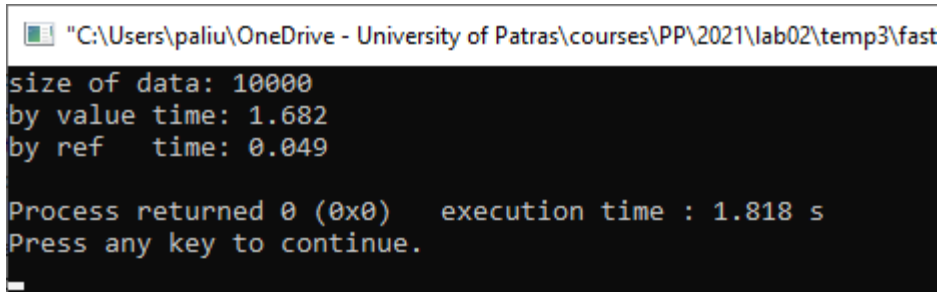
Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς

Τι είναι ταχύτερο;

```c
#include <stdio.h>
#include <time.h>
#define TIMES 10000000

typedef struct test {
            char data[10000];
} Test;

void byvalue(Test a) {
          Test b;
          /* do something */
}

void byref(Test * a ) {
          Test b;
          /* do something */
}
```




```c
int main( void) {
  Test a ;
  int i;
  clock_t start, stop;

  start = clock();
  printf("size of data: %d\n", sizeof (Test));
  for (i=0;i<TIMES; i++)
      byvalue(a);
  stop = clock();
  printf("by value time: %g\n",
      (double) (stop - start)/CLOCKS_PER_SEC);

  start = clock();
  for (i=0;i<TIMES; i++)
         byref(&a);
  stop = clock();
  printf("by ref   time: %g\n",
         (double) (stop - start)/CLOCKS_PER_SEC);

  return 0;
}
```

Κλήση με αξία και Κλήση με αναφορά

Πού αποθηκεύονται παράμετροι και τοπικές μεταβλητές: stack

```
#include <stdio.h>

int f (int );
int g (int );
double h (double );
int w (int, int);


int main(void) {

        f(1);
        g(1);
        h(1.0);
        w(1, 2);
        f(2);
        g(1);

        return 0;
}
```

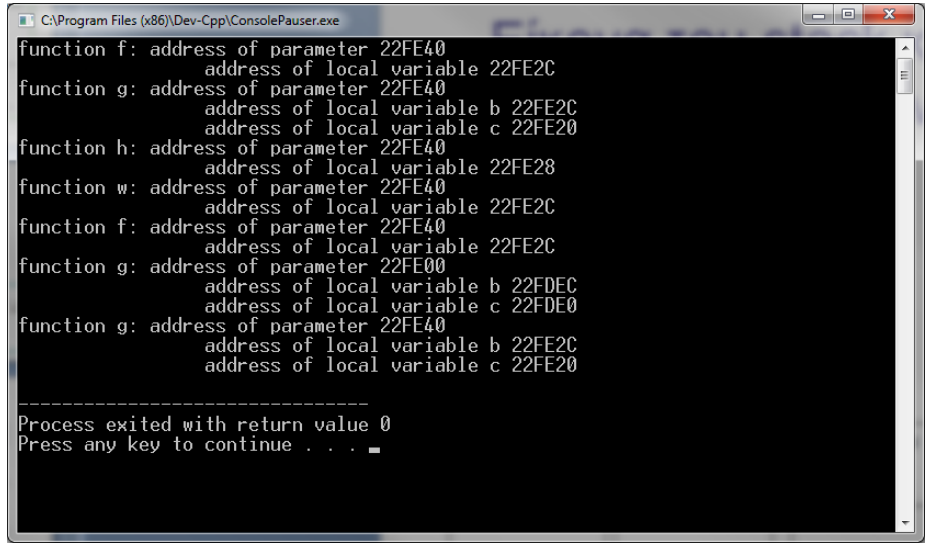
Η f δεν καλεί την g

Η f καλεί την g

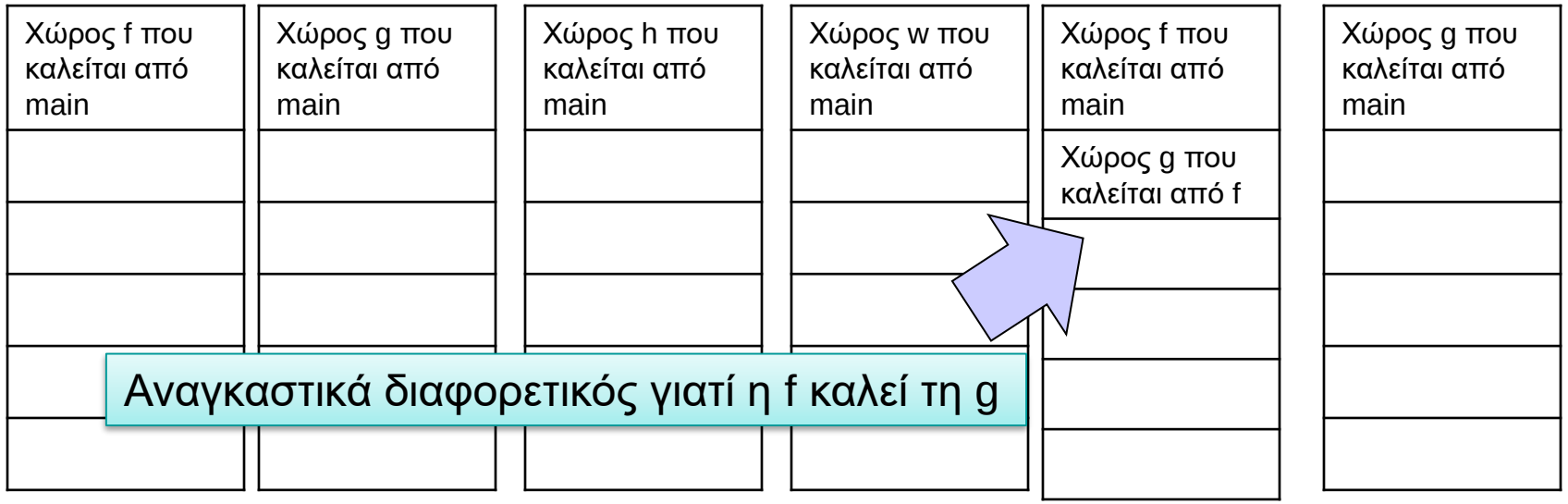
```
int f(int a) {
 int b = 1 ;
 printf("function f: address of parameter %X\n", &a);
 printf("\t\t address of local variable %X\n", &b);
 if (a>1)
    g(a);
return b + a;
}
```

```
int g (int a) {
        int b = 1 ;
        int *c = &a;
        printf("function g: address of parameter %X\n", &a);
        printf("\t\t address of local variable b %X\n", &b);
        printf("\t\t address of local variable c %X\n", &c);
        return b + a ;
}

int w (int a, int c) {
        int b = 1 ;
        printf("function w: address of parameter %X\n", &a);
        printf("\t\t address of local variable %X\n", &b);
        return b + a + c;
}

double h (double a) {
        double b = 2.0 ;
        printf("function h: address of parameter %X\n", &a);
        printf("\t\t address of local variable %X\n", &b);
        return b + a;
}
```

