Η ΓΛΩΣΣΑ C

Μάθημα 14:

Εμβέλεια Μεταβλητών

Δημήτρης Ψούνης

Περιεχόμενα Μαθήματος

Α. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών
    1. Καθολικές Μεταβλητές
    2. Τοπικές Μεταβλητές
    3. Μεταβλητές-Ορίσματα
2. Εξειδικευμένα Είδη μεταβλητών
    1. Στατικές Τοπικές Μεταβλητές
    2. Μεταβλητές-Καταχωρητές

Β. Πρόγραμμα σε πολλαπλά αρχεία κώδικα.

1. Διάσπαση του προγράμματος σε αρχεία
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση
3. Εξωτερικές Μεταβλητές (extern)
4. Στατικές Καθολικές Μεταβλητές

Γ. Ασκήσεις

Α. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών

- Στα προγράμματα των προηγούμενων μαθημάτων είδαμε δηλώσεις μεταβλητών σε 3 σημεία σε ένα πρόγραμμα:
  - Τις καθολικές μεταβλητές που δηλώνονται πριν από την main
  - Τις τοπικές μεταβλητές που δηλώνονται στην αρχή μιας συνάρτησης.
  - Τις μεταβλητές-ορίσματα σε συναρτήσεις που γράφουμε εμείς.

- Στόχος του μαθήματος είναι:
  - Να συνοψίσουμε τις παρατηρήσεις που είπαμε στα προηγούμενα μαθήματα.
    - Ποιες μεταβλητές «βλέπει» μια συνάρτηση που ορίζουμε.
    - Πότε δεσμεύεται χώρος μνήμης και πότε καταστρέφεται ο χώρος μνήμης τους
    - Δηλαδή να μελετήσουμε την εμβέλεια των μεταβλητών μας.
  - Επίσης να δούμε κάποιες ειδικές προγραμματιστικές χρήσεις που μας δίνει η C με τις μεταβλητές.

- Σημείωση: Στο μάθημα αυτό, με τον όρο μεταβλητές εννοούμε και πίνακες, στιγμιότυπα δομών, δείκτες και οι παρατηρήσεις που θα αναφέρουμε αφορούν και τις σταθερές.

Α. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών

1. Καθολικές Μεταβλητές

- Μία καθολική μεταβλητή ορίζεται πριν από την main και είναι ορατή (προσβάσιμη) από κάθε συνάρτηση του προγράμματος.
- Ο χώρος της μνήμης μιας καθολικής μεταβλητής είναι αυτόνομος και σε αυτόν έχουν πρόσβαση όλες οι συναρτήσεις

- Ο χρόνος ζωής των καθολικών μεταβλητών είναι από την αρχή της εκτέλεσης του προγράμματος μέχρι και όταν εκτελεστεί η τελευταία εντολή της main.
  - Σημείωση με τον όρο χρόνος ζωής αναφερόμαστε στην διάρκεια εκτέλεσης του προγράμματος που ο χώρος της μεταβλητής στην μνήμη διατηρείται
  - Έτσι ο χώρος μνήμης που δεσμεύει μια καθολική μεταβλητή, αποδεσμεύεται μόλις ολοκληρωθεί η εκτέλεση του προγράμματος.

Παρατήρηση:
- Η χρήση καθολικών μεταβλητών είναι συχνό να οδηγεί σε λογικά λάθη και γι' αυτό είναι καλό να αποφεύγονται!

Α. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών

1. Καθολικές Μεταβλητές

- Με βάση τα παραπάνω η καθολική μεταβλητή x είναι προσβάσιμη από όλες τις συναρτήσεις του παρακάτω προγράμματος:

```
/* global.c: Deixnei tin xrisi twn katholikwn metavlitwn */
#include <stdio.h>

int x; /* Katholiki metavliti */
void func();

main()
{
      x=5;
      printf("\nMain: x=%d",x);
      func();
      printf("\nMain: x=%d",x);
}

void func()
{
     printf("\nFunc: x=%d",x);
     x=8;
     printf("\nFunc: x=%d",x);
}
```

A. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών

2. Τοπικές Μεταβλητές

- (Σημείωση για τα επόμενα: Ό,τι αναφέρουμε για μια συνάρτηση ισχύει και για την main, αφού και αυτή είναι μία συνάρτηση)

- Μία τοπική μεταβλητή ορίζεται στην αρχή μιας συνάρτησης
- Θυμίζουμε ότι ο χώρος μνήμης μιας τοπικής μεταβλητής είναι ορατός μόνο από την συνάρτηση που την ορίζει.

