Εισαγωγή στην Πληροφορική
&
στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Παναγιώτης Παύλου

intro-hy-24@allos.gr

CLion : Δημιουργία ενός Project

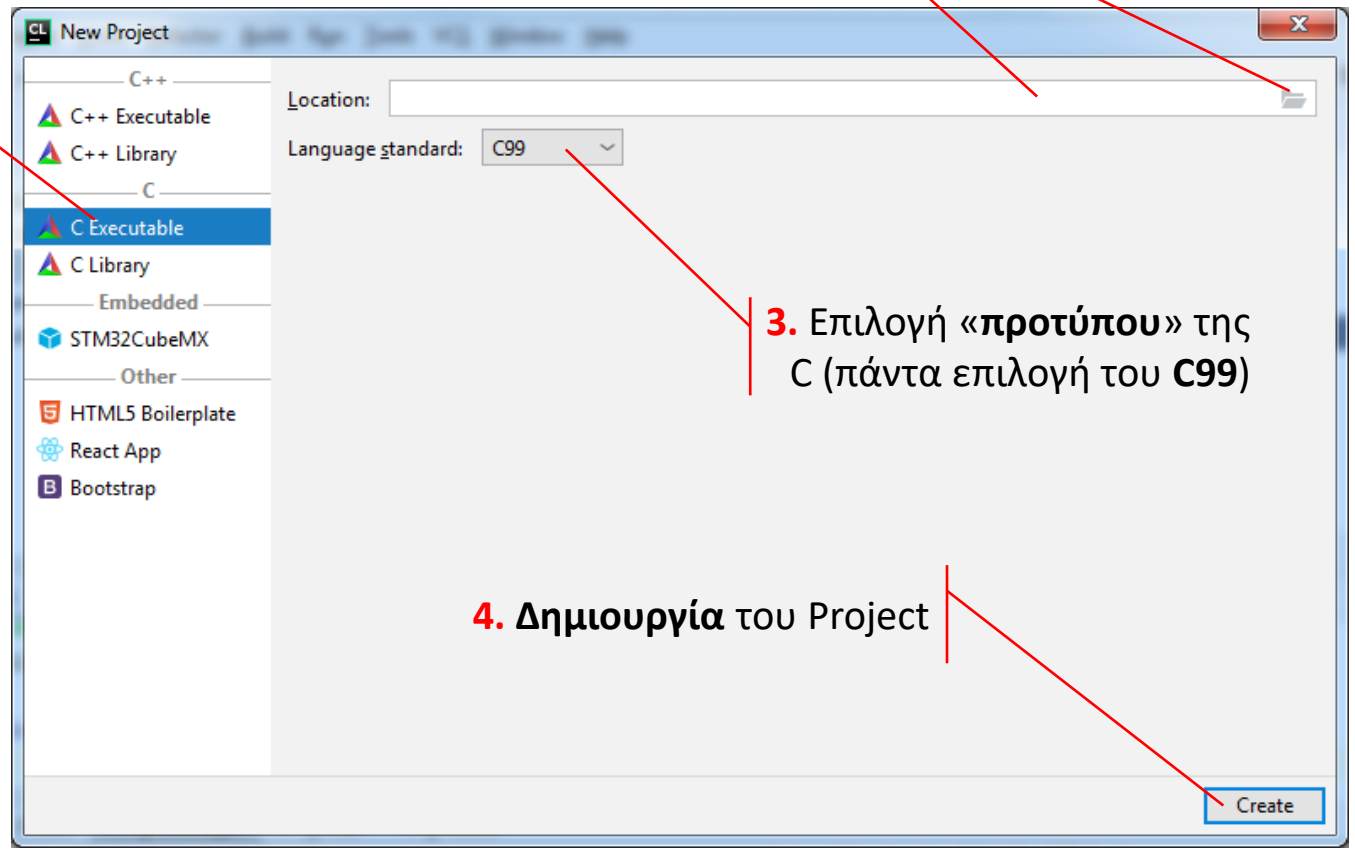
1. Επιλογή «**τι παράγει**» το project

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



3. Επιλογή «**προτύπου**» της C (πάντα επιλογή του **C99**)

