Notes for Chapter 2

Data Representation

Bits

Bit is a shortening of the words “Binary digIT.” Binary means two; thus, a binary digit can hold one of two values (bits), 0 or 1.

Bytes

An 8-bit combination is called a byte. Computer memories are commonly divided into 8-bit groupings.

Binary Number System

To represent data, computers use the base-2 number system, also known as the binary number system. Our base-10 number system, called the decimal system, uses ten symbols ranging from 0 to 9 to represent a value. Base 2 has only two symbols, and each bit holds a value of increasing powers of two.

Base 16, the hexadecimal numbering system, works on powers of 16. Base 8, the octal numbering system, uses powers of eight. Both are used to express binary numbers more compactly.

Character Sets

With only 8 bits, you can represent 2^8, or 256, different decimal values ranging from 0 to 255.
This is 256 different characters or different combinations of 0 and 1. The character set used by programmers of C# (pronounced C sharp) is called Unicode. A subset of Unicode, the first 128 characters, corresponds to the American Standard Code for Information Interchange (ASCII) character set.

Kilobyte, Megabyte, Gigabyte, Terabyte, Petabyte…

- kilo is about a thousand
- mega is about a million
- giga is about a billion
- tera is about a trillion
- peta is about a zillion

Memory Locations for Data

Programs manipulate data, and data can take the form of a number, single character, or combination of characters. Without identifying and labeling the data, it is meaningless.

Identifiers

Identifiers are names of elements that appear in a program, such as data items. Rules for creating an identifier in C#:

1. First character may not be a numeric digit.
2. Cannot separate words in a name by a space. Normally concatenate words by capitalizing the second and subsequent words.
3. Keywords cannot be used as identifiers. Review Tables 2-3 and 2-4. Table 2-4 shows the new contextual keywords. Contextual keywords were added in an attempt to avoid breaking code written using an earlier framework. You can use the "@" symbol as one of the characters. When used, the "@" symbol enables keywords to be used as identifiers. You should avoid using it unless you are developing applications that consist of program statements from more than one language.
4. C# is case sensitive.
5. Be descriptive.

Variables

Declaring a variable requires that you select an identifier and determine what type of data will appear in the memory cell. The syntax for doing this is: type identifier;

You can do a compile-time initialization by assigning a value to the variable at the time it is declared. This is just an initial value only—the value can be changed. The syntax for doing this is: type identifier = expression;

Literal Values

Literal values cannot be changed. Examples are the number 77 and the character 'A'.

Types, Classes, and Objects

C# is an object-oriented language that makes extensive use of classes and objects.

Types

There are a number of predefined types that exist as part of .NET. One type of number is called an integer or int. An int is a whole number (positive or negative) that contains no decimal point. Another type of number is a floating-point value. C# supports two types of floating-point types: float and double. Both of these types can contain a fractional portion.

Classes

Types are implemented in .NET languages through classes. Since C# was designed from the ground up to be and is a true object-oriented language, there is a one-to-one correspondence between a class and a type in C#.

Objects

An object is an instance of a class; an occurrence of the class. The class combines both the data and behaviors into a single package or unit. The data portion of an int is always a whole number value. The int also has certain behaviors or actions that can be performed on it. The behavior can be described by stating basic arithmetic operations such as addition and subtraction or logical comparisons that can be performed on the type.

Predefined Data Types

.NET Framework includes a Common Type System (CTS) that is supported by all .NET languages. These types are divided into reference and value types. Value types contain their own copy of data in binary notation. Reference types contain the address or location in which the sequence of bits is stored. The string data type is a reference type.

Value Types

These types are often called the fundamental data types or primitive data types. Twelve types belong to this category. The integral (int), floating-point (double, float), char, decimal, and Boolean (bool) types are among them. The decimal type is new; it is not found with C++, C, or Java. It was added to C# to eliminate the problems of loss of precision in mathematical operations. Review Table 2-8 for a list of the types and their associated .NET equivalent.