- Ο χρόνος ζωής των τοπικών μεταβλητών είναι από την αρχή της εκτέλεσης της κλήσης της συνάρτησης μέχρι και όταν εκτελεστεί η τελευταία εντολή της συνάρτησης.
  - Τονίζουμε ότι οι μεταβλητές που χρησιμοποιήθηκαν στη διάρκεια μιας κλήσης της συνάρτησης, καταστρέφονται και ξαναρχικοποιούνται σε επόμενη κλήση της συνάρτησης.

A. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών

2. Τοπικές Μεταβλητές

- Σημειώνουμε ότι σε περίπτωση σύγκρουσης ονομάτων μιας καθολικής με μία τοπική μεταβλητή (δηλαδή αν ορίσουμε καθολική και τοπική μεταβλητή με το ίδιο όνομα, επικρατεί η χρήση της τοπικής μεταβλητής.
- Έτσι το ακόλουθο πρόγραμμα θα τυπώσει την τιμή της τοπικής μεταβλητής (δηλ. 5)

```
/* local.c: Deixnei tin antimetwpisi sigkrousis onomatwn metavlitwn */
#include <stdio.h>

int x; /* Katholiki Metavliti */
void func();

main()
{
      x=1;
      func();

}

void func()
{
     int x=5; /* Topiki Metavliti */
     printf("\nLocal: x=%d",x);
}
```

A. Εμβέλεια Μεταβλητών

1. Είδη Μεταβλητών

3. Μεταβλητές – Ορίσματα

- Μία μεταβλητή που δηλώνεται ως όρισμα σε μια συνάρτηση έχει ακριβώς την ίδια συμπεριφορά με τις τοπικές μεταβλητές των συναρτήσεων.
  - Άρα είναι ορατές μόνο από την συνάρτηση που τις δέχεται ως όρισμα.
  - Ο χρόνος ζωής τους είναι όσος ο χρόνος εκτέλεσης της κλήσης της συνάρτησης.

- Σε περίπτωση που έχουμε σύγκρουση ονόματος μεταβλητής-ορίσματος με καθολικές μεταβλητές επικρατεί η μεταβλητή-όρισμα.

- Σε περίπτωση που έχουμε σύγκρουση ονόματος μεταβλητής-ορίσματος με τοπική μεταβλητή, θα διαμαρτυρηθεί ο μεταγλωττιστής.

Παρατήρηση:
- Γενικώς είναι δείγμα κακού προγραμματισμού να έχουμε συγκρούσεις ονομάτων. Ωστόσο θα πρέπει να γνωρίζουμε ποια μεταβλητή επικρατεί σε περίπτωση που έχουμε σύγκρουση.

A. Εμβέλεια Μεταβλητών

2. Εξειδικευμένοι Τύποι Μεταβλητών

1. Στατικές Τοπικές Μεταβλητές

- Σε κάποιες εξειδικευμένες εφαρμογές, δεν θέλουμε κάθε φορά που τελειώνει μια συνάρτηση να καταστρέφει μια συγκεκριμένη τοπική μεταβλητή της, αλλά να διατηρηθεί η τιμή της και την επόμενη φορά που θα κάνουμε κλήση της συνάρτησης.
- Η C μας δίνει αυτήν την ευκολία, δηλώνοντας την μεταβλητή με την λέξη κλειδί static πριν από την δήλωση της, π.χ.:

```
static int m;
```

- Συνεπώς μία στατική τοπική μεταβλητή είναι μία τοπική μεταβλητή που δεν καταστρέφεται όταν ολοκληρώνεται η κλήση της συνάρτησης, αλλά διατηρεί την τιμή της για την επόμενη φορά που θα γίνει κλήση της συνάρτησης.

