Day 1:

**Types of C# programs:**

1. Console Application - Console Applications run from the command line.

2. Window form Applications - Take advantage for graphical user interface (GUI.)

3. Web Services - Web Services are routines that can be called across the web.

4. Web form/ASP.NET applications - ASP.NET applications are executed on a

Web server and generate dynamic web pages.

**/t:winexe 'filename.cs' :**

- Including /t:winexe on the command line tells the C# complier to target this application as a Windows executable.

**C# is Object Oriented:**

1. Encapsulation

- the placing of functionality into a single package

2. Inheritance

- structured way of extending existing code and functionality into new programs

3. Polymorphism

- is the capability of adapting to what needs to be done.  
­

------------------------------------------------------------------------------------------------------------------------------------------

--------------------------------------------------------------------------

Day 2:

One Line Comments:

ex: // comment text

Multiline Comments:

ex /\* this is a comment \*/

- To start use: /\*

- To end use: \*/

ex: /\* This is

a comment.

\*/

Documentation Comments:

- Identified with three slashes (///)

- Enable you to create external documentation automatically

To get XML documentation:

add the /doc parameter when you compile at command line:

ex: csc /doc:xmlfile Xmlapp.cs

Key Parts of C# language:

- Whitespace

. The complier almost always ignores whitespace.

ex: int radius = 4;

ex: int radius = 4 ;

ex: int

raidus

=

4

;

Are all the same thing.

. The exception to this is text within quotation marks.

- c# keywords

. abstract as base bool break

. byte case catch char checked

. class const continue decimal default

. delegate do double else enum

. event explicit extern false finally

. fixed float for foreach goto

. if implicit in int interface

. internal is lock long namespace

. new null object operator out

. override params private protected public

. readonly ref return sbyte sealed

. short sizeof stackalloc static string

. struct switch this throw true

. try typeof uint ulong unchecked

. unsafe ushort using virtual void

- literals:

. Literals are straightforward hard-coded values.

ex: 4 and 3.14159 are both literals

- identifiers

. Words taht are not keywords or literals.

Expressions:

- are like phrases.

ex: PI = 3.14159

ex: PI \* radius \* radius

Statements:

- are like sentences.

- Empty statements: semicolon on line with nothing else

VARIABLES:

Naming Variables:

- The name can contain letters, digits, and underscores.

- The first character of name must be a latter, an underscore can also be use.

- Case matters.

- C# keywords cannot be used as variable names.

Using Variables:

Declaring a Variable:

typename varname;

typename = the variable type

varname = the name of the variable

.To delcare a variable that can hold an integer:

int my\_number;

.You can also declare multiple variables on the same line with commas:

ex: int count, number, start;

Assigning Values to Variables:

varname = value;

ex: my\_variable = 5;

. You can declare after variable has been declared or at the same time.

ex: int my\_variable = 5;

. Variables value can be changed:

ex: my\_variable = 1010;

C# DATA TYPES:

\* you must include extra info for complier sometimes, this info is referred to

\* as a 'flag'. you need to add /unsafe flag.

\* ex: csc /unsafe blah.cs

Numeric Variable Types:

. Integral

- Integers(int and uint)

. stores an signed number represented by 4 or 32 bytes.

. any 'signed' number between (+ or - 2,147,483,687)

. if you want an int to be higher you can make it unsigned.

. You can use uint to make number unsigned.

. You can not use decimals.

- Shorts(short and ushort)

. It is only 2 bytes.

. use short for pos and neg numbers.

. use short if all pos.

. values for short are (+ or - 32,768)

. values for ushort are(0 to 65,535)

- Longs(long and ulong)

. More than int and uint.

. long can store (+- 9,223,372,036, 854, 775, 808)

. ulong can store (0 to 18,446,744,073,709,551,615)

- Bytes(byte and sbyte)

. When needs are really small, use byte.

. sbyte can store (-128 to 127)

. byte can store ( 0 to 255)

- Character(char)

. Letters or extended characters.

. also $, %, \*

. all chars are stored as numbers.

- Extended characters:

. \b - backspace

. \n - newline

. \t - horizontal tab

. \\ - backslash

. \' - single quote

. \" - double quote

. Floating Point

- float

. a data type for storing numbers with decimal places.

ex: 1.23, 3.14159

. can store numbers in 4bytes of memory

- double

. can store numbers in 8 bytes of memory.

