Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

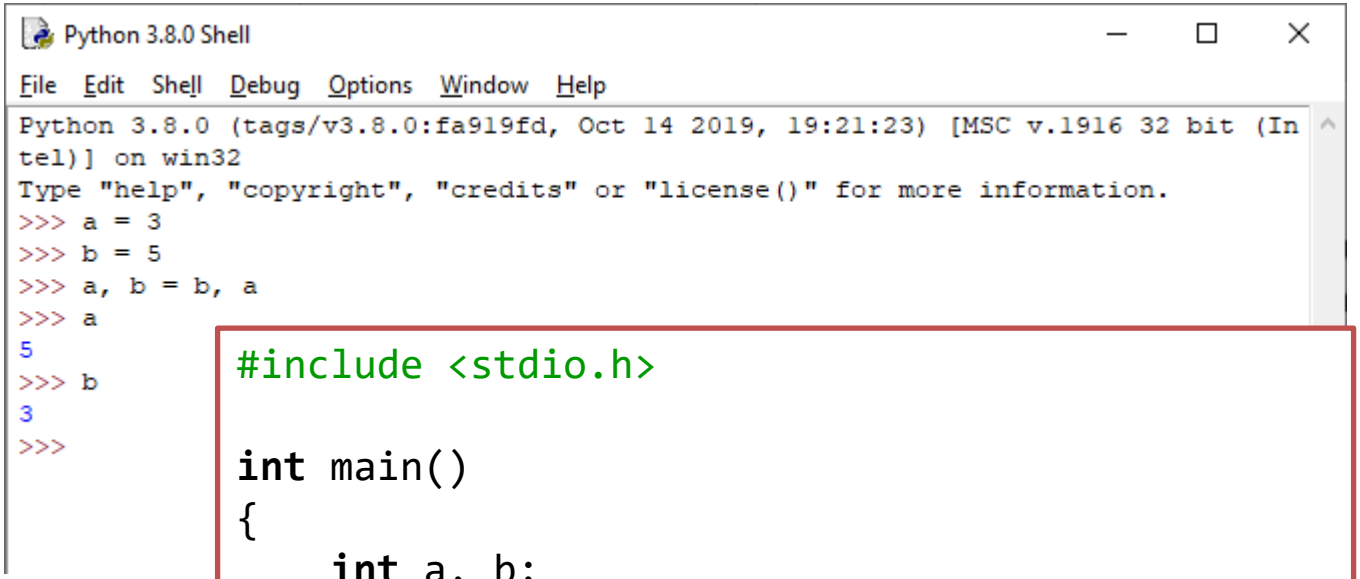
Μηχανισμοί κλήσης συναρτήσεων

Τι θα τυπωθεί;

```c
#include <stdio.h>

int myfun(int, int);

int main ( ) {
   int a = 3, b = 3;

   myfun(a, b);
   printf("main: a:%d b:%d\n", a, b);

   return 0;
}
```

```c
int myfun(int a, int b) {
   a++;
   b++;
   printf("myfun: a:%d b:%d\n", a, b);
   return 0;
}
```

Κλήση συνάρτησης με αξία
(Call by value)




```c
#include <stdio.h>

int main()
{
    int a, b;

    a = 3 ;
    b = 5;

    printf("%d %d\n",a, b);

    a, b = b ,a ;

    printf("%d %d\n",a, b);

    return 0;
}
```

```
3 5
3 5

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

Κλήση συνάρτησης κατ' αναφορά (call by reference)

- πρότυπο:

```
void swap(int * a_ptr, int *b_ptr);
```

- κλήση: `swap( &value1, &value2 );`

1. Η συνάρτηση επενεργεί **απευθείας στις θέσεις μνήμης** των μεταβλητών που χρησιμοποιούνται ως πραγματικά ορίσματα.
2. **Δεν** δημιουργούνται τοπικά αντίγραφα των δεδομένων.
   - Ισχύει για τις διευθύνσεις;
3. Στη C, ουσιαστικά, υλοποιείται ως κατ' αξία πέρασμα διευθύνσεων.

Ορισμός συνάρτησης swap( )

```
void swap(int * a_ptr, int * b_ptr) {
    int temp ;
    temp = *b_ptr ;
    *b_ptr = *a_ptr ;
    *a_ptr = temp ;
    return ;
}
```

Παράδειγμα χρήσης κλήσης κατ΄αναφορά

