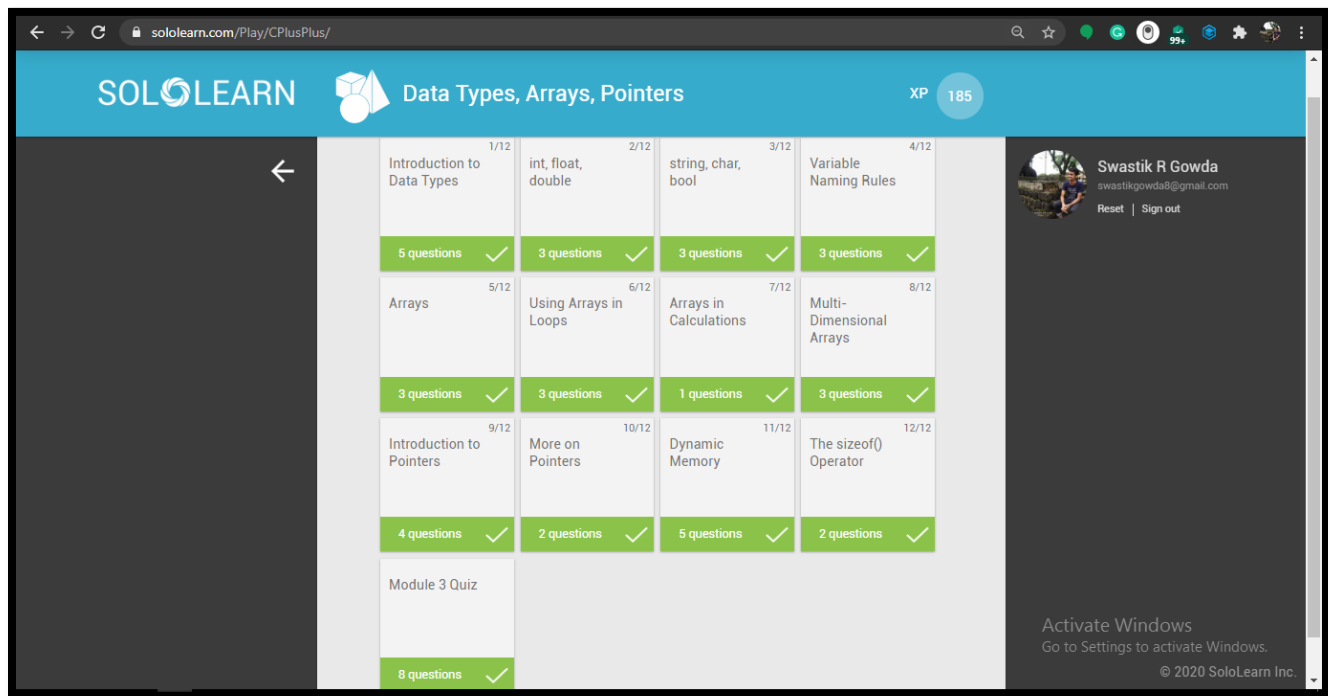


## DAILY ASSESSMENT

|                    |   |                     |                             |
|--------------------|---|---------------------|-----------------------------|
| Date:              | 23-June-2020                              | Name:               | Swastik R Gowda             |
| Course:            | Solo-Learn C++                            | USN:                | 4AL17EC091                  |
| Topic:             | Module - 3 : Data Types, Arrays, Pointers | Semester & Section: | 6 <sup>th</sup> Sem 'B' Sec |
| Github Repository: | swastik-gowda                             |                     |                             |

### FORENOON SESSION DETAILS

#### Image of session



**Report – Report can be typed or hand written for up to two pages.**

### Data Types

- ❖ The operating system allocates memory and selects what will be stored in the reserved memory based on the variable's data type.
- ❖ The data type defines the proper use of an identifier, what kind of data can be stored, and which types of operations can be performed.

### Integers

The integer type holds non-fractional numbers, which can be positive or negative. Examples of integers would include 42, -42, and similar numbers.

Use the int keyword to define the integer data type.

***int a = 42;***

Several of the basic types, including integers, can be modified using one or more of these type modifiers:

***signed:*** A signed integer can hold both negative and positive numbers.  
***unsigned:*** An unsigned integer can hold only positive values.  
***short:*** Half of the default size.  
***long:*** Twice the default size.

