C# Language Fundamentals

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	**a.** Open Visual Studio and create a new C# console application project named NYU.Lab2.1. **Note:** For convenience, you should create the project on your flash drive. **b.** Add a text file to your project. Name the file lipsum.txt. **c.** Open your browser and go to http://www.lipsum.com. Generate 2 paragraphs of Lorem Ipsum text, then copy and paste it into the lipsum.txt file.
2. Requirements	**a.** You will build an application that reads the contents of the lipsum.txt file and counts the number of words, letters, digits, and punctuation marks. **b.** After analysis, print out these metrics followed by the file contents in all caps. **c.** Create helper methods so that your code is modular. **d.** Be sure you include error handling to protect against illegal user input. **Note:** Use methods of the char class to determine if a character is a letter, digit, whitespace, or punctuation.

3. Output

```
**************************************************
**              Text Analysis                   **
**************************************************

             Number of Words:    231
           Number of Letters:    1,301
            Number of Digits:    No Digits
Number of Punctuation Marks:    76

File Contents:
--------------
LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT. NAM MI MAGNA, POSUERE
QUIS, DICTUM ID, TINCIDUNT NEC, PEDE. AENEAN ID ANTE. PELLENTESQUE PURUS QUAM, C
ONDIMENTUM VITAE, ELEMENTUM ET, ADIPISCING AT, AUGUE. QUISQUE VITAE TURPIS NON T
ORTOR PORTTITOR SOLLICITUDIN. VIVAMUS VOLUTPAT. MORBI DIAM DUI, PORTTITOR AC, TI
NCIDUNT QUIS, ULTRICES NEC, NUNC. PRAESENT SEM ODIO, ULTRICES ET, POSUERE VITAE,
 PHARETRA A, NISL. VIVAMUS FEUGIAT EUISMOD LACUS. NULLA CONSEQUAT MAGNA VITAE NI
SL. MAECENAS COMMODO LIBERO ID ARCU. DUIS ACCUMSAN. MORBI TRISTIQUE TINCIDUNT UR
NA. PROIN EGESTAS NIBH A NISI. AENEAN ARCU ODIO, VENENATIS SED, TINCIDUNT EGET,
PELLENTESQUE VITAE, NIBH. UT LACINIA. PRAESENT IN JUSTO. VIVAMUS MATTIS, JUSTO N
EC CONDIMENTUM TEMPUS, DUI DUI EUISMOD NULLA, ET CONDIMENTUM ELIT NUNC UT SEM. V
ESTIBULUM TURPIS TELLUS, EUISMOD AT, CONSECTETUER A, SODALES AT, RISUS.

NULLA ACCUMSAN. LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT. MORBI
TRISTIQUE DAPIBUS URNA. SED EGET NIBH SCELERISQUE IPSUM RHONCUS ALIQUET. PHASELL
US MI NISL, RHONCUS IN, PORTTITOR QUIS, FRINGILLA IN, ARCU. LOREM IPSUM DOLOR SI
T AMET, CONSECTETUER ADIPISCING ELIT. AENEAN UT DUI VEL PURUS TRISTIQUE FEUGIAT.
 NULLA PURUS DIAM, MOLESTIE SED, PRETIUM AT, ALIQUAM NON, MAURIS. CRAS DICTUM EU
ISMOD FELIS. LOREM IPSUM DOLOR SIT AMET, CONSECTETUER ADIPISCING ELIT. SED ULTRI
CES ODIO LAOREET LIBERO. VIVAMUS CONSECTETUER TELLUS VEL PURUS. SED CONVALLIS HE
NDRERIT LEO. CLASS APTENT TACITI SOCIOSQU AD LITORA TORQUENT PER CONUBIA NOSTRA,
 PER INCEPTOS HIMENAEOS. PROIN AC ORCI. SUSPENDISSE POTENTI. CURABITUR IMPERDIET
 LAOREET URNA.

<press any key to end>
```

