Object Oriented Programming (CT-260) Lab 01

Introduction to Object Oriented Programming with C++

Objectives

The objective of this lab is to familiarize students with the basic structure of C++ programs. By the end of this lab, students will be able to create, compile, execute basic input-output programs written in C++ language.

Tools Required

DevC++ IDE / Visual Studio / Visual Code

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Introduction to C++

C++ is a cross-platform language that can be used to create high-performance applications. C++ was developed by Bjarne Stroustrup, as an extension to the C language. C++ gives programmers a high level of control over system resources and memory. The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17.

C++ is very similar to the C Language.

- For the input/output stream we use <iostream> library (in C it was <stdio>).
- For taking input and out we cout and cin (in C it was printf and scanf).
- cout uses insertion (<<) operator.
- cin uses extraction (>>) operator.

Basics of C++ Program

/* Comments can also be written starting with a slash followed by a star, and ending with a star followed by a slash. As you can see, comments written in this way can span more than one line. */ /* Programs should ALWAYS include plenty of comments! */ /* This program prints the table of entered number */

- **The #include Directive**
  The #include directive causes the contents of another file to be inserted into the program Preprocessor directives are not C++ statements and do not require semicolons at the end
- **Using namespace std;**
  The names cout and endl belong to the std namespace. They can be referenced via fully qualified name std::cout and std::endl, or simply as cout and endl with a "using namespace std;" statement.
- **return 0;**
  The return value of 0 indicates normal termination; while non-zero (typically 1) indicates abnormal termination. C++ compiler will automatically insert a "return 0;" at the end of the the main () function, thus, this statement can be omitted.
- **Output using cout**
  - Cout is an object
  - Corresponds to standard output stream
  - << is called insertion or input operator
- **Input With cin**
  - Cin is an object
  - Corresponds to standard input stream
  - >> is called extraction or get from operator

Example of a Basic C++ Program:

```
#include <iostream>
using namespace std;
int main() {
    char a;
    int num;
    cout << "Enter a character and an integer: ";
    cin >> a >> num;
    cout << "Character: " << a << endl;
    cout << "Number: " << num;
    return 0;
 }
```

Output:

```
Enter a character and an integer: X
20
Character: X
Number: 20
```

Data Types in C++

C++ supports the following basic data types:

boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values
int	2 or 4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits

String Data Type:

Apart from these basic types, C++ also support 'string' data type to store multiple characters. i.e., a series of characters or text. Strings may only be used if an additional header file, the <string> library, is included in the source code.

```
#include<string>
```

In latest version of the C++ compliers <string> library has been included in <iostream> library, and you don't need to include it again.

String values must be enclosed in double quotations.

```
string introduction = "Welcome to OOP Lab";
cout << introduction;
```

Basic C++ String Program

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string greeting = "Hello";
  cout << greeting;
  return 0;
}
```

Comments in C++

A comment is text that is ignored by the compiler yet is beneficial to programmers. Normally, comments are intended to mark code for future reference. They are treated as white space by the compiler. C++ supports two types of comments

- **Single line comment** – The // (two slashes) characters, followed by any character sequence. This type of comment is terminated by a new line that is not immediately followed by a backslash.
- **Multiple Line Comment** – The characters /* (slash, asterisk), followed by any sequence of characters (including new lines), followed by the characters */.

Pointers

A Pointer is a variable whose content is a memory address.

- To declare a single pointer variable, you need to specify the data type, an asterisk symbol (*) and the name of the pointer variable.

```
dataType *ptrName;
```

- Following is the declaration of a Pointer variable.

```
int *ptr;
```

  - DataType: Integer
  - Name: ptr
- Pointer variable holds the memory address of the variable which is of same data type (integer in this case).
- To assign the memory address of any variable to the pointer variable we use **Address of Operator (&).**

```
int intVar = 5;
ptr = &intVar;
```

  - In this statement **ptr** now holds the memory address of an integer variable **'intVar'**.
- To access the value at the memory address (currently stored) in the variable we use Dereferencing Operator (*).
  - Do not confuse this with the symbol used for the declaration of a pointer.

