Top of Form



**Introduction to C# Programming: Lesson 6**

**Chapter 1**

**Introduction**

At this point, your programs, while challenging to write, have not been extremely useful. You have learned input, output, and how to do calculations. By writing methods, you are able to group your code into logical blocks that can be reused. Yet, all of the programs so far have only used the sequence programming structure. You have listed a set of instructions and the computer has followed each step in the order listed. Essentially, these programs have just been fancy, specialized calculators. The ability for a computer program to make decisions is the reason people think computers are so smart. For example, think about a bank ATM. This computer allows users to access money, provided that they enter the correct password for the account. If the user does not enter the correct password, then the machine must be sure not to dispense any money. So, you can see how important and common the decision structure is to programmers.

In this lesson, you will learn about the if selection structure. This structure will give your program the option of choosing which set of code to run, based on some condition. You will also learn about Boolean expressions and the different conditional operators that are available to you in C#.   
  
  
  
  
**Chapter 2**

**The if Statement**

The idea of a selection structure is one that you use every day. For example, imagine yourself driving home one afternoon and the little light on your dashboard turns on showing that you are low on gas. At this point, you must decide if you should stop to get gas. If you are close to home, you might decide to try to make it. On the other hand, if you have a long distance to travel, you might decide to stop and fill up. This is just one of many decisions that you might make on the way home. You will also have to decide which roads to take, which lane to drive in, whether or not to run that yellow light.

Each one of these decisions has a condition that can be answered either with a yes or a no. "Should I stop and get gas?" The answer is either yes or no. I realize that you may be thinking, sometimes that answer to that question is maybe. And I agree with that, but ultimately the decision to stop for gas is either yes or no. So, if the answer to the question is yes, then you will stop your car at the gas station and get gas. If the answer is no, then you can proceed to your house.

In a similar way, the computer can also evaluate a condition and then either do a set of instructions or not. One important thing to note about the computer's decision-making process: The computer can only evaluate yes or no questions. Therefore, when you are writing your programs, you must be sure to have conditions that result in either yes or no. Of course, when we talk about writing programs, we talk about our questions having answers of true or false.

Writing an *if statement* in C# is very natural. That is, it is very similar to reading a sentence in English. The format is as follows:

*if ( <condition> )  
  
statement;*

The computer will read the first line and determine if the condition is true or false. If that condition is true, then the next statement is executed.

Notice that the end of the first line does not end in a semicolon. This is not a typo. You may recall in earlier lessons how I made a big point about every statement in C# ending with a semicolon. Now it would seem that I am contradicting myself. However, this is not the case. Look again at the statement. This is really only the first part of a statement. That is, I have only asked a question here. You would never go up to someone and say, "If you need money for lunch today." That person would be waiting for the end of your sentence. Maybe you were going to buy lunch for them, or maybe you were going to give them your lunch, or maybe you were going to tell them to go hungry today. The end of the sentence is very important. Likewise, in C#, the end of the sentence is also important.

Also notice how the line following the if line is indented. Remember, white space does not matter to the compiler. This indenting is done to make your code easier to read.

The next thing you must learn is what kinds of things you can put in the condition part of the if structure. Remember that the condition must evaluate to either true or false. The following table shows the six *comparison operators*, or *relational operators*.

|  |  |
| --- | --- |
| **Table 6.1. C# comparison operators** | |
| **Operation** | **Operator** |
| less than | < |
| less than or equal to | <= |
| greater than | > |
| greater than or equal to | >= |
| equal to | == |
| not equal to | ! = |

There are a couple of things to note about these operators. First, when you use operators that have two symbols, be sure to type the characters with no spaces between them. For example, if you accidentally put a blank space between the less than and equals signs, the program will not compile. Second, remember that the equality conditional operator needs two equals signs. This is to distinguish it from the assignment operator (=). If you accidentally forget to put the second equals sign inside your condition, your program will not compile. For example:

if ( a = 1)

will give a compiler error. This statement should be written as:

if (a == 1)

In C#, you can use these comparison operators to compare whole numbers, decimal numbers, and characters. The following are examples of the if structure. Note that these are statements that could be placed inside any function in a program. The details of the program have been left out for brevity. Assume that the program does have the following variables:

int i = 1;  
char c = 'A';

Example 1:

if ( i == 2)  
Console.Out.WriteLine("True");

This code segment checks to see if the value of the variable i, which is 1, is equal to the value 2. As we all know, this condition is false. Therefore, the next statement would be skipped and program flow would continue with the next statement after the WriteLine statement.

