Course Object Oriented Programming

Lecture 2

OOP with C#. Introduction C#. Data Types. Variables, expressions, statements. C# decision and iteration constructs.

C# programming language

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented(class-based), and component-oriented programming disciplines.

The core syntax of C# language is similar to that of other C-style languages such as C, C++ and Java. In particular:

- Semicolons are used to denote the end of a statement.
- Curly brackets are used to group statements. Statements are commonly grouped into methods (functions), methods into classes, and classes into namespaces.
- Variables are assigned using an equals sign, but compared using two consecutive equals signs.
- Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them.

Data Types

Data is the fundamental currency of the computer. All computer processing deals with analysis, manipulation and processing of data. Data is entered, stored and retrieved from computers. It is not surprising then, to learn that data is also fundamental to the C# language.

Data Types supported by C#

C# is a strongly typed language, that is, every object or entity you create in a program must have definite type. This allows the compiler to know how big it is (i.e. how much storage is required in memory) and what it can do (i.e. and thereby make sure that the programmer is not misusing it). There are thirteen basic data types in C#, note that 1 byte equals 8 bits and each bit can take one of two values (i.e. 0 or 1).

System Data Types

Data Type	Storage	.NET Type	Range of Values
byte	1	Byte	0 to 255
char	2	Char	unicode character codes (0 to 65,535)
bool	1	Boolean	True or false
sbyte	1	SByte	-128 to 127
short	2	Int16	-32,768 to 32767
ushort	2	UInt16	0 to 65,535
int	4	Int32	-2,147,483,648 to 1,147,483,647
uint	4	UInt32	0 to 4,294,967,295
float	4	Single	$\pm 1.5 * 10^{-45}$ to $\pm 3.4 * 10^{38}$, 7 significant figures
double	8	Double	$\pm 5.0 * 10^{-324}$ to $\pm 1.8 * 10^{308}$, 15 significant figures
decimal	12	Decimal	for financial applications
long	8	Int64	$-9*10^{18}$ to $9*10^{18}$
ulong	8	UInt64	0 to $1.8*10^{19}$

Variables

The memory locations used to store a program's data are referred to as variables because as the program executes the values stored tend to change.

Each variable has three aspects of interest, its:

1. type.
2. value.
3. memory address.

The data type of a variable informs us of what type of data and what range of values can be stored in the variable and the memory address tells us where in memory the variable is located.

Declaration of Variables

Syntax: <type> <name>;

Example

int i;

char a, b, ch;

All statements in C# are terminated with a semi-colon.

Naming of Variables

The names of variables and functions in C# are commonly called identifiers. There are a few rules to keep in mind when naming variables:

1. The first character must be a letter or an underscore.
2. An identifier can consist of letters, numbers and underscores only.
3. Reserved words (int, char, double, …) cannot be used as variable names.

In addition, please note carefully that C# is case sensitive. For example, the identifiers Rate, rate and RATE are all considered to be different by the C# compiler.

Initialize during variable declaration

Syntax: type var_name = constant;
Example
int i = 20; //i declared and given the value 20
char ch = ‘a’//ch declared and initialised with value .a.
int i = 2, j = 4, k, l = 5; //i, j and l initialised, k not initialised

Declare first then assign
Example
int i, j, k; //declare
i = 2; //assign
j = 3;
k = 5;

Escape sequences and their meaning.

Escape Sequence	Meaning
\0	null character
\a	audible alert
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab
\'	single quote
\"	double quotes
\\	back slash
\?	question mark

Console Input/Output (I/O)

Output

Syntax:

Console.WriteLine(<control_string>,<optional _other_arguments);

For example:

Console.WriteLine("Hello World!");

Console Input/Output (I/O)

```
int a = 2, b = 3, c = 0;
c=a+b;
Console.WriteLine("c has the value {0}", c);
Console.WriteLine("{0} + {1} = {2}", a, b, c);
```

Here the symbols {0}, {1} etc. are placeholders where the values of the optional arguments are substituted.

Console Input/Output (I/O)

Input
Syntax: string Console.ReadLine();

The string before the method means that whatever the user types on the keyboard is returned from the method call and presented as a string.
It is up to the programmer to retrieve that data. An example is:

```
string input = "";
int data = 0;
Console.WriteLine("Please enter an integer value: ");
Console.ReadLine(); //user input is stored in the string input.
data = Convert.ToInt32(input);
Console.WriteLine("You entered {0}", data);
```

Operators

A strong feature of C# is a very rich set of built in operators including arithmetic, relational, logical and bitwise operators.

