Εισαγωγή στον δομημένο προγραμματισμό

Ενότητα 9η: Συναρτήσεις

Αν. καθηγητής Στεργίου Κώστας
e-mail: kstergiou@uowm.gr

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

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

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη 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.

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

Στόχοι της διάλεξης

- Περιγραφή των εννοιών της εμβέλειας και διάρκειας μεταβλητής.
  - Κατανόηση της ορθής χρήσης καθολικών και στατικών μεταβλητών.
- Περιγραφή του τρόπου κατανομής ενός προγράμματος σε πολλά αρχεία.
- Εισαγωγή στην έννοια της αναδρομής.
  - Περιγραφή απλών αναδρομικών συναρτήσεων.

Περιεχόμενα

- Εμβέλεια μεταβλητής:
  - Τοπική.
  - Καθολική.
  - Στατική.
- Κατανομή προγράμματος σε πολλά αρχεία.
- Διάρκεια μεταβλητής.
- Αναδρομή:
  - Αναδρομικές συναρτήσεις.
  - Παραδείγματα.

Εμβέλεια Μεταβλητής (1/2)

- Οι μεταβλητές που έχουμε δει μέχρι τώρα είναι γνωστές στο μπλοκ του κώδικα όπου δηλώνονται (**τοπικές μεταβλητές**).

```
#include <stdio.h>
int square_100 ()              /* Η y "δεν υπάρχει" εδώ*/
{
   int x;
   x = 100*100;
    return x;
}
int cube_100 ()                /* Η x "δεν υπάρχει" εδώ*/
{
   int y;
   y = 100*100*100;
    return y;
}
int main(void)                 /* Οι x, y "δεν υπάρχουν" εδώ*/
{
  printf("100 squared is %d and 100 cube is %d\n",square_100(),cube_100());
  return 0;
}
```

- Η μεταβλητή έχει εμβέλεια μέχρι να κλείσει το μπλοκ ({ }).

Εμβέλεια Μεταβλητής (2/2)

- Το τμήμα του πηγαίου κώδικα στο οποίο έχει ισχύ ή είναι ορατή η μεταβλητή.
- Μπορούμε να ορίσουμε για δύο μεταβλητές το ίδιο όνομα εφόσον έχουν διαφορετική εμβέλεια.
  - π.χ. Μπορεί να υπάρχουν 2 μεταβλητές με το όνομα *i* που να είναι εντελώς ανεξάρτητες.
- Αν οι περιοχές εμβέλειας έχουν επικάλυψη, το όνομα με τη μικρότερη εμβέλεια αποκρύπτει το όνομα με τη μεγαλύτερη.

Παράμετροι Συναρτήσεων

- Οι παράμετροι μιας συνάρτησης θεωρούνται τοπικές της μεταβλητές και συνεπώς είναι γνωστές μόνο στο μπλοκ της συνάρτησης.

```
#include <stdio.h>
int max (int x, int y)            /* Οι a και b "δεν υπάρχουν" εδώ*/
{
   return (x>y ? x : y);
}
int min (int a, int b)           /* Οι x και y "δεν υπάρχουν" εδώ*/
{
   return (a<b ? a : b);
}
   int main(void)               /* Οι x, y, a, b "δεν υπάρχουν" εδώ*/
{
  int num1, num2;
  printf("dwse duo arithmous"\n);
  scanf("%d %d",&num1,&num2);
  printf("max is %d and min is %d\n",max(num1,num2), max(num1,num2));
  return 0;
}
```

Εμβέλεια Μεταβλητής

- Τοπικές μεταβλητές με το ίδιο όνομα είναι διαφορετικές μεταβλητές.

```
#include <stdio.h>
int square_100 ()          /* Η x είναι διαφορετική από την x του cube_100*/
{
   int x;
   x = 100*100;
   return x;
}
int cube_100 ()            /* Η x είναι διαφορετική από την x του square_100
   */
{
   int x;
   x = 100*100*100;
   return x;
}
int main(void)             /* Καμία x "δεν υπάρχει" εδώ*/
{
  printf("100 squared is %d and 100 cube is %d\n",square_100(),cube_100());
  return 0;
}
```

