Notes for Chapter 5

Boolean Expressions

The bool type holds values of either true or false. Boolean variables are central to both selection and iteration control constructs, which are two of the three categories of programming statements. These categories are referred to as the basic programming constructs. The third category is simple sequence.

Boolean Results

Selection statements produce a Boolean result. One form of selection statement, the if statement, includes a conditional expression, enclosed inside parentheses. The conditional expression, sometimes called “the test condition," produces a result of true or false.

Conditional Expressions

Conditional expressions are referred to as the test condition. When you write an expression to be tested, you must first determine the operands. Unless a bool data type is used, two operands are required for equality and relational tests. To construct a conditional expression, after you identify the operands, determine what type of comparison to make.

Equality, Relational, and Logical Tests

To determine whether two values are the same, use the equality operator (==), not the assignment operator (=). The (!=) represents NOT equal. Be careful making equality comparisons with floating-point (float and double) and decimal types. The == and != are overloaded operators. Strings and char use lexicographical (dictionary) order for comparisons. Relational operators allow you to test variables to see if one is greater or less than another variable or value. The relational operators are <, >, <=, and >=. The logical operators in C# are &, &&, |, ||, and !. Just as you communicate a compound expression in English, you can combine expressions with AND or OR in C#. Review

Tables 5-4, 5-5, and 5-6 for logical AND, OR, or NOT.

Short-Circuit Evaluation

&& and || operators are the short-circuiting logical operators. These operators enable doing as little as is needed to produce the final result. With expressions involving &&, if the first evaluates as false, there is no need to evaluate the second. With expressions involving ||, if the first evaluates as true, there is no need to evaluate the second. The result will be true.

Boolean Data Type

If you place a variable that has been declared as a bool data type in a conditional expression, it is not necessary to compare the value to true or false. The bool variable can be placed in the parentheses, by itself.

Boolean Flags

This is because the bool variable holds the value of true or false; thus, it produces the result of true or false. Boolean data types are often used as flags to signal when a condition exists or when a condition changes.

if...else Selection Statements

The if statement facilitates specifying alternate paths based on the result of the conditional expression.

One-way if Statement

This statement is used when an expression needs to be tested, and if it evaluates to true, additional processing is performed. General format:

```
if (expression)
        statement;
```

Expression must be enclosed in parentheses. No semicolon is placed at the end of the line containing the expression. The expression must produce a Boolean result. With the one-way, when the expression evaluates to false, the statement immediately following the conditional expression is skipped or bypassed.

Two-way if Statement

An optional else clause can be added to an if statement to provide statements to be executed when the expression evaluates to false. With a two-way if statement, either the true statement(s) is (are) executed or the false statement(s), but not both. More than one statement may be included following the if or else by enclosing statements in curly braces. When you are writing selection statements, try to avoid repeating code.

TryParse( ) Method

When you invoke the Parse( ) method or methods of the Convert class, if the string value being converted is invalid with either of those options, your program crashes. An exception is thrown.

Another option to convert from string to another data type is to use the TryParse( ) method. It also returns a Boolean return value indicating whether the conversion succeeded or not. Zero is only stored in the result when the conversion fails.

Nested if…else Statement

When you place an if within an if, you create a nested if…else statement. Any statement can be included with the statements to be executed when the expression evaluates to true. The same holds for the false statements. It is acceptable to write an if within an if.

When evaluating the nested if statement, as long as the expressions evaluate to true, the true statements continue to be evaluated. As with the two-way, only one segment of code is executed. You never execute the true and its associated false statements. When the block is completed, all remaining conditional expressions and their statements are skipped or bypassed.

When programming a nested if…else statement, it is important to know which else matches which if statement. The rule for lining up, or matching, elses is that an else goes with the closest previous if that does not have its own else.

Several conventions exist for writing if statements. Some programmers line up all elses with their corresponding if statements, leaving the else on a line by itself. Others write else if on the same line. It is extremely important to be consistent.

Switch Selection Statements

When you need to test a single variable for equality against four or more values, a switch statement can be used. A switch statement only works for tests of equality. A switch statement is also called a case statement.

This variable or expression being evaluated must evaluate to an integral or string value. It cannot be used with a double, decimal, or float. But it is appropriate for short, int, long, char, and string data types. The general format is:

```
switch (expression)
{
    case value1: statement(s);
    break;
    case valueN: statement(s);
    break;
   [default:       statement(s);
    break;]
}
```

The expression called the selector; its value determines which of the cases will be executed. The case value must be a constant literal of the same type as the expression. You cannot use a variable or comparison expression in the value spot for the case.

The values for the case must be compatible with the expression. If the expression is a string variable, the value is enclosed in double quotes. If the expression is a char variable, the value is enclosed in single quotes.

Default is optional, but usually it is a good idea to include one. The default statements are only executed when there is no match of any of the case values. The break statements are required. There is no fall through, as found in some other languages.

Multiple case values can be associated with the same statement(s), and it is not necessary to repeat the statement(s) for each value. When more than one case is associated with the same statement(s), you group common cases together and follow the last case value with the executable statement(s) to be performed when any of the previous cases match.

Ternary Operator ? :

This conditional operator provides another way to express a simple if…else selection statement. The general format for this conditional expression is:

```
expression1 ? expression2 : expression3;
```

When expression1 evaluates to true, expression2, following the question mark, is executed; otherwise, expression3, following the colon, is executed.

Order of Operations

When an expression contains multiple operators, the precedence of the operators controls the order in which the individual operators are evaluated. Review Table 5-7. It includes the new operators introduced in this chapter. Except for the assignment operators, all binary operators are left-associative. The assignment operators and the conditional operator ? : are right-associative. Precedence and associativity can be controlled using parentheses.

PROGRAMMING EXAMPLE: SPEEDINGTICKET APPLICATION

This example demonstrates an application that uses a nested if…else selection statement to calculate fines for traffic tickets. Several different conditional expressions are constructed to determine the fine. Two separate classes are developed. One class

is designed to include characteristics of the ticket. The other class instantiates objects of the Ticket class and is used to test the Ticket class to ensure that all possible situations have been taken into consideration.

Structured English (pseudocode) and class diagrams are shown with the example. To test the SpeedingTicket algorithm, a test plan is developed. The test plan includes identified values for speed limit, ticketed speed, and classification.

Once the design is implemented, the test plan is used to validate that the program is running correctly by comparing the results with those obtained from your program output. Review Table 5-10. It shows a list of test values.

The desired output is to display the fine amount. This is shown on the prototype. The complete program listing is shown in the book and available as a Visual Studio project to demo for the class.