Τυχαίοι αριθμοί

Διεπαφές

Τυχαίοι αριθμοί (random numbers)

- **Τυχαίος αριθμός → δεν καθορίζεται εκ των προτέρων η τιμή που θα λάβει. Δίνεται από ένα σύνολο ισοπίθανων τιμών.**

- **Οι τυχαίοι αριθμοί που παράγονται στους υπολογιστές δεν είναι μαθηματικά τυχαίοι. Παράγονται ψευδοτυχαίοι αριθμοί (pseudo-random numbers).**

- **int *rand*(void);** **επιστρέφει ένα φυσικό αριθμό (ακέραιο, μη αρνητικό) ανάμεσα στο *0* και τον *RAND_MAX*.**
- **Ορίζεται στη βιβλιοθήκη stdlib.h**

Τυχαίοι αριθμοί

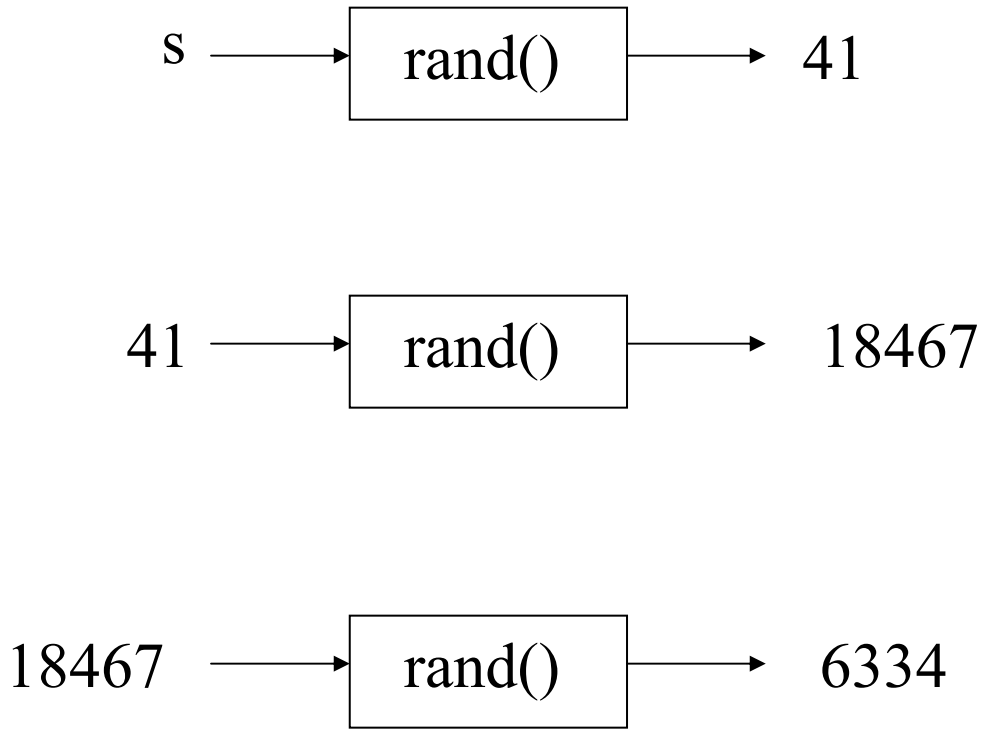
```
#include<stdio.h>
#include<stdlib.h>

void main()
{
  int i;
  for (i=0; i<10; i++) printf("%d\n", rand());
}
```

Το παραπάνω πρόγραμμα τυπώνει μία λίστα 10 τυχαίων ακεραίων ανάμεσα στο *0* και το *RAND_MAX*.

Ακολουθία ψευδοτυχαίων

Το αποτέλεσμα της προηγούμενης *rand()* είναι η είσοδος της διαδικασίας που παράγει το επόμενο αποτέλεσμα της rand().



