Sw itch Statement:

* Simplifies if-else-if statement. [ Multiple conditions ]
* The switch statement contains multiple blocks of code called **cases** and a single case is executed based on the expression which is being switched.
* It also enhances the readability of the program.

The syntax to use the switch statement is given below:

1. **switch** (expression) {
2. **case** value1:
3. statements1;
4. **break**;
5. **case** value2:
6. Statements2;
7. **break**;
8. .
9. .
10. .
11. **case** valueN:
12. statementN;
13. **break**;
14. **default**:
15. **default** statements;
16. }

Points to be noted about switch statement:

* Checks only the = condition
* The case variables can be , int, long, char, bool, enum data
* float and double data type are not allowed in switch statement.
* **‘break’** statement terminates the switch block when the condition is satisfied.  
  It is optional, if not used, next case is executed. It is known as **‘fall through’**
* Cases cannot be duplicated
* ‘default’ block must be specified at last.
* ‘default’ block is executed when any of the case doesn't match the value of expression. It is optional.
* No need to use ‘break’ statement in default block.
* While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

Consider the following example to understand the flow of the switch statement.

**Student.cpp**

// Program to check in which school the student is studying using switch statement

1. **void** main ( ) {
2. **char name[]** = {“Suresh”};
3. **int grade = 7;**
4. int hsc1 = 11, hsc2 = 12;
5. **switch** ( grade ) {
6. **case** 1:
7. **case 2**:
8. **case 3**:
9. **case 4**:
10. **case 5**:

cout << name << " is studying in primary school" ;

**break**;

1. **case** 6:
2. **case 7**:
3. **case 8**:

cout << name << " is studying in middle school"  ;

**break**;

1. **case 9**:
2. **case 10**:

cout << name << " is studying in high school"  ;

**break**;

1. **case hsc1**:
2. **case hsc2**:

cout << name << " is studying in higher secondary school"   ;

**break**;

1. **default**:

cout << name << " is not studying in school" ;

1. }
2. }

**Output:**

Suresh is studying in middle school

// Program to check in which school the student is studying using if-else-if statement

1. **void** main ( ) {
2. **char name[]** = {“Suresh”};
3. **int grade = 7;**
4. int hsc1 = 11, hsc2 = 12;
5. if ( grade>=1 && grade <= 5)
6. {

cout << name << " is studying in primary school"  ;

1. }
2. else if (grade>=6 && grade <= 8)
3. {

cout << name << " is studying in middle school"   ;

1. }
2. else if ( grade ==9 || grade == 10)
3. {

cout << name << " is studying secondary school"  ;

1. }
2. else if (grade == hsc1 || grade ==hsc2)
3. {

cout << name << " is studying in higher secondary school"  ;

1. }
2. else
3. {

cout << name << " is not studying in school";

1. }
2. }

/\* Program to allocate the work based on gender and grade point value

Gender is ‘M’ -> Grade is above 8.5 ------ ‘Office work’

Otherwise ‘Field work’

Gender is ‘F’ -> ‘Office work’

\*/

void main ( )

{

char name [50];

char gender;

float grade;

cin>>“Enter name, gender and grade of the person“>>”\n”;  
 cin >> name >> gender >> grade ;

if ( ( gender == ‘M’ || gender == ‘m’ ) && grade >= 8.5 )

{

cout<< name << “ is appointed for office work” << “\n”;

else

cout << name << “ is appointed for field work” << endl ;

}

else if ( gender == ‘F’ || gender == ‘f’ )

{

cout<<name << “ is appointed for office work” << endl;

}

else

cout << name << “ is not appointed “ <<””\n;

}

}

Loop Statements

* In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true.
* Loop statements are used to execute the set of instructions in a repeated order.
* The execution of the set of instructions depends upon a particular condition.

