

C# Language Fundamentals

Fundamentals of .NET
X52.9243
Spring 2010
Feb. 16 –May 4(10 sessions)
Tuesday 6:30pm – 9:30pm
48 Cooper Square, room 202
Keith R. Harris (keith9820@hotmail.com)

Exercise 1

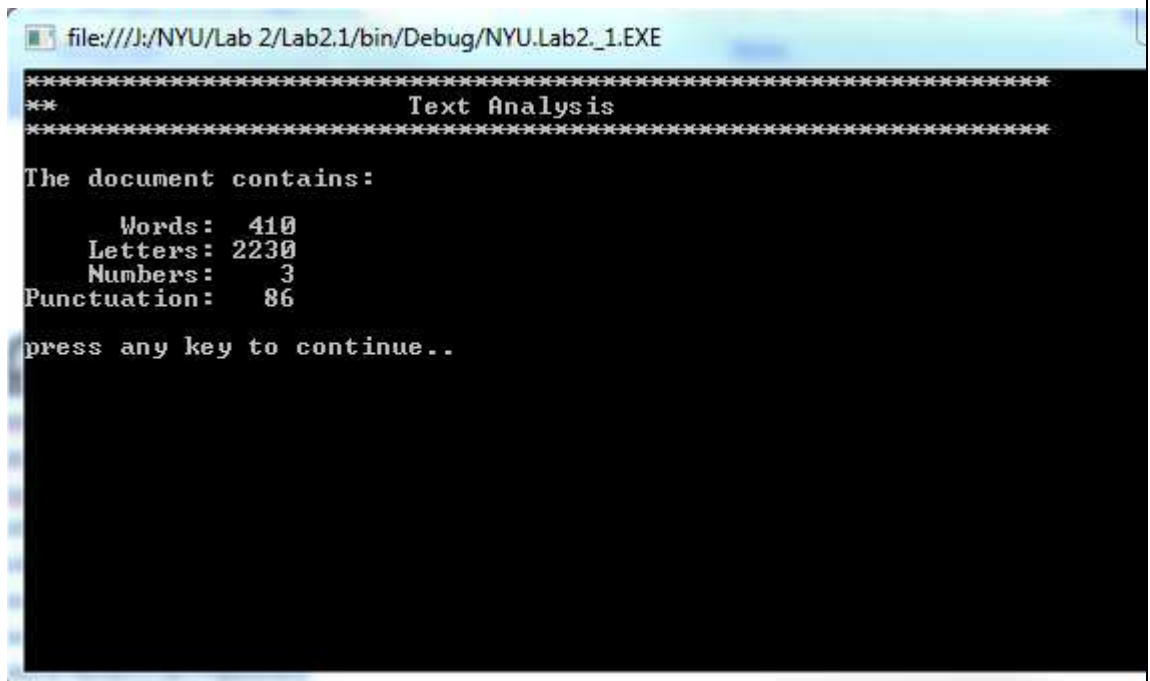
Create a string analysis application

Scenario

Create a console application to analyze the contents of a text file. You will print out the number of words, letters, digits, and punctuation marks contained within the file.

Tasks	Detailed Steps
1. Setting up the lab	<p>a. Open Visual Studio and create a new C# console application project named NYU.Lab2.1.</p> <p>Note: For convenience, you should create the project on your flash drive.</p> <p>b. Download the Lipsum.txt file from Blackboard and add it to your project. In the properties windows, set the "Copy to output directory" property to "Copy if Newer". This ensures that your text file will be in the same directory as the executable when built (bin\debug).</p>
2. Requirements	<p>a. You will build an application that reads the contents of the lipsum.txt file and counts the number of words, letters, numbers, and punctuation marks.</p> <p>b. After analysis, print out these metrics followed by the file contents in all caps.</p> <p>c. Create methods so that your code is modular.</p> <p>Note: The char type defines static methods that identify whether a character is a letter, digit, whitespace, or punctuation. i.e.:</p> <pre>char c = 'a'; if (char.IsDigit(c)) Console.WriteLine ("digit"); If (char.IsLetter(c)) Console.WriteLine ("letter");</pre>

3. Output



```
file:///J:/NYU/Lab 2/Lab2.1/bin/Debug/NYU.Lab2_1.EXE
***** Text Analysis *****
The document contains:
    Words: 410
    Letters: 2230
    Numbers: 3
    Punctuation: 86
press any key to continue..
```

(fig. 1)

Note: You can read a file using the ReadAllText method of System.IO.File.

4. Building

- a. Open the Program.cs source file. See listing 1 for a scaffold of the application you will write. Replace the //TODO labels with your own code. If you are uncomfortable using solely the scaffold application listing, the details of the program are listed in subsequent steps.
- b. Within the Program class, You will have 5 fields to hold the total number of each character type you want to count (letter, digit, etc...) and 4 parameterless methods which return void (including the Main method which already exists):
 - i. Main
 - ii. Pause
 - iii. ProcessTextFile
 - iv. WriteAnalysisReport

See Figure 2 for an overview of how your application should look.