Integral Data Types

This type enables values that represent whole numbers without decimal notation to be stored. Review Table 2-9 for a list of the different integral types supported in C#. The primary difference between the different integral types is in how large or small the value can be that is stored in that type. This determines how much memory is needed for the data type. One of the integral types, a char, can hold a single value, such as the letter ‘A’. Notice how the data associated with the char is always enclosed in single quotation marks.

Floating-Point Types

Floating-point types can store values with a fractional component. Very large or small values may be specified in scientific notation with an exponent or in standard decimal notation. The general form for the value in scientific notation is: n.ne±P

n is the decimal number; P is the number of decimal positions to move left or right, and + or – indicate the direction the decimal should be moved. A plus sign indicates move P positions to the right. The e can be uppercase or lowercase.

Double is the default type for floating-point numbers. When a compile-time initialization or assignment is made, no suffix is required if you initialize a double variable; however, with float, it is necessary to suffix the number with an ‘f’ or ‘F’.

Decimal Types

This value type is new to most modern programming languages and is appropriate for storing monetary data items. Greater precision is found with this type than floating point; 128 bits are used to represent the value. As is the case with the float type, it is necessary to attach the suffix ‘m’ or ‘M’ onto the end of a number to indicate decimal. Without the suffix, the number is treated as a double.

Boolean Variables

The only Boolean type in C# is bool. It can have a value of either true or false. The bool type will not accept integer values such as 0, 1, or –1.The keywords true and false are built into the C# language and are the only allowable values.

Declaring Strings

Strings are reference types. The memory location contains a reference to the location where data is stored. C# has two built-in reference types: string and object. The string type represents a string of Unicode characters. In order to assign a string value to a memory location, enclose the characters in double quotation marks. You may use the escape sequence characters inside a string.

Making Data Constant

When you add the keyword const to a declaration, it becomes a constant. const forces the functionality of not allowing the value to be changed. The value stored in the memory location can be used throughout the program. The syntax is: const type identifier = expression;

An advantage of defining a constant is that the value need only be assigned once, during declaration. If you need to change the value, you don't have to find the value in many different locations, but rather just find the declaration and change it there.

A common convention is to name the identifier for the constant using all uppercase characters.

Assignment Statements

Variables can be initialized during declaration, or a value can be assigned to them later in the program. To change the value of the variable, use an assignment statement. The syntax is: variable =expression; wherein the expression may be another variable, a compatible literal value, a mathematical equation, a call to a method that returns a compatible value, or any combination of the above. Notice that the variable that will hold the result of the expression is listed first on the left of the equal (=) symbol.

Basic Arithmetic Operations

The basic operations are +, -, *, /, and %. The simplest form of an assignment statement is: resultVariable = operand1 operator operand2;

Operands may be variables, constants, or literals. Review Table 2-12 for a list of the special symbols used for the operators. Modulus operator (%) is sometimes referred to as the remainder operator. C# allows you to use floating-point values as operands to the modulus operator. The result produced is the remainder of operand1 divided by operand2.

The + symbol is considered an overloaded operator. It behaves differently based on the type of operands it receives. If the operands are numeric, it performs addition. If the operands are strings, it performs concatenation.

Increment and Decrement Operations

A common operation is to add or subtract the value 1 to or from a memory location. The symbols used for increment and decrement are ++ and --. No space is permitted between the two symbols (++ or --). They are used with a single operand and are considered unary operators.

The placement of the ++ or -- is important when they are used as part of an expression involving other operations. If they appear as prefixes to the operand, or to the left of the variable, the increment or decrement is performed before using them in the

expression.

Compound Operations

Operations that modify a variable by using the original value as part of the calculation can be written in a shortcut. The +, -, *, /, and % can be placed before the equal symbol (=) to indicate the original value of the result should be used as part of the expression. In expressions involving multiple operations, the operation that is listed as the compound operator is always performed last. One very common use of the addition compound operation is for keeping a running total or accumulating a value. += is used for this operation.

