

# 3

## Introduction to C# Applications

### OBJECTIVES

- To write simple C# applications
  - To write statements that input and output data to the screen.
  - To declare and use data of various types.
  - To write decision-making statements.
  - To use relational and equality operators.

```
1 // Fig. 3.1: Welcome1.cs
2 // Text-printing application
3 using System;
4
5 public class Welcome1
6 {
7     // Main method begins execution of C# application
8     public static void Main( string[] args )
9     {
10        Console.WriteLine( "Welcome to C# Programming!" );
11    } // end method Main
12 } // end class Welcome1
```

Output: Welcome to C# Programming!

Annotations:

- // indicates the remainder of the line is a comment
- using directive helps compiler locate a class
- Begins a class declaration
- Main is the starting point of every application
- Print a string

### 3.2 A Simple C# Application: Displaying a Line of Text (Cont.)

```
1 // Fig. 3.1: Welcome1.cs
2 // Text-printing application.
```

- Comments start with: //
- Comments ignored during program execution
- Document and describe code
  - Provides code readability
- Traditional comments: /\* ... \*/

### 3.2 A Simple C# Application: Displaying a Line of Text (Cont.)

```
3 using System;
```

- **Keyword: using**
  - Helps the compiler locate a class that program will use
  - Identifies pre-defined class
  - All using directives must appear before any other code (except comments) in a C# source code file; otherwise a compilation error occurs.

### 3.2 A Simple C# Application: Displaying a Line of Text (Cont.)

- ```
4
```
- **Blank line**
    - Makes program more readable
    - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

- ```
5 public class Welcome1
```
- **Begins class declaration for class Welcome1**
    - Every C# program has at least one user-defined class
    - Keyword: words reserved for use by C#
      - **class** keyword followed by class name
    - Naming classes: capitalize every word
      - **SampleClassName**

C# Keywords			
abstract	as	Base	bool
byte	case	Catch	char
class	const	Continue	default
delegate	do	Double	else
event	explicit	Extern	finally
fixed	float	For	foreach
If	Implicit	In	Interface
Internal	Is	Lock	long
new	null	Object	operator
override	params	private	protected
readonly	ref	return	sbyte
short	sizeof	stackalloc	static
struct	switch	this	throw
try	typeof	uint	ulong
unsafe	ushort	using	virtual
volatile	while		void

Fig. 3.2 | C# keywords.

© 2006 Pearson Education, Inc. All rights reserved.

### 3.2 A Simple C# Application: Displaying a Line of Text (Cont.)

- C# identifier
  - Series of characters consisting of letters, digits and underscores ( \_ )
  - Does not begin with a digit, has no spaces
    - Valid: Welcome1, identifier, \_value, button7
    - 7button is invalid
  - C# is case sensitive (capitalization matters)
    - a1 and A1 are different

© 2006 Pearson Education, Inc. All rights reserved.

### 3.2 A Simple C# Application: Displaying a Line of Text (Cont.)

```
7 public static void Main( string[] args )
```

- Part of every C# console application
  - Applications begin executing at **Main**
    - Parentheses indicate **Main** is a method (Ch. 3 and 6)
    - C# applications contain one or more methods
  - Exactly one method must be called **Main**
- Methods can perform tasks and return information
  - **void** means **Main** returns no information

© 2006 Pearson Education, Inc. All rights reserved.

### 3.2 A Simple C# Application: Displaying a Line of Text (Cont.)

```
10 Console.WriteLine("Welcome to C# Programming!");
```

- Instructs computer to perform an action
  - Prints string of characters
    - String: series characters inside double quotes
    - White-spaces in strings are not ignored by compiler
- Method **Console.WriteLine**
  - Standard output
  - Print to command window (i.e., MS-DOS prompt)
  - Displays line of text
- This line is known as a statement
  - Statements must end with semicolon ;

© 2006 Pearson Education, Inc. All rights reserved.