For example:

***unsigned long int a;***

## **Float**

- ❖ A floating point type variable can hold a real number, such as 420.0, -3.33, or 0.03325.
- ❖ The words floating point refer to the fact that a varying number of digits can appear before and after the decimal point.
- ❖ There are three different floating point data types: float, double, and long double.
- ❖ In most modern architectures, a float is 4 bytes, a double is 8, and a long double can be equivalent to a double (8 bytes), or 16 bytes.

For example:

***double temp = 4.21;***

## **Strings**

- ❖ A string is an ordered sequence of characters, enclosed in double quotation marks.
- ❖ It is part of the Standard Library.
- ❖ You need to include the ***<string>*** library to use the string data type. Alternatively, you can use a library that includes the string library.

```
#include <string>  
using namespace std;  
int main()  
{  
    string a = "I am learning C++";  
    return 0;  
}
```

## **Characters**

- ❖ A char variable holds a 1-byte integer. However, instead of interpreting the value of the char as an integer, the value of a char variable is typically interpreted as an ASCII character.
- ❖ A character is enclosed between single quotes (such as 'a', 'b', etc.).

For example:

***char test = 'S';***

## Booleans

- ❖ Boolean variables only have two possible values: true (1) and false (0).
- ❖ To declare a Boolean variable, we use the keyword `bool`.

```
bool online = false;  
bool logged_in = true;
```

## Variable Naming Rules

Use the following rules when naming variables:

- ❖ All variable names must begin with a letter of the alphabet or an underscore (\_).
- ❖ After the initial letter, variable names can contain additional letters, as well as numbers.
- ❖ Blank spaces or special characters are not allowed in variable names.
- ❖ C++ keyword (reserved word) cannot be used as variable names.

There are two known naming conventions:

**Pascal case**: The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized. For example: ***BackColor***

**Camel case**: The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized. For example: ***backColor***

## Arrays

- ❖ An array is used to store a collection of data, but it may be useful to think of an array as a collection of variables that are all of the same type.
- ❖ Instead of declaring multiple variables and storing individual values, you can declare a single array to store all the values.
- ❖ When declaring an array, specify its element types, as well as the number of elements it will hold.

For example:

```
int a[5];
```

In the example above, variable ***a*** was declared as an array of five integer values [specified in square brackets].

You can initialize the array by specifying the values it holds:

```
int b[5] = {11, 45, 62, 70, 88};
```

The values are provided in a comma separated list, enclosed in {curly braces}.

|     |     |     |     |     |
|-----|-----|-----|-----|-----|
| 11  | 45  | 62  | 70  | 88  |
| [0] | [1] | [2] | [3] | [4] |

## Multi-Dimensional Arrays

A multi-dimensional array holds one or more arrays. Declare a multidimensional array as follows.

***type name[size1][size2]...[sizeN];***

Here, we've created a two-dimensional 3x4 integer array:

***int x[3][4];***

Visualize this array as a table composed of 3 rows, and 4 columns.

|       | Column 1             | Column 2             | Column 3             | Column 4             |
|-------|----------------------|----------------------|----------------------|----------------------|
| Row 1 | <code>x[0][0]</code> | <code>x[0][1]</code> | <code>x[0][2]</code> | <code>x[0][3]</code> |
| Row 2 | <code>x[1][0]</code> | <code>x[1][1]</code> | <code>x[1][2]</code> | <code>x[1][3]</code> |
| Row 3 | <code>x[2][0]</code> | <code>x[2][1]</code> | <code>x[2][2]</code> | <code>x[2][3]</code> |

## Pointers

- ❖ Every variable is a memory location, which has its address defined.
- ❖ That address can be accessed using the ampersand (&) operator (also called the address-of operator), which denotes an address in memory.

