Συναρτήσεις

Εργίνα Καβαλλιεράτου

Σήμερα

C++ Συναρτήσεις

Δήλωση συνάρτησης

Σύνταξη συνάρτησης

Πρότυπο συνάρτησης & συνάρτηση

Αλληλο-καλούμενες συναρτήσεις

Μεταβλητές (σφαιρικές, τοπικές, block)

Εμβέλεια μεταβλητών

Συναρτήσεις ως παράμετροι

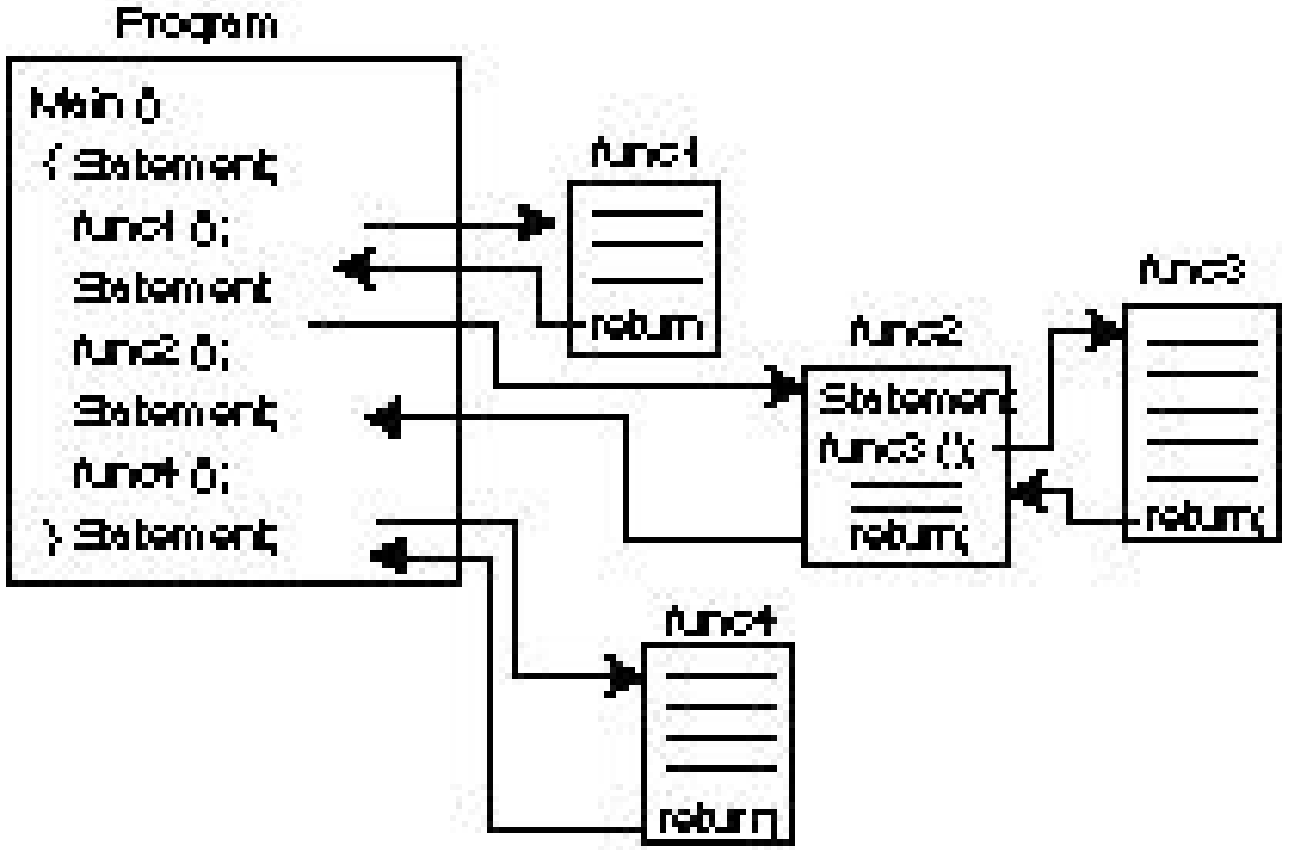
Πολλαπλά return

Προκαθορισμένοι Παράμετροι

C++ Συναρτήσεις

- Στον ΑΠ οι συναρτήσεις παραμένουν κεντρικό στοιχείο του προγράμματος.
- Η συνάρτηση είναι υποπρόγραμμα που επιδρά σε δεδομένα και μπορεί να επιστρέψει μια τιμή.
- Όλες οι συναρτήσεις στη C++ έχουν τύπο επιστροφής
  - ✓Όταν δεν χρειάζεται να επιστρέψει τίποτα μια συνάρτηση παίρνει τον τύπο **`void`**.
- Επιπλέον δέχονται παραμέτρους για να δουλέψουν πάνω σε αυτές.
  - ✓Κάθε παράμετρος έχει επίσης τύπο.