### 3.4 Modifying Your Simple C# Application

```
1 // Fig. 3.14: Welcome2.cs
2 // Printing one line of text with multiple statements.
3 using System;
4
5 public class Welcome2
6 {
7     // Main method begins execution of C# application
8     public static void Main( string[] args )
9     {
10        Console.WriteLine("Welcome to C# Programming!");
11        Console.WriteLine("C# Programming!");
12    } // end method Main
13 } // end class Welcome2
```

Cursor stays on same line after outputting

Cursor moves to next line after outputting

Welcome to C# Programming!

© 2006 Pearson Education, Inc. All rights reserved.

Escape sequence	Description
\n	Newline. Positions the screen cursor at the beginning of the next line.
\t	Horizontal tab. Moves the screen cursor to the next tab stop.
\r	Carriage return. Positions the screen cursor at the beginning of the current line—does not advance the cursor to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
\\	Backslash. Used to place a backslash character in a string.
\"	Double quote. Used to place a double-quote character (") in a string. For example,

```
Console.WriteLine(@"\" in quotes");
```

displays

"in quotes"

- Escape characters
  - Backslash ( \ )
  - Indicates special characters be output

© 2006 Pearson Education, Inc. All rights reserved.

```

1 // Fig. 3.18: Addition.cs
2 // Displaying the sum of two numbers input from the keyboard.
3 using System;
4
5 public class Addition
6 {
7     // Main method begins execution of C# application
8     public static void Main( string[] args )
9     {
10        int number1; // declare first number to add
11        int number2; // declare second number to add
12        int sum; // declare sum of number1 and number2
13
14        Console.WriteLine( "Enter first integer: " ); // prompt user
15        // read first number from user
16        number1 = Convert.ToInt32( Console.ReadLine() );
17
18        Console.WriteLine( "Enter second integer: " ); // prompt user
19        // read second number from user
20        number2 = Convert.ToInt32( Console.ReadLine() );
21
22        sum = number1 + number2; // add numbers
23
24        Console.WriteLine( "Sum is {0}, sum ); // display sum
25    } // end method Main
26 } // end class Addition

```

14: using System; → using declaration imports necessary components from the System namespace.  
 10-12: int number1; // declare first number to add, int number2; // declare second number to add, int sum; // declare sum of number1 and number2 → Declare variables number1, number2 and sum.  
 14-16: Console.WriteLine( "Enter first integer: " ); // prompt user, // read first number from user, number1 = Convert.ToInt32( Console.ReadLine() ); → Convert user's input into an int, and assign it to number1.  
 18-20: Console.WriteLine( "Enter second integer: " ); // prompt user, // read second number from user, number2 = Convert.ToInt32( Console.ReadLine() ); → Convert user's next input into an int, and assign it to number2.  
 22: sum = number1 + number2; // add numbers → Calculate the sum of the variables number1 and number2, assign result to sum.  
 24: Console.WriteLine( "Sum is {0}, sum ); // display sum → Display the sum using formatted output.

© 2006 Pearson Education, Inc. All rights reserved.

### 3.6 Another C# Application: Adding Integers

- Variables
  - Location in memory that stores a value
    - Declare with name and type before use
  - Variable name: any valid identifier
- Declarations end with semicolons ;
- Initialize variable in its declaration
  - Equal sign: int a = 1;

© 2006 Pearson Education, Inc. All rights reserved.

### 3.6 Another C# Application: Adding Integers (Cont.)

```

10 int number1; // first number to add
11 int number2; // second number to add
12 int sum; // second number to add

```

- Declare variable **number1**, **number2** and **sum** of type **int**
  - **int** holds integer values (whole numbers): i.e., 0, -4, 97
  - Types **float**, **double** and **decimal** can hold decimal numbers
  - Type **char** can hold a single character: i.e., x, \$, \n, 7
  - **int**, **float**, **double**, **decimal** and **char** are simple types
- Can add comments to describe purpose of variables
 

```

int number1, // first number to add
number2, // second number to add
sum; // second number to add

```
- Can declare multiple variables of the same type in one declaration
  - Use comma-separated list

© 2006 Pearson Education, Inc. All rights reserved.

