Πανεπιστήμιο Αιγαίου

Μεθοδολογίες και Γλώσσες Προγραμματισμού Ι

Συναρτήσεις ΙΙ

Εργίνα Καβαλλιεράτου (kavallieratou@aegean.gr)

Μόνιμη Επίκουρος Καθηγήτρια

Τμήμα Μηχανικών Πληροφοριακών & Επικοινωνιακών Συστημάτων

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 Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» 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.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημερινό μάθημα

- ✓ Εμβέλεια
- ✓ Εμφωλίαση
- ✓ Τύπος αποθήκευσης
- ✓ Συναρτήσεις ως παράμετροι
- ✓ Πέρασμα με τιμή
- ✓ Πολλαπλά return
- ✓ Προκαθορισμένοι Παράμετροι
- ✓ Υπερφόρτωση συναρτήσεων
- ✓ Inline συναρτήσεις
- ✓ Αναδρομή

Εμβέλεια

- ✓ Εμβέλεια μιας μεταβλητής είναι το τμήμα του προγράμματος που η μεταβλητή μπορεί να χρησιμοποιηθεί (που υπάρχει)
- ✓ Μια σφαιρική μεταβλητή έχει απεριόριστη εμβέλεια
- ✓ Η εμβέλεια μιας τοπικής μεταβλητής περιορίζεται μέσα στη συνάρτηση που δηλώνεται
- ✓ Η εμβέλεια μιας block μεταβλητής περιορίζεται μέσα στο block που δηλώνεται

Παράδειγμα

```
int main(void) {
  int y=10;

  {
     int a = y;
     cout << a << endl;
  }
  cout << a << endl;
}
```

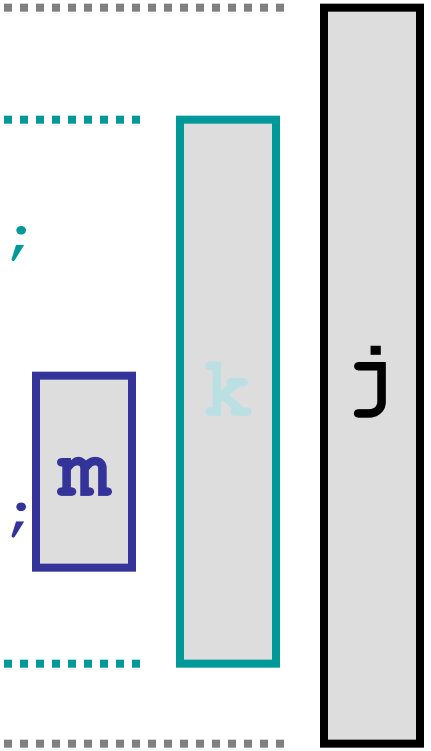
Λάθος!!!

Εμφωλίαση

- ✓ Στη C++ δεν υπάρχει εμφωλίαση συναρτήσεων αλλά υπάρχει εμφωλίαση blocks.

Εμφωλιασμένα Blocks

```
void foo(void) {
  for (int j=0;j<10;j++) {
      int k = j*10;
      cout << j << ",” << k << endl;
      {
      int m = j+k;
      cout << m << ",” << j << endl;
      }
  }
}
```



Τύπος αποθήκευσης

- ✓ Κάθε μεταβλητή έχει τύπο αποθήκευσης
- ✓ Ορίζει την περίοδο κατά την οποία μια μεταβλητή «υπάρχει» στη μνήμη
- ✓ Κάποιες μεταβλητές δημιουργούνται μια φορά και διατηρούνται στη μνήμη π.χ. Global
- ✓ Άλλες δημιουργούνται πολλές φορές π.χ local σε συνάρτηση

Τύπος αποθήκευσης

- **`auto`** – δημιουργούνται κάθε φορά που μπαίνουμε στο block που υπάρχουν
- **`register`** – το ίδιο με τις **`auto`**, αλλά υπαγορεύουν στον compiler να είναι πιο γρήγορος (πως;).
- **`static`** – δημιουργείται μόνο μια φορά
- **`extern`** – global μεταβλητή δηλωμένη αλλού

Τύποι αποθήκευσης

```
auto int j;

register int i_need_to_be_fast;

static char remember_me;

extern double a_global;
```

Πρακτική χρήση

- ✓ Οι Local μεταβλητές είναι **`auto`** εξ ορισμού
- ✓ Οι Global μεταβλητές είναι **`static`** εξ ορισμού
- ✓ Δηλώνοντας μια local μεταβλητή ως **`static`** σημαίνει ότι θα θυμάται την τελευταία τιμή της

`static` example

```
int countcalls(void) {
  static int count = 0;
  count++;
  return(count);
}
…
cout << countcalls() << endl;
cout << countcalls() << endl;
cout << countcalls() << endl;
```

Συναρτήσεις ως παράμετροι

Answer = (double(triple(square(cube(myValue)))));

