Western Washington University
Computer Science Department

CSCI 141 Computer Programming I
Fall 2011

Laboratory Exercise 6

Objectives

1. Practice in using fixed-point numbers.

2. Practice in the use of arrays.

3. Practice in the use of files.

Submitting Your Work

In this lab exercise you will write two Ada programs to perform cryptanalysis of data encrypted with the shift cipher. Save your program files (the .adb files) in the zipped tar file WnnnnnnnnLab6.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 6 Submission** item on the course web site. You must submit your programs by 4:00pm on Friday, November 18.

Cryptanalysis

The whole purpose of encrypting data is to prevent anyone from reading that data unless they know the encryption/decryption key. Cryptanalysis involves the use of various techniques in the analysis of encrypted data to determine the key and then decrypt the encrypted data. One fundamental technique in cryptanalysis is letter frequency analysis, in which we calculate the proportion (frequency) of alphabetic characters in the data for each letter of the alphabet.

For example, if we read through the characters of a file and find that in a total of 1,000 alphabetic characters, there are 87 occurrences of the letter 'A' (or 'a' – we don't distinguish between upper- and lower-case), the proportion (or frequency) of the letter A is 0.087. The total of the frequencies of all the letters in the alphabet in a file will be 1.000.

If we calculate the frequencies of alphabetic characters in a sample of the plaintext language (English, for example) and compare that to the frequencies in the encrypted data, we may be able to determine the encryption key.

The Shift Cipher

In the shift cipher each letter of the plaintext message is simply shifted along the alphabet by a fixed amount, determined by the encryption key. Therefore, if the key is G (the 7th

letter of the alphabet), each letter of plaintext is shifted by 6 places, so plaintext "hello" is encrypted as "NKRRU".

The simplicity of the shift cipher makes it highly vulnerable to cryptanalysis and therefore useless for keeping secrets from anyone other than pre-school children!

One technique of cryptanalysis is to compare the letter frequencies of a sample of plaintext with the letter frequencies of the encrypted message and determine how many places the plaintext frequencies have to be shifted along to make the frequencies match as closely as possible. For each possible shift value (0 through 25) we simply calculate the total absolute difference in letter proportions:

$$diff_{shift} = \sum_{i=1}^{26} |expected(i) - actual(j)|$$

where,

- expected(i) is the frequency of the i^{th} letter of the alphabet in a sample of plaintext,
- actual(j) is the frequency of the j^{th} letter of the alphabet in the encrypted message,
- j = (i + shift) mod 26

The value of *shift* for which $diff_{shift}$ is smallest indicates the key used to encrypt the plaintext message.

Program 1: Finding Letter Frequencies

This program is to read data from one or more data files, count the number of occurrences of each letter and output the proportional frequency of each letter.

1. The names of the files to be used for input are to be listed on the command line when you run your program. To get this information from the command line, you need the package Ada.Command_Line.
   - Function Ada.Command_Line.Argument_Count returns an integer indicating how many strings (separated by white space) are on the command line, following the name of your executable file. For example, if your program is called letter_count.adb and you run it with

     ```
     ./letter_count  Data1.txt  Data2.txt  Data3.txt
     ```

     Argument_Count will return the value 3.
   - Function Argument(I : Positive) returns the I^{th} command line parameter. From the example above, Argument(2) would return the string "Data2.txt".
2. If no file names are provided on the command line, your program must display an appropriate message and terminate. To do this, declare your own exception, raise

the exception if Argument_Count returns 0 and use an exception handler to display the message and terminate the program.

3. Your program must count only alphabetic characters and must not distinguish between lower- and upper-case.

4. Your program must also count the total number of alphabetic characters.

5. Use an array of natural numbers, indexed by the characters 'A' through 'Z' for your letter counts.

6. Your program must output, to standard output (the display), the proportion of the alphabetic characters for each letter of the alphabet. When the program is run, this output may be redirected to a file by using, for example:

```
./letter_count Data1.txt Data2.txt > Frequencies.txt
```

7. The output must have one number per line, starting with the proportion for 'A' (or 'a') on the first line and the proportion for 'Z' (or 'z') on the 26th line.

8. Each number output must be accurate to 3 decimal places. To do this you must declare a fixed-point data type with delta value 0.001 and instantiate and use the package fixed_io for that data type.

Program 2: Frequency Analysis

This program is to read the letter frequency data from two files produced by the first program – the first one from a sample of plaintext and the second one from the encrypted message – and, from that data, determine the key used for encryption, assuming that the shift cipher was used.

1. The names of the two letter frequency files are to be specified on the command line when you run the program. The first command line parameter is to be the letter frequency file for the plaintext sample and the second command line parameter is to be the letter frequency file for the encrypted message.

2. There must be exactly 2 command line parameters. If not, use exception handling to display a message and terminate the program.

3. The program is to use the same fixed-point type as in the first program and a fixed_io package to read the data from the files.

4. Use two arrays of the fixed-point data type, indexed by the characters 'A' through 'Z' – one array for the “normal” frequencies (from the plaintext sample) and the other array for the “actual” frequencies (from the encrypted message).

5. For each possible shift value in the range 0 through 25, calculate the total absolute difference in the frequencies, using the formula:

$$diff_{shift} = \sum_{i=1}^{26} |expected(i) - actual(j)|$$

6. Find which shift value gives the minimum absolute difference and from that value determine the encryption key.

7. The only output from this program is to be one line, identifying the encryption key, for example:

   ```
   The encryption key is Q
   ```

Submitting your program

1. Be sure that your program meets the coding standards listed below.

2. Include your program file in a zipped tar file, using the command

   ```
   tar –cf Wnnnnnnnn.tar file1.adb file2.adb
   ```

   (substitute the real file names for file1.adb and file2.adb)

   Now compress the tar file using the gzip program:

   ```
   gzip Wnnnnnnnn.tar
   ```

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.

2. Use comments at the start of the program to identify the purpose of the program, the author and the date written.

3. Use comments at the start of each procedure to describe the purpose of the procedure and the purpose of each parameter to the procedure.

4. Use comments at the start of each section of the program to explain what that part of the program does.

5. Use consistent indentation:

   - The declarations within a procedure must be indented from the Procedure and begin reserved words. The body of the procedure must be indented from the begin and end reserved words. Example of procedure indentation:

- procedure `DoStuff` is

```
     Count : integer;
     Valid : boolean;
begin
     Count := 0;
     Valid := false;
end DoStuff;
```

- The statements within the then-part, each elsif-part and the else-part of an if statement must be indented from the reserved words if, elsif, else and end.

```
if Count > 4 and not Valid then
     Result := 0;
     Valid := true:
elsif Count > 0 then
     result := 4;
else
     Valid := false;
end if;
```

- The statements within a loop must be indented from the loop and end loop.

```
loop
     Count := Count + 1;
     Get (Number);
     exit when Number < Count;
end loop;
```

- The exception handlers and statements within each exception handler must be indented.

```
begin
     ...  -- normal processing statements
exception
     when Exception1 =>
          Put_Line (“An error has occurred”);
          Total := 0;
     when others =>
          Put_Line (“Something weird happened”);
end;
```