# Chapter 2

**Using Data**

|  |
| --- |
| **At a Glance** |

#### Instructor’s Manual Table of Contents

* Overview
* Objectives
* Teaching Tips
* Quick Quizzes
* Class Discussion Topics
* Additional Projects
* Additional Resources
* Key Terms

|  |
| --- |
| Lecture Notes |

Overview

Chapter 2 explores how to store, manipulate, and display data in a C# program. Students will learn about declaring variables and displaying variable values. Next, they will learn about the integral and floating-point data types, and they will format floating-point values. Chapter 2 also describes how to use standard binary arithmetic operators and shortcut arithmetic operators. In addition, students will learn about the bool data type, numeric type conversion, the char data type, and the string data type. Finally, Chapter 2 shows students how to define named constants and enumerations, and how to accept console input.

Objectives

* Learn about declaring variables
* Display variable values
* Learn about the integral data types
* Learn about floating-point data types
* Use arithmetic operators
* Learn about the bool data type
* Learn about numeric type conversion
* Learn about the char data type
* Learn about the string data type
* Define named constants and enumerations
* Accept console input

Lecture Notes

Declaring Variables

1. Describe the two basic uses for data types that hold information in applications:

* **Constant** – a data type whose value is established when the application is compiled and does not change its value during the execution of the application. A **literal constant** takes its value literally with each execution of the application.
* **Variable** – a data type that is a named location in computer memory whose value can change during the execution of the program.

1. Explain that a **data type** describes the format and size of, or amount of memory occupied by, a data item. Use Table 2-1 to illustrate the **intrinsic types** of data in C#.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Read more about C# data types at: [www.csharpcorner.com/UploadFile/mahesh/cs\_lang\_211142005043100AM/cs\_lang\_2.aspx](http://www.csharpcorner.com/UploadFile/mahesh/cs_lang_211142005043100AM/cs_lang_2.aspx). |

1. Define a **variable declaration** as the statement that names a variable and reserves storage for it, and optionally stores an initial value in it. A variable must be declared before it can be used.
2. Mention that there are two basic ways to declare variables:

* Each variable can be declared and optionally initialized in its own statement or declaration.
* Multiple variables of the same type can be declared and optionally initialized in a single statement by using the type once and separating the variable declarations with a comma.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Tips on initializing variables:   * It is good practice to initialize variables when they are declared (e.g., int aInt = 0;). * If your application tries to use an unassigned variable, the compiler will return an error message such as “Use of unassigned local variable ‘variable name’”. * When declaring multiple variables in the same statement, each variable must be initialized separately (e.g., int aInt = 0, bInt = 0, cInt = 0;). * Optionally, for efficiency in declaring multiple variables, you could write something like this:   int aInt, bInt, cInt;  aInt = bInt = cInt = 0; |

Displaying Variable Values

1. Use Figures 2-1 through 2-4 to explain how to display a variable value using the Write() and WriteLine() methods.
   * Explain that the WriteLine() method includes a carriage return and a line feed that advances the cursor to the next line, while the Write() method does not.
2. Explain how to **concatenate** a string in C# using the plus (+) sign.
3. Explain that you can display a variable in a Write() or WriteLine() method:
   * by itself, as shown in Figures 2-1, 2-2, 2-3, and 2-4.
   * as part of a string concatenation using the + character.
   * as part of a format string using concatenation or placeholders.
   * by using placeholders, as shown in Figures 2-5, 2-6, and 2-7.
4. Define a **format string** as a string of characters that optionally contains fixed text and one or more variables using either concatenation or placeholders.
5. Explain that a **placeholder** consists of a pair of curly braces containing a number that indicates the desired variable’s position in a list that follows the string.
   * The first number must be 0 with other numbers following in sequence.
6. Use Figures 2-5 through 2-7 to show how to display variable values using placeholders.
7. Explain how to specify alignment and field size when displaying a variable value. Use Figures 2-8 and 2-9 to illustrate your explanation.

Using the Integral Data Types

1. Explain that **integral data types** store **integers**, or whole numbers.
2. Describe the nine integral data types supported by C#: **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long**, **ulong**, and **char**.
   * Consider mentioning that the long integral data type can hold numbers larger than 9 quintillion, and the ulong data type can hold numbers larger than 18 quintillion. Those are very large numbers!
3. Numeric integer literals in code (e.g., 123) are treated as int by default.

Using Floating-Point Data Types

1. Explain that a **floating-point** number contains decimal positions.
2. Describe the floating-point data types supported by C#: **float**, **double**, and **decimal**.
   * Explain that float and double are actually approximations. Therefore, they can cause unexpected results when two floating point values are compared, especially using equals (==).
   * Numeric floating-point literals in code (e.g., 123.45) are treated as double by default. To make the value a float, place an *F* after the value. To make the value a decimal, place an *M* after the value (e.g., 123.45F or 123.45M).
   * Ensure students understand the number of significant digits for each type of floating-point data type.
     + Float has 7 significant digits.
     + Double has 15 or 16 significant digits.
     + Decimal has 29 significant digits, and its largest value is 79,228,162,514,264,337,593,543,950,335. That’s over 79 octillion!

