CSCI 1320 (Principles of Algorithm Design I), Fall 2007

Homework 6

Assigned: November 6, 2007.

Due: November 13, 2007, at 5pm.

Credit: 40 points.

1 Reading

Be sure you have read chapter 7 and sections 1 through 4 of chapter 8.

2 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Every code file should begin with comments identifying you as the author and describing its purpose. It should be readable to human readers as well as to the compiler, so use consistent indentation and meaningful variable names. Feel free to cut and paste code from any of the sample programs on the course Web site.

Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 1320 homework 6"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in. Remember also that I will usually deduct points if your programs do not compile without warnings using the flags `-Wall` and `-pedantic`.

1. (10 points)   In the early days of the Internet, a popular way of disguising text that some might find offensive was to encode it using a scheme called rot13 (short for "rotate 13"). In this scheme, all letters are rotated 13 positions (so 'a' becomes 'n', 'b' becomes 'o', 'n' becomes 'a', etc.). Spaces, digits, punctuation, etc., are not changed. (An advantage of this scheme is that if you apply it twice, you get the original text back. Think about why!)

   Write a C program that uses this scheme to encode text from input file `hw6a-input.txt`, put the result in output file `hw6a-output.txt`, and print to standard output the number of characters processed (including the ones that were not changed). For example, if `hw6a-input.txt` contains the following

   ```
   Now is the time for all good persons to come to the aid of their
   party.  Hello world!  1234  !@#$
   ```

   then `hw6a-output.txt` will contain the following

   ```
   Abj vf gur gvzr sbe nyy tbbq crefbaf gb pbzr gb gur nvq bs gurve
   cnegl.  Uryyb jbeyq!  1234  !@#$
   ```

and the program will print that 98 characters had been processed (including spaces, newline characters, etc.). The program should print a reasonable error message if it cannot open the input and/or output file.

Hints:

- Here is a partial function to do the required encoding.

```
int encode(int input_char) {
    if (('a' <= input_char) && (input_char <= 'm')) {
        return input_char + 13;
    }
    else if (('n' <= input_char) && (input_char <= 'z')) {
        return input_char - 13;
    }
    /* add code for uppercase letters, other characters */
}
```

- You can check that your program is counting characters correctly by comparing the number it prints with the number you get by typing the command `wc -c hw6a-infile.txt`.

2. (30 points) Write a C program that reads text from standard input, up to end of file (control-D if reading from the terminal), and prints, for each possible character, how many times it occurs in the input. Print information only about the characters actually present in the file. For example, your program should print something like the following for the input shown for the previous problem:

```
count for character 10 (not printable) is 2
count for character 32 ( ) is 21
count for character 33 (!) is 2
count for character 35 (#) is 1
count for character 36 ($) is 1
count for character 46 (.) is 1
count for character 49 (1) is 1
count for character 50 (2) is 1
count for character 51 (3) is 1
count for character 52 (4) is 1
count for character 64 (@) is 1
count for character 72 (H) is 1
count for character 78 (N) is 1
count for character 97 (a) is 3
count for character 99 (c) is 1
count for character 100 (d) is 3
count for character 101 (e) is 7
count for character 102 (f) is 2
count for character 103 (g) is 1
count for character 104 (h) is 3
count for character 105 (i) is 4
count for character 108 (l) is 5
count for character 109 (m) is 2
```

```
count for character 110 (n) is 1
count for character 111 (o) is 11
count for character 112 (p) is 2
count for character 114 (r) is 5
count for character 115 (s) is 3
count for character 116 (t) is 7
count for character 119 (w) is 2
count for character 121 (y) is 1
```

Hints:

- Characters are actually small integers, and can be treated as such. If `c` is an `int` representing a character, you can print its numeric value and its value as a character with a line such as the following:
  `printf("%d %c\n", c, (char) c);`
- You will probably want an array to hold counts for the various characters. The size of the array should probably be `UCHAR_MAX` plus 1. (`UCHAR_MAX` is a constant defined in `limits.h` and represents the largest value of a character.)

Initially you will probably want to test your program with input from the keyboard, but when you get it working, try running it on a longer text file (which you might get from the Internet — or you could use a file containing source code for one of your programs!). Remember that you can have your program get its input from a file rather than the keyboard using input redirection.