COIS1020H: Programming for Computing Systems

Chapter 2
Using Data

Declaring Variables

- **Constant**
  - Cannot be changed after a program is compiled
- **Literal constant**
  - Its value is taken literally at each use
- **Variable**
  - A named location in computer memory that can hold different values at different points in time
- **Data type**
  - Describes the format and size of (amount of memory occupied by) a data item

Type	System Type	Bytes	Description	Largest Value	Smallest Value
byte	Byte	1	Unsigned byte	255	0
sbyte	Sbyte	1	Signed byte	127	−128
short	Int16	2	Signed short	32,767	−32,768
ushort	UInt16	2	Unsigned short	65,535	0
int	Int32	4	Signed integer	2,147,483,647	−2,147,483,648
uint	UInt32	4	Unsigned integer	4,294,967,295	0
long	Int64	8	Signed long integer	Approximately 9×10^{18}	Approximately -9×10^{18}
ulong	UInt64	8	Unsigned long integer	Approximately 18×10^{18}	0
float	Single	4	Floating-point	Approximately 3.4×10^{38}	Approximately -3.4×10^{38}
double	Double	8	Double-precision floating-point	Approximately 1.8×10^{308}	Approximately -1.8×10^{308}
decimal	Decimal	16	Fixed-precision number	Approximately 7.9×10^{28}	Approximately -7.9×10^{28}
char	Char	2	Unicode character	0xFFFF	0x0000
bool	Boolean	1	Boolean value (true or false)	NA	NA
string	String	NA	Unicode string	NA	NA
object	Object	NA	Any object	NA	NA

Table 2-1 C# data types

Declaring Variables (cont'd.)

- **Variable declaration**
  - Statement that names a variable and reserves storage
  - Example: `int myAge = 25;`
- You can declare multiple variables of the same type
  - In separate statements on different lines
- You can declare two variables of the same type in a single statement
  - By using the type once and separating the variable declarations with a comma

Displaying Variable Values

```
using System;
public class DisplaySomeMoney
{
   public static void Main()
   {
      double someMoney = 39.45;
      Console.WriteLine(someMoney);
   }
}
```

Figure 2-1 Program that displays a variable value

39.45

Displaying Variable Values (cont'd.)

```
using System;
public class DisplaySomeMoney2
{
   public static void Main()
   {
      double someMoney = 39.45;
      Console.Write("The money is $");
      Console.WriteLine(someMoney);
   }
}
```

Figure 2-3 Program that displays a string and a variable value

The money is $39.45

Displaying Variable Values (cont'd.)

- **Format string**
  - A string of characters that optionally contains fixed text
  - Contains one or more format items or placeholders for variable values
- **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

Displaying Variable Values (cont'd.)

```
using System;
public class DisplaySomeMoney3
{
   public static void Main()
   {
      double someMoney = 39.45;
      Console.WriteLine("The money is ${0} exactly",
         someMoney);
   }
}
```

Figure 2-5 Using a format string

The money is $39.45 exactly

Displaying Variable Values (cont'd.)

- Formatting output

```
int num1 = 4, num2 = 56, num3 = 789;
Console.WriteLine("{0, 5}", num1);
Console.WriteLine("{0, 5}", num2);
Console.WriteLine("{0, 5}", num3);
```

```
    4
   56
  789
```


Using the Integral Data Types

- **Integral data types**
  - Types that store whole numbers
  - **`byte`, `sbyte`, `short`, `ushort`, `int`, `uint`, `long`**, **`ulong`**, and **`char`**
- Variables of type **`int`**
  - Store (or hold) **integers**, or whole numbers
- Shorter integer types
  - **`byte`, `sbyte`** (which stands for signed byte), **`short`** (short **int**), or **`ushort`** (unsigned short **`int`**)

Using Floating-Point Data Types

- **Floating-point** number
  - Contains decimal positions
- Floating-point data types
  - **`float`**
    - Can hold up to seven significant digits of accuracy
  - **`double (default)`**
    - Can hold 15 or 16 significant digits of accuracy
  - **`decimal`**
    - Has a greater precision and a smaller range
    - Suitable for financial and monetary calculations

Using Floating-Point Data Types (cont'd.)

