Fundamentals of Programming in C++ (CoSc 2031)

CONTROL STATEMENT

Mekonnen K. (MSc)

Email: mekonnen307@gmail.com

2. Control Statement

- **Statements** are the instructions given to the computer to perform any kind of action.
- Action may be in the form of data movement, decision making etc. statements from the smallest executable unit with a c++ program. Statements are always terminated by semicolon.
- **A compound statement:** is a grouping of statements in which each individual statement ends with a semi-colon. The group of statements is called **block.** Compound statements are enclosed between the pair of braces ({ }).
- The opening brace ({) signifies the beginning and closing brace (}) signifies the end of the block.

Control Statement

- A **null statement** is used to provide a null operation in situations where the grammar of the language requires a statement, but the program requires no work to be done. The null statement consists of a semicolon:

```
;
```

- Any single input statement, assignment and output statement is **simple statement.**

Control statement

- A C++ **control statement** redirects the flow of a program in order to execute additional code.
- These statements come in the form of conditionals (if-else, switch) and loops (for, while, do-while).
- The order in which statements are executed in a program is called **flow of control.**

Types of control statement

- ***C++ supports Three basic control statements.***
  - ***Conditional statements***
  - ***Iteration statements***
  - ***Jump Statements***

i. Conditional Statements

- **Conditional statements**, also known as selection statements, are used to make decisions based on a given condition. If the condition evaluates to True, a set of statements is executed, otherwise another set of statements is executed.
- The different Conditional statements are
  - if statement
  - if – else statement
  - Nested – if statement
  - switch statement

if statement

- This is the simplest form of if statement. This statement is also called as one-way branching.
- This statement is used to decide whether a statement or set of statements should be executed or not.
- The decision is based on a condition which can be evaluated to TRUE or FALSE.
- The general form/syntax of simple – if statement is:

```
if (Test Condition) // This Condition is true
        Statement 1;
        Statement 2;
```

if statement

Ex:

```
if( amount > = 5000 )
discount = amount * (10/100);
net-amount = amount – discount;
```

if statement

- **Practical Program 1**: Write a C++ program to find the largest, smallest and second largest of three numbers using simple if statement.

```
#include<iostream.h>
void main( )
{
        int a, b, c;
        int largest, smallest, seclargest;
        cout<<"Enter the three numbers"<<endl;
        cin>>a>>b>>c;
```

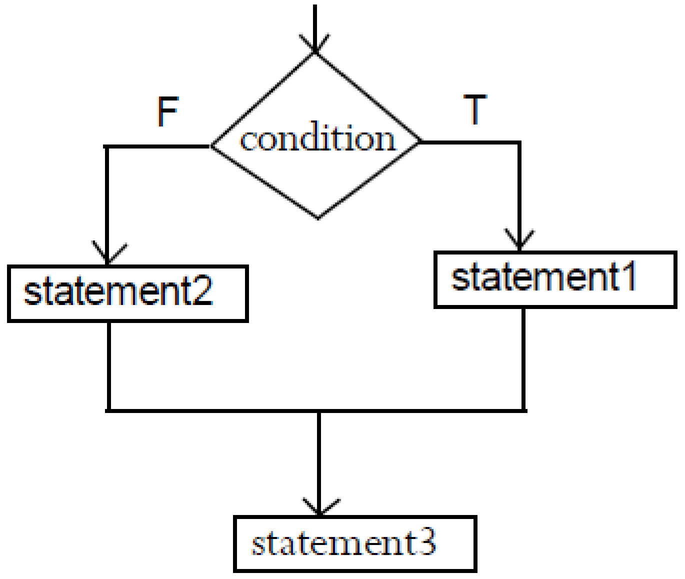
if statement

```
            largest = a; //Assume first number as largest
            if(b>largest)
                        largest = b;
            if(c>largest)
                        largest = c;
            smallest = a; //Assume first number as smallest
            if(b<smallest)
                        smallest = b;
            if(c<smallest)
                        smallest = c;
            seclargest = (a + b + c) – (largest + smallest);
            cout<<"Smallest Number is = "<<smallest<<endl;
            cout<<"Second Largest Number is = "<<seclargest<<endl;
            cout<<"Largest Number is = "<< largest<<endl;
}
```

