



Object Oriented Programming (CS02203)

Lab Report

Name: Noor Baho
Registration #: CSU-F12-116
Lab Report #: 02
Dated: 29-10-2018
Submitted To: Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 2

Selection and Iteration Statements

Objective

The objective of this lab is to understand the different types of loops and decision making statements in C++.

Software Tool

- **OS Windows 10**
- **Editor DEV C++**
- **Language C++**

1 Theory

Decision making

Decision making is about deciding the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met. C++ handles decision-making by supporting the following statements

- **if statement**
- **switch statement**
- **conditional operator statement**

Decision making with if statement

The IF statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are

- **Simple if statement**

- **If...else statement**
- **Nested if....else statement**

Simple if statement

The general form of a simple if statement is

```
if( expression )
{
    Statement - inside;
}
Statement - outside;
```

If the expression is true, then 'statement-inside' it will be executed, otherwise 'statement-inside' is skipped and only 'statement-outside' is executed.
Example:

```
int main( )
{
    int x,y;
    x=15;
    y=13;
    if (x > y )
    {
        cout << "x is greater than y";
    }
}
```

Output: x is greater than y

if...else statement

The general form of a simple if...else statement is

```
if( expression )
{
    Statement - block1;
}
else
{
```

```

        Statement – block2;
    }

```

If the 'expression' is true, the 'statement-block1' is executed, else 'statement-block1' is skipped and 'statement-block2' is executed.

Example

```

void main( )
{
    int x=15; y=18;
    if (x > y )
    {
        cout << "x is greater than y";
    }
    else
    {
        cout << "y is greater than x";
    }
}

```

Output: y is greater than x

Nested if....else statement

The general form of a nested if...else statement is

```

if( expression )
{
    if( expression1 )
    {
        statement-block1;
    }
    else
    {
        statement-block 2;
    }
}
else
{

```

```

        statement-block 3;
    }

```

if 'expression' is false the 'statement-block3' will be executed, otherwise it continues to perform the test for 'expression 1' . If the 'expression 1' is true the 'statement-block1' is executed otherwise 'statement-block2' is executed.
Example:

```

void main( )
{
    int a,b,c;
    clrscr();
    cout << "enter 3 number";
    cin >> a >> b >> c;
    if(a > b)
    {
        if( a > c)
        {
            cout << "a is greatest";
        }
        else
        {
            cout << "c is greatest";
        }
    }
    else
    {
        if( b> c)
        {
            cout << "b is greatest";
        }
        else
        {
            cout <<"c is greatest";
        }
    }
    getch();
}

```

Switch statement

If there are a number of decision making conditions instead of making an if.else construct, the programmer can use switch statement. The general syntax is:

```
switch (expression)
{
    case constant1:
        group of statements 1;
        break;
    case constant2:
        group of statements 2;
        break;
    .
    .
    .
    default:
        default group of statements
}
```

In the above example, the expression is first evaluated and the value is checked with the constant1 in the case statement. If this is evaluated with the same value, then the group of statements is executed. When a break statement is encountered, the control is switched to the end of the switch statement. If the value of the expression is not equal to constant1, it is checked with constant2. If it evaluates the same value, then the group of statements (in the case of constant2) is executed. When a break statement is encountered, the control is then switched to the end of the switch statement. This proceeds until the value of expressions equal one of the constant values given. If the value of the expression is not equal to any of the constant values given, then, by default, the statements present are executed. Example

```
void main()
{
    int x;
    cout<<"Enter Input Value: ";
    cin>>x;
```

```

switch(x)
{
    case 10:
        cout<< "Value is 10";
        break;
    case 20:
        cout<<"Value is 20";
        break;
    default:
        cout<<"Invalid Input";
}
}

```

Conditional operator

The conditional operator actually is a form of if...else statement.

```

if (a < b)
m=a;
else
m=b;

```

could be written using conditional operators as

```

m=(a < b)? a:b;

