Iqra National University Peshawar Pakistan

Department of Computer Science

Spring Semester, Final Term Exam, June 2020

Paper:	**Programming Fundamentals**	Student Name:	**Saad Ali**
Program:	**BS(SE)**	Student ID:	**16880**
Teacher Name:	**Dr. Fazal-e-Malik**		

Q#1. (a) What is the purpose of ***if statement***? Discuss its two different forms with examples.

if statement:

"***if***" is a keyword in C++ language. ***"if"*** statement is a decision-making statement. It is the simplest form of selection constructs. It is used to execute or skip a statement or set of statements by checking a **condition.** The condition is given as a relational expression. If the condition is true, the statement or set of statements after **if** statement is executed. if the condition is false, the statement or set of statements after **if** statement is not executed

Syntax:

The syntax of if statement is as follow:

```
if(condition)

statement(s);
```

Forms of "if" statement:

There are different forms of "if" statement which are as follow:

i) if-else statement

ii) nested if

1. if-else statement:

if-else statement is another type of if statement. It executes one block of statement(s) when the condition is true and the other when it is false. In any situation, one block is executed and the other is skipped. In ***if else*** statement:

Syntax:

```
 if(condition)
  {
   statement 1;
   statement 2;
}
else
{
statement 1;
statement 2;
.
.
statement N;
}
```

Example:

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


int main()
{
  int a;
  cout<<"Enter a number"<<endl;
  cin>>a;
  if(a%2==0)
  cout<<a<<" is even"<<endl;
  else
```

```
    cout<<a<<" is odd"<<endl;

    return(0);

}
```

2. Nested if:

An **if** statement within an if statement is called **nested if** statement. In nested structure, the control enters into the inner if only when the outer condition is true. Only one block of statements are executed and the remaining blocks are skipped automatically.

Syntax:

The syntax of nested If is as follow:

```
 if(condition)

     if(condition)

     {

       statement(s);

     }

    else

   {

    statement(s);

  }

else

{

statement(s);

}
```

Example:

```
#include<iostream>

using namespace std;


int main()
```

```
{
   int a,b,c;
   cout<<"Enter first numbers"<<endl;
   cin>>a;
   cout<<"Enter second number"<<endl;
   cin>>b;
   cout<<"Enter third number"<<endl;
   cin>>c;
   if(a<b)
    if(a<c)
      cout<<a<<" is smallest"<<endl;
    else
      cout<<c<<" is smallest"<<endl;
   else
     if(b<c)
      cout<<b<<" is smallest"<<endl;
     else
      cout<<c<<" is smallest"<<endl;
   return(0);
}
```

Q1(b) Write a C++ program to read two numbers from keyboard and then find the LARGEST number of them.

Code:

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

int main()
```

```
{
   int a,b;
   cout<<"Enter first number"<<endl;
   cin>>a;
   cout<<"Enter second number"<<endl;
   cin>>b;

   if(a>b)
    cout<<a<<" is the largest number"<<endl;
    if(b>a)
    cout<<b<<" is the largest number"<<endl;
   return(0);
}
```

Output:

```
Enter first number
3
Enter second number
6
6 is the largest number
```

****

Q#2 (a) What are the logical Operators? Explain them.

Logical Operators:

Logical Operator are used to evaluate compound conditions. There are three logical operators in C++.

1. AND (&&)
2. OR (||)
3. NOT (!)

1. AND Operator (&&):

The symbol used for AND operator is (&&). It is used to evaluate two conditions. It produces true result if both conditions are true. It produces false result if any one condition is false.

Condition 1	**Operator**	**Condition 2**	**Result**
False	&&	False	False
False	&&	True	False
True	&&	False	False
True	&&	True	True

Example

Suppose we have two variable A=100 and B=50. The compound condition (A>10) && (B>10) is true. It contains two conditions and both are true. So the whole compound condition is also true.

The compound condition (A>50) && (B>50) is false. It contains two conditions. One condition (A>50) is true and second condition (B>50) is false. So the whole compound condition is false.

2. OR Operator (||):

The symbol used for OR operator is (||). It is used to evaluate two conditions. It gives true result if either condition is true. It gives false result if both conditions are false.

