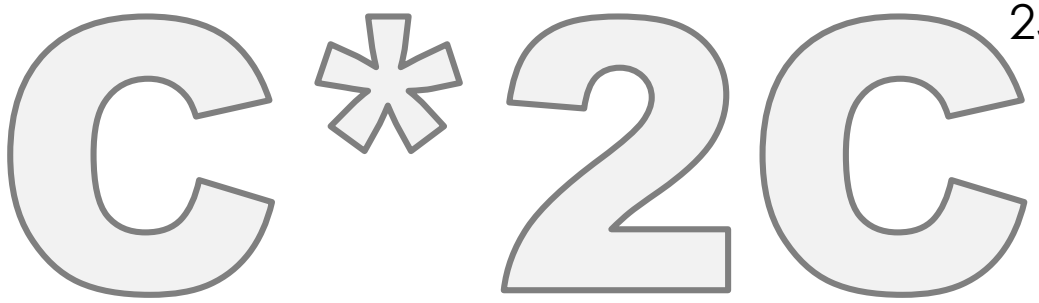
Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #6

25 Απριλίου 2024



Παναγιώτης Παύλου

c-programming-24@allos.gr

Από την $\mathbb{C}^*$ στην $\mathbb{C}$

Αξιοποιώντας τη θεωρία…

Η γλώσσα C

Διαφορές με την απλοποιημένη C και
το περιβάλλον ανάπτυξης (IDE) CLion

IDE : Το περιβάλλον ανάπτυξης

Στη C, αλλά και στις περισσότερες υψηλού επιπέδου γλώσσες (χωρίς να είναι αυτό απαραίτητο), όλη η διαδικασία της ανάπτυξης ενός προγράμματος, πραγματοποιείται μέσα σε ένα λογισμικό το οποίο ονομάζεται Integrated Development Environment (δηλαδή Ολοκληρωμένο Περιβάλλον Ανάπτυξης).

Αυτό περιλαμβάνει τουλάχιστον:

1. τον **editor** (τον επεξεργαστή κειμένου) όπου γράφεται ο κώδικας
2. τον **builder** όπου γίνεται η μετατροπή του κώδικα σε εκτελέσιμο
3. τη γραμμή εντολών όπου γίνεται η εκτέλεση του κώδικα
4. τον **debugger**, όπου βοηθά τον εντοπισμό και την επίλυση σφαλμάτων (bugs) στον κώδικα

Διαθέσιμα IDEs

Για τη C υπάρχουν πολλά διαθέσιμα IDEs. Μερικά από αυτά είναι:

- JetBrains CLion
- Bloodshed C++
- Netbeans
- Codeblocks
- Eclipse
- Microsoft Visual Studio
- VSCode

Από αυτά, στο μάθημα θα χρησιμοποιούμε το **CLion**, το οποίο είναι ένα επαγγελματικό IDE.

Υπάρχει ήδη εγκατεστημένο στο PC-Lab της Σχολής, αλλά μπορείτε (και πρέπει) να το εγκαταστήσετε σε έναν υπολογιστή σας ώστε να μπορείτε να κάνετε τις εργασίες σας και από το σπίτι.

CLion : Εγκατάσταση και 1η χρήση

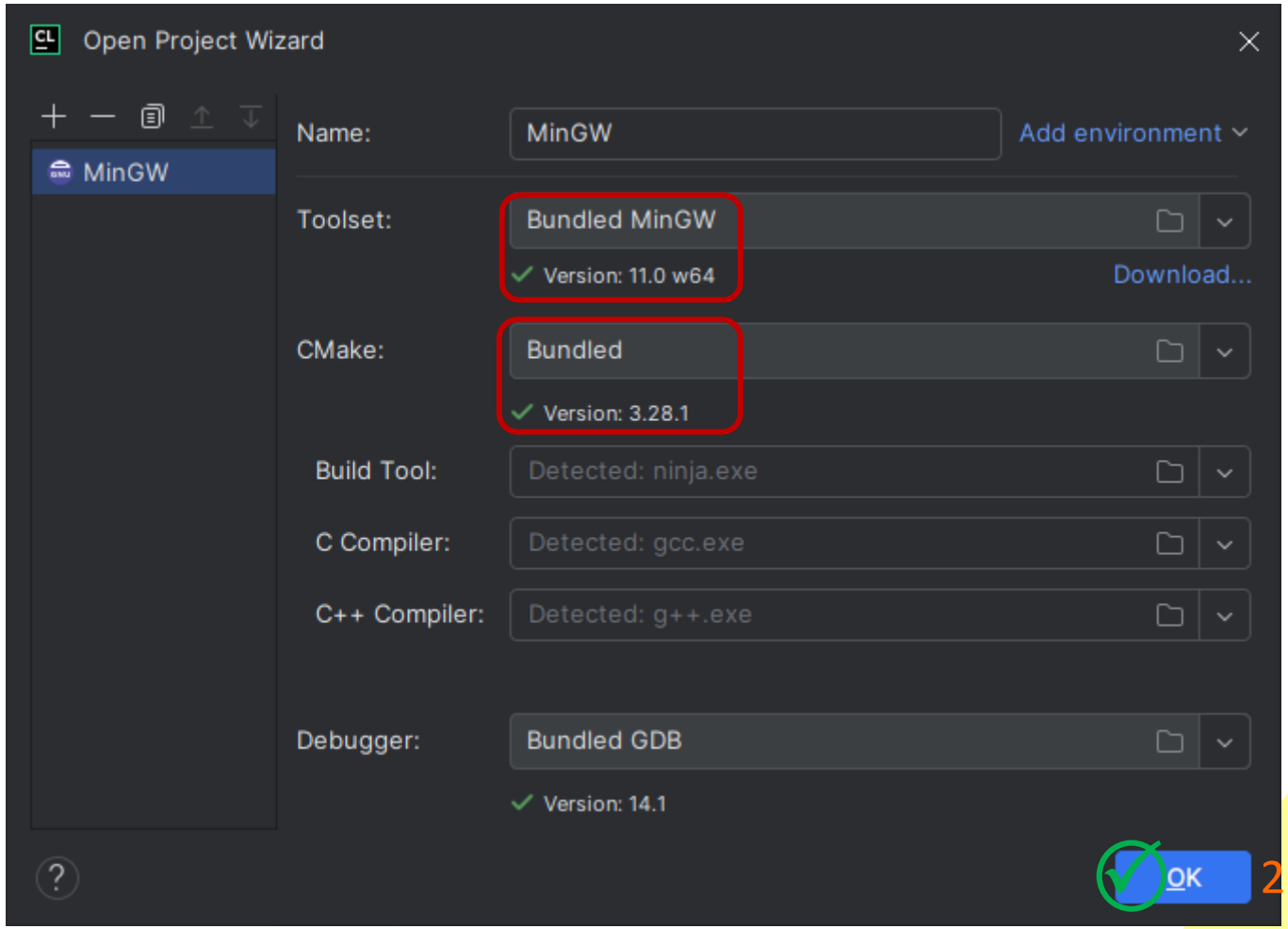
Η εγκατάσταση του CLion γίνεται με τα ακόλουθα βήματα:

1. Δημιουργία κωδικού χρήστη στην JetBrains, χρησιμοποιώντας το e-mail (mc######@mail.ntua.gr) του ΕΜΠ
2. Λήψη και εγκατάσταση (οι προεπιλεγμένες ρυθμίσεις είναι επαρκείς)
3. Ρύθμιση κατά την πρώτη χρήση (δεν απαιτούνται κάποια plugins)
4. Ενεργοποίηση του στον υπολογιστή σας (επιτρέπεται μόνο ένας Η/Υ ανά φοιτητή)
5. Δεν απαιτείται ρύθμιση του toolchain, μόνο επιβεβαίωση

CLion : Επιβεβαίωση toolchain

Το μόνο που απαιτείται είναι κατά τη δημιουργία του 1ου project στο παράθυρο που φαίνεται και εδώ, να βεβαιωθείτε ότι τα Toolset και Cmake είναι αυτά που γράφουν Bundled και ότι έχουν το πράσινο check από κάτω τους. Ο Debugger δεν πειράζει εάν φαίνεται κόκκινος.

Τέλος πατάτε το OK κάτω δεξιά.



CLion : Οι βασικές περιοχές του IDE

Κεντρικό Menu

Περιλαμβάνει όλες τις επιλογές του IDE. Είναι όλο και πιο χρήσιμο καθώς προοδεύει ο προγραμματιστής

Περιοχή Project

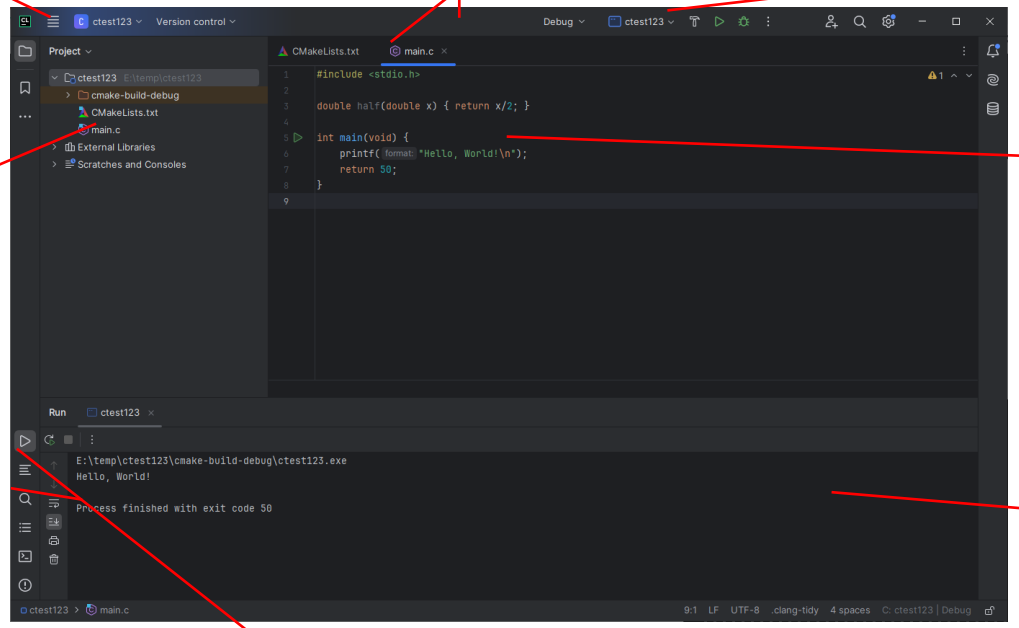
Όλα τα σχετικά και απαραίτητα αρχεία Περιλαμβάνουν και το αρχείο του κώδικα (εδώ main.c)

Καρτέλες Επεξεργαστή

Εμφανίζουν τα αρχεία που είναι ανοιχτά στον επεξεργαστή. Το τρέχον αρχείο ξεχωρίζει.