Εικόνα του stack κατά τη διάρκεια εκτέλεσης των συναρτήσεων



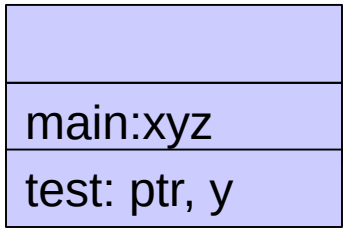
Κάθε φορά διατίθεται ο ίδιος χώρος μνήμης



Αναγκαστικά διαφορετικός γιατί η f καλεί τη g

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

int abc = 7;

int test (void);

int main(void) {
        int xyz ;
        printf("%d\n",test());
        printf("%d\n",test());
        printf("%d\n",test());
        return EXIT_SUCCESS;
}

int test (void) {
        static int x = 0;
        int * ptr ;
        int y = 0;
        x ++ ;
        y ++;
        ptr = malloc (10 * sizeof (int));
        ptr[0] = abc;
        printf("function: x: %d  y:%d ptr[0]:%d\n",x,y, ptr[0]);
        free(ptr);
        return x ;
}
```

Περιοχές μνήμης

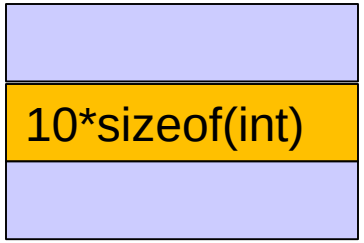


Stack

Τοπικές μεταβλητές, επαναχρησιμοποιούμενη περιοχή

Static/global

Διάρκεια ζωής, όση η εκτέλεση του προγράμματος



Heap

Περιοχή διαθέσιμη για δυναμική διαχείριση

Δυναμική διαχείριση μνήμης στη C

- Δέσμευση μνήμης:
  - **void** *malloc(size_t size);
  - Επιστρέφει δείκτη σε εξασφαλισμένη περιοχή μεγέθους size bytes ή NULL αν δεν υπάρχει τέτοια.

- Απελευθέρωση μνήμης:
  - **void** free(**void** *pointer);

Πώς δουλεύει ο μηχανισμός;

- Χρησιμοποιεί
  - Δεδομένα στο heap
  - Λεπτομερή διαχείριση ανά block
    - Διεύθυνση αρχής
    - Μέγεθος
- Μοιράζεται πληροφορία μεταξύ διαφορετικών συναρτήσεων
  - `malloc( ) , free( )`
  - Πώς γίνεται αυτό;

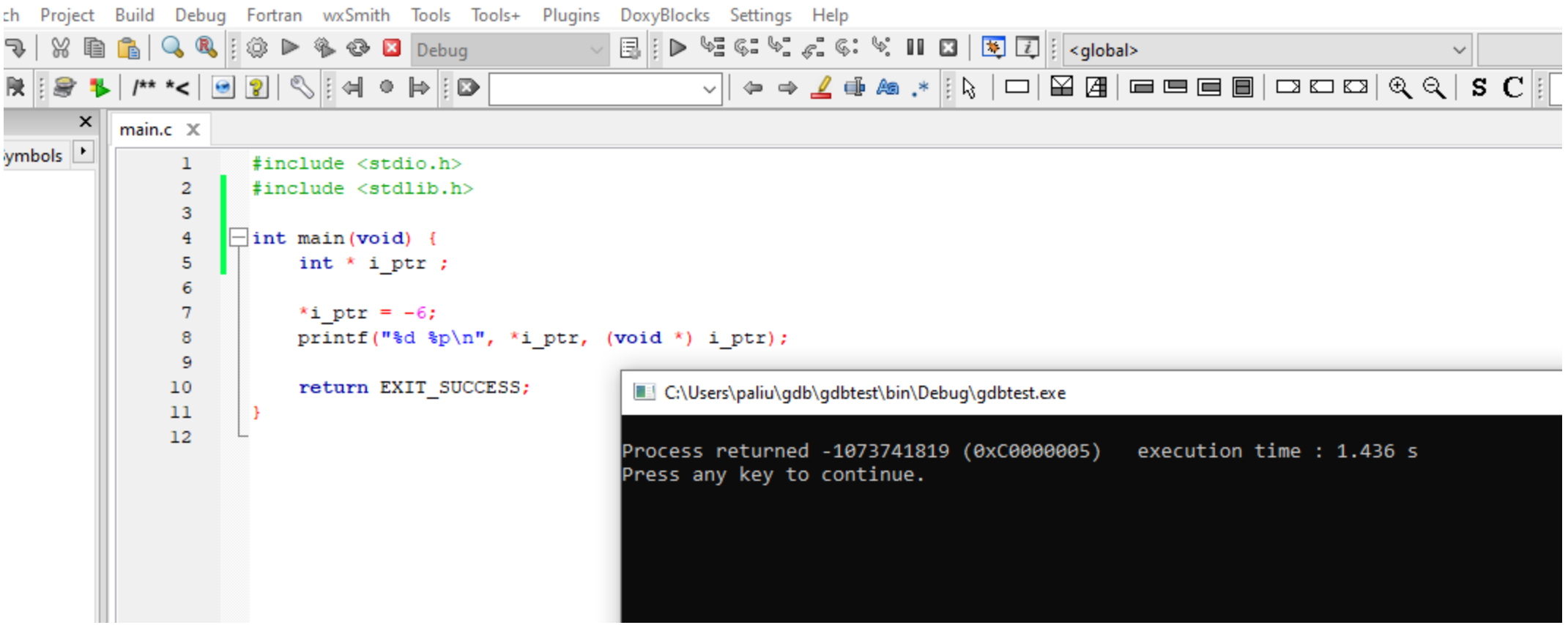
```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main ( void) {

   char matrix[N];

   scanf("%s", matrix);

   printf("Hello %s!\n", matrix);

   return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main (void ) {

   char matrix[N];
   char *dynamicdata;

   scanf("%s", matrix);

   printf("Hello %s!\n", matrix);

   dynamicdata = (char *) malloc( N * sizeof (char));

   scanf("%s", dynamicdata);

   printf("Hello dynamic %s!", dynamicdata);

   return EXIT_SUCCESS;