if – else statement

- This structure helps to decide whether a set of statements should be executed or another set of statements should be executed.
- This statement is also called as two-way branching.
- The general form of if – else statement is:

```
if (Test Condition)
        Statement 1;
else
        Statement 2;
```



if – else statement

- Here, the test condition is tested. If the test condition is TRUE, statement-1 is executed. Otherwise Statement 2 is executed.
- Ex:

```
        if( n % 2 == 0 )
                cout<<" Number is Even";
        else
                cout<<" Number is Odd";
```

Nested if statement

- When there is an if statement inside another if statement then it is called the nested if statement.
- The structure of nested if looks like this:

```
if(condition_1)
{
        Statement1(s);
        if(condition_2)
                {
                Statement2(s);
                }
}
```

- Statement1 would execute if the condition_1 is true. Statement2 would only execute if both the conditions(condition_1 and condition_2) are true

Example of Nested if statement

```
#include <iostream>
using namespace std;
int main(){
      int num=90;
               /* Nested if statement. An if statement
               * inside another if body */
               if( num < 100 ){
               cout<<"number is less than 100"<<endl;
               if(num > 50){
               cout<<"number is greater than 50";
               }
      }
      return 0;
```

if – else - if statement

- if-else-if statement is used when we need to check multiple conditions. In this control structure we have only one “if” and one “else”, however we can have multiple “else if” blocks.
- The general form of if – else – if statement is:

```
if(condition_1)
{
        /*if condition_1 is true execute this*/
        statement(s1);
}
else if(condition_2)
{
        /* execute this if condition_1 is not met and* condition_2 is met */
        statement(s);
}
```

if – else - if statement

```
else if(condition_3)
{
          /* execute this if condition_1 & condition_2 are not met and condition_3 is
          met */
          statement(s);
}
.
.
.
else {
          /* if none of the condition is true then these statements gets executed */
          statement(s);
}
```

if – else - if statement

- Example:

```
if( marks > = 85 )
        PRINT “Distinction”
        else
                if( marks > = 60 )
                        PRINT “First Class”
                else
                        if( marks > = 50 )
                                PRINT “Second Class”
                        else
                                if( marks > = 35 )
                                        PRINT “Pass”
                                else
                                        PRINT “Fail”
```