Γραμμή εκτέλεσης

Περιέχει επιλογή του χτισίματος, της εκτέλεσης, του debugging, κ.α.



Περιοχή Επεξεργαστή

Εμφανίζεται ο κώδικας του προγράμματος. Έχει διάφορα βοηθητικά χαρακτηριστικά όπως είναι ο χρωματισμός των εντολών (syntax highlighting) και άλλα.

Περιοχή Μηνυμάτων & Αποτελεσμάτων

Εμφανίζει τα μηνύματα κατά το Building ή τα μηνύματα κατά την εκτέλεση κ.α.

Καρτέλες Επιλογής

Από αυτές τις καρτέλες επιλέγεται τι εμφανίζει κάθε στιγμή η περιοχή αποτελεσμάτων. Τα δύο βέλη υποδεικνύουν τις δύο πιο συχνά χρησιμοποιούμενες καρτέλες. Η αριστερή εμφανίζει τα μηνύματα κατά τη διάρκεια του build ενώ η δεξιά εμφανίζει τα αποτελέσματα της εκτέλεσης.

CLion : Βασικά στοιχεία του editor

Ο editor (επεξεργαστής) του κώδικα είναι ένας κειμενογράφος απλών κειμένων (όπως το Σημειωματάριο/Notepad των Windows) με αρκετές πρόσθετες δυνατότητες. Για παράδειγμα:

- Syntax highlighting : Χρωματισμός των διαφόρων σημείων του κώδικα ώστε να βοηθά στον εντοπισμό λέξεων κλειδιών, σφαλμάτων, κλπ
- IntelliSense : Αυτόματη συμπλήρωση λέξεων κλειδιών, ονομάτων (identifiers), παραμέτρων συναρτήσεων, κατά την πληκτρολόγηση
- Μετονομασία μεταβλητών, όπου αυτόματα μετονομάζονται όλες οι εμφανίσεις τους
- Προειδοποιήσεις για πιθανά σφάλματα
- Προτάσεις για βελτίωση του κώδικα
- Αυτόματη μορφοποίηση του κώδικα

CLion : Δημιουργία ενός Project

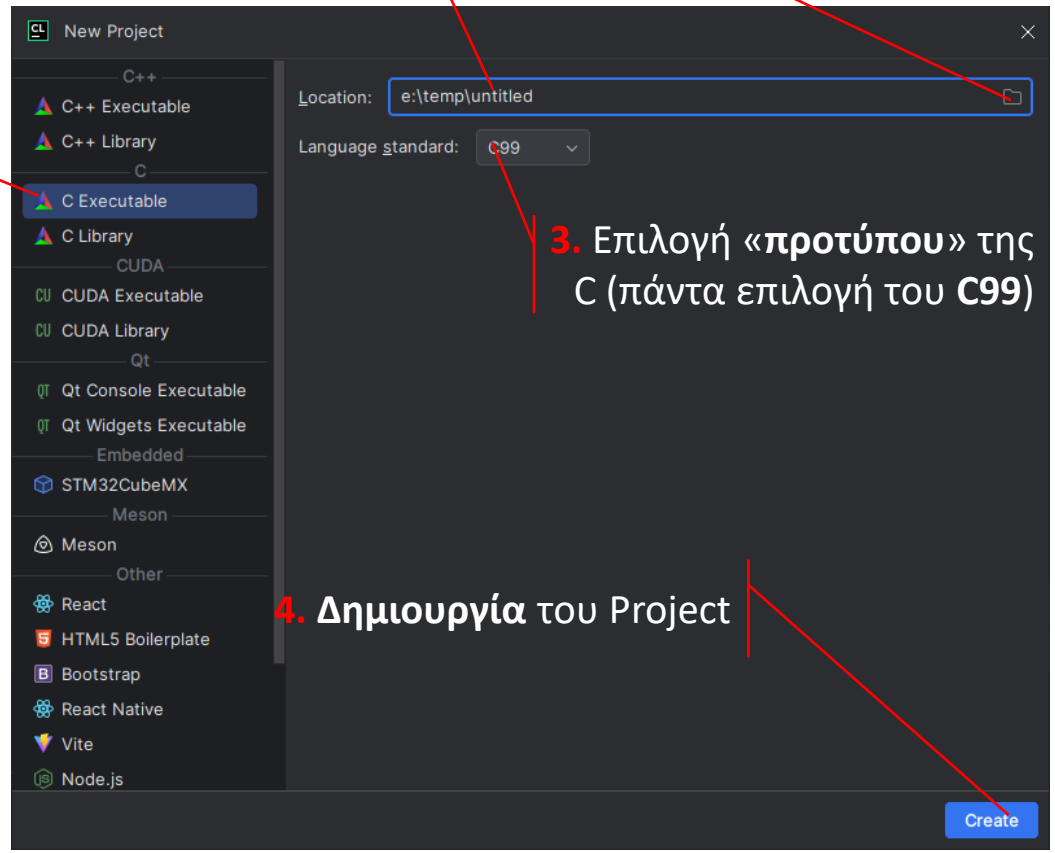
2. Πληκτρολόγηση ή επιλογή του **φακέλου αποθήκευσης** του Project

1. Επιλογή «**τι παράγει**» το project

Επιλέγοντας από το μενού του CLion, ***File > New Project*** εμφανίζεται το διπλανό πλαίσιο διαλόγου (**dialog**).

