FUNCTIONS

A ***function*** is a group of statements that together perform a task. Every C program has at least one function, which is **main**(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Types of functions

- ***Predefined standard library functions*** – such as **puts**(), **gets**(), **printf**(), **scanf**() etc – these are the functions which already have a definition in header files (.h files like stdio.h), so we just call them whenever there is a need to use them.
- ***User Defined functions*** – the functions which we can create by ourselves, for example we can create function **abc**() and call it in **main**() in order to use it.

Defining a Function

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list )
{
   body of the function
}
```

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function:

- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the **return_type** is the keyword **void**.
- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** − The function body contains a collection of statements that define what the function does.

Examples

Given below is the source code for a function called **max**(). This function takes two parameters *a* and *b* and returns the maximum value between them.

```
int max(int a, int b)
{
  if (a > b) return a;
  return b;
}
```

Function **max()** can be written using ternary operator:

```
int max(int a, int b)
{
  return (a > b) ? a : b;
}
```

Function **sqr()** returns square of a number.

```
int sqr(int n)
{
  return n * n;
}
```

Function $f(x, y) = \sqrt{xy} + x^y$:

```
double f(double x, double y)
{
  return sqrt(x*y) + pow(x,y);
}
```

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example:

```
#include <stdio.h>

int max(int a, int b)
{
  if (a > b) return a;
  else return b;
}

int main(void)
```

```
{
  int a = 111, b = 222;
  int res = max(a,b);

  printf("a = %d\n",a);
  printf("b = %d\n",b);
  printf("max(a,b) = %d\n",res);
  return 0;
}
```

Example

Function f(*n*) returns the sum of digits for two digital number *n*.

```
#include <stdio.h>

int f(int n)
{
  int a = n / 10; // number of tens
  int b = n % 10; // number of units
  return a + b;
}

int main(void)
{
  printf("%d\n",f(34));
  return 0;
}
```

E-OLYMP 2606. Minimum and maximum Find minimum and maximum between two positive integers *a* and *b*.

► Use **min()** and **max()** functions.

E-OLYMP 108. Mean number Three integers *a*, *b*, *c* are given. Find their mean.

► Mean number equals to *a* + *b* + *c* – min(*a*, *b*, *c*) – max(*a*, *b*, *c*).

Minimum of three numbers can be found like min(*a*, min(*b*, *c*)) or we can implement a function minimum of three:

```
int min(int x, int y, int z)
{
  return min(x, min(y, z));
}
```

E-OLYMP 920. Use the function Three real numbers *x*, *y*, *z* are given. Find the value of min(max(x,y), max(y,z), x + y + z).

► Use **min()** and **max()** functions.

E-OLYMP 906. Product of digits Write a function f(*n*) that returns the product of digits for three digital number *n*.

► Let *a* = *n* / 100 is a digit of hundrets, *b* = *n* / 10 % 10 is a digit of tens, *c* = *n* % 10 is a digit of units. Then f(*n*) = *a* * *b* * *c*.

E-OLYMP 939. The square of sum Write a function f(n) that returns the square of sum of digits for two digital positive integer n.

► Let a is a digit of tens, b is a digit of units. Then f(n) = $(a + b)^2$.

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the ***formal parameters*** of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function **Call by value** and **Call by reference**.

Call by value

This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. ***Different memory*** is allocated for both actual and formal parameters.

- ***Actual parameter*** is the argument which is used in function call.
- ***Formal parameter*** is the argument which is used in function definition

```
#include <stdio.h>

int max(int a, int b)
{
  return (a > b) ? a : b;
}

int main(void)
{
  int a = 111, b = 222;
  int res = max(a,b);
  . . .
}
```

formal parameters

actual parameters

Function f has two formal parameters a and b. Inside the body of the function f we work with formal parameters, changing them.

```
#include <stdio.h>

void f(int a, int b)
{
  a++; b = b + 2;
  printf("%d %d\n",a,b); // 11 22
}

int main(void)
{
  int a = 10, b = 20;
  f(a,b);
  printf("%d %d\n",a,b); // 10 20
```

```
  return 0;
}
```

Call by reference

This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

```
#include <stdio.h>

void f(int &a, int &b)
{
  a++; b = b + 2;
  printf("%d %d\n",a,b); // 11 22
}

int main(void)
{
  int a = 10, b = 20;
  f(a,b);
  printf("%d %d\n",a,b); // 11 22
  return 0;
}
```

Let's implement the function *swap* that swaps the values.

```
#include <stdio.h>

void swap(int &a, int &b)
{
  int temp = a; a = b; b = temp;
}

int main(void)
{
  int a = 10, b = 20;
  printf("%d %d\n",a,b); // 10 20
  swap(a,b);
  printf("%d %d\n",a,b); // 20 10
  return 0;
}
```

We want to create a function *f* that returns two values: the sum and the product of two numbers. Let's pass two parameters *a* and *b* by value, and parameters *sum* and *prod* by reference.

```
#include <stdio.h>

void f(int x, int y, int &add, int &mult)
{
  add = x + y;
  mult = x * y;
}

int main(void)
```

```
{
  int a = 3, b = 5, sum, prod;
  f(a,b,sum,prod);
  printf("%d %d\n",sum,prod); // 8 15
  return 0;
}
```