Condition 1	**Operator**	**Condition 2**	**Result**
False	||	False	False
False	||	True	True
True	||	False	True
True	||	True	True

Example

Suppose we have two variable A=100 and B=50. The compound condition (A>10) || (B>10) is true. So the whole compound condition is also true. The compound condition (A>500) || (B>500) is false because both conditions are false.

3. NOT Operator (!):

The symbol used for NOT operator is (!). it is used to reverse the result of a condition. It gives true result if the condition is false. It gives false result if the condition is true.

Operator	**Condition**	**Result**
!	True	False
!	false	True

Example

Suppose we have two variables A=100 and B=50. The condition !(A==B) true. The result of (A==B) is false but NOT operator convert it into true.

Q#2(b) Write a C++ program to get Temperature in Fahrenheit F and then find the Atmosphere according to the below rules;

- If the temperature F is above 40 degree Fahrenheit then display ……………………………Very Hot.
- If the temperature F is between 35 & 40 degree Fahrenheit then display…………………Tolerable.
- If the temperature F is between 30 & 35 degree Fahrenheit then display…………………Warm.
- If the temperature F is less than 30 degree Fahrenheit then display…………………………Cool.

Code:

```
#include<iostream>

using namespace std;


int main()

{

   float f;

   cout<<"Enter Temperature in Fahrenheit"<<endl;

   cin>>f;


   if(f>40)

    cout<<"Very Hot"<<endl;

   if(f<40 && f>35)

    cout<<"Tolerable"<<endl;

   if(f<35 && f>30)

    cout<<"Warm"<<endl;

   if(f<30)

    cout<<"Cool"<<endl;

   return(0);

}
```

Output:

```
Enter Temperature in Fahrenheit
36
Tolerable
```

****

Q#3(a) What does looping mean? Explain different loops in C++.

Looping:

A statement or a set of statement that is executed repeatedly is known as loop. The structure that repeats a statement(s) is known as iterative, repetitive, or looping construct. Loops are basically used for two purposes:

- To execute a statement or number of statements for a specified number of times. For example, a user may display his name on screen for 10 times.
- To use a sequence of values. For example, a user may display a set of natural number from 1 to 100.

Types of Loop:

There are three types of loop:

1. while loop
2. do-while loop
3. for loop

1. while loop:

while loop is the simplest loop in C++ language. This loop executes one or more statement while the condition remains true. It is useful where the number of iterations is not known in advance.

Syntax:

The syntax of while loop is as follow:

```
  while(condition)

     {

          statement;

     }
```

Example:

```
#include<iostream>

using namespace std;


int main()
```

```
{
   int n;
   n = 1;
   while(n<=10)
   {
      cout<<n<<endl;
      n++;
   }
 return(0);
}
```

2. do-while loop:

The do-while is an iterative control in C++ language. This loop executes one or more statements while the given condition is true. In this loop, the condition comes after the body of loop. The loop is important in a situation where a statement must be executed at least once.

Syntax:

The syntax of do-while loop is follow as:

```
do
{
  statement(s);
}
while(condition);
```

Example:

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


int main()
{
   int n;

   n = 1;

   do

   {

      cout<<"Pakistan"<<endl;

      n++;

   }

   while(n<=5);

 return(0);

}
```

3. For Loop:

for loop executes one or more statements for a specified number of times. This loop is also called **counted-controlled loop.** It is the most flexible loop. That is why the most programmers use this loop in programs.

Syntax:

The syntax of for loop is as follow:

```
for(initialization; condition; increment/decrement)

{

  statement(s);

}
```

Example:

```
#include<iostream>

using namespace std;


int main()

{

   int n;

   for(n=1; n<=5; n++)
```

```
    cout<<n<<endl;

 return(0);

}
```

Q#3(b) Write a C++ program to read a number from keyboard and then determine whether it is Even or Odd number?

Code:

```
#include<iostream>

using namespace std;


int main()

{

   int x;

   cout<<"Enter a number"<<endl;

   cin>>x;

   if(x%2==0)

   cout<<"You entered an even number"<<endl;

   else

   cout<<"You entered an odd number"<<endl;

   return(0);

}
```

Output:

```
Enter a number
100
You entered an even number
```

****

Q#4 (a) What is the purpose of using break and continue statements?

1. Break Statement:

The break statement is used in the body of the loop to exit from the loop. When this statement is executed in the loop body, the remaining iterations of the loop are skipped. The control directly moves outside the body and the statement that comes after the body is executed.
For example;