}
```

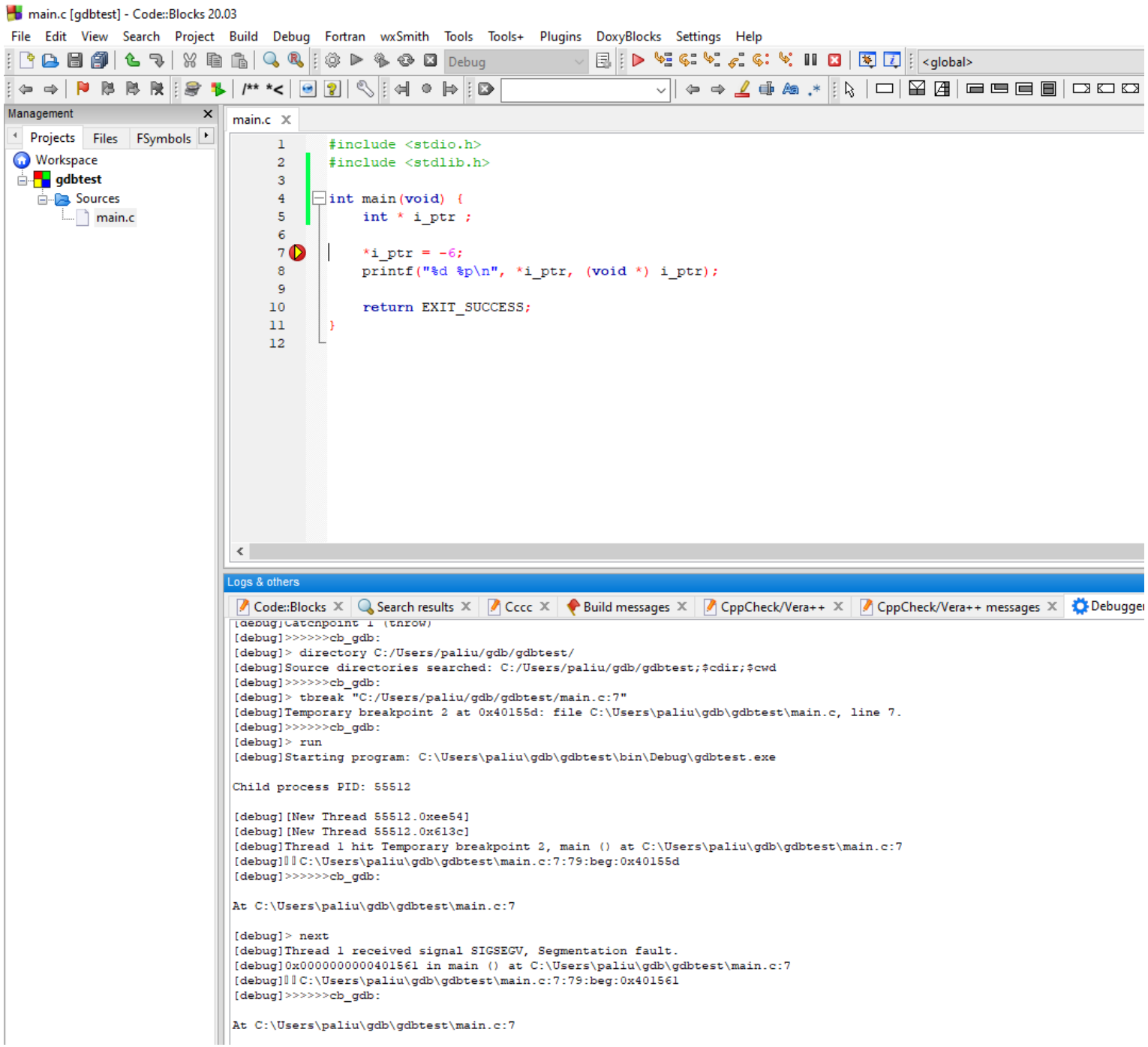
```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main (void ) {

    char matrix[N];
    char *dynamicdata;
    int i;

    scanf("%s", matrix);

    printf("Hello %s!\n", matrix);

    dynamicdata = (char *) malloc( N * sizeof (char));
    scanf("%s", dynamicdata);

    printf("Hello dynamic %s!\n", dynamicdata);

    for (i=0; dynamicdata[i]!=0; i++)
         printf("%c\n", dynamicdata[i]);

    return EXIT_SUCCESS;

}
```

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main ( void) {
    char matrix[N];
    char *dynamicdata;
    int i, nchars;

    scanf("%s", matrix);
    printf("Hello %s!\n", matrix);

    while (1) {
          printf("How many chars?");
          scanf("%d", &nchars);
          dynamicdata = (char *) malloc( nchars * sizeof (char));
          scanf("%s", dynamicdata);
          printf("Hello dynamic %s!\n", dynamicdata);
          for (i=0;dynamicdata[i]!=0;i++) {
                printf("%c\n", dynamicdata[i]);
          }
          free(dynamicdata);
    }
    return EXIT_SUCCESS;
}
```