4. Δημιουργία του Project

http:// 147.102.58.185 : 8888/

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

Αξιοποιώντας τη θεωρία…

Το 1ο πρόγραμμα σε C

```
function main() {
    smPrint("Hello world!\n");
    return 0;
}
```

```
#include <stdio.h>

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

Το 1ο πρόγραμμα σε C* μετατρέπεται όπως παραπάνω στο 1ο πρόγραμμα σε C.

Εδώ βλέπουμε όλες τις αλλαγές που χρειάζεται να γίνουν για να εκτελεστεί αυτό το πρόγραμμα. Θα δούμε και τη διαδικασία εκτέλεσής του.

Build
Δημιουργία εκτελέσιμων αρχείων

Πως παράγονται τα εκτελέσιμα αρχεία από τον κώδικα

Από τον κώδικα στο εκτελέσιμο αρχείο

Οι κώδικες των προγραμμάτων που γράφουμε σε διάφορες γλώσσες (από την assembly, τη C, την MATLAB, την Python, την Java, κοκ) είναι πάντα απλά αρχεία κειμένου χωρίς κάποια μορφοποίηση (άσχετα εάν το περιβάλλον στο οποίο γράφουμε τον κώδικα ενίοτε χρωματίζει κάποια κομμάτια του για καλύτερη κατανόησή).

Το κείμενο αυτό περνά μία διαδικασία – που ονομάζεται Building – μέχρι να μετατραπεί σε εκτελέσιμο αρχείο το οποίο μπορεί να σταθεί αυτόνομα στα πλαίσια ενός λειτουργικού συστήματος.

Τα βήματα αυτά συνοπτικά είναι:

- **Preprocessing**
  Σε αυτό το βήμα γίνεται μετασχηματισμός του αρχείου κειμένου και πάλι σε τροποποιημένο κείμενο
- **Compilation**
  Σε αυτό το βήμα το αρχείο κειμένου μετατρέπεται σε αρχείο object code, το οποίο – υπεραπλουστευτικά - είναι σαν μία επαυξημένη γλώσσα μηχανής
- **Linking**
  Σε αυτό το τελευταίο βήμα ολοκληρώνεται η μετατροπή του object code σε γλώσσα μηχανής και συνδέεται ο κώδικας αυτός με τον κώδικα των απαιτούμενων βιβλιοθηκών καθώς και με κάποιο κώδικα που αναλαμβάνει την εκκίνηση της εκτέλεσης σε συνεργασία με το λειτουργικό σύστημα (Loader)

Σημειώστε ότι στο τελευταίο στάδιο, τα αρχεία του object code μπορούν να προέρχονται ακόμα και από διαφορετικές compiled γλώσσες προγραμματισμού.

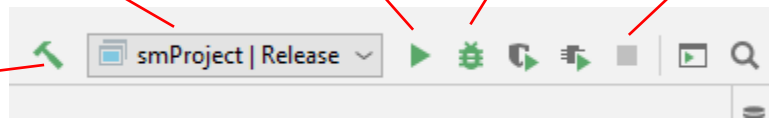
CLion : Χτίσιμο και Εκτέλεση

Για την **εκτέλεση** του προγράμματος πατώ εδώ

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

Όταν υπάρχουν περισσότεροι από έναν εκτελέσιμοι στόχοι επιλέγω από εδώ

Για τον **τερματισμό** ενός ήδη εκτελούμενου προγράμματος πατώ εδώ



Για το **χτίσιμο (build)** 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**.

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

  intro-hy-24@allos.gr

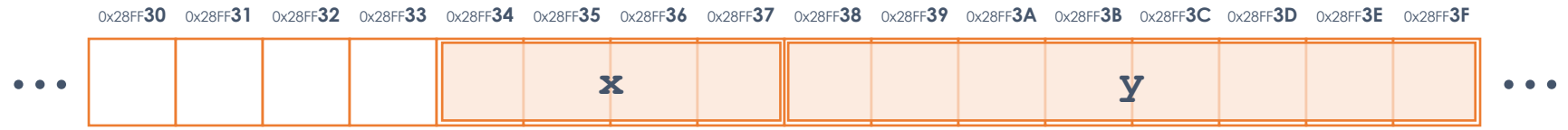
- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε μαζί τον κώδικα και τα μηνύματα λάθους από το CLion ως κείμενα με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Δεδομένα, Τιμές & Μεταβλητές

Πως παριστάνονται στον υπολογιστή και στην C

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**.

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 εξυπηρετούν τη 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 **Προσημασμένες** Τιμές			
`char`	8	-128	127
`short`	16	-32768	32767
`int`	32	-2147483648	2147483647
`long`	32	-2147483648	2147483647
`long long`	64	-9223372036854775808	9223372036854775807

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

Τύπος Δεδομένων	Bits	Ελάχιστη	Μέγιστη
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

- Κινητής υποδιαστολής
  (πάντα προσημασμένοι και στο δεκαδικό σύστημα αρίθμησης)
  - 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!

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.

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 ώστε να παραχθούν οι κατάλληλες εντολές. 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
```

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

  intro-hy-24@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.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Συναρτήσεις

Ορισμός και Δήλωση

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**.

