Introduction to Programming and Computational Physics

Lesson n.6

Functions
Scope

subprograms in C

When we execute a C program, the main function is executed line by line, according to the given instructions sequence.

The C language offers the possibility to aggregate a set of instructions in an *subprogram* defined outside the main function and *called* by it (or by another subprogram).

Subprograms in C are called *functions.* They can be used to:

- give the main function a more compact and clear structure

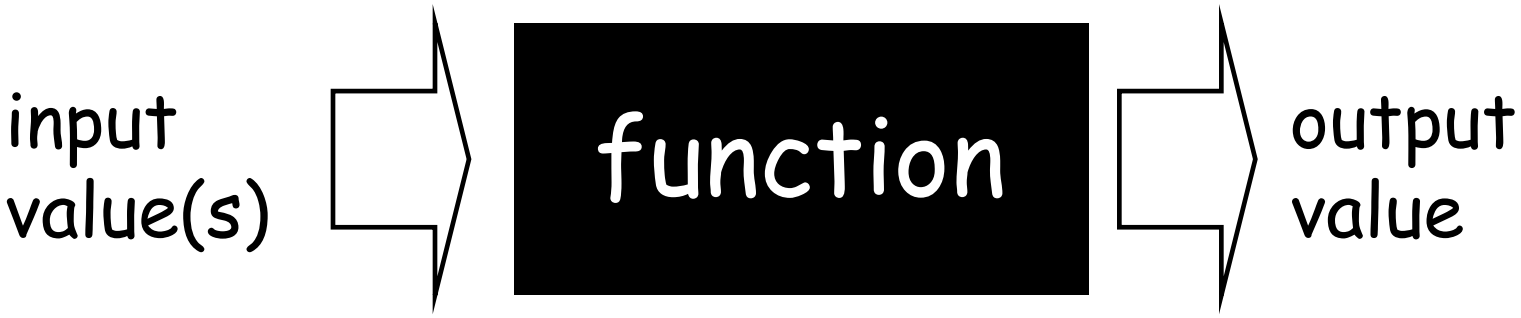
- avoid code replication

We used already several functions defined in external libraries sqrt, pow, printf ...

A library is a collection of functions

Modular programming

From a theoretical point of view, we may think at the use of functions as at a modular approach. Each function is assigned a given task and the programmer doesn't need to know how it is written. He deals just with input and output values.




```
double sqrt(double x);
```

An example: the cube

```c
#include <stdio.h>

double cube(double); //declaration

int main()
{
    double a,b;
    printf("Enter a number: ");
    scanf("%lf",&a);
    b = cube(a); //call
    printf("%lf to the third power is %lf \n",a,b);
    return 0;
}

double cube(double c)  //definition
{
    return (c*c*c);
}
```

Function declaration and definition

The general syntax for a function declaration is (prototype):

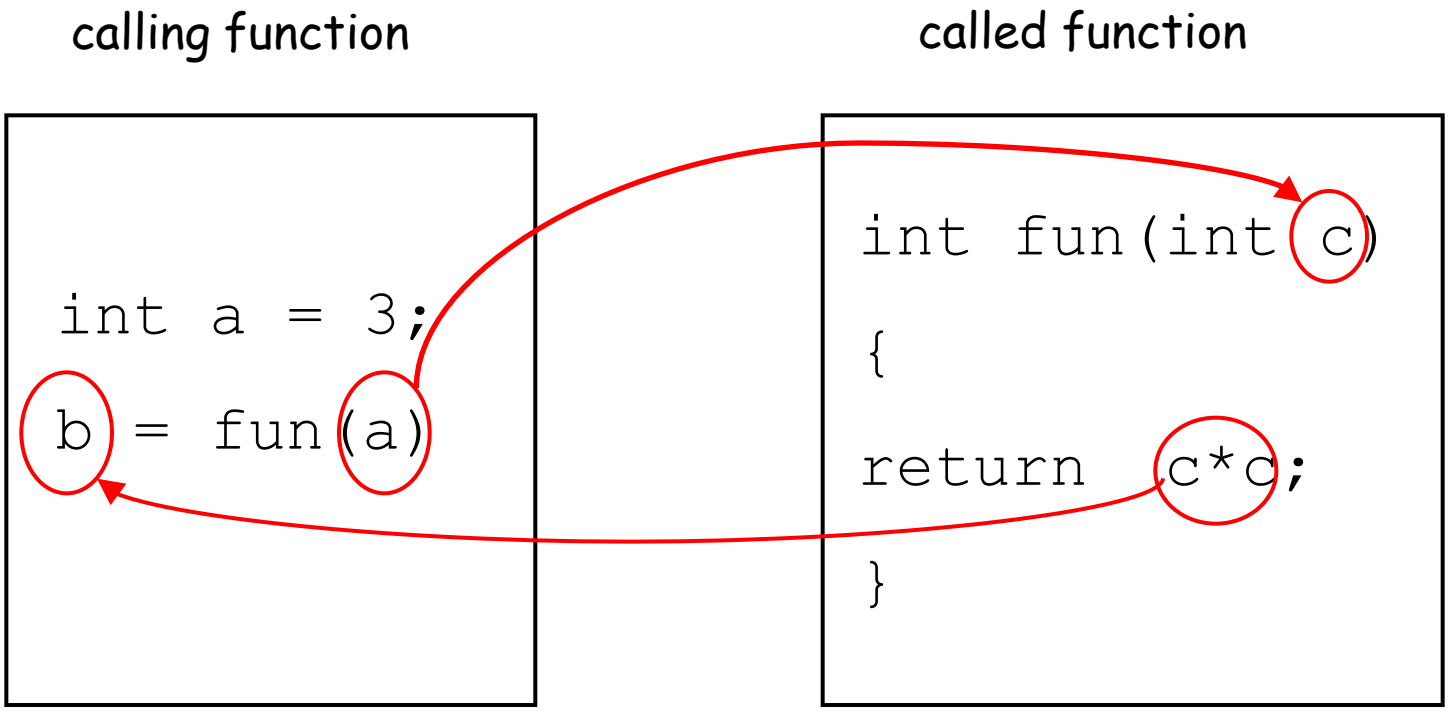
```
typeReturned nameFun(typePar1, typePar2, ...,typeParN);
```

