ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

ΑΡΧΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Κεφάλαιο 12

Επιμέλεια:
Βασίλης Παλιουράς , Αναπληρωτής Καθηγητής
Ευάγγελος Δερματάς , Αναπληρωτής Καθηγητής
Σταύρος Νούσιας , Βοηθός Ερευνητή
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τυπου άδειας χρησης, η άδεια χρησης αναφερεται ρητώς.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου των διδασκόντων καθηγητών.
- Το έργο «Ανοικτά Ακαδήμαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών» έχει χρηματοδοτηθεί μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκη Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικους πόρους.

Ανάπτυξη

Το παρόν εκπαιδευτικό υλικό αναπτύχθηκε στο τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
//
// C99 comment style
// Demonstration of C99 VLAs as parameters in functions
//
void readarray(int rows, int cols, int x[rows][cols]);
void printarray(int rows, int cols, int x[rows][cols]);

int main(int argc, char *argv[]) {
            int a[5][5];
            readarray(5,  5, a);
            printarray(3, 5, a);
            return 0;
}

void readarray(int rows, int cols, int x[rows][cols]) {
    int i, j;

    for (i = 0; i< rows; i++)
            for (j = 0; j < cols; j++)
                                    x[i][j] = -i*cols - j ;
}

void printarray(int rows, int cols, int x[rows][cols]) {
    int i, j;

    for (i = 0; i< rows; i++) {
            for (j = 0; j < cols; j++)
                printf("%4d", x[i][j]) ;
            printf("\n");
            }
```

Δεν υποστηρίζονται
Από όλους τους
Compilers!!!

Το mvsc δεν είναι
C99 compiler

Εδώ γράφουμε C90
Εκτός αν ζητείται αναλυτικά

Μηχανισμοί κλήσης συναρτήσεων

Τι θα τυπωθεί;

```
#include <stdio.h>

int myfun(int, int);

int main ( ) {
      int a = 3, b = 3;
      myfun(a, b);
      prin
      return 0;
}

int myfun(int a, int b) {
      a++;
      b++;
      printf("myfun: a:%d b:%d\n", a, b);

      return 0;
}
```

Κλήση συνάρτησης κατ' αναφορά
(call by reference)

πρότυπο: **void** swap(**int** *a, **int** *b);

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.
3. Ισχύει για τις διευθύνσεις;
4. Στη C, ουσιαστικά, υλοποιείται ως κατ' αξία πέρασμα διευθύνσεων.

Ορισμός συνάρτησης swap( )

```
void swap(int *a, int *b) {
        int temp ;
        temp = *b ;
        *b = *a ;
        *a = temp ;
}
```

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 , int *b) {
int temp;
 temp = *a ;
 *a = *b ;
 *b = temp ;

}
```

```
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 είναι κατ' αξία πέρασμα δεικτών.

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
void sumdiff(int, int, int *, int *);

int main(int argc, char *argv[])
{
int a = 3, b =4 ;
int sum;
int diff;

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

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

  system("PAUSE");
  return 0;
}

void sumdiff(int a, int b, int *s, int *d) {
     *s = a + b;
     *d = a - b;
     a++;
     b++;
     printf("%d %d\n",a,b);
     }
```

