| | |
|---|---|
| ○ Functions<br>○ Global and local variable<br>○ Define function<br>○ Call function<br>○ Ways of calling functions<br>○ Form of retrieving values from function<br>○ parameters arguments<br>○ factors effecting at using functions<br>○ functions of type void<br>○ User defined functions<br>○ Library of standards functions<br>○ String functions<br>○ Arithmetic functions<br>○ Date and time functions | الرابع عشر – الخامس عشر |

# Functions in C++ with example

BY CHAITANYA SINGH | FILED UNDER: LEARN C++

A function is block of code which is used to perform a particular task, for example let's say you are writing a large C++ program and in that program you want to do a particular task several number of times, like displaying value from 1 to 10, in order to do that you have to write few lines of code and you need to repeat these lines every time you display values. Another way of doing this is that you write these lines inside a function and call that function every time you want to display values. This would make you code simple, readable and reusable.

## *Syntax of Function*

```
return_type function_name (parameter_list)
{
   //C++ Statements
}
```

Let's take a simple example to understand this concept.

# A simple function example

```
#include <iostream>
using namespace std;
/* This function adds two integer values
 * and returns the result
 */int
sum(int num1, int num2){
   int num3 = num1+num2; return num3;
}

int main(){
   //Calling the function
   cout<<sum(1,99);
   return 0;
}
```

**Output:**

```
100
```

**The same program can be written like this:** Well, I am writing this program to let you understand an important term regarding functions, which is function declaration. Lets see the program first and then at the end of it we will discuss function declaration, definition and calling of function.

```
#include <iostream>
using namespace std;
//Function declaration
int sum(int,int);

//Main function
int main(){
   //Calling the function
   cout<<sum(1,99);
   return 0;
}
/* Function is defined after the main method
 */
int sum(int num1, int num2){
   int num3 = num1+num2;
   return num3;
}
```

**Function Declaration:** You have seen that I have written the same program in two ways, in the first program I didn't have any function declaration and in the second program I have function declaration at the beginning of the program. The thing is that when you define the function before the main() function in your program then you don't need to do function declaration but if you are writing your function after the main() function like we did in the second program then you need to declare the function first, else you will get compilation error.

**syntax of function declaration:**

```
return_type function_name(parameter_list);
```

**Note:** While providing parameter_list you can avoid the parameter names, just like I did in the above example. I have given `int sum(int,int);` instead of `int sum(int num1,int num2);`.

**Function definition:** Writing the full body of function is known as defining a function.
**syntax of function definition:**

```
return_type function_name(parameter_list) {
    //Statements inside function
}
```

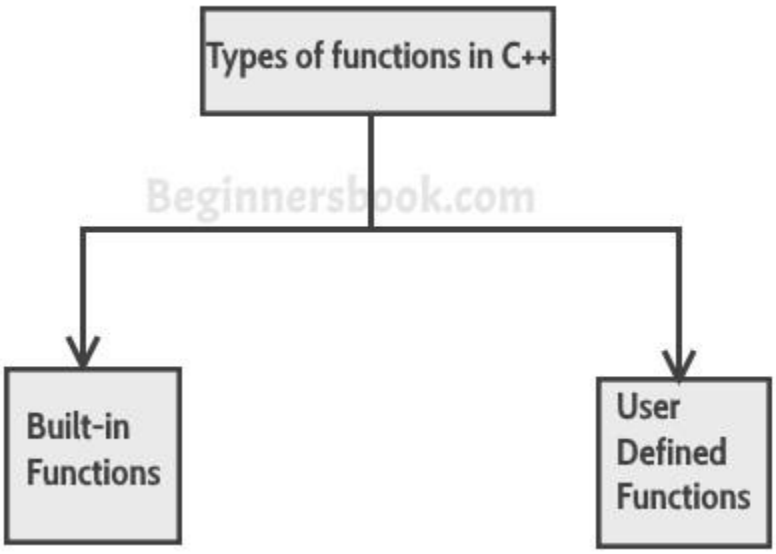
**Calling function:** We can call the function like this:

```
function_name(parameters);
```

Now that we understood the **working of function**, lets see the types of function in C++

# Types of function

We have two types of function in C++:




1) Built-in functions
2) User-defined functions

## 1) Build-it functions

Built-in functions are also known as library functions. We need not to declare and define these functions as they are already written in the C++ libraries such as iostream, cmath etc. We can directly call them when we need.