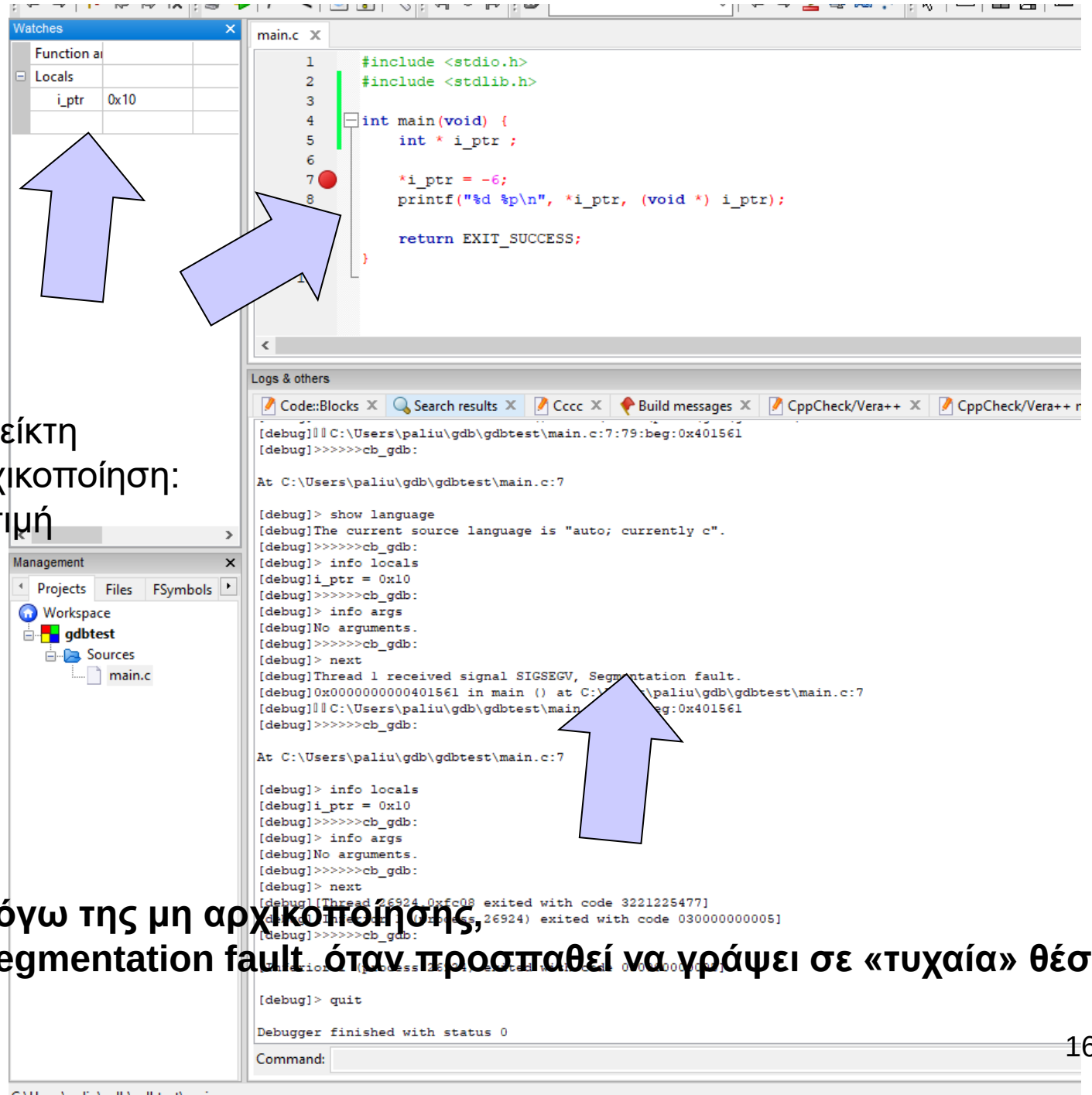
Βρείτε γιατί δεν τρέχει το πρόγραμμα, χρησιμοποιώντας τον debugger!



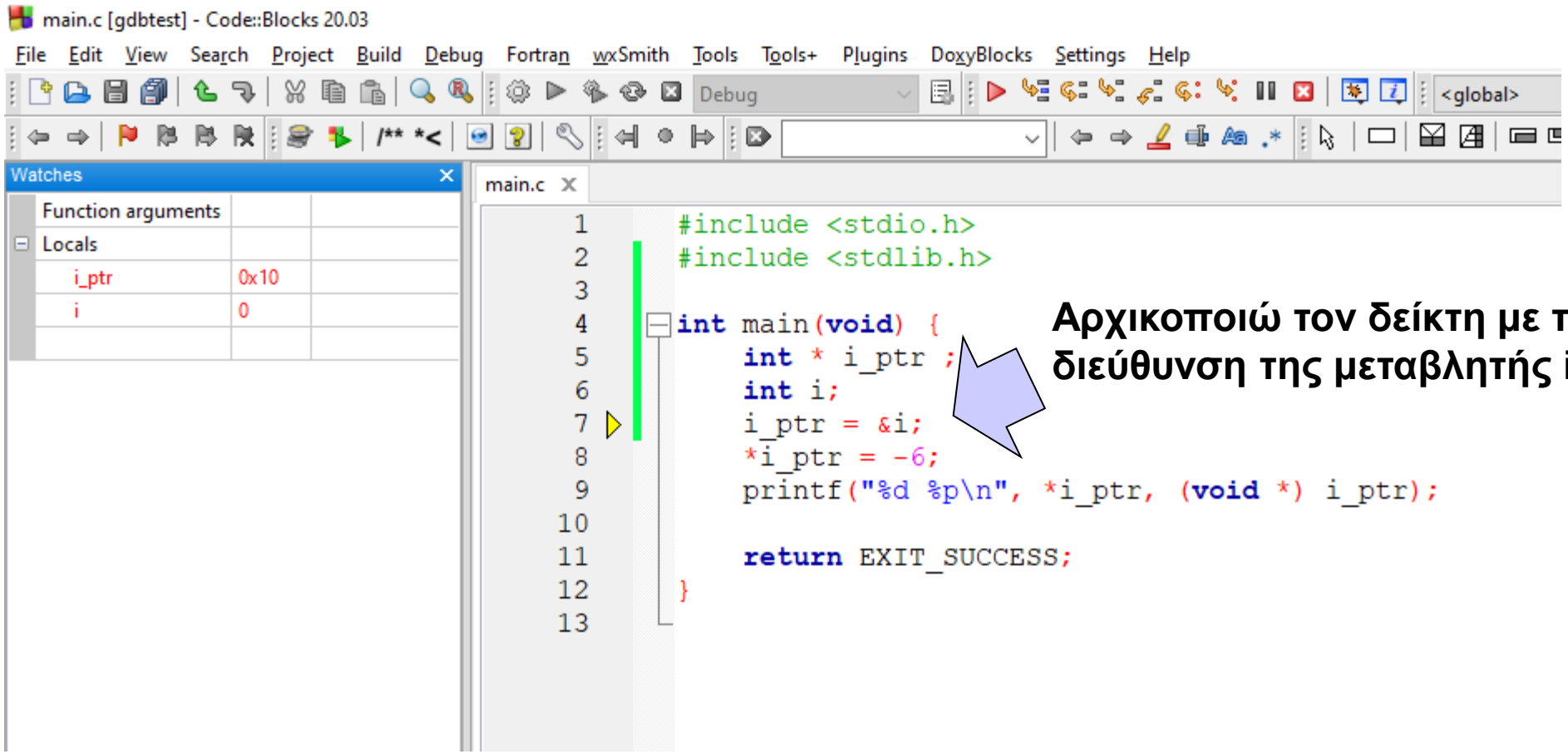
(τα warnings προειδοποιούν!!!)