Το κάθε project αποθηκεύεται σε ένα φάκελο στον δίσκο. Εκεί βρίσκονται όλα τα απαραίτητα αρχεία για αυτό το project.

Στο κάθε project αντιστοιχεί τουλάχιστον ένας «στόχος» (**target**) που είναι το αποτέλεσμα του build. Συνήθως αυτός ο στόχος είναι ένα εκτελέσιμο αρχείο.



Το 1ο πρόγραμμα σε C

```
#include <stdio.h>


int main() {
    printf("Hello world!\n");
    return 0;
}
```

Δίπλα παρουσιάζεται το αντίστοιχο εισαγωγικό πρόγραμμα της C* γραμμένο στη γλώσσα C.

Είναι και πάλι ένα απλό αρχείο κειμένου. Για να εκτελεστεί πρέπει να μετατραπεί σε εκτελέσιμο αρχείο.

Τη διαδικασία αυτή την αποκαλούμε συνήθως compilation, όμως αυτό είναι ανακριβές. Το compilation είναι μόνο ένα (το κεντρικό) βήμα της. Όλη η διαδικασία ονομάζεται building και περιλαμβάνει (απλουστευτικά) τα παρακάτω βήματα:

Preprocessing – Μετασχηματίζει το κείμενο του κώδικα και πάλι σε κείμενο

Compiling – Μετατρέπει το μετασχηματισμένο κείμενο σε γλώσσα μηχανής

Linking – Συνδέει τη γλώσσα μηχανής που παράχθηκε, μαζί με αυτή των βιβλιοθηκών που χρειάζονται, καθώς και με κάποιον κώδικα εκκίνησης, σε ένα εκτελέσιμο αρχείο.

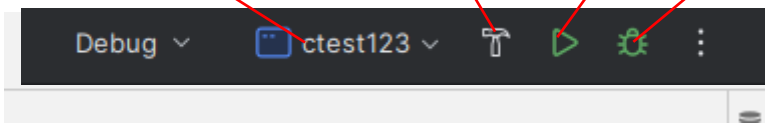
CLion : Χτίσιμο και Εκτέλεση

Για το **χτίσιμο (build)** του προγράμματος πατώ εδώ

Όταν υπάρχουν περισσότεροι από έναν εκτελέσιμοι στόχοι επιλέγω από εδώ

Για την **εκτέλεση** του προγράμματος πατώ εδώ

Για την εκτέλεση με **debugging** του προγράμματος πατώ εδώ



Όταν επιλέγει ο χρήστης **εκτέλεση** (**run**) τότε, εφόσον έχει αλλάξει ο κώδικας από το προηγούμενο build, αυτόματα γίνεται πρώτα **build** το project και κατόπιν ξεκινά η εκτέλεση.
Τα μηνύματα του **build** εμφανίζονται στο κάτω μέρος του IDE στην καρτέλα **Messages**.
Ενώ τα μηνύματα της **εκτέλεσης** του προγράμματος εμφανίζονται στην καρτέλα **Run**.

Δεδομένα, Τιμές & Μεταβλητές

Πως παριστάνονται στον υπολογιστή και στην C

Το μοντέλο της μνήμης

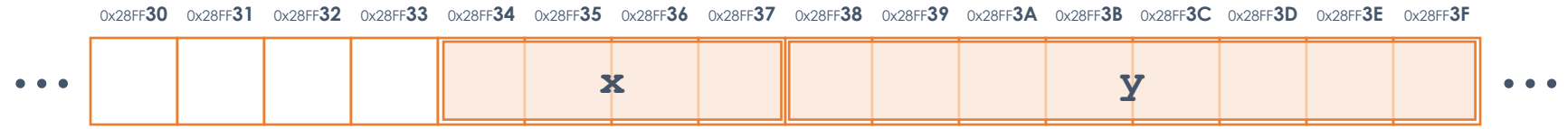
Η μνήμη του υπολογιστή μοντελοποιείται ως μία σειρά από "κουτάκια" που ονομάζονται **θέσεις μνήμης**. Η κάθε θέση μνήμης είναι μεγέθους 1byte, δηλαδή μπορεί να αποθηκεύσει μία τιμή των 8bits.

Οι θέσεις μνήμης έχουν συνεχόμενη αρίθμηση ξεκινώντας από το 0. Ο αύξων αριθμός που αντιστοιχεί στην κάθε θέση μνήμης ονομάζεται **διεύθυνση της θέσης μνήμης**.

Η διεύθυνση της θέσης μνήμης συνηθίζεται να γράφεται στο **δεκαεξαδικό** σύστημα αρίθμησης.