### *Example: C++ built-in function example*

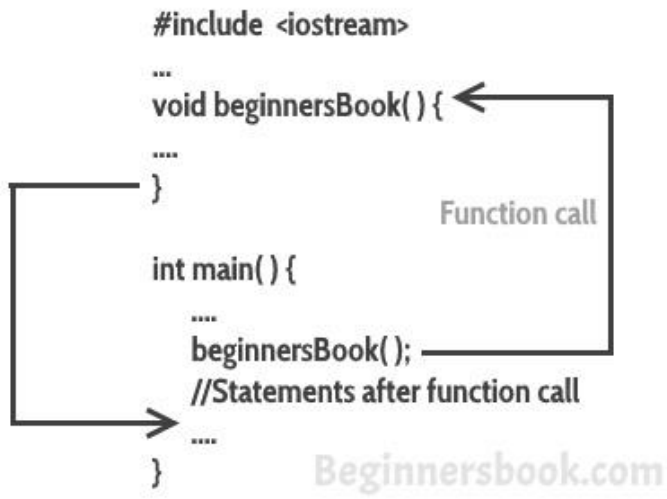
Here we are using built-in function pow(x,y) which is x to the power y. This function is declared in `cmath` header file so we have included the file in our program using `#include` directive.

```
#include <iostream>
#include <cmath>
using namespace std;
int main(){
    /* Calling the built-in function
     * pow(x, y) which is x to the power y
     * We are directly calling this function
     */
    cout<<pow(2,5);
    return 0;
}
```

**Output:**

```
32
```

## 2) User-defined functions



We have already seen user-defined functions, the example we have given at the beginning of this tutorial is an example of user-defined function. The functions that we declare and write in our programs are user-defined functions. Lets see another example of user-defined functions.

## *User-defined functions*

```
#include <iostream>
#include <cmath>
using namespace std;
//Declaring the function sum
int sum(int,int);

int main(){
   int x, y;
   cout<<"enter first number: ";
   cin>> x;

   cout<<"enter second number: ";
   cin>>y;

   cout<<"Sum of these two :"<<sum(x,y);
   return 0;
}
//Defining the function sum
int sum(int a, int b) {
   int c = a+b;
   return c;
}
```

**Output:**

```
enter first number: 22
enter second number: 19
Sum of these two :41
```

# C++ Date and Time

The C++ standard library does not provide a proper date type. C++ inherits the structs and functions for date and time manipulation from C. To access date and time related functions and structures, you would need to include <ctime> header file in your C++ program.

There are four time-related types: **clock_t, time_t, size_t**, and **tm**. The types - clock_t, size_t and time_t are capable of representing the system time and date as some sort of integer.

The structure type **tm** holds the date and time in the form of a C structure having the following elements −

```
struct tm {
   int tm_sec;   // seconds of minutes from 0 to 61
   int tm_min;   // minutes of hour from 0 to 59
   int tm_hour;  // hours of day from 0 to 24
   int tm_mday;  // day of month from 1 to 31
   int tm_mon;   // month of year from 0 to 11
   int tm_year;  // year since 1900
   int tm_wday;  // days since sunday
   int tm_yday;  // days since January 1st
   int tm_isdst; // hours of daylight savings time
}
```

Following are the important functions, which we use while working with date and time in C or C++. All these functions are part of standard C and C++ library and you can check their detail using reference to C++ standard library given below.