Εμβέλεια Αρχείου/Προγράμματος

- Μπορούμε να δηλώσουμε μεταβλητές έξω από όλες τις συναρτήσεις. Αυτές έχουν **εμβέλεια αρχείου ή εμβέλεια προγράμματος** και ονομάζονται **καθολικές ή εξωτερικές μεταβλητές.**

```
#include <stdio.h>
int square_100 ()              /* Η y "δεν υπάρχει" εδώ*/
{
    return 100*100;
}
int y;
int cube_100 ()                /* Η y "υπάρχει" εδώ*/
{
   y = 100*100*100;
    return y;
}
int main(void)                 /* Η y "υπάρχει" εδώ*/
{
  printf("100 squared is %d and 100 cube is %d\n",square_100(),y);
  return 0;
}
```

- Μια μεταβλητή με εμβέλεια αρχείου είναι γνωστή από το σημείο δήλωσης της και κάτω.

Διαίρεση προγράμματος σε πολλά αρχεία

- Μεγάλα προγράμματα συνήθως μοιράζονται σε πολλά αρχεία.
  - Ένας τρόπος επικοινωνίας συναρτήσεων που βρίσκονται σε διαφορετικά αρχεία είναι μέσω καθολικών μεταβλητών.
- Μια μεταβλητή έχει **εμβέλεια αρχείου** όταν είναι ορατή μόνο στο αρχείο στο οποίο ορίζεται.
- Μια μεταβλητή έχει **εμβέλεια προγράμματος** όταν είναι ορατή σε όλα τα αρχεία του προγράμματος.
- Λέξεις κλειδιά `static` και `extern`.

Καθολικές και τοπικές μεταβλητές

- Αν δηλώσουμε μια τοπική και μια καθολική μεταβλητή με το ίδιο όνομα, τότε η τοπική μεταβλητή επικαλύπτει την καθολική.

```
#include <stdio.h>
int x = 1;
int function1 ()          /* Η x είναι τοπική εδώ*/
{
   int x;
   x++;
}
int function2 ()          /* Η x είναι καθολική εδώ */
{
   x--;
}

int main(void)            /* Η x είναι καθολική εδώ */
{
  function1();
  function2();
  return 0;
}
```

Τι τιμή έχει η X πριν το return της main ;

Εμβέλεια Προγράμματος

- **Γενικές, Εξωτερικές** ή **Καθολικές μεταβλητές**.

- Μπορεί να είναι ορατές από όλες τις συναρτήσεις που απαρτίζουν τον πηγαίο κώδικα έστω και αν βρίσκονται σε διαφορετικά αρχεία.

- Η εμβέλεια μιας καθολικής μεταβλητής καθορίζεται:
  - Από τη θέση της δήλωσής της.
  - Από την ύπαρξη ή μη της λέξης-κλειδί **`static`** (όταν δεν δηλώνεται **`static`** έχει εμβέλεια προγράμματος).

Εμβέλεια Αρχείου

- Μεταβλητή με εμβέλεια αρχείου είναι ορατή μόνο στο αρχείο στο οποίο δηλώνεται.
- Είναι ορατή από το σημείο που δηλώνεται και κάτω.
- Δηλώνονται έξω από μπλοκ εντολών με τη λέξη κλειδί **`static`**.
  - `static int velocity;`

Εμβέλεια Μπλοκ

- Μεταβλητές που δηλώνονται μέσα σε ένα μπλοκ εντολών {...}.
- Μία μεταβλητή με εμβέλεια μπλοκ είναι ορατή από το σημείο δήλωσής της μέχρι το τέλος του μπλοκ.
- Μπλοκ μπορεί να είναι μία σύνθετη πρόταση αλλά και το σώμα μιας συνάρτησης.
- Εμβέλεια μπλοκ έχουν και οι παράμετροι των συναρτήσεων.

```
#include <stdio.h>
void print_stars (int n, int m)

{
   int i;
   for (i=0; i<n; i++) {
      int j;
      for (j=0; j<m; j++)
         printf("%c ",'*');
   }
}
…
```

- Καλό είναι να αποφεύγετε τέτοιες δηλώσεις μεταβλητών.
- Καλύτερα όλες οι δηλώσεις να γίνονται στην αρχή.