Αποθήκευση μεταβλητών στη μνήμη

Ήδη γνωρίζουμε ότι οι μεταβλητές αποθηκεύονται στη μνήμη του υπολογιστή. Στην πραγματικότητα η κάθε μεταβλητή χρειάζεται διαφορετικό αριθμό από bits για την αποθήκευσή της.



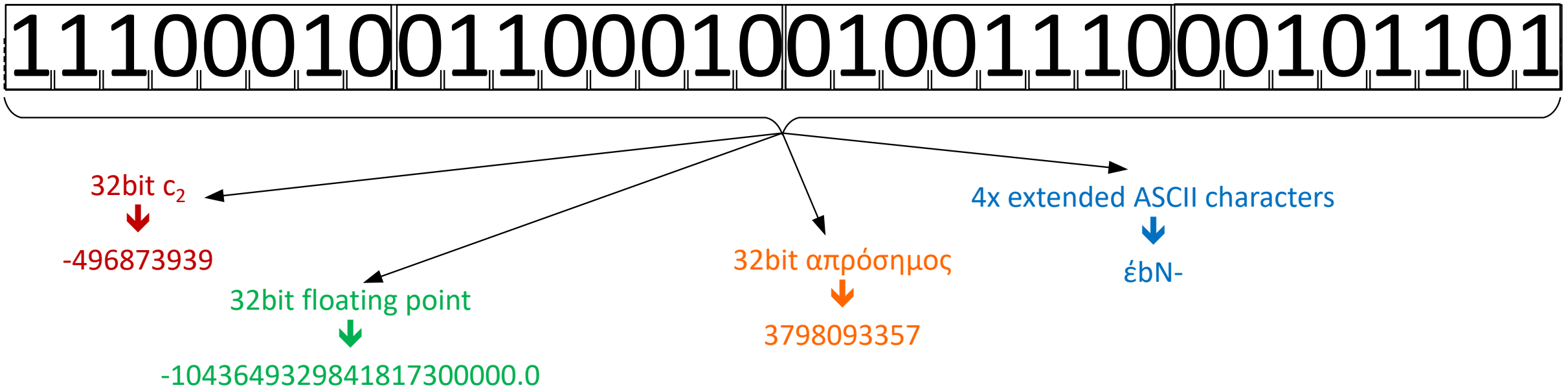
Οι μεταβλητές σχεδόν πάντα απαιτούν περισσότερες από μία **διαδοχικές** θέσεις μνήμης.

Τα bits αυτών των bytes, συνολικά αποτελούν το περιεχόμενο της κάθε μεταβλητής και **εξετάζονται ενιαία, ως ένας δυαδικός αριθμός** με π.χ. 32 (**x**) ή 64 (**y**) ψηφία (στην περίπτωση του σχήματος).

Μετάφραση της δυαδικής πληροφορίας

Η δυαδική πληροφορία που βρίσκεται στη μνήμη του Η/Υ από μόνη της «δεν λέει τίποτα».

Χρειάζεται να «γνωρίζει» κάπως ο Η/Υ (ο επεξεργαστής) το πώς πρέπει να την μεταφράσει.



Το ποια θα είναι η κωδικοποίηση της πληροφορίας, ο προγραμματιστής, στη γλώσσα μηχανής το καθορίζει επιλέγοντας την κατάλληλη εντολή που θα χρησιμοποιηθεί, ενώ στη C και άλλες υψηλού επιπέδου compiled γλώσσες, το καθορίζει επιλέγοντας τον κατάλληλο **τύπο των δεδομένων**.

Εγγενείς (Native) τύποι δεδομένων

Ο τύπος των δεδομένων καθορίζει την αποκωδικοποίηση μιας πληροφορίας σε δυαδική μορφή. Η κωδικοποίηση των βασικών τύπων δεδομένων που αφορούν αριθμούς (ακέραιους και πραγματικούς) καθορίζεται από τις εντολές της γλώσσας μηχανής, δηλαδή από τον ίδιο τον επεξεργαστή. Αυτό συμβαίνει επειδή η κάθε εντολή της γλώσσας μηχανής ενεργοποιεί τα αντίστοιχα εσωτερικά κυκλώματα του επεξεργαστή, τα οποία λειτουργούν με συγκεκριμένο τρόπο/συνδεσμολογία, ώστε να επιτελούν συγκεκριμένη κωδικοποίηση.

Οι τύποι αυτοί που υποστηρίζονται από τον επεξεργαστή ονομάζονται και εγγενείς (native).

Εγγενείς τύποι δεδομένων και όρια

Τύπος Δεδομένων	Bits	Ελάχιστη	Μέγιστη
Ακέραιες **Απρόσημες** Τιμές			
`unsigned char`	8	0	255
`unsigned short`	16	0	65535
`unsigned int`	32	0	4294967295
`unsigned long`	32	0	4294967295
`unsigned long long`	64	0	18446744073709551615
Ακέραιες **Προσημασμένες** Τιμές			
`char`	8	-128	127
`short`	16	-32768	32767
`int`	32	-2147483648	2147483647
`long`	32	-2147483648	2147483647
`long long`	64	-9223372036854775808	9223372036854775807
Δεκαδικές Τιμές (κινητής υποδιαστολής, πάντα προσημασμένες)			
`float`	32	±1.17549e-038	± 3.40282e+038
`double`	64	±2.22507e-308	± 1.79769e+308

Τιμές στη C για τους native τύπους δεδομένων

Μέσα σε ένα πρόγραμμα C οι τιμές των native τύπων δεδομένων μπορούν να γραφούν με έναν από τους ακόλουθους τρόπους.