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Warn your students not to simply use double data types by default. Floating-point data types are far less efficient to process than integral types. |

Formatting Floating-Point Values

1. Mention that C# displays floating-point numbers in the most concise way possible while maintaining the correct value.
2. Define **standard numeric format strings** as strings of characters expressed within double quotation marks that indicate a format for output. They take the form *X0*, in which *X* is the format specifier and *0* is the precision specifier.
3. Explain that a **format specifier** defines the most commonly used numeric format types. Use Table 2-2 to show the format specifiers supported by C#.
4. Explain that a **precision specifier** controls the number of significant digits or zeros to the right of the decimal point.
5. Explain the use of numeric format strings with examples.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Read more about numeric format strings in C# at: <http://msdn2.microsoft.com/en-us/library/dwhawy9k.aspx>. |

Using Arithmetic Operators

1. Mention that **binary operators** use two values (or **operands**). Use Table 2-3 to show the standard binary arithmetic operators in C#.
2. Define **operator precedence** as the rules that determine the order in which parts of a mathematical expression are evaluated. Multiplication, division, and remainder, also called modulus, always take place prior to addition or subtraction in an expression.
3. Normal operator precedence may be overridden by using parentheses.
4. Calculations are executed from left to right.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Ask students to read more about operators in C# at: <http://msdn2.microsoft.com/en-us/library/ms173145.aspx>. |

Using Shortcut Arithmetic Operators

1. Explain how to use the following arithmetic operators:
   * **Add and assign operator** (variations: +=, –=, \*=, /=, and %=)
   * **Prefix increment operator** and **postfix increment operator**
   * Prefix and postfix **decrement operators**

Using the bool Data Type

1. Explain that a **Boolean variable** can hold only one of two values—true or false. Declare a Boolean variable by using type **bool**.
2. Mention that a **comparison operator** compares two items. An expression containing a comparison operator returns a Boolean value. Use Table 2-4 to show the comparison operators in C#.

Quick Quiz 1

1. A(n) \_\_\_\_ is the statement that names a variable and reserves storage for it.

Answer: variable declaration

1. \_\_\_\_ are strings of characters expressed within double quotation marks that indicate a format for output.

Answer: Standard numeric format strings

1. The values that operators use in expressions are called \_\_\_\_.

Answer: operands

1. You can use a prefix or postfix \_\_\_\_ operator (--) that reduces a variable’s value by 1.

Answer: decrement

Understanding Numeric Type Conversion

1. Mention that when you perform arithmetic with variables or constants of the same data type, the result of the arithmetic operation retains that same data type.
2. Explain that when you perform arithmetic operations with operands of dissimilar types, C# chooses a **unifying type** for the result and **implicitly**, or automatically, converts nonconforming operands to the unifying type, which is the type with the higher **type precedence**.
3. Define an **implicit cast** as the automatic transformation that occurs when a value is assigned to a type with higher precedence.
4. Explain that an **explicit cast** involves placing the desired result type in parentheses in front of the variable or constant to be cast.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Read more about casting in C# at: <http://msdn2.microsoft.com/en-us/library/ms173105.aspx>. |

Using the char Data Type

1. Explain that the **char** data type holds any single Unicode character. You place character literals within single quotation marks (e.g., 'A').
2. Explain that an **escape sequence** stores a pair of characters that always begins with a backslash. The pair of symbols represents a single character. Use Table 2-5 to show common escape sequences in C#.
3. Mention that C# represents characters using **Unicode**, which uses 2 bytes.

|  |  |
| --- | --- |
| ***Teaching***  ***Tip*** | Read more about Unicode at: [www.unicode.org](http://www.unicode.org). |

Using the string Data Type

1. Explain that the **string** data type holds a series of Unicode characters. Values are expressed within double quotation marks.
   * You can think of a string as being an array of Unicode characters with the first character in position 0, the second character in position 1, and so on.
2. Explain how to compare strings using Figures 2-15 through 2-19.
   * Explain the difference between the **Compare() method** and the **CompareTo() method**.
3. Mention the **Length property** of a string.
   * Explain that the Length property can be very useful when it is necessary to analyze the contents of a string.
4. Explain the **Substring() method**. Use Figure 2-19 to illustrate your explanation.

Defining Named Constants

1. Define a **named constant**, often simply called a constant, as an identifier whose contents cannot change. Constants are created just like variables with the addition of the keyword const.
2. Mention that programmers usually name constants using all uppercase letters, inserting underscores for readability. This helps make your statements **self-documenting**.