Τυχαίοι αριθμοί

- **Η αρχική τιμή (*s*) ονομάζεται σπόρος (*seed*). Εάν ξεκινήσουμε από τον ίδιο σπόρο θα καταλήξουμε στην ίδια ακολουθία τυχαίων αριθμών.**

- **Για να έχουμε διαφορετική ακολουθία τυχαίων αριθμών σε κάθε τρέξιμο του προγράμματος θα πρέπει να αλλάζουμε το seed.**

- **Αυτό μπορεί να επιτευχθεί με χρήση της συνάρτησης:**

void srand(int)

Τυχαίοι αριθμοί

Για να διασφαλισθεί ότι κάθε φορά θα χρησιμοποιείται διαφορετικός σπόρος χρησιμοποιούμε το εσωτερικό ρολόι: Η συνάρτηση *time(NULL)* επιστρέφει την τρέχουσα ώρα, και μπορούμε να τη μετατρέψουμε σε ακέραιο για να τη χρησιμοποιήσουμε στη *srand()*:

```
srand( (int) time(NULL));
```

Τυχαίοι αριθμοί

Εναλλακτικά μπορούμε να χρησιμοποιήσουμε τη δομή

```
struct time_t    {
 unsigned char ti_min;      /* λεπτά (int) */
 unsigned char ti_hour;     /* ώρες (int) */
 unsigned char ti_hund;     /* εκατοστά δευτερολέπτου (int) */
 unsigned char ti_sec;      /* δευτερόλεπτα (int) */
};
```

η οποία ορίζεται στο *time.h*, και τη συνάρτηση *gettime(&t);*, η οποία αποδίδει το χρόνο του συστήματος στη μεταβλητή *t* τύπου *time*. Έτσι εάν χρησιμοποιήσουμε το μέλος *ti_hund*, έχουμε:

```
srand(t.ti_hund);
```

Τυχαίοι αριθμοί

- **Πώς θα καθορισθεί ένα συγκεκριμένο πεδίο τιμών;**
- ***Παράδειγμα:*** **Προσομοίωση του ριξίματος των ζαριών**

  Εφόσον χρειαζόμαστε ένα τυχαίο αριθμό ανάμεσα στο 1 και το 6, θα πρέπει να χρησιμοποιηθεί ο τελεστής υπολοίπου (modulus)

- **x = rand()%6; Θέτει στο *x* έναν τυχαίο ακέραιο ανάμεσα στο *0* και το *5* (0, 1, 2, 3, 4, 5)**
- **x = rand()%6 +1; παράγει έναν τυχαίο ακέραιο ανάμεσα στο *1* και το *6*.**

Τυχαίοι αριθμοί

Πώς μπορούμε να παράξουμε τυχαίους πραγματικούς αριθμούς με πεδίο τιμών ένα συγκεκριμένο διάστημα [low, high];

- **Αρχικά απεικονίζουμε τη *rand()* στο διάστημα [0 1]:**

```
d = (double) rand() / RAND_MAX;
```

ο d διατηρείται στο [0 1]: (0.0 <= d <= 1.0)

Ή, για να εξαιρέσουμε το 1.0, (0.0 <= d < 1.0):

```
d = (double) rand() / (RAND_MAX+1);
```

- **Στη συνέχεια κάνουμε την αλλαγή κλίμακας του d ως εξής:**

```
lo_hi = low + d * (high - low);
```

(υπενθύμιση) Πρωτότυπα συναρτήσεων

- Το πρωτότυπο μίας συνάρτησης ορίζει το **όνομα**, τους τύπους **εισόδου** και **εξόδου**.
- Αυτά είναι όλα όσα απαιτούνται για να χρησιμοποιηθεί μία συνάρτηση.
- Μερικές συναρτήσεις μοιάζουν να είναι κρυμμένες

- **Το αρχείο #include<stdio.h> και τα υπόλοιπα αρχεία '.h'**
  - Αυτά τα αρχεία κεφαλίδας είναι γεμάτα από πρωτότυπα συναρτήσεων
  - Το <stdio.h> έχει τα πρωτότυπα (όχι όμως και τα σώματα) των συναρτήσεων **printf()**, **scanf()** και πολλών άλλων.

- **Πού βρίσκονται τα σώματα των συναρτήσεων;**

  Σε αρχεία βιβλιοθήκης, κρυμμένα κάπου αλλού …

- **Γιατί;**

Ποιο είναι το κίνητρο;

- Μετά από ένα διάστημα τα αρχεία .c γεμίζουν με μεγάλη ποσότητα υλικού, γεγονός που έχει ως αποτέλεσμα:
  - Να υπάρχουν πάρα πολλά σώματα και δηλώσεις συναρτήσεων
  - Να μην υπάρχει οργάνωση και ομαδοποίηση των συναρτήσεων
- Μπορούμε να τοποθετήσουμε τα πρωτότυπα των συναρτήσεων στο αρχείο myfunc.h και να τα αντικαταστήσουμε με το #include "myfunc.h"
- Μπορούμε να τοποθετήσουμε τα σώματα στο αρχείο myfunc.c και να κάνουμε αυτό το αρχείο τμήμα του project.

