**Lecture 3 – Programming constructs**

**Conditional Statement**

C# conditional statements allow you to branch your code depending on certain value of expression or condition. C# has two constructs for branching code – the*if*statement and the *switch* statement.

**If/else construct**

The*if* statement allows you to test whether a specific condition is met or not. The syntax for declaring if statement as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | if (condition)     Statement  else     statement |

For example, in the following program, we want to see whether the string you are working with is longer than zero character or not:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | staticvoid Main(string[] args)  {  string str = "welcome home";  if (str.Length > 0)  Console.WriteLine("length is > 0");  else  Console.WriteLine("length is < 0");  Console.ReadKey();  } |

You can also use an *if* statement without final *else* statement:

|  |  |
| --- | --- |
| 1  2  3  4  5 | int data=10;  if (data !=5)   {  Console.WriteLine("value is not equal");   } |

http://2we26u4fam7n16rz3a44uhbe1bq2.wpengine.netdna-cdn.com/wp-content/uploads/040113_1125_CCoreProgra8.png

**Note:** It is not required to use curly braces on only one *if* statement.

http://2we26u4fam7n16rz3a44uhbe1bq2.wpengine.netdna-cdn.com/wp-content/uploads/040113_1125_CCoreProgra9.png

If more than one statement is to be executed as part of either condition, these statements will need to be joined together into a block using curly braces ({…}) as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | bool isFalse;  int data;    if (data == 0)   {   isFalse = true;  Console.WriteLine("data is 0");   }  else  {   isFalse = false;  Console.WriteLine("data is not 0");  } |

You can also combine the *elseif* statement to test multiple conditions. For example, we are checking the highest number of three as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | int x=10, y=20,z=22;  if (x >y)   {  Console.WriteLine("x is Highest");   }  elseif (x > z)   {  Console.WriteLine("x is Highest");   }  elseif (y > z)   {  Console.WriteLine("y is Highest");   }  else   {  Console.WriteLine("z is Highest");   } |

http://2we26u4fam7n16rz3a44uhbe1bq2.wpengine.netdna-cdn.com/wp-content/uploads/040113_1125_CCoreProgra10.png

**Note:** the expression in the *if* clause must evaluate to a Boolean (true / false)

http://2we26u4fam7n16rz3a44uhbe1bq2.wpengine.netdna-cdn.com/wp-content/uploads/040113_1125_CCoreProgra11.png

**Switch construct**

The switch statement allows you to handle program flow based on predefined sets of conditions. It takes a switch arguments followed by a series of case clause. The syntax of switch construct as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | switch(argument)  {   case 1:   // Do anything   break;   case 2:   // Do anything   break;   default:   // Do anything   break;  } |

When the expression in the switch argument is evaluated, the code immediately following the *case* clause executes and you mark the end of statements by the*break* clause. If the expression evaluates to none of the other clauses, then you can include *default* clause.

http://2we26u4fam7n16rz3a44uhbe1bq2.wpengine.netdna-cdn.com/wp-content/uploads/040113_1125_CCoreProgra12.png

**Note:** the order of case doesn’t matters; you can put the *default* case first.

http://2we26u4fam7n16rz3a44uhbe1bq2.wpengine.netdna-cdn.com/wp-content/uploads/040113_1125_CCoreProgra13.png

The following example performs some short math operations like addition, multiply, etc. you can ask to the user at run time in form of choices what operation he want to do. Then we read two numeric values from the console and execute the operation as selected earlier.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | Staticvoid Main(string[] args)   {  int x,y;  Console.WriteLine("Enter two Integers");   x = int.Parse(Console.ReadLine());   y = int.Parse(Console.ReadLine());  Console.WriteLine("Operations\n-----------------------------------\n");  Console.WriteLine("1= Addition\n2= Subtraction \n3= Multiplication");  Console.WriteLine("Operations\n-----------------------------------\n");  Console.Write("Enter the operation code::");  int op = int.Parse(Console.ReadLine());  switch (op)   {  case 1:  Console.WriteLine("Add=" + (x + y));  break;  case 2:  Console.WriteLine("Subs=" + (x - y));  break;  case 3:  Console.WriteLine("Multiply=" + (x \* y));  break;  default:  Console.WriteLine("wrong choice");  break;   }  Console.ReadKey();   } |

If all the *case* clauses are executed in a sequence, but you want to fire a particular*case* clause early, then you can do this by the *goto case* clause as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | switch (country)   {  case"india":  goto case"Canada";  case"USA":  break;  case"Canada":  break;   } |

**Loops**

Loops are essential encounters in C# which allow you to execute a block of code repeatedly until a certain condition is met. C# caters the following four constructs as;

* *for* loop
* *foreach* loop
* *while* loop
* *do/while* loop

**The for loop construct**

The *for* loop used in case of iterating a block of code to fixed number of time where you can test a particular condition before you perform another iteration. The *for* loop construct syntax is as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | for (initializer; condition; iterator)  {   Statements;  } |

Here the initializer is evaluated before the first loop is executed. The loop is executed until condition expressions are false and the iterator is responsible for incrementing of decrementing the loop counter.