Δείκτες σε ακέραιο

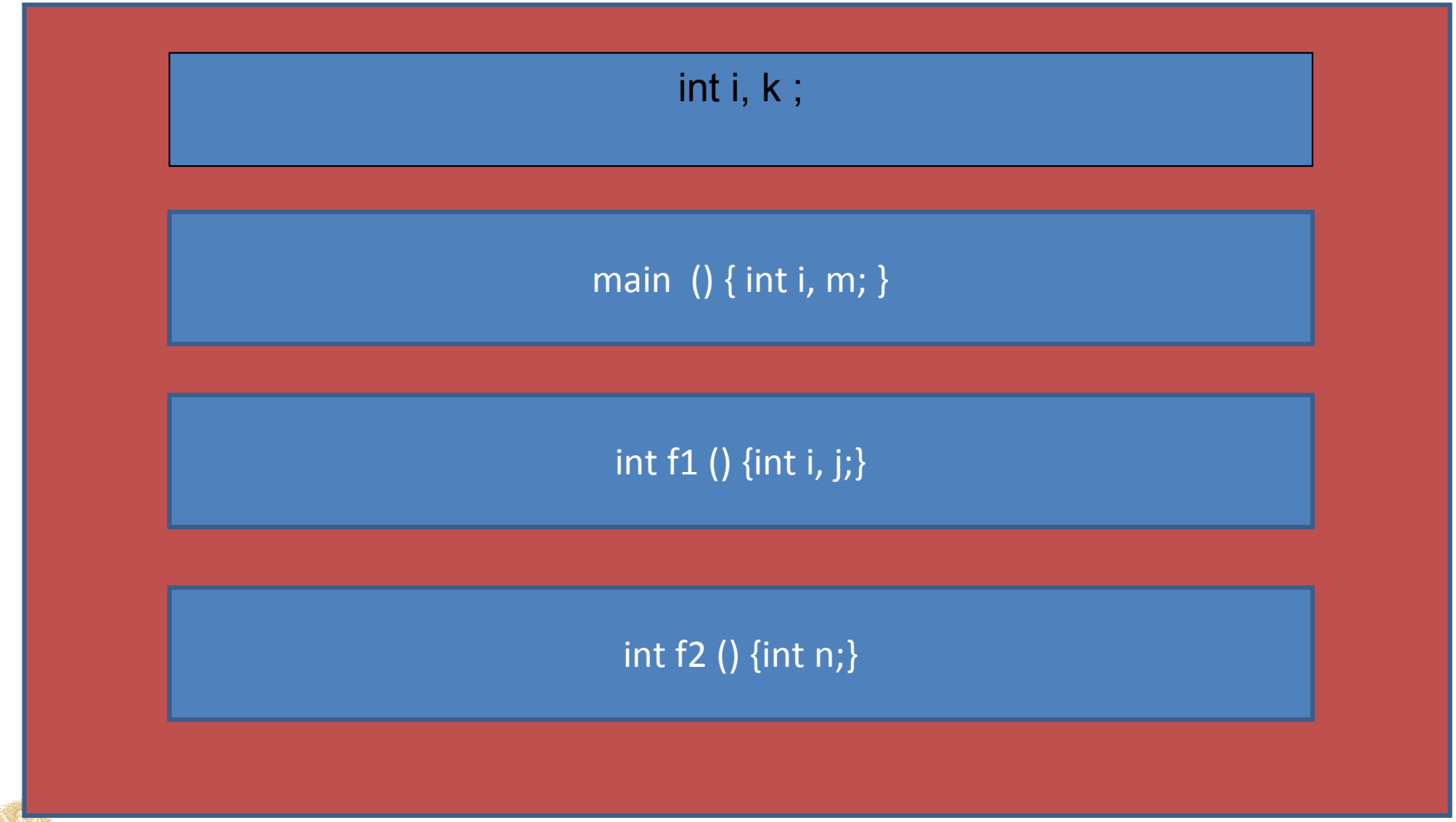
Στη sumdiff( ) ορίζονται ***νέες μεταβλητές*** a,b
Οι a,b της main() είναι διαφορετικές μεταβλητές

Διευθύνσεις των μεταβλητών που θα αλλάξουν τιμές

Η s αρχικοποιείται στην τιμή &sum

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

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

Διάρκεια ζωης μεταβλητής (2)

```
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);
}
```

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

Εμφανίζονται εξαρτήσεις (είναι δυνατόν να ...)

Λύση;;;

```
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);
}
```

Αρχικοποιώντας τις μεταβλητές

κάθε φορά που καλείται η συνάρτηση,

οι τοπικές μεταβλητές μηδενίζονται κάθε

φορά που καλείται η συνάρτηση.

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

```
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);
}
```

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

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

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);
}
```

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();
}
```

Παράδειγμα

```
#include <stdio.h>

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


int main(int argc, char *argv[]) {

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

        return 0;
}
```

```
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;
}
```

Παράδειγμα

```
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;
}
```

Σημείωμα αναφοράς

- Copyright Πανεπιστήμιο Πατρών, Παλιουράς Βασίλειος , Δερματάς Ευάγγελος «Αρχές Προγραμματισμού ». Έκδοση: 1.0. Πάτρα 2015
- Διαθέσιμο από τη δικτυακιακή διέυθυνση https://eclass.upatras.gr/modules/