- **Significant digits**
  - Specifies the mathematical accuracy of the value
- Suffixes
  - Put an `F` after a number to make it a `float`

```
float val = 54.7;   // incorrect
float val = 54.7F;  // correct
```

  - Put a `D` after it to make it a `double`
  - Put an `M` after it to make it a `decimal`

```
decimal wage = 12.55;   // incorrect
decimal wage = 12.55M;  // correct
```

Using Floating-Point Data Types (cont'd.)

- **Scientific notation**
  - Includes an *E* (for exponent)

  -123.78 is the same as -1.2378E2

  0.000382 is the same as 3.82E-4

Formatting Floating-Point Values

- C# displays floating-point numbers in the most concise way it can
  - While maintaining the correct value
- **Standard numeric format strings**
  - Strings of characters expressed within double quotation marks that indicate a format for output
  - Take the form *X0*
    - *X* is the format specifier; *0* is the precision specifier
- **Format specifiers**
  - Define the most commonly used numeric format types

Formatting Floating-Point Values (cont'd)

```
The number is 34.456
The number is 34.46
The number is $34.46
The number is       34.5
The number is 34.456
The number is $23,456.780
```

```
using System;
public static class FormatExample
{
   public static void Main()
   {
      double val = 34.456;
      Console.WriteLine("The number is {0}", val);
      Console.WriteLine("The number is {0:F}", val);
      Console.WriteLine("The number is {0:C}", val);
      Console.WriteLine("The number is {0,7:F1}", val);
      Console.WriteLine("The number is {0:F3}", val);
      val = 23456.78;
      Console.WriteLine("The number is {0:C3}", val);
   }
}
```


Formatting Floating-Point Values (Alt)

```
The number is 34.456
The number is 34.46
The number is $34.46
The number is       34.5
The number is 34.456
The number is $23,456.780
```

```
using System;
public class FormatExampleAlt
{
   public static void Main()
   {
      double val = 34.456;
      Console.WriteLine("The number is {0}", val);
      Console.WriteLine("The number is {0}", val.ToString("F"));
      Console.WriteLine("The number is {0", val.ToString("C"));
      Console.WriteLine("The number is {0,7}", val.ToString("F1"));
      Console.WriteLine("The number is {0}", val.ToString("F3"));
      val = 23456.78;
      Console.WriteLine("The number is {0}", val.ToString("C3"));
   }
}
```

Formatting Floating-Point Values (cont'd.)

Format Character	Description	Default Format (if no precision is given)
C or c	Currency	$XX,XXX.XX ($XX,XXX.XX)
D or d	Decimal	[-]XXXXXXX
E or e	Scientific (exponential)	[-]X.XXXXXXE+xxx [-]X.XXXXXXe+xxx [-]X.XXXXXXE-xxx [-]X.XXXXXXe-xxx
F or f	Fixed-point	[-]XXXXXXX.XX
G or g	General	Variable; either with decimal places or scientific
N or n	Number	[-]XX,XXX.XX
P or p	Percent	Represents a numeric value as a percentage
R or r	Round trip	Ensures that numbers converted to strings will have the same values when they are converted back into numbers
X or x	Hexadecimal	Minimum hexadecimal (base 16) representation

Table 2-2 Format specifiers

Using the Standard Binary Arithmetic Operators

- **Binary operators**
  - Use two values (**operands**)

    int x = 8, y = 9, z;

    z = x + y; // z would store 17

- **Unary operators**
  - Uses one operand

    int x = 8, z;

    z = -x; // z would store -8

Using the Standard Binary Arithmetic Operators

- **Operator precedence**
  – Rules that determine the order in which parts of a mathematical expression are evaluated
  – Multiplication, division, and remainder always take place prior to addition or subtraction in an expression
  – You can override normal operator precedence with parentheses

Using the Standard Binary Arithmetic Operators (cont'd.)

