JIMMA UNIVERSITY

Jimma Institute of Technology

Faculty Of Computing And Informatics

Department of Information Science

Fundamental of Computer programming 1

Individual Assignment

By: Honelign Daniel

Id : RU 2102/14

April , 2023

Jimma, Ethiopia

Computer Programming 1 Assignment

1) Demonstrate Unit 2 (C++ Programming Basics.)

Programs in C++ are simple and well structured. There are some basic components in every C++ program that we write. There are some special lines called Preprocessor Directives that are interpreted even before the compilation of the program begins. Programmers can give comments within their programs to make the statements understandable to novice users. The Standard Library of C++ consists of classes and functions.

These classes and functions are written in the core language. They even form part of the C++ ISO Standard.

Data Types in C++ are used by users to declare variables. Primary Data Types in C++ are Integer, Character, Boolean, Floating Point etc. C++ uses Datatype Modifiers to modify the length of data that a particular datatype can hold. Datatype Modifiers are used with the built-in data types. In C++, data can be manipulated according to the choice of display of the programmer. To format the output, special operators called Manipulators are used.

The rest of the unit is organized as follows.

1.1 Basic C++ Program Construction

Let us start with a simple program that can help us to understand the fundamental components of a C++ program. The following program prints the text as shown in the Output below.

```
1. // First C++ Program
2. #include <iostream>
3.
4. using namespace std;
5. int main ()
6. {
7. cout << "Hello World! ";
8. cout << " This is my first C++ Program ";
9. return 0;
10. }
```

OUTPUT: Hello world! This is my first C++ Program.

Let us analyze the program one statement at a time.

1st Line: // First C++ Program

The first statement in the program is just a comment inserted by the programmer within the program. A comment line is indicated by two leading slash signs as shown here. Comments have no effect on the behavior of the program. This is known as single line comment. Multi line comment is shown as below:

/* This is my first program.

I will learn how to write C++ program and how to compile the same*/

2nd Line: #include <iostream>

Lines that begin with hash signs (#) are called preprocessor directives. These special lines are interpreted even before the program compilation starts. The preprocessor directive #include <iostream> informs the preprocessor to include the header iostream, which is a part of the standard C++ code. The header iostream permits the accomplishment of the standard input and output operations, for e.g., in this case it prints the output on the computer screen.

3rd Line: A blank line.

There is no effect of blank lines on the program. It just makes the program easy to read.

4th Line: using namespace std;

Suppose the code that you are writing has the function abc() in it. Similarly, there is another library that has the same function abc(). The compiler is not aware that which abc() function you are referring to in your

code.

A namespace solves this problem and it is used as an extra information to distinguish between similar classes, functions, variables etc. that have the same name but are available in different libraries. Namespace basically defines the scope.

In C++, namespaces are used to define a scope. They make it possible to group global classes, objects and/or functions into a single group. Through using namespace std;, C++ compiler is instructed to use the standard C++ library. If this instruction is not used, then each time you use a standard C++ function or entity, you have to use std::.

Like without using namespace same program will be as below:

1. // First program in C++

```
2. #include <iostream>
3.
4. int main ()
5. {
6. std::cout << "Hello World! ";
7. std::cout << " This is my first C++ Program ";
8. return 0;
9. }
```

5th Line: int main ()

This line initiates the declaration of a function. A function is nothing but a collection of statements that is given a name. In this case the function is given the name “main”. The declaration of a function is preceded by a type (int), followed by a name (main) and a pair of parentheses (()). Optionally parameters can be included within the parentheses.

The main function is a special function that is called automatically. The code in any other function is executed only if that function is called from the main function (directly or indirectly). For all C++ programs, the execution begins with the main function. It does not matter where the main function is located in the program. 6th and 10th Line: { and } The body of a function is enclosed within an open brace ({) and a closing brace (}). It is defined within these braces that what actually happens when the function is called. In the above program, the body of the main function is enclosed within the open brace at line 6 and the closing brace at line 10.

7th Line: cout << "Hello World!";

8th Line: cout << " This is my first C++ Program ";

Lines 7 and 8 are C++ statements. Statements are executed in the exact order in which they appear inside the body of a function.

