FUNDAMENTALS OF PROGRAMMING WITH C#

VARIABLE, DATA TYPES AND OPERATORS

Liu Fan
isslf@nus.edu.sg

Total:68

Objectives

- Understand the concepts of data types in C#
- Understand the concept of variables and able to use variables in solving the problem
- Write program that perform mathematical and string operation using appropriate operators

Agenda

- Types
- Variables
- Data Types
- Operators
  - String operators
  - Numeric operators

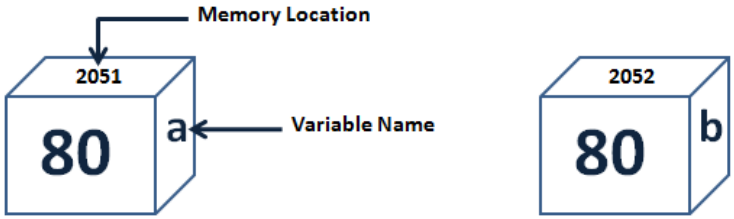
Types and Variables

- There are two kinds of variable of types in C#: *value types and reference types*.
  - Variables of *value types* directly contain their data whereas variables of *reference types* store references to their data (object).
- With *value types*, the variables each have their own copy of the data, and it is not possible for operations on one to affect the other (except in the case of *ref* and *out* parameter variable)
- With *reference types*, it is possible for two variables to reference the same object and thus possible for operations on one variable to affect the object referenced by the other variable.

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/types-and-variables

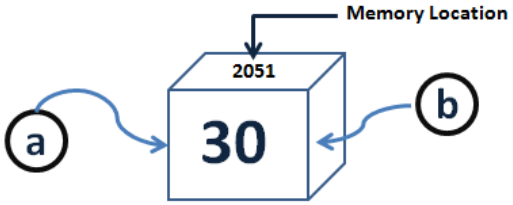
Value Type and Reference Type

VALUE TYPE - Example




```
int a;
a=80;
b=a;
```
← This will just create a copy of "a" in other memory location with variable named "b"

Reference Type - Example




```
Employee a=new Employee();
a.age = 26;
Employee b=a;
b.age = 30
```
← This will just create a new variable which will point to same location as "a" points.

Value Type stores its contents in memory allocated on the stack. When you created a Value Type, a single space in memory is allocated to store the value and that variable directly holds a value. If you assign it to another variable, the value is copied directly and both variables work independently.

Reference Types are used by a reference which holds a reference (address) to the object but not the object itself. Because reference types represent the address of the variable rather than the data itself, assigning a reference variable to another doesn't copy the data. Instead it creates a second copy of the reference, which refers to the same location of the heap as the original value. Reference Type variables are stored in a different area of memory called the heap.

https://manojbhoir.wordpress.com/2015/09/29/value-type-and-reference-types/

Value Types

- Value types
  - Simple types
    - Signed integral: short, int, long
    - Unsigned integral: byte, ushort, uint, ulong
    - Unicode characters: char
    - IEEE binary floating-point: float, double
    - High-precision decimal floating-point: decimal
  - Boolean: bool
  - Enum types
    - User-defined types of the form enum E {...}
  - Struct types
    - User-defined types of the form struct S {...}

Reference Types

- Reference types
  - Class types
    - Ultimate base class of all other types: object
    - Unicode strings: string
    - User-defined types of the form class C {...}
  - Interface types
    - User-defined types of the form interface I {...}
  - Array types
    - Single- and multi-dimensional, for example, int[] and int[,]
  - Delegate types
    - User-defined types of the form delegate int D(...)
  - String (Will cover later)

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types

Variables

- In computing, variable has a slightly different meaning than in mathematics
  - Can be confusing and misleading if you apply the wrong definition
- Variables are:
  - Containers that are named
  - Contain values of a particular type
    - E.g. int variables and bool variables hold different kinds of value
  - Values can be assigned and modified in the program
    - Variables that cannot be modified are called constants
    - Assignment is done using equal sign **=**
      - A = 5; statement assign the value 5 to a variable called A