C++ Συναρτήσεις



C++ Συναρτήσεις

- Για να χρησιμοποιήσουμε μία συνάρτηση στο πρόγραμμα μας πρέπει πρώτα να τη δηλώσουμε και μετά να την ορίσουμε.
- Η δήλωση λέει στον compiler το όνομα της συνάρτησης, τι παραμέτρους παίρνει ως είσοδο.
- Ο ορισμός λέει στον compiler πως λειτουργεί μια συνάρτηση.
- Μία συνάρτηση δεν μπορεί να κληθεί από μία άλλη συνάρτηση αν δεν έχει πρώτα δηλωθεί.

Δήλωση συνάρτησης

Υπάρχουν τρεις τρόποι να δηλώσουμε μία συνάρτηση:

1. Να γράψουμε το πρότυπο της συνάρτησης σε ένα αρχείο και να δηλώσουμε με #include το αρχείο στο αρχείο του κυρίως προγράμματος.
2. Να γράψουμε το πρότυπο της συνάρτησης στο αρχείο του προγράμματος πριν από τη χρήση της.
3. Να ορίσουμε τη συνάρτηση πριν από την κλήση της και έτσι δηλώνεται αυτόματα.

Σύνταξη συνάρτησης

- Πρέπει να αποφασίσουμε:
  - Τύπο επιστροφής
  - Όνομα συνάρτησης
  - Παράμετροι (τύπο & πλήθος)
- Να γράψουμε τον κώδικα του κυρίου σώματος

Πρότυπο συνάρτησης & συνάρτηση

Επιστρεφ. τύπος

Όνομα Συν.

Παράμετροι

```
int add2ints(int a, int b);
```

```
int add2ints(int a, int b) {
  return(a+b);
}
```

Κυρίως σώμα

Πρότυπο συνάρτησης & συνάρτηση

Σύνταξη Προτύπου

return_type function_name ([type [parameterName]]...);

Σύνταξη συνάρτησης

```
return_type function_name ( [type parameterName]...)
{
    statements;
}
```

- Το πρότυπο συνάρτησης δε χρειάζεται να περιέχει τα ονόματα παραμέτρων αλλά επιβάλλεται να περιέχει τον τύπο τους.

Αλληλο-καλούμενες συναρτήσεις

foo1

```
...
foo2()
...
```

foo2

```
...
foo1()
...
```

Αλληλο-καλούμενες συναρτήσεις

```
char *chicken( int generation ) {
  if (generation == 0)
    return("Chicken!");
  else
    return(egg(generation-1));
}
```

```
char *egg( int generation ) {
  if (generation == 0)
    return("Egg!");
  else
    return(chicken(generation-1));
}
```

Αλληλο-καλούμενες συναρτήσεις

```
char *egg( int );
char *chicken( int );


int main(void) {
        int startnum;
        cout << "Enter starting generation of your chicken" << endl;
        cin >> startnum;
        cout <<"Your chicken started as a "<<chicken(startnum)<<endl;
        return(0); }
```

Παράδειγμα/1

```
typedef unsigned short USHORT;
#include <iostream.h>
USHORT FindArea(USHORT length, USHORT width);
int main()  {
    USHORT lengthOfYard, widthOfYard, areaOfYard;
    cout << "\nHow wide is your yard? ";
    cin >> widthOfYard;
    cout << "\nHow long is your yard? ";
    cin >> lengthOfYard;
```

Παράδειγμα/2

```
  areaOfYard= FindArea(lengthOfYard,widthOfYard);
  cout << "\nYour yard is ";
  cout << areaOfYard;
  cout << " square feet\n\n";
        return 0;
}
USHORT FindArea(USHORT l, USHORT w)
{
    return l * w;
}
```

Παράμετροι συναρτήσεων

- Οι παράμετροι λειτουργούν σαν τοπικές μεταβλητές μέσα στη συνάρτηση:
  - ✓Όταν κληθεί η συνάρτηση θα περάσουν μέσα οι τιμές των παραμέτρων
  - ✓Οι συναρτήσεις χρησιμοποιούν αντίγραφα των παραμέτρων για αυτό και οι αρχικές τιμές δεν αλλάζουν εκτός της συνάρτησης.

Παράδειγμα

```
int add2nums( int firstnum, int secondnum ) {
        int sum;

        sum = firstnum + secondnum;

        firstnum = 0;
        secondnum = 0;

        return(sum);
}
```

Τι γίνεται;

```
int add2nums(int a, int b) {
  a=a+b;
  return(a);
}
```

Testing add2nums

```
int main(void) {
	int y,a,b;

	cout << "Enter 2 numbers\n";
	cin >> a >> b;

	y = add2nums(a,b);

	cout << "a is " << a << endl;
	cout << "b is " << b << endl;
	cout << "y is " << y << endl;
	return(0);
}
```