Example 2:

if (c <= 'A')  
Console.Out.WriteLine("True");

This code segment compares the value of the variable c to the value of the character *A*. When C# is asked to compare characters, the Unicode value is compared. Therefore, the Unicode value of variable c's value, 65, is compared to the Unicode value of the character *A*, which is also 65. Since these two values are equal, the condition is true and the next statement is executed. It is not important for you to remember the Unicode value for *A* or any other character. It is useful for you to remember that in Unicode, all of the capital letters come before any of the lowercase letters. That means that *A* is less than *a*.

There will be times when you will want to make comparisons with strings. Remember that a string is a class and therefore, string variables behave a little differently. This concept is a little more advanced, and I would prefer you to get more experience with the basic selection structure. Therefore, we will explore string comparisons in the next lesson.  
  
  
  
  
**Chapter 3**

**The else Statement and Statement Blocks**

In the last chapter, you saw how to direct the computer to do something if the result of the condition was true. However, oftentimes you will want the computer to perform one set of tasks if the result is true and a different set of instructions if the result is false. Take the example of driving home again. Suppose on your way home, you get hungry and decide to get some fast food. At this point, you will have to decide if you want a hamburger or if you want chicken. You might ask yourself, "Do I feel like a hamburger today?" If the answer to the question is yes, then you will order a hamburger. However, if the answer to the question is no, then you will order chicken. Notice now that there is only one question, and by answering that question, you will follow only one of two choices. That is, you will only order a hamburger or chicken, but not both.

In C#, this is known as an *if . else statement*. Although it could be done with two if statements, it is so common that C# allows you to write what is called an else statement. So, the else statement works in combination with the if statement. The format is as follows:

*if ()  
true statement;  
else  
false statement;*

When the computer encounters an if statement, it will evaluate the condition. If this condition is true, then the next statement is executed. However, if the statement is false and there is an else statement, then the statement after the keyword 'else' is executed. It is important to remember that you can have an if statement without an else, but you cannot have an else statement without an if.

Example:

if (age >= 18)

Console.Out.WriteLine("You are old

enough to vote.");

else

Console.Out.WriteLine("You are not old

enough to vote.");

This code segment might be part of a function that prompts the user to enter their age. The code you see above could then be used to let the user know if they are old enough to vote or not. The first line compares the value of the age variable to the value 18. If the value is greater than or equal to 18, then the program will display a message telling the user they are old enough to vote. However, if the value is less than 18, then a message tells the user that they are not old enough to vote.

Again, notice how the if . else structure is very natural and easy to read. You should also notice how the statement inside the false path is indented. This makes your program easier to read.

At this point, you may be quite impressed with C#'s ability to make decisions, but one question usually comes up. Students often point out that it is quite limiting to be able to execute only one statement for the true path and one statement for the false path. Oftentimes you will have many steps that need to be taken based on the result of the decision. We could just take all of those statements, put them into their own function, and proceed that way. Sometimes this will make sense and you will want to do just that. However, we have another option.

A *statement block* can be created, which allows the programmer to have more than one statement executed inside an if or an else statement. Statement blocks are created by enclosing all of the statements in a set of braces ({ }). For example, suppose in the voting example above, you not only want to tell the user whether they are old enough to vote, but you also want to set the value of a Boolean variable to either true or false, based on whether they are old enough or not. For this, we could do the following:

if (age >= 18)

{

Console.Out.WriteLine("You are old enough to

vote.");

oldEnough = true;

}

else

{

Console.Out.WriteLine("You are not old enough to

vote.");

oldEnough = false;

}

In this example, you will see that if the value of the age variable is greater than or equal to 18, then both a message is displayed to the user and a Boolean variable's value is set. Again, notice the indenting. You will find as your programs get longer and as the number of statements inside your program blocks get longer, the need for proper indenting becomes more important.