Variable Names

- As a convention, variable name typically starts with lowercase and use a camel case style. Example:
  - thisIsAVariableFollowingCamelCaseStyle
  - this_variable_does_not_follow_camel_case
- Must follow the rules of valid identifiers in C#, some of the important ones:
  - Can be made up of letters, numbers and a few special characters.
  - They must begin with a letter or underscore _
  - Cannot contains characters used as an operator in C#
  - Cannot use names that is already used by C# language (C# reserved words)
    - There's a way to overcome this in C# if necessary, but it's best to have a habit to avoid reserved words of programming languages

Variable Names

- Identifiers:
  - cannot start with a number: 1route, 2abc
  - Cannot use C# reserved words: int
  - Cannot include whitespace: first name
  - Use meaningful name: fn (firstname)
  - Can start with letters or _
  - Call identifier in C# is case sensitive: number and Number are different identifiers
- Naming conventions
  - Camel case: firstName
  - Pascal case: FirstName

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/identifier-names

Value Types

- Built-in numeric types
- String/integer
- Double
- Boolean

Data Types

- Common built-in C# numeric types

Types	Used for
int	Whole number (**int**egral/integer number with 32 bit)
long	**Long** integer (64 bit in C#)
float	**Float**ing point number (real number) with single precision
double	Floating point number with **double** precision
decimal	Floating point number with higher precision than double
string	Text value (**string** of characters)
char	A single **char**acter
bool	**Bool**ean (true or false) value
byte	One **byte** (8 bits) integer 0-255

String

- Declared by the keyword `string`
- Hold text data such as names, address, book title, etc.
- Cannot be used for mathematical calculation
- Can hold Unicode characters
  - Unicode is an encoding standard to encode the characters used in most of the world's writing systems
- Can hold a very large amount of text
  - Maximum size is not documented, but there are reports where they can allocate text with more than 100 millions characters – you are probably doing something wrong if you need a string of this size.

Integer

- Declared by the keyword `int`
  - `int` is an alias of `System.Int32`
- Holds whole number both positive and negative number
- Cannot hold real number/floating point number such as 3.2 or 5.6
- Range from -2,147,483,648 to 2,147,483,647
- Uses 32-bit

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/int

Double

- Declared by the keyword `double`
  - `double` is an alias of `System.Double`
- Holds 64-bit floating point number
  - Uses double the memory space of a 32-bit floating point number (thus the name)
- Is limited to values between -1.798×10^{308} to -4.9407×10^{-324} for negative numbers and 4.9407×10^{-324} to 1.7977×10^{308} for positive numbers
- 15-17 digits precision
- Is the default data type to represent real number

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/double

Very High Precision Numbers

- Declared by the keyword `decimal`
- Holds numbers that are larger or need to be more precise than double precision variables allow.
  - Used predominantly in financial situations where precision needs to be high and number of significant digits is important; i.e., cannot be converted to Exponent Notation.
- Use with caution as has some limitations and lack of flexibility.
- Is limited to values between 1.0×10^{-28} to 7.9×10^{28}
- 28 to 29 significant digits
- Uses 128-bits = 16 bytes

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/decimal

Logical (Boolean) value

- Declared by keyword `bool`
- Holds `true` or `false`

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/bool

Declaring Variables

- Declaring a variable allow the computer to prepare the variable before we can use them
- We do this using the declaration statement:

Example	Explanation
`double weight;`	Declare a variable called "weight" that stores double value
`int age;`	Declare a variable called "age" that stores whole number value
`string myname;`	Declare a variable called “myname" that stores string value

- All variables must be declared.

Declaring Variables

- We can declare multiple variables in a single line:

```
double weight, height;
```

This declares two variables called “weight” & "height" of type “Floating point double precision numeric”.

