Flow Control

To alter the order in which the program's statements are executed the concept of *Control Flow* is used.

Control structures:

These are some sections of the program code that contains set of statements, which get executed based on some conditions. There are two kinds of control structures- *Conditionals* and *Loops.*

Conditionals:

- C++ has *if* and *switch-case* conditional structures.

- To check whether some condition is true or false conditional structures use relational and logical operators. Relational operators are used to test the relation between two well defined expressions. On the other hand, logical operators are often used to combine relational expressions into more complicated Boolean Expressions.

Operator	**Meaning**
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Fig 1. Relational Operators.

Example:
Let $x = 7$ and $y = 9$
$x \geq y$ will return a $False$ and $x > 10$ will return a $True$
Note: An expression that returns $True$ and $False$ is called as the Boolean expression.

Operator	**Meaning**
&&	and
||	or
!	not

Fig 2. Logical Operators

According to the rules of logic, the operators return either *true* or *false.*

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Fig 3. Behaviour of && and ||.

a	!a
true	false
false	true

Fig 4. Behaviour of NOT.

Example:

Let, $x = 6$ and $y = 2$

```
!(x > 2) → false
(x > y) && (y > 0) → true
(x < y) && (y > 0) → false
(x < y) || (y > 0) → true
```

if, if-else and else-if:

```
if(condition)
{
        statement1
        statement2
        …
}
```

if

```
if(condition)
{
        statementA1
        statementA2
        …
}
else
{
        statementB1
        statementB2
        …
}
```

if-else

```
if(condition1)
{
        statementA1
        statementA2
        …
}
else if(condition2)
{
        statementB1
        statementB2
        …
}
```

else-if

Example:

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

int main() {
   int x = 6;
   int y = 2;

   if(x > y)
         cout << "x is greater than y\n";
   else if(y > x)
         cout << "y is greater than x\n";
   else
         cout << "x and y are equal\n";

   return 0;
}
```

switch-case:

```
switch(expression)
{
        case constant1:
                statementA1
                statementA2
                ...
                break;
        case constant2:
                statementB1
                statementB2
                ...
                break;
        ...
        default:
                statementZ1
                statementZ2
                ...
}
```

Example:

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

int main() {
   int x = 6;

   switch(x) {
         case 1:
               cout << "x is 1\n";
               break;
         case 2:
         case 3:
               cout << "x is 2 or 3";
               break;
         default:
               cout << "x is not 1, 2, or 3";
   }

   return 0;
}
```

break and continue in C++:

break statement is used to jump out of a loop

Example 1:

```
#include<iostream>
using namespace std;
int main()
{

    cout<<"Example of break"<<endl;
    for(int i =0; i<10; i++)
    {
        cout<<"i="<<i<endl;
        if(i==5)
        {
            break;
        }
    }
}
```

Example 2:

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

int main() {
  int i = 0;
  while (i < 10) {
    cout << i << "\n";
    i++;
    if (i == 4) {
      break;
    }
  }
  return 0;
}
```

Example 3:

```
#include<iostream>
 using namespace std;
 int main()
 {
   for(int i =1;i<=3;i++)
   {
       for(int j =1;j<=3;j++)
       {
           if(i==2 && j==2)
           {
               break;
           }
           cout<<i<<"\t"<<j<<"\n";
       }
   }

 }
```

continue statement breaks one iteration (in the loop), if a specified condition occurs and continues with the next iteration in the loop.

Example 1:

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

int main() {
  for (int i = 0; i < 10; i++) {
    if (i == 4) {
      continue;
    }
    cout << i << "\n";
  }
  return 0;
}
```

Example 2:

```
// program to calculate positive numbers till 50 only
// if the user enters a negative number,
// that number is skipped from the calculation

// negative number -> loop terminate
// numbers above 50 -> skip iteration

#include <iostream>
using namespace std;

int main() {
    int sum = 0;
    int number = 0;

    while (number >= 0) {
        // add all positive numbers
        sum += number;

        // take input from the user
        cout << "Enter a number: ";
        cin >> number;

        // continue condition
        if (number > 50) {
            cout << "The number is greater than 50 and won't
be calculated." << endl;
            number = 0;   // the value of number is made 0
again
            continue;
        }
    }

    // display the sum
    cout << "The sum is " << sum << endl;

    return 0;
}
```

Example 3:

```
// nested for loop

#include <iostream>
using namespace std;

int main() {
    int number;
    int sum = 0;

    // nested for loops

    // first loop
    for (int i = 1; i <= 3; i++) {
        // second loop
        for (int j = 1; j <= 3; j++) {
            if (j == 2) {
                continue;
            }
            cout << "i = " << i << ", j = " << j << endl;
        }
    }

    return 0;
}
```