main.c [gdbtest] - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Debug
<global>
Management
Projects Files FSymbols
Workspace
gdbtest
Sources
main.c
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void) {
5 int * i_ptr ;
6
7 *i_ptr = -6;
8 printf("%d %p\n", *i_ptr, (void *) i_ptr);
9
10 return EXIT_SUCCESS;
11 }
12
Logs & others
Code::Blocks
Search results
Cccc
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Debugger
[debug]Catchpoint 1 (throw)
[debug]>>>>>cb_gdb:
[debug]> directory C:/Users/paliu/gdb/gdbtest/
[debug]Source directories searched: C:/Users/paliu/gdb/gdbtest;$cdir;$cwd
[debug]>>>>>cb_gdb:
[debug]> tbreak "C:/Users/paliu/gdb/gdbtest/main.c:7"
[debug]Temporary breakpoint 2 at 0x40155d: file C:\Users\paliu\gdb\gdbtest\main.c, line 7.
[debug]>>>>>cb_gdb:
[debug]> run
[debug]Starting program: C:\Users\paliu\gdb\gdbtest\bin\Debug\gdbtest.exe
Child process PID: 55512
[debug][New Thread 55512.0xee54]
[debug][New Thread 55512.0x613c]
[debug]Thread 1 hit Temporary breakpoint 2, main () at C:\Users\paliu\gdb\gdbtest\main.c:7
[debug]▯▯C:\Users\paliu\gdb\gdbtest\main.c:7:79:beg:0x40155d
[debug]>>>>>cb_gdb:
At C:\Users\paliu\gdb\gdbtest\main.c:7
[debug]> next
[debug]Thread 1 received signal SIGSEGV, Segmentation fault.
[debug]0x0000000000401561 in main () at C:\Users\paliu\gdb\gdbtest\main.c:7
[debug]▯▯C:\Users\paliu\gdb\gdbtest\main.c:7:79:beg:0x401561
[debug]>>>>>cb_gdb:
At C:\Users\paliu\gdb\gdbtest\main.c:7

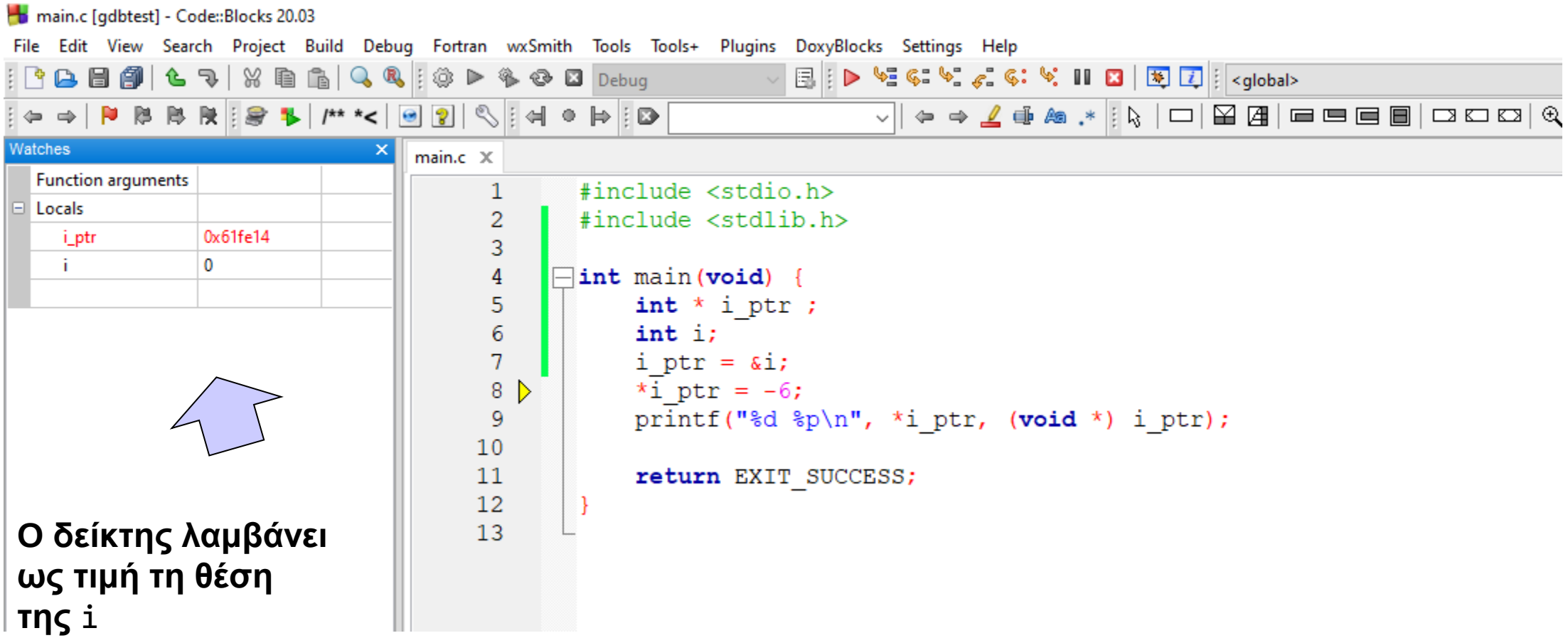
Δήλωση δείκτη
Χωρίς αρχικοποίηση:
«τυχαία» τιμή



