Top of Form



**Introduction to C# Programming: Lesson 7**

**Chapter 1**

**Introduction**

Now that you know how to make your programs make a decision and proceed with the appropriate set of instruction based on that decision, you can write very powerful programs. However, up to this point, you have only learned how to compare values of simple data types, numbers, and characters. This could be a problem, though, because there are many times when you will want to compare two string values. Since strings are objects, we cannot directly compare them. Therefore, in this lesson you will first learn how to use string functions to help us compare strings.

Next in the lesson, I will address a question that some of you may have: "Can I put an if statement inside another if statement?" For example, maybe I want to ask two different questions and then run a different set of code based on every combination of the results. Well, the answer to the questions is, of course! This is called *nested if statements*. You will learn about nested if statements, when they are appropriate, and how to code them.

The lesson will finish up with a discussion of a second form of the selection structure called the *switch statement*. As you will see, switch statements are sometimes useful as a replacement for a nested if structure.  
  
  
  
  
**Chapter 2**

**Comparing Strings**

Up to this point, all of our comparisons have dealt with values of simple data types. Recall from earlier in the course that our number variables had a fixed amount of space allocated to them in memory. By fixing the amount of memory, C# also limits the largest and smallest values. However, string variables do not have a fixed amount of memory allocated to them. That is because our string variables can hold one character, three characters, or 103 characters.

The reason that strings can have an arbitrary size and hold an arbitrary number of characters is because strings are objects. *Objects* are variables that are created from classes. These variables are things that can hold more than one value, and they can have methods associated with them. You will learn a lot more about classes and objects later in the course. For now, just realize that they are much different than simple variables.

Since strings are objects, they behave differently than other variables. Let me show you with an example:

String s1 = "Adam";

String s2 = "Barbara";

if (s1 < s2)

Console.Out.WriteLine(s1 +

" comes before " + s2);

If this code were part of your program, it would not compile. This is because C# does not allow you to make comparisons between objects in such a manner. Instead, you should use the value-returning function named *Compare()*. The syntax for calling this function is as follows:

*String.Compare( , )*

This function will compare the values of the strings, character by character, using the character's Unicode value. In Unicode, as you learned earlier in the course, the letters are in alphabetical order with all of the capital letters coming before all the lowercase letters. The following table summarizes the possible results of the comparison.

|  |  |
| --- | --- |
| **Table 7.1. Summary of the Compare() function from class String** | |
| **Comparison of s1 and s2** | **Value Returned from String.Compare(s1, s2)** |
| s1 comes before s2 | an int less than zero |
| s1 is the same as s2 | the int zero |
| s1 comes after s2 | an int greater than zero |

At first, the return values of Compare() may be hard to remember. The way I always remember it is that if the two values are the same, then zero is returned. If the first value comes before, or is *less* than the second value, then you get a number that is *less* than zero. However, if the first value comes after, or is *greater* than the second value, then you get a number that is *greater* than zero.

Therefore, using the Compare() function to rewrite the code segment above correctly, you would type:

String s1 = "Adam";

String s2 = "Barbara";

if (String.Compare(s1, s2) < 0)

Console.Out.WriteLine(s1 + " comes

before " + s2);

If you do a little research into class String, you will find that there are many different ways to use Compare. One very useful version of Compare() will compare the characters, ignoring the case of the letters. That means that *A* would be treated as equal to *a*. The syntax for this function is as follows:

*String.Compare( , , )*

You will notice that the syntax is almost the same as the previously discussed version of Compare() except that this function also takes a Boolean value. If you put *true* in for *ignore case*, then the case is ignored. However, if you put in *false*, then this version of Compare() will work the same as the one we previously discussed. For example:

if (String.Compare("A", "a", true) == 0)

Console.Out.WriteLine("Equal");

This program segment would display the word *Equal* because the true argument tells the function to ignore the case of the characters. As stated earlier, there are many other versions of the Compare() function, but for simplicity, they will not be addressed here.