Steps in looping statements

* 1. Initialization
  2. Condition
  3. Set of statements
  4. Increments/Decrements (Step Size)

Types of loops ( Based on the condition specified )

* 1. Entry condition loop ( while loop, for loop )
  2. Exit condition loop ( do – while loop )

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

while loop

* Entry condition loop
* Iterate over the number of statements multiple times.
* If we **don't know the number of iterations in advance**, it is recommended to use a while loop.

General Format for while loop

initialization

**while** ( condition) {

//looping statements

Set of statements

Increment/ decrement

}

The flow chart for the while loop is given in the following image.



Consider the following example.

**// Print all the numbers divisible by 3 from 50 to 100 using while loop**

**void** main( ) {

**int** a = 50;

cout << "Printing the list of numbers divisible by 3 from 50 to 100 \n";

**while** ( a<=100 ) {

if ( a % 3 == 0 )

cout << ( I );

a ++ ; //// ++ a; ///// a = a + 1; ///// a += 1;

}

}

do-while loop

* Exit condition loop
* Condition is specified at the last, so that the body of the loop is executed at least once though the condition is false at the first time.
* If w**e don't know the number of iterations in advance**, it is recommended to use a do-while loop.

The syntax of the do-while loop is given below:

initialization

**do** ( condition) {

//looping statements

Set of statements

Increment/ decrement

}   while ( condition );

**// Print all the numbers in the series 30 25 20 15 10 5 0 -5 -10 -15 -20 -25 -30 using do-while loop**

**void** main ( ) {

**int** a = 30;

cout << "Print the series of numbers \n";

do {

cout << a << “\t “ ;

a = a - 5; ///// a -= 5;

}   **while** ( a>= -30 ) ;

cout << “Program terminated \n”;

}

for loop

* Entry condition loop
* Enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
* Semicolon ( ; ) must be used to separate each part
* Use the for loop only when we exactly know the number of times

Syntax for **‘for’** loop

**for** ( initialization ; condition ; increment/decrement ) {

Set of statements //block of statements

}

The flow chart for the for-loop is given below.



Consider the following example to understand the proper functioning of for loop:

// Factorial of a given number using for loop ------ n! = 1 \* 2 \* 3 \* 4 \* 5

**void** main( ) {

int n;

**int** fac = 1;

**cout << “Enter n value “;**

**cin >> n ;**

**for**( **int** i = 1; i <= n ; i++) {

fac = fac \* i;

}

cout << "Factorial =  " << fac ;

}

}

**Output:**

Factorial = 120

Nested looping structure

* Any loop (while, do-while, for) can be nested
* The entire loop is specified within another loop
* Loop can’t be overlapped

Example :

int i =1;

while ( i <= 3 )

{

int j = 1;

while ( j <= 5 )

{

cout << “ I = “ << i << “ , J = “ << j << “\n” ;

j++ ;

}

I++;

}

Output :

I = 1 , J = 1

I = 1 , J = 2

I = 1 , J = 3

I = 1 , J = 4

I = 1 , J = 5

I = 2 , J = 1

I = 2 , J = 2

I = 2 , J = 3

I = 2 , J = 4

I = 2 , J = 5

I = 3 , J = 1

I = 3 , J = 2

I = 3 , J = 3

I = 3 , J = 4

I = 3 , J = 5

* Outer loop ( I ) is executed 3 times
* Inner loop ( J ) is executed 3 \* 5 ( I \* j ) 15, times

Nested loop using for loop

for ( I = 1; I <= 3; i++ )

{

for ( j = 1; j<= 5; j++ )

{

cout << “ I = “ << i << “ , J = “ << j << “\n” ;

}

}

**Variation in for loop**

* 1. // No initialization part in for loop

int a = 1;

for ( ; a <= 10 ; a++)

{

cout << a << “ \t “;

}

* 1. // No initialization as well as step size in for loop

int a = 1;

for ( ; a <= 10 ; )

{

cout << a << “ \t “;

a++;

}

**Note :** This for loop is similar to while loop.

* 1. Infinite loop

int a = 1;

for ( ; ; )

