Calling a function

Let's see how to call our own function in a program.

We'll cover the following

Introduction

The functions created in a program are not executed until we call them. When we call the function, control is given to the very first statement inside the called function. The basic syntax for calling a function is given below:

```
int  main  (  )
{
function_name ( values of parameters ) ;

return 0;

}
```

To call a function in a program, we have to write a function name followed by values of arguments in the round brackets and the semicolon.

> We can call a function from any other function in a program.

Example program

Consider the blender example given in this lesson. Let's declare, define, and call a

function `make_juice`.

Run the code below and see the output!

```
#include <iostream>

using namespace std;
// Function declaration
int make_juice(int water, int fruit);

int main() {
  // Initialize variables apple and water
  int apples = 5;
  int water_glass = 3;
  // Declares a variable juice_glass
  int juice_glass;
  // Calls function make_juice and save its output in juice_glass
  juice_glass = make_juice(water_glass, apples);
  // Prints value of juice_glass
  cout << "Number of juice glass = " << juice_glass;

  return 0;
}

// Function definition
int make_juice(int water, int fruit) {
  // Define new  variable juice of int type
  int juice;
  // Adds water in apple and save output in juice
  juice = water + fruit;
  // Prints text on the screen
  cout << "Your juice is ready" << endl;
  // Returns juice value in output
  return juice;

}
```

Explanation

In the code above, we have already discussed the function definition in this lesson. Let's discuss how to call a function in C++.

Line No. 9: Initialize `apples` to `5`

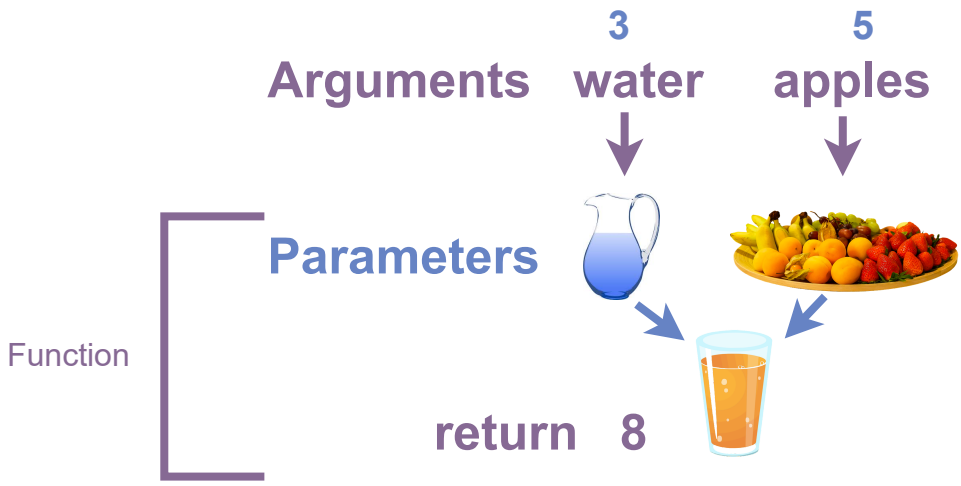
Line No. 10: Initialize `water_glass` to `3`

Line No. 12: Declares a variable `juice_glass`

Line No. 14: Calls the function `make_juice` We call a function by writing its name, followed by round brackets. It returns an integer value in the output, which is

stored in `juice_glass`. When we call a function `make_juice` in the `main()`, the program control is given to the first statement in the function's body.

Line No. 16: Prints value of `juice_glass`



Is it necessary to declare a function?

In the above code, we declare a function before the `main` function. Then, we define it later after the main function. In C++, statements are executed from top to bottom. If we don't declare the function before `main()`, our program is unaware of it, and we get a compilation error.

> We cannot declare the function after the `main` function; else, we get an error.

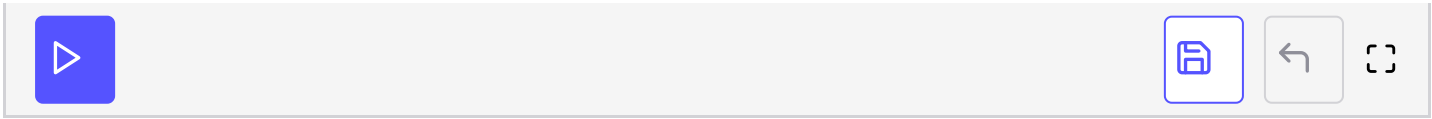
You are probably wondering whether it's possible to define a function before `main()` and then call it later in a program?

