ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Προγραμματισμός Η/Υ

Ενότητα 7: Συναρτήσεις

Νίκος Καρακαπιλίδης, Καθηγητής
Δημήτρης Σαραβάνος, Καθηγητής
Πολυτεχνική Σχολή
Τμήμα Μηχανολόγων & Αεροναυπηγών Μηχανικών

Σκοποί ενότητας

- Κατανόηση του ορισμού και της κλήση των συναρτήσεων
- Κατανόηση των εννοιών της εμβέλειας και της αναδρομικότητας
- Κατανόηση των κανόνων εμβέλειας ενός κώδικα και της διάρκειας των μεταβλητών

Περιεχόμενα ενότητας

- Περί συναρτήσεων (βλ. ενότητα #3)
- Η έννοια της εμβέλειας
- Διάρκεια μεταβλητών
- Μεταβίβαση παραμέτρων
- Η έννοια της αναδρομικότητας

Μέρος 1ο

Περί συναρτήσεων

Περί συναρτήσεων

Δήλωση, ορισμός και κλήση συναρτήσεων

Δήλωση (ή πρωτότυπο) συνάρτησης

Προσδιορίζει τον τρόπο αναφοράς σε αυτή (διεπαφή της συνάρτησης)

Γενική μορφή

<τύπος> <όνομα_συνάρτησης>(<τύπος_1> [<όνομα παραμέτρου_1>], ...,
<τύπος_N> [<όνομα παραμέτρου_N>]);

το όνομα του τύπου της **επιστρεφόμενης** τιμής της συνάρτησης

το όνομα του τύπου της παραμέτρου N

τυπικά ορίσματα (formal arguments)

Παραδείγματα

```
int max(int a, int b);
void draw_circle(double x, double y, double r);
```

Ορισμός συνάρτησης

Γενική μορφή

τύπος_επιστρεφόμενης_τιμής όνομα_συνάρτησης (λίστα_παραμέτρων)

{

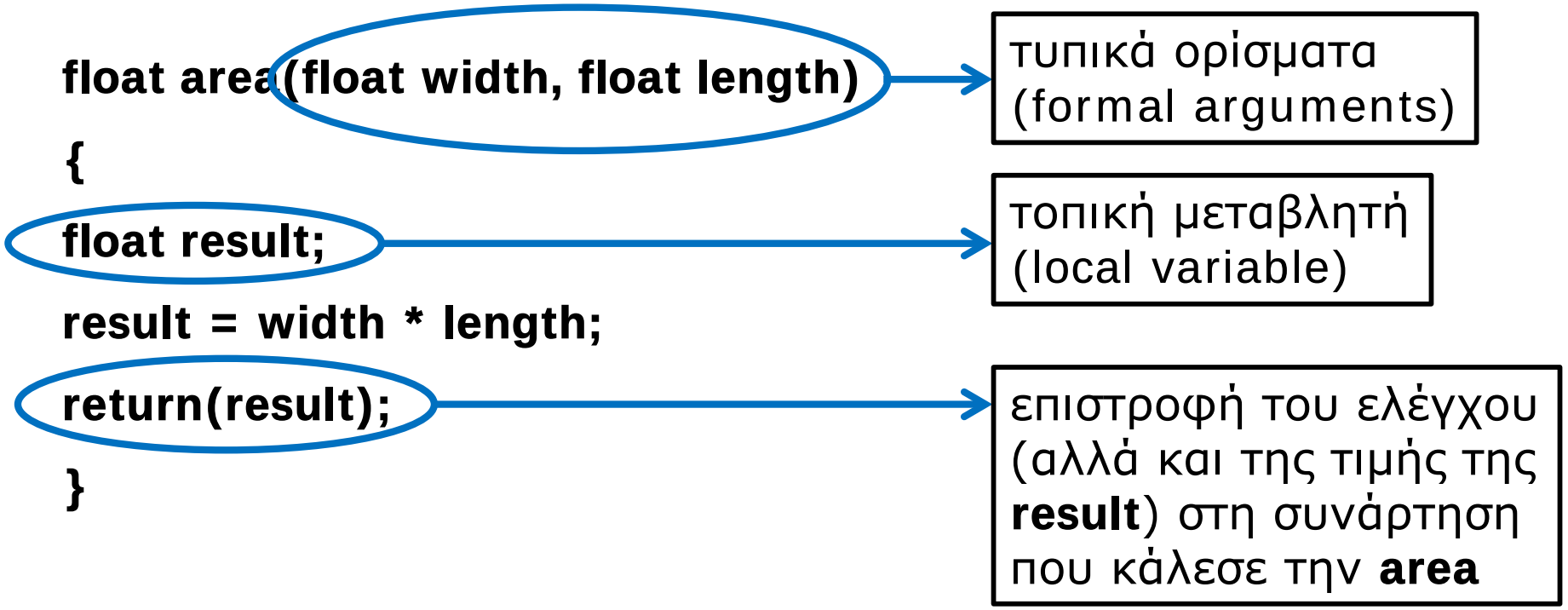
δηλώσεις τοπικών μεταβλητών

προτάσεις

}

Ορισμός συνάρτησης

Παράδειγμα



Κλήση συνάρτησης

Κλήση συνάρτησης που δεν επιστρέφει τιμή

<όνομα_συνάρτησης>(όρισμα_1, όρισμα_2, …, όρισμα_N);