Η ιδέα; Διεπαφές (Interfaces)

- **Interface**: ένα κοινό σύνορο ανάμεσα σε δύο ξεχωριστές οντότητες:
- Η χρήση μίας συνάρτησης (π.χ. **printf(), scanf()**) είναι **ασυσχέτιστη** με τη συγγραφή της, γι' αυτό μπορούμε να συλλέγουμε σύνολα παρόμοιων συναρτήσεων μέσα σε «**βιβλιοθήκες**» και να τα χρησιμοποιούμε.
- **«Η λογική του μαύρου κουτιού»:** να αποκρυφθούν οι εσωτερικές διεργασίες μίας βιβλιοθήκης, όπως ακριβώς συμβαίνει στις συναρτήσεις.

Διεπαφές (Interfaces)

Στη C:

- **Αρχείο κεφαλίδας** == **διεπαφή** == **`αρχείο .h`**

Ένα σταθερό, αξιόπιστο σύνορο ανάμεσα στον κώδικα και τη βιβλιοθήκη των συναρτήσεων. Το μόνο που απαιτείται να γνωρίζουμε.

- Κρατά λίστα με τα πρωτότυπα των συναρτήσεων
- Ορίζει για όλες τις συναρτήσεις το **όνομα,** τις **εισόδους,** τις **εξόδους**
- Περιέχει πολλά σχόλια με τις «οδηγίες χρήσης»).

- **Βιβλιοθήκες** ==εύχρηστες συλλογές συναρτήσεων, που έχουν ήδη γραφεί και αποσφαλματωθεί.

Συγγραφή μίας διεπαφής

Μία καλοσχεδιασμένη διεπαφή:

- **Έχει ένα συγκεκριμένο θέμα, που αντανακλά στο όνομά της** (Παράδειγμα: graphics.h, math.h, windows.h, …)
- **Είναι απλή** (αποκρύπτει τις σύνθετες λεπτομέρειες από το χρήστη)
- **Είναι πλήρης και επαρκής** (επιτελεί όλες τις εργασίες που την αφορούν)
- **Είναι γενική**
- **Είναι σταθερή** (η διεπαφή παραμένει η ίδια ακόμη κι αν βελτιωθεί/ αλλαχθεί η υλοποίηση)

Συγγραφή μίας διεπαφής

- Βήμα 1: Δημιούργησε ένα αρχείο διεπαφής ***myfuncs.h***, αποτελούμενο από πρωτότυπα συναρτήσεων και σχόλια.

- Βήμα 2: Γράψε όλες τις υλοποιήσεις των συναρτήσεων σε χωριστό αρχείο ***myfuncs.c*** .

- Βήμα 3: Για να χρησιμοποιήσεις οποιαδήποτε από αυτές τις συναρτήσεις στη ***main()***, το μόνο που απαιτείται είναι η εντολή:

#include "myfuncs.h"

Σύνταξη του αρχείου διεπαφής

Περιεχόμενα του αρχείου κεφαλίδας:

```
#ifndef _funcs_h
#define _funcs_h
```

γραμμές #include

ορισμοί σταθερών

ορισμοί τύπων

πρωτότυπα συναρτήσεων

```
#endif
```

'Wrapper': περισσότερες εντολές του compiler, "περιέλαβε μόνο μία φορά"

Όσα περιλαμβάνονται στο .h δεν περιλαμβάνονται εκ νέου στο κύριο πρόγραμμα γιατί ενσωμάτώνονται μέσω του .h.