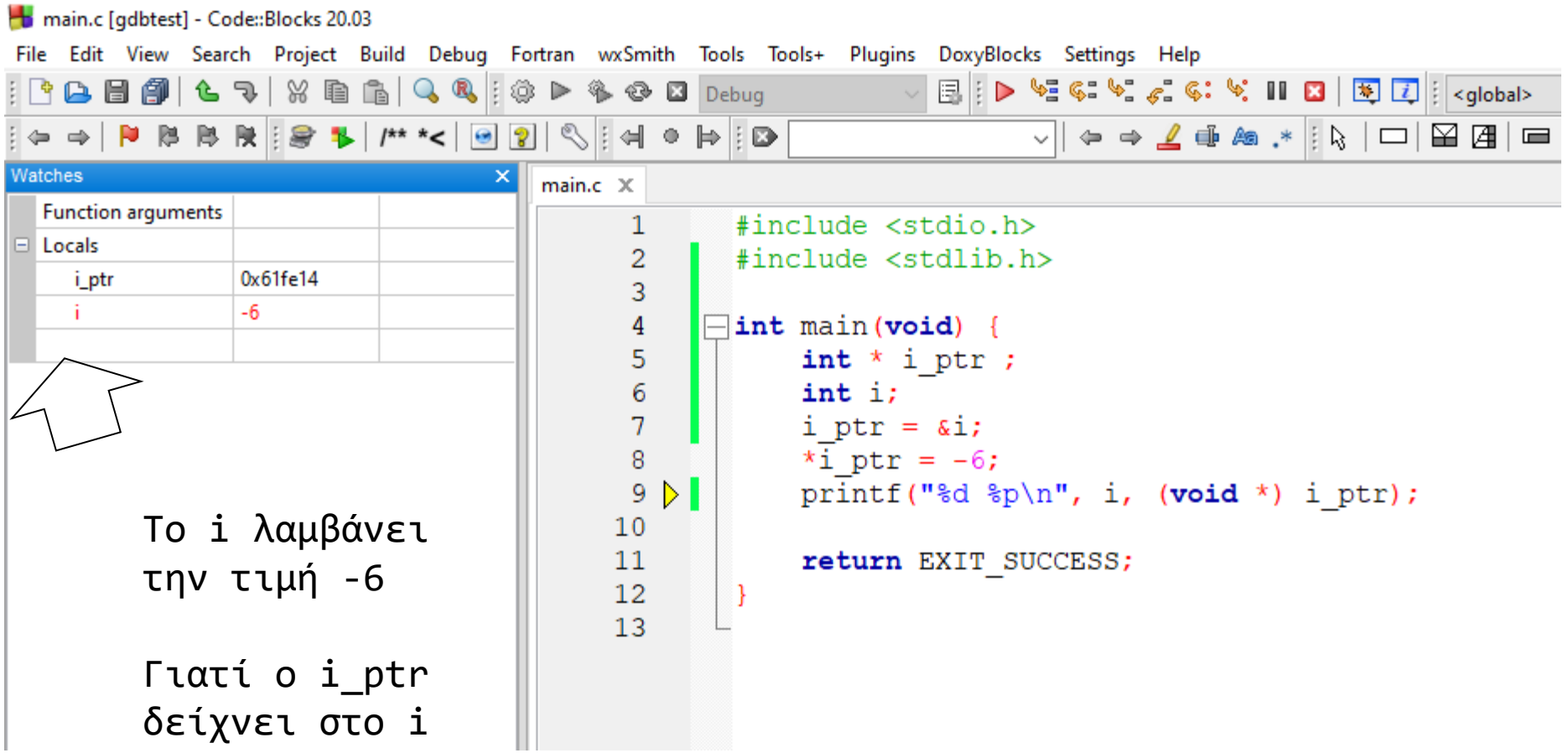
Λόγω της μη αρχικοποίησης,
Segmentation fault όταν προσπαθεί να γράψει σε «τυχαία» θέση μνήμης



Αρχικοποιώ τον δείκτη με την διεύθυνση της μεταβλητής i

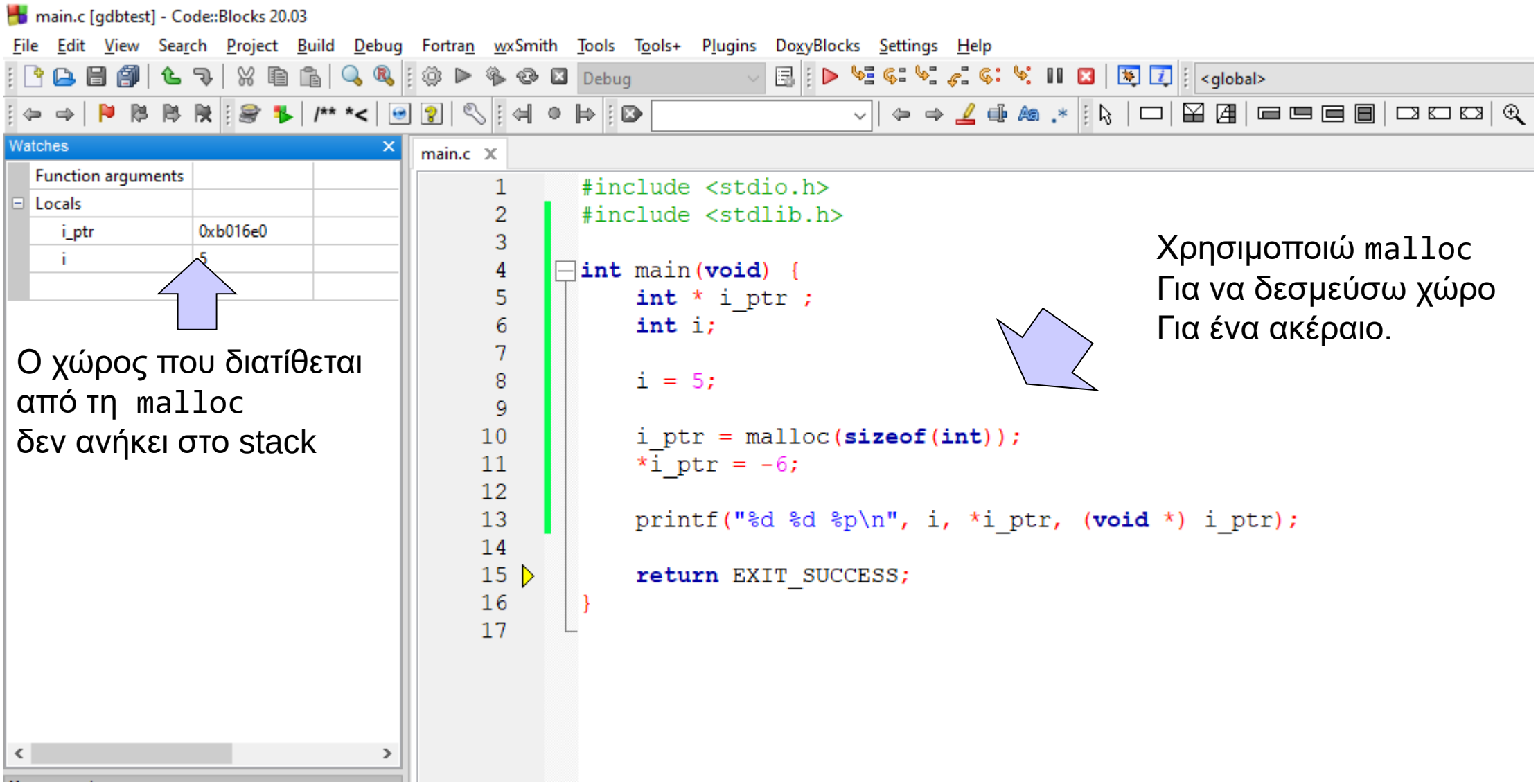


Ο δείκτης λαμβάνει ως τιμή τη θέση της `i`



Το i λαμβάνει την τιμή -6

Γιατί ο i_ptr δείχνει στο i



Ο χώρος που διατίθεται
από τη `malloc`
δεν ανήκει στο stack

Χρησιμοποιώ `malloc`
Για να δεσμεύσω χώρο
Για ένα ακέραιο.

Μέγεθος stack

- Οι πίνακες χρησιμοποιούν το stack
  - Όπως οι αυτόματες μεταβλητές στη C
- Περιορισμένο μέγεθος stack
- Μπορεί να αυξηθεί
  - με οδηγία στο linker
    - `-Wl,--stack,<μέγεθος σε bytes>`
  - Σε linux, πχ με την εντολή ulimit

Global/static

- Οι πίνακες global, static δεν χρησιμοποιούν το stack
- Θέματα που σχετίζονται με το OS, τον compiler, την έκδοση κ.ά.

