύπου **string**: καλείται η συνάρτηση **GetLine** της βιβλιοθήκης **simpio** που διαβάζει μια ολόκληρη γραμμή και την επιστρέφει ως αλφαριθμητικό

- **Εμφάνιση τιμής** τύπου **string** σε μια **printf**: χρησιμοποιείται ο κωδικός μορφοποίησης **%s**

Παράδειγμα με αλφαριθμητικά

```
/*
 * Αρχείο: greeting.c
 * ------------------
 * Αυτό το πρόγραμμα εμφανίζει έναν πιο προσωπικό χαιρετισμό
 * από εκείνον του αρχικού προγράμματος "Hello, world."
 * εισάγοντας το όνομα του χρήστη.
 */

#include <stdio.h>
#include "genlib.h"
#include "simpio.h"

main()
{
    string user;

    printf("What is your name? ");
    user = GetLine();
    printf("Hello, %s.\n", user);
}
```

```
What is your name? Eric ↵
Hello, Eric.
```

Παραστάσεις

Μια **παράσταση** (expression) αποτελείται από:

- **όρους** (terms) που παριστάνουν τιμές δεδομένων και
- **τελεστές** (operators) που υποδεικνύουν μια υπολογιστική πράξη.

Ένας **όρος** μπορεί να είναι:

- Μια **σταθερά** (constant): κάθε συγκεκριμένη τιμή δεδομένων που εμφανίζεται ως τμήμα του κειμένου του προγράμματος και μπορεί να είναι ακέραια, κινητής υποδιαστολής, αλφαριθμητική.
- Μια μεταβλητή
- Μια κλήση συνάρτησης, όπως για παράδειγμα η `GetInteger()`
- Μια παράσταση σε παρενθέσεις: οι παρενθέσεις χρησιμοποιούνται για να υποδείξουν τη σειρά των πράξεων. Μια παράσταση σε παρενθέσεις είναι από μόνη της ένας όρος που αντιμετωπίζεται από τον μεταγλωττιστή ως μια μονάδα που πρέπει να υπολογιστεί πριν να συνεχιστεί ο υπολογισμός.

Τελεστές και τελεστέοι

Αριθμητικοί τελεστές:

+	πρόσθεση
−	αφαίρεση

(ή άρνηση, αν γράφεται χωρίς τιμή στα αριστερά του: - x. Σε αυτή την περίπτωση ονομάζεται **μονομελής τελεστής** (unary operator) γιατί εφαρμόζεται σε ένα μόνο τελεστέο. Όλοι οι υπόλοιποι τελεστές ονομάζονται **διμελείς**.)

*	πολλαπλασιασμός
/	διαίρεση
%	τελεστής υπολοίπου

Συνδυασμοί ακεραίων & αριθμών κιν. υποδιαστολής

- Στη C, οι τιμές των τύπων **int** και **double** μπορούν να συνδυαστούν ελεύθερα.
- Αν χρησιμοποιήσουμε ένα διμελή τελεστή με δύο τιμές τύπου **int** τότε το αποτέλεσμα θα είναι τύπου **int**.
- Αν, έστω και μία από τις δύο τιμές, είναι τύπου **double** τότε το αποτέλεσμα θα είναι τύπου **double** .

Παραδείγματα

- 9 / 4 = 2 Προσοχή: Το υπόλοιπο αγνοείται!!!
- 9.0 / 4 = 2.25
- 9 / 4.0 = 2.25
- 9.0 / 4.0 = 2.25

Ο τελεστής υπολοίπου %

Ο τελεστής του υπολοίπου **%** απαιτεί να είναι και οι δύο τελεστέοι τύπου **`int`** και επιστρέφει το υπόλοιπο της διαίρεσης του πρώτου τελεστέου με τον δεύτερο.

