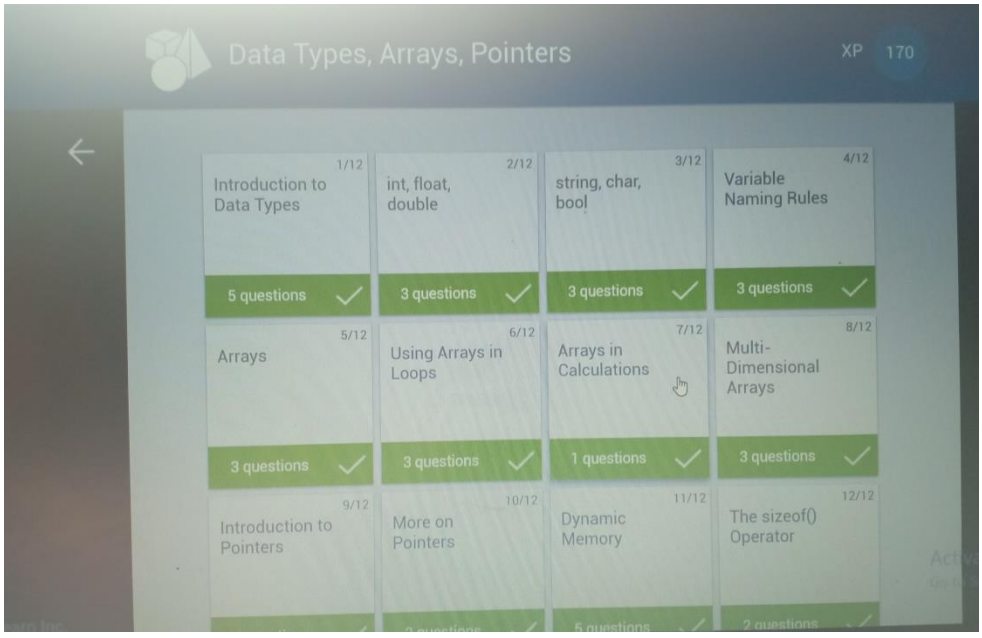
Date:	**23/06/2020**	**Name:**	**Gaganashree P**
Course:	**C++**	**USN:**	**4AL15EC024**
Topic:	**Module 3**	**Semester & Section:**	**8TH SEM & A Section**
Github Repository:	**Gaganashree-P**		

DAILY ASSESSMENT

FORENOON SESSION DETAILS



Report:

C++ Data Types

- While writing program in any language, you need to use various variables to store various information.
- Variables are nothing but reserved memory locations to store values.
- This means that when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

Enumerated Types

- An enumerated type declares an optional type name and a set of zero or more identifiers that can be used as values of the type.
- Each enumerator is a constant whose type is the enumeration.
- Creating an enumeration requires the use of the keyword **enum**. The general form of an enumeration type is −

enum enum-name { list of names } var-list;

Here, the enum-name is the enumeration's type name. The list of names is comma separated.

For example, the following code defines an enumeration of colors called colors and the variable c of type color. Finally, c is assigned the value "blue".

```
enum color { red, green, blue } c;
c = blue;
```

By default, the value of the first name is 0, the second name has the value 1, and the third has the value 2, and so on. But you can give a name, a specific value by adding an initializer. For example, in the following enumeration, **green** will have the value 5.

```
enum color { red, green = 5, blue };
```

Here, **blue** will have a value of 6 because each name will be one greater than the one that precedes it.

C++ Arrays

- C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

- To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows −

```
type arrayName [ arraySize ];
```

This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare

a 10-element array called balance of type double, use this statement −

double balance[10];

Initializing Arrays

You can initialize C++ array elements either one by one or using a single statement as follows −

double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array −

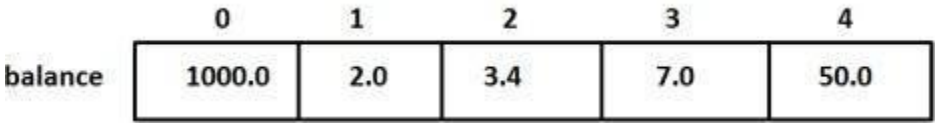
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};

You will create exactly the same array as you did in the previous example.

balance[4] = 50.0;

The above statement assigns element number 5^{th} in the array a value of 50.0. Array with 4^{th} index will be 5^{th}, i.e., last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representaion of the same array we discussed above −



Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example

–

double salary = balance[9];

The above statement will take 10^{th} element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays

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

#include <iomanip>
using std::setw;

int main () {

  int n[ 10 ]; // n is an array of 10 integers

  // initialize elements of array n to 0
  for ( int i = 0; i < 10; i++ ) {
    n[ i ] = i + 100; // set element at location i to i + 100
  }
  cout << "Element" << setw( 13 ) << "Value" << endl;