πραγματικά ορίσματα (actual arguments)

Παράδειγμα

draw_circle(a/2.0, 2.0*b, c/3.0);

μπορεί να είναι σταθερές, μεταβλητές ή εκφράσεις, αλλά **του ίδιου τύπου** με τα **αντίστοιχα** τυπικά ορίσματα (ο compiler χρησιμοποιεί τη δήλωση της **draw_circle** για έλεγχο τύπων σε κάθε κλήση της)

Κλήση συνάρτησης

Κλήση συνάρτησης που επιστρέφει τιμή

Παραδείγματα

```
max_num = max(num_1, num_2);

result = num_1 + max(num_2, num_3);

printf("ο μεγαλύτερος αριθμός είναι ο %d\n", max(num_1, num_2));
```

Περί ορισμάτων

Τυπικά ορίσματα (formal arguments)

- Αναφέρονται μια φορά, στον ορισμό της συνάρτησης
- Καλούνται συχνά απλώς **παράμετροι**

Πραγματικά ορίσματα (formal arguments)

- Αναφέρονται σε κάθε ξεχωριστή κλήση της συνάρτησης
- Καλούνται συχνά απλώς **ορίσματα**

Περί ορισμάτων

- Παριστάνουν:
  - **Τιμές** (values), οι οποίες μπορεί να διατυπωθούν με οποιαδήποτε έκφραση (απ' ευθείας τιμή, μεταβλητή, κλήση συνάρτησης κλπ.)
    - Η τιμή αποτιμάται σε τιμή συμβατή με τον τύπο του αντίστοιχου τυπικού ορίσματος (παραμέτρου)
  - **Διευθύνσεις** (addresses, references) χώρων μνήμης, οι οποίες μπορεί να διατυπωθούν ως διευθύνσεις μεταβλητών, τιμές δεικτών κλπ.
    - Ο τύπος της διεύθυνσης του ορίσματος πρέπει να είναι συμβατός με τον τύπο της διεύθυνσης του αντίστοιχου τυπικού ορίσματος (παραμέτρου)

Περί ορισμάτων

Αντιστοιχία ορισμάτων με παραμέτρους

Ο αριθμός των ορισμάτων σε κάθε κλήση μιας συνάρτησης θα πρέπει να είναι ο ίδιος με τον αριθμό των παραμέτρων στον ορισμό της συνάρτησης

Η αντιστοιχία αποφασίζεται από τη σειρά των ορισμάτων.

- Το πρώτο όρισμα αντιστοιχεί στην πρώτη παράμετρο, το δεύτερο όρισμα στη δεύτερη παράμετρο κ.ο.κ.

Περί ορισμάτων

Ο τύπος κάθε ορίσματος θα πρέπει να είναι συμβατός με τον τύπο της αντίστοιχης παραμέτρου

- Το όρισμα θα πρέπει να μπορεί να «καταχωρηθεί» στην αντίστοιχη παράμετρο χωρίς (απροσδόκητη) απώλεια πληροφορίας
- Π.χ., **int** μπορεί να ανατεθεί σε **double** χωρίς καθόλου απώλεια πληροφορίας. Επίσης, **double** μπορεί να ανατεθεί σε **int** , με αναμενόμενη όμως την απώλεια των δεκαδικών ψηφίων

Περί συναρτήσεων

Σημασία κλήσης

- Κατανομή μνήμης για παραμέτρους και τοπικές μεταβλητές της συνάρτησης (εάν υπάρχουν)
- Αντιγραφή των τιμών των ορισμάτων στις παραμέτρους (εάν υπάρχουν)
- Ξεκίνημα εκτέλεσης της συνάρτησης (από την πρώτη εντολή της)

Σημασία επιστροφής

- Αποτίμηση της έκφρασης που ακολουθεί το **return** και αντιγραφή της τιμής στο σημείο κλήσης (εάν επιστρέφεται τιμή)
- Συνέχιση εκτέλεσης του προγράμματος με την εντολή που ακολουθεί την κλήση της συνάρτησης

Παράδειγμα (1)

```
#include <stdio.h>
int compute_sum(int a, int b)
{
        int sum;
        sum = a + b;
        return sum;
}
void main()
{
        int number_a, number_b;
        int sum;
        printf("Enter two numbers: ");
        scanf("%d%d",&number_a, &number_b);
        sum = compute_sum(number_a, number_b);
        printf("The sum of %d and %d is %d\n",
                number_a, number_b, sum);
}
```

Παράδειγμα (2)

```
...

#define PI 3.14159

/* Συνάρτηση για τον υπολογισμό του εμβαδού ενός κύκλου */

double area(double radius){
  return PI * radius * radius;
}
```

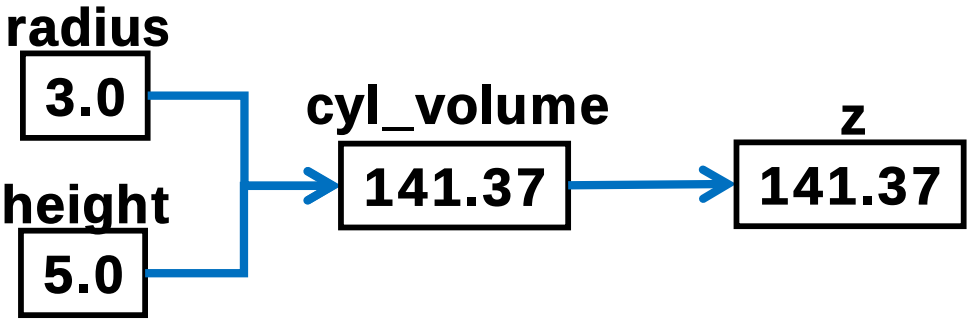
Παράδειγμα (2)

```
...

/* Συνάρτηση για τον υπολογισμό του όγκου ενός κυλίνδρου */

double cyl_volume(double radius, double height){
   double vol = PI * radius * radius * height;
   return vol;
}
```

area(radius)