Yes, it is possible. If you are defining your function before the `main` function, then the function declaration is not necessary.

Run the program below and see the output!

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

// Function definition
int make_juice ( int water , int fruit){
// Define new  variable juice of int type
  int juice ;
// Adds water in apple and saves the output in juice
  juice = water + fruit;
// Prints text on the screen
```

```
  cout << "Your juice is ready" << endl ;
 // Returns juice value in output
  return juice;

}


int main() {
  // Initialize variables apple and water
  int apples = 5;
  int water_glass = 3;
  // Declares a variable juice_glass
  int juice_glass;
  // Calls function make_juice and save its output in juice_glass
  juice_glass = make_juice ( water_glass , apples);
  // Prints value of juice_glass
  cout << "Number of juice glass = " << juice_glass;

  return 0;
}
```



In the above code, we have removed the function declaration and defined our function before the `main` function. This gives us the same output.

Calling function multiple times

We can call the function as many times as we want with different inputs.

Press the **RUN** button and see the output!

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

// Function definition
int make_juice ( int water , int fruit){
// Define new  variable juice of int type
  int juice ;
// Adds water in apple and saves the output in juice
  juice = water + fruit;
// Prints text on the screen
  cout << "Your juice is ready" << endl ;
 // Returns juice value in output
  return juice;

}


int main() {
  // Declares a variable juice_glass
  int juice_glass;

  // Calls function make_juice and save its output in juice_glass
```

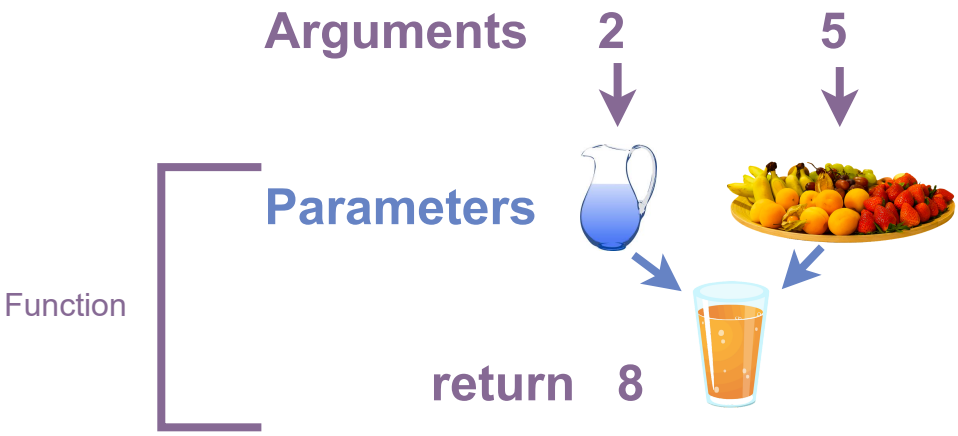
```
  juice_glass = make_juice ( 2 , 5);
  // Prints value of juice_glass
  cout << "Number of juice glass = " << juice_glass << endl;
  juice_glass = make_juice ( 6 , 11);
  // Prints value of juice_glass
  cout << "Number of juice glass = " << juice_glass << endl;

  return 0;
}
```

In the above code, we call the `make_juice` function twice in a program.

Line No. 23: Calls the `make_juice` function and then stores the returned value in `juice_glass`. Now, you can notice that we are passing values directly as arguments to the function.

`make_juice (2 , 5)`

> We can initialize a variable and then pass the identifier to the function parameter, or we can pass the value directly to the function parameters.



Line No. 26: Calls the `make_juice` function and then stores the returned value in `juice_glass`. Now, we are calling the function with different values.

`make_juice (6 , 11)`

What is the output of the following code?

```
int number_sum (int num1 , int num2){
```

```
int number_sum (int num1 , int num2){
return num1 + num2;
}

int main() {
  float value1 = 10.1;
  float value2 = 20.9;
  int sum = number_sum ( value1 , value2 ) ;
  cout << sum ;
  return 0;
}
```

Retake Quiz

Let’s get into the details of the types of function parameters in C++.