Assignment =

Syntax: <lhs> = <rhs>;

where lhs means left hand side and rhs means right hand side.

Example

```
int i, j, k;
i = 20; // value 20 assigned to variable i
i = (j = 25); /* in C#, expressions in parentheses are always evaluated
first, so j is assigned the value 25 and the result of this assignment
    (i.e.
25) is assigned to i */
i = j = k = 10;
```

Arithmetic Operators

Arithmetic Operators (+, -, *, /, %)

+ addition

- subtraction

* multiplication

/ division

% modulus

+ and - have unary and binary forms, i.e. unary operators take only one operand, whereas binary operators require two operands.

Example

```
x = -y; // unary subtraction operator
p = +x * y; // unary addition operator
x = a + b; // binary addition operator
y = x - a; // binary subtraction operator
```

Increment and Decrement operators (++, - -)

Increment (++) and decrement (- -) are unary operators which cause the value of the variable they act upon to be incremented or decremented by 1 respectively. These operators are shorthand for a very common programming task.

Example

x++; //is equivalent to x = x + 1;

++ and - - may be used in prefix or postfix positions, each with a different meaning. In prefix usage the value of the expression is the value after incrementing or decrementing. In postfix usage the value of the expression is the value before incrementing or decrementing.

Example

int i, j = 2;

i = ++j; // both i and j have the value 3

i = j++; // now i = 3 and j = 4

Special Assignment Operators (+=, -=, *=, /=, %=, &=)

Example

x += i + j; // this is the same as x = x + (i + j);

These shorthand operators improve the speed of execution as they require the expression and variable to be evaluated once rather than twice.

Statements

Expression Statements

x = 1;*//simple statement*

Console.WriteLine(.Hello World!.);*//also statement*

x = 2 + (3 * 5) – 23;//*complex statement*

Compound Statements or Blocks

{

statement

statement

statement

}

Decision Statements

If statement

syntax 1:

```
if ( condition )
    true statement block;
else
    false statement block;
```

syntax 2:

```
if ( condition )
    true statement block;
```

Example

```
int numerator, denominator;
Console.WriteLine(“Enter two integer values for the numerator and
    denominator”);
numerator = Convert.ToInt32(Console.ReadLine());
denominator = Convert.ToInt32(Console.ReadLine());
if (denominator != 0)
Console.WriteLine(“{0}/{1} = {2}”, numerator, denominator,
    numerator/denominator);
else
Console.WriteLine(“Invalid operation can’t divide by 0”);
```

The statement body can include more than one statement but make sure they are group into a code block i.e. surrounded by curly braces.

Example

```
int x, y, tmp;
Console.WriteLine(“Please enter two integers”);
x = Convert.ToInt32(Console.ReadLine());
y = Convert.ToInt32(Console.ReadLine());
if ( x > y)
{
tmp = x;
x = y;
y = tmp;
}
```

Nested if Statement

Nested if statements occur when one if statement is nested within another if statement.

Example

```
if (x > 0)
if ( x > 10)
Console.WriteLine(“x is greater than both 0 and 10”);
else
Console.WriteLine(“x is greater than 0 but less than or equal to 10”);
else
Console.WriteLine(“x is less than or equal to 0”);
```

if - else - if operator

If a program requires a choice from one of many cases, successive if statements can be joined together to form a if - else - if ladder.

```
if ( expression_1 )
statement_1;
else if ( expression_2 )
statement_2;
else if ( expression 3 )
statement_3;
else if (condition_n)
statement_n;
else
statement_default;
```

EXAMPLE

```
int age;
Console.WriteLine("Please enter your age\n\n");
age = Convert.ToInt32(Console.ReadLine());
if ( age > 0 && age <= 10 )
Console.WriteLine("You are a child?\n");
else if ( age > 10 && age <= 20 )
Console.WriteLine("You are a teenager?\n");
else if ( age > 20 && age <= 65 )
Console.WriteLine("You are an adult?\n");
else if ( age > 65 )
Console.WriteLine("You are old!\n");
```

Conditional Operator ?:

There is a special shorthand syntax that gives the same result as

if (expression)
true_statement;
else
false_statement;

***syntax**: expression ? true_statement : false_statement;*

The ?; requires three arguments and is thus ternary. The main advantage of this operator is that it is succinct.