Παράδειγμα (2)

```
double cyl_volume(double radius, double height)

double z = cyl_volume(3.0, 5.0);
```

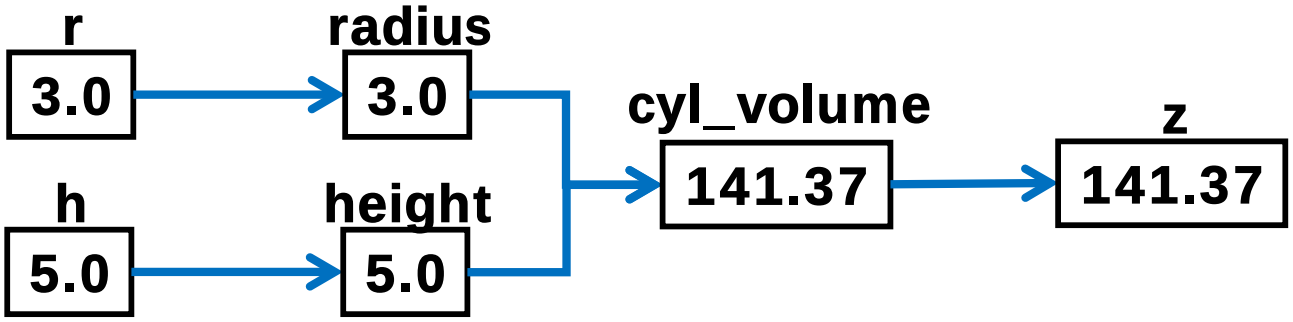


«κλήση κατ' αξία» (call by value) ή «πέρασμα δια τιμής»

Παράδειγμα (2)

```
double r=3.0, h=5.0;
double y = cyl_volume(r, h);
```

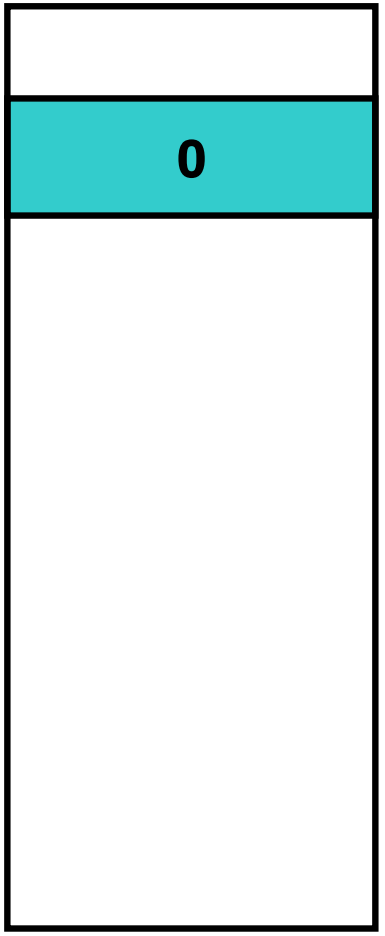
Η διεργασία της κλήσης της συνάρτησης cyl_volume επηρεάζει τις μεταβλητές h και r;



Παράδειγμα (3)

