THE UNIVERSITY COLLEGE
OF THE CARIBOO

Computing Science – **COMP213**

Lab Exercises #11 – C – *Text Files*

Solutions

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);
}
```

The mistakes identified with ← above are corrected below,

```
int main(void)
{
   FILE *fp;   ←
   int k;

   fp = fopen ("numbers","w");   ←
   for (k=0; k<3000; k++)
      fprintf (fp,"%d ",k);   ←

   fclose (fp);   ←
   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.

```
int main(void)
{
   FILE *in;
   char filename[40];

   char inChar=0;     // input character
   int totA=0;        // total of ASCII values
   int numChars=0;    // number of characters read in
   char avrgChar=0;   // average character
   //------------------------------------------------------------------

   printf ("For which file shall I determine the average char? ");
   scanf ("%s",filename);
   getchar();
   in = fopen (filename,"r");  // open file as read

   inChar = getc(in);
   while ( inChar != EOF )    // or, !feof(in) ;while not file eof yet
   {
      numChars++;                  // incr. number of characters read
      totA += (int)inChar;         // add ASCII value to total
      inChar = getc(in);           // read char from file
   }
     // calculate average
   avrgChar = (char)(totA/numChars);

   fclose (in);

     // display results
   printf ("For file %s, %d chars. read. \n Average char is: %c",
       filename,totA,avrgChar);

   getchar();
   return (0);
}
```

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.

```
int main(void)
{
   FILE *infile;
   char filename[40];

   char inChar=0;     // input character
   int  letters[26];  // array to contain character counts
   int  i=0;          // loop control

   //-----------------------------------------------------------------
    // zero the character count array
   for (i=0; i<26; i++)
      letters[i] = 0;

   printf ("For which file shall I produce a character count? ");
   scanf ("%s",filename);
   getchar();

   infile = fopen (filename,"r");   // open file as read
   inChar = getc(infile);          // read char from file
   while ( inChar != EOF )         // while not file eof yet
   {
        // ensure character is in uppercase
      if ((97 <= inChar) && (inChar <= 122)) // if char 'a'<=x<='z'
         inChar = inChar - 32;                 // lower to uppercase

      if ((65 <= inChar) && (inChar <= 90))  // if char 'A'<=x<='Z'
         letters[inChar-65]++;                 // incr. approp. element

      inChar = getc(infile);                 // read char from file
   }
   fclose (infile);

     // display character table
   printf ("The character counts\n");
   for (i=0; i<26; i++)
      if (letters[i]>0)    // display letters that have counts
         printf (" %c : %d \n",(char)(65+i),letters[i]);

   getchar();
   return (0);
}
```

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 one 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
```

Using the program depends completely on the command-line arguments. Examples,

./arg_copy → *(argc = 1) results in the message, "Insufficient Arguments."*

./arg_copy file.txt → *(argc = 2) displays the contents of* file.txt *to stdout (the console)*

./arg_copy file.txt other.txt → *(argc = 3) copies the contents of* file.txt *to* other.txt

./arg_copy file.txt other.txt thing → *(argc = 4) displays,"Too many arguments."*

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 <u>must be</u> 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*?

```
/* File: lab11q5.c
   (a modification of the program 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.

      (modification)
      args[3] - contains the encoding format: 'E'-encode, 'D'-decode

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

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

char rotateRight1 (char source);     // rotate char parameter to right by 1 bit
char rotateLeft1 (char source);      // rotate char parameter to left by 1 bit

 // 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 codeType = 0;         // the type of coding: 'E'-encode, 'D'-decode
   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;
      case 4:        // 4 arguments; copy to other file, but D/Encode
          finput = fopen (args[1],"r");    // open file as read
          foutput = fopen (args[2],"w");   // open file as write
          codeType = args[3][0];     // grab the character: E or D
          if ((codeType != 'E') && (codeType != 'D'))  // if not E or D
          {
             printf ("Encode or Decoding not indicated correctly.\n");
             exit(0);
          }
          break;

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

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

     // copy content
   ch = getc(finput);        // get initial character
   while ( !feof(finput) )   // loop while not end of file
   {
          // determine what to do with the character
       if (codeType == 'E')      // codeType is to Encode: rot. 1 bit to right
          ch = rotateRight1(ch);    // call function to rotate right by 1
       else if (codeType == 'D') // codeType is to Decode: rot. 1 bit to left
          ch = rotateLeft1(ch);     // call function to rotate left by 1
       //else...don't to anything to the character!!

       putc(ch,foutput);        // output character
       ch = getc(finput);       // get next character
   }
     // close files
   fclose(finput);           // close input file
   fclose(foutput);          // close output file
}

/*  rotateRight1() - rotate char parameter to right by 1 bit */
char rotateRight1 (char source)
{
   unsigned char lostBit = source & 1;     // store LSB bit that will be lost
   source = source >> 1;          // shift source to right by 1; MSB becomes 0
   lostBit = lostBit << 7;        // move lost bit to MSB position
   source = source | lostBit;     // put them back together
   return (source);               // return as a char that is rotated to right
}

/*  rotateLeft1() - rotate char parameter to left by 1 bit */
char rotateLeft1 (char source)
{
   unsigned char lostBit = source & 128;   // store MSB bit that will be lost
   source = source << 1;          // shift source to left by 1; LSB becomes 0
   lostBit = lostBit >> 7;        // move lost bit to LSB position
   source = source | lostBit;     // put them back together
   return (source);               // return as a char that is rotated to right
}
```

Of special note for the program is the use of the datatype **unsigned char** in the *rotation* functions. By using this type, any possible integer operations on the type maintain that the value is always positive.

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.

```
/* File:  lab11q2.c
   Modified to:  lab11q6.c
   Purpose:
       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.

   Modification: shows the file pointer address.
      The purpose is to display what the file pointer is "pointing to",
      *not* the address of the file pointer!
*/
#include <stdio.h>

int main(void)
{
   FILE *in;
   char filename[40];

   char inChar=0;     // input character
   int totA=0;        // total of ASCII values
   int numChars=0;    // number of characters read in
   char avrgChar=0;   // average character
   //------------------------------------------------------------------

   printf ("For which file shall I determine the average char? ");
   scanf ("%s",filename);
   getchar();

   in = fopen (filename,"r");   // open file as read

   inChar = getc(in);
   printf ("\nAddress in file pointer: %p",in);
   while ( inChar != EOF )          // or, !feof(in) ;while not file eof yet
   {
      numChars++;                  // incr. number of characters read
      totA += (int)inChar;         // add ASCII value to total
      inChar = getc(in);           // read char from file
      printf ("\n -> %p",in);
   }
     // calculate average
   avrgChar = (char)(totA/numChars);

   fclose (in);

     // display results
   printf ("For the file %s, %d chars. were read. \n The average char is: %c",
       filename,totA,avrgChar);

   getchar();
   return (0);
}
```

Output from the program will indicate that the address stored in the file pointer does not change.

The file stream pointer (whether for input or output) is not a dynamic variable. The file stream pointer always points to the same memory location, but the memory location contains different data as the various file functions move data between the file and memory.

File Input: input functions read an appropriate amount of data from the file and store at the memory location.

File output: output functions take the value at the memory location and write it to the file.

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).