  // output each array element's value
  for ( int j = 0; j < 10; j++ ) {
    cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;
  }

  return 0;
```

```
}
```

This program makes use of **setw()** function to format the output. When the above code is compiled and executed, it produces the following result −

Element	Value
0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108
9	109

Arrays in C++

Arrays are important to C++ and should need lots of more detail. There are following few important concepts, which should be clear to a C++ programmer −

C++ Pointers

C++ pointers are easy and fun to learn. Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as dynamic memory allocation, cannot be performed without them.

As you know every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory. Consider the following which will print the address of the variables defined −

```
#include <iostream>

using namespace std;
int main () {
   int  var1;
   char var2[10];

   cout << "Address of var1 variable: ";
   cout << &var1 << endl;

   cout << "Address of var2 variable: ";
   cout << &var2 << endl;

   return 0;
}
```

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

```
Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6
```

What are Pointers?

A **pointer** is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is −

```
type *var-name;
```

Here, **type** is the pointer's base type; it must be a valid C++ type and **var-name** is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk

is being used to designate a variable as a pointer. Following are the valid pointer declaration −

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Using Pointers in C++

There are few important operations, which we will do with the pointers very frequently. **(a)** We define a pointer variable. **(b)** Assign the address of a variable to a pointer. **(c)** Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations −

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

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Pointers in C++

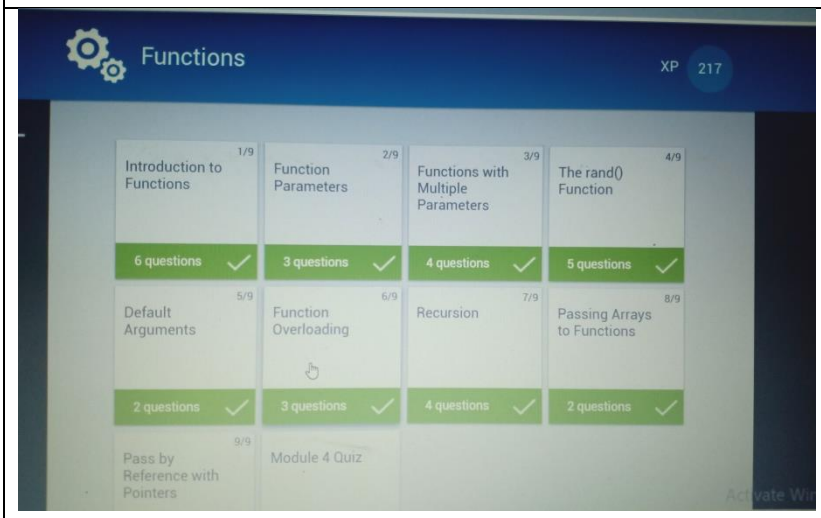
Pointers have many but easy concepts and they are very important to C++ programming. There are following few important pointer concepts which should be

clear to a C++ programmer −

DAILY ASSESSMENT

Date:	**23/06/2020**	**Name:**	**Chesmi B R**
Course:	**C++**	**USN:**	**4AL15EC024**
Topic:	**Module 4:**	**Semester & Section:**	**8TH SEM & A Section**
Github Repository:	**Gaganashree-P**		

AFTERNOON SESSION DETAILS



Report-

C++ 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 usually 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.

The C++ standard library provides numerous built-in functions that your program can call. For example, function **strcat()** to concatenate two strings, function **memcpy()** to copy one memory location to another location and many more functions.

A function is known with various names like a method or a sub-routine or a procedure etc.

Defining a Function

The general form of a C++ function definition is as follows −

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

A C++ function definition 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.

Example

Following is the source code for a function called **max()**. This function takes two parameters num1 and num2 and return the biggest of both −

```
// function returning the max between two numbers

int max(int num1, int num2) {
  // local variable declaration
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts −

return_type function_name(parameter list);

For the above defined function max(), following is the function declaration −

int max(int num1, int num2);

Parameter names are not important in function declaration only their type is required, so following is also valid declaration −

int max(int, int);

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

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 or invoke that function.

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

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

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

// function declaration
int max(int num1, int num2);
```

```
int main () {
   // local variable declaration:
   int a = 100;
   int b = 200;
   int ret;

   // calling a function to get max value.
   ret = max(a, b);
   cout << "Max value is : " << ret << endl;

   return 0;
}

// function returning the max between two numbers
int max(int num1, int num2) {
   // local variable declaration
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

I kept max() function along with main() function and compiled the source code. While running final executable, it would produce the following result −

Max value is : 200

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.

The 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 that arguments can be passed to a function −

Sr.No	Call Type & Description
1	Call by Value This method copies the actual value of an argument into the formal parame function. In this case, changes made to the parameter inside the function effect on the argument.
2	Call by Pointer This method copies the address of an argument into the formal parameter. I function, the address is used to access the actual argument used in the means that changes made to the parameter affect the argument.
3	Call by Reference This method copies the reference of an argument into the formal paramet the function, the reference is used to access the actual argument used in This means that changes made to the parameter affect the argument.

By default, C++ uses **call by value** to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function and above mentioned example while calling max() function used the same method.

Default Values for Parameters

When you define a function, you can specify a default value for each of the last parameters. This value will be used if the corresponding argument is left blank when calling to the function.