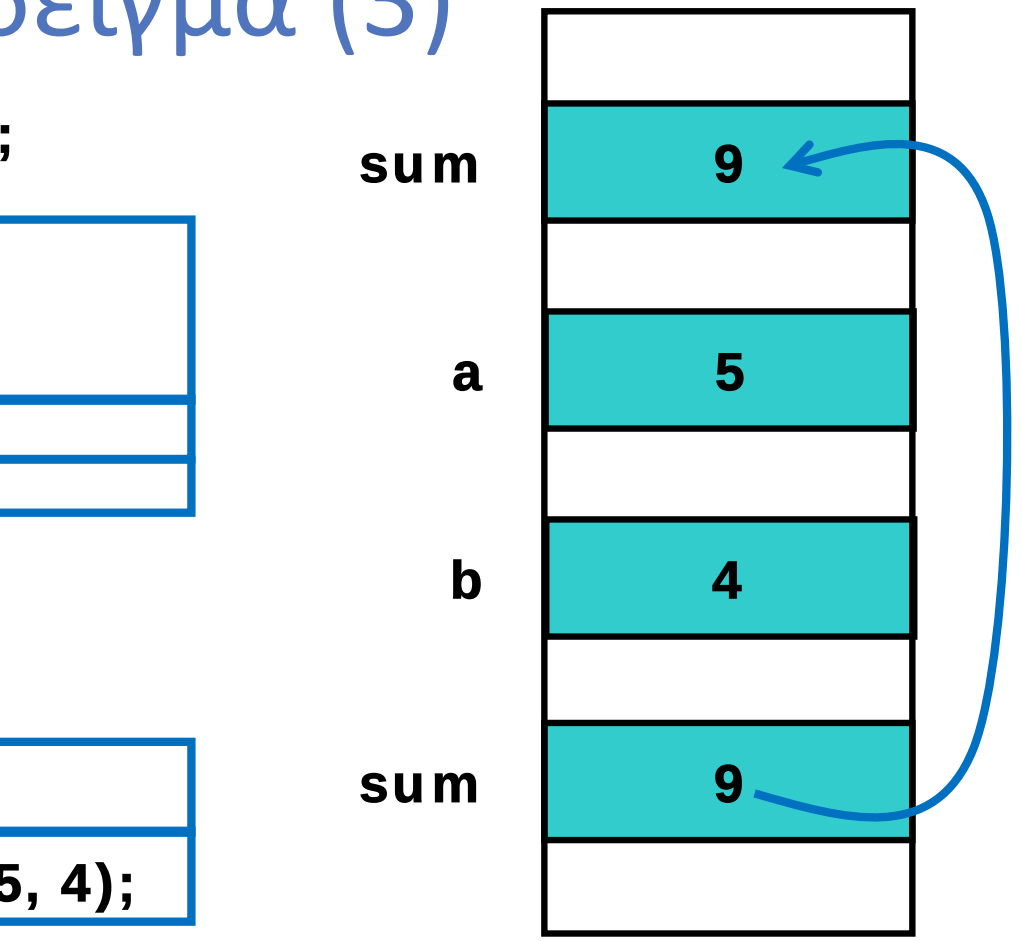
```
#include <stdio.h>
int compute_sum(int x, int y);

int compute_sum(int a, int b)
{
        int sum;
        sum = a + b;
        return sum;
}

int main()
{
        int sum=0;

        sum = compute_sum(5, 4);

        printf("The sum of %d and %d is %d\n",4, 5, sum);
        return(0);
}
```

sum



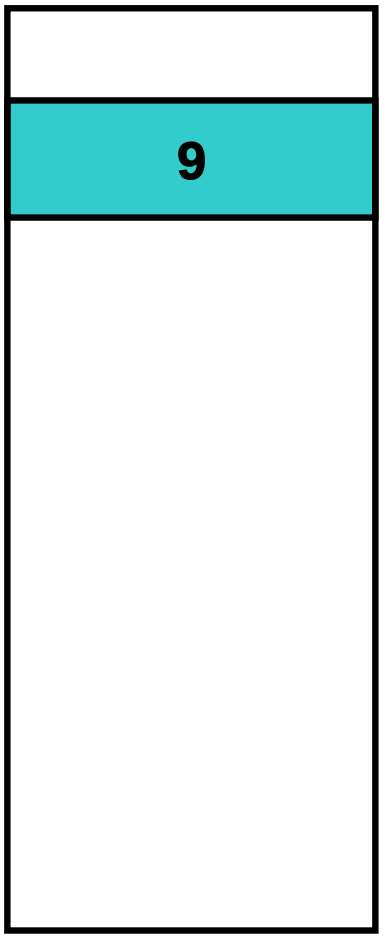
Παράδειγμα (3)

```
#include <stdio.h>
int compute_sum(int x, int y);

int compute_sum(int a, int b)
{
        int sum;
        sum = a + b;
        return sum;
}

int main()
{
        int sum=0;

        sum = compute_sum(5, 4);

        printf("The sum of %d and %d is %d\n",4, 5, sum);
        return(0);
}
```



Παράδειγμα (3)

```
#include <stdio.h>
int compute_sum(int x, int y);

int compute_sum(int a, int b)
{
        int sum;
        sum = a + b;
        return sum;
}

int main()
{
        int sum=0;

        sum = compute_sum(5, 4);

        printf("The sum of %d and %d is %d\n",4, 5, sum);
        return(0);
}
```



Μέρος 2ο

Η έννοια της εμβέλειας

Η έννοια της εμβέλειας

Εμβέλεια προγράμματος, αρχείου, συνάρτησης και μπλοκ

Η έννοια της εμβέλειας

Εμβέλεια ονομάτων

- Δηλώσεις σταθερών, μεταβλητών και συναρτήσεων
- Μπορώ να χρησιμοποιήσω το ίδιο όνομα για αναφορά σε δύο ή περισσότερες κατασκευές;
  - Αν ναι, ποιοι κανόνες διέπουν την αντιστοίχιση ενός ονόματος σε περισσότερες κατασκευές;

Η έννοια της εμβέλειας

Κανόνες εμβέλειας (scope rules): οι κανόνες που προσδιορίζουν το τμήμα του πηγαίου κώδικα στο οποίο ένα όνομα είναι «ενεργό» ή «ορατό»

- Λεκτικοί (lexical)
  - Η αντιστοίχιση γίνεται κατά τη διάρκεια της μεταγλώττισης
  - Χρησιμοποιούνται από τις περισσότερες γλώσσες προγραμματισμού
- Δυναμικοί (dynamic)
  - Η αντιστοίχιση γίνεται κατά τη διάρκεια της εκτέλεσης