Παραδείγματα και ένα πρόβλημα…

```c
#include <stdio.h>

void printOranges(int oranges) {
    printf("Exw %d portokalia\n",
           oranges);
    return;
}

int main() {
    printOranges(5);
    return 0;
}
```

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

double ypoteinousa(double a, double b) {
    return sqrt( a*a + b*b );
}

void proveUnitCycle() {
    double f = M_PI / 4;
    printf("%lf\n", ypoteinousa(
                        sin(f),
                        cos(f)
                    )
           );
}

int main() {
    proveUnitCycle();
    return 0;
}
```

Δηλώσεις συναρτήσεων

Όπως φαίνεται και στα προηγούμενα παραδείγματα είναι «παράλογη» η τοποθέτηση των συναρτήσεων αφού πρώτα εμφανίζονται οι πιο «εξειδικευμένες» συναρτήσεις και μετά οι γενικότερες και στο τέλος η… αρχική (η main)!

Επίσης σε περίπτωση όπου δύο συναρτήσεις χρειάζονται η μία να καλέσει την άλλη για να ολοκληρωθούν, τότε δεν γίνεται να γραφτούν και οι δύο… πρώτες!

Γι'αυτό τον λόγο υπάρχει και η δυνατότητα απλής **δήλωσης** (**declaration**) των συναρτήσεων, όπου γίνεται περιγραφή της συνάρτησης χωρίς να δίνεται ο κώδικάς της. Η δήλωση γίνεται όπως ο ορισμός, αλλά χωρίς το σώμα των εντολών ενώ αντί για άγκιστρα απλά μπαίνει ο τερματισμός εντολής `;` π.χ.

```
double inverse(double x);
```

Επίσης το όνομα των παραμέτρων μπορεί να παραληφθεί.

Μετά από τη δήλωση της, μια συνάρτηση μπορεί να κληθεί οπουδήποτε. Θα πρέπει όμως σε οποιοδήποτε σημείο του κώδικα από εκεί και κάτω να δοθεί και ο ορισμός.

Παράδειγμα

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

double ypoteinousa(
        double a, double b
) {
    return sqrt( a*a + b*b );
}

void proveUnitCycle() {
    double f = M_PI / 4;
    printf("%lf\n",
        ypoteinousa(
            sin(f),
            cos(f)
        )
    );
}

int main() {
    proveUnitCycle();
    return 0;
}
```

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

void proveUnitCycle();
double ypoteinousa(double a, double b);

int main() {
    proveUnitCycle();
    return 0;
}

void proveUnitCycle() {
    double f = M_PI / 4;
    printf("%lf\n", ypoteinousa( sin(f), cos(f) ));
}

double ypoteinousa(double a, double b) {
    return sqrt( a*a + b*b );
}
```

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

  intro-hy-24@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.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

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.

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

  intro-hy-24@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.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Διάλειμμα

μικρή ανάσα

Ανέκδοτο της ημέρας

-Τι κάνεις κύκλωπα;

-Καλά είμαι ·)

Σταθερές 1: Preprocessor (`#define`)

Η **`#define`** αντικαθιστά ένα λεκτικό με την παράσταση που το ακολουθεί σε όλη την έκταση του κώδικα από την ψευδοεντολή και κάτω, ακριβώς όπως ένας editor θα έκανε find/replace (εύρεση/αντικατάσταση). Π.χ.

```
#define THIS_YEAR 2021
```

Το όνομα (identifier) συνηθίζεται να γράφεται με κεφαλαίους χαρακτήρες.

Επίσης παρέχεται και μία άλλη παραλλαγή όπου το όνομα ακολουθείται από παρενθέσεις με παραμέτρους όπως οι συναρτήσεις. Αυτά τα «κατασκευάσματα» ονομάζονται macros ή μακροεντολές και δεν είναι συναρτήσεις. Π.χ.