- Ακέραιοι αριθμοί (προσημασμένοι ή απρόσημοι)
  - **Δεκαδικό** σύστημα αρίθμησης : **[πρόσημο]ψηφία(0-9)**
    π.χ. **+123 1952384 -2435 0**
  - **Οκταδικό** σύστημα αρίθμησης : **[πρόσημο]0ψηφία(0-7)**
    π.χ. **+0123 0153425 -05234 0**
  - **Δεκαεξαδικό** σύστημα αρίθμησης : **[πρόσημο]0xψηφία(0-9 ή A-F ή a-f)**
    π.χ. **+0x53A4F -0x123 +0x0FFF 0x0**

Σημειώστε ότι αυτό που αλλάζει είναι ο τρόπος έκφρασης στο κείμενο του κώδικα και όχι οι το δυαδικό περιεχόμενο που αντιστοιχεί στον ίδιο αριθμό. Π.χ. το `0x100` και το `256` **είναι** ο ίδιος αριθμός, άρα είναι ίδια η δυαδική αναπαράσταση, ανεξαρτήτως του σε ποια βάση δηλώθηκε.

Τιμές στη C για τους native τύπους δεδομένων

- Κινητής υποδιαστολής (πάντα προσημασμένοι)
  - Απλή γραφή :

    [πρόσημο]ψηφία(0-9).ψηφία(0-9)

    υποχρεωτικά υπάρχουν η υποδιαστολή (που είναι τελεία, όχι κόμμα) και ένα ψηφίο

  - Επιστημονική ή εκθετική γραφή :

    [πρόσημο] ψηφία(0-9).ψηφία(0-9)e[πρόσημο]ψηφία(0-9)

    υποχρεωτικά υπάρχουν το **e** (ή **E**), στα αριστερά του η βάση γράφεται σε απλή γραφή και στα δεξιά ο εκθέτης ως προσημασμένος ακέραιος. Σε όλη την έκταση του αριθμού δεν υπάρχουν καθόλου κενά.

Σωστό	Λάθος
1234.5678	1234
-1000.0001	1234,5678
+123.345	12 .34
-.34	11.22.33
123.	.

Σωστό	Λάθος
-12.34e10	12e 3
13.1002e-30	1e8.2
12.32E+4	
.132E+2	
1e3	

ΣΗΜΕΙΩΣΗ : Οι αγκύλες [ ] εδώ συμβολίζουν ότι το πρόσημο είναι προαιρετικό το να μπει (για τους θετικούς αριθμούς). Δεν πληκτρολογούνται και δεν εμφανίζονται πουθενά όπως βλέπετε και στα παραδείγματα στα δεξιά!

Μεταβλητές

Στην **C***, οι μεταβλητές δηλώνονταν ως εξής:

```
let x = 1234;
```

Επειδή η C είναι compiled γλώσσα, δηλαδή από των κώδικα παράγεται γλώσσα μηχανής, θα πρέπει να είναι γνωστός ο τύπος δεδομένων αυτής της μεταβλητής εκ των προτέρων. Γι'αυτό αντικαθιστούμε τη λέξη κλειδί let της C* με το κατάλληλο λεκτικό του προηγούμενου πίνακα.

Δηλαδή για μία προσημασμένη ακέραια τιμή γράφουμε:

```
int x = 1234;
```

Ο τύπος δεδομένων χαρακτηρίζει την μεταβλητή, ενώ η πληροφορία ακολουθεί αυτόν τον τύπο δεδομένων.

Ενώ για μία πραγματική μεταβλητή διπλής ακρίβειας γράφουμε:

```
double x = 1234.0;
```

Παρατηρείστε ότι πλέον θα πρέπει
οι σταθερές να γράφονται κατάλληλα ώστε να ταιριάζουν με τον τύπο δεδομένων της μεταβλητής.

Μετατροπή τύπων δεδομένων

Οι πράξεις μεταξύ δύο τιμών του ίδιου τύπου δεδομένων δίνουν αποτέλεσμα του ίδιου τύπου (εκτός από τους τελεστές σύγκρισης που το αποτέλεσμα είναι τύπου bool). Έτσι η πράξη 5/8 δίνει αποτέλεσμα 0 αφού και οι δύο αριθμοί είναι ακέραιοι, οπότε και το αποτέλεσμα είναι ακέραιο.

Για να γίνουν αριθμητικές πράξεις μεταξύ δύο τιμών διαφορετικών τύπων, μετατρέπονται πρώτα σε τιμές του ίδιου τύπου και μάλιστα προς τον τύπο που καλύπτει μεγαλύτερο εύρος τιμών. Για παράδειγμα πράξη μεταξύ `int` και `double` μετατρέπει πρώτα και τις δύο ποσότητες σε double. Κατόπιν γίνεται η πράξη.

Συνήθως αυτή η μετατροπή (από τον «φτωχότερο» στον «πλουσιότερο» τύπο δεδομένων) γίνεται αυτόματα από τον compiler και αυτό θέλει προσοχή από εμάς.