```

Looping statement

Sometimes we require a set of statements to be executed a number of times by changing the value of one or more variables each time to obtain a different result. This type of program execution is called looping. C++ provides the following construct

- while loop
- do-while loop
- for loop

While loop

Syntax of while loop

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

If the condition is true, the loop body is executed and the condition is re-evaluated. Hence, the loop body is executed repeatedly as long as the condition remains true. As soon as the condition becomes false, it comes out of the loop and goes to the statement next to the ‘while’ loop. Example

```
int main()
{
    int counter , howmuch;
    cin >> howmuch;
    counter = 0;
    while ( counter < howmuch)
    {
        counter++;
        cout << counter << '\n';
    }
    return 0;
}
```

do-while loop

Syntax of do-while loop

```
do
{
    statements;
} while ( condition );
```

The loop body is always executed at least once. One important difference between while loop and do-while loop is the relative ordering of the conditional test and loop body execution. In the while loop, the loop repetition

test is performed before each execution the loop body; the loop body is not executed at all if the initial test fail. In the do-while loop, the loop termination test is performed after each execution of the loop body. hence, the loop body is always executed least once. Example

```
int main()
{
    int counter, howmuch;
    cin >> howmuch;
    counter = 0;
    do
    {
        counter++;
        cout << counter << '\n';
    }
    while ( counter < howmuch);
    return 0;
}
```

For loop

It is a count controlled loop in the sense that the program knows in advance how many times the loop is to be executed. Syntax of for loop:

```
for (initialization; decision; increment/decrement)
{
    statement(s);
}
```

In for loop three operations take place:

- **Initialization of loop control variable**
- **Testing of loop control variable**
- **Update the loop control variable either by incrementing or decrementing**

If the condition is true, the program executes the body of the loop and then the value of loop control variable is updated. Again it checks the condition and so on. If the condition is false, it gets out of the loop. Example

```
int main()
{
    int i;
    for (i = 0; i < 10; i++)
    {
        cout << "Hello" << "\n";
        cout << "There" << "\n";
    }
    return 0;
}
```

Jump Statements

The jump statements unconditionally transfer program control within a function.

- **goto statement**
- **break statement**
- **continue statement**

The goto statement

goto allows to make jump to another point in the program.

goto pqr;

pqr: pqr is known as label. It is a user defined identifier. After the execution of goto statement, the control transfers to the line after label pqr.

The break statement

The break statement, when executed in a switch structure, provides an immediate exit from the switch structure. Similarly, you can use the break statement in any of the loop. When the break statement executes in a loop, it immediately exits from the loop. Example

```
int main()
{
```

```

    int i;
    i = 0;
    while ( i < 20 )
    {
        i++;
        cout << "Hello\n";
        if ( i == 10)
            break;
    }
    return 0;
}

```

In the example above, the while loop will run, as long i is smaller than twenty. In the while loop there is an if statement that states that if i equals ten the while loop must stop (break). The result is that only ten Hello will be printed.

The continue statement

The continue statement is used in loops and causes a program to skip the rest of the body of the loop.

```

while (condition)
{
    Statement 1;
    If (condition)
        continue;
    statement;
}

```

The continue statement skips rest of the loop body and starts a new iteration.

Example

```

int main()
{
    int i;
    i = 0;
    while ( i < 20 )
    {
        i++;

```

```

        continue;
        cout << "Hello\n";
        if ( i == 10)
            break;
    }
    return 0;
}

```

In the example above, the cout function is never called because of the “continue;”.