```c
#include <stdio.h>
void swap (int *, int *);

int main ( ) {
 int value1 = 5;
 int value2 = 3;

 printf("value1: %d value2: %d\n",
                value1, value2);

 swap (&value1, &value2) ;

 printf("value1: %d value2: %d\n",
                value1, value2);
 return 0;
}
```

```c
void swap(int *a_ptr, int *b_ptr)
{
 int temp;
 temp = *a_ptr  ;
 *a_ptr  = *b_ptr  ;
 *b_ptr  = temp ;
  return ;
}
```

```
Paliuras@MOB ~/nlect9
$ ./a
value1: 5 value2: 3
value1: 3 value2: 5
```

Μερικές διαφορές

- Μηχανισμός κλήσης κατ' αξία (call by value)
  - δημιουργούνται τοπικά αντίγραφα των τιμών των ορισμάτων
  - Αν τα ορίσματα έχουν μήκος πολλών bytes, επιβαρύνεται η εκτέλεση.
  - Η συνάρτηση δεν μπορεί να αλλάξει τις τιμές ορισμάτων στο σημείο κλήσης.
- Μηχανισμός κλήσης με αναφορά (call by reference)
  - δεν δημιουργούνται τοπικά αντίγραφα τιμών,
    - μόνο των διευθύνσεων !!!
    - η συνάρτηση ενημερώνεται για τη θέση μνήμης στην οποία είναι αποθηκευμένο ένα όρισμα.
    - μπορεί να είναι ταχύτερο
  - είναι δυνατόν να αλλάξουν οι τιμές ορισμάτων στο σημείο κλήσης.
    - χρειάζεται προσοχή, μπορεί να γίνει εκ παραδρομής.
    - Είναι ένας τρόπος να *επιστρέψω περισσότερες από μία τιμές*
    - *Για τη C, καταχρηστικά λέγεται έτσι: επί της ουσίας υλοποιείται ως κατ' αξία πέρασμα δεικτών.*

Δείκτες σε ακέραιο

```c
#include <stdio.h>

void sumdiff(int, int, int *, int *);

int main( )
{
int a = 3, b = 4 ;
int sum;
int diff;

  sumdiff(a,b, &sum, &diff);

  printf("%d %d\n", sum, diff);

  return 0;
}
```

Διευθύνσεις των μεταβλητών που θα αλλάξουν τιμές

Η s αρχικοποιείται στην τιμή &sum

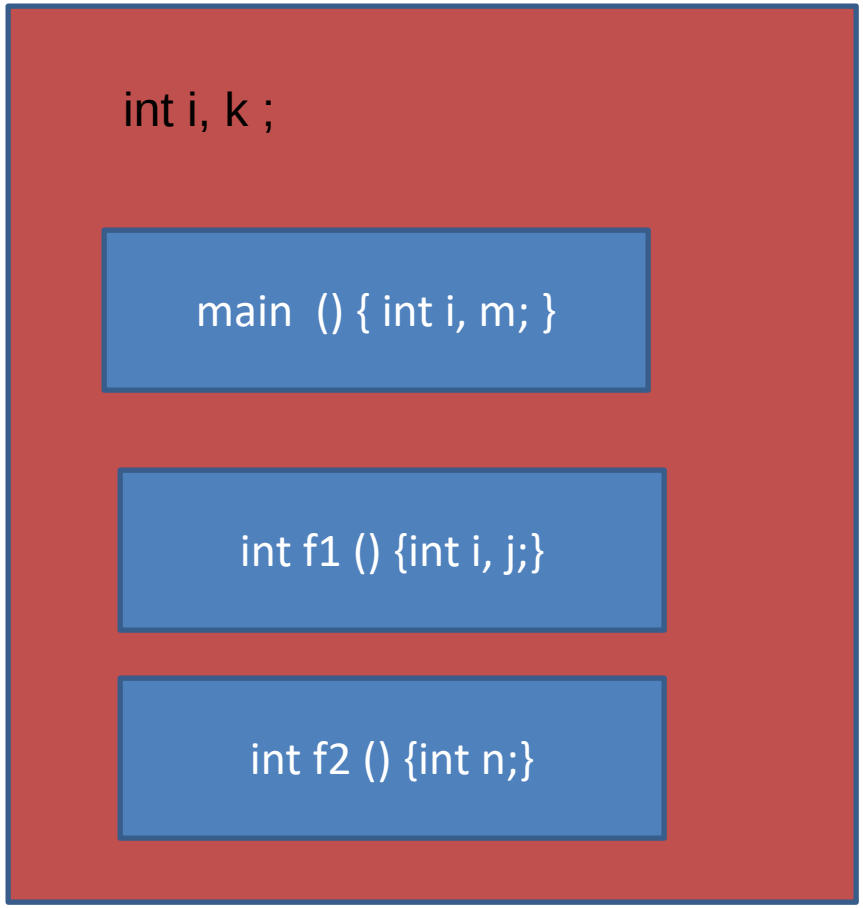
```c
void sumdiff(int a, int b, int *s, int *d) {
     *s = a + b;
     *d = a - b;
     return ;
     }
```

Αλλάζουν τιμές τα ***περιεχόμενα*** των θέσεων με **διευθύνσεις** s, d.

Στη sumdiff( ) ορίζονται ***νέες μεταβλητές*** a, b

Οι a,b της main() είναι διαφορετικές μεταβλητές

Εμβέλεια ονομάτων



καθολική

Τι θα γίνει αν αφαιρεθεί το /* */ από τη δήλωση;

τοπική

καθολική i

Τοπική m

