Assignment 2: Ada Programming (25%)

Choose **one** of the following two topics.

1. WORD SCRAMBLER

Read the following paragraph as quickly as you can and see if you encounter any difficulties.

Aoccdrnig to rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer is at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae we do not raed ervey lteter by itslef but the wrod as a wlohe.

This was published as an example of a principle of human reading comprehension. If you keep the first letter and the last letter of a word in their correct positions, then scramble the letters in between, the word is still quite readable in the context of an accompanying paragraph.

We are able to read the passage because our brains process all of the letters in a word at once. Maybe. There is not conclusive evidence to pinpoint any exact process. Obviously if a word is too long, it may be so scrambled that it becomes unreadable. But most letter transpositions preserve the sound of the word. For example letter - ltteer, leettr, lteter, letetr.

TASK

Design and implement an Ada program (called **`wordscram.adb`**) that will reproduce this behavior on a file of human readable English text. The text could contain anywhere from 100 to 1000 words.

The algorithm synopsis is:

1. Open a file for reading. Prompt the user for the file name. Perform error checking to make sure the file exists.
2. Read that file, a line at a time.
3. Extract all the words in a line
4. Process each word such that:
   4.1.Each word maintains its first and last letter in the correct position
   4.2. For every word (not number) larger than three letters long, the middle letters (between the first and last letter) should be randomly scrambled (re-ordered).
   4.3. Capitalization and punctuation must be preserved.
5. Output the revised text to the screen.

Here is an example of the process:

English has a length of 7. The first and last letters remain where they are: English

E	n	g	l	i	s	**h**
1	2	3	4	5	6	7

To scramble this word, the first letter to be processed is selected, in this case n. A random number between 2 and 6 is derived. Let's say it is 5. The scrambled word now looks like this:

E				n		**h**

The next letter, g, is processed in the same way. Careful because the allowable random spots are between 2 and 6, but not 5, as it has already been used. Let's say the number chosen is 2. The word now looks like this:

E	g			n		**h**

And so on... until the whole word is processed.

E	g	i	s	n	l	**h**

Create the following functions (apart from the main function) in your program:

`getFilename()`	This subroutine prompts the user for a filename, checks the existence of the file, and returns that filename to the `main` function. If no file exists with that name, the user is re-prompted.
`processText()`	This subroutine will take the filename as input and process (i.e. scramble) the words within the file, using the function `scrambleWord()`. This process involves determining the length of the word. For words of length less than three, nothing is done. For all other words, they are processed as outlined above (using the function `scrambleWord()`). See example provided. Invoked by the main program. This subroutine should also provide a count of the number of words processed. It uses `isWord()` to process only words, not numbers.

`scrambleWord()`	Take a word, and the words length as input, and scramble it. This involves deriving a random number between **a** and **b**. The number returned should not already have been used. If a word has N letters, then there are N-2 numbers to randomize: 2 (a) to N-1 (b). Next time around there are N-3 numbers etc. For example, if N=7, then in the first round, a number is randomly selected from (2,3,4,5,6). If the number selected is 5, then in the next round, a number can be randomly selected from (2,3,4,6).
`randomInt()`	Generate a random integer between **a** and **b**. Used by `scrambleWord()`.
`isWord()`	Determine if a piece of text is a word, i.e. contains only alphabetic characters.

The main program will call `getFilename()` once. The main program will then call the function `processText()` which extracts words from each line of the file and processes each word, calling the function `wordScramble()`, which in turn calls `randomInt()`.

TESTING THE PROGRAM

Your program should take a piece of text like this:

```
Rebel spaceships, striking from a hidden base, have won their
first victory against the evil Galactic Empire. During the
battle, Rebel spies managed to steal secret plans to the Empires
ultimate weapon, the Death Star, an armored space station with
enough power to destroy an entire planet. Pursued by the Empires
sinister agents, Princess Leia races home aboard her starship
custodian of the stolen plans that can save her people and
restore freedom to the galaxy.
```

and turn it into something like this:

```
Rebel spphecaiss, stirnikg from a hdeidn base, hvae won their
fsirt vocrity aansgit the eivl Gitaaclc Erpime. Driung the
bltate, Rbeel sipes magenad to steal sercet plans to the Epirmes
ualtmtie wepaon, the Daeth Satr, an aroermd scpae satiton wtih
egnuoh poewr to dsorety an etnrie panlet. Puresud by the Eripems
ssniietr aengts, Pcrisens Leia rcaes home aarbod her sristahp,
coisdutan of the stoeln plnas taht can svae her ppeloe and
rersote fodeerm to the gaaxly.
```