- you can get rounding errors.

ex: subtracting 9.90 from 10 could result in .09999999999999645 vs .10.

. Decimal:

- can also store special decimal numbers.

- stores with greater precision.

. Boolean

- When you need to no if something is on or off, true or false, yes or no.

- set to 0 or 1.

- bool is either true or false.

C# VS .NET DATE TYPES

sbyte = System.SByte

byte = System.Byte

short = System.Int16

ushort = System.UInt16

int = System.Int32

uint = System.UInt32

long = System.Int64

ulong = System.UInt64

char = System.Char

float = System.Float

double = System.Double

bool = System.Boolean

decimal = System.Decimal

LITERALS VS VARIABLES

Numerical literals

- by default a numerical literal is either a integer or a double.

ex: nbr = 100; - this is int.

ex: nbr = 99.9; = this is double.

Understanding the integer literal defaults

- if you want to specify that data type of the literal,

- you can use a suffix.

ex: if you want 10 as a literal long value use:

. 10L; - make it unsigned: 10UL

Understanding floating-point literal defaults

- to declare a literal that is of type float, include f or F.

ex: to assign 4.4 to variable my float:

. my\_float = 4.4f;

- to declare a literal of typfor decimal us m or M.

ex: to assign 1.32 to my\_decimal:

. my\_decimal = 1.32m;

Understanding String Literals

-A string literal is any set of characters between double quotes.

ex: "Hello, World!"

ex: "123456789"

CREATING CONSTANTS:

- sometimes you wnat to put a value in a variable and freeze it.

- if you create PI and set it to 3.14159 and want it to stay like that.

- to declare variable to hold constant value, use 'const':

ex: const float PI = 3.14159;

------------------------------------------------------------------------------------------------------------------------------------

Day 3

**Displaying Basic Information**:

* System.Console.WriteLine()

- Writes info and then goes to new line.

* System.Console.Write()

- Does not go to new line when info is written.

**Manipulating Variable Values with Operators:**

* Operators can be broken into a few categories:
* The basic assignment operator
* Mathematical/arithmetic operators
* Relational operators
* The conditional operator
* Other operators (type, size)
* Three types of operator structures exist:
* Unary

- Operators that impact a single variable.

Ex: to have a -1, you type this: -1

Ex: If you have a variable x, you can change it to negative like this: -x

- The format of a unary variable is one of the following:

[operator][variable] or [variable][operator]

* Binary

- Work with two variables; the format is:

[variable1][operator][variable2]

Ex: 5 + 4

Ex: 3 – 2

Ex: 100.4 – 92348.67

* Ternary

- Ternary works with three variables.

- C# only has one true ternary operator; the conditional operator.

* Understanding Punctuators
* Semicolon:

- The primary use is to end each C# statement.

- Can also be used to control program flow

* Comma:

- The comma is used to stack multiple commands on the same line.

- The most common time to use the comma is when declaring multiple variables of the same type.

* Parentheses ()

- Parentheses are used in multiple places.

- Parentheses are used with functions.

* Braces {}

- Braces are used to group pieces of code.

* Moving Values with the Assignment Operator
* The basic operator: an equal sign ( = ) :

- The basic operator is used to assign values

Ex: To assign 142 to x -> x = 142;

Ex: x = y = 123;

- You cannot do operations on the left side:

Ex: 1 + x = y;

* Working with Mathematical/Arithmetic Operators
* Adding and Subtracting

- ‘+’ and ‘-‘

Ex: NewVal1 = Value1 + Value2;

Ex: NewVal2 = Value1 – Value2;

* Doing Multiplicative Operators

- ‘\*’ and ‘/’ and ‘&’

Ex: NewVal1 = Value1 \* Value2; - Multiplication

Ex: NewVal1 = Value1 / Value2; - Division

Ex: Val = 4 % 3; - This is the modulus operator, it will give remainder.