Η έννοια της εμβέλειας

Τύποι εμβέλειας

- Εμβέλεια προγράμματος
  - Μεταβλητές με τέτοια εμβέλεια λέγονται γενικές ή καθολικές (global)
  - Είναι ορατές από όλες τις συναρτήσεις που απαρτίζουν το πρόγραμμα (έστω κι αν βρίσκονται σε διαφορετικά αρχεία πηγαίου κώδικα)
  - Μεταβλητή που δηλώνεται έξω από μπλοκ και χωρίς τη λέξη κλειδί **static** έχει εμβέλεια προγράμματος
- Εμβέλεια αρχείου
  - Τέτοιες μεταβλητές είναι ορατές μόνο στο αρχείο που δηλώνονται (από το σημείο δήλωσής τους και κάτω)
  - Μεταβλητή που δηλώνεται έξω από μπλοκ, με τη λέξη κλειδί **static** (πριν από τον τύπο της) έχει εμβέλεια αρχείου

Η έννοια της εμβέλειας

- Εμβέλεια συνάρτησης
  - Προσδιορίζει την ορατότητα ενός ονόματος σε όλο το σώμα της συνάρτησης
  - Τέτοια εμβέλεια έχουν μόνο οι **goto** ετικέτες
- Εμβέλεια μπλοκ
  - Προσδιορίζει την ορατότητα ενός ονόματος από το σημείο δήλωσής του μέχρι το τέλος του μπλοκ στο οποίο δηλώνεται
  - Τέτοια εμβέλεια έχουν και τα τυπικά ορίσματα των συναρτήσεων (θυμηθείτε: μπλοκ είναι η σύνθετη πρόταση αλλά και το σώμα μιας συνάρτησης)

Παράδειγμα (εμβέλεια ονομάτων)

```
#include <stdio.h>

int max(int a, int b);
static void func(int a);

int a;              /* εμβέλεια προγράμματος (γενική μεταβλητή) */
static int b;       /* εμβέλεια αρχείου */

main(){
   a=12; b=a--;
   printf("a:%d\t b:%d\t max(b+5,a):%d\n", a, b, max(b+5,a));
   func(a+b);
}

int c=13;           /* εμβέλεια προγράμματος */
```

Παράδειγμα (εμβέλεια ονομάτων)

```
int max(int a, int b){ /* εμβέλεια μπλοκ (αποκρύπτουν τις γενικές
                                          a και b για το σώμα της max) */
  return(a>b?a:b);
}

static void func(int x){          /* εμβέλεια μπλοκ */
    int b=20;                  /* εμβέλεια μπλοκ (τοπική μεταβλητή) */
    printf("a:%d\t b:%d\t c:%d\t x:%d\t max(x,b):%d\n",
      a,b,c,x,max(x,b));
```

output προγράμματος

```
a:11    b:12    max(b+5,a):17
a:11    b:20    c:13      x:23     max(x,b):23
```

Μέρος 3ο

Διάρκεια μεταβλητών

Διάρκεια μεταβλητών

Διάρκεια μεταβλητών

Ορίζει το χρόνο κατά τον οποίο το όνομα μιας μεταβλητής είναι συνδεδεμένο με τη θέση μνήμης που περιέχει την τιμή της μεταβλητής

- Χρόνος δέσμευσης και αποδέσμευσης
- Καθολικές μεταβλητές
  - Δεσμεύεται χώρος με την έναρξη της εκτέλεσης του προγράμματος
  - Η μεταβλητή συσχετίζεται με την ίδια θέση μνήμης μέχρι το τέλος του προγράμματος (πλήρους διάρκειας)

Διάρκεια μεταβλητών

- Τοπικές μεταβλητές
  - Δεσμεύεται χώρος με την είσοδο στο μπλοκ όπου είναι δηλωμένη η μεταβλητή (αποδεσμεύεται κατά την έξοδο από το μπλοκ)
  - Μια τοπική μεταβλητή δεν διατηρεί την τιμή της από τη μία κλήση της συνάρτησης στην επόμενη (περιορισμένης διάρκειας)
  - Η διάρκεια μιας τοπικής μεταβλητής μπορεί να μετατραπεί από περιορισμένη σε πλήρη χρησιμοποιώντας τη λέξη κλειδί **static** στη δήλωση της μεταβλητής

Διάρκεια μεταβλητών

```
...

static int num;

void func(int x){
  static int count=0;
  int num=100;
  ...
}
```

Διάρκεια μεταβλητών

Η **static** στη δήλωση της καθολικής μεταβλητής **num** περιορίζει την ορατότητά της μόνο στο αρχείο που δηλώνεται

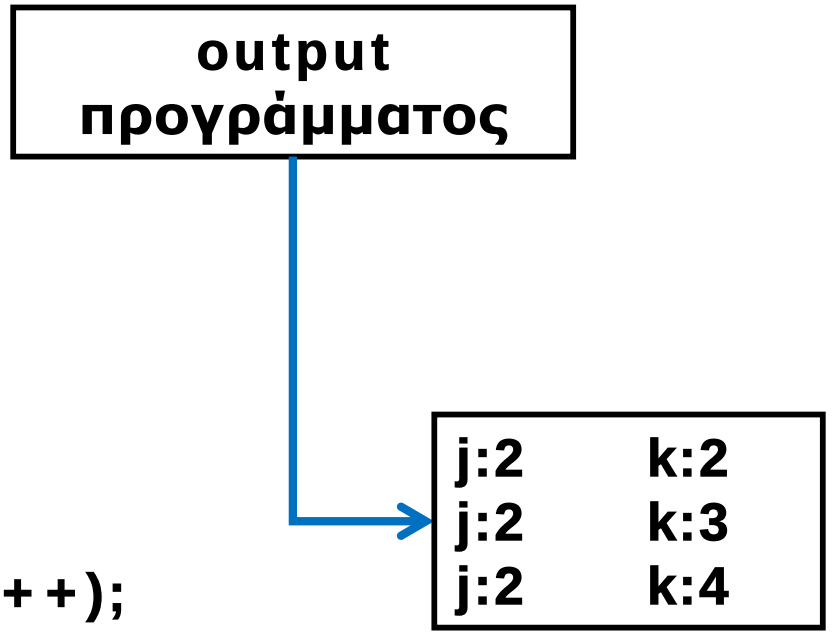
Αντίθετα, η **static** στη δήλωση της τοπικής μεταβλητής **count** ορίζει γι' αυτήν διάρκεια προγράμματος