(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. In the Main method, call a method named ReportFileStats (you will build it in the next step). **Note:** Be sure to make a call to Console.ReadKey() before the end of the Main method otherwise the console window will close before you have a change to see the output. **b.** Create the ReportFileStats method that accepts no parameters and returns void. You can use the IDE's smart tag to auto generate this method if you wish. Since this is a console app, make sure the method is marked as static. **c.** The ReportFileStats method will do the following: i. Declare fields to hold the word, letter, digit, punctuation counts. ii. Split out the words. Each word is separated with a space character. Use the Split (instance) method of a string to do this. iii. Enumerate over each word. iv. Enumerate over each letter of a word and increment the appropriate counter based on whether the character is a letter, number, or punctuation mark. Use the static IsNumber(), IsLetter(), and IsPunctuation() methods of the char type. v. Call a method named ReportCounts which you will create. This method should accept all the counters in order to report on them. vi. Create the ReportCounts method. This method will create output similar to fig. 1. **Note:** Click on the lipsum.txt file and set the "Copy to Output Directory" property to "Copy if newer". This ensures that the txt file ends up in the same directory as the executable (bin\debug). **Note:** if you want to color your output. Use the ForegroundColor or BackgroundColor properties of the Console class. To change the color back, use Console.ResetColor(); **Note:** If you need to clear the screen, the Console class provides a Clear method.

```
class Program
{
    static void Main(string[] args)
    {
        ReportFileStats();

        Pause();
    }

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

    private static void ReportFileStats()
    {
        //todo:  read the file and store the text in a local field.
        string filecontents = System.IO.File.ReadAllText("lipsum.txt");

        int numletters = CountLetters(filecontents);
        int numnumbers = CountNumbers(filecontents);
        int numpuncs = CountPuncuation(filecontents);
        int numspaces = CountSpaces(filecontents);

        WriteAnalysis(filecontents, numletters, numnumbers, numpuncs, numspaces);
    }

    private static int CountLetters(string filecontents)
    {
        int sum = 0;
        foreach (char c in filecontents.ToCharArray())
        {
            if (char.IsLetter(c))
                sum++;
        }
        return sum;
    }
    private static int CountNumbers(string filecontents)
    {
        //todo: return the count of numbers
    }
    private static int CountPuncuation(string filecontents)
    {
        //todo:  return the count of punctuation marks
    }
    private static int CountSpaces(string filecontents)
    {
        //todo:  return the count of spaces
    }
```

```
    private static void WriteAnalysis(string filecontents, int numletters, int
numnumbers, int numpuncs, int numspaces)
    {
        //get the number of words
        string[] words = filecontents.Split(' ');

        //todo:  write output
    }
}
```

Exercise 2

.NET Collections

Scenario

Create a console application that deals with collections

Tasks	Detailed Steps

1. Overview	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.
	
	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.
	
	`ArrayList list = new ArrayList();`
	`list.Add("Hello");`
	`list.Add("World");`
	
	Other collection types in this namespace include dictionary, stack, queue, sortedlist, and hashtable.
	
	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).
	
	`List<string> list = new List<string>();`
	`list.Add("Hello");`
	`list.Add("World");`

2. Setting up the lab	**a.** Create a new console application named NYU.Lab2.2. **Note:** For convenience, you should create the project on your flash drive.
3. Build I	**a.** Create an applicaton that accepts 5 numbers from the user. Store these numbers into an array of type int[]. **b.** Print out the number of even inputs as well as the number of odd inputs. **c.** What are the limitations of using an array? How would you accept a variable number of inputs?

4. Build II **5.** Build III	**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? **a.** Modify your application above to use the generic List<int> object instead of ArrayList. **b.** Notice that this object also allows for variable-lenth 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>.

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	**a.** Create a new console application named NYU.Lab2.3. **Note:** For convenience, you should create the project on your flash drive.
2. Overview	**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 exhaused. The winner of War is the player who has accumulated the most cards at the end of play.