* Working with the Compound Arithmetic Assignment Operators

|  |  |  |
| --- | --- | --- |
| Operator | Description | Noncompound Equivalent |
| += | X += 4 | X = X + 4 |
| -= | X -= 4 | X = X - 4 |
| \*= | X \*= 4 | X = X \* 4 |
| /= | X /=4 | X = X / 4 |
| %= | X %= 4 | X = X % 4 |

* Doing Unary Math
  + (++)

Ex: ++x; - adds 1 to x

* + (--)

Ex: --x; - subtracts 1 from x

* Using the if Statement
  + The standard format of the if statement is:

if( val1 [operator] val2)

Statement(s);

* Conditional Logical Operators
* The Conditional AND operator

- &&

- If you want to verify that all conditions are met.

Ex: if (sex == female && age >=21)

- You can place more than two relationship within a single if statement.

Ex: if (x<5 && y < 10 && z > 10)

* The Conditional OR operator

- ||

- If only one of the conditions have to be true.

Ex: if ( day == Sunday || day == Saturday)

* Understanding Logical Bitwise Operators
* Type Operators

- typeof

- is

- as

- sizeof

* Shortcutting with the conditional operator
* C# has one ternary operator: the conditional operator:

*Condition ? if\_true\_statement : if\_false\_statement;*

* Understanding Operator Precedence

|  |  |  |
| --- | --- | --- |
| Level | Operator Types | Operators |
| 1 | Primary Operators | () . [] x++ x-- new typeof sizeof |
| 2 | Unary | + - ! - ++x - - x |
| 3 | Multiplicative | \* / % |
| 4 | Additive | + - |
| 5 | Shift | << >> |
| 6 | Relational | < > <= >= is |
| 7 | Equality | == != |
| 8 | Logical AND | & |
| 9 | Logical XOR | ^ |
| 10 | Logical OR | | |
| 11 | Conditional AND | && |
| 12 | Conditional OR | || |
| 13 | Conditional | ?: |
| 14 | Assignment | = \*= /= %= += .= <<= >>= &= ^= |= |

* Converting Data Types
* Implicit- Happen automatically without error
* Explicit – Conversions of data that are forced

- cast: the forcing of a data value to another data type:

ToVariable = (datatype) FromVariable;

Ex: int IntVariable = 0;

long LongVariable = 1234;

IntVariable = (int) LongVariable;

----------------------------------------------------------------------------------------------------------------------------------------

**Day Four: Controlling Your Program’s Flow**

* Controlling Program Flow:
  + Selection Statements

- Enable you to execute specific blocks of code based on the results of a condition.

Ex: if ( *condition* )

{

// If condition is true, do these lines

}

else

{

// If condition is false, do these lines

}

// code after if... statement

* + - Nesting and Stacking if Statements

Ex: if( gender == ‘m’ )

{

// it is a male

}

else

{

**if ( gender == ‘f’ )**

**{**

**// it is a female**

**}**

**else**

**{**

**//neither a male or a female**

**}**

}

* + - The Switch Statement

Ex: switch ( *value* )

{

case result\_1 :

// do stuff for result\_1

break;

case result\_2 :

// do stuff for result\_2

break;

...

case result\_n :

// do stuff for result\_x

break;

default:

// do stuff for default case

break;

}

* + - * + Executing a single solution for multiple cases

Ex: switch (roll)

{

case 1:

case 3:

case 5:

System.Console.WriteLine(“Roll is odd”);

break;

case 2:

case 4:

case 6:

System.Console.WriteLine(“Roll is even”);

break;

default:

System.Console.WriteLine(“Roll is not 1 through 6”);

break;

}

The same code is executed if the roll is 1, 3, or 5. Additionally, the same code is executed if the roll is 2, 4, or 6.

* + - * + Executing More than One case statement

Ex: switch (roll)

{

case 1:

goto case 5;

break;

case 2:

goto case 6;

break;

case 3:

goto case 5;

break;

case 4:

goto case 6;

break;

case 5:

System.Console.WriteLine(“Roll is odd”);

break;

case 6:

System.Console.WriteLine(“Roll is even”);

break;

default:

System.Console.WriteLine(“Roll is not 1 through 6”);

break;

}

* + Iterative Statements
    - while
      * Used to repeat a block of code as long as it is tru.

Ex: while ( *condition* )

{

Statement(s)

*}*

* + - Do
      * Often referred to as a do…while statement.