Εμβέλεια Συναρτήσεων

- Όπως και οι μεταβλητές, οι συναρτήσεις έχουν τη δική τους εμβέλεια.
- Η εμβέλεια μιας συνάρτησης εκτείνεται από τη δήλωσή της μέχρι το τέλος του προγράμματος.
- Αν μία συνάρτηση δηλωθεί ως **`static`** τότε η εμβέλειά της περιορίζεται στο αρχείο που δηλώθηκε.

Παράδειγμα 1ο

```
int max(int a, int b);
static void func(int a);
int a;
static int b;
main() {
   a=12; b=a--;
   printf("%d\t%d\t%d",a,b,max(a,b));
   func(a+b);
}
int c=13;
int max(int a, int b) {
   return (a>b?a:b);
}
static void func(int x) {
   int b=20;
   printf("%d\t%d\t%d",a,b,max(x,b));
}
```

a: καθολική μεταβλητή, εμβέλεια προγράμματος.

b: εμβέλεια αρχείου.

c: εμβέλεια προγράμματος, μη ορατή από την main.

a,b: εμβέλεια μπλοκ, αποκρύπτουν τις εξωτερικές a,b.

x, b: εμβέλεια μπλοκ, η b αποκρύπτει την εξωτερική b.

Παράδειγμα 2ο

```
int max(int a, int b);
static void func(int a);
int a;
static int b;
main() {
   a=12; b=a--;
   printf("%d\t%d\t%d",a,b,max(a,b));    /* 11, 12, 12 */
   func(a+b);
}
int c=13;
int max(int a, int b) {
   return (a>b?a:b);
}
static void func(int x) {
   int b=20;
   printf("%d\t%d\t%d",a,b,max(x,b));    /* 12 20 23 */
}
```

Καθολικές Μεταβλητές

- Προσπαθήστε να αποφεύγετε τη χρήση καθολικών μεταβλητών.
- Αυξάνουν την πολυπλοκότητα του προγράμματος και καθιστούν δύσκολη τη συντήρησή του.
- Έχουν όμως το πλεονέκτημα ότι δημιουργούν ταχύτερο κώδικα.
- Επιδιώξτε την χρήση των δηλώσεων `static` μεταβλητών.
  - Βελτιώνει την αναγνωσιμότητα και διευκολύνει την τροποποίηση του κώδικα.
  - Διασφαλίζει από την λανθασμένη αναφορά στη μεταβλητή από άλλα αρχεία του προγράμματος.

Διάρκεια Μεταβλητών (1/2)

- Ορίζει το χρόνο που το όνομα της μεταβλητής είναι συνδεδεμένο με τη θέση μνήμης.
- Χρόνος δέσμευσης: όταν συνδέεται η θέση μνήμης με το όνομα.
- Χρόνος αποδέσμευσης: όταν αποσυνδέεται η θέση μνήμης με το όνομα.

Διάρκεια Μεταβλητών (2/2)

- Καθολικές μεταβλητές → πλήρους διάρκειας.
  - Δεσμεύεται χώρος μνήμης με την έναρξη του προγράμματος και διατηρείται μέχρι το τέλος.
- Τοπικές μεταβλητές → περιορισμένης διάρκειας.
  - Χρόνος δέσμευσης είναι η είσοδος στο μπλοκ που δηλώνονται.
  - Χρόνος αποδέσμευσης είναι με το τέλος του μπλοκ.

Τοπικές Μεταβλητές Πλήρους Διάρκειας

- Οι τοπικές μεταβλητές δεν διατηρούν την τιμή τους από τη μία κλήση μιας συνάρτησης στην επόμενη κλήση.
- Σε περίπτωση που θέλουμε να τις διατηρούν πρέπει να τις δηλώσουμε ως static.

```
void func(int x)
{
int temp;  /* Δεν διατηρεί την τιμή της */
static int num;/* Διατηρεί την τιμή της */
…
}
```

Παράδειγμα 3ο

```
static int num;              /* Εμβέλεια εντός αρχείου */
void func(int a)
{
static int count=0;          /* Διάρκεια προγράμματος */
int num=100;
…
}
```

- Η num αρχικοποιείται κάθε φορά που καλείται η func.
- Η count αρχικοποιείται μόνο τη πρώτη φορά.