### 3.6 Another C# Application: Adding Integers (Cont.)

```

14 Console.WriteLine( "Enter first integer: " ); //prompt user
16 number1 = Convert.ToInt32( Console.ReadLine() );

```

- Message called a prompt: directs user to perform an action
- **ReadLine** waits for user to type a string of characters
- **Convert.ToInt32** method converts the sequence of characters into data of type **int**
- Result of call to **ReadLine** given to **number1** using assignment operator =
  - Assignment statement
  - = is a binary operator - takes two operands
    - Expression on right evaluated and assigned to variable on left
  - Read as: **number1** gets the value of `Convert.ToInt32(Console.ReadLine());`

© 2006 Pearson Education, Inc. All rights reserved.

### 3.6 Another C# Application: Adding Integers (Cont.)

```

18 Console.WriteLine( "Enter second integer: " ); // prompt user
20 number2 = Convert.ToInt32( (Console.ReadLine() );
22 sum = number1 + number2; // add numbers

```

- Similar to previous statement
  - Prompts the user to input the second integer
- Similar to previous statement
  - Assign variable **number2** to second integer input
- Assignment statement
  - Calculates sum of **number1** and **number2** (right hand side)
  - Uses assignment operator = to assign result to variable **sum**
  - Read as: **sum** gets the value of **number1 + number2**
  - **number1** and **number2** are operands

© 2006 Pearson Education, Inc. All rights reserved.

### 3.6 Another C# Application: Adding Integers (Cont.)

```

24 Console.WriteLine( "Sum is {0} ", sum ); //display sum
Console.WriteLine( "Sum is {0} ", (number1 + number2) );

```

- Use **Console.WriteLine** to display results
- {0} is placeholder for sum
- Calculations can also be performed inside **WriteLine**
- Parentheses around the expression **number1 + number2** are not required

© 2006 Pearson Education, Inc. All rights reserved.

## 3.7 Memory Concepts

### • Variables

- Every variable has a name, a type, a size and a value
  - Name corresponds to location in memory
- When new value is placed into a variable, replaces (and destroys) previous value
- Reading variables from memory does not change them



Fig. 3.19 | Memory location showing the name and value of variable **number1**.

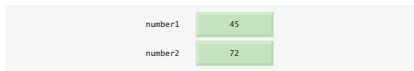


Fig. 3.20 | Memory locations after storing values for **number1** and **number2**.

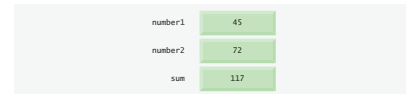


Fig. 3.21 | Memory locations after calculating and storing the sum of **number1** and **number2**.

## 3.8 Arithmetic

### • Arithmetic calculations used in most programs

- Usage
  - \* for multiplication
  - / for division
  - % for remainder
  - +, -
- Integer division truncates remainder
  - 7 / 5 evaluates to 1
- Remainder operator % returns the remainder
  - 7 % 5 evaluates to 2

C# operation	Arithmetic operator	Algebraic expression	C# expression
Addition	+	$f + 7$	<b>f + 7</b>
Subtraction	-	$p - c$	<b>p - c</b>
Multiplication	*	$b \cdot m$	<b>b * m</b>
Division	/	$x / y$ or $\frac{x}{y}$ or $x \div y$	<b>x / y</b>
Remainder	%	$r \text{ mod } s$	<b>r % s</b>

Fig. 3.22 | Arithmetic operators.

### 3.8 Arithmetic (Cont.)

#### • Operator precedence

- Some arithmetic operators act before others (i.e., multiplication before addition)
  - Use parenthesis when needed or for clarity
- Example: Find the average of three variables **a**, **b** and **c**
  - Do not use:  $a + b + c / 3$
  - Use:  $( a + b + c ) / 3$

Operator(s)	Operation(s)	Order of evaluation (associativity)
-------------	--------------	-------------------------------------

*Evaluated first*

*	Multiplication	If there are several operators of this type, they are evaluated from left to right.
/	Division	
%	Remainder	

*Evaluated next*

+	Addition	If there are several operators of this type, they are evaluated from left to right.
-	Subtraction	

Fig. 3.23 | Precedence of arithmetic operators.

