Western Washington University
Computer Science Department

CSCI 145 Computer Programming and Linear Data Structures
Winter 2012

Laboratory Exercise 1

Objectives

1. Revision in the use of the Ada programming language.
2. Practice in the use of the Linux operating system.

Submitting Your Work

Save your program as WnnnnnnnnLab1.adb (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 1 Submission** item on the course web site. You must submit your program by 3:00pm on Fri, January 20, 2011

Your Task

Your task is to write an Ada program to perform encryption and decryption using the Vigenère cipher. The program is use 2, 3 or 4 command line arguments. You can use functions Ada.Command_Line.Argument_Count and Ada.Command_Line.Argument to get the required information.

The command line arguments are:

1. The encryption key, a string of alphabetic characters of arbitrary length. The same key is used for both encryption and decryption. The key may contain either or both lower-case and upper-case alphabetic characters. This parameter is required for the program to run.

2. A single character specifying whether encryption or decryption is to be applied to the input. 'E' or 'e' signifies encryption; 'D' or 'd' specifies decryption. This parameter is required for the program to run.

3. (optional) The name of the file to be used for input to the encryption or decryption process. If this parameter is not specified, the program must read from standard input.

4. (optional and only used if the third command line parameter is also provided) The name of the file to be used for output from the encryption or decryption process. If this parameter is not specified, the program must write to standard output.

For example, if the name of your program file is vigenere.adb, you produce an executable file by using the command:

```
gnatmake vigenere.adb
```

This will result in an executable file called vigenere, which you can run with the command:

```
./vigenere VICTORY E plain.txt cipher.txt
```

In this example:

- The encryption key is VICTORY.
- The input will be encrypted (because of the 'E' on the command line).
- The input will be read from the file plain.txt.
- The encrypted output will be written to the file cipher.txt.

The command

```
./vigenere VICTORY E plain.txt
```

will run the program for encryption with the key VICTORY, reading its input from the file plain.txt and writing to standard output.

The command

```
./vigenere VICTORY E
```

will run the program for encryption with the key VICTORY, reading from standard input and writing to standard output.

Of course, the command line redirection symbols < and > can be used to redirect standard input from a file and redirect standard output to another file. So the command

```
./vigenere VICTORY E < plain.txt > cipher.txt
```

has only two command line arguments (VICTORY and E) but will run the program for encryption with the key VICTORY, reading its input from the file plain.txt and writing its output to the file cipher.txt.

Vigenère Cipher.

The Vigenère cipher is a polyalphabetic cipher, meaning that it uses more than one alphabet for the ciphertext. The number of alphabets used is determined by the length of the key. For a key with *m* letters, the cipher cycles through *m* alphabets, where each alphabet is determined by an alphabetic shift based on a letter of the key.

For example, if the key is "VICTORY", the cipher uses 7 alphabets, namely:

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
1	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
2	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H

3	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
4	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
5	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
6	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
7	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X

Plaintext is encrypted by substitution, using these alphabets in cycle. For example, the plaintext "this course is fun" is encrypted as "OPKLQFSMAGBGWSI" (omitting spaces).

Plaintext character	**Alphabet row used**	**Ciphertext chatacter**
t	1	O
h	2	P
i	3	K
s	4	L
c	5	Q
o	6	F
u	7	S
r	1	M
s	2	A
e	3	G
i	4	B
s	5	G
f	6	W
u	7	S

Plaintext character	Alphabet row used	Ciphertext chatacter
n	1	I

Requirements

Your program must do the following things:

1. Check that there are at least two command line arguments and otherwise terminate with a usage message, for example

```
Usage: vigenere key E|D [infile [outfile]]
```

1. Check that the first command line argument is all alphabetic and otherwise terminate with an appropriate message.

2. Check that the second command line argument is 'E', 'e', 'D', or 'd' and otherwise terminate with an appropriate message.

3. If there is a third command line argument, use that as the name of a file which is opened for input. You can use the procedure Ada.text_IO.Set_Input to redirect standard input to come from that file. If the file does not exist, terminate with an appropriate message.

4. If there is a fourth command line argument, use that as the name of a file which is opened for output. You can use the procedure Ada.Text_IO.Set_Output to redirect standard output to go to that file.

5. Any non-alphabetic characters that are input to the encryption or decryption process are to be ignored and should not appear in the output– simply skip to the next character.

6. Encrypted text must be all upper-case alphabetic characters.

7. Decrypted text must be all lower-case alphabetic characters.

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) for the Vigenère cipher program. 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 thses 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 Wnnnnnnnn.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 Wnnnnnnnn.tar in your directory.

If you use the command

```
tar -tf Wnnnnnnnn.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 Wnnnnnnnn.tar
```

By using the `ls` command again, you should see the file Wnnnnnnnn.tar.gz in your directory. This is the file that you need to submit through the **Lab Exercise 1 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;
```