Τα καλά σχόλια είναι ΑΚΡΩΣ ΣΗΜΑΝΤΙΚΑ

ΑΝΟΙΞΤΕ ΕΝΑ ΑΡΧΕΙΟ .h ΚΑΙ ΜΕΛΕΤΗΣΤΕ ΤΟ

Επιτρέπονται οι ΕΝΘΕΤΕΣ ΒΙΒΛΙΟΘΗΚΕΣ

- Το αρχείο βιβλιοθήκης (***myfuncs.c***) μπορεί να απαιτεί άλλες συναρτήσεις όπως η printf().
- Θέσε τις προτάσεις include του `#include <stdio.h>` μέσα στη **διεπαφή** *(myfuncs.h)*, (όχι στο *myfuncs.c*).
- **ΠΡΟΣΟΧΗ! Τα 'κυκλικά' include δε λειτουργούν!**

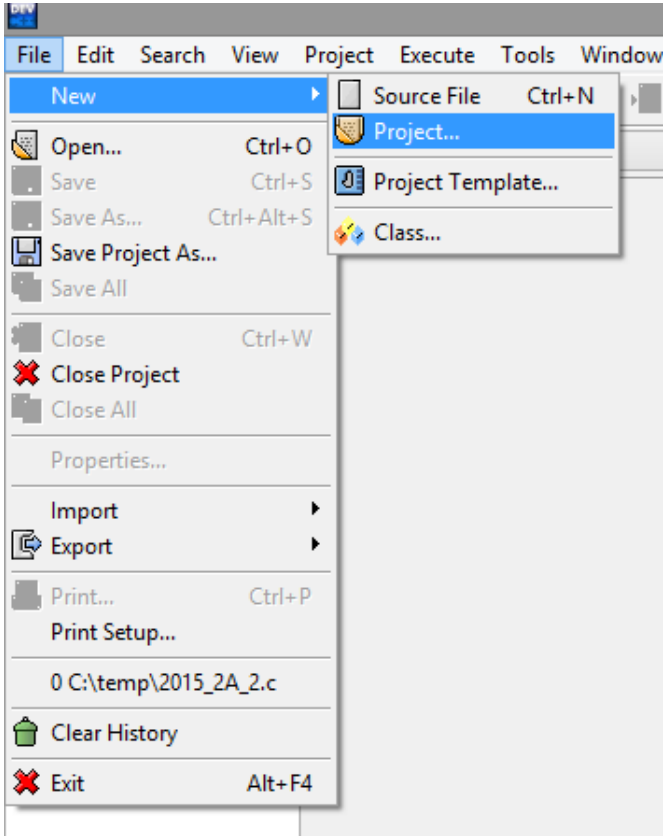
```
/*
 *  my1.h file
 */

#include "my2.h"
…
```

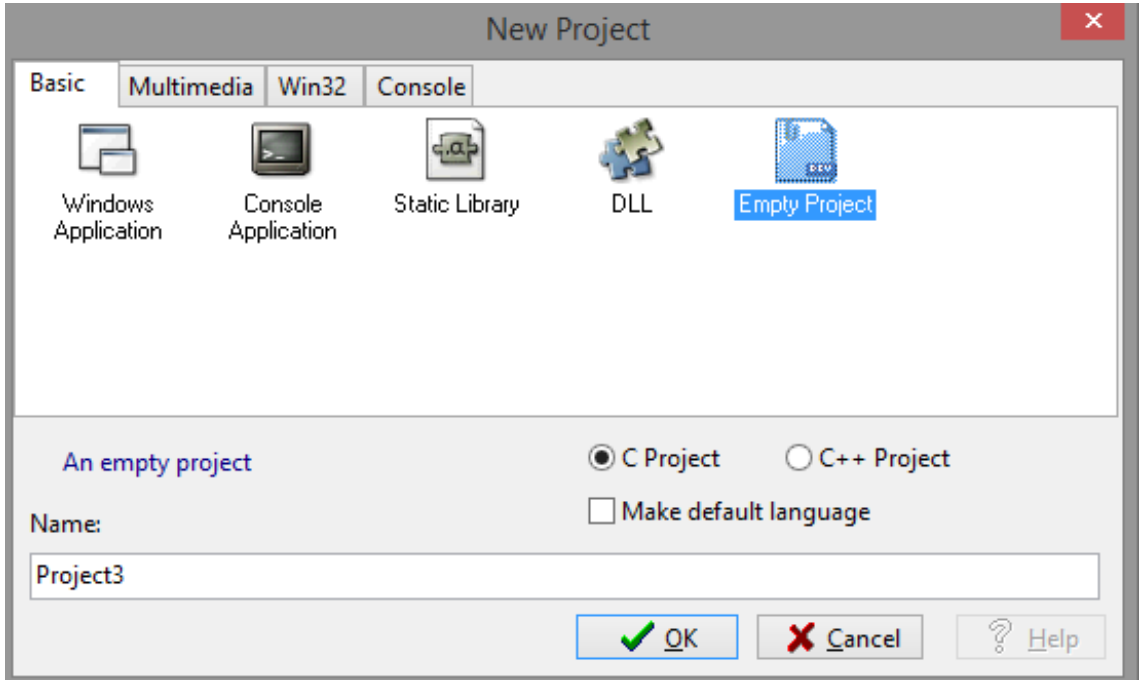
```
/*
 * my2.h  file
 */

#include "my1.h"
…
```