The above is equivalent to:

```
double weight;
double height;
```

- People generally group their declaration together to improve readability
  - Many people like to have some variable declaration in the beginning of their code
- Variable have scope (it doesn't exist forever) - will be discussed in later lesson.

Assigning Values

- How do we assign a value to a variable?

```
string name;
double weight, height;
name = "John"
weight = 68.2;
height = 175;
```

- Can we assign integer value into a double variable?
  - Yes. Since integer values are subset of what can be stored in double variable.
  - The reverse is illegal and the program compilation would fail.
  - What we did is referred as "widening conversion" which is accepted.

Assigning Values

- Will these codes work?

```
string name;
int weight, height;
name = “John";
weight = 68.2;
height = 175;
```

```
int weight;
weight = 68.0;
```

Assigning Values

- Will these codes work?

```
string name;
int weight, height;
name = “John";
weight = 68.2;
height = 175;
```

```
int weight;
weight = 68.0;
```

You are not allowed to store double value in an integer variable. 68.0 is also considered as double value because the way we write it in the program suggest that we want C# to treat the value as a double.

Type conversion

- Implicit type conversion
- Explicit type conversion (casting)
- Conversion between non-compatible types

Implicit type conversion

- Example 1:

```
int val1 = 10;
double val2  = val1;
```

- Example 2:

```
int val1 = 10;
double val2 = val1;
int val3 = val2; //won't compile
int val3 = (int) val2;
```

Explicit Type Conversion (Casting)

- We can explicitly ask C# to convert a double value to an integer with the consequence that we will lose the fractional portion.

```
int weight, height;
weight = (int) 68.2;  // weight becomes 68
height = (int) 175.7; // height becomes 175
```

- Please note that it is always truncated.
- Please note that the prefix (int) would convert the value succeeding it to integer which is the data type of the variable on LHS.
- These are called **explicit conversions** or **casting**

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions

Conversion and Parsing for other data types (Non-compatible types)

- Casting works for data types that belong to the same "family" (not a formal concept in C#)
- For other types of conversion, we need to use or write a conversion method.
  - Example: Convert.ToInt32() to convert string into int

```
int iWeight;
string sWeight;
sWeight = "68" // sWeight assigned "68" a string
iWeight = Convert.ToInt32(sWeight);
                  // iWeight becomes 68 an integer
```

Conversion and Parsing for other data types (Non-compatible types)

Data Type	Example
double	`double value = double.Parse(input);` `double value = Convert.ToDouble(input);`
int	`int value = int.Parse(input);` `int value = Convert.ToInt32(input);`
bool	`bool value = bool.Parse(input);` `bool value = Convert.ToBoolean(input);`

Initialising Values

- Initialising = assigning an initial values to variable

- Why should we initialise?
  - C# requires "definite assignment" before a variable value can be obtained – initialization can prevent certain kind of errors from being raised
  - You may get these errors once you use conditionals and loops in your program.

- Initialization can be done together with declaration

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/variables

Initialisation Examples

- Initialisation example:

```
double weight = 0;
```

Here the variable weight is declared and assigned is a single statement. This is equivalent to:

```
double weight;
weight = 0;
```

Initialisation Examples

- Do these two codes mean the same thing?

```
double weight = 0, height = 0;
```

```
double weight, height = 0;
```

Initialisation Examples

- Do these two codes mean the same thing?

```
double weight = 0, height = 0;
```

```
double weight, height = 0;
```

- In the second code, weight variable is still unassigned with value.

Initialisation Examples

- Can we initialise multiple variables declared in single statement?
  - Yes:

```
double weight = 0, height = 0;
```

  - Here the variable weight and height are declared and initialised in a single statement.

- Does this work too?

```
double weight, height = 0;
```

  - There is NO error in the above statement. But it is not the same as the one above. Whereas in the first case both height and weight are declared and assigned, in the second both are declared but only height is initialised, weight continues in an un-initialised state.