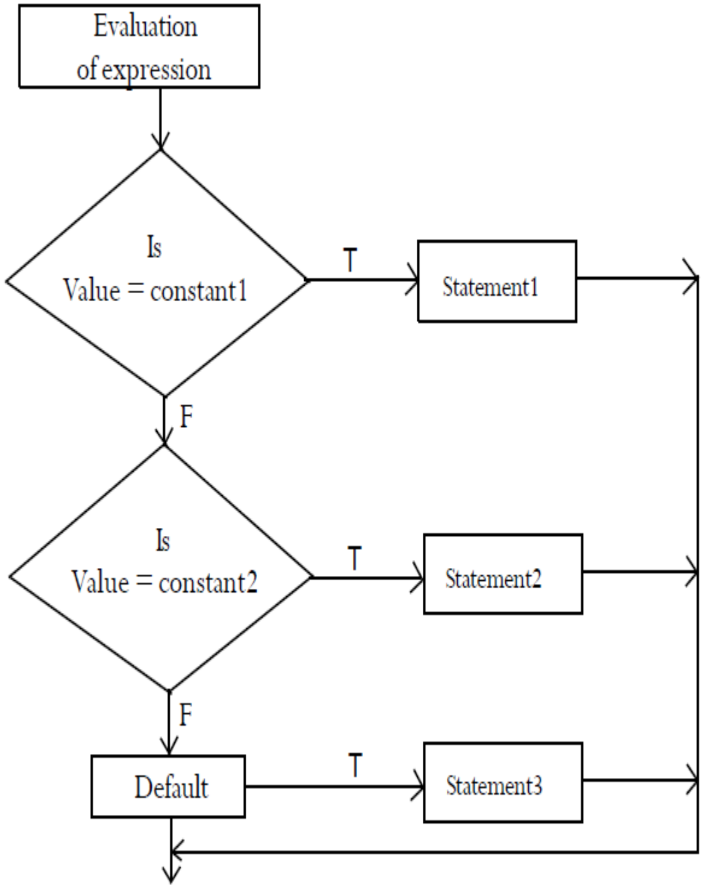
Switch Statement

- The **switch multiple-selection** statement performs different actions based on the value of an **expression.**
- Each action is associated with the value of a **constant integral expression** or a **constant string expression** that the expression may assume.
- It begins with the evaluation of expression.
- The switch statement has four components:
  - ***Switch***
  - ***Case***
  - ***Default***
  - ***Break***

Switch Statement

- If there are more than two alternatives to be selected, multiple selection construct is used. The general form/syntax of Switch statement is:

```
Switch ( Expression )
{
            Case Label-1: Statement 1;
                          Break;
            Case Label-2: Statement 1;
                          Break;
                          ………….
            Case Label-N: Statement N;
                        Break;
            Default : Default- Statement;
}
```



Switch Statement

⇒ *Where Default and Break are Optional the General Syntax might be:*

```
switch(expression)
{
        case constant1:
                statements;
                .
                .
                .
        case constant n:
                statements;
        default:
                statements;
}
```

Switch Statement

- Ex: To find the name of the day given the day number

```
switch ( dayno )
{
          Case 1: cout<< “Sunday”<<endl;
                    break;
          Case 2: cout<< “Monday” <<endl;
                    break;
          Case 3: cout<< “Tuesday” <<endl;
                    break;
          Case 4: cout<< “Wednesday” <<endl;
                    break;
```

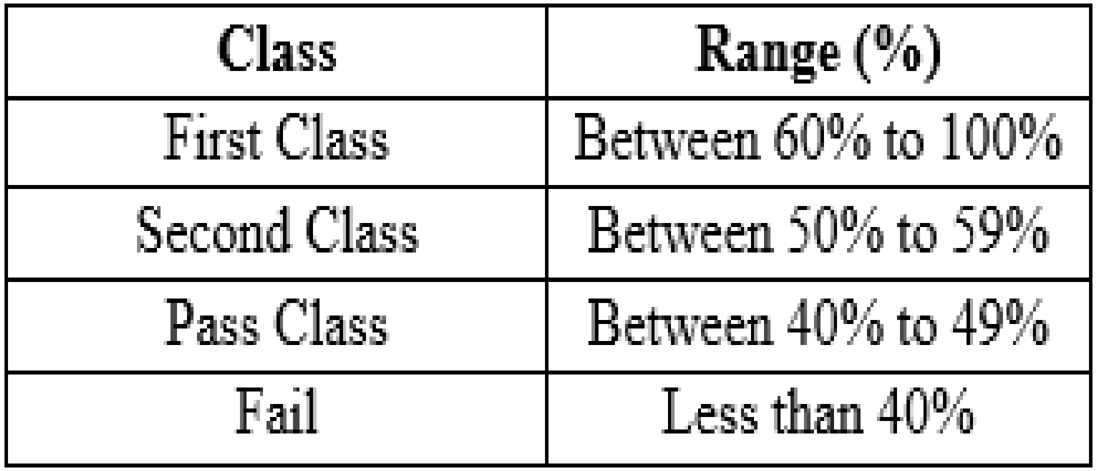
Switch Statement

```
                Case 5: cout<< “Thursday” <<endl;
                        break;
                Case 6: cout<< “Friday” <<endl;
                        break;
                Case 7: cout<< “Saturday” <<endl;
                        break;
                default: cout<< “Invalid Day Number” <<endl;
}
```

- The switch statement is a bit peculiar within the C++ language because it uses labels instead of blocks.
- This force up to put break statements after the group of statements that we want to execute for a specific condition.