This example shows statement blocks in both the if clause and the else clause. However, it should be mentioned that you can have a statement block in the if clause or the else clause, both, or neither. Another question beginning programmers often have is whether they can use the braces around only a single statement. For example:

if (a != 1)

{

Console.Out.WriteLine("a does not equal 1");

}

This is perfectly legal in C#. However, I usually recommend that beginning programmers do not do this. The reason is because, although it has no effect on how the program runs, there will be serious problems if you forget the closing brace. If you leave out the closing brace, the compiler will go looking through your program for it and will usually find the brace at the end of the function. It then mistakes the closing brace for the class for the closing brace for the function. That means that there is no closing brace for the class and the program does not compile. This can get quite confusing, and nobody needs more confusion in their life.  
  
  
  
  
**Chapter 4**

**Logical Operators**

Sometimes when you are writing your programs, you will want to evaluate two or more conditions before performing any instructions. Take the voting example again. Being 18 years old is not the only condition that must be met in order to vote in an election. You also must be registered to vote. That means that our program would need to evaluate two different conditions, 18 or older and registered to vote. This could be done using two if statements, but there is an easier way. Instead, we can write a *compound condition*.

In order to write a compound condition, you must use a logical operator. C# provides logical operators for *And* and for *Or*. The And operator in C# is two ampersand symbols (&&) and the Or operator is two pipe symbols (||). On a standard keyboard, the pipe symbol (|) is on the same key as the backslash character (\).

In order to evaluate a compound condition using a logical operator, the computer follows what is called a *truth table*.

|  |  |  |
| --- | --- | --- |
| **Table 6.2. Truth table for the And operator** | | |
| **Value of Condition 1** | **Value of Condition 2** | **Value of Condition 1 && Condition 2** |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

You will see that for our compound condition, there are two conditions, each of which will evaluate to either true or false. In the first two columns, the table lists all possible combinations of true and false for the two conditions. The final column shows the result of the first condition and the second condition. By closely inspecting this table, you will see that both condition 1 and condition 2 must be true in order for the compound condition to evaluate to true.

This would be the operator that we would choose to use for the previous voting example. That is, assuming there was a char variable named 'registered' that stored either 'Y' or 'N,' we would write:

if (age >= 18 && registered == 'Y')

In addition to the And operator, C# also provides the Or logical operator.

|  |  |  |
| --- | --- | --- |
| **Table 6.3. Caption truth table for the Or operator** | | |
| **Value of Condition 1** | **Value of Condition 2** | **Value of Condition 1 || Condition 2** |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

As you can see, a compound condition using Or results in true if one or both of the conditions is true. Another way to look at this is that the Or operator always results in true, unless both conditions are false.

For example, imagine if you asked the user if they wanted to perform some calculation, and you asked them to answer with a *y* or an *n*. As you know, people don't always follow directions, so you want to make sure that your code runs even if the user enters *Y*. Assuming that you had a char variable named 'answer,' you could write:

if (answer == 'Y' || answer == 'y')

The following complete program demonstrates all of the principles that were taught in this lesson to compute the user's weekly paycheck. The program will prompt the user for the number of hours worked and their hourly rate. It then determines whether the employee qualifies for overtime pay. The weekly pay is calculated and displayed for the user. Finally, the user is asked if they are happy with their check, and based on the answer, a message is displayed. You might want to try to type it in to make sure that you understand all of the concepts.

using System;

class SelectionExample

{

public static void Main( )

{

string strHours, strRate, strAnswer;

double hours, rate, pay;

char answer;

Console.Out.Write("How many hours did ¬

you work last week? ");

strHours = Console.ReadLine();

hours = Convert.ToDouble(strHours);

Console.Out.WriteLine();

Console.Out.Write("What is your hourly ¬

rate? ");

strRate = Console.ReadLine();

rate = Convert.ToDouble(strRate);

if (hours <= 40.0)

pay = hours \* rate;

else

{

pay = hours \* rate;

// Calculate overtime pay as time

// and a half for all

// hours over 40

pay = pay + ( (hours - 40) \* ¬

(rate / 2.0) );