For example:

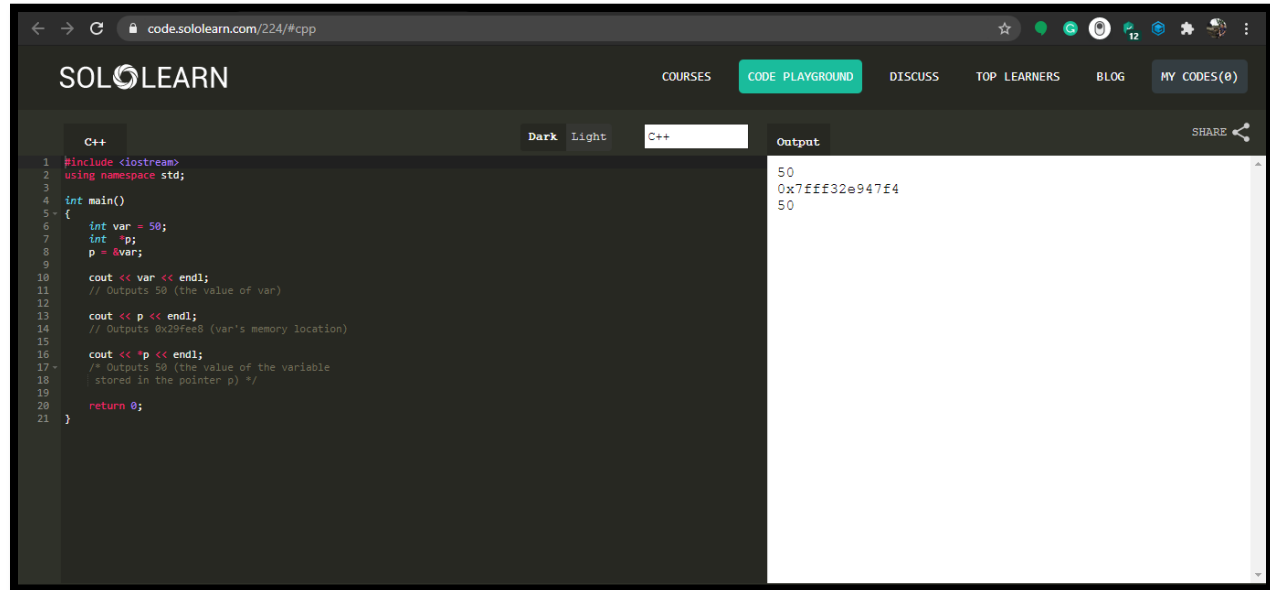
```
int score = 5;  
cout << &score << endl;
```

- ❖ A pointer is a variable, with the address of another variable as its value.
- ❖ In C++, pointers help make certain tasks easier to perform.
- ❖ Other tasks, such as dynamic memory allocation, cannot be performed without using pointers.
- ❖ All pointers share the same data type - a long hexadecimal number that represents a memory address.

There are two operators for pointers:

**Address-of operator (&):** returns the memory address of its operand.

**Contents-of (or dereference) operator (\*):** returns the value of the variable located at the address specified by its operand.



The screenshot shows a web-based C++ code editor with the following code:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int var = 50;
7     int *p;
8     p = &var;
9
10    cout << var << endl;
11    // Outputs 50 (the value of var)
12
13    cout << p << endl;
14    // Outputs 0x7ffef32e947f4 (var's memory location)
15
16    cout << *p << endl;
17    // Outputs 50 (the value of the variable
18    // stored in the pointer p) */
19
20    return 0;
21 }
```

The output on the right shows:

```
50
0x7ffef32e947f4
50
```

## Static & Dynamic Memory

In a C++ program, memory is divided into two parts:

**The stack:** All of your local variables take up memory from the stack.

**The heap:** Unused program memory that can be used when the program runs to dynamically allocate the memory.

**new int;**

## Sizeof

While the size allocated for varying data types depends on the architecture of the computer you use to run your programs, C++ does guarantee a minimum size for the basic data types:

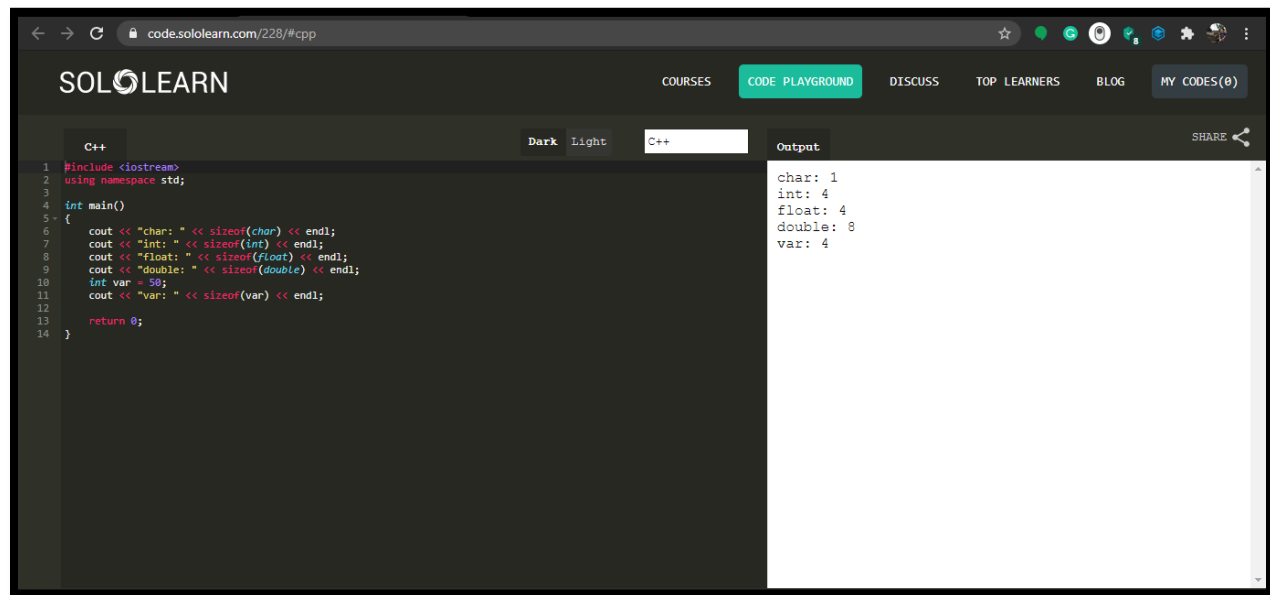
| Category       | Type        | Minimum Size |
|----------------|-------------|--------------|
| boolean        | bool        | 1 byte       |
| character      | char        | 1 byte       |
| integer        | short       | 2 bytes      |
|                | int         | 2 bytes      |
|                | long        | 4 bytes      |
|                | long long   | 8 bytes      |
| floating point | float       | 4 bytes      |
|                | double      | 8 bytes      |
|                | long double | 8 bytes      |

The sizeof operator can be used to get a variable or data type's size, in bytes.

Syntax:

***sizeof (data type)***

The sizeof operator determines and returns the size of either a type or a variable in bytes.



The screenshot shows a web-based C++ code playground interface. The browser address bar displays 'code.sololearn.com/228/#cpp'. The page header includes the 'SOLOLEARN' logo and navigation links for 'COURSES', 'CODE PLAYGROUND' (highlighted), 'DISCUSS', 'TOP LEARNERS', 'BLOG', and 'MY CODES(0)'. The interface is set to 'Dark' theme. The code editor on the left contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "char: " << sizeof(char) << endl;
7     cout << "int: " << sizeof(int) << endl;
8     cout << "float: " << sizeof(float) << endl;
9     cout << "double: " << sizeof(double) << endl;
10    int var = 50;
11    cout << "var: " << sizeof(var) << endl;
12
13    return 0;
14 }
```

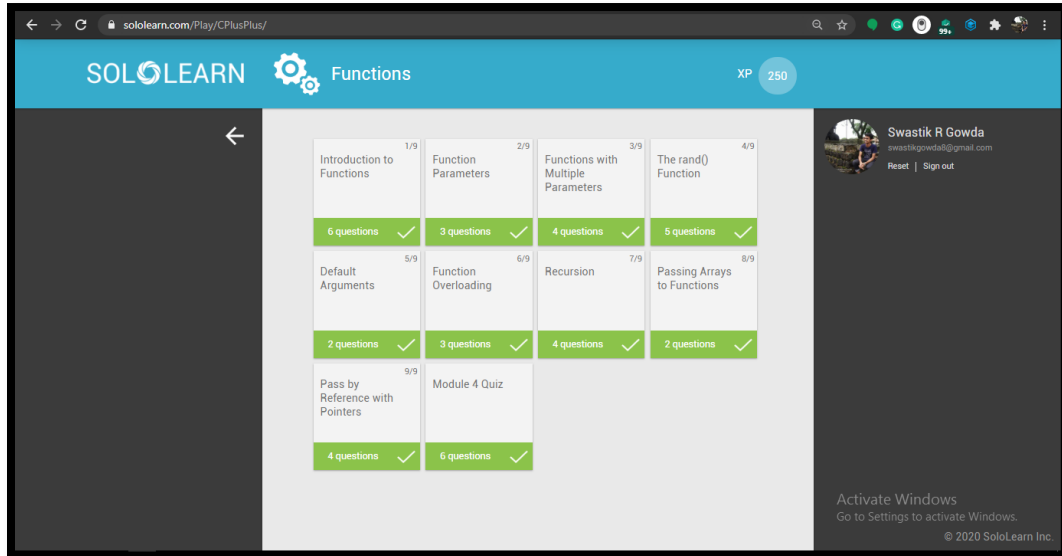
The 'Output' panel on the right displays the results of the program execution:

```
char: 1
int: 4
float: 4
double: 8
var: 4
```