### 3.9 Decision Making: Equality and Relational Operators

#### • Condition

- Expression can be either **true** or **false**

#### • if statement

- Simple version in this section, more detail later
- If a condition is **true**, then the body of the **if** statement executed
- Control always resumes after the **if** statement
- Conditions in **if** statements can be formed using equality or relational operators (Fig. 3.25)

Standard algebraic equality and relational operators	C# equality or relational Soperator	Sample C# condition	Meaning of C# condition
--	-------------------------------------	---------------------	-------------------------

*Equality operators*

=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

*Relational operators*

>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 3.25 | Equity and relational operators.

```

1 // Fig. 3.25: Comparison.cs
2 // Comparing integers using if statements, equality operators,
3 // and relational operators.
4 using System;
5
6 public class Comparison
7 {
8     // Main method begins execution of C# application
9     public static void Main( string[] args )
10    {
11        int number1; // declare first number to compare
12        int number2; // declare second number to compare
13
14        //prompt user and read first number
15        Console.WriteLine( "Enter first integer: " );
16        number1 = Convert.ToInt32( Console.ReadLine() );
17
18        //prompt user and read second number
19        Console.WriteLine( "Enter second integer: " );
20        number2 = Convert.ToInt32( Console.ReadLine() );
21
22        if ( number1 == number2 )
23            Console.WriteLine( "{0} == {1}", number1, number2 );
24
25        if ( number1 != number2 )
26            Console.WriteLine( "{0} != {1}", number1, number2 );
27
28        if ( number1 < number2 )
29            Console.WriteLine( "{0} < {1}", number1, number2 );
    }
}
    
```

Test for equality, display result using WriteLine.

Compares two numbers using relational operator <.

Compares two numbers using relational operator >, <= and >=.

Compares two numbers using relational operator >, <= and >=.

Outline

Comparison.cs  
(1 of 2)

```

30
31
32
33
34
35
36
37
38
39
40 // end class Comparison
    
```

```

Enter first integer: 42
Enter second integer: 42
42 == 42
42 <= 42
42 >= 42

Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000

Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
    
```

Compares two numbers using relational operator >, <= and >=.

Outline

Comparison.cs  
(2 of 2)

### 3.9 Decision Making: Equality and Relational Operators (Cont.)

31

- Line 6: begins class Comparison declaration
- Lines 11-12: declare `int` variables
- Lines 15-16: prompt the user to enter the first integer and input the value
- Lines 19-20: prompt the user to enter the second integer and input the value



© 2006 Pearson Education, Inc. All rights reserved.

### 3.9 Decision Making: Equality and Relational Operators (Cont.)

32

```
22     if ( number1 == number2 )
23         Console.WriteLine( "{0} == {1}", number1, number2 );
```

- `if` statement to test for equality using `==`
  - If variables equal (condition true)
    - Line 23 executes
  - If variables not equal, statement skipped
  - No semicolon at the end of `if` statement
  - Empty statement
    - No task is performed
- Lines 25-26, 28-29, 31-32, 34-35 and 37-38
  - Compare `number1` and `number2` with the operators `==`, `<`, `>`, `<=` and `>=`, respectively



© 2006 Pearson Education, Inc. All rights reserved.

### Common Programming Error 3.8

33

Confusing the equality operator, `==`, with the assignment operator, `=`, can cause a logic error or a syntax error. The equality operator should be read as “is equal to,” and the assignment operator should be read as “gets” or “gets the value of.” To avoid confusion, some people read the equality operator as “double equals” or “equals equals.”



© 2006 Pearson Education, Inc. All rights reserved.

### Common Programming Error 3.9

34

It is a syntax error if the operators `==`, `!=`, `>=` and `<=` contain spaces between their symbols, as in `=` , `! =`, `> =` and `< =`, respectively.

Reversing the operators `!=`, `>=` and `<=`, as in `!=` , `=>` and `<=`, is a syntax error.



© 2006 Pearson Education, Inc. All rights reserved.