Παραδείγματα

0 % 4 = 0

1 % 4 = 1

4 % 4 = 0

19 % 4 = 3

20 % 4 = 0

2001 % 4 = 1

- Ο τελεστής % χρησιμοποιείται συχνά προκειμένου να ελεγχθεί αν ένας αριθμός διαιρείται με κάποιον άλλο.
- Η χρήση του τελεστή % με αρνητικούς τελεστέους θα πρέπει να αποφεύγεται γιατί η C συμπεριφέρεται με απρόβλεπτο τρόπο.

Κανόνες προτεραιότητας

Όταν σε μία παράσταση υπάρχουν περισσότεροι από ένας τελεστές τότε η σειρά εκτέλεσης των πράξεων είναι πολύ σημαντική και καθορίζεται από τους **κανόνες προτεραιότητας** (rules of precedence):

- Εφαρμόζονται πρώτα τυχόν μονομελείς τελεστές **-**
- Στη συνέχεια εφαρμόζονται οι πολλαπλασιαστικοί τελεστές *****, **/** και **%**.
- Τέλος, εφαρμόζονται οι προσθετικοί τελεστές **+** και **-**.

Παρατηρήσεις:

- Μπορείτε να καθορίζετε τη σειρά εκτέλεσης των πράξεων περικλείοντας μεμονωμένες υποπαραστάσεις σε **παρενθέσεις**.
- Αν στον ίδιο τελεστέο εφαρμόζονται δύο τελεστές με την ίδια προτεραιότητα τότε εκτελούνται οι πράξεις από αριστερά προς τα δεξιά.

Εφαρμογή κανόνων προτεραιότητας

8 * (**7 - 6** + 5) % (4 + 3 / 2) - 1

8 * (**1 + 5**) % (4 + 3 / 2) - 1

8 * 6 % (4 + **3 / 2**) - 1

8 * 6 % (**4 + 1**) - 1

8 * 6 % 5 - 1

48 % 5 - 1

3 - 1

2

Αυτόματη μετατροπή τύπου

Όταν σε ένα πρόγραμμα C συνδυάζετε τιμές διαφορετικών τύπων τότε η C μετατρέπει αυτόματα τις τιμές ενός τύπου σε ένα άλλο συμβατό τύπο, διεργασία γνωστή ως **αυτόματη μετατροπή τύπου** (automatic type conversion).

Παραδείγματα

- πριν να εκτελεστεί η πρόσθεση

  `1 + 2.3`

  ο ακέραιος 1 μετατρέπεται εσωτερικά στον αριθμό κινητής υποδιαστολής 1.0 και εκτελείται η πρόσθεση

  `1.0 + 2.3`

-
```
double total;
total = 0;
```
ο αριθμός 0 κατά τη διεργασία ανάθεσης θα μετατραπεί σε αριθμό τύπου **double**, δηλ. **0.0**

-
```
int n;
n = 1.9999;
```
ο αριθμός 1.9999 κατά τη διεργασία ανάθεσης θα μετατραπεί στον αριθμό τύπου **int 1**.

Έχουμε, δηλαδή, απαλοιφή του δεκαδικού κλάσματος, διεργασία γνωστή ως **αποκοπή** (truncation)

Τελεστής ρητής μετατροπής

Στη C, μπορείτε να ζητήσετε ρητή μετατροπή τύπου μιας τιμής χρησιμοποιώντας τον μονομελή **τελεστή ρητής μετατροπής** (type cast), ο οποίος αποτελείται από τον επιθυμητό τύπο περικλειόμενο σε παρενθέσεις και ακολουθούμενο από την τιμή που θέλουμε να μετατρέψουμε.

Παράδειγμα

```
int num, den;

double quotient;

quotient = num / den;
```

Λάθος!!!! Το κλασματικό μέρος απαλείφεται

π.χ. 9 / 4 = 2 και όχι 2.25

```
quotient = num / (double) den;

quotient = (double) num / den;

quotient = ((double) num) / den;
```

Σωστό