```
#define PRODUCT(x,y) ((x)*(y))
```

Σε οποιοδήποτε σημείο του κώδικα πλέον γραφεί `PRODUCT(...,...)` τότε γίνεται η παραπάνω αντικατάσταση. Όμως με τα macros αυτά **δεν** θα ασχοληθούμε

Σταθερές 2: Λέξη κλειδί const

Η **const** , λειτουργεί όπως και στην C* με τη διαφορά ότι ανάμεσα σε αυτή και το όνομα της μεταβλητής γράφεται και ο τύπος δεδομένων. Π.χ.

```
const double MY_PI = 3.14;
```

Ουσιαστικά η **const** δεν παράγει σταθερές, αλλά μεταβλητές που δεν μπορεί να αλλαχθεί η τιμή τους (read only). Δηλαδή μία **const** αποθηκεύεται στη μνήμη του υπολογιστή, ενώ η προηγούμενη μέθοδος **(#define**) απλά αντικαθιστά τις τιμές μέσα στο κείμενο του κώδικα πριν από το build, οπότε είναι πιο αποδοτικός ο κώδικας.

Προσοχή! Η λέξη κλειδί **const** δεν είναι διαθέσιμη σε παλαιότερες εκδόσεις της C.

Και για τους δύο παραπάνω λόγους, **προτείνεται η χρήση της #define όπου είναι δυνατόν**. Σε πιο προχωρημένα θέματα, μπορεί να ταιριάζει η **const** καλύτερα από την **#define**.

Παράδειγμα

```
#include <stdio.h>

#define BEGIN {
#define END }

#define ONEpTHREE 1+3

#define PROD(x,y) ((x)*(y))

int main()
BEGIN
    printf("Result: %d\n", PROD(ONEpTHREE, 2+4));
    // Προσοχή! Η παραπάνω παράσταση γίνεται ((1+3)*(2+4)) πριν το build
    return 0;
END
```

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

  intro-hy-24@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.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Άλλες διαφορές

Σε όσα είδαμε μέχρι εδώ υπάρχουν και οι παρακάτω «κρυφές» διαφορές:

- Οι **μη αρχικοποιημένες μεταβλητές** δεν έχουν πλέον την ειδική τιμή undefined, αλλά κάποια «τυχαία» τιμή, οπότε και πάλι πρέπει να τις αρχικοποιούμε.
- Επειδή οι τύποι δεδομένων είναι συγκεκριμένοι, πλέον είναι πιο εύκολο να προκύψει **υπερχείλιση** και πλέον δεν υπάρχει κάποια ειδική τιμή (όπως η Infinity) που να την υποδεικνύει.
- Ο τελεστής % εφαρμόζεται μόνο σε ακέραιες μεταβλητές, ενώ για τις πραγματικές υπάρχει η συνάρτηση fmod.

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");
}
```

Λογικές ποσότητες

Οι επεξεργαστές επεξεργάζονται την πληροφορία πάντα σε ομάδες από bits.

Όμως μια boolean μεταβλητή ουσιαστικά χρειάζεται μόνο ένα bit για την αποθήκεσή της. Έτσι οι επεξεργαστές δεν έχουν εντολές (της γλώσσας μηχανής) που να καλύπτουν τέτοιες λειτουργίες σε όλο το απαιτούμενο εύρος δυνατοτήτων.

Γι'αυτό και η C δεν έχει εγγενή υποστήριξη τύπου δεδομένων bool.

Έτσι στην C 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.

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 **(προτείνεται)**
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

```
#include <stdbool.h>

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

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

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

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.

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

  intro-hy-24@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.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

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;
}
```