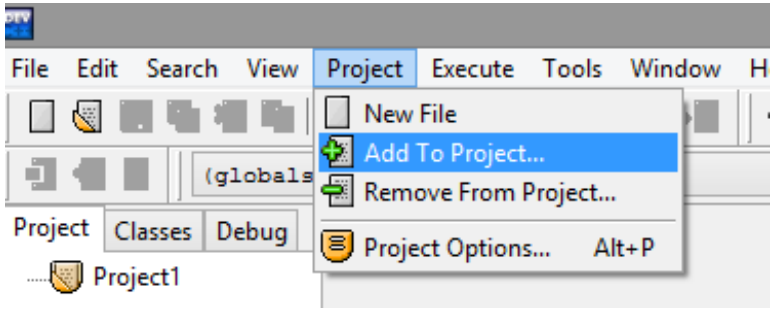
(α) Δημιουργείται νέο Project:



(β) Επιλέγεται Empty Project και ακολούθως C Project:



(γ) Τα αρχεία με κατάληξη .c προστίθενται στο Project. Δεν προστίθενται τα αρχεία με κατάληξη .h. Επίσης δεν απαιτούνται αρχεία κεφαλίδας ή μακροεντολές που ενσωματώθηκαν ήδη σε "*.h".



Παράδειγμα:

- *Στο κεφάλαιο της δυναμικής δέσμευσης μνήμης ορίσθηκαν συναρτήσεις που δεσμεύουν και θα αποδεσμεύουν μνήμη για πίνακες float, int κ.λ.π. Οι συναρτήσεις δέσμευσης μνήμης έχουν παραμέτρους τα μεγέθη της μνήμης που ζητείται να δεσμευθεί και επιστρέφουν το δείκτη που θα διαχειρίζεται τη μνήμη. Αντίστοιχα, οι συναρτήσεις αποδέσμευσης μνήμης έχουν παραμέτρους το δείκτη που διαχειρίσθηκε τη μνήμη και το μέγεθος αυτής και δεν έχουν επιστρεφόμενη τιμή.*

- *Εφόσον οι λειτουργίες αυτές απαντώνται στην πλειοψηφία των προγραμμάτων, μπορεί να δημιουργηθεί μία διεπαφή που θα τις περιλαμβάνει.*

//interfaces1//…

Αρχείο κεφαλίδας mymemory.h

```
#ifndef _mymemory_h
#define _mymemory_h

#include <malloc.h>                    /* απαραίτητα αρχεία κεφαλίδας */
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

float *allocate_1(int size);           /* δέσμευση size θέσεων float */
float **allocate_2(int size1, int size2);  /* δέσμευση size1xsize2 θέσεων */
void free_2(float **deikt, int size1);  /* αποδέσμευση της μνήμης που
                                           δεσμεύθηκε με την allocate_2 */

#endif
```

Αρχείο mymemory.cpp

```
#include "mymemory.h"

float **allocate_2(int size1, int size2)
{
  int i;     float **deikt;
  deikt=(float **)malloc(size1*sizeof(float *)); assert(deikt!=NULL);
  for (i=0;i<size1;i++) {
    deikt[i]=(float *)malloc(size2*sizeof(float));   assert(deikt[i]!=NULL);
  }
  return(deikt);
}
```

```
void free_2(float **deikt, int size1)
{
  int i;
  for (i=(size1-1);i>=0;i--) free(deikt[i]);
  free(deikt);
}

float *allocate_1(int size)
{
  float *deikt;
  deikt=(float *)malloc(size*sizeof(float));  assert(deikt!=NULL);
  return(deikt);
}
```