```
 int x;
for(x=1; x<=5; x++)
 cout<<”Questioning”<endl;
 {
 break;
 cout<<”Gateway to knowledge”<<endl;
 }
 cout<<”Bye”<<endl;
```

the above program uses break statement in for loop. The counter variable x indicates that the loop should executed for five times. But it is executed only once. In the first iteration, the cout statement in the loop displays “Questioning” and the control moves to break statement. This statement moves the control out of the loop. So the message appears only once.

2. Continue Statement:

The continue statement is used in the body of the loop. It is used to move the control to the start of the loop body. When this statement is executed in the loop body, the remaining statement of current iteration are not executed. The control directly moves to the next iteration.
For example;

```
int x;
for(x=1; x<=5; x++)
(
 cout<<”Hello World”<<endl;
 continue;
 cout<<”Knowledge is power”<<endl;
}
```

The above example has two cout statements. One statement is before the continue statement and one is after continue statement. The second statement is never executed. This is because each time continue statement is executed, the control moves back to the start of the body. So “Knowledge is power” is never displayed.

Q#4 (b) Write a program to find the sum of the following numbers:
1+2+3+…………..+10

Code:

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

int main()
{

   int sum;
   cout<<"Taking sum of 1+2+3+....+10"<<endl;

   sum=1+2+3+4+5+6+7+8+9+10;
```

```
    cout<<"Sum of All number is "<<sum<<endl;
    return(0);

}
```

Output:

```
Taking sum of 1+2+3+....+10
Sum of All number is 55
```

****

Q#5 What is an array? Explain One-Dimensional and Two-Dimensional Arrays with examples.

Array:

"An array is a collection of variables that can store data of same type. Each memory location holds a single value which is called an element of an array".

Another definition of array is "An array is a consecutive group of memory locations that all have same type. To refer to a particular location or element in the array".

The array is represented in the computer's memory by a set of consecutive memory locations. The memory locations are referred to as elements of the array. Each array is given a name and the elements of the array are accessed with the reference to their position or location number. This position number is called index or subscript.

The subscript or index value is written in parentheses with the name of array. The first element of array has an index value of 0 unless specified otherwise, and the index is incremented for each next element. A subscript must be an integer on integer expression (using any integral type).

Declaring Array:

Array occupy space in memory. To specify the type of the elements and the number of elements required by an array use a declaration of the form:

type arrayName[arraysize];

example;

```
int c[12];
```

Types of Array:

There are two major types of an array which are as follow:

1. One-Dimensional Array

2. Two-Dimensional Array

1. One-Dimensional Array:

One-Dimensional array is also known as **linear array or vector array**. It consists of only one row or column. It is also called 1-D array. One-dimensional arrays are the most used and the easiest to work with from a coding perspective. For example, on the video game screen, you see names and scores. When it comes to the score, the computer only sees what is represented in below figure.

500	550	605	682	765

Figure shows a representation of an array of integers. There are five high scores listed. Each store has its own container. In programming , we like to call these buckets or index.

Example:

```
#include<iostream>

using namespace std;

int main()

{

   int i,a[5]={2,3,5,6,7};

   cout<<"The elements of array a "<<endl;

   for(i=0; i<=4;i++)

     cout<<a[i];


    return(0);

}
```

Output:

```
The elements of array a
23567
```

2. Two-Dimensional Array:

Two-Dimensional Array consists of rows and columns. It is also known as **table or matrix.** Two-dimensional array is also defined as: array of one-dimensional arrays. The elements of two-dimensional array is referenced by two index values. One index value represents the rows and the second represent the column. An array that requires two subscripts to identify a particular element is also known as the **double-subscripted array**.

Example:

```
#include<iostream>

using namespace std;


int main()
```

```
{
   int a,b;
   int ArrayA[a][b];


   cout<<"Input 1: ";
   cin>>ArrayA[0][0];
   cout<<"Input 2: ";
   cin>>ArrayA[1][0];


   cout<<"Row 1 column 1: "<<ArrayA[0][0]<<endl;
   cout<<"Row 2 column 2: "<<ArrayA[1][0]<<endl;


system("pause")
return (0);
}
```

****