Η **count** ως τοπική μεταβλητή πλήρους διάρκειας αρχικοποιείται μια φορά με την είσοδο στο πρόγραμμα

Αντίθετα, η **num** ως τοπική μεταβλητή περιορισμένης διάρκειας αρχικοποιείται σε κάθε ενεργοποίηση της **func**

Διάρκεια μεταβλητών

```
...

main(){
   increment();
   increment();
   increment();
}

void increment(void){
   int j=2;
   static int k=2;

   printf("j:%d\t k:%d\n", j++, k++);
}
```



Μέρος 4ο

Μεταβίβαση παραμέτρων

Μεταβίβαση παραμέτρων

Κλήση ***κατ' αξία*** και ***κατ' αναφορά***

Μεταβίβαση παραμέτρων

«Κλήση κατ' αξία» ή «πέρασμα δια τιμής» (call by value)

- Η συνάρτηση δουλεύει πάνω σε αντίγραφα των πραγματικών ορισμάτων

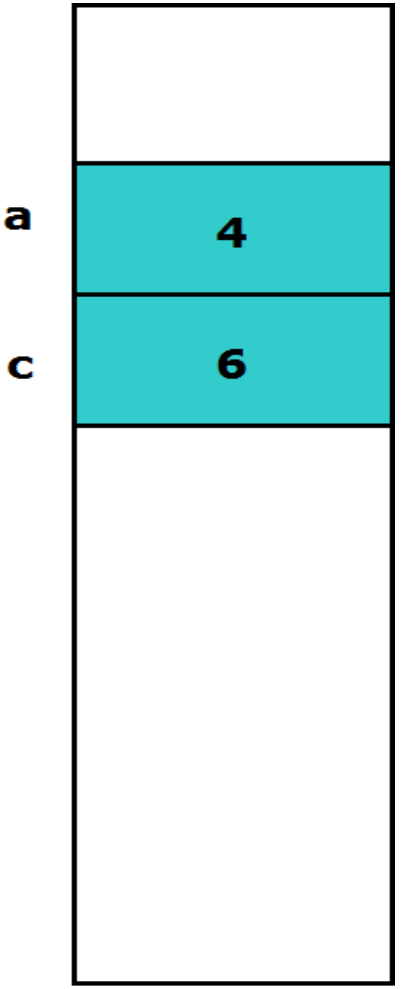
«Κλήση κατ' αναφορά» ή «πέρασμα δια αναφοράς» (call by reference)

- Η συνάρτηση δουλεύει πάνω στα πραγματικά ορίσματα

Παράδειγμα: Να γραφεί πρόγραμμα που να ανταλλάσσει τις τιμές δύο μεταβλητών **a** και **b** (βλέπε επόμενες 6 διαφάνειες)

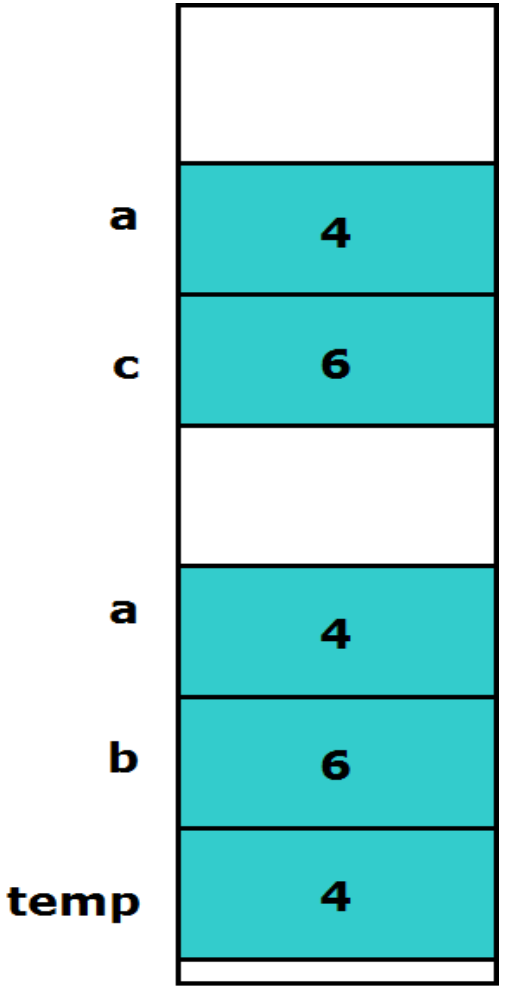
Μεταβίβαση παραμέτρων

```
#include <stdio.h>
int swap(int x, int y);

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

int main()
{
        int a=4, c=6;
        printf("a: %d, b: %d\n",a,c);
        swap(a, c);
        printf("a: %d, b: %d\n",a,c);

        return(0);
}
```