The following program typically writes the value 0 to 4 on the screen by using the *for*loop construct. We are declaring a local variable x and initialize it to 0 which is used as loop counter. Then we test whether it is less than 5 or not in the condition block and finally you increase the counter by one and walk through the process again as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5 | int x;  for (x = 0; x < 5; x++)   {  Console.WriteLine(x);   } |

Note that we can declare the local counter variable in the *for* loop block, also, and x++ can be simply written as x=x+1. You can create complex condition, endless loops and make use of *goto*,*break* and *continue* statements in the for loop. The program will terminate in between if the counter variable is reached to 8 as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | for (int i = 0; i < 10; i++)   {  if (i == 8)   {  break;   }  Console.WriteLine(i);   } |

In some circumstances you want your loops to be executed infinitely.ThenC# allows you to write infinite loops by using the following syntax as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | for (; ;) { }    for (; ;)   {  Console.WriteLine("printing");   } |

The *for* loop can be nested.That means we implement an inner for loop into outer loop. The inner loop executes once completely for each iteration of an outer loop. The following code displays prime numbers till 100 by using nested for loop.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | staticvoid Main(string[] args)   {  int i, j;  bool isPrime = false;    for (i = 2; i <= 100; i++)   {  for (j = 2; j < i; j++)   {  if (i % j == 0)   {   isPrime = true;  break;   }   }  if (isPrime == false)  Console.Write("{0} ", j);  else   isPrime = false;   }  Console.ReadKey();   } |

**The foreach loop construct**

The *foreach* loop construct allow you to iterate through each item into a collection without the need to test the upper limit condition. This construct can also be applied on user-defined collections. In the following, you can traverse an array of string as;

|  |  |
| --- | --- |
| 1  2  3  4  5 | string[] country = { "India", "USA", "Canada", "China" };  foreach (string x in country)   {  Console.WriteLine(x);   } |

Here, *foreach* loop steps through the array an element at a time. With each element, it places the value of the element in the string x variable and then performs an iteration of the loop.

**The while loop construct**

The while loop is mostly used to repeat a statement or a block of statements for a number of times that is not known before the loop begins. The syntax for while loop as follows:

while(condition)

Statement;

The while loop performs the expression evaluation first, then executes its body statements. A statement inside the loop body will set a Boolean flag to false on certain iteration till the end of the loop as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | bool status= false;  while (!status)   {   } |

The following program checks whether the value of *int* x is less than 10 using while loop and display them.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | int x = 0;  while (x<10)   {  Console.WriteLine("value of x is"+ x);   x++;   } |

**The do/while loop constructs**

The do/while is almost similar to its previous construct, the while loop.But ithas slight differences like the condition that is evaluated after the body of the loop has been executed. This loop is useful for the situation in which statements must execute at least once. The syntax for the do/while loop as follows:

do

{

//body

} while(condition);

The following program depicted has the same objective as in while loop but it executed the body statements first and later it checks the condition.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Int x = 0;  do   {  Console.WriteLine("value of x is" + x);   x++;   } while (x < 10); |

**Jump Statement**

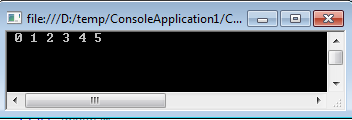
C# caters special statements that allow you to jump immediately to another line in the program. The *break*, *continue* and *goto* statements are known as jump statements.

**The break statement**

The *break* statements mainly used to exit the current iteration containing in *for*,*foreach*, *while* and *do.while* loops. The following example shows the break statements to exit a nested *for* loops as following.

|  |  |
| --- | --- |
| 1  2  3  4  5 | for (int i = 0; i < 10; i++)   {  Console.Write(" "</span> + i);  if (i == 5) break;   } |

Here the *for* loops stops its execution when the value of *i* is reached to 5 using the*break* statement. The output of this program isas follows:



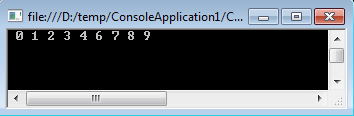
**Image 1.6 Break statements Output**

**The continue statement**

The *continue* statements is to break statements and must also be used in *for*,*foreach*, *while* and *do.while* loops. It stop the current execution and begin or restart a new iteration, it does not exit from the current iteration as like *break* statement.

|  |  |
| --- | --- |
| 1  2  3  4  5 | for (int j = 0; j < 10; j++)   {  if (j == 5) continue;  Console.Write(" "+j);   } |

The above program does not stop the its execution when the j value reached to 5 instead it restarts the loop and continue printing the rest of output as follows:



**Image 1.7 Continue Statement output**

**The goto statement**

The *goto* statement allows you to jump directly to another specific line in the program indicated by a *label* identifier. This statement is quite handy jumping between *cases* in a switch statement. The following program declares a label identifier lableX to display the incremented value of x each time jumped from the condition as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5 | int i = 0;  lableX: Console.WriteLine("label construct"+ i++);    if (i < 5)  goto lableX; |

**Questions:**

1. **What conditional operators do You know?**
2. **Why do You need conditional operators?**
3. **What type of cycles do You know?**
4. **Why programmers don’t recommend to use GoTo statement?**
5. **What continue statement does?**