Expressions

- Expressions are used to calculate values:

  8 * 9 22 - 13

- These values are often loaded into variables:

  X = 8 * 9 Y = 22 - 13

- Variables can be used in expressions:

  X = 8 * Y Z = X - 13

- Keep in mind that variables hold a single value. A variable can, therefore, be used in an expression to calculate a value that is then loaded into the same location:

  X = X + 1

Expressions and Operators

- Expressions are built up using operators and operands. These concepts should be familiar to you from your math classes.

- Depending on the operators used and the purpose of the expression, they fall into three general groups:
  - Mathematical
  - Comparison
  - Logical

- These types may be mixed.

Kinds of Expressions

- Mathematical Expressions
  - Used to calculate values.
- Comparison Expressions
  - Compare two values and evaluate a relationship between them.
- Logical Expressions
  - Combine other expressions to form more sophisticated operations.

Mathematical Operators

Operator	Operation	Operands Type	Result Type / Comments
*	Multiplication	Any Numeric	Precision corresponding to operand having highest precision.
/	Division	Numeric with at least one of them non-integer	Precision corresponding to operand having highest precision.
/	Integer Division	Both Integers	Integer; Result is truncated.
%	Remainder	Both Integer	Integer; Gives the remainder.
%	Remainder	Numeric (at least one not Integer)	Precision corresponding to operand with highest precision. Gives the remainder.
+	Addition	Any Numeric.	Precision corresponding to operand having highest precision.
-	Subtraction	Any Numeric.	Precision corresponding to operand having highest precision.

Math Operators: Notes

- Note:
  - Those familiar with other languages such as VB would notice that C# does not have ^ (Power or Exponentiation) symbol. This operation should be handled by the Math function.
  - The result's data type is largely dictated by the data type of the operands.
  - Division and Modulus operations are "overloaded" to perform different operations depending on whether operations are integers or real (non-integer) numbers.
  - While using constants or operands, a number without a decimal point is always interpreted as Integer and a number with a decimal point is always (by default) interpreted as Double.
    - Eg: 7 is an Integer; 1932 is an Integer
    - Eg: 7.0 is Double; 7.1 is Double; 187.32122 is Double
  - When one or both operands are strings, the plus (+) sign when used in between them is understood as Concatenation and not Addition.

Division operators

Expression	Result	Division Type
`7.0  / 3.0`	`2.33333333`	
`7.0 / 3`	`2.33333333`	
`7 / 3.0`	`2.33333333`	
`7  / 3`	`2`	
`5.0 / 2`	`2.50000000`	
`12.0 / 3.0`	`4.00000000`	
`12 / 3`	`4`	
`3.0 / 4.0`	`0.75000000`	
`3 / 4`	`0`	

Division operators

Expression	Result	Division Type
`7.0  / 3.0`	`2.33333333`	Normal
`7.0 / 3`	`2.33333333`	Normal
`7 / 3.0`	`2.33333333`	Normal
`7  / 3`	`2`	Integer
`5.0 / 2`	`2.50000000`	Normal
`12.0 / 3.0`	`4.00000000`	Normal
`12 / 3`	`4`	**Integer**
`3.0 / 4.0`	`0.75000000`	Normal
`3 / 4`	`0`	**Integer**

Remainder operator

Expression	Result
7.0 % 3.0	1.00000000
7.0 % 3	1.00000000
7 % 3.0	1.00000000
7 % 3	1
12.0 % 3.0	0.00000000
12 % 3	0
3.0 % 4.0	3.00000000
3 % 4	3
14.3 % 3.0	2.30000000

More Mathematical Operators

Operator	Operation	Operands Type	Result Type / Comments
+	Unary Plus	One Numeric to the right of operator.	Same as operand.
-	Unary Minus (Negation)	One Numeric to the right of the operator.	Same as operand.
++	Increment	One numeric variable either preceding OR succeeding the operator.	Same as operand.
--	Decrement	One numeric variable either preceding OR succeeding the operator.	Same as operand.