The statements consist of three parts: The first part, i.e. cout, recognizes the standard character output device (generally the computer screen). The second part, i.e. insertion operator (<<), specifies that anything that follows the insertion operator is inserted into cout. The third part, i.e. a sentence within quotes (for e.g “Hello World!”) is the content that is to be inserted into the standard output.

The reader should notice that the statement ends with a semicolon (;). In fact, all the statements in C++ should end with a semicolon character. A common mistake by programmers is leaving the semicolon at the end of statements.

9th Line: return 0;

The task of the return statement is to send a status report to the OS about the program execution. It informs the OS whether the program has executed properly or not.

Program: print the given lines on computer screen.

I am 18 years old.

I am student of Polytechnic.

Solution:

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
int age;
age=18;
cout<<" I am "<<age<<" years old.\n";
cout<<" I am student of Polytechnic.\n";
return 0;
}
```

Program: write first program using class

```
#include <iostream>
using namespace std;
class Test //class declaration start with keyword class and then name of the class
{
int i; //data variable
public:
void show() //Member Function
{
cout<<"Inside Member Function";
}
```

```
}; // end of Class
int main()
{
Test obj; // Creating object of Abc class
obj.show(); // Use of class object to call member function
}
```

Output:

Inside Member Function

How to define a class in C++?

To define a class in C++, we use the keyword class, which is followed by the class name. The class body is enclosed within the opening and closing curly brackets. The closing curly bracket is followed by a semicolon.

```
class ClassName
{
// data
// functions
};
```

How to create object in C++?

Syntax to define the object in main function.

```
className cn; //cn is the object
```

This is very similar to declaration of variable in C / C++.

How to access data member and member function in C++?

The dot operator (.) is used to access the data members and member functions. For e.g.,

```
obj2.func1();
```

The above statement will call the function func1() inside the Test class for object obj2.

To access the data member, the following statement is used: obj1.data1 = 7.3;

The reader should note that if he wants to access private members, then it can be done from within the class only.

Program: usage of classes and objects in C++

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

```
class Sample
{
private:
int data_1;
float data_2;
public:
void insertIntegerNumber(int d_1)
{
data_1 = d_1;
cout << "Number: " << data_1;
}
float insertFloatingNumber()
{
cout << "\nEnter data: ";
cin >> data_2;
return data_2;
}
};
int main()
{
Sample obj1, obj2;
float f;
obj1.insertIntegerNumber(15);
f = obj2.insertFloatingNumber();
cout << "You entered " << f;
return 0;
}
```

Output:

```
Number: 15
Enter data: 23.3
You entered 23.
```

2) Exercise from chapter 2

Chapter 2 question 1

1. This program calculates the product of three integers.
2. Int x, y, z, result;
3. Please enter three integers: (prompt)
   Cin >> x >> y >> z;
4. Result = x * y * z;
5. Cout << "The product is " << result << endl;
6. Return 0; //assuming successful termination

Chapter 2 question 2

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

Int main() {
   Int num1, num2, sum, diff, prod;
   Double div;

   Cout << "Enter two integers: ";
   Cin >> num1 >> num2;

   Sum = num1 + num2;
   Diff = num1 – num2;
```

```
    Prod = num1 * num2;
    Div = num1 / num2;

    Cout << “Sum = “ << sum << endl;
    Cout << “Difference = “ << diff << endl;
    Cout << “Product = “ << prod << endl;
    Cout << “Division = “ << div << endl;

    If (num1 > num2) {
        Cout << num1 << “ is greater than “ << num2 << endl;
        Cout << num2 << “ is smaller than “ << num1 << endl;
    } else if (num1 < num2) {
        Cout << num2 << “ is greater than “ << num1 << endl;
        Cout << num1 << “ is smaller than “ << num2 << endl;
    } else {
        Cout << “Both numbers are equal” << endl;
    }

    Return 0;
}
```

Chapter 2 question 3

```
#include <iostream>
#include <cmath>
Using namespace std;

Int main() {
   Double radius, circumference;
   Const double PI = 3.14159;

   Cout << “Enter the radius of the circle: “;
   Cin >> radius;

   Circumference = 2 * PI * radius;

   Cout << “The circumference of the circle is: “ << circumference << endl;

   Return 0;
}
```

