University Center Abdelhafid Boussouf -Mila-
Institute of Science and Technology

Computer Science 2

2023/2024

Chapter 01: Functions In C

What does it mean a function?

In C programming language, a function is a series of statements or instructions that known under one name. These statements have been grouped together to accomplish a task. A program in C can be considered as a series of functions, so that in the program we have at least one function which is "main( )" function. A function building block is a subprogram (subroutine) defined outside the main program. The benefit of using functions is that the function is defined once and can be used multiple times within the program. There is a distinction, which must be taken into account, between functions in C **library** and the functions created by the programmer.

Steps to create a function

There are three steps to create a function which are *the function declaration, function definition and function calls.* These steps can be explored in the following program

Program	*Output*

`#include <stdio.h>`
 	
`void message()  //function declaration`
`{`
`  printf("hello world");  //function definition`
`}`
`int main()`
` {`
 	
`  message();   //function call`
 	
`  return 0;`
` }`
	hello world

1- Function declaration:

The general form to make a function declaration is
return_type **name_of_the_function** (*parameter_1*, *parameter_2,…,….*);

In the program above: **void** is the return type, **message** is the function name. The appropriate way to declare a function without a parameter or an argument (as we see in the previous example) is that
void **name_of_the_function** **( )**;

void is used when a function does not return any value. There are other return-types of function rather than **void** likewise **int, float, double**… ,and this depends on the type of parameters (see the Figure1).

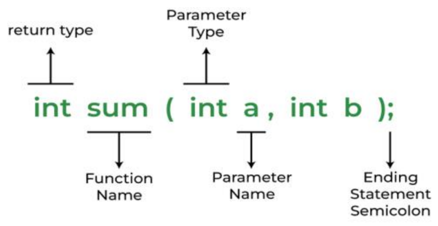


Figure1: *The form of a parameter dependent function declaration.*
Source of the image:*https://media.geeksforgeeks.org/wp-content/uploads/20230227121756/Functions-in-C-(1)-768.png*

2- Function definition:

The definition of a function (also called body of the function) contains the statements must be executed enclosed by a curly brackets { }.

3- Function call:

Declaring a function is accomplished when we write the function name followed by a semicolon “;” inside the main program. A function call is a command to the compiler to access the body of the function and start executing it.

Different cases of a function declaration

According to the function definition there are many ways to declare and use a function, and this due to the result of the function sub-program execution.

1- Function without argument (parameter) and without a return value

Example: (display a message)

Program	*Output*

#include <stdio.h>
 	
void letter()
  {
   printf("To me, you are the center of the universe. ");
    }
int main()
{
 	
   letter();
 	
   return 0;
}
	To me, you are the center of the universe.

If you need to print the message multiple times, simply make a function call repeatedly

Program	*Output*

#include <stdio.h>
 	
void letter()
  {
   printf("To me, you are the center of the universe.\n ");
    }
int main()
{
 	
   letter();
   letter();
   letter();
 	
   return 0;
}
	To me, you are the center of the universe.
	To me, you are the center of the universe.
	To me, you are the center of the universe.

2- Function with arguments and without a return value

Example: (the sum of two numbers)

Program

```c
#include <stdio.h>
void sum(int a, int b)
{
   int c;
   c=a+b;
   printf("%d",c);
}
int main()
{
   sum(5,6);

   return 0;
}
```

Output

```
11
```

In this program it is worth noting that C copies the values (5,6) into the function parameters (a,b). This form (**sum(5,6);**) of parameter passing named "call by value".

3- Function without argument and with a return value

Example: (the sum of two numbers)

Program

```c
#include <stdio.h>

int sum()
  {int a, b;
  printf("enter a:\n");
  scanf("%d",&a);
  printf("enter b:\n");
  scanf("%d",&b);
  return a+b;

    }
int main()
 {
 int c;
 c=sum();
printf("sum=%d",c);

   return 0;
 }
```

Output

```
sum=8
```

There's something different about this program that we have not seen before, which is represented by this statement "**c=sum( );**" and this is because the function returns a value (**return a+b;**). Meaning that the sum( ) function is just a value and this value must be declared within a variable which is **c** in the main program.

