THE UNIVERSITY COLLEGE
OF THE CARIBOO

Computing Science – **COMP213**

Lab Exercises #11 – C – *Text Files*

Marks: *5 marks* Due: *Wednesday, Dec 1., 2004*

Introduction

It is said by many that, "practise makes perfect."

Well, it is only assumed that *perfect* is the result, yet there must be practise in any case. Hence, the following exercises provide another opportunity to play with files, as well as review ideas from previous exercises.

Resources and References

Even though this is the final set of lab exercises, you are still ***strongly*** encouraged to use the C programming references placed on the course website.

Exercises / Programming Problems

1. There are a few problems with this program that should output a series of values to a file—*fix it!*

```
int main(void)
{
   int fp;
   int k;

   fp = fopen ("numbers");
   for (k=0; k<3000; k++)
      putc (k,fp);

   fclose ("numbers");
   return (0);
}
```

2. Write a program that reads all the characters in a text file.

   The purpose of the program is to calculate and display the "average character." The average character is determined by adding together the integer ASCII values of the characters read from the file, divided by the number of characters. As an integer, the average value represents an ASCII symbol: the average character.

3. Write a program that asks the user for an input text file and counts the number of occurrences of each alphabetic character in the file (any symbols other than A..Z are ignored).

   A table is displayed showing a count of only the letters in the file (if a particular letter count is zero (0), that letter is not displayed in the table).

   Consider the following suggestions for the program,

   - case is not important: 'A' and 'a' are the same alphabetic character
   - declare an **int** array of 26 long (the number of letters in the English alphabet), with each element storing the count for a particular character: [0] – 'A', [1] – 'B', [2] – 'C', …
   - instead of using a large **if-**or **switch-case** statement to determine which element to incr. the count, recall that all ASCII characters are in alphabetic sequence starting with 'A' (65); therefore, by subtracting 65 from the ASCII value of the character just read, this is the index to the array.

   Test the program with a small file that contains a known number of specific characters.

4. Running a program from a command-line (CL) allows for an extra opportunity to provide a program with input as it runs, as compared to just double-clicking on a GUI icon.

   Command-line arguments are processed to a program through the **main()** function parameters.

   Compile, and test, the following program that echoes the contents of a file to the screen, or echoes the contents to another file, with the names of the files being obtained from the command-line.

```
/* File: arg_copy.c
   Purpose: program that copies contents of source file to console or other
     file, depending on command-line arguments:

      args[1] - contains name of source file
      args[2] - contains name of destination file; if empty send output to
                  console.

      Any file errors (error opening, or missing args[1]) results in calling
        exit(0).

*/
#include <stdio.h>
#include <stdlib.h>    // for exit(0);

 // CL params: argc - number of arguments
 //   args - array of c-strings (array of char): argument data
 //   (note: char *args[] can also be coded as char **args)
int main (int argc, char *args[])
{
    // args[0] - name of command/program being executed
    // args[1]..[n] - command-line arguments 1..n
   FILE *finput, *foutput;    // file input & output pointers

   char ch=0;                 // the transfer character

   //-------------------------------------------------
   switch (argc)        // decide what to do on number of arguments
   {
       case 0:        // no arguments; impossible (will never happen!)
          exit(0);        // stop

       case 1:        // 1 argument (the program name: arg_copy
          printf ("Insufficient arguments.\n");
          exit(0);        // stop

       case 2:        // 2 arguments; copy to console
          finput = fopen (args[1],"r");    // open file as read
          foutput = stdout;                // open to console
          break;

       case 3:        // 3 arguments; copy to other file
          finput = fopen (args[1],"r");    // open file as read
          foutput = fopen (args[2],"w");   // open file as write
          break;

       default:       // 4, or more, arguments
          printf ("Too many arguments.\n");
          exit(0);       // stop
   }

     // if file open errors; similar to (finput==NULL) || (foutput==NULL)
   if ( (ferror(finput)!=0) || (ferror(foutput)!=0) )
   {
       printf ("Error opening on of the files.\n");
       exit(0);
   }
```

```
    // copy content
    ch = getc(finput);          // get initial character
    while ( !feof(finput) )     // loop while not end of file
    {
        putc(ch,foutput);          // output character
        ch = getc(finput);         // get next character
    }

    // close files
    fclose(finput);             // close input file
    fclose(foutput);            // close output file

}// end of main(): arg_copy.c
```

5. Modify the **arg_copy.c** (from the previous question), so that a 4th argument is possible. This parameter, called **security**, is a single character that must be either an 'E' (for encoding) or a 'D' (for decoding); other values are an error and the program stops.

   If security is to encode ('E'), each character is *rotated one bit to the right* before being written; if security is to decode ('D'), each character is *rotated one bit to the left* before being written.

   Question: *Can the program be executed, and the encoding/decoding performed, if the arguments describe showing to the console?*

   You will need to use a modification of the **rotateInt()** function, calling it **rotateChar()** instead.
   Also, use the nature of a "string" in C just being an array of char to select the first character in the argument: **args[3][0]** .

   Test the program by encoding a source file to an intermediate file, decoding the intermediate file to a destination file, and examining the source and destination files: *are they the same*?

6. Modify the "average character" program you wrote in question 2.

   Add a line that displays the address of the input file pointer as it reads a file, and examine: *does the address change*? Explain why this address does, or does not, change.

Conclusion

You are encouraged to complete all problems, but only problems #3 and #5 are required for submission.
Provide properly documents source code, output captures necessary (output prints only where reasonable).