2. TEXT ANALYZER

The Unix system utility **wc**, calculates basic statistics of words in files - the number of words, lines and characters. It could be more useful though, more like a text analyzer. When studying languages it is often important to analyze textural material in a statistical manner. Such statistical analysis can be performed in many ways. One example is computing averages, such as the average number of words in a sentence, or the average number of vowels per sentence.

TASK

Design and implement an ADA program (called **textyzer.adb**) to perform text analysis of a file of human readable English text. The text could contain anywhere from 100 to 1000 words. The analysis would involved calculating the following numerics:

- the number of sentences
- the number of lines of text
- the number of words
- the number of characters (all characters)
- the number of numbers, i.e. items like dates that only contain digits.
- the number of punctuation characters

It would also calculate the following:

- the "average number of words/sentence"
- the "average number of letters per word"
- a histogram representing the word length distribution

The algorithm synopsis is:

1. Open a file for reading. Prompt the user for the file name. Perform error checking to make sure the file exists.
2. Read that file, processing it to calculate the quantitative values described above.
3. Output the statistics to the screen, in an appropriate manner that is easy to read.

Create the following functions (apart from the main function) in your program:

`getFilename()`	This subroutine prompts the user for a filename, checks the existence of the file, and returns that filename to the `main` function. If no file exists with that name, the user is re-prompted.

`analyzeText()`	Take the filename as input and process the text within the file. The function extracts each "word" (a series of characters between two blanks and/or punctuation marks), and then calculates and prints to standard output a series of basic statistics, using functions described below. The output should be similar to that shown in the "Sample Output" below. Invoked by the main program.
`isWord()`	Determine if a piece of text is a word, i.e. contains only alphabetic characters.
`isNumber()`	Determine if a piece of text is a number, i.e. contains only numeric digits.
`printHist()`	Prints a histogram representing the word length distribution. It is easy to do this as a horizontal histogram, harder as a vertical histogram. The example output shows a horizontal histogram.

SAMPLE OUTPUT

```
Enter the name of the text file to process:
middleE.txt

T h e  T e x t
--------------

The Lord of the Rings is an epic fantasy saga by the
British author J. R. R. Tolkien, his most popular work
and a sequel to his popular fantasy novel, The Hobbit.
The Lord of the Rings was written during World War II
and originally published in three volumes in 1954 and 1955.

T e x t  S t a t i s t i c s
---------------------------
Number of characters =  272
Number of words =  51
Number of numbers =  2
Number of sentences =  5
Number of lines =  5
Number of punctuation = 7
Average letters per word =  5.33
Average words per sentence = 10.2

Histogram - Word Length Distribution
    1   ****
```

```
 2   *********
 3   ************
 4   ********
 5   *****
 6   ****
 7   ********
 8
 9   *
10   *
11
12
13
14
15
16
17
18
19
20
```

WRITE-UP INFORMATION
(FOR EITHER CHOICE)

COMPILING

Please do not include a Makefile, and make sure your program compiles in the following manner:

```
> gnatmake -Wall filename.adb
```

REFLECTION REPORT

Describe your Ada program in one (1) page (single spaced) **reflection report**, explaining decisions you made in the process of designing the program. Consider the design document a synopsis of your programming process. One page should include a synopsis of the approach you took to design the program (e.g. it could be a numbered list showing each step in the process).

Some of the questions that should be answered include:

- Was Ada well suited to solving the problem?
- What particular structures made Ada a good choice?
- Benefits / limitations?

DELIVERABLES

The submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course).
  `wordscram.adb OR textyzer.adb`

- Both the code and the reflection report can be submitted as a ZIP, TAR, or GZIP file.

SKILLS

- Ada programming, program comprehension, problem solving