Working with Enumerations

1. Explain the idea of an **enumeration** as a way of defining a set of constants.
2. State that the default form is an integer, but you can use other integral types.
3. Explain that values are assigned successively by default.

Accepting Console Input

1. Define an **interactive program** as a program that allows user input.
2. Explain that the **Console.ReadLine() method** accepts user input from the keyboard. This method accepts all of the characters entered by a user until the user presses Enter. The characters can be assigned to a string. You must use a Convert() method to convert the input string to the proper type.

Using the Convert Class

1. Use Table 2-6 to show selected Convert() class methods.

Using the Parse() Methods

1. Mention that the Parse() methods can be used as an alternative to the Convert() class methods for converting strings.

|  |  |
| --- | --- |
| *Teaching*  ***Tip*** | Read more about the Convert() class at: <http://msdn.microsoft.com/en-us/library/system.convert.aspx>. |

You Do It

1. Guide your students while they explore the following activities:
   * Declaring and Using Variables
   * Performing Arithmetic
   * Working with Boolean Variables
   * Using Escape Sequences
   * Writing a Program that Accepts User Input

Quick Quiz 2

1. A(n) \_\_\_\_ is the automatic transformation that occurs when a value is assigned to a type with higher precedence.

Answer: implicit cast

1. The characters used in C# are represented in \_\_\_\_, which is a 16-bit coding scheme for characters.

Answer: Unicode

1. Sometimes you want to create a(n) \_\_\_\_ (often simply called a constant), an identifier whose contents cannot change.

Answer: named constant

1. A program that allows user input is a(n) \_\_\_\_.

Answer: interactive program

Class Discussion Topics

1. Should variables that contain numbers always be declared as integer or floating-point data types? Why or why not? Name potential examples.
2. What is the difference between an implicit cast and an explicit cast?

Additional Projects

1. Write a C# program that asks your name and prints the following message:   
   Hello, *yourName*!
2. Write a C# program that asks for three integers and prints the average value as a double.
   * Example (3 + 6 + 7) / 3 = 5.33333333333333
3. Write a C# enum using the last names of people shown on U.S. currency and their corresponding values. It’s ok if students need to do some research. The final enum should look like this:

enum USCurrency

{

Washington = 1,

Jefferson = 2,

Lincoln = 5,

Hamilton = 10,

Jackson = 20,

Grant = 50,

Franklin = 100,

McKinley = 500,

Cleveland = 1000,

Madison = 5000,

Chase = 10000,

Wilson = 100000

}

Additional Resources

1. C# data types:

[www.geekpedia.com/tutorial29\_Csharp-data-types.html](http://www.geekpedia.com/tutorial29_Csharp-data-types.html)

1. C# data types:

<http://msdn2.microsoft.com/en-us/library/ms173104.aspx>

1. Read more about declaring variables in C#: [www.codeguru.com/csharp/sample\_chapter/article.php/c11387](http://www.codeguru.com/csharp/sample_chapter/article.php/c11387)
2. Custom Numeric Format Strings:

<http://msdn2.microsoft.com/en-us/library/0c899ak8.aspx>

1. Convert methods:

<http://msdn.microsoft.com/en-us/library/system.convert.aspx>

Key Terms

|  |  |
| --- | --- |
|  | The **add and assign operator** (+=) adds the operand on the right to the operand on the left and assigns the result to the operand on the left in one step. |
|  | An **alias** is another name for something. |
|  | An **assignment** is a statement that provides a variable with a value. |
|  | The **assignment operator** is the equal sign (=); any value to the right of the assignment operator is assigned to, or taken on by, the variable to the left. |
|  | **Associativity** specifies the order in which a sequence of operations with the same precedence are evaluated. |
|  | **Binary operators** use two arguments—one value to the left of the operator and another value to the right of it. |
|  | The **bool** data type holds a Boolean value. |
|  | A **Boolean variable** can hold only one of two values—true or false. |
|  | The nine integral types are **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long**, **ulong**, and **char**. The first eight always represent whole numbers, and the ninth type, char, is used for characters like 'A' or 'a'. |
|  | The **char** data type can hold any single character; a literal character is contained between single quotation marks. |
|  | The **Compare() method** requires two string arguments. When it returns 0, the two strings are equivalent; when it returns a positive number, the first string is greater than the second; and when it returns a negative value, the first string is less than the second. |
|  | The **CompareTo() method** uses a string, a dot, and the method name. When it returns 0, the two strings are equivalent; when it returns a positive number, the first string is greater than the second; and when it returns a negative value, the first string is less than the second. |
|  | A **comparison operator** compares two items; an expression containing a comparison operator has a Boolean value. |
|  | To **concatenate** a string with another value is to join the two values end to end using a plus sign. |
|  | The **Console.ReadLine() method** accepts user input from the keyboard. |
|  | **Constant** describes data items whose values are fixed. |
|  | A C# **culture** is a set of rules that determines how culturally dependent values such as money and dates are formatted. |
|  | A **data type** describes the format and size of a data item and defines what types of operations can be performed with the item. |
|  | The **decimal** data type is a floating-point type that has a greater precision and a smaller range than a float or double, which makes it suitable for financial and monetary calculations. |
|  | The **decrement operator** (--) reduces a variable’s value by 1. There is a prefix and a postfix version. |
|  | A **double** data type can hold a floating-point number with 15 or 16 significant digits of accuracy. |
|  | An **enumeration** is a set of constants represented by identifiers. |
|  | The String class **Equals() method** determines if two strings have the same value; it requires two string arguments placed within its parentheses, separated by a comma. |
|  | An **escape sequence** is two symbols beginning with a backslash that represent a nonprinting character such as a tab. |
|  | An **explicit cast** purposefully assigns a value to a different data type; it involves placing the desired result type in parentheses followed by the variable or constant to be cast. |
|  | **Explicitly** means purposefully. |
|  | A **float** data type can hold a floating-point number with as many as seven significant digits of accuracy. |
|  | A **floating-point** number is one that contains decimal positions. |
|  | The **format specifier** in a format string can be one of nine built-in format characters that define the most commonly used numeric format types. |
|  | A **format string** is a string of characters that contains one or more placeholders for variable values. |
|  | **Hexadecimal**, or **base 16**, is a mathematical system that uses 16 symbols to represent numbers. |
|  | **Immutable** describes unchangeable objects such as strings that refer to a new memory location when a new value is assigned. |
|  | An **implicit conversion** or **implicit cast** is the automatic transformation that occurs when a value is assigned to a type with higher precedence. |
|  | **Implicitly** means automatically. |
|  | An **initialization** is an assignment made when a variable is declared. |
|  | **Integers** are whole numbers. |
|  | **Integral data types** are those that store whole numbers. |
|  | An **interactive program** is one that allows user input. |
|  | **Intrinsic types** of data are basic types; C# provides 15 intrinsic types. |
|  | The **Length property** contains a string’s length. |
|  | **Lexically** means alphabetically. |
|  | A **literal constant** is a value that is taken literally at each use. |
|  | A **named constant** is an identifier whose value must be assigned upon declaration and whose contents cannot change. |
|  | **Operands** are the values that operators use in expressions. |
|  | **Order of operation** is another term for operator precedence. |
|  | **Operator precedence** determines the order in which parts of a mathematical expression are evaluated. |
|  | To **parse** an item is to break it into component parts. |
|  | A **placeholder** in a format string consists of a pair of curly braces containing a number that indicates the desired variable’s position in a list that follows the string. |
|  | The **postfix increment operator** (++ after a variable) evaluates a variable and then adds 1 to it. |
|  | The **precision specifier** in a format string controls the number of significant digits or zeros to the right of the decimal point. |
|  | The **prefix increment operator** (++ before a variable) increases the variable’s value by 1 and then evaluates it. |
|  | A **prompt** is an instruction to the user to enter data. |
|  | **Scientific notation** is a means of expressing very large and small numbers using an exponent. |
|  | A **self-documenting** program element is one that is self-explanatory. |
|  | **Significant digits** provide a measure of the mathematical accuracy of a value. |
|  | A **simple type** is one of the following in C#: byte, sbyte, short, ushort, int, uint, long, ulong, float, double, decimal, char, and bool. |
|  | **Standard numeric format strings** are strings of characters expressed within double quotation marks that indicate a format for output. |
|  | The **StartsWith() method** is used with a string and a dot, and its parentheses contain another string. It returns true if the first string starts with the characters contained in the second string. |
|  | The **string** data type is used to hold a series of characters; a literal string is contained between double quotation marks. |
|  | The **Substring() method** can be used to extract a portion of a string from a starting point for a specific length. |
|  | **Type precedence** is a hierarchy of data types used to determine the unifying type in arithmetic expressions containing dissimilar data types. |
|  | **Type-safe** describes a data type for which only appropriate behaviors are allowed. |
|  | **Unary operators** are operators used with one operand. |
|  | **Unicode** is a 16-bit coding scheme for characters. |
|  | A **unifying type** is the type chosen for an arithmetic result when operands are of dissimilar types. |
|  | A **variable** is a named location in computer memory that can hold different values at different points in time. |
|  | A **variable declaration** is the statement that names a variable; it includes the data type that the variable will store, an identifier that is the variable’s name, an optional assignment operator and assigned value when you want a variable to contain an initial value, and an ending semicolon. |