{

cout << a << “ \t ”;

a++;

}

NOTE:

bool a = true;

while (a)

{

-------

}

Or

while (true)

{

}

Note :

* Infinite loop is not included in the program
* There must be any condition with **break** statement to come out (terminate) the loop at any time, otherwise the system will go to the hanging state

int a = 1;

for ( ; ; )

{

if ( a <= 10 )

cout << a << “ \t “ ;

else

break;

a++;

}

* 1. Null loop

for (int I = 1; i<=2000; i++ ) ;

* 1. Empty loop ///// To specify delay

for (int I = 1; I <=2000; i++ )

{ }

(OR)

for (int I = 1; i<=2000; i++ ) ;

Note :

An empty statement in C++ is written as a single semicolon.

The empty statement doesn't do anything, but the syntax is occasionally useful.

For example,

You can use it to indicate an empty loop body of a for loop:

int a [ ] = { 10, 20, 30, 40 ,50 };

for(int i = 0; i < 10; a[i++]++) // Increment array elements

/\* empty \*/; // Loop body is empty statement

Output: The array elements would be

11, 21, 31, 41, 51 ///// Array elements

* 1. For loop with more than one initialization
* The for loop syntax does not restrict you to writing loops that use only a single variable.
* Both the initialize and increment expressions of ‘for’ loop can use a comma to separate multiple initializations and increment expressions.

For example:

for (int i = 0, j = 10 ; i < 10 ; i++, j-=2)

sum += i \* j;

* 1. For loop with more than one conditions

for (int i = 1, j = 10 ; i <= 10, j>=0 ; i++, j-=2)

cout << “ I = “ << i << “ , J = “ << j << “\n”;

* More than one conditions in the for loop specifies and ( && ) condition
* When both the conditions are true, the body of the loop will be executed

for (int i = 1, j = 10 ; i <= 10 && j>=0 ; i++, j-=2)

cout << “ I = “ << i << “ , J = “ << j << “\n” ;

Note :

3 iterations are performed, when j=4 the loop is terminated.

**‘break’ statement**

> Come out immediately from the containing loop

> Terminates the current block

> When break; is encountered within the inner loop, the control comes out from the inner loop only.

> To come out from outer loop also, we can specify labelled break statement.

* Used to break the current flow of the program
* Transfer the control to the next statement outside a loop or switch statement.
* Breaks only the inner loop in the case of the nested loop.
* It can only be written inside the loop or switch statement.

Example :

for ( I = 1; I <= 3; i++ )

{

for ( j = 1; j<= 5; j++ )

{

if ( j>3 ) break;

cout << “ I = “ << i << “ , J = “ << j << “\n” ;

}

}

Output :

I = 1 , J = 1

I = 1 , J = 2

I = 1 , J = 3

I = 2 , J = 1

I = 2 , J = 2

I = 2 , J = 3

I = 3 , J = 1

I = 3 , J = 2

I = 3 , J = 3

**‘continue’ statement**

> Continue same loop again by skipping set of statement after continue statement

Example I :

for (i = 1; I <= 3; i++ )

{

for ( j = 1; j<= 5; j++ )

{

if ( j==3 ) continue;

cout << “ I = “ << i << “ , J = “ << j << “\n”;

}

}

Java break statement

* **The break statement example with for loop**

Consider the following example in which we have used the break statement with the for loop.

1. **void** main( ) {
2. **for**(**int** i = 0; i<= 10; i++) {

cout << i << endl ;

**if**(i==6) {

**break**;

1. }
2. }
3. }

**Output:**

0

1

2

3

4

5

6

continue statement

It doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement.

1. **void** main( ) {
2. **for**(**int** i = 0; i<= 2; i++) {
3. **for** (**int** j = i; j<=5; j++) {
4. **if**(j == 4) {
5. **continue**;
6. }
7. cout << ( j );
8. }
9. }
10. }

**Output:**

0

1

2

3

5

1

2

3

5

2

3

5