Order of Operations

When multiple arithmetic operators are included in an expression, execution begins with the operator that has the highest level of precedence. This is determined by the rules of the language. Review Table 2-14. Most operators are left-associative, performed from left to right. The exceptions are the unary and assignment operators.

Mixed Expressions

A mixed mode expression has numeric integral types and floating-point types in an expression. When the operands are of the same type, the result of the operation will be of that type. However, if the binary operation involves a double and an int, implicit type coercion is performed. Integral types convert to floating-point types. This is also considered an automatic coercion. No conversion occurs if you attempt to store a double in an int variable; a syntax error will be produced.

When one of the operands is a number literal, you can make the literal act like a floating-point type by simply affixing a decimal followed by a zero.

Casts

Casting makes a variable temporarily behave as if it is a different type. This is considered an explicit type coercion and takes the form of: (type) expression;

Formatting Output

C# includes a number of special format specifiers that can be used to format data by adding dollar signs or to separate digits with commas or to show the fractional portion of a value. These format specifiers can be used to suppress leading zeros or pad a value with characters. You can use these formatting specifiers in Write( ) or WriteLine( ) methods. The currency format is specified using C or c. The format specifier is placed inside the curly braces ({ }) as a string with Write( ) or WriteLine( ) methods.

Write("{0:c}", 26666.7888); produces $26,666.79 as output. To indicate that a value should be formatted with a fixed or decimal point and four digits should be printed to the right of the decimal, use{0:F4}. The 0 indicates that the first argument is the one to be formatted.

If the standard format specifier does not provide the type of formatting you require, you can also create your own custom format string. Format specifiers can also be stored in a string variable and then used as arguments to methods such as the ToString( ) method.

Recall, as introduced in Chapter 1, a new feature available with Visual Studio 2015 is the capability of adding a reference to a class that has static members. Once this is done, you no longer need to fully quality calls to methods in the class. Console is an example of a static class.

The directive is added at the top of the program as shown below:

```
using static System.Console;
```

Once this is added, you can omit Console from calls to methods like Write( ), WriteLine( ) and ReadKey( ).

Width Specifier

You can right or left justify text using width specifier. Following the placeholder in the format specifier, type a comma and the field's length. The value is right justified and padded with spaces to the left. If the number is negative, the number is left justified with white space to the right.

PROGRAMMING EXAMPLE: CARPETCALCULATOR

This example demonstrates the use of data items in a program. It begins by showing a problem specification that details the problem definition. Focus is placed on understanding the problem definition.

No input from the user is used in the program; however, there are a number of variables needed. The dimensions of the room are given in feet and inches as two separate memory locations. These values are used to determine the total square feet of carpet needed. The number of square yards is calculated once the total square feet is calculated. The carpet price is stored as a floating-point type. A memory location is needed for the cost per square yard and the total cost of the carpet. Table 2-18 lists the variables, their data type, and their domain.

Constants for the number of square feet in a yard, the number of inches in a foot, and the names of the two types of carpet are defined. These are shown in Table 2-19.

The output should consist of a display showing the costs associated with each kind of carpet, given a specific room size. A prototype for the output is shown with the example. Algorithms are developed using flowchart and Structured English (pseudocode). For this example, a class diagram is shown illustrating the need for data fields and actions or behaviors on those data items. The data fields are shown in the middle of the diagram. Behaviors are listed at the bottom of the diagram.

After the algorithm is developed, the design should be checked for correctness by desk checking the algorithm. One way to do this is to follow the logic of the program, using a calculator, and write down the results obtained. Then when the program is complete, verify those results are produced by the program.

Note that the statements inside the Main ( ) method are executed in sequential order. Because methods have not been introduced, all calculations are performed in the main method. The program listing is given, and the Visual Studio project is available in its entirety.

Coding Standards

Describe the naming conventions used for identifiers. Make a distinction between camel and Pascal cases, identifying when to use which standard. Discuss spacing in

terms of readability. Spend time showing examples of how variables are declared and initialized.