The flow of program control is specified by control statements in C#. It includes the following −

**if statement**

An if statement consists of a boolean expression followed by one or more statements.

The following is the syntax −

if(boolean\_expression) {

/\* statement(s) will execute if the boolean expression is true \*/

}

**if-else statement**

An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

The following is the syntax −

if(boolean\_expression) {

/\* statement(s) will execute if the boolean expression is true \*/

} else {

/\* statement(s) will execute if the boolean expression is false \*/

}

**for loop**

It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

The following is the syntax −

for ( init; condition; increment ) {

statement(s);

}

**while loop**

It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.

The following is the syntax −

while(condition) {

statement(s);

}

**do…while loop**

It is similar to a while statement, except that it tests the condition at the end of the loop body.

The following is the syntax −

do {

statement(s);

} while( condition );

//single selection  
if (i > 0)  
    Console.WriteLine("The number {0} is positive", i);  
//if-then-else selection  
  
if (i > 0)  
    Console.WriteLine("The number {0} is positive", i);  
else  
    Console.WriteLine("The number {0} is not positive", i);  
  
//multicase selection  
if (i == 0)  
    Console.WriteLine("The number is zero");  
else if (i > 0)  
    Console.WriteLine("The number {0} is positive", i);  
else  
    Console.WriteLine("The number {0} is negative", i);

The variable i is the object of evaluation here. The expression in an if statement must resolve to a boolean value type.   
  
// Compiler Error  
if (1)  
    Console.WriteLine("The if statement executed");  
Console.ReadLine();  
  
When the C# compiler compiles the preceding code, it generates the error "Constant value 1 cannot be converted to bool."

hows how conditional or (||) and conditional and (&&) operators are used in the same manner.  
  
**Listing 5.24: If-Then-Else Example 2**  
//Leap year  
int year = 1974;  
if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)  
    Console.WriteLine("The year {0} is leap year ", year);  
else  
    Console.WriteLine("The year {0} is not leap year ", year);

string day = "Monday";  
Console.WriteLine("enter the day :");  
day = Console.ReadLine();  
  
switch (day)  
{  
    case "Mon":  
        break;  
    case "Monday":  
        Console.WriteLine("day is Monday: go to work");  
        break;  
    default:  
        Console.WriteLine("default");  
        break;  
}  
  
switch (strVal1)  
{  
    case "reason1":  
        goto case "reason2"; // this is a jump to mimic fall-through  
    case "reason2":  
        intOption = 2;  
        break;  
    case "reason 3":  
        intOption = 3;  
        break;  
    case "reason 4":  
        intOption = 4;  
        break;  
    case "reason 5":  
        intOption = 5;  
        break;  
    default:  
        intOption = 9;  
        break;  
}

**Do-While**  
The while loop allows the user to repeat a section of code until a guard condition is met. Listing 5.27 presents a simple while loop designed to find out the number of digits in a given value.   
  
**While Example**  
//find out the number of digits in a given number  
int i = 123;  
int count = 0;  
int n = i;  
  
//while loop may execute zero times  
while (i > 0)  
{  
    ++count;  
    i = i / 10;  
}  
Console.WriteLine("Number {0} contains {1} digits.", n, count);

**Do Example**  
//find out the number of digits in a given number  
int i = 0;  
int count = 0;  
int n = i;  
do  
{  
    ++count;  
    i = i / 10;  
} while (i > 0);  
Console.WriteLine("Number {0} contains {1} digits.", n, count);  
  
The do-while construct checks the condition at the end of the loop. Therefore, the do-while loop executes at least once even though the condition to be checked is false from the beginning.

**For**  
The for loop is useful when you know how many times the loop needs to execute. An example of a for statement is presented in Listing 5.29.  
  
  
  
//For loop with break and continue statements  
for (int i = 0; i < 20; ++i)  
{  
   if (i == 10)  
       break;  
   if (i == 5)  
       continue;  
   Console.WriteLine(i);  
}

**The output of the code in the listing is as follows:**  
  
0  
1   
2   
3   
4   
6   
7   
8   
9

**ForEach**  
The foreach statement allows the iteration of processing over the elements in arrays and collections. Listing 5.31 contains a simple example.  
  
**Listing 5.31: ForEach Example 1**  
//foreach loop  
string[] a = { "Chirag", "Bhargav", "Tejas" };  
foreach (string b in a)  
Console.WriteLine(b);  
  
Within the foreach loop parentheses, the expression consists of two parts separated by the keyword in. To the right of in is the collection, and to the left is the variable with the type identifier matching whatever type the collection returns.   
  
  
**ForEach Example 2**  
Int16[] intNumbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };  
foreach (Int16 i in intNumbers)  
{  
   System.Console.WriteLine(i);  
}  
  
Each iteration queries the collection for a new value for i. As long as the collection intNumbers returns a value, the value is put into the variable i and the loop will continue. When the collection is fully traversed, the loop will terminate.

**Break**  
The break statement, used within for, while, and do-while blocks, causes processing to exit the innermost loop immediately. When a break statement is used, the code jumps to the next line following the loop block, as you'll see in Listing 5.34.  
  
**Listing 5.34: Break Example**  
while (true)  
{  
    //...  
    if (x == 0)  
        break;  
    //...  
}  
Console.WriteLine("break");  
 **Continue**  
The continue statement (shown in Listing 5.35) is used to jump to the end of the loop immediately and process the next iteration of the loop.   
  
**Continue Example**  
int x = 0;  
while (true)  
{  
    //...  
    if (x == 0)  
    {  
        x = 5;  
        continue;  
    }  
    //...  
  
    if (x == 5)  
        Console.WriteLine("continue");  
    //...  
}  
  
**Return**  
The return statement is used to prematurely return from a method. The return statement can return empty or with a value on the stack, depending upon the return value definition in the method (Listing 5.36 shows both). Void methods do not require a return value. For other functions, you need to return an appropriate value of the type you declared in the method signature.   
  
**Return Example**

void MyFunc1()  
{  
// ...  
if(x == 1)  
return;  
// ...  
}  
  
int MyFunc2()  
{  
// ...  
if(x == 2)  
return 1919;  
// ...  
}