| Sr.No | Function & Purpose |
|---|---|
| 1 | **time_t time(time_t *time);**<br>This returns the current calendar time of the system in number of seconds elapsed since January 1, 1970. If the system has no time, .1 is returned. |
| 2 | **char *ctime(const time_t *time);**<br>This returns a pointer to a string of the form *day month year hours:minutes:seconds year\n\0*. |
| 3 | **struct tm *localtime(const time_t *time);**<br>This returns a pointer to the **tm** structure representing local time. |
| 4 | **clock_t clock(void);**<br>This returns a value that approximates the amount of time the calling program has been running. A value of .1 is returned if the time is not available. |
| 5 | **char * asctime ( const struct tm * time );**<br>This returns a pointer to a string that contains the information stored in the structure pointed to by time converted into the form: day month date hours:minutes:seconds year\n\0 |

| | |
|---|---|
| 6 | **struct tm *gmtime(const time_t *time);**<br>This returns a pointer to the time in the form of a tm structure. The time is represented in Coordinated Universal Time (UTC), which is essentially Greenwich Mean Time (GMT). |
| 7 | **time_t mktime(struct tm *time);**<br>This returns the calendar-time equivalent of the time found in the structure pointed to by time. |
| 8 | **double difftime ( time_t time2, time_t time1 );**<br>This function calculates the difference in seconds between time1 and time2. |
| 9 | **size_t strftime();**<br>This function can be used to format date and time in a specific format. |

## Current Date and Time

Suppose you want to retrieve the current system date and time, either as a local time or as a Coordinated Universal Time (UTC). Following is the example to achieve the same −

```
#include <iostream>
#include <ctime>

using namespace std;

int main() {
   // current date/time based on current system
   time_t now = time(0);

   // convert now to string form
   char* dt = ctime(&now);

   cout << "The local date and time is: " << dt << endl;

   // convert now to tm struct for UTC
   tm *gmtm = gmtime(&now);
   dt = asctime(gmtm);
   cout << "The UTC date and time is:"<< dt << endl;
}
```

When the above code is compiled and executed, it produces the following result −

```
The local date and time is: Sat Jan  8 20:07:41 2011
```

```
The UTC date and time is:Sun Jan  9 03:07:41 2011
```

# Format Time using struct tm

The **tm** structure is very important while working with date and time in either C or C++. This structure holds the date and time in the form of a C structure as mentioned above. Most of the time related functions makes use of tm structure. Following is an example which makes use of various date and time related functions and tm structure −

While using structure in this chapter, I'm making an assumption that you have basic understanding on C structure and how to access structure members using arrow -> operator.

```
#include <iostream>
#include <ctime>

using namespace std;

int main() {
   // current date/time based on current system
   time_t now = time(0);

   cout << "Number of sec since January 1,1970:" << now << endl;

   tm *ltm = localtime(&now);

   // print various components of tm structure.
   cout << "Year" << 1900 + ltm->tm_year<<endl;
   cout << "Month: "<< 1 + ltm->tm_mon<< endl;
   cout << "Day: "<<  ltm->tm_mday << endl;
   cout << "Time: "<< 1 + ltm->tm_hour << ":";
   cout << 1 + ltm->tm_min << ":";
   cout << 1 + ltm->tm_sec << endl;
}
```

When the above code is compiled and executed, it produces the following result −

```
Number of sec since January 1,1970:1563027637
Year2019
Month: 7
Day: 13
Time: 15:21:38
```

# C++ Math functions

C++ has many functions that allows you to perform mathematical tasks on numbers.