Όταν ο προγραμματιστής θέλει να μετατραπεί μία ποσότητα σε άλλο τύπο δεδομένων, τότε μπορεί να γράψει ακριβώς πριν την ποσότητα, τον επιθυμητό τύπο μέσα σε παρενθέσεις. Η διαδικασία αυτή λέγεται type casting ή απλά casting.

Το δεκαδικό αποτέλεσμα στο παραπάνω παράδειγμα προκύπτει με δύο τρόπους:

```
5.0/8          5/(double)8
```

ή και γενικότερα στην περίπτωση μεταβλητών (όπου δεν είναι δυνατή και η προσθήκη υποδιαστολής)

```
y/(double)x
```

Ορισμός συνάρτησης

Στη C* για τον ορισμό μιας συνάρτησης απλά γράφαμε:

```
function squareOf(x)
{
  return x*x;
}
```

Στη C όμως για τον **ορισμό** (που στο κείμενο του κώδικα θα πρέπει να προηγείται της χρήσης) θα πρέπει να καθορίσουμε τον τύπο δεδομένων κάθε παραμέτρου, καθώς και τον τύπο δεδομένων του αποτελέσματος, έτσι γράφουμε:

```
double squareOf(double x)
{
  return x*x;
}
```

Έτσι κάθε παράμετρος έχει καθορισμένο τύπο δεδομένων και λειτουργεί – κατά τα γνωστά – ως μία τοπική μεταβλητή.

Το ίδιο ισχύει και για το αποτέλεσμα, όμως στην περίπτωση που μία συνάρτηση δεν επιστρέφει κάποια τιμή, δηλαδή η return εκτελείται χωρίς κάποιο δεδομένο, τότε χρησιμοποιείται ο ειδικός τύπος δεδομένων **void**.

Preprocessor (`#include`)

Στη C κατά τη διαδικασία του build, πριν ξεκινήσει το compilation, προηγείται η εκτέλεση του προεπεξεργαστή ή preprocessor. Για τη θέση του στη διαδικασία του build δείτε ένα απλό άρθρο εδώ. Ο σκοπός του είναι να παράγει από το αρχείο του κώδικα ένα νέο αρχείο κώδικα τροποποιημένο βάσει των ψευδοεντολών του.

Προσοχή! Οι ψευδοεντολές του δεν αποτελούν εντολές της C και δεν παράγουν εκτελέσιμο κώδικα. Όλες ξεκινούν με τον χαρακτήρα **#**

Αν και οι δυνατότητές του preprocessor είναι πολλές θα περιοριστούμε σε δύο βασικές. Την `#include` και την `#define` που θα δούμε αργότερα.

Η **`#include`** φέρνει τα περιεχόμενα ενός αρχείου στο σημείο στο οποίο γράφεται, σαν να είχαν γραφεί αυτά εκεί. Γράφεται ως:

```
#include <some_file>   ή   #include "some_file"
```

Οι δύο παραλλαγές με τα < > ή τα εισαγωγικά " " αφορούν το σημείο στο οποίο θα αναζητηθεί το εν λόγω αρχείο. Εν γένει τα αρχεία των τυπικών βιβλιοθηκών συντάσσονται με τα < >, ενώ τυχόν «δικά μας» ή άλλων βιβλιοθηκών με τα εισαγωγικά.

Βιβλιοθήκες

Οι βιβλιοθήκες (libraries), **πολύ απλουστευτικά**, είναι συλλογές από συναρτήσεις, σταθερές και τύπους δεδομένων που συνεργάζονται μεταξύ τους για να πετύχουν κάποιον σκοπό.

Για παράδειγμα η μαθηματική βιβλιοθήκη έχει μαθηματικές σταθερές και μαθηματικές συναρτήσεις. Οι βιβλιοθήκες είναι κώδικας που έχει ήδη γίνει build.

Όταν όμως ένα πρόγραμμα θέλει να χρησιμοποιήσει μία βιβλιοθήκη πρέπει να έχει δύο πράγματα:

1. Τον κώδικα (=το αρχείο της βιβλιοθήκης), το οποίο πρέπει να γίνει link μαζί με τον κώδικα του προγράμματος (δεν θα χρειαστεί για τις ανάγκες του μαθήματος, αλλά για την πληρότητα των σημειώσεων, η προσθήκη μιας βιβλιοθήκης στο project του CLion περιγράφεται εδώ).
2. Τις δηλώσεις των συναρτήσεων, των σταθερών και των τύπων δεδομένων ώστε να μπορεί να γίνει compile ο κώδικας του προγράμματος που τις χρησιμοποιεί.

Όλα τα απαραίτητα περιέχονται σε αρχεία που ονομάζονται header files και έχουν κατάληξη .h , είναι δε, απλά αρχεία της C που περιέχουν όλες τις σταθερές, τους τύπους δεδομένων και τις δηλώσεις όλων των συναρτήσεων της βιβλιοθήκης.