Switch Statement

- Otherwise the remainder statements including those corresponding to other labels also are executed until the end of the switch selective block or a break statement is reached.
- **Practical Program 5:** Write a C++ program to input the marks of four subjects. Calculate the total percentage and output the result as either "First Class" or "Second Class" or "Pass Class" or "Fail" using switch statement.



Class	Range (%)
First Class	Between 60% to 100%
Second Class	Between 50% to 59%
Pass Class	Between 40% to 49%
Fail	Less than 40%

Switch Statement

```
void main ( )
{
        int m1, m2, m3, m4, total, choice;
        float per;
        cout<<"Enter the First subject marks"<<endl;
                cin>>m1;
        cout<<"Enter the Second subject marks"<<endl;
        cin>>m2;
        cout<<"Enter the Third subject marks"<<endl;
        cin>>m3;
        cout<<"Enter the Fourth subject     marks"<<endl;
        cin>>m4;
        total = m1 + m2 + m3 + m4;
```

Switch Statement

```
per = (total / 400) * 100;
cout<<"Percentage = "<<per<<endl;
choice = (int) per/10;
cout<<"The result of the student is: "<<endl;
                switch(choice) {
                                case 10:
                                case 9:
                                case 8:
                                case 7:
                                case 6: cout<<"First Class"<<endl;
                                                break;
                                case 5: cout<<"Second Class"<<endl;
                                                break;
                                case 4: cout<<"Pass Class"<<endl;
                                                break;
                                default: cout<<"Fail"<<end; }
                                }
```

Eg:- Write a c++ program build a simple calculator using switch Statement?

```
#include<iostream>
using namespace std;
int main() {
        int n1,n2;
        char op;
        cout<<"Enter the Case"<<endl;
        cin>>op;
        cout<<"Enter the two numbers"<<endl;
        cin>>n1;
        cin>>n2;
        switch(op) {
        case '1':
        cout<<n1<<"+"<<n2<<"="<<n1+n1<<endl;
        break;
        case '2':
        cout<<n1<<"-"<<n2<<"="<<n1-n2<<endl;
        break;
                case '3':
cout<<n1<<"*"<<n2<<"="<<n1*n2<<endl;
break;
case '4':
cout<<n1<<"/"<<n2<<"="<<n1/n2<<endl;
break;
case '5':
cout<<n1<<"%"<<n2<<"="<<n1%n2<<endl;
break;
default:
cout<<"No Operator"<<endl;
                }
                }
```

The following rules apply to a switch statement

- Case Label must be unique
- Case Labels must ends with Colon
- Case labels must have constants / constant expression
- Case label must be of integral Type (Integer, Character)
- Switch case should have at most one default label
- Default label is Optional
- Default can be placed anywhere in the switch
- Break Statement takes control out of the switch
- Two or more cases may share one break statement
- Nesting (switch within switch) is allowed.
- Relational Operators are not allowed in Switch Statement.
- Macro Identifier are allowed as Switch Case Label.
- Const Variable is allowed in switch Case Statement.
- Empty Switch case is allowed.

2. Repetition Statements (Iterative Constructs or Looping)

- Iterative statements are the statements that are used to repeatedly execute a sequence of statements until some condition is satisfied or a given number of times.
- The process of repeated execution of a sequence of statements until some condition is satisfied is called as iteration or repetition or loop.
- Iterative statements are also called as repetitive statement or looping statements.
- There are three types of looping structures in C++.
  - for loop
  - while loop
  - do while loop

For Loop

- Used as a counting loop
- Used when we can work out in advance the number of iterations, i.e. the number of times that we want to loop around.
- Semicolons separate the items in the for loop block
    - There is no semi colon at the end of the for loop definition at the beginning of the statement

For Loop

- The general form of for structure is as follows:

```
for ( Expression 1; Expression 2; Expression 3)
{
        Statement 1;
        Statement 2;
        Statement N;
}
```

Where, Expression 1 represents Initialization

Expression 2 represents Condition

Expression 3 represents Increment/Decrement

The for Statement Syntax

start condition while condition change expression

```
Example:
    for (count=1; count < 7; count++)
    {
        cout << count << endl;
    }
    //next C++ statements;
```