  - Μια στατική τοπική μεταβλητή πρέπει να έχει πάντα και αρχικοποίηση κατά την δήλωση που θα εκτελεστεί μόνο την πρώτη φορά που θα εκτελεστεί η συνάρτηση.

A. Εμβέλεια Μεταβλητών

2. Εξειδικευμένοι Τύποι Μεταβλητών

1. Στατικές Τοπικές Μεταβλητές

- Εκτελέστε και μελετήστε το ακόλουθο πρόγραμμα που δείχνει την διαφορά μιας συνηθισμένης τοπικής μεταβλητής σε μια συνάρτηση σε σχέση με μία στατική τοπική μεταβλητή.

```
/* static.c: Deixnei tin xrisi twn statikwn metavlitwn */
#include <stdio.h>

void func();
main()
{
      int i;

      for (i=0; i<10; i++)
         func();
}
void func()
{
    int y=0;
    static int x=0;

    x=x+1;
    y=y+1;
    printf("\nx=%d, y=%d", x,y);
}
```

Α. Εμβέλεια Μεταβλητών

2. Εξειδικευμένοι Τύποι Μεταβλητών

2. Μεταβλητές – Καταχωρητές

- Οι καταχωρητές είναι χώροι αποθήκευσης της CPU
  - Είναι λίγοι στο πλήθος (εξαρτάται από τον επεξεργαστή μας πόσοι είναι)
  - Το κύριο χαρακτηριστικό τους είναι ότι είναι πολύ γρήγορα προσβάσιμοι σε σχέση με την μνήμη του υπολογιστή.
- Έτσι σε κάποιες εξειδικευμένες εφαρμογές, που σε κάποια μεταβλητή απαιτείται να έχουμε συνέχεια πρόσβαση, την δηλώνουμε ως μεταβλητή σε καταχωρητή με την προσθήκη της λέξης-κλειδί register στην δήλωσή της.
- Π.χ.: για μια ακέραια μεταβλητή:

```
register int m;
```

  - Και έπειτα την χρησιμοποιούμε κανονικά στο πρόγραμμά μας, όπως χρησιμοποιούμε οποιαδήποτε ακέραια μεταβλητή.
- Σημείωση: Εδώ ζητάμε να χρησιμοποιηθεί ένας καταχωρητής για την μεταβλητή m. Αν δεν υπάρχει διαθέσιμος καταχωρητής την στιγμή της εκτέλεσης, τότε η μεταβλητή θα είναι μια συνήθης ακέραια μεταβλητή.

Σημείωση:

- Εδώ ζητάμε να χρησιμοποιηθεί ένας καταχωρητής για την μεταβλητή m. Αν δεν υπάρχει διαθέσιμος καταχωρητής την στιγμή της εκτέλεσης, τότε η μεταβλητή θα είναι μια συνήθης ακέραια μεταβλητή.

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα

1. Διάσπαση του Προγράμματος σε Αρχεία

- Σε μεγάλα προγράμματα θεωρείται καλή προγραμματιστική πρακτική να γράφουμε τον κώδικα του προγράμματος σε διαφορετικά αρχεία.
  - Το συνηθισμένο είναι κάθε αρχείο που χρησιμοποιούμε να ορίζει συναρτήσεις που κάνουν μια συναφή δουλειά.
  - Π.χ. μπορούμε στο ένα αρχείο κώδικα να γράψουμε τις συναρτήσεις που θα διαχειρίζονται μια δομή δεδομένων που έχουμε ορίσει και στο δεύτερο αρχείο να έχουμε την συνάρτηση main μας.

- Ακόμα πιο προχωρημένα:
  - Μπορούμε να ορίσουμε την δική μας βιβλιοθήκη συναρτήσεων, τις οποίες θα μπορούμε να ενσωματώνουμε στα προγράμματα μας.