Πέρασμα με τιμή-παράδειγμα

```
#include <iostream.h>
    void swap(int x, int y);
    int main()  {
      int x = 5, y = 10;
      cout << "Main. Before swap, x: " << x << " y: " << y << "\n";
      swap(x,y);
      cout << "Main. After swap, x: " << x << " y: " << y << "\n";
        return 0;   }
    void swap (int x, int y)    {
      int temp;
      cout << "Swap. Before swap, x: " << x << " y: " << y << "\n";
      temp = x;
      x = y;
      y = temp;
      cout << "Swap. After swap, x: " << x << " y: " << y << "\n"; }
```

Πέρασμα με τιμή – έξοδος

Main. Before swap, x: 5 y: 10

Swap. Before swap, x: 5 y: 10

Swap. After swap, x: 10 y: 5

Main. After swap, x: 5 y: 10

Πολλαπλά return/1

```
#include <iostream.h>
int Doubler(int AmountToDouble);
int main()
{
   int result = 0;
   int input;
   cout << "Enter a number between 0 and 10,000 to double: ";
   cin >> input;
   cout << "\nBefore doubler is called... ";
   cout << "\ninput: " << input << " doubled: " << result << "\n";
   result = Doubler(input);
   cout << "\nBack from Doubler...\n";
   cout << "\ninput: " << input << "   doubled: " << result << "\n";
   return 0;
 }
```

Πολλαπλά return/2

```
int Doubler(int original)
{
    if (original <= 10000)
        return original * 2;
    else
        return -1;
    cout << "You can't get here!\n";
}
```

Προκαθορισμένοι Παράμετροι

```
#include <iostream.h>
int AreaCube(int length, int width = 25, int height = 1);
int main()  {
    int length = 100;
    int width = 50;
    int height = 2;
    int area;
    area = AreaCube(length, width, height);
    cout << "First area equals: " << area << "\n";
    area = AreaCube(length, width);
    cout << "Second time area equals: " << area << "\n";
    area = AreaCube(length);
   cout << "Third time area equals: " << area << "\n";
     return 0;  }
AreaCube(int length, int width, int height)   {
    return (length * width * height); }
```

Προκαθορισμένοι Παράμετροι - έξοδος

First area equals: 10000

Second time area equals: 5000

Third time area equals: 2500

Υπερφόρτωση συναρτήσεων

- ✓ Η C++ επιτρέπει τη πολλαπλή δημιουργία συναρτήσεων με το ίδιο όνομα
- ✓ Οι συναρτήσεις αυτές πρέπει να διαφέρουν στο πλήθος ή τον τύπο των παραμέτρων:

  int myFunction (int, int);

  int myFunction (long, long);

  int myFunction (long);

- ✓ Κάθε φορά καλείται αυτόματα η κατάλληλη ανάλογα με τις παραμέτρους που περνάμε.

Υπερφόρτωση συναρτήσεων

```
#include <iostream.h>
int Double(int);
long Double(long);
float Double(float);
double Double(double);
 int main()
 {
    int      myInt = 6500;
    long     myLong = 65000;
    float    myFloat = 6.5F;
    double   myDouble = 6.5e20;
```

Υπερφόρτωση συναρτήσεων

```
int      doubledInt;
long     doubledLong;
float    doubledFloat;
double   doubledDouble;
cout << "myInt: " << myInt << "\n";
cout << "myLong: " << myLong << "\n";
cout << "myFloat: " << myFloat << "\n";
cout << "myDouble: " << myDouble << "\n";
doubledInt = Double(myInt);
doubledLong = Double(myLong);
```

Υπερφόρτωση συναρτήσεων

```
    doubledFloat = Double(myFloat);
    doubledDouble = Double(myDouble);
    cout << "doubledInt: " << doubledInt << "\n";
    cout << "doubledLong: " << doubledLong << "\n";
    cout << "doubledFloat: " << doubledFloat << "\n";
    cout << "doubledDouble: " << doubledDouble << "\n";
    return 0;
  }
int Double(int original)   {
   cout << "In Double(int)\n";
   return 2 * original;  }
```

Υπερφόρτωση συναρτήσεων

```
long Double(long original)  {
    cout << "In Double(long)\n";
    return 2 * original;
  }
  float Double(float original)  {
    cout << "In Double(float)\n";
    return 2 * original;
  }
  double Double(double original)  {
    cout << "In Double(double)\n";
    return 2 * original;
 }
```

Υπερφόρτωση συναρτήσεων
Output

myInt: 6500

myLong: 65000

myFloat: 6.5

myDouble: 6.5e+20

In Double(int)

In Double(long)

In Double(float)

In Double(double)