Chapter 2 question 4

```
#include <iostream>
#include <cmath>
Using namespace std;

Int main()
{
   Double a, b, c, root1, root2, discriminant;
   Cout << "Enter the coefficients a, b and c: ";
   Cin >> a >> b >> c;
   Discriminant = b * b – 4 * a * c;
   If (discriminant > 0)
   {
      Root1 = (-b + sqrt(discriminant)) / (2 * a);
      Root2 = (-b – sqrt(discriminant)) / (2 * a);
      Cout << "Roots are real and different." << endl;
      Cout << "Root 1 = " << root1 << endl;
      Cout << "Root 2 = " << root2 << endl;
   }
   Else if (discriminant == 0)
   {
      Root1 = root2 = -b / (2 * a);
      Cout << "Roots are real and same." << endl;
      Cout << "Root 1 = Root 2 = " << root1 << endl;
```

```
    }
    Else
    {
        Double realPart = -b / (2 * a);
        Double imaginaryPart = sqrt(-discriminant) / (2 * a);

        Cout << "Roots are complex and different." << endl;
        Cout << "Root 1 = " << realPart << "+" << imaginaryPart << "I" << endl;
        Cout << "Root 2 = " << realPart << "-" << imaginaryPart << "I" << endl;
    }
    Return 0;
}
```

3) Demonstrate chapter 3

Program Control Statements

This chapter discusses C++'s rich and varied program control statements. C and C++ categorize statements into these groups:

1- Selection: consists of ***if*** and ***switch*** statements.
2- Iteration (Loops): consists of ***while*** and ***for*** statements.
3- Jump: consists of ***goto*** , ***break***, ***continuo***, and ***return*** switch statements.
4- Label: consists of ***case*** and ***default*** statements.
5- Expression
6- Block

The selection statements are the **if** and **switch**. The term *conditional statement* is often used in place of selection statement. The iteration statements are **while**, **for**, and **do/while**. These are also commonly called *loop* statements. The jump statements are **break**, **continue**, **goto**, and **return**. The label statements include the **case** and **default** statements (discussed along with the **switch** statement) and the label statement itself (discussed with **goto**). Expression statements are statements composed of a valid expression. Block statements are simply blocks of code. (A block begins with a **{** and ends with a **}**.) Block statements are also referred to as *compound statements.*

1-Selection Statements

C++ language is supported by two types of selection statements: **if** and **switch**.

A- if statements

The general form of an if statement is

if (condition) statement;

If *condition* is true, then *statement* is executed; otherwise, *statement* is skipped. After the if Statement has been executed, control passes to the next statement. A *condition* is an expression or a declaration with initialization that selects flow of control. Where *statement* may consist of a single statement, a block of statements, or nothing (in the case of empty statements). The general form of the **if** using a block of statements is

if(condition) {
statement sequence
}

Ex: Write a program in C++ language to print on the screen of computer the sentence (I am Iraqi) when the number enters from keyboard is equal 100.

Ans:

```
#include<iostream.h>
main()
{
int a;
cout<<" Enter integer number:";
cin>>a;
if(a==100) cout<<" I am Iraqi";
}
```

Q: Rewrite the program for the number is more than or equal to 100.

 if…else statements

The general form of the **if ..else** statement is

if(condition) statement;
else statement;

If *condition* is true, then *statement1* is executed and *statement2* is skipped; if *condition* is false, then *statement1* is skipped and *statement2* is executed. After the if-else statement has been executed, control passes to the next statement.

Ex: Write a program in C++ language to print on the screen of computer the word (OKEY) when the character entering from keyboard is 'Y' or 'y' otherwise print (NO).

