THE UNIVERSITY COLLEGE
OF THE CARIBOO

Computing Science – **COMP213**

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

Solutions

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"
```

The <> tell the pre-processor that the header file is located in the standard compiler directory defined in the OS. The "" tell the pre-processor that the header file is in the local (same) directory as the source file being compiled.

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
```

The file **funcs.h** contains the following,

```
#ifndef FUNCS_H
#define FUNCS_H

  ... headers/prototypes for functions

#endif
```

The application file with the **main()** will contain the line,

```
#include "funcs.h"      // include headers for funcs.c
```

The funcs.c source file may also contain an #include "funcs.h" if required by the compiler (depending on the methods present, and if the prototypes need to be loaded).

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?

The header file is still required, because the pre-processor requires the prototypes for the functions in funcs.c in order to compile the application file.

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 */
```

Although the header file contains the absolute minimum to define a **boolean**-style datatype, along with values for TRUE and FALSE, it ignores the possibility of the header files being "included" more than once.

The situation that demands the #ifndef…#define…#endif directives is something such as,

- the header file **spec.h** is included in an application program, along with **other.h**
- **spec.h** also includes **other.h** because it used by the source code file **spec.c**

With **other.h** being included more than once, it's prototypes and constants will be declared multiple times, which is something the compiler considers an error. With the required directives added to the header file, its prototypes and constants are only declared once.

The proper form of the header file,

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

#ifndef BOOL_H
#define BOOL_H

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

#endif
/* 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?)*

There are many advantages and disadvantages, but the following are the main reasons,

- advantage: user interaction is not required, and file-usage code is not required in the program
- disadvantage: limited flexibility in the input values, since the user is not available for input

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()**

   fopen() – returns NULL if there is an error

   getc() – returns the EOF character (ASCII value: 26)

   fgets() – places a NULL in the char array (c-string)

   fscanf() – returns 0 if error, 1 if all okay; and places "zeroes" in variables that did not get values

   feof() – returns 0 if not-EOF, >0 if at EOF

   ferror() – returns 0 if no file error occurred, >0 if an error has occurred

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

   The solution is split into the following files,

   Lab10_7_rev.c – support file containing the two functions that perform the reversal

   Lab10_7_rev.h – header file containing the two functions that perform the reversal

   Lab10_7.c – application file containing the necessary initialisations and function calls

Support File (lab10_7_rev.c)

```
/* File:   lab10_7_rev.c
   Author: Yanni Giftakis
   Date:   Nov 20, 2004
   Purpose:  Reverse the content from the source text file to destination file.

      For experimentation's sake, two functions have been defined:
          revContentArray() - uses an array to reverse the content
          revContentRec()   - uses a recursive approach to reverse content

*/
#include <stdio.h>
#include "lab10_7_rev.h"       // included for the sake of completeness

//-------------------------------------------------
// reverse content of source, and write to dest; count contains # of chars
void revContentArray (FILE *source, FILE *dest, int count)
{
    char chars[count];       // declare an array of the correct size
    int  i=0;                // loop control variable
    //-----------------------------------------------------------------
    for (i=0; i<count; i++)     // loop to store characters in array
       chars[i] = getc(source);    // read next character fraom file

    for (i=count-1; i>=0; i--) // loop from last char to first
       putc(chars[i], dest);       // write character to file
}// end of revContentArray()

//-------------------------------------------------
// reverse content of source, and write to dest
void revContentRec (FILE *source, FILE *dest)
{
    char ch=0;         // input character
    //-----------------------------------------------------------------
    if ( !feof(source) )    // if not at the end of source file
    {
        ch = getc(source);                 // read next character
        revContentRec (source, dest);      // recur. call for next char
        putc (ch, dest);                   // output char to dest
    }
    // if at end, just end of the function
}// end of revContentRec()
// end of lab10_7_rev.c
```

Header File (lab10_7_rev.h)

```
/* File:   lab10_7_rev.h
   Author: Yanni Giftakis
   Date:   Nov 20, 2004
   Purpose:  header file containing definitions of the following functions:
      revContentArray() - uses an array to reverse the content
      revContentRec()   - uses a recursive approach to reverse content
*/
#include <stdio.h>

#ifndef LAB10_REV_H
#define LAB10_REV_H

   // reverse content using array concept
void revContentArray (FILE *source, FILE *dest, int count);
  // reverse content using recursive concept
void revContentRec (FILE *source, FILE *dest);

#endif
```

Application File (lab10_7.c)

```
/* File:   lab10_7.c
   Author: Yanni Giftakis
   Date:   Nov 20, 2004
   Purpose:  Reverse the content from the source text file to destination file.

      For experimentation's sake, two functions have been defined:
          revContentArray() - uses an array to reverse the content
          revContentRec()   - uses a recursive approach to reverse content

      These two functions are located in the files: lab10_7_rev.h & .c
*/
#include <stdio.h>
#include "lab10_7_rev.h"

int main ()
{
    FILE *source=NULL, *dest=NULL;    // source & destination file handles
    char sourceN[40], destN[40];      // source & destination file names

    int count = 0;                    // number of characters in file

    //-------------------------------------------------------------------
    printf ("Array version:\nEnter the name of the source file:");
    scanf ("%s",sourceN);
    printf ("Enter the name of the destination file:");
    scanf ("%s",destN);

      // open files
    source = fopen (sourceN, "r");
    dest = fopen (destN, "w");
      // count characters
    count = countChar (source);
      // close and open read file again
    fclose (source);
    source = fopen (sourceN, "r");

      // reverse content
    revContentArray (source, dest, count);   // using array concept

        // close files
    fclose (source);
    fclose (dest);

    //-------------------------------------------------------------------
    printf ("Recursive version:\nEnter the name of the source file:");
    scanf ("%s",sourceN);
    printf ("Enter the name of the destination file:");
    scanf ("%s",destN);

      // open files
    source = fopen (sourceN, "r");
    dest = fopen (destN, "w");

        // reverse content
    revContentRec (source, dest);            // using recursive concept

        // close files
    fclose (source);
    fclose (dest);
}// end of main()
```

```
//counts the number of characters in a file
int countChar (FILE *fp)
{
   int count = 0;
   for (count=0; !feof(fp); count++)  // loop while not EOF
       getc (fp);
   return (count);
}// end of countChar()

//end of lab10_7.c
```