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;
}
```

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

  intro-hy-24@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.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Στατικές μεταβλητές

Μία νέα εμβέλεια που μας παρέχει η C είναι η δήλωση μεταβλητής ως στατική με τη λέξη κλειδί `static`. Η επίδραση της «διευκρίνισης» static εξαρτάται από το αν η μεταβλητή είναι global ή local.

Για τις τοπικές μεταβλητές η επίδραση είναι η εξής: Μετά το τέλος του block στο οποίο είναι δηλωμένη η μεταβλητή, αυτή δεν διαγράφεται από τη μνήμη, παρότι δεν είναι προσβάσιμη από τον κώδικα. Έτσι όταν η εκτέλεση του κώδικα επιστρέψει στο ίδιο block η μεταβλητή είναι και πάλι προσβάσιμη και περιέχει την προηγούμενη τιμή της. Φυσικά όταν δίνεται αρχική τιμή σε αυτή κατά τη δήλωσή της, η τιμή αποδίδεται μόνο στην πρώτη εκτέλεση της συνάρτησης (ή του block).

Για τις global μεταβλητές η δήλωσή της ως static, απλά περιορίζει την εμβέλειά τους στο αρχείο στο οποίο δηλώνονται, κάνοντάς τες ασφαλείς ως προς «συνωνυμίες» με global μεταβλητές άλλων αρχείων.

Πρακτικά ζητήματα

smProject – Γιατί σας βοηθά να παραδώσετε σωστότερο αποτέλεσμα

Χρήση smProject

Η κάθε εργασία, εκτός από την εκφώνησή της, θα συνοδεύεται και από ένα CLion project το οποίο έχει σαν σκοπό να σας βοηθά να παραδίδετε πιο σωστές εργασίες, επειδή:

1. Περιλαμβάνει κάποιες δοκιμές για κάποια συνηθισμένα σφάλματα πάνω στα ζητούμενα. Έτσι θα έχετε την ευκαιρία να δείτε μόνοι σας εάν ο κώδικάς σας έχει κάποιο τέτοιο σφάλμα.
   Η λίστα των ελέγχων είναι ενδεικτική και όχι εξαντλητική, έτσι εάν πετύχουν όλοι οι έλεγχοι, πιθανώς να εξακολουθούν να υπάρχουν αρκετά και σοβαρά λογικά σφάλματα στον κώδικά σας. Και τα σταματημένα ρολόγια 2 φορές τη μέρα δείχνουν τη σωστή ώρα.
2. Έχει έτοιμες τις δηλώσεις των συναρτήσεων που ζητούνται, οπότε έχετε μία ευκολία παραπάνω. Σωστό όνομα συνάρτησης, σωστές παραμέτρους και τύπο δεδομένων του αποτελέσματος.
3. Έχει οριοθετημένο τον κώδικα με ένα σχόλιο στην αρχή και ένα στο τέλος. Εφόσον εσείς γράψετε όλο τον κώδικά σας ανάμεσα σε αυτά τα σχόλια, τότε σε συνδυασμό με το αναβαθμισμένο σύστημα υποβολής, εάν δεν έχετε αντιγράψει ολόκληρο τον κώδικα θα εμφανιστεί μία ειδοποίηση γι'αυτό.

Από τα παραπάνω είναι προφανές ότι για κάθε άσκηση θα πρέπει να κατεβάζετε το αντίστοιχο smProject αυτής της άσκησης και σε αυτό το project να γράφετε τον κώδικα.

Για να είναι δυνατή η λειτουργία του smProject, χρειάζεται η παραδοχή ότι αντί της `main`, η κύρια συνάρτηση του προγράμματος θα είναι η `smMain` με την ίδια ακριβώς σύνταξη που έχει η `main`.

Στην επόμενη διαφάνεια θα δείτε πως, όταν ανοίξετε το smProject στο CLion, θα επιλέγετε την εκτέλεση της κανονικής λειτουργίας ή των δοκιμαστικών ελέγχων.

Κανονική εκτέλεση και δοκιμές

Στην γραμμή εκτέλεσης εμφανίζονται πλέον δύο **στόχοι**.

- **smProject** , που αφορά την κανονική εκτέλεση του κώδικα
- **RunTests** , που αφορά την εκτέλεση των δοκιμών

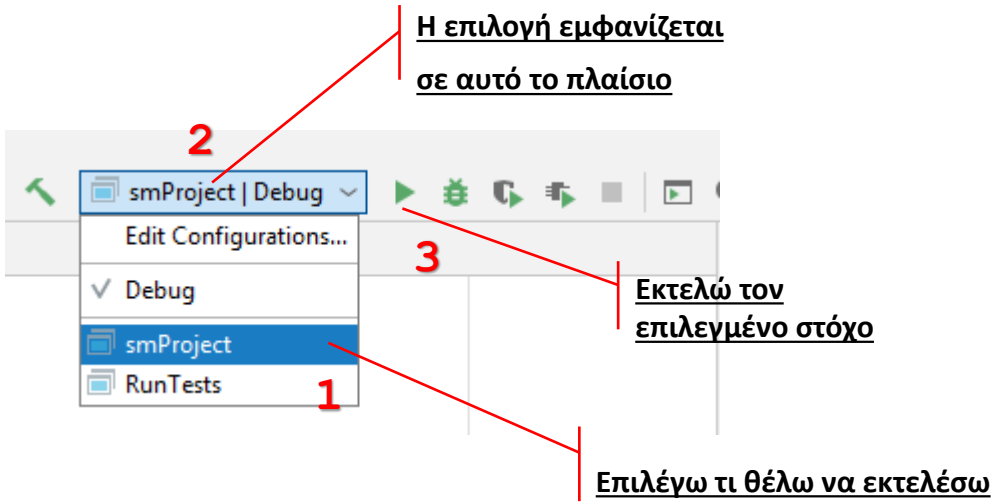
Θα πρέπει λοιπόν ο χρήστης, ανάλογα με την επιθυμία του, να επιλέξει τον αντίστοιχο στόχο.

Αφού έχει γίνει η επιλογή αυτή τα υπόλοιπα κουμπιά της γραμμής λειτουργούν κατά τα γνωστά.

Ως συνήθης πρακτική προτείνεται να γίνονται τα πάντα σε κανονική εκτέλεση μέχρι να θεωρήσει ο προγραμματιστής ότι ο κώδικάς είναι έτοιμος.

Κατόπιν να γίνεται αλλαγή του στόχου σε RunTests για να βεβαιωθεί ότι όλα εκτελούνται καλά.

Ενδεικτικό αποτέλεσμα ελέγχων φαίνεται δίπλα. *Το **ζητούμενο** είναι να μην υπάρχει η λέξη FAILED, να εμφανίζεται μόνο το Ok και η επιστρεφόμενη τιμή να είναι 0.*