The exit () function

The execution of a program can be stopped at any point with exit () and a status code can be informed to the calling program. The general format is exit (code) ;

```
void exit (int exitcode);
```

The exitcode is used by some operating systems (UNIX / Linux) and may be used by calling programs. An exit code of 0 means that the program finished normally and any other value means that some error occurred.

```
C:\Users\M Abid\Desktop\CS\OOP\LAB1 TASK2.exe
Enter 1 to Add, 2 to Subtract, 3 to Multiply, 4 to Divide. : 1
Enter 1st number 5
Enter 2nd number 4
Result: 9
Enter 1 to proceed again, 0 to terminate: 2

Enter 1 to Add, 2 to Subtract, 3 to Multiply, 4 to Divide. : 2
Enter 1st number 10
Enter 2nd number 2
Result: 8
Enter 1 to proceed again, 0 to terminate: 3

Enter 1 to Add, 2 to Subtract, 3 to Multiply, 4 to Divide. : 3
Enter 1st number 5
Enter 2nd number 3
Result: 15
Enter 1 to proceed again, 0 to terminate: 4

Enter 1 to Add, 2 to Subtract, 3 to Multiply, 4 to Divide. : 4
Enter 1st number, The dividend: 20
Enter 2nd number, The divisor: 2
Result: 10
Enter 1 to proceed again, 0 to terminate: 0
```

Figure 1: Basic Calculator

2 Task

2.1 Procedure: Task 1

Write a program of basic calculator (+, -, *, /) using switch statement.

Solution:

```
#include <iostream>
#include <conio.h>
float add(float x, float y)
{
    return x+y;
}
float sub(float x, float y)
{
    return x-y;
}
float mul(float x, float y)
{
    return x*y;
}
float div(float x, float y)
```

```

{
    return x/y;
}
using namespace std;
int main ()
{
    float a, b;
    int mod;
    int rep=1;

    while(rep>0){
    cout<<"Enter 1 to Add, 2 to Subtract , 3 to Multiply , 4 to Divide.
    cin>>mod;
    switch (mod)
    {
        case 1:
        case 2:
        case 3:
            {
                cout<<endl<<"Enter 1st number ";
                cin>>a;
                cout<<"Enter 2nd number ";
                cin>>b;
                break;
            }

        case 4:
            {
                cout<<endl<<"Enter 1st number, The dividend
                cin>>a;
                cout<<"Enter 2nd number, The divisor: ";
                cin>>b;
                break;
            }
    }

    }

    switch(mod)
    {

```

```

        case 1:
        {
            cout<<"Result: "<<add(a, b);
            break;
        }

        case 2:
        {
            cout<<"Result: "<<sub(a, b);
            break;
        }

        case 3:
        {
            cout<<"Result: "<<mul(a, b);
            break;
        }

        case 4:
        {
            cout<<"Result: "<<div(a, b);
            break;
        }
    }

    cout<<endl<<endl<<"Enter 1 to proceed again, 0 to terminate: ";
    cin>>rep;
    cout<<endl<<endl;

}

getche();
return 0;
//clrscr();
}

```

Explanation: Required is to program a basic calculator which perform addition, subtraction, multiplication and division. For the purpose I declared a variable 'mod' to ask user what he want to do from the four up

mentioned operations. Take input and applied switch over that variable mod. For each case at back end is a function defined, which is taking values of x and y as arguments and returning the respective result. Also I declared two variables x and y to take input from user which is used in the functions. Overall while loop is controlled by variable 'rep' asking user if he want to perform more calculations or not. if yes then proceeding again, if no then terminating the program.



Figure 2: Number Factorial Calculator

2.2 Procedure: Task 2

Write a program to find the factorial of a number.

Solution:

```
#include <iostream>
#include <conio.h>
//factorial finding
using namespace std;

int main()
{
    int a, f=1;
    cout<<"Enter the number to find factorial: ";
    cin>>a;

    for(int x=a; x>0; x--)
    {
        f=f*x;
    }
    cout<<"Factorial of number "<<a<<" is: "<<f;
    getch();
    return 0;
}
```

Explanation: Required is to find factorial of the number inserted by user. So we declared a variable a and took input from user into 'a'. another variable f initialized with 1 so that multiplication does not convert a into zero. used for loop over the value of a. Now x (loop variable) is decreasing each time and in each loop f is multiplied by value of x and result is again stored in f. The loop will excute until x decreases and reaches the 0 value, and then it will be terminated. So in f variable multiplication of a to 1 numbers will be stored and that is the factorial of number a, will be shown to user.