Ans:

```
#include<iostream.h>
main()
{
char a;
cout<<" Enter the character: ";
cin>>a;
if('Y'==a||'y'==a) cout<<"OKEY";
else cout<<"NO";}
```

Q: what is the difference in the result if you change the if instruction by

```
if(a=='Y'||a=='y')cout<<"OKEY";
```

 The if-else-if Ladder

A common programming construct is the *if-else-if ladder,* sometimes called the *if-else-if*

staircase because of its appearance. Its general form is
if (condition)statement 1;
else if (condition)statement 2;
else if (condition)statement 3;
...
else statement N;
The conditions are evaluated from the top downward. As soon as a true condition is found, the statement associated with it is executed and the rest of the ladder is bypassed. If none of the conditions are true, the final **else** is executed. The final **else** often acts as a default condition; that is, if all other conditional tests fail, the last **else** statement is performed. If the final **else** is not present, then no action takes place if all other conditions are false.
Ex: Write a program in C++ language to obtain and print on the screen of computer the maximum number from three different float numbers.
Ans:

```
#include<iostream.h>
main()
{
float a, b, c;
cout<<"Enter Three different Float Numbers :";
cin>>a>>b>>c;
if (a>b&&a>c)cout<<" The Maximum Number is "<<a;
else if(b>c)cout<<" The Maximum Number is "<<b;
else cout<<" The Maximum Number is "<<c;
}
```

Q: Modify the program to consist of all probability.
Q: Try to obtain the middle number from three float numbers.
B- switch statement
The switch statement is a multiway conditional statement generalizing the if-else statement. The general form of the switch statement is given by:
switch(expression) {
case constant1:statement sequence 1: break;
case constant2:statement sequence 2; break;
case constant3:statement sequence 3; break;
...
default: statement sequence;
}
Technically, the **break** statements inside the **switch** statement are optional. They terminate the statement sequence associated with each constant. If the **break** statement is omitted, execution continues on into the next **case**'s statements until either a **break** or the end of the **switch** is reached. You can think of the **case**s as labels. Execution starts at the label that matches and continues until a **break** statement is found, or the **switch** ends. Each statement sequence may be from one to several statements long. The **default** portion is optional. Both *expression* and the **case** constants must be integral types.
Note: the ***break*** statement is most importantly used within a switch statement, which can select among several cases. To interrupt the normal flow of control within a loop, the programmer can use the special statement:

break;

Ex: Write a program in C++ language to obtain and print on the screen of computer the (Next for N, Back for B, and Cancel for C) if the characters N. B, and C entering from keyboard.

Ans:

```
#include<iostream.h>
main()
{
char d;
cout<<"Enter the character: ";
cin>>d;
switch(d){
case ‘C’:cout<<" CANCEL ";break;
case ‘B’:cout<<" BACK";break;
case ‘N’:cout<<"NEXT";break;
default: cout<<"ERROR";
}
}
```

2- Iteration Statements

C++ language is supported by two types of iteration (Loops) statements: **while** and **for**

A- while statement

The first iteration statement in C++ is the **while** loop. Its general form is

while(condition) statement;

where *statement* is either an empty statement, a single statement, or a block of statements. The *condition* may be any expression, and true is any nonzero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line of code immediately following the loop.

Ex: Write a program in C++ language to obtain and print on the screen of computer the cubic of the number that entering from keyboard and terminate it with the number is less than –56.5.

Ans:

```
#include<iostream.h>
main()
{
float d;
cout<<"Enter the float number to obtain the cubic, terminate with less -56.5\n\t";
cin>>d;
while(d>-56.5){
cout<< "The cubic of "<<d<<"is equal "<<(d*d*d)<<endl<<'\t';
cin>>d;
}
cout<<"The number is less than -56.5";
}
```

Q: Rewrite the above program to obtain the root of the number and terminate with zero.

Q: Write a program in C++ language to print on the screen of computer the number entering from keyboard is accepted to divide by 5 without carry, and the result. Otherwise is terminated.

 do-while statement

Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do-while** loop checks its condition at the bottom of the loop. This means that a **do-while** loop always executes at least once. The general form of the **do-while** loop is

do {
statement sequence
} while(condition);

first, statement is executed, and then condition is evaluated. If it is true, control passes back to the beginning of the do statement, and the process repeat itself. When the value of condition is false, control passes to the next statement.

Ex: Write a program in C++ language to obtain and print on the screen of computer the factorial of the number entering from keyboard using do…while statement.