|         |                        |                     |                             |
|---------|------------------------|---------------------|-----------------------------|
| Date:   | 23-June-2020           | Name:               | Swastik R Gowda             |
| Course: | Solo-Learn C++         | USN:                | 4AL17EC091                  |
| Topic:  | Module - 4 : Functions | Semester & Section: | 6 <sup>th</sup> Sem 'B' Sec |

### AFTERNOON SESSION DETAILS

#### Image of session



Report – Report can be typed or hand written for up to two pages.

## Functions

A function is a group of statements that perform a particular task.

- ❖ Using functions can have many advantages, including the following:
- ❖ You can reuse the code within a function.
- ❖ You can easily test individual functions.
- ❖ If it's necessary to make any code modifications, you can make modifications within a single function, without altering the program structure.
- ❖ You can use the same function for different inputs.

### Return Type

- ❖ A function's return type is declared before its name.
- ❖ Occasionally, a function will perform the desired operations without returning a value. Such functions are defined with the keyword void.

## Defining a Function

Define a C++ function using the following syntax:

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

**return-type:** Data type of the value returned by the function.

**function name:** Name of the function.

**parameters:** 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.

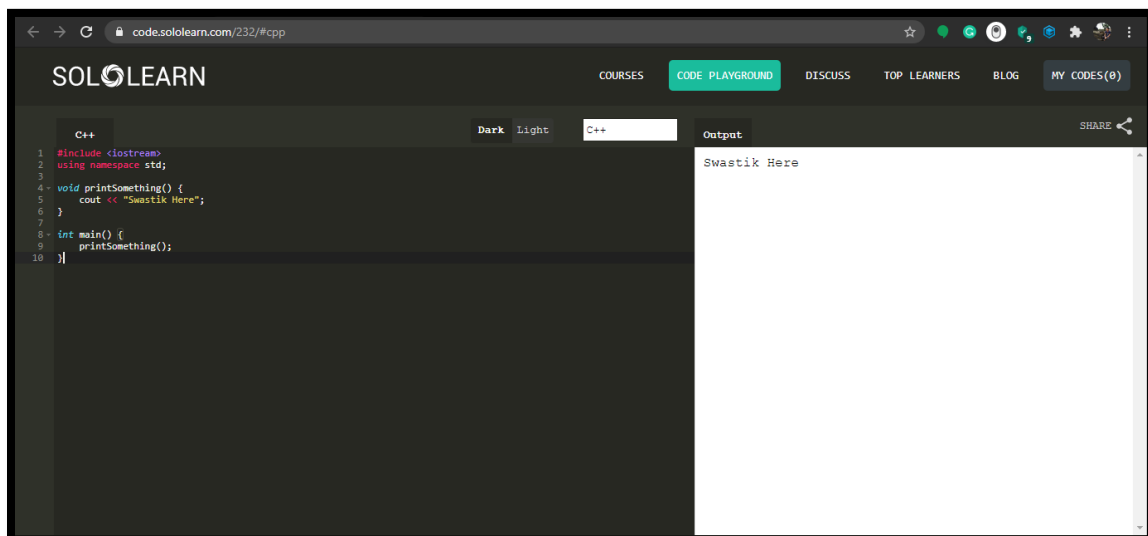
**body of the function:** A collection of statements defining what the function does

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

```
void printSomething() {  
    cout << " Swastik Here";  
}
```

```
int main() {  
    printSomething();
```

```
    return 0;  
}
```



The screenshot shows the Sololearn Code Playground interface. The browser address bar displays 'code.sololearn.com/232/#cpp'. The page has a dark theme. The 'CODE PLAYGROUND' tab is active. The code editor on the left contains the following C++ code:

```
1 #include <iostream>  
2 using namespace std;  
3  
4 void printSomething() {  
5     cout << " Swastik Here";  
6 }  
7  
8 int main() {  
9     printSomething();  
10 }
```

The 'Output' panel on the right shows the result of running the code: 'Swastik Here'. The top navigation bar includes links for COURSES, CODE PLAYGROUND, DISCUSS, TOP LEARNERS, BLOG, and MY CODES(0).