Figure 2.

```
using System;

namespace NYU.Lab2._1
{
    class Program
    {
        Declare fields

        static void Main()...

        static void Pause()...

        static void ProcessTextFile()...

        static void WriteAnalysisReport()...
    }
}
```

- c. In the Main method you will do the following tasks:
 - i. Call the ProcessTextFile method.
 - ii. Call the WriteTextAnalysis method.
 - iii. Call the Pause method.
- d. Build out the ProcessTextFile method. The ProcessTextFile method will:
 - i. Read in all the text from the lipsum.txt file as a string.
 - ii. Loop through each character in the string.
 - iii. Increment the correct counter based on the type of character found.
- e. Build out the WriteAnalysisReport method. The WriteAnalysisReport method will:
 - i. Use the Console.WriteLine method to print out the counts
- f. The Pause method will:
 - i. Write out the text "press any key to continue".
 - ii. Wait for user input by calling Console.ReadKey().

Note: The purpose of the Pause method is to keep the console window open when it is run from the windows environment. This is not necessary when running a console application from the DOS command line.

Listing 1 - contents of program.cs

```
using System;

namespace NYU.Lab2._1
{
    class Program
    {
        #region Declare fields
        static int _numwords = 0;
        static int _numspaces = 0;
        static int _numpuncs = 0;
        //TODO: declare fields for the cont of numbers

        //TODO: declare fields for the cont of letters

        #endregion

        static void Main()
        {
            ProcessTextFile();

            WriteAnalysisReport();

            Pause();
        }

        static void Pause()
        {
            Console.Write("\npress any key to continue..");
            Console.ReadKey();
        }

        static void ProcessTextFile()
        {
            //TODO: declare a variable named filecontents of type string
            //          to hold the contents of lipsum.txt

            // NOTE: The System.IO namespace defines a class type named File
            // this type defines a static method named "ReadAllText" which
            // accepts the file name (as a string) and returns the contents of the
            // file as a string.

            //TODO: read the contents of lipsum.txt into the variable filecontents

            foreach (char c in filecontents)
            {
                if (char.IsPunctuation(c))
                {
                    _numpuncs++;
                }

                //TODO: If character is a number, increment the correct counter

                //TODO: If character is a letter, increment the correct counter
            }
        }
    }
}
```

```

        if (char.IsWhiteSpace(c))
        {
            _numWords++; //whitespace delimites words
            _numspaces++;
        }
    }

static void WriteAnalysisReport()
{
    Console.WriteLine("*****");
    Console.WriteLine("**\t\t\tText Analysis");
    Console.WriteLine("*****");

    Console.WriteLine("\nThe document contains:\n");

    Console.WriteLine("{0,11}: {1,4}", "Words", _numWords);

    //TODO: write out the cont of numbers

    //TODO: write out the cont of letters

    Console.WriteLine("{0,11}: {1,4}", "Punctuation", _numpuncs);
}
}
}

```

Exercise 2

.NET Collections

Scenario

Create a console application that deals with collections

Tasks	Detailed Steps
1. Overview	<p>The .NET CTS only provides one collection type – the array. You use an array when you want to store a fixed set of values of the same type.</p> <p>In the System.Collections namespace, the ArrayList class is used to define a variable length collection of objects. Use the ToArray method to return an array of types.</p> <pre>ArrayList list = new ArrayList(); list.Add("Hello"); list.Add("World");</pre> <p>Other collection types in this namespace include dictionary, stack, queue, sortedlist, and hashtable.</p> <p>The System.Collections.Generic namespace contains all the collection types of System.Collections but the collected values can be declared of specific types (not just object).</p> <pre>List<string> list = new List<string>(); list.Add("Hello"); list.Add("World");</pre>
2. Setting up the lab	<p>a. Create a new console application named NYU.Lab2.2.</p> <p>Note: For convenience, you should create the project on your flash drive.</p>
3. Build I	<p>a. Create an application that accepts 5 numbers from the user. Store these numbers into an array of type int[].</p> <p>b. Print out the number of even inputs as well as the number of odd inputs.</p> <p>c. What are the limitations of using an array? How would you accept a variable number of inputs?</p>

<p>4. Build II</p>	<ul style="list-style-type: none"> a. Modify the application above to use an ArrayList collection instead of int[]. b. Notice that the ArrayList object allows you to collect a variable number of inputs. c. Notice that the ArrayList objects only stores members of type object. This is not a problem since all .NET types ultimately derive from the object class but it does not provide type safety. What would happen if you stored numbers along with names?
<p>5. Build III</p>	<ul style="list-style-type: none"> a. Modify your application above to use the generic List<int> object instead of ArrayList. b. Notice that this object also allows for variable-length input. c. Notice that unlike that ArrayList object, a generic list enforces type safety. Only an integer value can be placed into a list declared as List<int>.