```c
#include <stdio.h>
#include <stdlib.h>
int f1(int);

int i = 5;

int main(int argc, char *argv[])
{
 /*int i = 2;*/
 int k = 1;
   i++;

   f1(i);
   f1(k);
  system("PAUSE");
  return 0;
}

int f1(int m) {
    printf("%d\n", m+i);
    return 0;
}
```

Διάρκεια ζωής μεταβλητής

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

void function (void);
void function1 (void);
int main () {

        function1();
        function ( ) ;
        function1 ( ) ;
        function ( ) ;

        return 0;
}

void function1() {
int j = 0;
    j++;
    printf("function1, j: %d at %X\n", j, &j);

}
void function ( ) {
int i ;
    i++ ;
    printf("function, i: %d at %X\n", i, &i);
    printf("%d\n", i);
}
```

Η τοπική μεταβλητή

i δεν αρχικοποιείται!

Τι βγαίνει εδώ;

i, j, είναι τοπικές μεταβλητές

Διάρκεια ζωής: όσο εκτελείται η συνάρτηση

Η θέση μνήμης επαναχρησιμοποιείται

```
void function1 (void);
void function2 (void);
void function(void);
main ( ) {
        function();
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
}
void function() {
int k = 0;
}
void function1( ) {
int i;
        i++ ;
        printf("f1:%d\n", i);
}
void function2( ) {
int j;
        j++ ;
        printf("f2:%d\n", j);
}
```

Διάρκεια μεταβλητής (2)

Αν η κλήση της function1( ) γίνεται εναλλάξ με την function2( ), η συμπεριφορά αλλάζει...

Εμφανίζονται εξαρτήσεις (είναι δυνατόν να ...)

```
void function1 (void);
void function2 (void);

main ( ) {
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
}
void function1 () {
int i = 0;
        i++ ;
        printf("f1:%d\n", i);
}
void function2 () {
int j = 0;
        j++ ;
        printf("f2:%d\n", j);
}
```

Λύση;;;

Αρχικοποιώντας τις μεταβλητές

κάθε φορά που καλείται η συνάρτηση,

οι τοπικές μεταβλητές μηδενίζονται κάθε

φορά που καλείται η συνάρτηση.

```
void function1 (void);
void function2 (void);

main ( ) {
        function1( )  ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
}
void function1 ( ) {
static int i = 0;
        i++ ;
        printf("f1:%d\n", i);
}
void function2 ( ) {
static int j = 0;
        j++ ;
        printf("f2:%d\n", j);
}
```

Η λέξη κλειδί **static** σε δηλώσεις τοπικών μεταβλητών

static: ο χώρος μνήμης της μεταβλητής δεν αποδεσμεύεται όταν ολοκληρωθεί μια εκτέλεση της συνάρτησης.

Άλλη χρήση της **static**: Ορισμός εμβέλειας αρχείου

- αρχείο πηγαίου κώδικα: code1.c

```
int func (int);
void report(void);

main () {
int i ;

for ( i=0; i< 5; i++)
    printf("%d\n", func(i));

report();
}
```

- αρχείο πηγαίου κώδικα: code2.c

```
static int sum = 0;

int func(int k) {
int i;
for (i=0;i<5; i++)
    sum += k ;  /* sum = sum + k */
return 2 * k;
}
void report() {
    printf("%d\n", sum);
}
```