Unary Operators

- Unary
  - Unary Plus and Unary Minus operators are used as prefix to numeric constant or variable to indicate whether it is positive or negative.
  - Unary Plus is the default if the variable/constant does not have a prefix unary operator (as with mathematics).

- Examples:

```
-5.1
+7.2
-a
+a
```

Increment Operators

- Increment
  - This is a special type of operator that would increase the value by one.
  - This operator can be used preceding a numeric variable or constant.
  - The numeric can be integer (most common) or any other numeric.
  - The increment is done by using two successive plus signs either before or after the variable constant. E.g. `++count` or `count++`
  - There is a subtle difference between the prefix and suffix notation.

- Decrement
  - The decrement operator is complement to the increment operator.
  - The concepts are same; only difference is that instead of increasing the value by unity, the value is decreased by unity in decrement.
  - Use successive minus signs for decrement. Eg `--num` or `num--`

Increment Operator Working

Program:

```
int k = 5;
Console.WriteLine(k);
k = k + 1;
Console.WriteLine(k);
```

Output:

```
5
6
```

Comments

This traditional way of adding one to k and assigning it to the same memory location i.e., k

```
int k = 5;
Console.WriteLine(k);
k++;
Console.WriteLine(k);
```

```
5
6
```

The result is the same as above. But this is considered more efficient. The reason is full addition operation is not performed

Increment Examples

Program:

```
int k = 5;
Console.WriteLine(k);
k++;
Console.WriteLine(k);
Console.WriteLine(k++);
Console.WriteLine(k);
```

Output:

```
5
6
6
7
```

Comments

Increment takes place after operation.

In the third write statement k is incremented after WriteLine operation takes place.

```
int k = 5;
Console.WriteLine(k);
++k;
Console.WriteLine(k);
Console.WriteLine(++k);
Console.WriteLine(k);
```

```
5
6
7
7
```

Increment takes place before operation.

In the third write statement k is incremented before WriteLine operation takes place.

Decrement Examples

Program:

```
int k = 5;
Console.WriteLine(k);
k--;
Console.WriteLine(k);
Console.WriteLine(k--);
Console.WriteLine(k);
```

Output:

```
5
4
4
3
```

Comments

decrement takes place after operation.

In the third write statement k is decremented after WriteLine operation.

```
int k = 5;
Console.WriteLine(k);
--k;
Console.WriteLine(k);
Console.WriteLine(--k);
Console.WriteLine(k);
```

```
5
4
3
3
```

decrement takes place before operation.

In the third write statement k is decremented before WriteLine operation.

Math Operators Precedence

- Order of Precedence tells us which operators to evaluate first.

1	+, -, ++k, --k	Unary Increment/Decrement operations
2	* , /, %	Multiplication, Division, Modulo
3	+ , -	Addition, Subtraction

  - Where an expression has two or more operators having the same level of precedence then the operation proceeds from left to right.
  - Similar to algebraic expressions, bracket may be used to force an operation having lower precedence to be performed ahead of an operation having higher precedence.
  - You may use only simple parenthesis i.e. (…) and should not use square or curly brackets for this purpose. You may nest parentheses.

Comparison Operators

There are six basic comparison (relational) operators

<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
==	Equal To
!=	Not Equal To

Use of Comparison Operators

Comparison Operators are used to ask questions:

Is the value in X bigger than the value in Y?

```
X > Y
```

Is the value in Y at least as large as the value in Z?

```
Y >= Z
```

Is the value in X the same as the value in Z?

```
X = = Z
```

Values of Comparison Expressions

Expression that use comparison operators all return one of two values: true or false.

Either the relationship holds or it doesn't

Either A > B, so the result is True

if A isn't greater than B, then the result is False

Eg: if A is 5 and B is 3 then (A > B) is TRUE

if A is 5 and B is 7 then (A > B) is FALSE

if A is 5 and B is 5 then (A > B) is FALSE

Logical Operators

There are three logical operators:

&&	AND
||	OR
!	NOT

They are used to combine relational and other logical expressions so we can frame composite conditions/expressions.