## Max and min

The `max(x,y)` function can be used to find the highest value of *x* and *y*:

### Example

```
cout << max(5, 10);
```

And the `min(x,y)` function can be used to find the lowest value of *x* and *y*:

### Example

```
cout << min(5, 10);
```

## C++ <cmath> Header

Other functions, such as `sqrt` (square root), `round` (rounds a number) and `log` (natural logarithm), can be found in the `<cmath>` header file:

### Example

```
// Include the cmath library
#include <cmath>

cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```

# Other Math Functions

A list of other popular Math functions (from the `<cmath>` library) can be found in the table below:

| Function | Description |
|---|---|
| abs(x) | Returns the absolute value of x |
| acos(x) | Returns the arccosine of x, in radians |
| asin(x) | Returns the arcsine of x, in radians |
| atan(x) | Returns the arctangent of x, in radians |
| cbrt(x) | Returns the cube root of x |
| ceil(x) | Returns the value of x rounded up to its nearest integer |
| cos(x) | Returns the cosine of x, in radians |
| cosh(x) | Returns the hyperbolic cosine of x, in radians |
| exp(x) | Returns the value of $E^x$ |
| expm1(x) | Returns $e^x$ -1 |
| fabs(x) | Returns the absolute value of a floating x |
| fdim(x, y) | Returns the positive difference between x and y |
| floor(x) | Returns the value of x rounded down to its nearest integer |
| hypot(x, y) | Returns sqrt($x^2$ +$y^2$) without intermediate overflow or underflow |

| | |
|---|---|
| fma(x, y, z) | Returns x*y+z without losing precision |
| fmax(x, y) | Returns the highest value of a floating x and y |
| fmin(x, y) | Returns the lowest value of a floating x and y |
| fmod(x, y) | Returns the floating point remainder of x/y |
| pow(x, y) | Returns the value of x to the power of y |
| sin(x) | Returns the sine of x (x is in radians) |
| sinh(x) | Returns the hyperbolic sine of a double value |
| tan(x) | Returns the tangent of an angle |
| tanh(x) | Returns the hyperbolic tangent of a double value |

C++ provides following two types of string representations −

- The C-style character string.
- The string class type introduced with Standard C++.

## The C-Style Character String

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

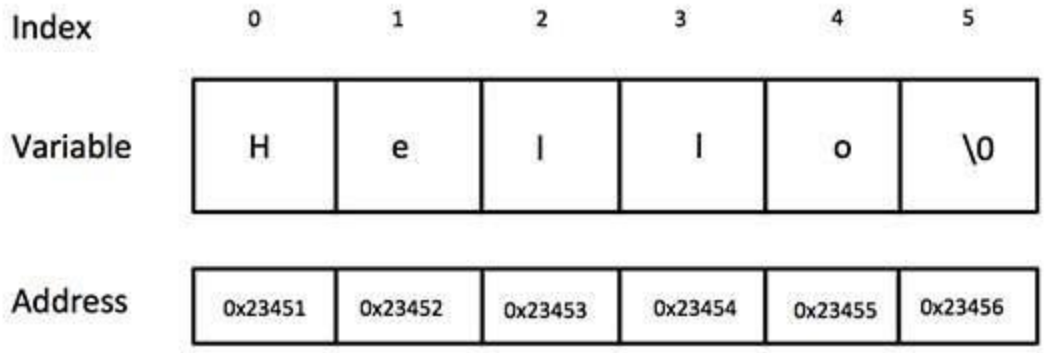
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

If you follow the rule of array initialization, then you can write the above statement as follows −

```
char greeting[] = "Hello";
```

Following is the memory presentation of above defined string in C/C++ −




Actually, you do not place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array. Let us try to print above-mentioned string −

Live Demo

```
#include <iostream>

using namespace std;

int main () {

   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

   cout << "Greeting message: ";
   cout << greeting << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Greeting message: Hello
```

C++ supports a wide range of functions that manipulate null-terminated strings –

| Sr.No | Function & Purpose |
|---|---|
| 1 | **strcpy(s1, s2);**<br>Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**<br>Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**<br>Returns the length of string s1. |
| 4 | **strcmp(s1, s2);**<br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**<br>Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**<br>Returns a pointer to the first occurrence of string s2 in string s1. |

Following example makes use of few of the above-mentioned functions −

```
#include <iostream>
#include <cstring>

using namespace std;

int main () {

   char str1[10] = "Hello";
   char str2[10] = "World";
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows −

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```