Standard Βιβλιοθήκες της C

Η C, προκειμένου να μπορεί ο κώδικάς της να γίνεται build σε διάφορα συστήματα, έχει κάποιες βιβλιοθήκες που είναι τυποποιημένες και διαθέσιμες σε κάθε σύστημα που μπορεί να κάνει compile τη C. Έτσι ο κώδικάς που χρησιμοποιεί αυτές τις εντολές είναι αυτό που είχαμε ονομάσει cross-platform. Οι βιβλιοθήκες αυτές είναι:

#include	Περιεχόμενο
<**math**.h>	Μαθηματικές σταθερές και συναρτήσεις. Όσες έχουμε δει από τη C*ισχύουν.
<**stdio**.h>	Συναρτήσεις που αφορούν την είσοδο και έξοδο πληροφορίας από το πρόγραμμά μας.
<**stdlib**.h>	Διάφορες συναρτήσεις γενικότερης χρήσης.
<**string**.h>	Συναρτήσεις που αφορούν κείμενα.
<**ctype**.h>	Συναρτήσεις που αφορούν χαρακτήρες κειμένου.
<**time**.h>	Συναρτήσεις που αφορούν χρόνο.

Οι βιβλιοθήκες αυτές με κάποιες από τις πιο συνηθισμένες σταθερές και συναρτήσεις που παρέχουν, περιγράφονται στο εγχειρίδιο του μαθήματος.

Μορφοποιημένη έξοδος (`printf`)

Παρότι στην C οι παράμετροι των συναρτήσεων έχουν πάντα εκ των προτέρων συγκεκριμένο τύπο δεδομένων, αυτό δεν θα μπορούσε να ισχύει για την αντίστοιχη της `smPrint`. Όντως εκεί δεν ξέρουμε τι δεδομένα θα πρέπει να παρουσιαστούν και με τι σειρά, έτσι η αντίστοιχη συνάρτηση η `printf` χρησιμοποιεί έναν άλλο μηχανισμό κλήσης που ξεφεύγει από το πλαίσιο αυτού του μαθήματος.

	Τύπος Δεδομένων
d	**int** δεκαδικό/**d**ecimal
ld	**long**
f	**float**
lf	**double**
i	**int**
o	**int** οκταδικό/**o**ctal
x	**int** δεκαεξαδικό/he**x**adecimal

Πρακτικά πάντως, για να την χρησιμοποιήσουμε πρέπει να ξέρουμε μόνο ότι αυτή λειτουργεί όπως η `smPrint`, με μία σημαντική διαφορά:

Αντί του % όταν πρέπει να παρουσιαστεί κάποια τιμή, θα πρέπει στα δεξιά του να προστεθεί κάποια επεξήγηση (δηλαδή ένας ή δύο χαρακτήρες) που καθορίζει τον τύπο δεδομένων που θα χρησιμοποιηθεί για να ερμηνευθεί η αντίστοιχη παράμετρος.

Εφαρμογή

```
function printFactors(n)
{
    for (
        let factor = 2;
        factor <= n/factor;
        factor++
    ) {
        while (n % factor == 0) {
            n /= factor;
            smPrint("% ", factor);
        }
    }

    if (n > 1) {
        smPrint("% ", n);
    }
    smPrint("\n");
    return;
}
```

```
#include <stdio.h>

void printFactors(int n)
{
    for (
        int factor = 2;
        factor <= n/(double)factor;
        factor++
    ) {
        while (n % factor == 0) {
            n /= factor;
            printf("%d ", factor);
        }
    }

    if (n > 1) {
        printf("%d ", n);
    }
    printf("\n");
    return;
}
```

Πίνακες

Στην **C**, οι πίνακες δηλώνονται όπως και στις απλές μεταβλητές, όμως επειδή ο τύπος δεδομένων είναι συγκεκριμένος, αυτό σημαίνει ότι είναι και κοινός για όλα τα στοιχεία του πίνακα. Ομοίως θα πρέπει να δίνονται και οι τιμές τις αρχικοποίησης. Π.χ.:

```
int x[] = { 12, 34, 56, 78 };
```

ή αντίστοιχα:

```
double x[] = { 12.34, 34.0, 56, 7.888 };
```

Το ίδιο ισχύει και για τους πίνακες δύο διαστάσεων:

```
int x[2][2] = { { 12, 34 }, { 56, 78 } };
```

Πίνακες ως παράμετροι

Μία ακόμη διαφορά για τις συναρτήσεις υπάρχει όταν κάποια ή κάποιες παράμετροι είναι πίνακες. Σε αυτή την περίπτωση θα πρέπει η δήλωση της παραμέτρου να συνοδεύεται και από τη διάσταση του πίνακα μέσα σε αγκύλες.

Επίσης όταν αυτή η διάσταση δεν είναι γνωστή εκ των προτέρων, τότε θα πρέπει η διάσταση αυτή να δοθεί ως παράμετρος και μάλιστα να προηγηθεί της παραμέτρου του πίνακα, ώστε να χρησιμοποιηθεί στη δήλωσή του. Δείτε και τον ακόλουθο κώδικα.

```
#include <stdio.h>

void printArray(int N, int a[N]) {
    for (int i=0; i<N; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}
```

Εφαρμογή

```
function findInArray(
            value,
            a,
            N
) {
    for (let i = 0; i < N; ++i) {
        if (a[i] == value) {
            return i;
        }
    }
    return -1;
}
```

```
int findInArray(
        double value,
        int N,
        double a[N]
) {
    for (int i = 0; i < N; ++i) {
        if (a[i] == value) {
            return i;
        }
    }
    return -1;
}
```

Τέλος Εργαστηρίου

Καλή συνέχεια!