For Loop

This looping structure works as follows:

- **Initialization** is executed. Generally it is an initial value setting for a counter variable and is executed only one time.
- **Condition** is checked. if it is TRUE the loop continues, otherwise the loop ends and control exists from structure.
- Statement is executed as usual, is can either a single statement or a block enclosed in curled braces { and }.
- At last, whatever is specified in the increase field is executed and the loop gets back to executed step 2.
- The initialization and increase fields are optional. They can remain empty, but in all cases the semicolon sign between them must be written compulsorily.
- Optionally, using the comma operator we can specify more than one expression in any of the fields included in a for loop.

For Loop

- **Practical Program 6:** Write a C++ program to find the factorial of a number using for statement.(Hint: 5! = 5 * 4 * 3 * 2 * 1 = 120)

```
        void main ( )
        {
                int num, fact, i;
                cout<<"Enter the number"<<endl;
                cin>>num;
                fact = 1;
                for( i = 1; i<= num; i++)
                        fact = fact * i;
                cout<<" The factorial of a "<<num<<"! is: "<<fact<<endl;
        }
```

For Loop

- **Practical Program 7**: Write a C++ program to generate the Fibonacci sequence up to a limit using for statement. (Hint: 5 = 0 1 1 2 3)

```
void main ( )
{
        int num, first, second, count, third;
        clrscr( );
        cout<<"Enter the number limit for Fibonacci Series"<<endl;
        cin>>num;
        first = 0; second = 1;
        cout<<first<<"\t"<<second;
        third = first + second;
```

For Loop

```
        for( count = 2; third<=num; count++)
        {
                cout<<"\t"<<third
                first = second;
                second = third;
                third = first + second;
        }
        cout<<"\n Total terms = "<<count<<endl;
        getch( );
}
```

while loop

- This is a pre-tested loop structure.
- This structure checks the condition at the beginning of the structure.
- The set of statements are executed again and again until the condition is true.
- When the condition becomes false, control is transferred out of the structure.
- The general form of while structure is

```
`while ( Test Condition)
{
        Statement 1
        Statement 2 ……..
        Statement N
}//End of While
```

while loop

- **Example:**

```
n = 10;
While ( n > 0)
{
        cout<<n<<”\t”;
        - - n;
}
cout<<”End of while loop \n”;
```

Output: 10 9 8 7 6 5 4 3 2 1 End of while loop

while loop

- **Practical Program** 8: Write a C++ program to find sum of all the digits of a number using while statement.

```
void main ( ) {
             int num, sum, rem;
            clrscr( );
            cout<<”Enter the Number”<<endl;
            cin>>num;
            sum = 0;
            while(num!=0)
            {
                        rem = num % 10;
                        sum = sum + rem;
                        num = num/10;
            }
            cout<<”Sum of the digits is “<<sum<<endl;
```

Do While Statements

- This is a **post-tested loop** structure.
- This structure checks the condition at the end of the structure.
- The set of statements are executed again and again until the condition is true.
- When the condition becomes false, control is transferred out of the structure.
- The general form of while structure is

```
do
{
        Statement 1
        Statement 2
        ……..
        Statement N
} while ( Test Condition);
```

Example:

```
i = 2;
do
{
        cout<<i<<”\t”;
        i = i + 2;
}
while ( i < = 25);
```

Do While Statements

Practical Example 9:- C++ Program to determine Factorial of a number

```
int main()
{
        int n,i=1;
        int fact=1;
        cout<<"Enter a Number "<<endl;
        cin>>n;
        do
        {
                fact=fact*i;
                i++;
        }while(i<=n);
        cout<<"Factorial of a number is:"<<fact<<"\n";
        }
```

Nested for loop

```
for(initialization ; condition ; increase/decrease)
      for(initialization ; condition ; increase/decrease)
            for(initialization ; condition ; increase/decrease)
                  for(initialization ; condition ;incr/decr)
                     .
                      .
                        statement;
```