## Overloading

Function overloading allows creating multiple functions with the same name, so long as they have different parameters.

For example, you might need a `printNumber ()` function that prints the value of its parameter.

```
void printNumber(int a) {  
    cout << a;  
}
```

This is effective with integer arguments only. Overloading it will make it available for other types, such as floats.

```
void printNumber(float a) {  
    cout << a;  
}
```

Ex-

```
void printNumber(int x) {  
    cout << "Prints an integer: " << x << endl;  
}  
void printNumber(float x) {  
    cout << "Prints a float: " << x << endl;  
}  
int main() {  
    int a = 16;  
    float b = 54.541;  
    printNumber(a);  
    printNumber(b);  
}
```

### Output:

Prints an integer: 16

Prints a float: 54.541

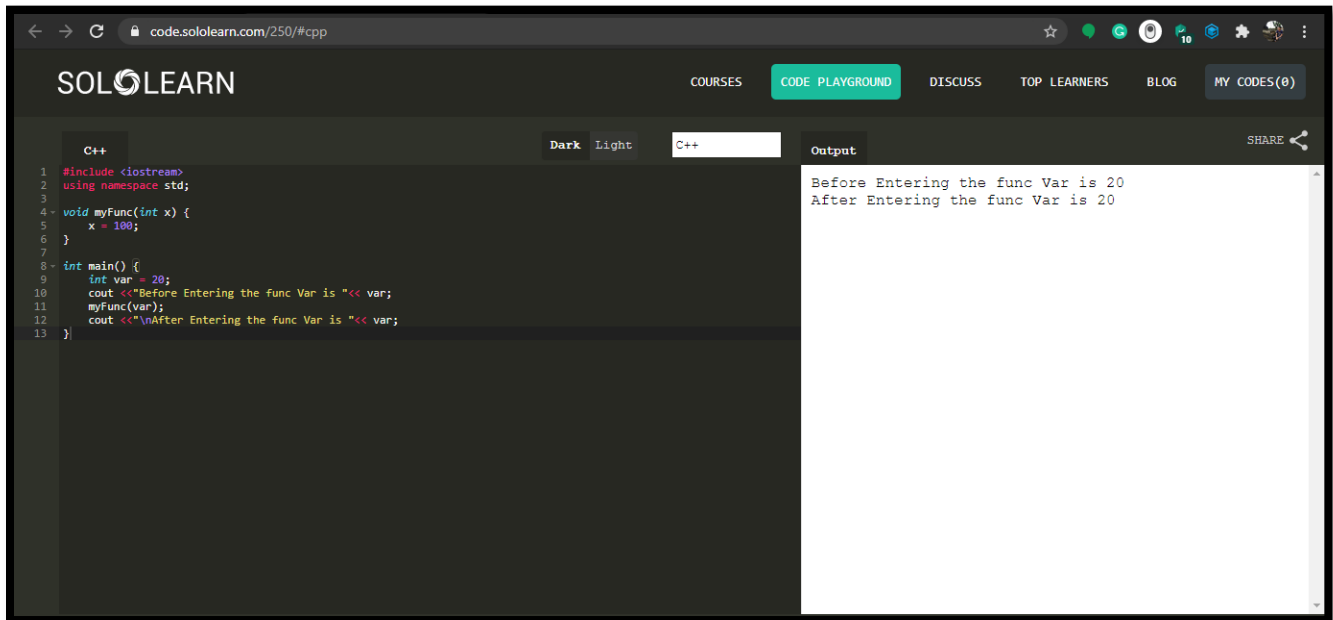
## Function Arguments

There are two ways to pass arguments to a function as the function is being called.

**By value:** This method copies the argument's actual value into the function's formal parameter. Here, we can make changes to the parameter within the function without having any effect on the argument.