Παράδειγμα 4ο

```
void increment(void);
main()
{
   increment();
   increment();
   increment();
}

void increment(void)
{
   int j=2;
   static int k=2;
   printf("j:%d\tk:%d\n",j++,k++);
}
```

```
j:2    k:2
j:2    k:3
j:2    k:4

_
```

Οργάνωση Προγράμματος σε Πολλά Αρχεία Πηγαίου Κώδικα

- Σε πολύπλοκα προγράμματα οργανώνουμε το πηγαίο κώδικα σε διαφορετικά αρχεία.
- Το κάθε αρχείο πρέπει να έχει τις κατάλληλες συναρτήσεις και μεταβλητές, η ορατότητα των οποίων καθορίζεται από τη λέξη-κλειδί `static`.
- Με τη λέξη-κλειδί `extern` χρησιμοποιούμε μία καθολική μεταβλητή που ορίζεται σε άλλο αρχείο.

Παράδειγμα 5ο

```c
/* file1.c */
#include <stdio.h>
void func2(void);
static int func1(int a);
int x=1;
static int y=13;

void main()
{
…
}

static int func1()
{
…
}
```

```c
/* file2.c */
#include <stdio.h>
static void func3(void);
extern int x;
static int z=1;

void func2(void)
{
…
}

static void func3(void)
{
…
}
```

Οργάνωση Προγράμματος σε Πολλά Αρχεία Πηγαίου Κώδικα

- Συχνά οι καθολικές μεταβλητές, οι σταθερές, και οι συναρτήσεις που χρησιμοποιούνται σε πολλά αρχεία δηλώνονται σε ένα αρχείο επικεφαλίδας.

```
/* file1.h */

void func1(void);
int func2(int a);
int x=1;
int y=13;
```

```
/* file2.c */
#include <stdio.h>
#include "file1.h"

void main()
{
…
}

void func1()
{
…
}
int func2(int a)
{
…
}
```

ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΛΟΓΙΣΜΙΚΟΥ

Αναδρομικές Συναρτήσεις

Αναδρομικότητα

- Μία αναδρομική συνάρτηση καλεί τον εαυτό της, κάθε φορά με διαφορετικό όρισμα.
- Η κλήσεις τερματίζονται εφόσον ισχύει μία τερματική συνθήκη.
- Η αναδρομικές συναρτήσεις:
  - Συνήθως παρέχουν πιο συμπαγή κώδικα.
  - Είναι ιδιαίτερα χρήσιμες για αναδρομικώς ορισμένα δεδομένα (λίστες, δέντρα).
  - Δεν παρέχουν ταχύτερο κώδικα.
  - Η κακή χρήση τους μπορεί να οδηγήσει σε stack overflow.