AND and OR operations have Relational/Boolean operands on either side.

NOT operations have one Relational/Boolean operand to its right.

Truth Tables

Truth Table for AND operation:

A	B	A && B
True	True	True
False	True	False
True	False	False
False	False	False

Truth Table for OR operation:

A	B	A II B
True	True	True
False	True	True
True	False	True
False	False	False

Truth Table for NOT operation:

A	! A
True	False
False	True

Logical Operators Example

To enrol at ISS you must have a Degree and Work Experience:

(Degree == “Yes”) && (Experience > 2)

If you do not have a Degree then you cannot get admission (i,e., Admission is FALSE) even if you have a lot of work experience.
If you have a degree but do not have work experience again you do not get admission (i.e., Admission is FALSE).
In the above example, the resultant will be TRUE only if both the relational expressions are TRUE.

That is,

If the person does not have a degree but has 3 years experience then Degree has a value "No" and Experience has a value 3.
Then you are checking:

Is ("No" == "Yes")? && Is (3 > 2)?

False && True

which evaluates to FALSE.

Logical Operators Example

For travel in MRT you need either a EZ-Link card or must have cash.

(ezLinkCard == "Yes") || (Cash > $2)

If you have an EZ-Link card (with value assumed) but no cash, you can travel
If you do not have an EZ-Link card but have some money (assume $2 would cover most journeys) then again you can travel
However if you do not have an EZ-Link card nor do you have money - you cannot travel.
The full expression becomes FALSE only when both the relational operations are FALSE.
That is,

Assume ezLinkCard is "No" and Cash is $30. Then you are checking:

Is ("No" == "Yes") ? || Is (Cash > $2) ?

False || True

which evaluates to TRUE.

Logical Operators Example

To obtain the Diploma student should have an acceptable CPA, completed internship and should not fail in any subject.

There are two formats of writing the expression

1. (CPA >= 2.5) && (Internship = “Yes”) && (! (Failed == “Yes”))

2. (CPA >= 2.5) && (Internship = “Yes”) && (Failed != “Yes”)

This is a multiple AND condition let us concentrate on the last expression:

Failed == “Yes” means student has Failed the exam;
Failed == “No” means student has passed the exam.

In the above assume that Failed = "No"

Case1: So (Failed == “Yes”) is FALSE since Failed has a value "No".
Hence (! (Failed == “Yes”)) is TRUE

Case2: (Failed != “Yes”) is TRUE since Failed has a value "No".

Both options give the same result.

Built In Class Libraries

- Languages usually support libraries where some standard programs are stored.
  - In the flow charting example we had seen the iterative procedure for determining the square root.
  - However you need not write this every time you require it since most languages provide this to you as a library function - you just need to call it.
- In pure OO languages like C#, Java etc., the libraries are not provided as functions but as Class Libraries.
  - You hence need to use the appropriate Classes and Methods.
  - Most often you would be using the Math Class Library for mathematical operations.

Class Library

- Things you should know:
  - Where can I get the required function?
    - Refer C# Documentation.
    - Mathematical Libraries under the Math Class
    - String Operation Libraries under the String Class etc.
    - Within the class look up for the Method that does the operation
  - What is the composition of Method:
    - Method Type - we confine to what are called Static methods only now.
    - Method Name
    - Method Return data type
    - Method arguments: number of arguments, type and sequence.
  - How do I use this Static Method?
    - variable = ClassName.MethodName(Arguement1, Arg2, …)

Class Library

- How do I use this Static Method?

```
variable = ClassName.MethodName(Arguement1, Arg2, …)
```

  - Note:
    - Class Name and Method Name should be as given in documentation.
    - Class Name and Method Name are separated by a period (dot).
    - Variable is a variable that you have declared in your program.
    - The variable type should match with Return Data Type of the method.
    - Method may have more than one arguments.
    - Note the sequence, data type and pass in appropriate matching values.
    - A method may be overloaded to perform variations in the tasks. Choose the one that is most suitable.
    - The result of the action is stored in the variable.