Δυναμική διαχείριση

- Ο χώρος μνήμης που διατίθεται με `malloc`, `realloc`, `calloc` δεν είναι στο stack

- Μπορώ να διαλέξω στο Project Options>Compiler>Code generation το pointer width (32 bit / 64 bit) σε περίπτωση που ενδιαφέρει διαχείριση μνήμης > 4GB

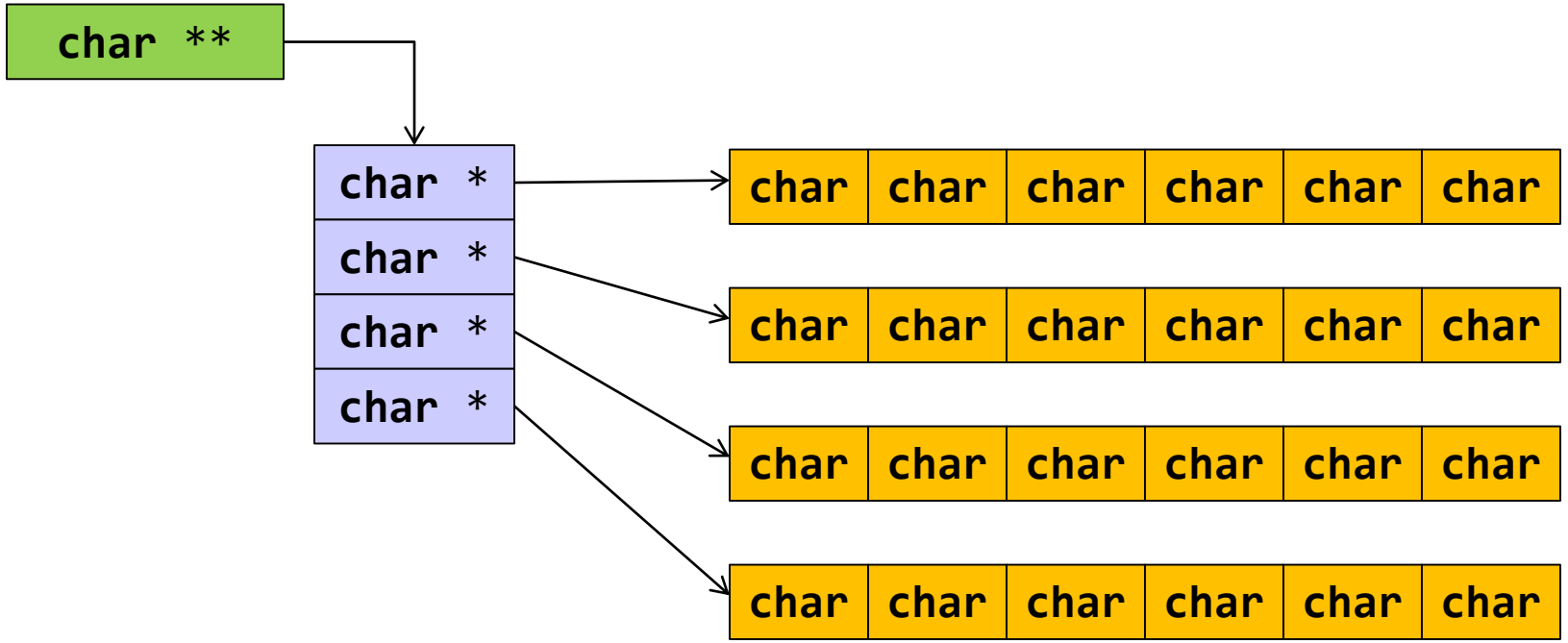
- 32 bit pointer → 2^{32} θέσεις = 4×2^{30} = 4 G
- 64 bit pointer → 16 Exabytes

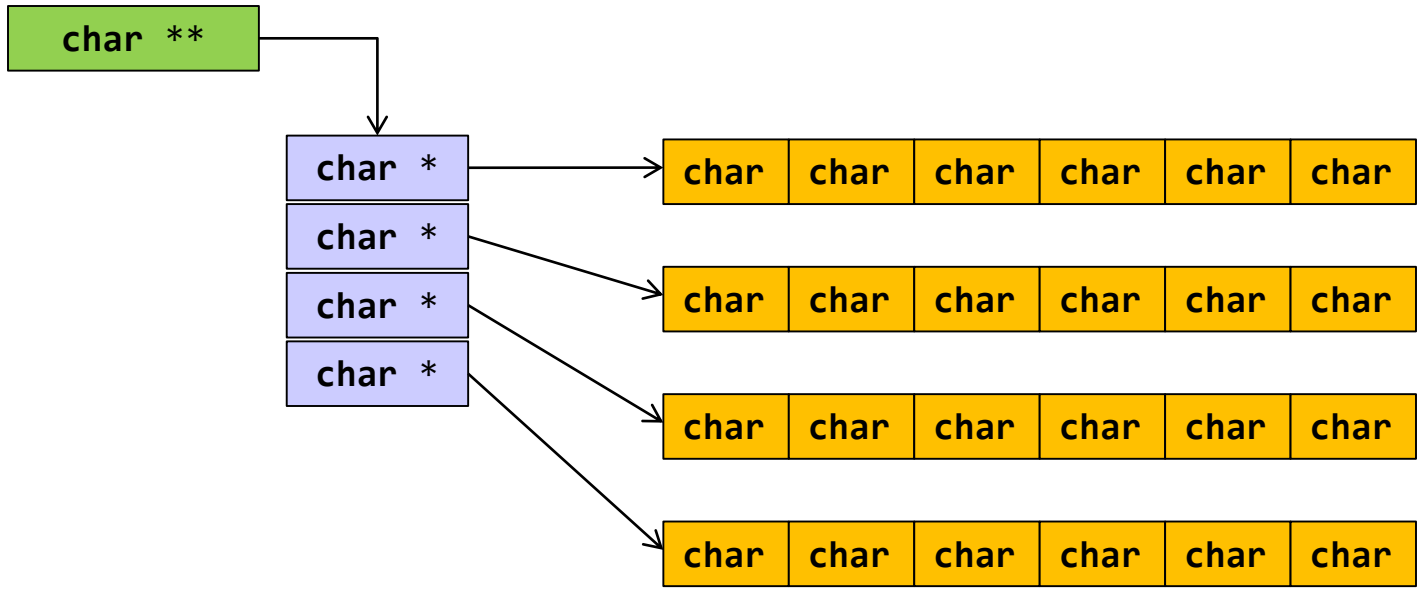
realloc

- **void** *realloc(**void** **ptr*, **size_t** *size*);
- Αλλάζει το μέγεθος περιοχής μνήμης με αρχή τη διεύθυνση *ptr* ώστε να έχει τελικό μέγεθος *size* bytes
- Μπορεί να επεκτείνει τη διαθέσιμη περιοχή αν είναι εφικτό ή να βρει νέα περιοχή μεταφέροντας δεδομένα.
- Η περιοχή μνήμης θα πρέπει να έχει ήδη ανατεθεί πριν την κλήση της `realloc` ή ο `ptr` να έχει την τιμή `NULL`
- Επιστρέφει `NULL` σε περίπτωση αποτυχίας

- Να γραφεί πρόγραμμα για ανάγνωση θετικών αριθμών και τοποθέτησή τους σε δυναμικό πίνακα. Όταν δοθεί ως είσοδος 0 ή αρνητικός αριθμός, εκτυπώνονται όσοι αριθμοί έχουν εισαχθεί νωρίτερα.