Another useful function provided by class String is the Equals() function. Although you can test for equality using the Compare() function, it would be better to use Equals() for two reasons. The first reason is because using Equals() makes your code easier to read. And remember that the easier your code is to read, the easier it will be to debug. Second, you will later find that most classes have an Equals() function defined, which allows for comparison between two objects.

The Equals() function works very naturally. That is, if the two objects contain the same value, then the function returns true. Otherwise, the function will return false. So, to test for equality of our two string variables from the earlier example, you would type:

if (String.Equals(s1, s2))

There is one last point I want to make about testing for equality between strings. You may find out by accident that C# allows you to test for equality and inequality using the == or != operators. While this may seem like a better way to code, because there is less to type or remember, it is not good programming style. In general, you cannot use these operators on objects, and therefore, you would be wise not to get used to using them. C# has provided this ability as a convenience, but really, it is making for dangerous programming. Another reason that you may not want to use these operators is because other programming languages such as C++ and Java do not allow the use of them. So, if you get used to using them in C# and decide later to program in another language, you will be setting yourself up for problems.  
  
  
  
  
**Chapter 3**

**Nested if Statements**

Now that you have learned how to compare strings, I want to turn your attention back to different uses of the selection structure. I mentioned earlier that there might be a point in time when you want to ask multiple questions. Using the example of voting from the previous lesson, maybe you want to ask the user for their age and if they are registered. You may recall that we used a compound condition to test for this. However, maybe you wanted to display one message to people who were old enough and registered, another message for people who were old enough but not registered, and a third message to people who were too young. This could be done with multiple if statements, each containing a compound condition. In this example, the age of the user is stored in an int variable named *age*, and their registration status is stored as either *Y* or *N* in a char variable named *registered*.

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

Console.Out.WriteLine("You are ready to

vote.");

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

Console.Out.WriteLine("You must register

before you can vote.");

if (age < 18)

Console.Out.WriteLine("Too young.");

Although this will work, you can see that it is a lot of work. There is quite a bit of thinking and quite a bit of typing. And the more thinking and typing there is, the more likely there is for an error in logic or syntax to occur. This also hurts performance, because if the user is old enough and they are registered, the program still has to evaluate the next two comparisons. Instead, it is better programming practice to put one if statement inside of another. By doing this, you are creating what is called a *nested if* statement. This means that we could rewrite the above code as follows:

if (age >= 18)

if (registered == 'Y')

Console.Out.WriteLine("You are ready

to vote.");

else

Console.Out.WriteLine("You must

register before you can vote.");

else

Console.Out.WriteLine("Too young.");

Notice in this example that the age of the user is tested first. If the user is not old enough to vote, there is no need to test to see if they are registered. The computer would simply skip over the else statement and the program would continue on. However, if the user were old enough, the program would then test to see if the user was registered. This structure makes for very efficient code and code that is much easier to read.

Now some of you may be looking at the above code and thinking, "I thought if you wanted to execute multiple statements, you need to use a block statement." This is exactly correct. However, I want you to remember that all of the code in an if . else statement is all part of the same statement. Recall the example from the previous chapter where I said that you would never walk up to someone and say, "If you don't have enough money for lunch." It is only a partial statement. Well, the same is true in C# for the if . else. Just remember that each statement ends with a semicolon. Therefore, when the computer executes the code in an if . else, exactly one path will be run. Therefore, only one semicolon will be encountered.

With all of this in mind, I want you know that when you are writing code and you come to an example like the one above and you are not sure if it is one statement or multiple statements, always err on the side of caution and put in the braces. I often do this when I am writing code. In fact, when I typed up this example, I actually used the braces to make a block statement out of all of the code after the first *else* and then removed them. Again, the word of caution is to be sure that your braces are in the correct place.