4- Function with argument and with a return value

Example: (Average of three numbers)

Program	*Output*

```
#include <stdio.h>

float average(float a, float b, float c)
{
   float d=(a+b+c)/3;
   return d;
}
int main() {
   float w, x=5, y=6.5, z=10;
   w=average(x,y,z);
   printf("The average of the three numbers=%f",w);

   return 0;
}
```
	The average of the three numbers=7.166667

The return type **float**, of the function, represents the data type that the function returns. This type of function declaration called **"function prototype".** Every function prototype is a function declaration. But not every declaration is a prototype. A function declaration without specifying parameters types (or with a void) is not a prototype.

The variable scope:

The scope of a variable is the extent to which the declared variable remains alive. In other words, the scope is the region in the program where the variable is available, and it cannot be accessed outside of this region. Based on this scope, the variables are divided into two parts

1. **Global variables:** the variable defined outside a function is called a *global variable* and its access is available to all functions throughout the entire program. The global variable is typically declared at the top of the program.

2. **Local variables:** when a declared variable is accessible only in a specific region or a block rather than the entire program, it is referred to as a *local variable*. The variable that is declared inside the body of the function is only known in this block and it is still unknown to other functions.

Example:

Program

```c
#include <stdio.h>
#include <math.h>

int z=2; //global variable declaration

float myfunc()
{
   int y=5; //local variable declaration  of the function myfunc()
   y+=z+2;
   return y;
}
int main()
{
float a,b; //local variables declaration of the main() function

a=sqrt(myfunc())+pow(z,2);
printf("a=%f\n",a);
b=2/(myfunc()+1);
printf("b=%f",b);
   return 0;
}
```

Output

```
a=7.000000
b=0.200000
```

In the previous program we see that the access to the global variable z is available throughout the program, and this variable becomes persistently recognized in both the main() and myfunc() blocks, and we do not need to re-declare it again within the blocks. On the other hand, the variables y, a and b are only defined in the function building block that it is part of and that defines these variables scopes.

In addition, it is essential to distinguish between two other types of variables which are:

1. **Actual parameters:** are values mentioned when calling the function. These values are substituted for the function parameters.
2. **Formal parameters:** are the variables mentioned in the function declaration, and one of the characteristics of these variables is their availability to capture the values from actual parameters.

Example:

Program	*Output*
`#include <stdio.h>`	
`void power(int a) // a is the formal parameter`	power(5,2)=25
`   {`	
`      int b;`	
`      b=a*a;`	
`      printf("power(%d,2)=%d\n", a, b);`	
`   }`	
`int main()`	
`{`	
`  power(5); // 5 is the actual parameter`	
`   return 0;`	
`}`	

Recursive function

In general, a recursion is a technique used to solve a difficult problem by dividing it into connected and less difficult parts. During resolving, we move to solve a less difficult part, then to the less difficult one, and so on until we finish.

In C, recursion occurs when a function calls itself, and this function said to be recursive.

At all times, the function does not stop calling itself. In order to avoid this indefinite case, a termination condition is required, in the function definition, to stop the recursion. This condition is called **base case**.

In order for the function to call itself, it is necessary to make a statement that makes a backward change in the function argument, which is known as the **recursive case**.

Example: a recursive function to find the sum of numbers from 1 to N

Program

```
#include <stdio.h>
 int recsum(int N)
   {
      if(N==1)   //base case
      {
         return 1;
      }
      return N+recsum(N-1); //recursive case
   }

int main()
{
  int m;
  m=recsum(5);
  printf("%d\n",m);

   return 0;
}
```

Output

```
15
```

Example explanation:

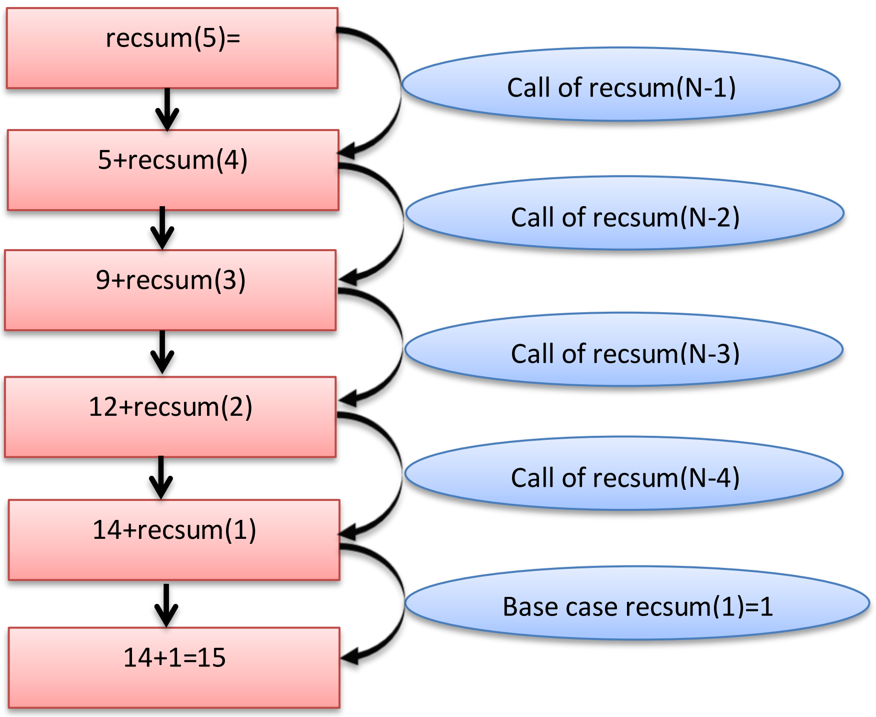


Figure 2: A simple diagram to illustrate the previous example.

In the previous program, the execution was done step by step, each step representing an individual call for the function as follows:

Recsum(5)=5+**recsum(4)**

=5+(4+**recsum(3)**)

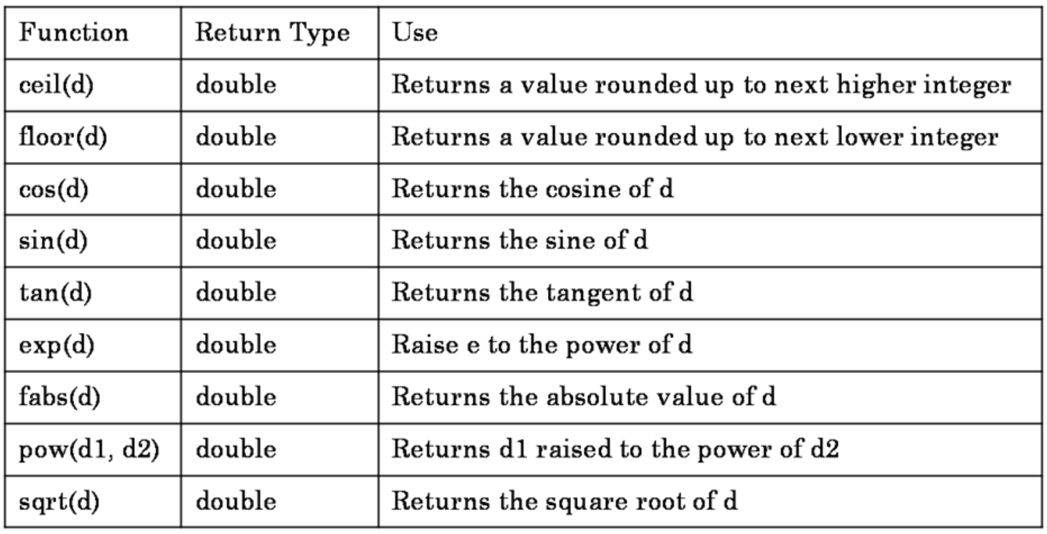
=5+(4+(3+**recsum(2)**))

=5+(4+(3+(2+**recsum(1)**)))

=5+4+3+2+1

=15

Some functions in C library



Function	Return Type	Use
ceil(d)	double	Returns a value rounded up to next higher integer
floor(d)	double	Returns a value rounded up to next lower integer
cos(d)	double	Returns the cosine of d
sin(d)	double	Returns the sine of d
tan(d)	double	Returns the tangent of d
exp(d)	double	Raise e to the power of d
fabs(d)	double	Returns the absolute value of d
pow(d1, d2)	double	Returns d1 raised to the power of d2
sqrt(d)	double	Returns the square root of d

Figure 3: Standard liberary functions in c