Operator	Description	Example
+	Addition	45 + 2: the result is 47
–	Subtraction	45 – 2: the result is 43
*	Multiplication	45 * 2: the result is 90
/	Division	45 / 2: the result is 22 (not 22.5)
%	Remainder (modulus)	45 % 2: the result is 1 (that is, 45 / 2 = 22 with a remainder of 1)

Table 2-3 Binary arithmetic operators

Using Shortcut Arithmetic Operators

- **Add and assign operator**
  - Example: `total = total + val;`
    `total += val;`
  - Variations: `-=`, `*=`, and `/=`
- **Prefix increment operator**
  - Example: `someValue = someValue + 1;`
    `++someValue;`
- **Postfix increment operator**
  - Example: `someValue = someValue + 1;`
    `someValue++;`
- **Decrement operator** (--)

Using the `bool` Data Type

- **Boolean variable**
  - Can hold only one of two values—true or false
  - Declare a Boolean variable with type **`bool`**
    `bool done = true;`
- **Comparison operator**
  - Compares two items
  - An expression containing a comparison operator has a Boolean value

Using the `bool` Data Type (cont'd.)

Operator	Description	`true` Example	`false` Example
<	Less than	3 < 8	8 < 3
>	Greater than	4 > 2	2 > 4
==	Equal to	7 == 7	3 == 9
<=	Less than or equal to	5 <=5	8 <= 6
>=	Greater than or equal to	7 >= 3	1 >= 2
!=	Not equal to	5 != 6	3 != 3

Table 2-4 Comparison operators

Understanding Numeric Type Conversion

- Arithmetic with variables or constants of the same type
  - Result retains the same type
- Arithmetic with operands of dissimilar types
  - C# chooses a **unifying type** for the result
  - **Implicitly** (or automatically) converts nonconforming operand(s) to the unifying type
    - Type with the higher **type precedence**
    - Can automatically convert Integral data types up the hierarchy to larger Integral and Floating Point types
      - Eg. **int** will convert to **long**, **float**, **double** or **decimal**.
    - For floating point data types, only **float** will automatically convert to **double**

Type	System Type	Bytes	Description	Largest Value	Smallest Value
byte	Byte	1	Unsigned byte	255	0
sbyte	Sbyte	1	Signed byte	127	–128
short	Int16	2	Signed short	32,767	–32,768
ushort	UInt16	2	Unsigned short	65,535	0
int	Int32	4	Signed integer	2,147,483,647	–2,147,483,648
uint	UInt32	4	Unsigned integer	4,294,967,295	0
long	Int64	8	Signed long integer	Approximately 9×10^{18}	Approximately -9×10^{18}
ulong	UInt64	8	Unsigned long integer	Approximately 18×10^{18}	0
float	Single	4	Floating-point	Approximately 3.4×10^{38}	Approximately -3.4×10^{38}
double	Double	8	Double-precision floating-point	Approximately 1.8×10^{308}	Approximately -1.8×10^{308}
decimal	Decimal	16	Fixed-precision number	Approximately 7.9×10^{28}	Approximately -7.9×10^{28}
char	Char	2	Unicode character	0xFFFF	0x0000
bool	Boolean	1	Boolean value (true or false)	NA	NA
string	String	NA	Unicode string	NA	NA
object	Object	NA	Any object	NA	NA

Table 2-1 C# data types

Understanding Numeric Type Conversion (cont'd.)

- **Implicit cast**
  - Automatic transformation that occurs when a value is assigned to a type with higher precedence
- **Explicit cast**
  - Placing the desired result type in parentheses
    - Followed by the variable or constant to be cast

      double result = 7 / 4; // result = 1.0

      double result = 7 / (**double**) 4; // result = 1.75

Using the `char` Data Type

- `char` data type
  - Holds any single character
- Place constant character values within single quotation marks

  char letter = 'r';
- **Escape sequence**
  - Stores a pair of characters
  - Begins with a backslash
  - Pair of symbols represents a single character

Using the `char` Data Type (cont'd.)

Escape Sequence	Character Name
`\'`	Single quotation mark
`\"`	Double quotation mark
`\\`	Backslash
`\0`	Null
`\a`	Alert
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab

Table 2-5 Common escape sequences

Using the `string` Data Type

- **`string`** data type
  - Holds a series of characters
    string name = “Richard”;
- Values are expressed within double quotation marks
- Comparing strings
  - Use `==` and `!=`
  - Methods `Equals()`, `Compare()`, `CompareTo()`

Using the `string` Data Type (cont'd.)

```
using System;
public class CompareNames1
{
   public static void Main()
   {
      string name1 = "Amy";
      string name2 = "Amy";
      string name3 = "Matthew";
      Console.WriteLine("compare {0} to {1}: {2}",
         name1, name2, name1 == name2);
      Console.WriteLine("compare {0} to {1}: {2}",
         name1, name3, name1 == name3);
   }
}
```

Figure 2-11 Program that compares two strings using == operator (not recommended)

Compare Amy to Amy: True
Compare Amy to Matthew: False

Using the `string` Data Type (cont'd.)