The declaration must be followed by the function definition, otherwise the compiling will return an error

The general syntax for a function definition is:

```
typeReturned nameFun(typePar1 par1, typePar2 par2,
...,typeParN parN)
{
return …
}
```

Passing by value



The calling function **passes the value** to the input parameter(s) of the called one

Return

The result of a function call is given back by means of a return statement. Its syntax is:

```
return (expression);
```

Sometimes in one function more than one `return` is present

```
int absval(int a)
{
  if (a>=0) return a;
  else return -a;
}
```

If the first `return` is executed, the function execution stops and the control is given back to the calling function.

The type void

Let's now consider the case where a function doesn't need input parameter or return value. The C language has a special type: `void`

```
void fun1(int);
```

1) No returning value

```
int a;

fun1(a);
```

Function call

```
int fun2(void);

int fun2();
```

2) No input parameter

```
int b = fun2();
```

Function call

The type void

```c
#include <stdio.h>

void CheckIfPositive(float);

int main()
{
    float a;
    printf("Please enter a positive number ");
    scanf("%f",&a);
    CheckIfPositive(a);
    return 0;
}

void CheckIfPositive(float var)
{
    if(var<=0)
        printf("Hey! This number is not positive!\n");
    return;
}
```

The function `main`

If we simply declare a function as:

```
fun1(parameters);
```

The compiler assumes that the returned value is an integer.

If the main function is defined as:

```
main()
```

And no return value is given, usually the compilers return a warning (and convert to void the return value)

Scope

A variable is *visible* only in the block where it is declared.

```
...
{
  int a;
  a = 3;
}
printf("%d",a);
...
```

WRONG !!!

The variable `a` exists only inside the block. As soon as the program terminates the block execution it is deleted and its allocated memory made free.

Scope

It is possible to use the same variable name in two different blocks.

```
if(...)
{
  int a;
  a = 3;
}
while(...)
{
  float a;
  a = 4.5;
}
```

Unless it is really needed, this is anyway *not suggested*

Scope

The same applies to functions. A variable declared inside a function (main or any other) is visible only inside that function.

```
double myfun(double,int);

int main()

{

  double myvar1; int myvar2;

  double ret;

  ret = myfun(myvar1, myvar2);

...

}

double myfun(double myvar1, int myvar2)

{

  return sqrt(pow(myvar1,myvar2));

}
```

the volume of a cylinder

```c
#include <stdio.h>
#define pi 3.1415

float volcyl(float,float);
float areacircle(float);

int main()
{
   float radius;
   float height;
   float vol;
   printf ("Enter the radius: ");
   scanf("%f",&radius);
   printf("Enter the height: ");
   scanf("%f", &height);
   vol = volcyl(radius,height);
   printf("The volume is %f\n",vol);
   return 0;
}

float areacircle(float radius)
{
   return (pi*radius*radius);
}

float volcyl(float radius, float height)
{
   return height*areacircle(radius);
}
```

1)

3)

4)

2)

1) Calling function: `main`

   called function: `volcyl`

   passed values: `radius,height`

2) Calling function: `volcyl`

   called function: `areacircle`

   passed value: `radius`

3) `areacircle` returns its output to `volcyl`

4) `volcyl` returns its output to `main`

Global variables

If we want a variable to be visible everywhere, we have to define it outside any block

```
int glob; //this is a "global" variable


double fun1(int,int);

int fun2(double,char);

int main()

{

...

}

...
```

In this case the variable `glob` is visible by `fun1`, `fun2` and `main`