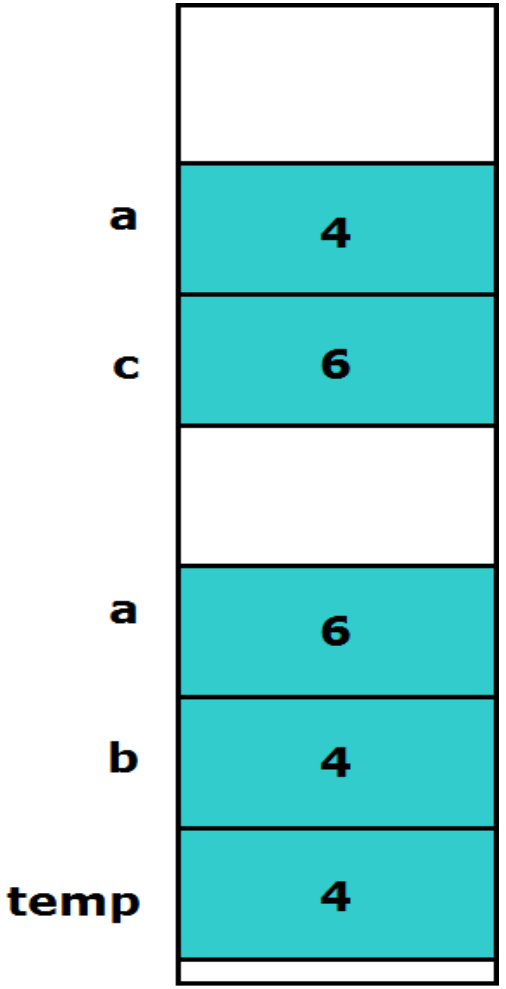
Μεταβίβαση παραμέτρων

```
#include <stdio.h>
int swap(int x, int y);

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

int main()
{
        int a=4, c=6;
        printf("a: %d, b: %d\n",a,c);
        swap(a, c);
        printf("a: %d, b: %d\n",a,c);

        return(0);
}
```



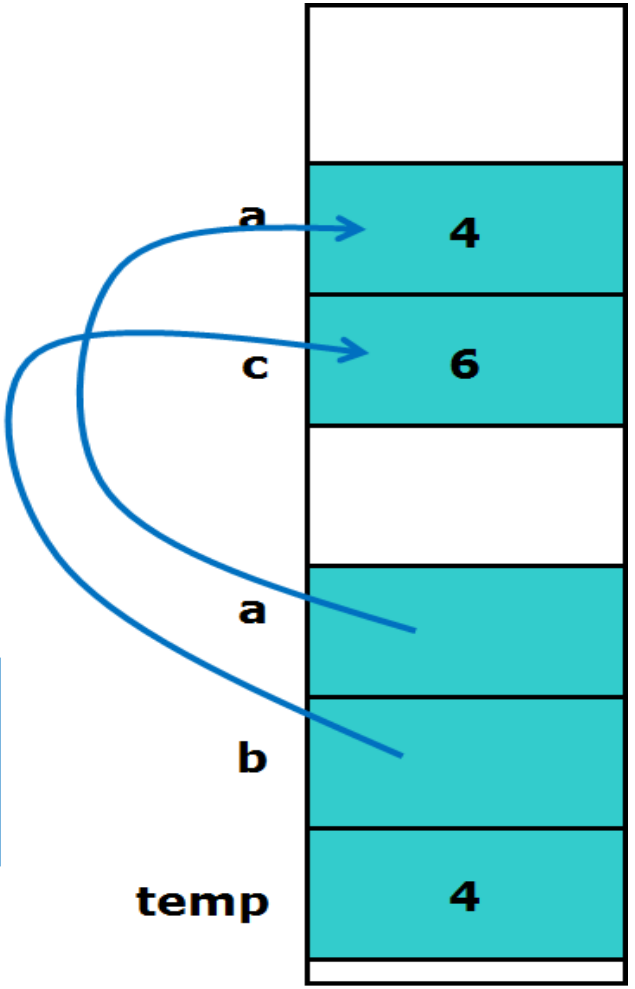
Μεταβίβαση παραμέτρων

```
#include <stdio.h>
int swap(int x, int y);

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

int main()
{
        int a=4, c=6;
        printf("a: %d, b: %d\n",a,c);
        swap(a, c);
        printf("a: %d, b: %d\n",a,c);

        return(0);
}
```