Κύριο πρόγραμμα prog1.cpp

```
#include "mymemory.h"
#include <time.h>

main()  {
  float **s;
  int i,j;
  s=allocate_2(3,250);
  printf("\n\ns=%d\t\taddr(s)=%d\n",s,&s);
  printf("s[0]=%d\t\taddr(s[0])=%d\n",s[0],&s[0]);
  printf("s[1]=%d\t\taddr(s[1])=%d\n",s[1],&s[1]);
  printf("s[2]=%d\t\taddr(s[2])=%d\n",s[2],&s[2]);
```

```
for (i=0;i<2;i++)
 {
  srand((int) time(NULL)+i);
  for (j=0;j<2;j++)  {
   s[i][j]=rand()/(float)(j+1);
   printf("addr(s[%d][%d])=%ds[%d][%d]=%f\n",i,j,&s[i][j],i,j,s[i][j]);
  }
  printf("\n\n");
 }

 printf("\n(Prior to free)   s[0][0]=%f\n",s[0][0]);
 free_2(s,3);
 printf("(afterwards) s[0][0]=%f\n\n\n\n",s[0][0]);
}
```

mymemory.h

C:\temp\interfaces1\Project1.exe

```
s=7804096                    addr(s)=2358832
s[0]=7804128                 addr(s[0])=7804096
s[1]=7822112                 addr(s[1])=7804104
s[2]=7823120                 addr(s[2])=7804112
addr(s[0][0])=7804128      s[0][0]=20422.000000
addr(s[0][1])=7804132      s[0][1]=2740.500000

addr(s[1][0])=7822112      s[1][0]=20426.000000
addr(s[1][1])=7822116      s[1][1]=8115.000000


(Prior to free)    s[0][0]=20422.000000
(afterwards) s[0][0]=0.000000
```

Παράδειγμα:

Στην περίπτωση που δημιουργήσουμε δικούς μας τύπους δεδομένων, οι οποίοι θα χρησιμοποιηθούν τόσο από το κύριο πρόγραμμα όσο και από διασυνδέσεις, θα πρέπει να ορισθεί ο τύπος δεδομένου σε μία διασύνδεση και αυτή να περιλαμβάνεται σε όλα τα αρχεία που θα χρειασθούν το νέο τύπο.

- ***Έστω ότι ορίζεται ο τύπος vector για το δισδιάστατο διάνυσμα {x,y} και ζητάμε το εσωτερικό γινόμενο δύο διανυσμάτων. Δημιουργούμε τη διασύνδεση "mytypes.h", στην οποία ορίζουμε τον τύπο και τη συνάρτηση inner_product(vector *a, vector *b).***

- ***Έστω ότι δημιουργούμε τη διασύνδεση "mymath.h", στην οποία θα ορίσουμε μία σειρά μαθηματικών συναρτήσεων όπως π.χ. το μέτρο ενός δισδιάστατου διανυσματος fabs_vector(vector *a). Στη διασύνδεση αυτή θα πρέπει να περιληφθεί η "mytypes.h", ώστε να είναι ορατός ο τύπος vector.***

//interfaces2//…

Αρχείο κεφαλίδας mytypes.h

```
#ifndef _mytypes_h
#define _mytypes_h

   #ifndef _vector_
   #define _vector_
     typedef struct vector   {  float x,y;  };
   #endif

  float inner_product(struct vector *a, struct vector *b);

#endif
```

Έλεγχος ύπαρξης του τύπου

Ορισμός του νέου τύπου

Αρχείο mytypes.c

```
#include "mytypes.h"

float inner_product(struct vector *a, struct vector *b)   {
  return((a->x)*(b->x)+(a->y)*(b->y));
}
```

Αρχείο κεφαλίδας mymath.h

```
#ifndef _mymath_h
#define _mymath_h
#include "mytypes.h"
#include <math.h>
float fabs_vector(struct vector *a);
#endif
```

Για να αναγνωρισθεί ο τύπος vector

Για να αναγνωρισθεί η συνάρτηση sqrt

Αρχείο mymath.c

```
#include "mymath.h "
float fabs_vector(vector *a)
{
 return(sqrt((a->x)*(a->x)+(a->y)*(a->y)));
}
```

Το mytypes.h δε συμπεριλαμβάνεται γιατί περιελήφθη στο mymath.h

Κύριο πρόγραμμα prog2.c

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

main()
{
  struct vector a={3,4},b={1,2};
  printf("The inner product of vector a and b equals  %f\n",
                                                  inner_product(&a,&b));
  printf("The euclidean measure of vector a is %f\n",fabs_vector(&a));
}
```