Western Washington University
Computer Science Department

CSCI 145 Computer Programming and Linear Data Structures
Winter 2012

Laboratory Exercise 3

Objectives

1. Practice in the use of arrays.

2. Practice in the use of the command line parameters.

Submitting Your Work

Save your program file (the .adb file) in the zipped tar file WnnnnnnnnLab3.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 3 Submission** item on the course web site. You must submit your program by 3:00pm on Tuesday, Feb 14, 2012.

Your Task

Your task is to write an Ada program to perform cryptanalysis on data encrypted with a grid cipher. The grid cipher is a form of permutation cipher, in which none of the characters in the plain-text message is changed but the characters are shuffled into another order.

The Grid Cipher

Encryption with the grid cipher (in its simplest form) is performed by writing the plain-text characters into a rectangular grid, row by row, and then reading the characters from the grid column by column. The width of the grid is the encryption key.

For example, given the plain-text message "FRIENDSROMANSCOUNTRYMENLENDMEYOUREARS" and grid width 6, the plain-text characters are arranged in a grid as follows:

F	R	I	E	N	D
S	R	O	M	A	N
S	C	O	U	N	T
R	Y	M	E	N	L
E	N	D	M	E	Y
O	U	R	E	A	R

S					

The encrytped text, read from the grid column by column is
"FSSREOSRRCYNUIOOMDREMUEMENANNEADNTLYR".

In its more general form, the grid cipher transposes the columns. Instead of reading the columns from left to right, it reads them in some order determined by a more complicated encryption key. However, for this exercise, you can assume the simplest form of grid cipher, with the columns read from left to right.

Cryptanalysis of a Grid Cipher

Since the encryption key-space is small, the easiest method of cryptanalysis is *brute force*, in which we just try various grid widths to see if any of them result in plain-text that looks like english. For example, here is the output of a cryptanalysis program for the encrypted text shown above, with grid widths 2 through 8:

```
Grid width  2: 'FESMSUREEMOESNRARNCNYENAUDINOTOLMYDRR'
Grid width  3: 'FINSOASONRMNEDEORASEDRMNRUTCELYMYNERU'
Grid width  4: 'FYENSNMESUUARIEDEOMNOOETSMNLRDAYRRNRC'
Grid width  5: 'FRMMASCDEDSYRNNRNEATEUMNLOIUNYSOEERRO'
Grid width  6: 'FRIENDSROMANSCOUNTRYMENLENDMEYOUREARS'
Grid width  7: 'FSUDENNSRIRMNTSROEEELRCOMNAYEYMUADRON'
Grid width  8: 'FOYOMNETSSNMUAALSRUDENDYRRIRMNNRECOEE'
```

It is obvious from this output that the grid width was 6.

Program Requirements

1. The program must get the minimum and maximum grid sizes and the name of the input file from the command line. The command line arguments are:

   - The minimum grid width to be tried.
   - The maximum grid width to be tried.
   - The name of the input file.

   All of the command-line arguments are required. If too few or incorrect command line arguments are given, the program is to display an appropriate message and terminate without any further processing.

   Note: Each command line argument is a string. You can obtain the integer value of the string by using a form of the Get procedure from Ada.Integer_Text_IO.

```
procedure Get(From: in String;     -- the string argument
              Item: out Integer;   -- the integer value
              Last: out Positive); -- the string length
```

Procedure Get raises a Data_Error if the string cannot be translated into an integer.

2. The program is to output just one line for each grid width used for a cryptanalysis attempt, each in the form shown in the example above.

3. Your program is to use arrays declared as **exactly** the correct size for the input data. Note that you can use function Ada.Directories.Size to determine the number of characters in a file. The result from that function may include one or more endline character, depending on the operating system that you use. Those characters are not part of the encryption.

   Since the Ada.Directories package is new to Ada 2005 you may get a compiler warning. To get rid of this, add the option -gnat05 to the gnatmake command, for example:

   ```
   gnatmake -gnat05 gridreader.adb
   ```

4. You cannot assume that the number of characters in the input data is an exact multiple of the grid width. Therefore, as demonstrated in the example above, the last row of the grid might be incomplete.

To assist you in writing your program, the encryption program used to generate the example data is provided on the web site, along with several examples of plain-text messages and the corresponding encrypted messages.

Your program will be graded on correct functionality, as specified above, and conformance to the coding standards, described below.

What you must submit

You must submit the Ada source file (the .adb file). This file must be included in a zipped tar file.

Saving your files in a zipped tar file

Note: you will probably have just one Ada source file for this lab, so these instructions may seem like an overkill, but this is the method you are to use for later labs and assignments, so treat this as practice!

You only submit .adb and .ads files. First you need to bundle them up into a single tar file. The term "tar" is an abbreviation of "tape archive" and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

Use the command:

```
tar -cf WnnnnnnnnLab3.tar *.adb *.ads
```

(where Wnnnnnnnn is your W-number).

The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

This will include every file in the current directory whose name ends with ".adb" or ".ads".

If you now use the command ls you should now see the file WnnnnnnnnLab3.tar in your directory.

If you use the command

```
tar -tf WnnnnnnnnLab3.tar
```

(where Wnnnnnnnn is your W-number), it will list the files within the tar file.

Now compress the tar file using the gzip program:

```
gzip WnnnnnnnnLab3.tar
```

By using the ls command again, you should see the file WnnnnnnnnLab3.tar.gz in your directory. This is the file that you need to submit through the **Lab Exercise 3 Submission** link in the moodle web site.

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;
```