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

int main(void) {

        int *datatable = NULL;
        int d;
        int numbers = 0;
        int i;

        while (scanf("%d", &d), d>0) {
                numbers ++;
                datatable = realloc(datatable, numbers*sizeof(int));
                datatable[numbers - 1] = d;
        }

        for (i=0; i< numbers; i++)
                printf("%d\n", datatable[i]);

        free(datatable);

        return EXIT_SUCCESS;
}
```


char **
char *
char *
char *
char *
char char char char char char
char char char char char char
char char char char char char
char char char char char char



- Δημιούργησε χώρο για τη νέα λέξη
- Δημιούργησε χώρο για τη διεύθυνση της νέας λέξης
- Αποθήκευσε τη διεύθυνση της νέας λέξης

realloc

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CHARS 10
int main( void ) {
    char **mytext = NULL;
    int words = 0;
    char word[CHARS] = "";
    int i;

    while (scanf("%s", word), strcmp(word,"TELOS")) {
           words++;
           mytext = realloc(mytext, words*sizeof(char *));
           mytext[words-1] = malloc (CHARS*sizeof(char));
           strcpy(mytext[words-1], word);
    }

    for (i=0; i<words; i++)
         printf("%s\n", mytext[i]);

    return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CHARS 10
int main( void ) {
        char **mytext = NULL;
        int words = 0;
        char *word;
        int i;

        while (scanf("%s", word=malloc(CHARS*sizeof(char))),
                                        strcmp(word,"TELOS")) {
                words++;
                mytext = realloc(mytext, words*sizeof(char *));
                mytext[words-1] = word;
        }
        free(word);

        for (i=0; i<words; i++)
                printf("%s\n", mytext[i]);

        return EXIT_SUCCESS;
}
```

Διαχείριση πίνακα χαρακτήρων μεταβλητού μεγέθους

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

char * getname(void) ;

int main( ) {
  char other[] = "DO NOT ERASE ME";
  char *name;

  name = getname();

  printf("name : %s at %X\n", name, name);
  printf("size of name %d chars\n", strlen(name));
  printf("other: %s at %X\n", other, other);

  return EXIT_SUCCESS;
}
```

```
char *getname(void ) {
 int i = 0;
 int c ;
 char *more = (char *) malloc(1 * sizeof (char));

 while ((c = getchar())!='\n') {
  more[i] = c;
  if ((more = (char *)realloc(more, (1+(++i))*(sizeof (char))))==NULL)
                {
                        printf("reallocation failed!");
                        exit(1);
                 }
  printf("more: %X\n", more);
  }

  more[i] = '\0';
  printf("\ncharacters read i: %d\n", i);

  return more;
}
```

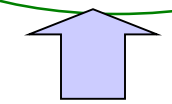
Η δομή

```
struct <όνομα δομής> {
   <τύπος 1ου μέλους> <όνομα 1ου μέλους>;
   <τύπος 2ου μέλους> <όνομα 2ου μέλους>;
   <τύπος 3ου μέλους> <όνομα 3ου μέλους>;
   …
   <τύπος nου μέλους> <όνομα nου μέλους>;
} <λίστα ονομάτων μεταβλητών> ;
```

Ορισμός τύπου **struct** address

```
struct address {
    char street[20];
    int number;
    int code;
    char city[20];
};
```

```
struct address myaddress ;
```

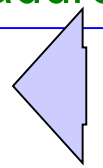


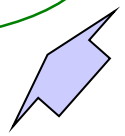
χρήση τύπου **struct** address

Παράδειγμα

```
struct address {
    char
      street[20]
      ;
    int number;
    int code;
    char
      city[20];
} myaddress;
```

συνδυασμένος ορισμός τύπου **struct** address και δήλωση μεταβλητής τύπου **struct** address





όνομα μεταβλητής τύπου **struct** address

Αρχικοποίηση μεταβλητών

```
struct address {
 char street[20];
 int number;
 int code;
 char city[20];
} ;

struct address myaddress = {"Anthewn", 1, 123,
  "Patra" };
```

typedef

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct address {
 char street[20];
 int number;
 int code;
 char city[20];
} ;
typedef struct address Address;

void report (Address) ;
Address readaddress (void) ;

int main (void ) {

Address myaddress ;

     myaddress = readaddress( );
     report (myaddress);

return EXIT_SUCCESS;
}
```

```c
Address readaddress (void)  {
 Address localaddress;

     printf("Odos:\t");
     scanf("%s", localaddress.street);
     printf("Ar. :\t");
     scanf("%d", &localaddress.number);
     printf("Code:\t");
     scanf("%d", &localaddress.code);
     printf("Poli:\t");
     scanf("%s", localaddress.city);

   return localaddress;
}

void report (Address local) {
     printf("Odos: %20s\n", local.street);
     printf("Ar. : %20d\n", local.number);
     printf("T.C.: %20d\n", local.code);
     printf("Poli: %20s\n", local.city);
   return ;
}
```

Ένθεση Δομών

```
struct person {
  char firstname[20];
  char surname[20];
  int age;
  struct address homeaddress;
  struct address bussinessaddress;
} ;
```

Δείκτες ως μέλη δομών

```
struct person {
  char firstname[20];
  char surname[20];
  int age;
  struct address homeaddress;
  struct address bussinessaddress;
  struct person *next_person;
} ;
```

Συνοπτικά οι δομές (1)

- Ορισμός δομής ⇒ ορισμός τύπου

```
struct test {
        int a;
        char d[10]; };
struct test mytest;
```

- Επιστρέφονται και περνούν κατ' αξία από συναρτήσεις.

```
struct test dosomething(int a, struct test b) {
<κώδικας>
}
```

- Βοηθάει το **typedef**

```
typedef struct test Test;
Test dosomething(int a, Test b) {
<κώδικας>
}
```