Ex: Do

{

Statement(s)

} while ( condition );

* + - For

Ex: for ( initializer; condition; incrementor )

{

Statement(s);

}

Ex: for ( ctr = 1; ctr < 10; ctr++ )

{

//do some stuff

}

1. Set a counter to value of 1.
2. Check to see if counter is less than 10. If the counter is not less than 10, go to end of the for statement.
3. Do some stuff
4. Add 1 to the counter
5. Go to step 2.
   * + - To use multiple expression, use the comma.
     + Foreach
       - Very similar to the for statement, however it can also loop through collections such as arrays.
   * goto
     + Used in three ways
       - goto case
       - goto default
       - goto label
         * A label statement is a command that marks a location.
         * label\_name:

-------------------------------------------------------------------------------------------------------------------------------

**Day Five:**

* Declaring Classes:
  + The format for declaring an object from a class:
    - *class\_name object\_identifier = new class\_name();*
      * class\_name is the name of the class
      * object\_identifier is the name of the object being declared
    - Ex: There is a class called ‘Point’, we can create object called ‘startingPoint’
      * *Point startingPoint = new Point();*
* The Members of a Class:
  + Data Members
    - Data Members include variables and constants.
  + Function Members
    - Function members are routines that perform and action.
  + Working with Data Members, aka Fields
    - Another name for a variable is a field.
    - Accessing Data Members
      * To access a data member, you use both the name of the object and the data member.
      * Ex: startingPoint.x
* Properties
  + Properties enable you to create object-oriented fields within your classes.

----------------------------------------------------------------------------------------------------------------

**Day Six**

* Methods:
  + A method is a named piece of code that is placed in a reusable format.
  + Format of method:

Ex. Method\_header

{

Method\_body

}

* + The Method Header:
    - The access that programs have to the method
    - The return data type of the method
    - Any values that are being sent to the method
    - The name of the method
  + Returning Data From a Method:
    - To return a value, use the return keyword.
      * The value or variable must be of same type as header.
    - If a method does not need to return value, use the void keyword.
* Access Attributes for Parameters:
  + There are three types of access attributes:
    - Value
      * Refers to when a copy is made of the data being sent to the method.
    - Reference
      * A reference is a variable that has access to the original variable.
    - Out
      * You can add parameters to your method header specifically for returning values by adding the out keyword.
* Types of Class Methods
  + Property accessor methods
    - set
    - get
  + Constructors
    - Instance Constructors
      * A method that is automatically called whenever an object is instantiated.
      * Format:

Ex: modifiers classname()

{

// Constructor body

}

* + Destructors/finalizers
    - A destructor can perform some operations when an object is finished.
    - Format:

Ex: ~xyz()

{

// Destructor body

}

-------------------------------------------------------------------------------------------------------------------------------

Day 7:

* Structures
  + Structures are a value data type.
  + Declaring Structure Members

Ex: struct Point

{

public int x;

public int y;

}

* + Structure Constructors
    - You must include declarations with parameters.
* Enumerators
  + Enable you to create variables that contain a limited number of values.

Ex: modifiers enum enumName

{

enumMember1,

enumMember2,

}

* Changing the Default Value of Enumerators

1. You could put a filler value in the first position.
2. Explicitly set the value of your enumerator members.

* Changing the Underlying Type of an Enumerator
  + Enumerator default type is int.
  + To change the default type:

Modifiers enum enumname : typename { members}

* Using Arrays to Store Data
  + If you need to keep track of a number of items that are of the same data type, use arrays.
  + An array is a single data variable that can store multiple pieces of data that are the same data type.
  + Ex: datatype[] name;
  + Each item in an array is called an element which can be accessed by using an index.
  + Initializing Array Elements
    - decimal[] balances = new decimal[] {1000.00m, 2000.00m, 3000.00m};
* Multidimensional Arrays
  + To declare a two-dimensional array
    - Byte[,] scores = new byte[15, 30];
* foreach Statement
  + Can be used to simplify working with arrays.

Ex foreach( datatype varname in arrayName)

{

Statements;

}

---------------------------------------------------------------------------------------

Day Eight:

* Overloading Methods
  + Method Overloading is the process of creating multiple methods with the same name.