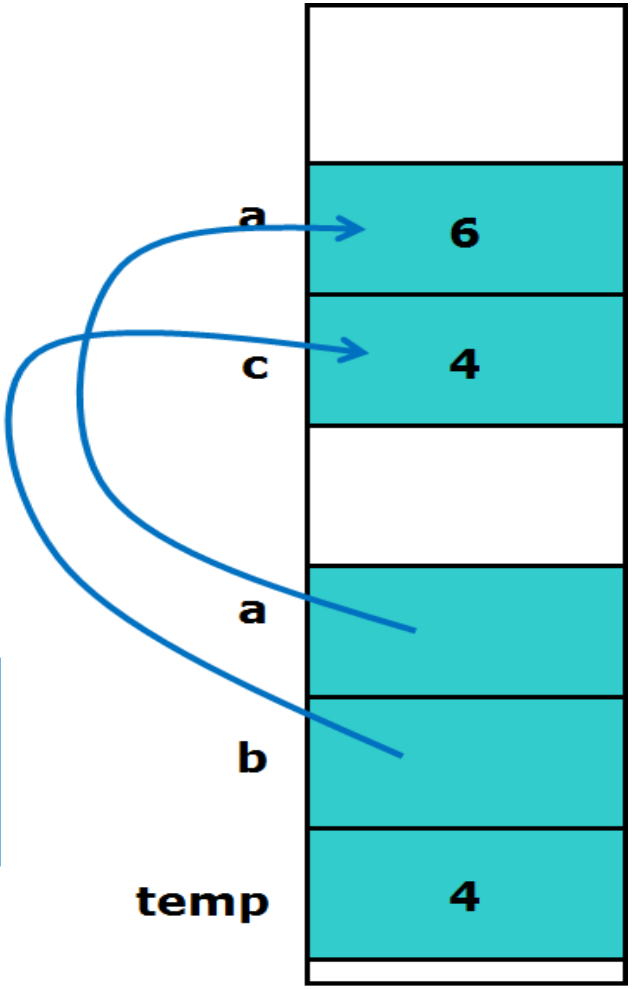
Μεταβίβαση παραμέτρων

```
#include <stdio.h>
int swap(int x, int y);

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

int main()
{
        int a=4, c=6;
        printf("a: %d, b: %d\n",a,c);
        swap(a, c);
        printf("a: %d, b: %d\n",a,c);

        return(0);
}
```



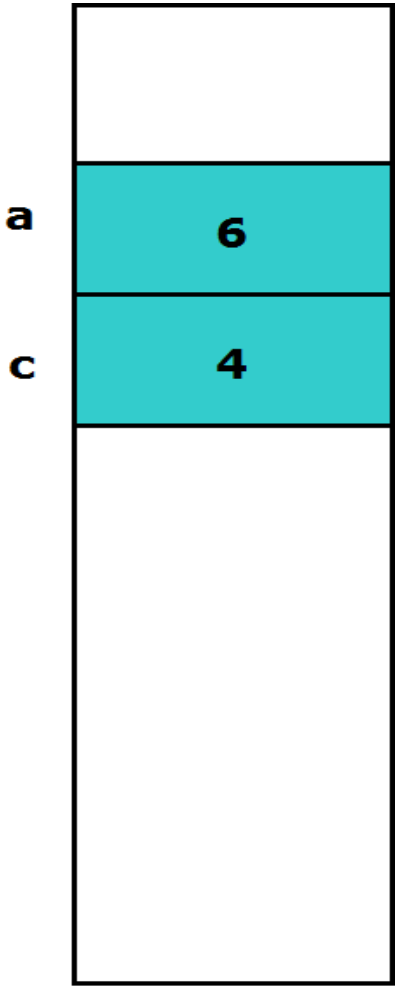
Μεταβίβαση παραμέτρων

```
#include <stdio.h>
int swap(int x, int y);

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

int main()
{
        int a=4, c=6;
        printf("a: %d, b: %d\n",a,c);
        swap(a, c);
        printf("a: %d, b: %d\n",a,c);

        return(0);
}
```



Μεταβίβαση παραμέτρων

```
#include <stdio.h>
int swap(int x, int y);

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

int main()
{
        int a=4, c=6;
        printf("a: %d, b: %d\n",a,c);
        swap(a, c);
        printf("a: %d, b: %d\n",a,c);

        return(0);
}
```



Μέρος 5ο

Η έννοια της αναδρομικότητας

Η έννοια της αναδρομικότητας

Η έννοια της αναδρομικότητας

Μια συνάρτηση είναι αναδρομική εάν ο ορισμός της περιέχει κλήση στον εαυτό της

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί με επανάληψη

Η έννοια της αναδρομικότητας

Αναδρομική λύση ενός προβλήματος είναι συχνά ο «φυσικός» τρόπος επίλυσής του

- Διαμόρφωση προβλήματος για αναδρομική λύση
- Παράδειγμα (1): Ακολουθία Fibonacci
  - Fib(n) = Fib(n – 2) + Fib(n – 1) και Fib(0) = 0, Fib(1) = 1
- Πρέπει να υπάρχει τουλάχιστον μια περίπτωση που τερματίζει την αναδρομή
  - Αλλιώς, αναδρομή επ’ άπειρο ή δέσμευση ολόκληρης μνήμης
- Η αναδρομική κλήση πρέπει να είναι ένα βήμα πιο κοντά στην περίπτωση τερματισμού από την κλήση που οδήγησε σε αυτήν

Η έννοια της αναδρομικότητας

Παράδειγμα: Υπολογισμός n!

- n! = n * n-1 * n-2 * … * 1 και 1!=1, 0!=1
- n! = n * n-1!

```
float factorial(int n)
{
    int i ;
    int fact = 1;
    for(i=n;i>0;--i)
          fact = fact *i;

    return fact;
}
```

```
float factorial(int n)
{
    if (n<2)
          return 1;
    return n*factorial(n-1);
}
```

Τέλος Ενότητας

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου των διδασκόντων.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων