Dalhousie University
CSCI 2132 — Software Development
Winter 2017
Assignment 5

Distributed Wednesday, March 1 2017.

Due 3:00PM, Wednesday, March 15 2017.

Instructions:

1. The difficulty rating of this assignment is *silver*. Please read the course web page for more information about assignment difficulty rating, late policy (no late assignments are accepted) and grace periods before you start.

2. Each question in this assignment requires you to create one or more regular files on bluenose. Use the exact names (case-sensitive) as specified by each question.

3. C programs will be marked for correctness, design/efficiency, and style/documentation. Please refer to the following web page for guidelines on style and documentation for short C programs (which applies to this assignment):

   `http://web.cs.dal.ca/~mhe/csci2132/assignments.htm`

   You will lose a lot of marks if you do not follow the guidelines on style and documentation.

4. Create a directory named `a5` that contains the following files (these are the files that assignment questions ask you to create): `a5q1.c` and `a5q2.c`. Submit this directory electronically using the command `submit`. The instructions of using `submit` can be found at:

   `http://web.cs.dal.ca/~mhe/csci2132/assignments.htm`

5. Do NOT submit hard copies of your work.

Questions:

1. [15 marks] This question asks you to implement Caesar cipher, one of the simplest and most widely known encryption techniques.

   The Caesar cipher is named after Julius Caesar, who used it to encode messages of military importance in ancient times. The idea is to replace any letter in the message by a letter three positions down the alphabet.

More precisely, to use it for a message written in English (Caesar, of course, would use it for Latin), we replace `a` by `d`, `b` by `e`, `c` by `f`, ..., `w` by `z`, `x` by `a`, `y` by `b`, and `z` by `c`. Hence this is essentially a cyclic shift (pay attention to the three letters `x`, `y` and `z`). We also do something similar to uppercase letters as well, replacing `A` by `D`, etc. For any character that is not an English letter, including spaces, digits and punctuations, we do not replace them and leave them in the encrypted message as they are.

For example, using this strategy, the message `To be, or not to be` would be encrypted as `Wr eh, ru qrw wr eh`

Your implementation reads a line of text and then either performs encryption or decryption, depending on the first character read. If the first character is `$`, then you are supposed to encrypt the rest of the line, and precede your output by the character `?`. If the first character is `?`, then you are supposed to decrypt the rest of the line, and precede your output by the character `$`.

For example, if the input to your program is

```
$To be, or not to be
```

Then your program should print

```
?Wr eh, ru qrw wr eh
```

If the input to your program is

```
?Wr eh, ru qrw wr eh
```

Then your program should print

```
$To be, or not to be
```

General Requirements: Use `a5q1.c` as the name of your C source code file.

We will use the following command on bluenose to compile your program:

```
gcc -std=c99 -o encry a5q1.c
```

Your program should NOT print anything to prompt for user input. It accepts user input that meets the syntax specified above. It then either encrypts or decrypts the message, and generates a single line of output as specified above, which is terminated by a newline character.

Therefore, when you run your program by entering `./encry`, the program will wait for your input. If you enter

```
$To be, or not to be
```

the program will output

```
?Wr eh, ru qrw wr eh
```

Error Handling: Your program should print an appropriate error message and terminate if the first character in the user input is not `$` or `?`

Testing: To test your program automatically, we will use it as a UNIX filter.

For the example above, we will make use of input redirection to test your program. We will first create a file whose content is the input line of message (terminated by a newline). Suppose that its content is:

```
$To be, or not to be
```

Suppose that the name of this file is `a5q1.in.0`. We will then test your program using the following command in the directory containing the executable program:

```
./encry < a5q1.in.0
```

The output should be:

```
?Wr eh, ru qrw wr eh
```

To help you make sure that the output format of your program is exactly what this questions asks for, several files are provided in the following folder on bluenose to show how exactly your program will be automatically tested:

```
/users/faculty/prof2132/public/a5test/
```

In this folder, open the file `a5q1test` to see the commands used to test the program with two test cases. The output files, generated using output redirection, are also given.

To check whether the output of your program on any test case is correct, redirect your output into a file, and use the UNIX utility `diff` (learned in Lab 3) to compare your output file with the output files in the above folder, to see whether they are identical.

Since these output files are given, we will **apply a penalty to any program that does not strictly meet the requirement on output format**. This includes the case in which your output has extra spaces, or has a missing newline at the end.

We will use different test cases to test your program thoroughly. Therefore, construct a number of test cases to test your program thoroughly before submitting it.

Examples: The following are some examples on running the executable file:

```
mhe@bluenose:~/csci2132/a5$ ./encry
$The quick brown fox jumps over the lazy dog
?Wkh txlfn eurzq ira mxpsv ryhu wkh odcb grj
mhe@bluenose:~/csci2132/a5$ ./encry
?Wkh txlfn eurzq ira mxpsv ryhu wkh odcb grj
$The quick brown fox jumps over the lazy dog
mhe@bluenose:~/csci2132/a5$ ./encry
$12345
?12345
mhe@bluenose:~/csci2132/a5$ ./encry
to be or not to be
The message must start with either $ (for encryption) or ? (for decryption)
```

Hint: Since the encryption process is essentially a cyclic shift of English alphabet, modulo arithmetic would be helpful. You might be inclined to use the `%` operator for this. Be careful when using this for decryption: in C, when one and only one of the two operands of the `%` operator is negative, the result is also negative. To avoid issues caused by this, when decrypting a message, you can (cyclically) shift characters $(26-3)$ positions to the right, instead of 3 positions to the left.

2. [15 marks] One strategy of problem solving is called "reducing to known problems". Informally speaking, this means that when we encounter a new problem, we can try to solve it by making use of solutions to other problems that we already know how to solve.

   Here, you are required to use this strategy to count the number of distinct elements in an `int` array.

   For example, if the elements in an array are `1 15 2 32 15 4 7 9 -1 15`, then the number of distinct elements in the array is 8, since there are eight different numbers in this array.

   This problem can be solved by making use of solutions to the sorting problem. You are asked to find out how to make use of sorting to solve this problem, and then implement your solution. To simplify your work, in your implementation, you can use any reasonable sorting algorithm including insertion sort and bubble sort (though you should know why they are not as efficient as better solutions), and you will not lose any marks if you do not implement the most efficient sorting algorithms. You are forbidden to use any existing C library functions that can be used to sort an array, such as `qsort`. You will lose a lot of marks if you do so.

   No marks will be given if your solution is not based on sorting at all.

   General Requirements: Use `a5q2.c` as the name of your C source code file.

   We will use the following command on bluenose to compile your program:

```
gcc -std=c99 -o distinct a5q2.c
```

Your program should NOT print anything to prompt for user input. It reads exactly **ten** `int` values from stdin, and stores them in an array. It then prints the number of distinct elements in this array, followed by a newline symbol.

Therefore, when you run your program by entering `./distinct`, the program will wait for your input. If you enter `1 15 2 32 15 4 7 9 -1 15`, the program will output `8`

For this question, no error handling is required. You can assume that the user correctly enters 10 `int` values.

Testing: To test your program automatically, we will use it as a UNIX filter.

For the example above, we will test it using the following command in the directory containing the executable program:

```
echo "1 15 2 32 15 4 7 9 -1 15" | ./distinct
```

The output should be:

```
8
```

To help you make sure that the output format of your program is exactly what this question asks for, several files are provided in the following folder on bluenose to show how exactly your program will be automatically tested:

```
/users/faculty/prof2132/public/a5test/
```

In this folder, open the file `a5q2test` to see the commands used to test the program with two test cases. The output files, generated using output redirection, are also given.

To check whether the output of your program on any test case is correct, redirect your output into a file, and use the UNIX utility `diff` (learned in Lab 3) to compare your output files with the output files in the above folder, to see whether they are identical.

Since these output files are given, we will **apply a penalty to any program that does not strictly meet the requirement on output format**. This includes the case in which your output has extra spaces, or has a missing newline at the end.

We will use different test cases to test your program thoroughly. Therefore, construct a number of test cases to test your program thoroughly before submitting it.

Note: Since after sorting, all that is required to compute the answer is essentially a linear scan, this means that our solution is as efficient as the sorting algorithm used in our implementation. As sorting is very well studied, this means that counting the number of distinct elements in this way is very efficient. As you may already know, the number of comparisons required of the best sorting algorithms is proportional to only $n \lg n$.

This type of questions are typical technical interview questions for software developers (and your interviewer will not simply tell you the solution).