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα

1. Διάσπαση του Προγράμματος σε Αρχεία

- Ορίζουμε τα πρωτότυπα των συναρτήσεων στο αρχείο lib_array.h:

```
/* lib_array.h: Prwtotipa sinartisewn */

void init_array(int *pinakas, int n, int a, int b);
void print_array(int *pinakas, int n);
int sum_array(int *pinakas, int n);
```

- Σημείωση: Αν επιθυμούσαμε να ορίσουμε και δομές, τότε εδώ θα είχαμε τις δηλώσεις των δομών!

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα

1. Διάσπαση του Προγράμματος σε Αρχεία

- Ο κώδικας των συναρτήσεων γράφεται στο αρχείο «lib_array.c»
  - (συνηθίζεται να έχει το ίδιο όνομα με το αρχείο βιβλιοθήκης)

```
/* lib_array.c: Kwdikas twn sinartisewn */

#include <stdio.h>
#include <stdlib.h>
#include "lib_array.h"

void init_array(int *pinakas, int n, int a, int b)
{
        int i;

        srand(time(NULL));

        for (i=0; i<n; i++)
                pinakas[i]=a+rand()%(b-a+1);
}
void print_array(int *pinakas, int n)
{
```

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα

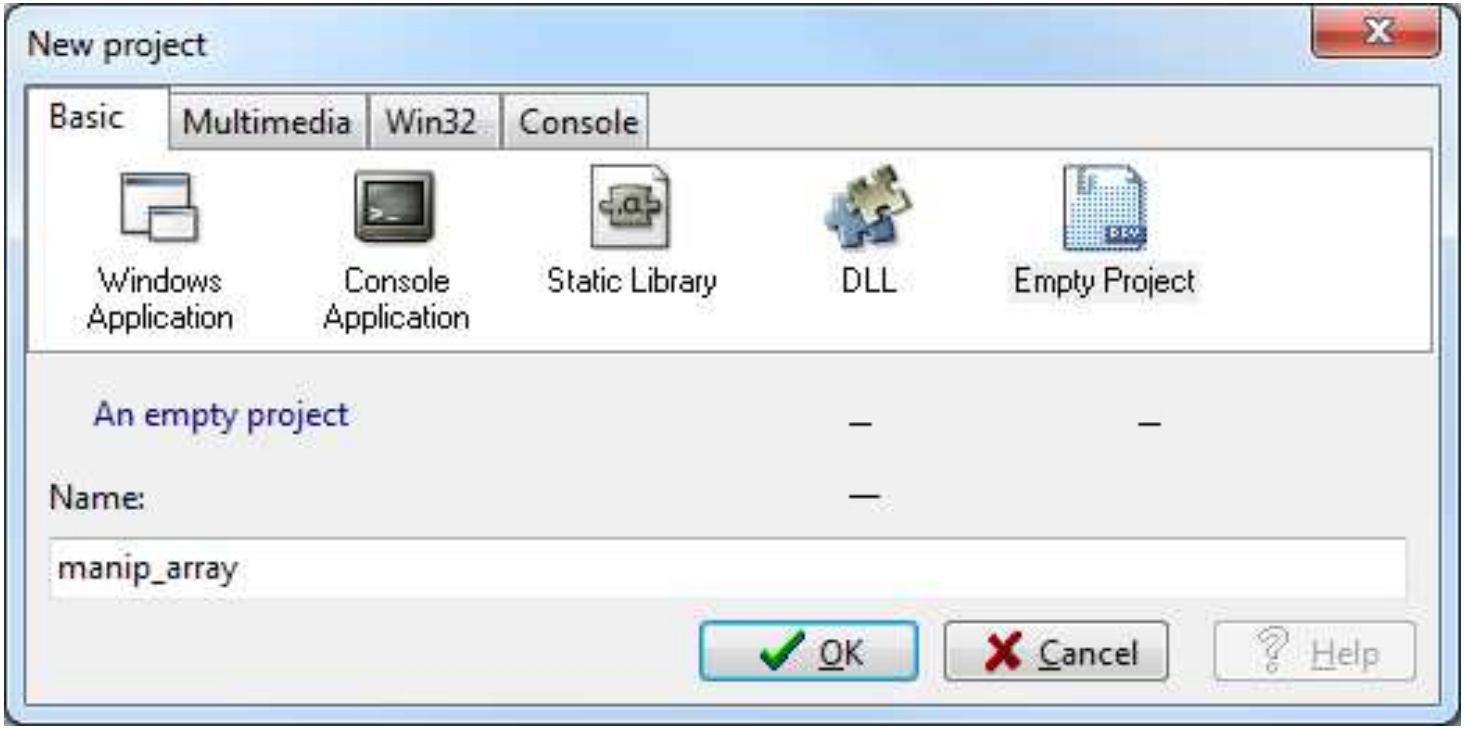
1. Διάσπαση του Προγράμματος σε Αρχεία

- Τέλος η main κάνει include μόνο το «.h» αρχείο και ενσωματώνει τις συναρτήσεις που περιέχονται σε αυτό.

```
#include <stdio.h>
#include <stdlib.h>
#include "lib_array.h"

#define N 50

main()
{
        int pin[N];
        init_array(pin,N,1,100);

        print_array(pin,N);

        printf("\n\nTo athroisma einai: %d\n\n", sum_array(pin,N));
}
```

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

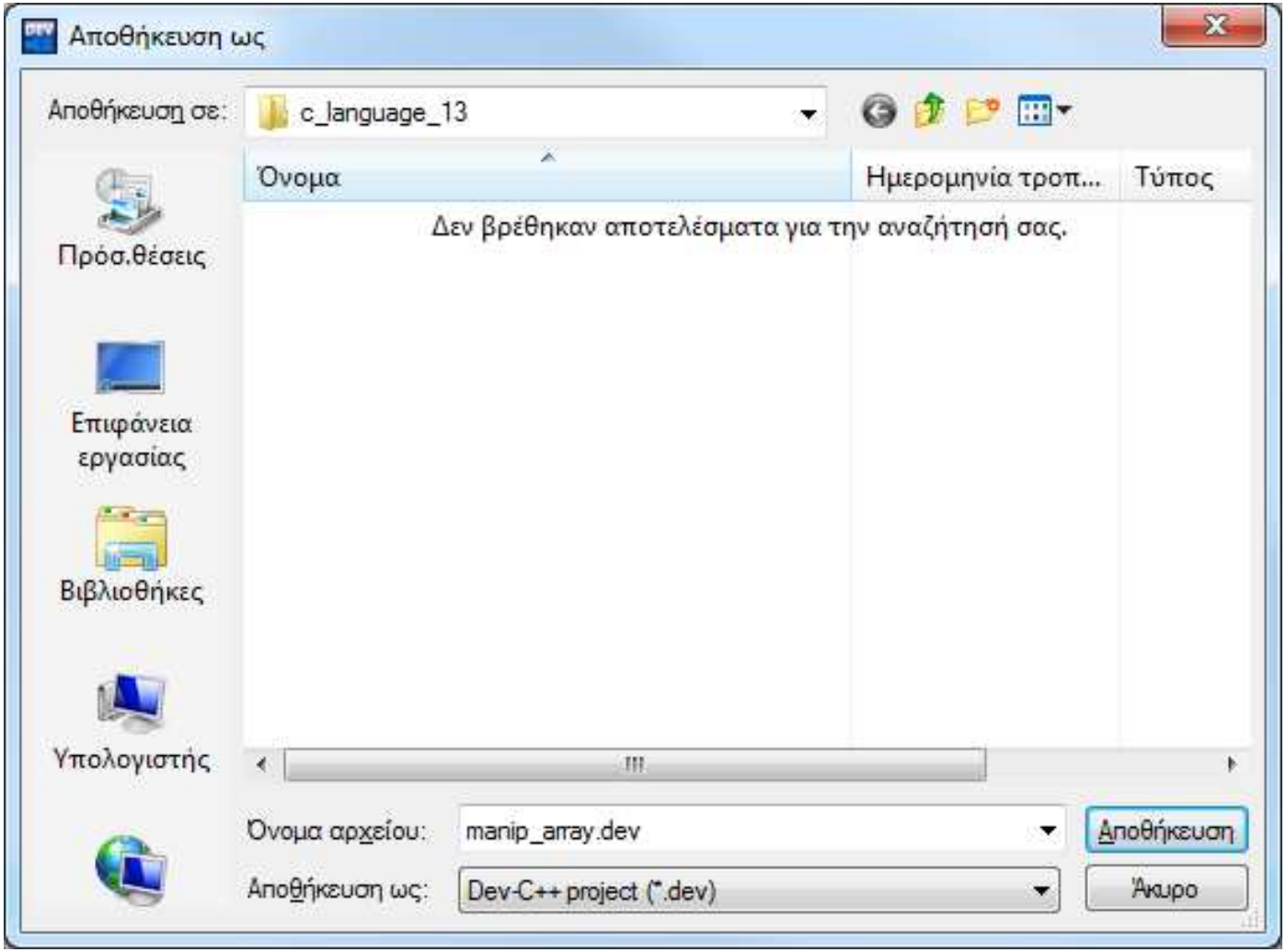
- Ας δούμε τις ενέργειες που πρέπει να κάνουμε στο DEV-C++ για την δημιουργία του τελικού εκτελέσιμου αρχείου:
  - Επιλέγουμε File->New Project
  - Στην οθόνη που εμφανίζεται:



  - Πληκτρολογούμε το όνομα του project, επιλέγουμε ότι είναι C Project και επιλέγουμε το εικονίδιο Empty Project. Πατάμε OK

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

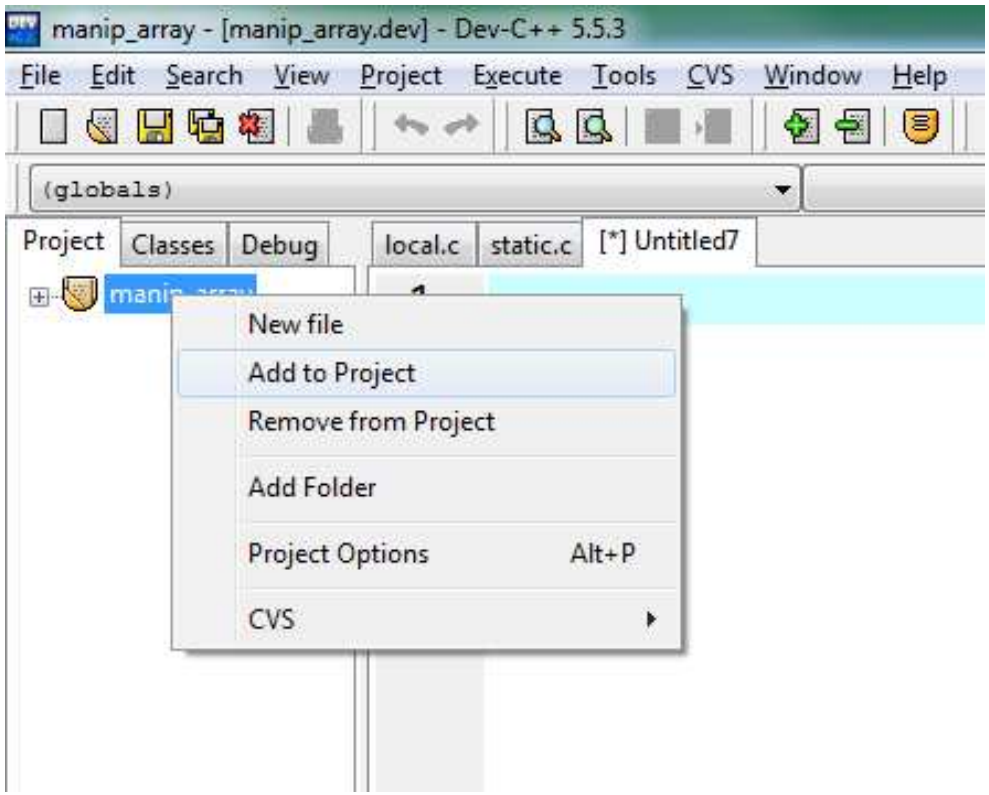
- Διαλέγουμε τον φάκελο που θα αποθηκεύσουμε το πρόγραμμα:



- Και πατάμε OK

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

- Κάνουμε δεξί κλικ στο όνομα του project:

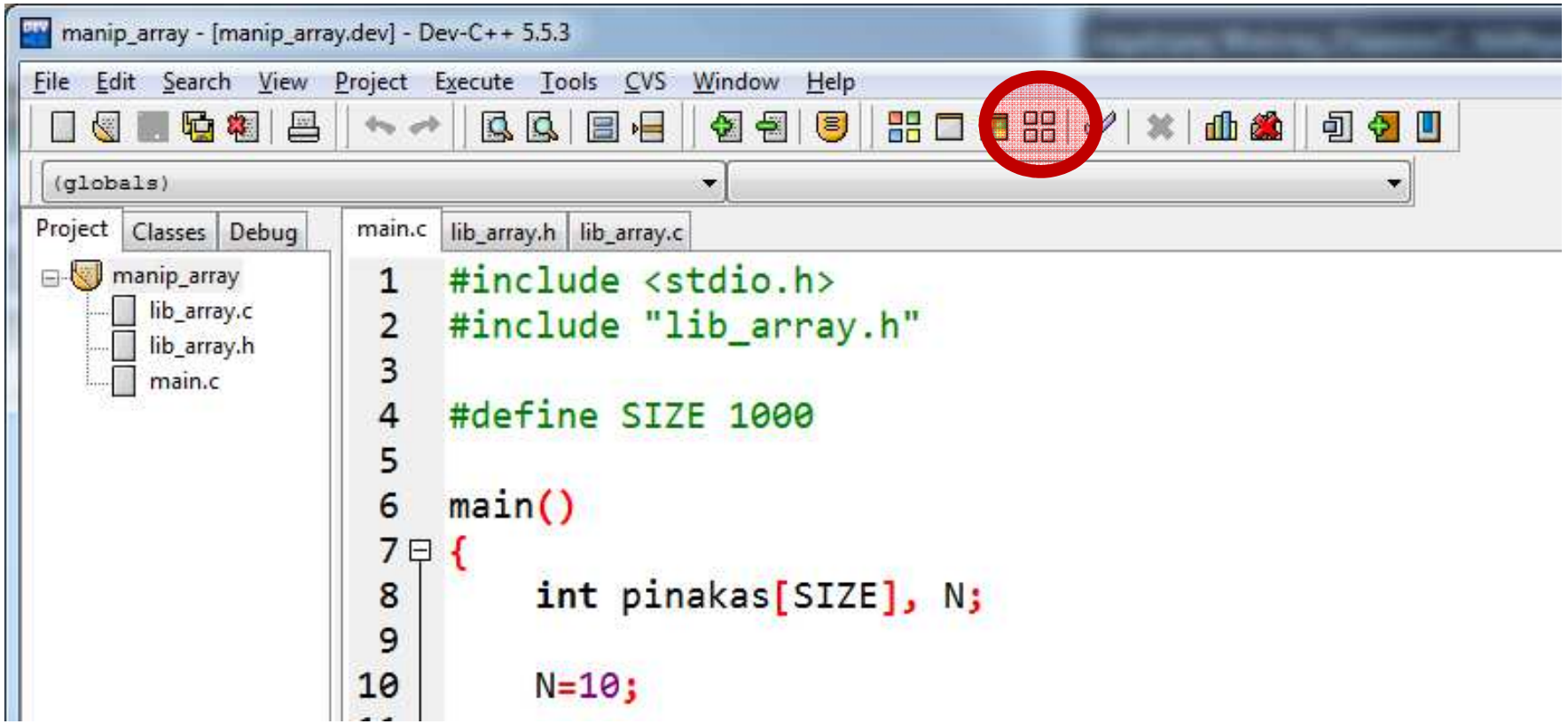


- Και επιλέγουμε add to project. Εκεί επιλέγουμε τα αρχεία που θέλουμε να ενσωματώσουμε στο project μας (Αυτά που έχουμε γράψει ήδη)

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα

2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

- Η εικόνα του DEV-C++ πρέπει να είναι η εξής.