**By reference:** This method copies the argument's reference into the formal parameter. Within the function, the reference is used to access the actual argument used in the call. This means that any change made to the parameter affects the argument.

**Passing 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.



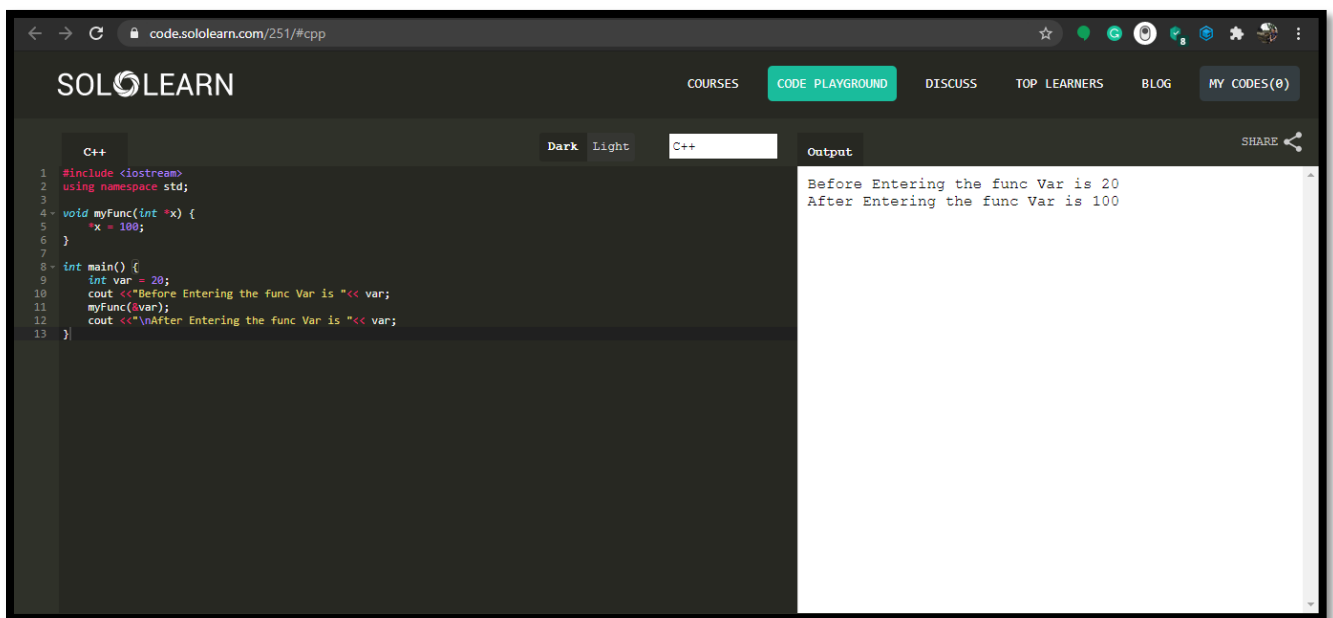
The screenshot shows a web browser at code.sololearn.com/250/#cpp. The interface includes a header with 'SOLOLEARN', navigation links (COURSES, CODE PLAYGROUND, DISCUSS, TOP LEARNERS, BLOG, MY CODES(0)), and a 'SHARE' button. The main area is split into a code editor and an output window. The code editor is set to 'C++' and 'Dark' theme. It contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3
4 void myFunc(int x) {
5     x = 100;
6 }
7
8 int main() {
9     int var = 20;
10    cout << "Before Entering the func Var is " << var;
11    myFunc(var);
12    cout << "\nAfter Entering the func Var is " << var;
13 }
```

The output window on the right displays the following text:

```
Before Entering the func Var is 20
After Entering the func Var is 20
```

**Passing by reference:** This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. So, changes made to the parameter also affect the argument.



The screenshot shows a web browser at code.sololearn.com/251/#cpp. The interface is similar to the previous one, with 'SOLOLEARN' header and navigation links. The code editor is set to 'C++' and 'Dark' theme. It contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3
4 void myFunc(int &x) {
5     *x = 100;
6 }
7
8 int main() {
9     int var = 20;
10    cout << "Before Entering the func Var is " << var;
11    myFunc(&var);
12    cout << "\nAfter Entering the func Var is " << var;
13 }
```

The output window on the right displays the following text:

```
Before Entering the func Var is 20
After Entering the func Var is 100
```