A screenshot of a Windows command prompt window titled "C:\Users\IM Abid\Desktop\CS\OOP\LAB2 TASKS\LAB2 TASK3.exe". The window displays the output of a C++ program, which is a sequence of numbers printed on separate lines: 1, 12, 123, 1234, 12345, 123456, 1234567, 12345678, and 123456789. The numbers are left-aligned, creating a triangular shape. The rest of the window is black.

Figure 3: Incrementing the Number Per Row

2.3 Procedure: Task 3

Write a program to print output like: 1, 12, 123, 1234.

Solution:

```
#include <iostream>
#include <conio.h>
//prints like 1, 12, 123, 1234, 12345 etc
using namespace std;

int main()
{
    for (int a=1; a<=9; a++)
    {
        for (int b=1; b<=a; b++)
        {
            cout<<b;
        }
        cout<<endl;
    }

    getch();
    return 0;
```

}

Explanation: Required is the pattern of numbers shown above. So we used nested for loop. Outer for will control the rows and inner for loop will control the columns. Inner for loop relies over variable b initialized with 1 and will execute to the number of row in which operating i.e: in first row, one column. In second row, two columns and so on. And inner loop is doing another thing, it is printing the number of column in which control exists. if 1st column, it will print 1, if 2nd column it will print 2. So inner loop will execute only one time at first as row is 1 and prints 1. For second time, it will print first 1 then 2 and the terminates as row is 2 and proceeding same like that it will draw the same pattern as required.

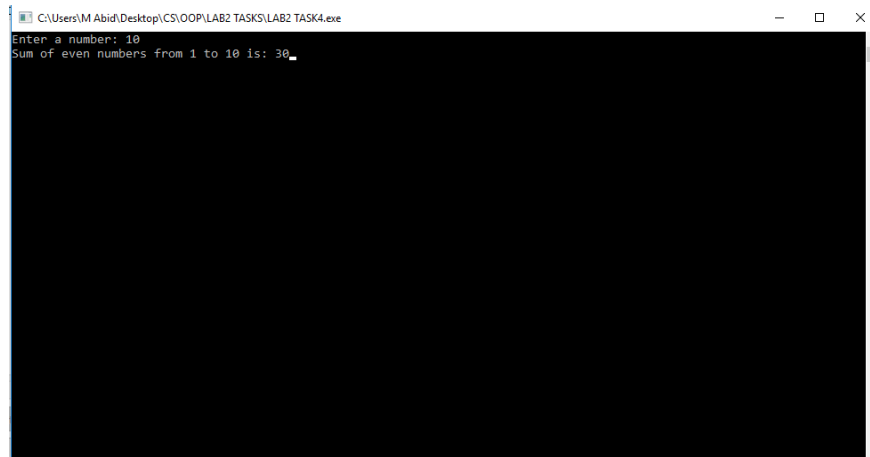


Figure 4: 1 to n Even Sum Calculator

2.4 Procedure: Task 4

Write a program using loop and continue statement to find sum of even numbers from 1 to n.

Solution:

```
#include <iostream>
#include <conio.h>
//find sum of even numbers from 1 to n. using loop and continue.
using namespace std;

int main()
{
    int num, sum;
    cout<<"Enter a number: ";
    cin>>num;

    for (int x=num; x>=0; x--)
    {
        if (x%2==0)
            sum= sum+x;
    }
    cout<<"Sum of even numbers from 1 to "<<num<<" is: "<<sum;
```

```
        getch();  
        return 0;  
}
```

Explanation: Asked is to find sum of even numbers lies between 1 to n number entered. So we declared an integer variable num and took any random number from user. Then used for loop which relies on variable x initialized by the variable num entered by user; any random number. Now loop will execute until it reaches 1 and each time x will decrement to attain previous number then the executed. The body of loop consists of if statement that is taking that x number each time and checking through remainder either it is even or not. if it is even then ti will sum up the value of x into the previous sum and if not even it terminates. On termination of loop the result of all even numbers between the boundary set, will be displayed to the user.