```
TESTING MODE!


Test ARC_NORMAL_VALUE_TESTS :
  VALUE_1_2 FAILED : arc(1.2) returns unexpected result!
ARC_NORMAL_VALUE_TESTS FAILED!
Test aktina_0 : Ok
Test aktina_PI : Ok
Process finished with exit code 1
```

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-24@allos.gr

- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε τον κώδικα ως κείμενο με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Η γλώσσα C

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

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

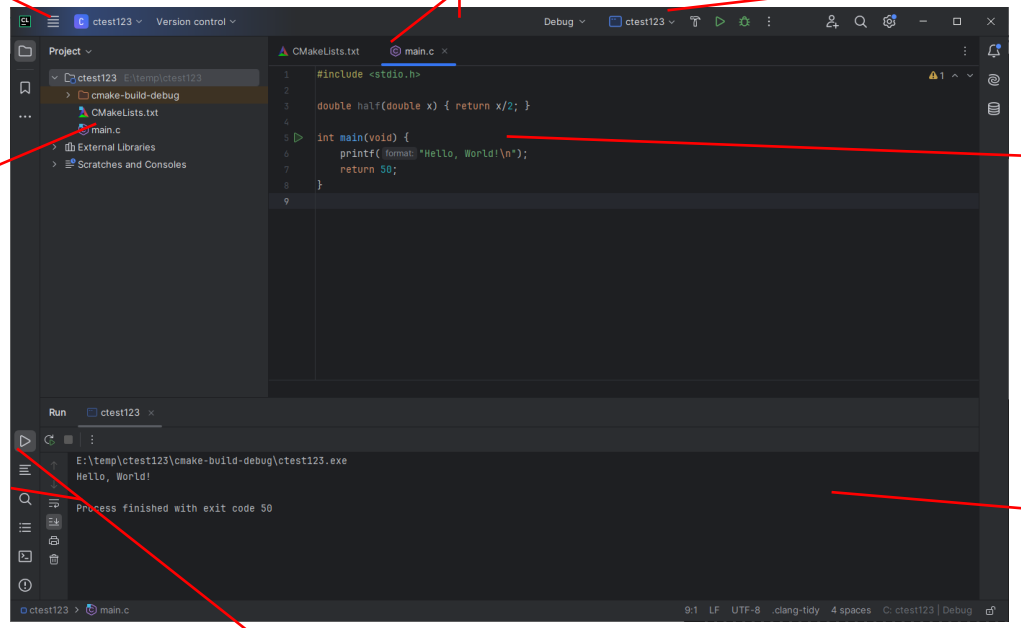
Όλα τα σχετικά και απαραίτητα αρχεία Περιλαμβάνουν και το αρχείο του κώδικα (εδώ main.c)

Καρτέλες Επεξεργαστή Κειμένου

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 Κειμένου

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 κειμένου) 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

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.