Ans:

```
#include<iostream.h>
main()
{
int n,f=1;
cout<<"Enter the positive integer number ";
cin>>n;
cout<<"The factorial of "<<n<<" is ";
do{
f*=n;
n--;
}while(n>1);
cout<<f<<endl;
}
```

Q: Write a program in C++ language to print on the screen of computer the counter from 2.1 to 13.1 increments by 0.2 using do…while statement.

Q: Write a program in C++ language to find and print on the screen of computer the times of divided by 2 without fraction for number enter from keyboard using do…while statement.

B- for statement

The general design of the **for** loop is reflected in some form or another in all procedural Programming languages. However, in C++, it provides unexpected flexibility and power. The general form of the **for** statement is

for(initialization; condition; update) statement;

The **for** loop allows many variants, but there are three main parts:

1. The *initialization* is usually an assignment statement that sets the loop control variable.
2. The *condition* is a relational expression that determines when the loop exits.
3. The *update* defines how the loop control variable changes each time the loop is repeated.

These three major sections must be separated by semicolons. The **for** loop continues to execute as long as the condition is true. Once the condition becomes false, program

execution resumes on the statement following the **for**.
Ex: Write a program in C++ language to print on the screen of computer the numbers from 1 to 50 increments by1.
Ans:

```
#include<iostream.h>
main()
{
for(int I=1;I<=100;I++)cout<<I<<'\t';
}
```

Result:

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
```

Q: Rewrite the above program for the increments by 2.5.
Q: Write a program in C++ language to print on the screen of computer the numbers obtain from 2i where the i is integer number enter from keyboard.
Q: Write a program in C++ language to execute and print counter. However, the beginning and ending and update number entering from keyboard. for
a- Octal numbers.
b- Hexadecimal numbers.
c- Binary numbers.

 The Infinite Loop

One of the most interesting uses of the **for** loop is to create an infinite loop. Since none of the three expressions that form the **for** loop are required, you can make an endless loop by leaving the conditional expression empty, as here:

for(;;) statement;

Ex: Write a program in C++ language to print the message "you are in outside of loop" when you enter the right password, otherwise print "you are inside the loop" take one character as the password.
Ans:

```
#include<iostream.h>
main()
{
char f;
for(;;){
cout<<" Enter The Password:";
cin>>f;
if('D'==f)break;
cout<<"you are in loop:\n";
}
cout<<"you are in outside of loop";
}
```

3- Jump Group

A- goto statement:
The goto keyword causes program execution to jump to the label specified in the goto statement. The general form of goto is
goto label;
.
.
.
label: statement;
B- return statement:
The return statement forces a return from a function and can be used to transfer a value back to the calling routine. It has these two forms:
return;
Or ***return expression;***
C- continue Statement:
The continue statement works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between. The general form of continue is:
continue;
D- break Statement:
The break statement has two uses. You can use it to terminate a case in the switch statement and can also use it to force immediate termination of a loop, bypassing the normal loop conditional test. . The general form of break is:
break;
Ex: Write a program in C++ language to obtain and print on the screen of computer , when the number input from keyboard prove that:

First: Accept divided by 2 without carry to continue.
Second: Accept divided by 3 without carry to break and print " outside of loop".
Third: Otherwise print " bottom of loop".

Ans:
```
#include<iostream.h>
main()
{
int n;
for(;;){
cout<<"Enter integer number";
cin>>n;
if(n%2==0)continue;
if(n%3==0)break;
cout<<"bottom of loop";
}
cout<<"outside of loop";
}
```

Ex: Write a program in C++ language to obtain and print on the screen of computer ,

method of using goto and return statements.

Ans:

```
#include<iostream.h>
main()
{
int n=5;
s1: cout<<" Now at step 1 with n="<<n<<endl;
--n;
if(n<2)return 0;
s2: cout << " Now at step 2 with n="<< n<<endl;
if(n<7) goto s4;
s3: cout<<" Now at step 3 with n="<<n<<endl;
if(n%2==0) goto s1;
s4: cout<<" Now at step 4 with n="<<n<<endl;
n-=2;
if(n>4) goto s1;
else goto s3;
}
```

result:

```
Now at step 1 with n=5
Now at step 2 with n=4
Now at step 4 with n=4
Now at step 3 with n=2
Now at step 1 with n=2
```