}

Console.Out.WriteLine();

Console.Out.WriteLine("Your weekly pay ¬

is: " + pay);

Console.Out.WriteLine();

Console.Out.WriteLine("Are you happy ¬

with your pay? ");

strAnswer = Console.ReadLine();

answer = Convert.ToChar(strAnswer);

if (answer == 'y' || answer == 'Y')

Console.Out.WriteLine("I'm glad ¬

that you are happy!");

else

Console.Out.WriteLine("Maybe next ¬

week you can work more hours");

}

}

One final operator that I want to talk about is the Not operator, which is simply a single exclamation point (!). This operator is not used very often, but it is important that you know that it exists. You can use the Not operator to make a false condition evaluate to true and a true condition evaluate to false. This operator, unlike the And and the Or operators, acts on a single condition.

For example, maybe you ask your user if they want to perform a calculation. You could have the statement:

if ( !(answer == 'y') )

Now, I realize that it would be much easier to use the not equal relational operator (!=). However, I am trying to make the point that this is available to you. There may be a time when you write a very complex compound condition and it evaluates exactly wrong. That is, when you expect it to evaluate to true, it comes up false and vice versa. You may have considerable trouble rewriting the condition. Instead of doing that, you could just put the Not operator in front of the condition and you are done!  
  
  
  
  
**Chapter 5**

**Summary**

In this lesson, you have learned how to make the computer perform one of the most powerful tasks, decisions. You started by thinking about decisions you make in life everyday. Many of these decisions ended up being answered with either a yes or a no. These are the only questions that a computer can make a decision on. Programmers usually think of these questions as true or false questions. You next learned the syntax of an if statement. You also learned that you can write an else clause; that way, you can specify what the computer should do if the condition evaluates to true and what it should do if it evaluates to false. The computer will perform either the statements in the true path or the false path, but not both.

You also learned that C# allows you to do only one statement in each of the paths, but that we can specify that more statements can be executed by writing a statement block. The lesson finished up with a discussion of the And and the Or logical operators.

The next lesson will also focus on the selection structure. We will look a little more at the if structure and talk about nested ifs. That is, an if statement inside another if statement. You will also learn about another selection structure, the switch structure. This structure will allow us to more easily write code to make multiple decisions.  
  
  
  
**Supplementary Material**

|  |
| --- |
| [C# Control Statement](http://www.csharp-station.com/Tutorials/Lesson03.aspx)  http://www.csharp-station.com/Tutorials/Lesson03.aspx |
| This page gives a different description of the if statement. It also includes a discussion of the switch statement, which will be covered later. |

**FAQs**   
  
**Q:** I typed in my if statement as follows and it never evaluates to true, why?  
  
int a = 0;  
if (a == 1);  
Console.Out.WriteLine("a is equal to 1");  
  
  
**A:** I agree this code appears as if it should not print out the message that says "a is equal to 1", but it does. The reason is because you included a semicolon at the end of the if statement. Remember, the if statement does not end with a semicolon. What happened was that the condition evaluates to false and then the next statement is skipped. That next statement will end with a semicolon correct? Well, the next statement says to do nothing. Let me indent your code another way and see what you think:  
  
  
int a = 0;  
if (a==1)  
;  
Console.Out.WriteLine("a is equal to 1");  
  
  
See the difference?

**Assignment**   
  
  
Write a program that will display the user's letter grade based on their numerical average. Prompt the user for a number, and then display the corresponding letter grade based on the 10-point scale:

A 90 - 100  
B 80 - 89.9  
C 70 - 79.9  
D 60 - 69.9  
F < 60.0="">

[Click here for solution: **LetterGrade.zip**](https://api.ed2go.com/CourseBuilder/2.0/images/resources/prod/cpb-0/LetterGrade.zip)

|  |
| --- |
| Copyright © 1997 - 2012 Education To Go. All rights reserved. The material on this site cannot be reproduced or redistributed unless you have obtained prior written permission from Education To Go. Education To Go and ed2go are registered trademarks of Education To Go, a part of Cengage Learning. |

Bottom of Form