Note: Unless you are maintaining a collection of object types, you should always prefer to use .NET collections defined in the System.Collections.Generic namespace such as List<string> instead of ArrayList .

Exercise 3

Creating a Console Game

Scenario

Create a console application to play the card game “WAR”.

Tasks	Detailed Steps
1. Setting up the lab	<p>a. Create a new console application named NYU.Lab2.3.</p> <p>Note: For convenience, you should create the project on your flash drive.</p>
2. Overview	<p>a. Create an application to play the classic card game War. In War, two players each draw a card from a deck that is placed face-down between them. Each player turns over his card to reveal the face and the player with the highest card wins the hand. In the case of a tie, each player turns over a card from the deck until one player has a card whose face value is greater than that of his opponent. Play continues in this way until the deck has been exhausted. The winner of War is the player who has accumulated the most cards at the end of play.</p>
3. Build	<p>a. Open the Program.cs file and declare 6 static fields of the Program class:</p> <pre>static Random rand = new Random(); static int numGames, numBattles, numWins, numLosses, numDraws = 0;</pre> <p>Note: Declare a field of type Random to generate random numbers. You will use the GetNext method to retrieve the next value.</p> <p>b. Create an enum to represent the outcome of a battle. Add a new class file to your project named BattleStatus.cs. In this source file you will create an enum named BattleStatus that has 3 values: Win, Lose, Draw.</p> <p>c. In the Main method you will:</p> <ol style="list-style-type: none"> Call a method to print the header banner. Continually prompt the user to enter a command and carry out that command. Valid commands are: <ol style="list-style-type: none"> Play – draw and play a card, and show the result of play. Score – print the score board. Quit – end the game. The scoreboard is shown before exiting.

- d.** In the `ProcessCommandLine` method you will:
 - i. Print the command line options.
 - ii. If the user enters "p" or "play", call the `WageWar()` method.
 - iii. If the user enters "s" or "score", call the `PrintScore()` method.
 - iv. If the user enters "q" or "quit", call the `PrintScore()` method and exit.
 - v. If the user enters an invalid command, print "*** Invalid command ***".
- e.** In the `WageWar` method you will:
 - i. Increment the number of games.
 - ii. Print the game number.
 - iii. Do battle by calling the `DoBattle()` method. This method will return a value that indicates the outcome of the battle.
 - iv. While the status == `BattleStatus.Draw`, print "Tie! TO WAR!" and continue doing battle.
 - v. If the status == `BattleStatus.Win`, print "You Win".
 - vi. If the status == `BattleStatus.Lose`, print "You Lose".
- f.** In the `DoBattle` method you will:
 - i. Increment the `numBattles` counter.
 - ii. Declare 2 fields to hold the value of each player's card
 - iii. Generate a random card for each player. Since there are only 13 acceptable values (2-10, J, Q, K, A), you will need to provide min/max values to the `Random` field's `Next` method.
 - iv. Print out the card for each player. If the random value is an 11, 12, 13, or 14 then print Jack, Queen, King, Ace respectively (create a helper method for this named `DisplayCard`).
 - v. If player1's card is higher than player2's card, increment the number of wins and return `BattleStatus.Win`.
 - vi. If player1's card is lower than player2's card, increment the number of losses and return `BattleStatus.Lose`.
 - vii. If Player1's card is the same value as Player2's card, increment the number of draws and return `BattleStatus.Draw`.
- g.** In the `DisplayCard` method you will:
 - i. Accept an integer value as input and return a string that represents the value.
 - ii. Evaluate the value and return:
 - a) Return "Ace" if the value is 14.
 - b) Return "King" if the value is 13.
 - c) Return "Queen" if the value is 12.
 - d) Return "Jack" if the value is 11.
 - e) Otherwise, just return the given value.
- h.** In the `PrintScore` method you will:
 - i. Accept a boolean value to control whether an exit should occur.
 - ii. Print the number of games, battles, wins, losses, and draws.

iii. If the input value is true then exit the game.

4. Output

```
*****
**                Welcome to WAR                **
*****
Command <[P]lay, [S]coreboard, [Q]uit>:
```

(fig. 1 – opening screen)

```
*****
**                Welcome to WAR                **
*****
Command <[P]lay, [S]coreboard, [Q]uit>:

Game# 1:
=====
>You draw a King
>Dealer draws a 6
You WIN!
-----
Command <[P]lay, [S]coreboard, [Q]uit>:

Game# 2:
=====
>You draw a 9
>Dealer draws a King
You LOSE!
-----
Command <[P]lay, [S]coreboard, [Q]uit>:
```

(fig. 2 – play)

```
*****
**                SCOREBOARD                **
*****

Games:    2
Battles:  2
Wins:     1
Losses:   1
Draw:     0

<press the enter key to return>
```

(fig. 3 – score)

```
*****
**                SCOREBOARD                **
*****

Games:    2
Battles:  2
Wins:     1
Losses:   1
Draw:     0

<Game Over.  Press the enter key to exit.>
```

(fig. 4 – quit)