Nested for loop

Example:- What is The output of the following program

```
int main()
{
        for(int i=1;i<=5;i++)
        {
                for(int j=1;j<=5;j++)
                {
                        cout<<j;
                }
                cout<<endl;
        }
        return 0;
}
```

OUTPUT

```
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
```

Nested for loop

Example:- What is The output of the following program

```
int main()
{
        for(int i=1;i<=5;i++)
        {
                for(int j=1;j<=5;j++)
                {
                        cout<<"*";
                }
                cout<<endl;
        }
        return 0;
}
```

OUTPUT

```
*   *   *   *   *
*   *   *   *   *
*   *   *   *   *
*   *   *   *   *
*   *   *   *   *
```

Difference between while and do while loop

while	do while
○ This is pre- tested looping structure	○ This is post tested looping structure
○ It tests the given condition at initial point of looping structure	○ It tests the given condition at the last of looping structure.
○ Minimum execution of loop is zero	○ Minimum execution of loop is once.

Syntax:
while (Test condition)
{
statement 1;
statement 2;
…………….;
statement n;
}
	Syntax:
	do
	{
	statement 1;
	statement 2;
	statement n;
	}
	while (Test condition);

○ Semi colon is not used.	○ Semi colon is used.

3. Jump statements

- Jump statements are used to manipulate the flow of the program if some conditions are met. It is used to terminating or continues the loop inside a program or to stop the execution of a function. In C++ there is four jump statement:
  - break,
  - continue,
  - goto and
  - return.

Break Statement

- The break statement has two uses
  - You can use it to terminate a case in the switch statement.
  - You can also use it to force immediate termination of a loop like while, do-while and for, by passing the normal loop conditional test.
- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement.
- The general form of break statement is:

```
break;
```

Break Statement

- Example:

```
                    for (n=0; n<100; n++)
                    {
                              cout<<n;
                              if(n==10) break;
                              }
```

- **Program:** To test whether a given number is prime or not using break statement.

```
          void main( )
          {
                    int n, i, status;
                    clrscr( );
                    cout<<"Enter the number";
                    cin>>n;
```

Break Statement

```
status=1;
for(i=2; i<=n/2; i++)
{
        if(n % i ==0)
        {
                status=0
                cout<<"It is not a prime"<<endl;
                break;
        }
}
if(status)
cout<<"It is a prime number"<<endl;
getch( );}
```

Continue Statement

- The continue statement is used to skip the current iteration of the loop and the control of the program goes to the next iteration.
- The syntax of the continue statement is:

```
continue;
```

Example:

```
for (int n=10; n>0; n--)
{
        if(n==10) continue;
        cout<<n;
}
```

go to statement

- goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.
- The go to allows to makes an absolute jump to another point in the program.
- The destination is identified by a label, which is then used as an argument for the goto statement.
- A label is made of a valid identifier followed by a colon (:).

go to statement

The general syntax of goto statement is:

```
            statement1;
            statement2;
      goto label_name;
            statement 3;
            statement4;
      label_name: statement5;
            statement6;
```

go to statement

- **Program:** To print from 10 to 1 using goto statements.

```
void main( )
{
        int n=10;
        loop:
        cout<<"\t"<<n;
        n--;
        if(n>0) goto loop;
        cout<<"End of loop";
        getch( );
}
```

Problem

1. Construct a C++ program for determining income taxes. Assume that these taxes are assessed at 2% of taxable incomes less than or equal to 20,000ETB. For taxable income greater than ETB20,000, taxes are 2.5% of the income that exceeds 20,000ETB plus a fixed amount of 400ETB.
2. Mr Tomoshke invests 1000.00ETB in a saving account with 5 percent interest. Assuming that all interest is left on deposit in the account, calculate and print the amount of money in the account at the end of each year for 10 years. Use the following formula for determining these amounts: **a = p (1 + r)^n** where p is the original amount invested, **r** is the annual interest rate and **n** is the number of years and **a** is the amount on deposit at the end of the **nth** year.

The End!