**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **23 June 2020** | **Name:** | **Persis P** |
| **Course:** | **C++ programming** | **USN:** | **4AL17EC069** |
| **Topic:** | **Module 3 : Data types, Arrays and Pointers** | **Semester & Section:** | **6th sem & B sec** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |

# 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.

# Floating Point Numbers

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. You could say that the decimal has the ability to "**float**".   
  
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;

# 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}**.

# Initializing Arrays

If you omit the size of the array, an array just big enough to hold the initialization is created.   
**For example:**int b[] = {11, 45, 62, 70, 88};  
This creates an identical array to the one created in the previous example.  
  
Each element, or member, of the array has an **index**, which pinpoints the element's specific position.  
The array's first member has the index of 0, the second has the index of 1.   
So, for the array **b** that we declared above:   
To access array elements, index the array name by placing the element's index in square brackets following the array name.

# 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;  
  
//Outputs "0x29fee8"

# Pointer Operations

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.  
  
**For example:**

int var = 50;  
int \*p;  
p = &var;  
  
cout << var << endl;  
// Outputs 50 (the value of var)  
  
cout << p << endl;  
// Outputs 0x29fee8 (var's memory location)  
  
cout << **\*p** << endl;  
/\* Outputs 50 (the value of the variable  
stored in the pointer p) \*/

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **23 June 2020** | **Name:** | **Persis P** | |
| **Course:** | **C++ programming** | **USN:** | **4AL17EC069** | |
| **Topic:** | **Module 4 : Functions** | **Semester & Section:** | **6th sem & B sec** | |
| **Github Repository:** |  |  |  | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |

# Functions

A **function**is a group of statements that perform a particular task.  
You may define your own functions in C++.  
  
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.

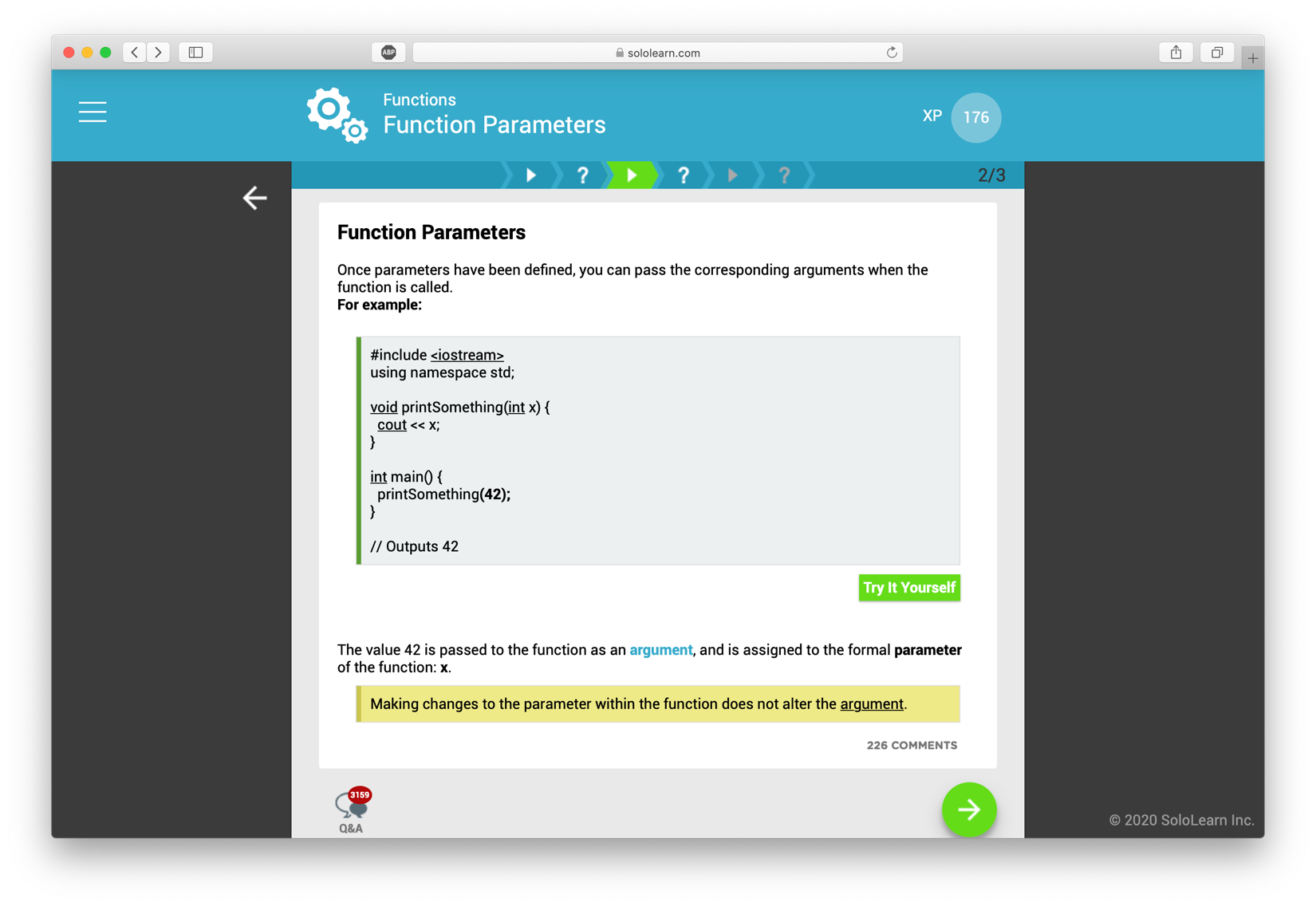
# Defining a Function

As an example, let's define a function that does not return a value, and just prints a line of text to the screen.void printSomething()   
{  
cout << "Hi there!";  
}  
Our function, entitled **printSomething**, returns **void**, and has no parameters.  
Now, we can use our function in **main()**.

int main()   
{  
**printSomething()**;  
  
return 0;  
}

# Function Parameters

For a function to use **arguments**, it must declare formal **parameters**, which are variables that accept the argument's values.  
  
For example: void printSomething(**int x**)   
{  
cout << x;  
}  
This defines a function that takes one **integer**parameter and prints its value.

****