THE UNIVERSITY COLLEGE
OF THE CARIBOO

Computing Science – **COMP213**

Lab Exercises #10 – C – *Multi-file Compilation* and *Text Files*

Marks: *5 marks* Due: *Wednesday, Nov. 24, 2004*

Introduction

Multi-file Compilation

Except for simple class exercises and assignments, very few real-world applications have such limited scope to be encompassed by only a single program file. By using multiple source code files, complex projects can be split amongst programmers and code can be segmented and shared effectively between programs.

Further, once a library file is compiled, and proven stable, it can be used across various projects. This leads to reliable and consistent program development.

Text Files

Long term data storage requires programs to have the ability to accept input (read) and generate output (write) to disk files. Files are either text files, using only ASCII or Unicode character coding, or pure binary files, in which the data is understood only by the program.

It is understood that text files are just binary files with all data organised as ASCII character codes: each data value is 1-byte (or as Unicode as 2-bytes).

For this exercise, only ASCII text files are considered.

Resources and References

You are **strongly** encouraged to use the C programming references placed on the course website. For both topics examined in this exercise, the best learning tool is experience and example.

Exercises / Programming Problems

1. What is the difference in usage between the <> and "" in the following example **#include** directives.

```
#include <stdio.h>
#include "utilprog.h"
```

2. Modify your solution to problem #8 in Lab Exercises #9 – the **rotateInt**() function.

   Change the program such that all functions, except for **main**(), are placed in a file called **funcs.c**. Create a proper header file, called **funcs.h** used to access the functions in **funcs.c**.

   With a valid header file present, use one of the following command-line compilation methods:

   1) separate compilation and linkage:

```
__> gcc -c funcs.c                          ← creates the object file, funcs.o
__> gcc lab8_9.c funcs.o -o lab8_9.out      ← if lab8_9 is the application file
```

   or

   2) combined compilation and linkage:

```
__> gcc lab8_9.c funcs.c -o lab8_9.out      ← again, if lab8_9 is the application file
```

3. Based on the solution to the problem above, if the file **funcs.c** was compiled, and the object file **funcs.o** was then distributed for use without its source-code file, does the **funcs.h** still need to be present during compilation of a final application?

4. Assume the following header file is available, does it require an accompanying library or resource file (_.c)? If so, what would the file look like?
   Considering that this header file could itself be included within many files (both .h and .c) in a single multi-file compilation, what seems to be missing? *(hint: compiler directives)*

```
/* file: bool.h */

typedef int BOOLEAN;    // create new type
#define TRUE 1          // TRUE is defined by 1
#define FALSE 0         // FALSE is defined by 0

/* end of: bool.h */
```

5. Rather than having the program open/close input and output text files, the operating system can redirect text files for input to a program, and redirect program output to a file.

   Consider the following example, in which a program **sample** is provided input via **in.dat** (so the user never types) and console output saved to **result.txt**.

```
__> ./sample <in.dat >result.txt
```

   Describe one advantage this technique has over requiring the program to open both files.
   Also, describe one disadvantage. *(hint: what type of program-user interaction is occurring?)*

6. Describe how all the following functions indicate that a "file error" or "end of file" has been reached and no further reading is possible? *(Some research is required; consider the Linux* **man** *pages.)*

   fopen(), getc(), fgets(), fscanf()
   and **feof(), ferror()**

7. Write a program that reverses the content of a text file, for example,

   input file content:

```
Sammy was a puppy.
Sammy was a brown puppy.
```

   output file content:

```
.yppup nworb a saw ymmaS
.yppup a saw ymmaS
```

   In the program, create a function **reverseContent()** that performs the reversal. This function has two parameters: an input file pointer, and output file pointer, and returns nothings. Place this function in a resource file along with an accompanying header file.

   Of the techniques to solve the problem, the following are the most popular:

   - use a recursive function that reads a character, calls itself to read another, then displays the character it read before calling itself; or
   - read the text file once, determine the number of characters, close the file, declare a character array of the correct size, then read the file again using the array to store the data; then display the array in reverse to the output file

Conclusion

You are encouraged to complete all problems, but only problems #2 and #7 are required for submission.
Provide properly documents source code (output prints only where reasonable).