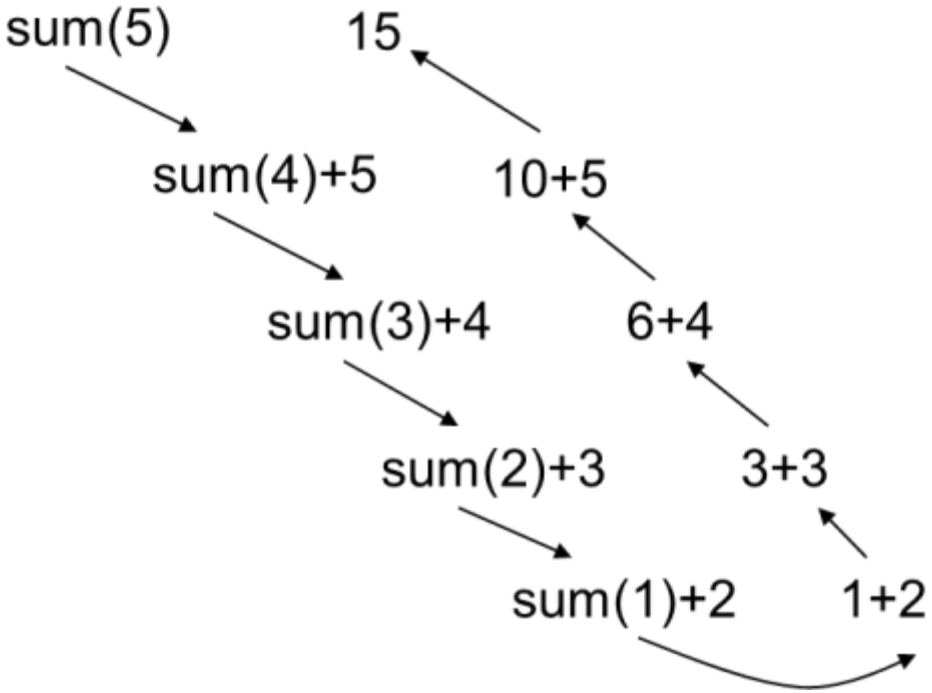
Παράδειγμα: Υπολογισμός Αθροίσματος 1+2+...+n

```
/* Μη-Αναδρομική */
int sum1(int n)
{
   int i, sum=0;
   for (i=1; i=<n; i++)
        sum +=i;
   return sum;
}
```

```
/* Αναδρομική */
int sum2(int n)
{
/* Συνθήκη τερματισμού */
      if (n == 1)
            return 1;
      else
            return
        (sum2(n-1)+n);
}
```

Παράδειγμα 6ο

- Αναδρομικές κλήσεις της sum για n=5.



Παράδειγμα: Υπολογισμός x^n

```
float power(float x, int n)
{
  if (n==0)              /* Τερματική συνθήκη */
     return(1.0);
  else if (n>0)
     return(x * power(x,n-1));
  else
     return(1.0 / power(x,-n));
}
```

Παράδειγμα: Υπολογισμός Αριθμών Fibonacci

```
int fibonacci(int n)
{
if (n <= 2)
  return(1);
else
  return(fibonacci(n-1)+fibonacci(n-2));
}
```

Άσκηση

```
int func(int a, int b)
{  int i;
   i = a;
   if (i < b)
       {      i += func(2*i,b);
              printf("%d ",i);
              return(i);
       }
   return(0);
   printf("%d\n",a);
}
```

- Τι θα εκτυπωθεί από την κλήση της συνάρτησης `func(1,30);`
- Τι θα εκτυπωθεί από την κλήση της συνάρτησης `func(30,1);`

16 24 28 30 31

Γρήγορη Ταξινόμηση Πίνακα (quicksort)

- Επιλέγεται ένα στοιχείο του πίνακα.
- Τοποθετείται στη σωστή θέση του:
  - Όλα τα στοιχεία δεξιά του είναι μικρότερά του.
  - Όλα τα στοιχεία αριστερά του είναι μεγαλύτερά του.
- Ταξινομείται με την ίδια διαδικασία ο δεξιός υποπίνακας.
- Ταξινομείται με την ίδια διαδικασία ο αριστερός υποπίνακας.

qsort1: Ταξινόμηση των a[left]…a[right] σε αύξουσα σειρά

```
void swap(int a[], int i, int j);

void qsort1(int a[], int left, int right)
{
   int i, last;
   if (left >= right)
       return;
   last=left;       /* Επιλογή του πρώτου στοιχείου */
   for (i=left+1; i <= right; i++)
       if (a[i] < a[left])
              swap(a, ++last, i);
   swap(a, left, last);
   qsort1(a, left, last-1);
   qsort1(a, last+1, right);
}
```

qsort2: Ταξινόμηση των a[left]…a[right] σε αύξουσα σειρά

```
void qsort2(int a[], int left, int right)
{
   int l, r, pivot;
   if (left<right)
   {   l=left;
       r=right;
       pivot=a[left];
       while (l<r) {
             while (a[r]>pivot) r--;
             while (a[l]<=pivot && l<r) l++;
             if (l<r) swap(a, l, r); }
       swap(a, left, r);
       qsort2(a, left, r-1);
       qsort2(a, r+1, right);
   }
}
```

Τέλος Ενότητας

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.