Another point to be made from this example is the indenting. Notice how the nested if statement, the *if* that is inside the *else*, is still indented, and then the code in its true path is indented again. Doesn't this make the code easier to read? If you don't believe me, look at the example again without the indenting.

if (age >= 18)

if (registered == 'Y')

Console.Out.WriteLine("You are ready to vote.");

else

Console.Out.WriteLine("You must register before you

can vote.");

else

Console.Out.WriteLine("Too young.");

Wow, see how much harder it is to understand? Also, imagine if the second else statement were not there. Then it becomes even more confusing, because it may not be obvious which if statement the else statement matches up with. Just remember: An else statement always matches up with the most recent unmatched if statement. Consider the example of logging in to a computer account. In order to do this, your user name must be valid and the password must be correct. For this example, assume that there is a function called *validUser()* that takes a username and returns either true or false based on whether the username is valid or not. Assume also that there is a function called *validPassword()* that takes a password and determines if the password is correct for that user.

if (validUser(username))

if(validPassword(password))

Console.Out.WriteLine("Welcome!");

else

Console.Out.WriteLine("Login not

valid");

For this example, if the user name is valid and the password is valid, then the user is logged in. However, if the user name is valid and the password is not valid, then the user cannot be logged in. But what will happen if the user name is not valid? In this case, no message will be displayed at all. This is because the else statement matches with the most recent if, which is if (validPassword()).

Notice in this example that we really want the else statement to match up with the first if. That is because if the username is not valid, then the login would not be valid, and we would want to display a message to our user. We can get this done by using the block statement around our nested if.

if (validUser(username))

{

if(validPassword(password))

Console.Out.WriteLine("Welcome!");

}

else

Console.Out.WriteLine("Login not valid");

You see in this example that the indenting is still the same. However, the braces create a block statement. Now when the compiler reaches the else statement, it will look over the nested if and find the first if statement, if (validUser(username)).

Therefore, the moral of the story is to be very careful when writing nested if statements. They can be very helpful and even necessary at times. However, you must be careful with the arrangement of your if statements. Sometimes it is even necessary to use block statements to force C# to execute the logic that you have in your mind.  
  
  
  
  
**Chapter 4**

**The switch Statement**

In the previous section, you learned about putting an if statement inside an if statement. Now, I want to push that idea a little more. What about an if inside of an if . inside of an if. Wow, that may seem like something you would never need to consider. But think about a program that will display a message based on the user's letter grade in a class.

|  |  |
| --- | --- |
| **Table 7.2. Letter grades and corresponding messages** | |
| **Letter Grade** | **Message** |
| A | Outstanding |
| C | Average |
| F | Poor |

For this example, I am going to put the code into a function that will receive a character and return the string. If the character received is not one of the values listed, a string containing the word *Error* will be returned.

public static String getMessageIf(char grade)

{

if (grade == 'A')

return "Outstanding";

else

if (grade == 'C')

return "Average";

else

if (grade == 'F')

return "Poor";

return "Error";

}

As you can see by indenting this way, the lines in your code become very long. For this reason, some people choose to write code that is indented like the following:

public static String getMessageIf(char grade)

{

if (grade == 'A' || grade == 'a')

return "Outstanding";

else if (grade == 'C' || grade == 'c')

return "Average";

else if (grade == 'F' || grade == 'f')

return "Poor";

return "Error";

}

Either style is perfectly acceptable. Therefore, you should feel free to use either or both, if you want. You will find that as you have more levels of nested ifs, the second method will become a better choice. In this example, I only used three different messages plus the error message, but you can easily see that there may be times when you will need to use even more.

As your list of different cases gets longer and longer, the if selection structure becomes more and more cumbersome. To make our coding easier to write and easier to read, C# has provided what is called the *switch selection structure*. The syntax of the switch statement is as follows:

switch(<expression> )

{

case <value>:

<statements>

case <value>:

<statements>

. . .

default:

<statements>

}

When the program reaches the switch statement, it compares the expression> to the first value>. If the two are equal, then the statements listed after that case are executed. If the two are not equal, the program will continue with the next case to check for their equality. This will continue until all of the cases have been evaluated.