3. Build	**a.** Open the Program.cs file and declare 6 static fields of the Program class:
	`static Random rand = new Random();`
	`static int numGames, numBattles, numWins, numLosses, numDraws = 0;`
	
	Note: Declare a field of type Random to generate random numbers. You will use the GetNext method to retrieve the next value.
	
	b. Create an enum to represent the outcome of a battle. Add a new class file to your project named BattleStatus.cs. In this souce file you will create an enum named BattleStatus that has 3 values: Win, Lose, Draw.
	
	c. In the Main method you will:
	i. Call a method to print the header banner.
	ii. Continually prompt the user to enter a command and carry out that command. Valid commands are:
	a) Play – draw and play a card, and show the result of play.
	b) Score – print the score board.
	c) Quit – end the game. The scoreboard is shown before exiting.

d. In the PrecessCommandLine 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 mehod 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:

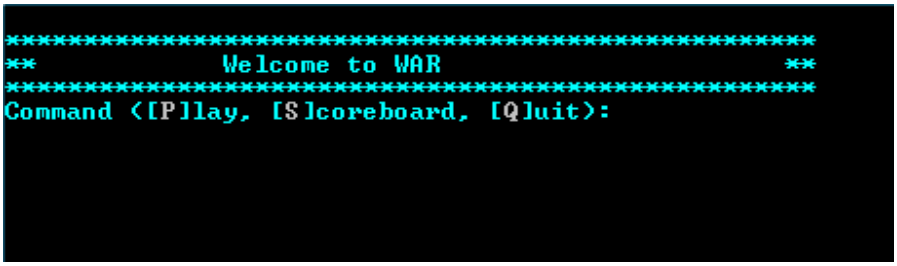
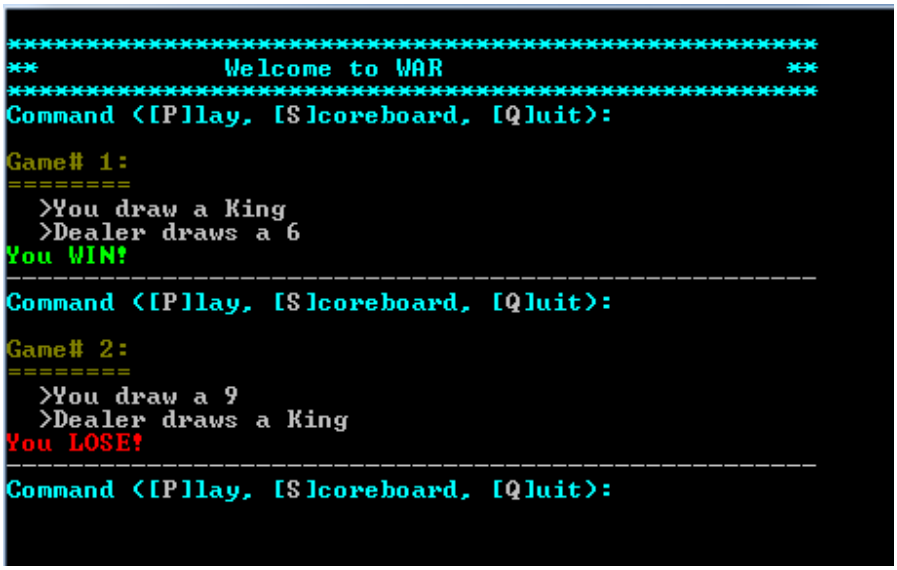
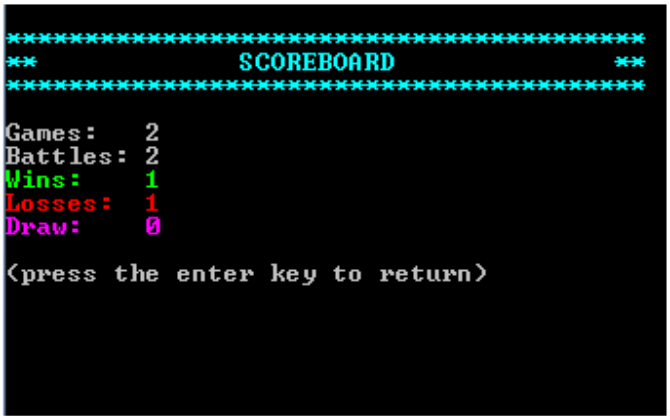
i. Increment the numBattles counter.
ii. Declare 2 fieds to hold the value of each players card
iii. Generate a random card for each player. Since there are only 13 acceptable values (2-10, J, Q, K, A), you will need to providemin/max values to the Random field’s Next method.
iv. Print out the card for each player. If the random value is an 11, 12, 13, or 14 then print Jack, Queen, King, Ace respectively (create a helpermethod 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 an 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	 (fig. 1 – opening screen)  (fig. 2 – play)  (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)