Math Library

- What is the Math Library?
  - Math Library is a .Net Framework Class Library used in C#.
  - This Library provides static methods (functions) for common mathematical operations like square root, exponentiation (power), trigonometric functions etc.

- How do I use it?
  - The Math class is part of the System Namespace; So you can invoke these methods either by:

```
Variable = System.Math.MethodName( Arg )
```

  - OR

```
using System;
Variable = Math.MethodName( Arg )
```

Math Library Examples

- To find the square root of a number use the Sqrt method:

Program:

```
double A = 9.5; double B,C;
B = System.Math.Sqrt(A);
Console.WriteLine(B);
C = 5.7 * System.Math.Sqrt(A);
Console.WriteLine(C);
Console.WriteLine(System.Math.Sqrt(9));
```

Output:

```
3.08220700148449
17.5685799084616
3.00000000000000
```

This is the square root of A, where A is 9.5

This is 5.7 times the square root of A. Part of an expression.

This is the square root of 9 - Integer argument accepted by promotion & directly printed.

- The value returned is the square root of the argument.
- The method Sqrt in Math class takes in a numeric (integer, float or double) but always returns a double data type.
- Note the the method can be used independently or as part of an expression or as an argument.

Commonly Used Math Methods

- Common Numeric Methods in Math Class:

Method	Meaning
Abs	Overloaded. Returns the absolute value of a specified number. Return data type same as argument.
Exp	Returns **e** raised to the specified power.
Log10	Returns the base 10 logarithm of a specified number.
Pow	Returns a specified number raised to the specified power.
Sqrt	Returns the square root of a specified number.

Commonly Used Math Methods

- Common Numeric Methods in Math Class:

Method	Meaning
Ceiling	Returns the smallest whole number greater than or equal to the specified number.
Floor	Returns the largest whole number less than or equal to the specified number.
Round	Overloaded. Returns the number nearest the specified value.
Sign	Overloaded. Returns a value indicating the sign of a number.

Commonly Used Math Methods

- Common Numeric Methods in Math Class:

Method	Meaning
Max	Overloaded. Returns the larger of two specified numbers. Returns the same data type as the arguments.
Min	Overloaded. Returns the smaller of two specified numbers. Returns the same data type as the arguments.
Log	Overloaded. Returns the logarithm of a specified number. Calling the method with ONE “double data type” argument returns Log to base **e**. Calling the method with TWO “double data type” argument returns Log of First arg. to base of Second arg.

Commonly Used Math Methods

- Selected Trigonometric Function Methods in Math Class:

Method	Meaning
Sin	Returns the sine of the specified angle.
Cos	Returns the cosine of the specified angle.
Tan	Returns the tangent of the specified angle.
Atan	Returns the angle whose tangent is the specified number.
Tanh	Returns the hyperbolic tangent of the specified angle.

Further study

- Check out the documentation for Math class at
  - https://docs.microsoft.com/en-us/dotnet/api/system.math?view=netframework-4.7.2
  - Or just search for "System.Math class"
- Find the methods that has been mentioned in the previous slides and check out how to use the method
- Write a program that uses some of them
  - Some of the exercises will require you to do that

Summary

- In C#, we can use variables to store data temporarily in our program memory
- Variables has a type and C# will check the validity of values assigned to the variables
- We can use expression so that we can perform calculation without storing the result into a variable
- Expression is typically written using operators or by calling methods from another class (our own class or class from a class library)

APPENDIX

Overflowing

- Example:

```
byte number = 255;
number = number + 1; //0

Checked
{
        byte number = 255;
        number = number + 1;
}
```