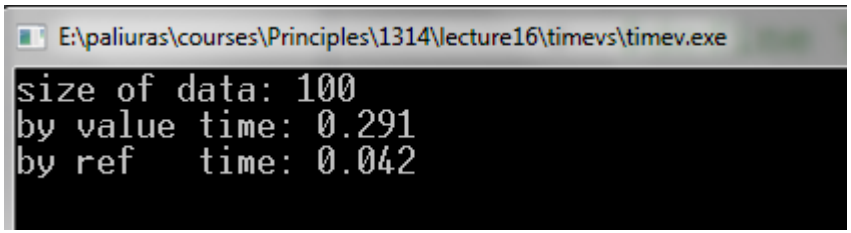
Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς

Τι είναι ταχύτερο;

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

typedef struct test {
            char data[100];
} 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;
}
```

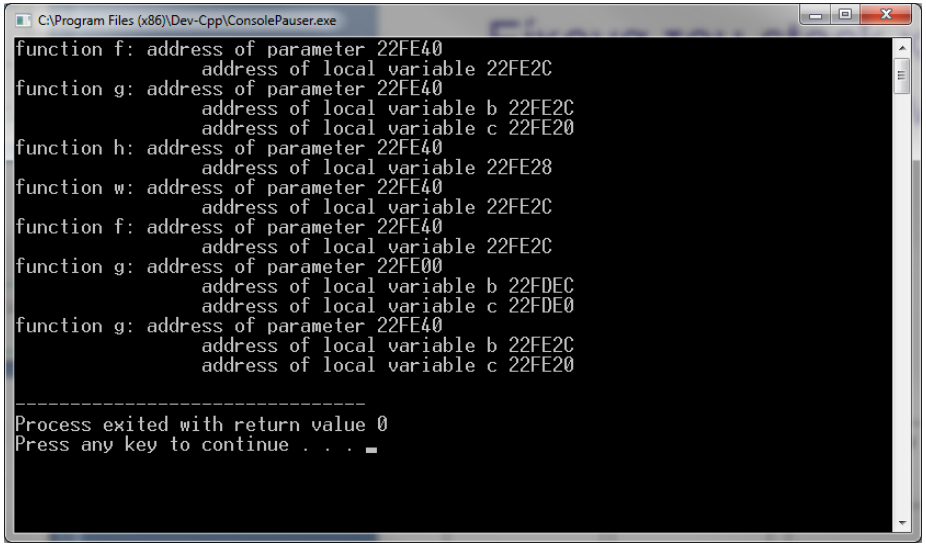
Η f δεν καλεί την g

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



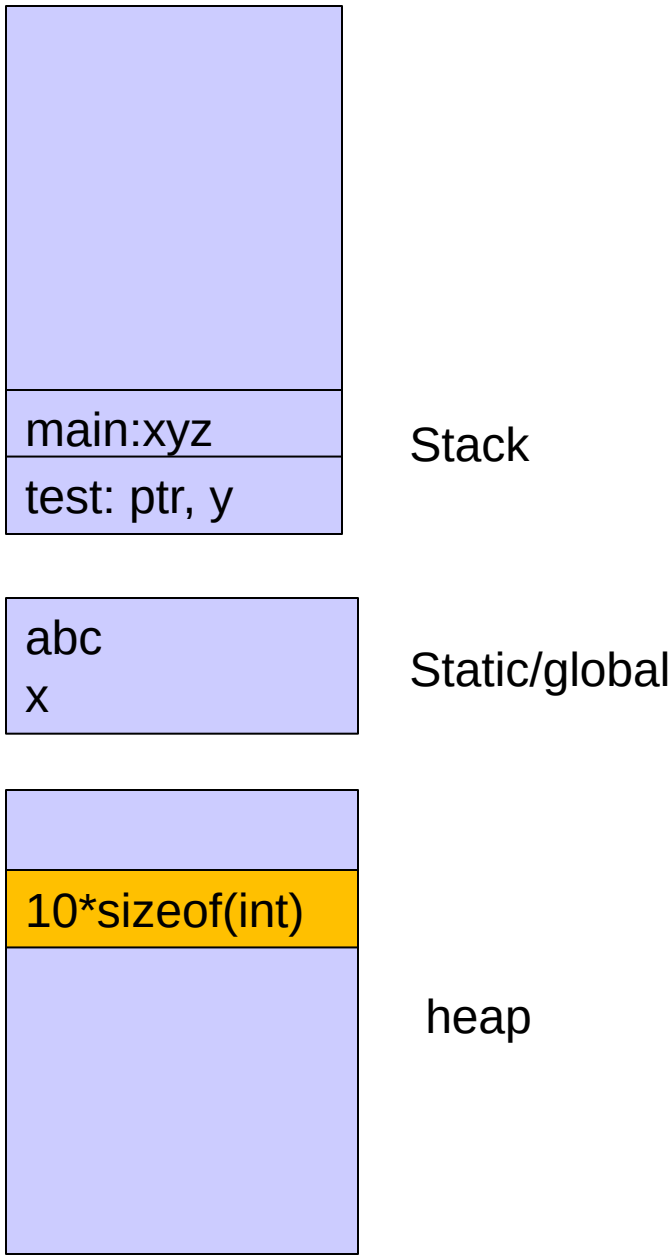
Κάθε φορά διατίθεται ο ίδιος χώρος μνήμης

Χώρος f που καλείται από main	Χώρος g που καλείται από main	Χώρος h που καλείται από main	Χώρος w που καλείται από main	Χώρος f που καλείται από main	Χώρος g που καλείται από main
				Χώρος g που καλείται από f	

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



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;