Example

```
max = x >= y ? x : y;
```

which is the equivalent of

```
if ( x >= y)
max = x;
else
max = y;
```

Switch Statement

This statement is similar to the if-else-if ladder but is clearer, easier to code and less error prone.

```
syntax:
switch( expression )
{
case constant1:
statement1;
break;
case constant2:
statement2;
break;
default:
default_statement;
}
```

Example

```
double num1, num2, result;
char op;
Console.WriteLine(“Enter number operator number \n”);
num1 = Convert.ToInt32(Console.ReadLine());
op = Convert.ToChar(Console.ReadLine());
num2 = Convert.ToInt32(Console.ReadLine());
switch(op)
{
case “+”:
result = num1 + num2;
break;
case “-”:
result = num1 - num2;
break;
case “*”:
result = num1 * num2;
break;
case “/”:
if(num2 != 0)
{
result = num1 / num2;
break;
} //else fall through to error statement
default:
Console.WriteLine(“ERROR- invalid operation or divide by 0.0 \n”);
}
Console.WriteLine(“{0} {1},{2} = {3}\n”, num1, op, num2, result);
```

Iterative Statements

- For statement
- While statement
- Do while statement
- Break statement
- Continue statement

The while Looping Constructs

- The while looping construct is useful should you wish to execute a block of statements until some terminating condition has been reached. Within the scope of a while loop, you will need to ensure this terminating event is indeed established; otherwise, you will be stuck in an endless loop. In the following example, the message “In while loop” will be continuously printed until the user terminates the loop by entering *yes* at the command prompt:
-
- *static void ExecuteWhileLoop()*
- *{*
- *string userIsDone = "";*
- *// Test on a lower-class copy of the string.*
- *while(userIsDone.ToLower() != "yes")*
- *{*
- *Console.Write("Are you done? [yes] [no]: ");*
- *userIsDone = Console.ReadLine();*
- *Console.WriteLine("In while loop");*
- *}*
- *}*

The do/while Looping Constructs

- Closely related to the while loop is the do/while statement. Like a simple while loop, do/while is used when you need to perform some action an undetermined number of times. The difference is that do/while loops are guaranteed to execute the corresponding block of code at least once. In contrast, it is possible that a simple while loop may never execute if the terminating condition is false from the onset.
-
- *static void ExecuteDoWhileLoop()*
- *{*
- *string userIsDone = "";*
- *do*
- *{*
- *Console.WriteLine("In do/while loop");*
- *Console.Write("Are you done? [yes] [no]: ");*
- *userIsDone = Console.ReadLine();*
- *}while(userIsDone.ToLower() != "yes"); // Note the semicolon!*
- *}*

Decision Constructs

- C# defines two simple constructs to alter the flow of your program, based on various contingencies:

- -The if/else statement
- -The switch statement

- *C# Relational and Equality Operators*

C# Equality/Relational Operator	Example Usage	Meaning in Life
==	`if(age == 30)`	Returns `true` only if each expression is the same.
!=	`if("Foo" != myStr)`	Returns `true` only if each expression is different.

<
>
<=
>=
	`if(bonus < 2000)`
	`if(bonus > 2000)`
	`if(bonus <= 2000)`
	`if(bonus >= 2000)`
		Returns `true` if expression A (`bonus`) is less than, greater than, less than or equal to, or greater than or equal to expression B (`2000`).

- *Logical operators*

Logical Operator	Example	Meaning in Life
&&	`if(age == 30 && name == "Fred")`	AND operator. Returns `true` if all expressions are `true`.
||	`if(age == 30 || name == "Fred")`	OR operator. Returns `true` if at least one expression is `true`.
!	`if(!myBool)`	NOT operator. Returns `true` if `false`, or `false` if `true`.

The if/else statement

- *static void IfElseExample()*
- *{*
- *// This is illegal, given that Length returns an int, not a bool.*
- *string stringData = "My textual data";*
- *if(stringData.Length)*
- *{*
- *Console.WriteLine("string is greater than 0 characters");*
- *}*
- *}*

The switch Statement

```
// Switch on a numerical value.
static void ExecuteSwitch()
{
Console.WriteLine("1 [C#], 2 [VB]");
Console.Write("Please pick your language preference: ");
string langChoice = Console.ReadLine();
int n = int.Parse(langChoice);
switch (n)
{
case 1:
Console.WriteLine("Good choice, C# is a fine language.");
break;
case 2:
Console.WriteLine("VB: OOP, multithreading, and more!");
break;
default:
Console.WriteLine("Well...good luck with that!");
break;
}
}
```