DoubledInt: 13000

DoubledLong: 130000

DoubledFloat: 13

DoubledDouble: 1.3e+21

Inline συναρτήσεις

```
#include <iostream.h>
inline int Double(int);
int main()  {
    int target;
    cout << "Enter a number to work with: ";
    cin >> target;
    cout << "\n";
    target = Double(target);
    cout << "Target: " << target << endl;
    target = Double(target);
    cout << "Target: " << target << endl;
    target = Double(target);
    cout << "Target: " << target << endl;
      return 0;  }
 int Double(int target)  {
   return 2*target; }
```

Αναδρομή

- ✓Έχουμε αναδρομή όταν μία συνάρτηση καλεί τον εαυτό της
- ✓Η αναδρομή είναι πολύ χρήσιμη σε περίπτωση πολύπλοκων υπολογισμών

Αναδρομή

```
char *chicken_or_egg( int gen ) {
  if (gen == 0)
    return("Chicken!");
  else if (gen == 1)
    return(“Egg!”);
  else
    return(chicken_or_egg(gen-1));
}
```

Υπολογισμός παραγοντικού

```
int factorial( int x ) {
  if (x == 1)
    return(1);
  else
    return(x * factorial(x-1));
}
```

Σχεδιασμός αναδρομικών συναρτήσεων

- ✓ Ορίζουμε συνθήκη τερματισμού:
  - Πρόκειται για την περίπτωση όπου η συνάρτηση παύει να καλεί τον εαυτό της
- ✓ Ορίζουμε επανάκληση της συνάρτησης από τον εαυτό της

συνθήκη τερματισμού

- ✓ Η συνθήκη τερματισμού αντιστοιχεί σε μια περίπτωση που γνωρίζουμε ήδη την απάντηση ή υπολογίζεται εύκολα
- ✓ Αν δεν έχουμε συνθήκη τερματισμού δεν πρέπει να χρησιμοποιήσουμε αναδρομή ή δεν έχουμε καταλάβει το πρόβλημα

Επανάκληση

- ✓ Η επανάκληση χρησιμοποιείται για να λύσουμε κάποιο υπο-πρόβλημα
  - Σε κάθε επανάκληση οι παράμετροι θα πρέπει να διαφέρουν ώστε να πλησιάζουμε στη λύση

Αναδρομή

```
#include <iostream.h>
int fib(int n);
int main()    {
  int n, answer;
  cout << "Enter number to find: ";
  cin >> n;
  cout << "\n\n";
  answer = fib(n);
  cout << answer << " is the " << n << "th Fibonacci number\n";
   return 0;
}
```

Αναδρομή

```
int fib (int n)    {
  cout << "Processing fib(" << n << ")... ";
  if (n < 3 )      {
     cout << "Return 1!\n";
     return (1);
  }
  else     {
    cout << "Call fib(" << n-2 << ") and fib(" << n-1 << ").\n";
    return( fib(n-2) + fib(n-1));
  }
}
```

Αναδρομή - ouput

Enter number to find: 5

Processing fib(5)... Call fib(3) and fib(4).

Processing fib(3)... Call fib(1) and fib(2).

Processing fib(1)... Return 1!

Processing fib(2)... Return 1!

Processing fib(4)... Call fib(2) and fib(3).

Processing fib(2)... Return 1!

Processing fib(3)... Call fib(1) and fib(2).

Processing fib(1)... Return 1!

Processing fib(2)... Return 1!

5 is the 5th Fibonacci number

Παράδειγμα 1.1

```
#include <iostream>
int divide (int a, int b=2){
  int r;
  r=a/b;
  return (r);
}
int main (){
  cout << divide (12) << '\n';
  cout << divide (20,4) << '\n';
  return 0;
}
```

Παράδειγμα 2.1

```
#include <iostream>
void odd (int x);
void even (int x);
int main(){
  int i;
  do {
    cout << "Please, enter number (0 to exit): ";
    cin >> i;
    odd (i);
  } while (i!=0);
  return 0;}
```

Παράδειγμα 2.2

```
void odd (int x)
{
  if ((x%2)!=0) cout << "It is odd.\n";
  else even (x);
}
void even (int x)
{
  if ((x%2)==0) cout << "It is even.\n";
  else odd (x);
}
```

Παράδειγμα 3.1

```
#include <iostream>
long factorial (long a){
  if (a > 1)
   return (a * factorial (a-1));
  else
   return 1;
}
int main (){
  long number = 9;
  cout << number << "! = " << factorial (number);
  return 0;}
```

Λάθη?

```
#include <iostream.h>
int myFunc(unsigned short int x);
int main()
{
unsigned short int x, y;
y = myFunc(x);
cout << "x: " << x << " y: " << y << "\n";
return 0;
}

int myFunc(unsigned short int x)
{
return (4*x);
}
```

Λάθη?

```
#include <iostream.h>
void myFunc(unsigned short int x);
int main()
{
unsigned short int x=5, y;
y = myFunc(int);
cout << "x: " << x << " y: " << y << "\n";
}

void myFunc(unsigned short int x)
{
return (4*x);
}
```