- Use the length property of a string to determine its length
  - The length of "`water`" is 5
- Use the `Substring()` method to extract a portion of a string from a starting point for a specific length

Using the `string` Data Type (cont'd.)

```
string word = "water";
```

Position:	0	1	2	3	4
	w	a	t	e	r

Start position   Length to extract

```
word.Substring(0, 1) is "w"
word.Substring(0, 2) is "wa"
word.Substring(1, 2) is "at"
word.Substring(2, word.Length - 1) is "ter"
word.Substring(0, word.Length - 1) is "water"
```

Figure 2-15 Using the `Substring()` method

Defining Named Constants

- **Named constant**
  - Often simply called a constant
  - An identifier whose contents cannot change
  - Created using the keyword `const`

```
const int INCHES_IN_A_FOOT = 12;
```

- Programmers usually name constants using all uppercase letters
  - Inserting underscores for readability
- **Self-documenting** statement
  - Easy to understand even without program comments

```
lengthInches = lengthFeet * INCHES_IN_A_FOOT;
```


Accepting Console Input

- **Interactive program**
  - A program that allows user input
- `Console.ReadLine()` method
  - Accepts user input from the keyboard
  - Accepts all of the characters entered by a user until the user presses **Enter**
  - Characters can be assigned to a `string`
    - i.e. all input is by default read is as a string!!
  - Must use a conversion method to convert the input string to the proper type

Accepting Console Input (cont'd.)

```
using System;
public class InteractiveSalesTax
{
   public static void Main()
   {
      const double TAX_RATE = 0.06;
      string itemPriceAsString;
      double itemPrice;
      double total;
      Console.Write("Enter the price of an item >> ");
      itemPriceAsString = Console.ReadLine();
      itemPrice = Convert.ToDouble(itemPriceAsString);
      total = itemPrice * TAX_RATE;
     Console.WriteLine("With a tax rate of {0}, a {1} item " +
        "costs {2} more.", TAX_RATE, itemPrice.ToString("C"),
         total.ToString("C"));
   }
}
```

Figure 2-16 InteractiveSalesTax program

Enter the price of an item >> 28.77
With a tax rate of 0.06, a $28.77 item costs $1.73 more.

Accepting Console Input (cont'd.)

Method	Description
ToBoolean()	Converts a specified value to an equivalent Boolean value
ToByte()	Converts a specified value to an 8-bit unsigned integer
ToChar()	Converts a specified value to a Unicode character
ToDecimal()	Converts a specified value to a decimal number
ToDouble()	Converts a specified value to a double-precision floating-point number
ToInt16()	Converts a specified value to a 16-bit signed integer
ToInt32()	Converts a specified value to a 32-bit signed integer
ToInt64()	Converts a specified value to a 64-bit signed integer
ToSByte()	Converts a specified value to an 8-bit signed integer
ToSingle()	Converts a specified value to a single-precision floating-point number
ToString()	Converts the specified value to its equivalent String representation
ToUInt16()	Converts a specified value to a 16-bit unsigned integer
ToUInt32()	Converts a specified value to a 32-bit unsigned integer
ToUInt64()	Converts a specified value to a 64-bit unsigned integer

Table 2-6 Selected Convert class methods

Summary

- Constant: cannot be changed after compilation
- Can display variable values with `Write()` or `WriteLine()`
- Nine integral data types: **`byte`, `sbyte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`**, and **`char`**
- Three floating-point data types: **`float`, `double`**, and **`decimal`**
- Use the binary arithmetic operators `+`, `–`, `*`, `/`, and `%` to manipulate values in your programs
- Shortcut arithmetic operators

Summary (cont'd.)

- A **`bool`** variable can be true or false
- Implicit cast versus explicit cast
- **`char`** data type holds any single character
- **`string`** data type holds a series of characters
- Named constants are program identifiers whose values cannot change.
- `Console.ReadLine()` method accepts user input