```
int intVar2 = *ptr;
```

  - In this statement another integer variable **'intVar2'** is now initialized with the value at the memory address which is stored in **ptr** (that is the value of intVar).

Example Code for Pointers:

```
#include <iostream>
using namespace std;
int main( ){
      int *p;
      int x = 37;
      cout << "Line 1: x = " << x << endl; //Line 1
      p = &x; //Line 2
      //Line 3
      cout << "Line 3: *p = " << *p << ", x = " << x << endl;
      *p = 58; //Line 4
      //Line 5
      cout << "Line 5: *p = " << *p << ", x = " << x << endl;
      cout << "Line 6: Address of p = " << &p << endl; //Line 6
      cout << "Line 7: Value of p = " << p << endl; //Line 7
      cout << "Line 8: Value of the memory location " << "pointed to
      by *p = " << *p << endl; //Line 8
      cout << "Line 9: Address of x = " << &x << endl; //Line 9
      cout << "Line 10: Value of x = " << x << endl; //Line 10
      return 0;
}
```

Output:

```
Line 1: x = 37
Line 3: *p = 37, x = 37
Line 5: *p = 58, x = 58
Line 6: Address of p = 0x6ffe08
Line 7: Value of p = 0x6ffe04
Line 8: Value of the memory location pointed to by *p = 58
Line 9: Address of x = 0x6ffe04
Line 10: Value of x = 58
```

Dynamic Memory Allocation

If we need a variable amount of memory that can only be determined during runtime, then we use dynamic memory which can be implemented by using new and delete operators. For normal variables like "int a", "char str[10]", etc, memory is automatically allocated and de-allocated on stack. For dynamically allocated memory like "int *p = new int[10]", it is programmers responsibility to de-allocate memory when no longer needed.

new Operator

Variables created during the program execution are called dynamic variables. To create a dynamic variable, we use new operator.

```
new dataType;          // to allocate a single variable
new dataType [ size];          // to allocate an array of variables.
```

- The new operator allocates the memory of a designated type.
- It returns a pointer to the allocated memory.

Following is the declaration of a dynamic variable.

```
int *p = new int;
char *cArray = new char[5];
```

- Line 01: creates a single variable of integer type whose address in stored in p.
- Line 02: Creates an array of 5 characters whose initial address is stored in cArray.

Example of new Operator:

```
// Pointer initialized with NULL, Then request memory for the variable
int *p = NULL;
p = new int;
```

OR

```
// Combine declaration of pointer and their assignment
int *p = new int;
```

We can also initialize the memory using new operator:

```
pointer-variable = new data-type(value);
```

Example:

```
int *p = new int(25);
float *q = new float(75.25);
```

new operator is also used to allocate a block (an array) of memory of type data-type.
Note: size(a variable) specifies the number of elements in an array.

```
pointer-variable = new data-type[size];
```

Example Code:

```
#include<iostream>
using namespace std;
int main(){
      int* intPtr;
      char* charArray;
      int Size;
      intPtr = new int; // allocating memory to single variable
      cout << "Enter an Integer Value: ";
      cin >> *intPtr;
      cout << "Enter the size of the Character Array : ";
      cin >> Size;
      charArray = new char[Size];//allocating memory to array
      for (int i = 0; i < Size; i++)
            cin >> charArray[i];
      for (int i = 0; i < Size; i++)
            cout << charArray[i];
      return 0;
}
```

Output:

```
Enter an Integer Value: 2
Enter the size of the Character Array : 2
a b
ab
```

delete Operator

Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

```
delete ptrVar;          //to deallocate single dynamic variable
delete[] ptrArray;      //to deallocate dynamically created array
```

delete operator is used to free the memory which is dynamically allocated using new operator.

Function

A function is a self-contained program segment that carries out a specific well-defined task. In C++, every program contains one or more functions which can be invoked from other parts of a program, if required.

When passing parameters to a function there are ways to do that.

1. Value Parameters (Pass by Value).
2. Reference Parameters (Pass by Reference).

Value Parameters (Pass by Value)

When passing value parameters in a function, the parameter is copied into the corresponding formal parameter. There is no connection between the actual and formal parameter values, this means that these parameters cannot be used to pass the result back to the calling function.

Example Code:

```
#include <iostream>
using namespace std;
void funcValueParam (int num);
int main (){
      int number = 6; //Line 1
      cout << "Line 2: Before calling the function " <<
      "funcValueParam, number = " << number << endl; //Line 2
      funcValueParam(number); //Line 3
      cout << "Line 4: After calling the function "    <<
      "funcValueParam, number = " << number << endl; //Line 4
      return 0;
}
void funcValueParam (int num){
      cout << "Line 5: In the function funcValueParam, " <<
      "before changing, num = " << num << endl; //Line 5
      num = 15; //Line 6
      Value Parameters | 367
      cout << "Line 7: In the function funcValueParam, " <<
      "after changing, num = " << num << endl; //Line 7
}
```

Output:

```
Line 2: Before calling the function funcValueParam, number = 6
Line 5: In the function funcValueParam, before changing, num = 6
Line 7: In the function funcValueParam, after changing, num = 15
Line 4: After calling the function funcValueParam, number = 6
```

Reference Parameters (Pass by Reference)

When a reference parameter is passed in a function, it receives the address (memory location) of the actual parameter. Reference parameters can change the value of the actual parameter.

Reference parameters are useful in following situations.

1- When the value of the actual parameter needs to be changed.
2- When you want to return more than one value from a function.
3- When passing the address would save memory space and time relative to copying a large amount of data.

Example Code:

```
//This program reads a course score and prints the
//associated course grade.
#include <iostream>
using namespace std;
void getScore (int& score);
void printGrade (int score);
int main (){
      int courseScore;
      cout << "Line 1: Based on the course score, \n" << " this
      program computes the " << "course grade." << endl; //Line 1
      getScore(courseScore); //Line 2
      printGrade(courseScore); //Line 3
return 0;  }
void getScore (int& score){
      cout << "Line 4: Enter course score: "; //Line 4
      cin >> score; //Line 5
      cout << endl << "Line 6: Course score is "<< score << endl;
      //Line 6
```

```
}
void printGrade (int cScore){
      cout << "Line 7: Your grade for the course is "; //Line 7
      if (cScore >= 90) //Line 8
            cout << "A." << endl;
      else if (cScore >= 80)
            cout << "B." << endl;
      else if (cScore >= 70)
            cout << "C." << endl;
      else if (cScore >= 60)
            cout << "D." << endl;
      else
            cout << "F." << endl;
}
```

Output:

```
Line 1: Based on the course score, this program computes the course grade.
Line 4: Enter course score: 85
Line 6: Course score is 85
Line 7: Your grade for the course is B.
```

Static and Automatic Variables in C++

A variable for which memory is allocated at block entry and de-allocated at block exit is called an automatic variable. - A variable for which memory remains allocated as long as the program executes is called a static variable. Global variables are static variables by default and variables declared in a block are automatic variables. Syntax for declaring a static variable is:

```
static datatype identifier;
```

Example Code for Static Variables:

```
//Program: Static and automatic variables
#include <iostream>
using namespace std;
void test ();
int main (){
      int count;
      for (count = 1; count <= 5; count++)
            test ();
      return 0;
}
void test (){
      static int x = 0;
      int y = 10;
      x = x + 2;
      y = y + 1;
      cout << "Inside test x = " << x << " and y = "<< y << endl;
}
```

Output:

```
Inside test x = 2 and y = 11
Inside test x = 4 and y = 11
Inside test x = 6 and y = 11
Inside test x = 8 and y = 11
Inside test x = 10 and y = 11
```

Exercise

1. Write a program that take input of your roll number along with the marks obtained in five subjects and display the total marks obtained and the percentage.

2. Write a program to swap three numbers entered by a user using pointers.

3. Write a program to convert temp from Fahrenheit to Celsius unit using equation C=(F-32)/1.8

4. Using 2-D arrays, write a program that allows the user to input two, 3x3 matrices. Write a function for adding two matrices. Write another function for multiplying the two matrices.

5. Write a program to find Surface area and volume of a sphere using functions.

6. Write a program to help a bank create its withdrawal system. Your program should allow the user to input their account type. Account types are: savings, current. Following business rules apply when withdrawing from a certain account:
   - **Savings:**
     User must provide the savings account number and code 'S' (for savings). When withdrawing from a savings account, users need to pay a set 2% of the money that they withdraw. If the amount of money withdrawn is over 50,000, then a 5% tax will be deducted. The money deducted shall be from the remaining money in the account.
   - **Current:**
     User must provide the current account number and code „C" (for current). When withdrawing from a current account, users need to pay a withdrawal fee of 100. If the amount of money withdrawn is over 50,000, then a 5% tax will be deducted. The money deducted shall be from the remaining money in the account.

   Assume all users have the 200,000 in their accounts, and cannot withdraw more than 100,000 at a time.