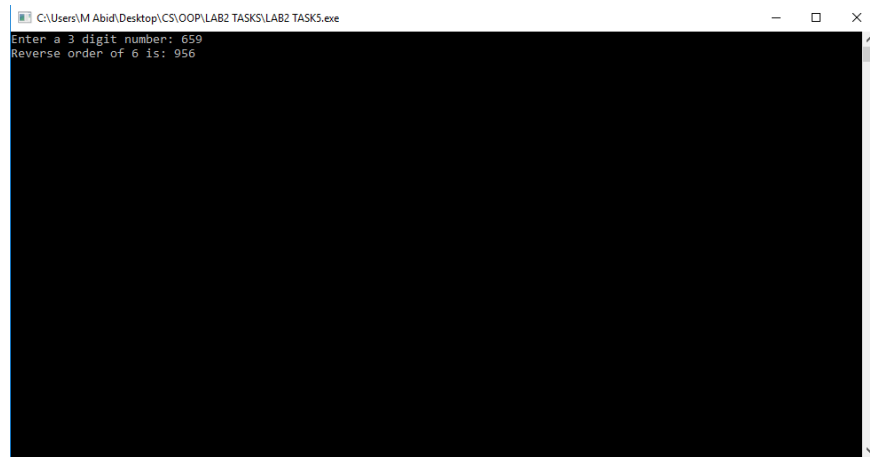


Figure 5: Digit Reverser

2.5 Procedure: Task 5

Write a program that reverses an integer number . i.e: 133=331, 987=789.

Solution:

```
#include<iostream>
#include<conio.h>
//reverse the order of numbers
using namespace std;
int main()
{
    int num;
    int rnum[2];
    cout<<"Enter a 3 digit number: ";
    cin>>num;
    rnum[0]= num%10;                //LAST DIGIT SEGREGATED

    rnum[1]= num%100;
    rnum[1]= rnum[1]/10;           //second digit segregated

    rnum[2]= num/100;              //first digit segregated

    cout<<"Reverse order of "<<num<<" is: ";                //variable
```

```

    for (int a=0; a<=2; a++)
    {
        cout<<rnum[ a ];
    }
    getch ();
    return 0;
}

```

Explanation: Asked is to reverse the digits of an integer number. For the purpose we will use an array of relevant index size as we considered 2 which contains 0-2 indexes and a variable named num which will contain user entered integer. Now it is a 3 digit integer, If we get num remainder 10 the unit number will be segregated (last digit). Now num remainder 100 will segregate the last two digits (the unit and tenth positioned digit) stored in index 1 and then this two digit number is divided by 10 to acquire first digit (tenth positioned) and is stored in index 1 of array. Now comes to the last step of separation, num will be divided by 100 and stored answer into index 2 will be the first digit. To reverse its order we have to print the array in reverse order. index 2 to 1 to 0.



Figure 6: Sum of Digits Calculator
Solution:

2.6 Procedure: Task 6

Write a program that sums the digits of a number. i.e: $133=7$, $679=22$.

```
#include<iostream>
#include<conio.h>
//sums the digits of number
using namespace std;

int main()
{
    int num, sum;
    int rnum[2];

    cout<<"Enter a 3 digit number: ";
    cin>>num;

    rnum[0]= num%10;                //LAST DIGIT SEGREGATED

    rnum[1]= num%100;
    rnum[1]= rnum[1]/10;           //second digit segregated
```

```

    rnum[2]= rnum[2]/100;    //first digit segregated

    sum= rnum[0]+ rnum[1]+ rnum[2];
    cout<<"Sum of digits: "<<sum;

    getch();
    return 0;
}

```

Explanation: Required is to get the sum of digits of entered number. So Segregation of digits will be performed same as we discussed above. After then simply sum up value of the indexes of array and you will get the sum of digits of a randomly entered integer number.

3 Conclusion

Overall we practiced the use of loops, nested loops and decision making statements. With a bit difference in the question statement use of variables, operators and loops also changes accordingly.