You may have noticed that I used the word *statements* above. That is, if a case is matched, then all of the statements after the case clause are executed. That means that if the first case is true, the program will run the code for all of the cases that follow. This is something that you may want to do, but sometimes you will not want this to happen. To avoid all of the cases being run, the *break;* statement is used. This statement will cause the program to stop executing where it is and skip to the end of the switch block.

Another point about the switch syntax is the default label. The code listed after default is code that will be run if the expression does not match any of the other cases. This is not required by C#, but I strongly urge you to use this. The reason is because you may write a switch statement and accidentally leave out a case. Maybe you actually left it out, or maybe you mistyped the value. If this should happen in the middle of a long program, then you may never realize that no case was met. However, if you used the default statement to display an error, you would know right away that something was wrong.

Using all of this knowledge about the switch statement, our grade function can be rewritten. The following program uses that function in a complete program that prompts the user for a letter grade and then displays the appropriate message based on that grade. You may want to type in this example and run it a few times to be sure you understand how the switch statement works.

using System;

class Switch

{

public static void Main()

{

String strLetter;

char letter;

Console.Out.WriteLine("Enter your ¬

letter grade: ");

strLetter = Console.ReadLine();

letter = Convert.ToChar(strLetter);

Console.Out.WriteLine("Your grade ¬

is " + getMessageSwitch(letter));

}

public static String getMessageSwitch( ¬

char grade)

{

String message;

switch (grade)

{

case 'a':

case 'A':

message = "Outstanding";

break;

case 'c':

case 'C':

message = "Average";

break;

case 'f':

case 'F':

message = "Poor";

break;

default:

message = ¬

":an error occurred";

break;

}

return message;

}

}

**Chapter 5**

**Summary**

In this lesson, you learned a little more about comparing objects. Specifically, you learned that string objects should be compared to one another using the Compare() and Equals() functions. Additionally, you learned about nesting if statements. You saw that it is possible to put a second if statement in the true path, or the false path, or both paths of another if statement. It is sometimes necessary to write code with many different levels of nesting. When your program has many choices based on some decision, you saw that it was easiest to use the switch statement. Using the switch statement for decisions with multiple choices makes your code easier to both read and write.

In the next lesson, you will learn about the repetition structure, also referred to as a loop. It has been said that if the computer's ability to make decisions is what makes it smart, then its ability to loop is what makes it useful. In the next lesson, you will see that this is indeed true. As you may remember, the repetition structure is the third of the three control structures, with sequence and selection being the other two. After learning about writing loops, you will have the necessary tools to write very significant and useful programs.  
  
  
  
**Supplementary Material**

|  |
| --- |
| [String Functions](http://msdn2.microsoft.com/en-us/library/system.string_members(vs.80).aspx)  http://msdn2.microsoft.com/en-us/library/system.string\_members(vs.80).aspx |
| This link is to the Microsoft Developer's Network (MSDN) site where all the different functions in class String are discussed. |

**FAQs**   
  
**Q:** I am still a little confused about determining which else matches up with which if. Is there an easier way to remember this?  
  
**A:** The best way to remember is to remember the phrase, the else matches with the most recent unmatched if. This series of words may be very confusing at first, especially if you try to say them fast. But slow down and read them one by one, thinking about what each word is referring to. An unmatched if, is an if statement that does not have an else that is associated with it. So, just work your way up the code until you reach an if statement that does not have an else statement. You may also consider using braces and indenting to make the matching clearer.  
  
Remember, making the code easy for a human to read is beneficial to you because if there is an error, your code will be easier to debug.

**Assignment**   
  
  
Write a program that will prompt the user for their year number in school, 1 through 4. Based on this response, your program should display their class rank as Freshman (1), Sophomore (2), Junior (3), Senior (4), or Error (any other number).

To demonstrate your knowledge of the different selection structures, after prompting the user for their year number, also ask them if they want the program to use a nested if structure or a switch structure.

Make your program more organized by doing all the input and output in the Main function, and create two different value returning functions, getRankIf() and getRankSwitch().

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