Τοπικές μεταβλητές

- Οι παράμετροι και οι μεταβλητές που δηλώνονται μέσα σε μία συνάρτηση είναι **τοπικές**.
- Υπάρχουν μόνο μέσα στο σώμα της συνάρτησης
- Μετά τον τερματισμό της συνάρτησης παύουν να υπάρχουν

Block μεταβλητές

- Μπορούν επίσης να δηλωθούν μεταβλητές που θα υπάρξουν μόνο μέσα σε ένα ορισμένο block:

```
{
int foo;
      …
      …
}
```

Block μεταβλητές

```
void myFunc();
int main()    {
  int x = 5;
  cout << "\nIn main x is: " << x;
  myFunc();
  cout << "\nBack in main, x is: " << x;
  return 0;
}
```

Block μεταβλητές

```
void myFunc()   {
    int x = 8;
    cout << "\nIn myFunc, local x: " << x << endl;
    {
      cout << "\nIn block in myFunc, x is: " << x;
      int x = 9;
      cout << "\nVery local x: " << x;
    }
    cout << "\nOut of block, in myFunc, x: " << x << endl;
}
```

Block μεταβλητές - output

```
In main x is: 5
In myFunc, local x: 8

In block in myFunc, x is: 8
Very local x: 9
Out of block, in myFunc, x: 8

Back in main, x is: 5
```

Σφαιρικές (global) μεταβλητές

- Μπορούμε να δηλώσουμε μεταβλητές έξω από κάθε συνάρτηση και ονομάζονται **σφαιρικές**
- Οι σφαιρικές μεταβλητές είναι προσβάσιμες και μπορούν να αλλαχθούν από όλες τις συναρτήσεις.

Τοπικές μεταβλητές– παράδειγμα/1

```
#include <iostream.h>
   float Convert(float);
   int main() {
      float TempFer, TempCel;
      cout << "Please enter the temperature in Fahrenheit: ";
      cin >> TempFer;
      TempCel = Convert(TempFer);
      cout << "\nHere's the temperature in Celsius: ";
      cout << TempCel << endl;
       return 0;   }
```

Τοπικές μεταβλητές– παράδειγμα/2

```
float Convert(float TempFer) {
    float TempCel;
    TempCel = ((TempFer - 32) * 5) / 9;
    return TempCel; }
```

Σφαιρικές μεταβλητές

```
#include <iostream.h>
 void myFunction();
 int x = 5, y = 7;
 int main()  {
    cout << "x from main: " << x << "\n";
    cout << "y from main: " << y << "\n\n";
    myFunction();
    cout << "Back from myFunction!\n\n";
    cout << "x from main: " << x << "\n";
    cout << "y from main: " << y << "\n";
    return 0;  }
```

```
x from main: 5
y from main: 7

x from myFunction: 5
y from myFunction: 10

Back from myFunction!

x from main: 5
y from main: 7
```

Σφαιρικές μεταβλητές

```
void myFunction()  {
    int y = 10;
    cout << "x from myFunction: " << x << "\n";
    cout << "y from myFunction: " << y << "\n\n"; }
```

```
x from main: 5
y from main: 7

x from myFunction: 5
y from myFunction: 10

Back from myFunction!

x from main: 5
y from main: 7
```

Παράδειγμα/1

```
#include <iostream>
using namespace std;

int subtraction (int a, int b){
  int r;
  r=a-b;
  return r;
}
```

Παράδειγμα/2

```
void main (){
  int x=5, y=3, z;
  z = subtraction (7,2);
  cout << "The first result is " << z << '\n';
  cout << "The second result is " << subtraction (7,2) << '\n';
  cout << "The third result is " << subtraction (x,y) << '\n';
  z= 4 + subtraction (x,y);
  cout << "The fourth result is " << z << '\n';
}
```

Εμβέλεια μεταβλητών

- 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.

Παράδειγμα

```
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

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)))));

- Πρέπει η επιστροφή κάθε συνάρτησης να είναι ιδίου τύπου με την παράμετρο της οποίας θα πάρει τη θέση.

Παράδειγμα/1

```
#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;   }
```

```
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
```

Παράδειγμα/2

```
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"; }
```

```
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";
```

Πολλαπλά return/2

```
    result = Doubler(input);
    cout << "\nBack from Doubler...\n";
    cout << "\ninput: " << input << "   doubled: " << result << "\n";
    return 0;     }
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, width = 50, height = 2, 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); }
```

```
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/1

```
#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/2

```
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/3

```
    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/4

```
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;  }
```

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

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

Αναδρομή

- 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>
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 x);
int main() {
unsigned short x, y;
y = myFunc(x);
cout << "x: " << x << " y: " << y << "\n";
return 0; }
int myFunc(unsigned short x)
{
return (4*x);
}
```

Λάθη?

```
#include <iostream.h>
void myFunc(unsigned short x);
int main()
{
unsigned short x=5, y;
y = myFunc(int);
cout << "x: " << x << " y: " << y << "\n"; }
void myFunc(unsigned short x)
{
return (4*x);
}
```