- Όπου μπορούμε να επιλέξουμε και να επεξεργαστούμε το αντίστοιχο αρχείο κώδικα. Πιέζουμε το κουμπί Rebuild All

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

- Με το οποίο μπορούμε να κάνουμε ταυτόχρονη μεταγλώττιση και σύνδεση των αρχείων κώδικα και θα παραχθεί το εκτελέσιμο αρχείο που μπορούμε να τρέξουμε με το κουμπι Run.

- Σημαντικό! Δείτε στο φάκελο που έχετε τα αρχεία, ότι έχει παραχθει και το αντικειμενικό αρχείο που αντιστοιχεί σε κάθε αρχείο κώδικα (προέκταση αρχείου .o)

- Αν επιλέξετε μεταγλώττιση ενός αρχείου κώδικα θα παραχθεί μόνο το αντίστοιχο αντικειμενικό αρχείο και όχι το τελικό εκτελέσιμο αρχείου.

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

- Ας δηλώσουμε μία καθολική μεταβλητή (π.χ στο αρχείο που περιέχει την main). Τότε η μεταβλητή αυτή ΔΕΝ θα είναι προσβάσιμη από άλλο αρχείο κώδικα.

```
int x;
```

- Αν θέλουμε να είναι προσβάσιμη και στα άλλα αρχεία, θα πρέπει στα αρχεία αυτή να δηλώσουμε την μεταβλητή με το ίδιο όνομα και με την λέξη κλειδί extern μπροστά από την δήλωσή της.

```
extern int x;
```

  - Με τον τρόπο αυτό η καθολική μεταβλητή, δηλώνεται ως εξωτερική καθολική μεταβλητή στα υπόλοιπα αρχεία.
- Ας δούμε ένα παράδειγμα για την χρήση των εξωτερικών μεταβλητών.

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

- Όπως φαίνεται στα αρχεία κώδικα η καθολική μεταβλητή δηλώνεται τοπικά στην extern_main.c και έπειτα ως εξωτερική καθολική μεταβλητή στην extern_lib.c

```
/* extern_main.c */
#include <stdio.h>
#include "extern_lib.h"

int x=4;

main()
{
   printf("\nMAIN: %d",x);

   func();

}
```

```
/* extern_lib.h */
void func();
```

```
/* extern_lib.c */
#include "extern_lib.h"

extern int x;

void func()
{
     printf("\nFUNC: %d",x);
}
```

Β. Πρόγραμμα σε Πολλαπλά Αρχεία Κώδικα
2. Μεταγλώττιση, Σύνδεση και Εκτέλεση

- Έστω μία καθολική μεταβλητή ορισμένη σε ένα αρχείο κώδικα.
- Αν θέλουμε την μεταβλητή αυτή να την βλέπει μόνο το συγκεκριμένο αρχείο και να απαγορεύσουμε την χρήση της από άλλα αρχεία, γράφουμε μπροστά από την δήλωσή της, την λέξη-κλειδί static:

```
static int x;
```

- Με τον τρόπο αυτό η συγκεκριμένη καθολική μεταβλητή, δεν μπορεί να χρησιμοποιηθεί από άλλα αρχεία, ακόμη κι αν την δηλώσει άλλο αρχείο σαν εξωτερική μεταβλητή.

Γ. Ασκήσεις

1. Μέτρηση Αναδρομικών Κλήσεων

- Χρησιμοποιήστε στατική τοπική μεταβλητή για να μετρήσετε το πλήθος των αναδρομικών κλήσεων που γίνονται στο πρόγραμμα υπολογισμού των αριθμών Fibonacci που δημιουργήσαμε στο «Μάθημα 6 – Συναρτήσεις και Αναδρομή.

Γ. Ασκήσεις

2. Σπάσιμο σε Αρχεία με Δομές Δεδομένων

- Με τις γνώσεις που αποκτήσατε από το μάθημα, διασπάστε το πρόγραμμα «Μάθημα 13: Δομές – Εφαρμογή 6», ώστε οι συναρτήσεις και η δήλωση της δομής να πραγματοποιείται σε ξεχωριστό αρχείο βιβλιοθήκης.