CIS 345/545 – Homework 2

In this assignment you will write several small C programs that will be used to learn some additional C programming techniques. Follow the steps given below carefully, recording the results as you go through them. Each program must print your name when it first starts.

Passing parameters to C programs from the command line:

This example shows how to access the command line arguments, as well as using conditional compilation to include a main program or not, depending on the symbol STAND_ALONE being defined or not. This will be a useful technique in the following parts of this assignment. Search the web for "Command line arguments in C", "Conditional compilation in C" or "C preprocessor" for more info.

```
---------------------------------------------------------------------------
/* This program is saved as showargs.c, compiled as showargs */
#include <stdio.h>

void showargs(int, char **);

#define STAND_ALONE 1

#ifdef STAND_ALONE
int main (int argc, char *argv[]) {
   printf("Written by YOUR NAME\n");
   showargs(argc, argv);
   return (0);
}
#endif

void showargs(int argc, char *argv[]) {
   int i;
   printf("Program name: %s\n", argv[0]);
   printf("There are %d arguments\n",argc);
   for (i=1; i<argc; i++)
      printf("%2d. %s\n",i, argv[i]);
}
---------------------------------------------------------------------------
Reading and displaying an ASCII text file line by line:
---------------------------------------------------------------------------
/* This file is saved as readtext.c, compiled as readtext */
#include <stdio.h>
void main(int argc, char *argv[]) {
   FILE *fin;
   char buffer[100];
   printf("Written by YOUR NAME\n");
   if (argc != 2) {printf("Usage: %s filename\n", argv[0]); exit(1);}
   fin = fopen(argv[1], "r");
   if (!fin) {printf("Unable to open %s\n", argv[1]); exit(1);}
   while (fgets(buffer, 99, fin)) fputs(buffer, stdout);
   fclose (fin);
}
---------------------------------------------------------------------------
```

Part 1: Write individual C programs to do the following:

findtext.c Given a string and a file name as parameters, if any lines in the file contain the string, this program displays (once) the file name and then the appropriate lines from the file with line numbers. For example, **`findtext be hamlet.txt`** would show:

hamlet.txt
1. To be or not to be- that is the question:
9. Devoutly to be wish'd. To die- to sleep.
15. For who would bear the whips and scorns of time,
21. With a bare bodkin? Who would these fardels bear,
26. And makes us rather bear those ills we have

right.c Displays the 3 integers it is passed as parameters and prints either "is a right triangle" or "is not a right triangle" as may be appropriate. The quotation marks are not printed. For example, **`right 3 4 5`** would print:
3 4 5 is a right triangle
while **`right 4 5 6`** would print:
4 5 6 is not a right triangle

count.c Displays the number of characters, words, lines, and file name of the file(s) given as parameters. For this program, a word is something that is surrounded by white space. For example,
`count speech.txt` might print:
speech.txt
2345 characters
514 words
62 lines

Each program should be written using the conditional compilation technique shown in the program containing STAND_ALONE above.

Each file should contain a function that does the real work named the same as the source file name, but without the .c suffix, just as source file named showargs.c contains the function called showargs above.

Compile each program, giving it the name of its source file, less .c suffix (e.g., right.c compiles to right).

Run each program as follows, capturing the output with the Linux script command:

```
ls -l >tempfile
cat tempfile
findtext .c tempfile
right 4 3 5
count *.c
right 3 7 23
```

Part 2: Multifunction Utility Program

Modify the three C source files created in part 1 by removing the definition of the identifier STAND_ALONE. Write a new C program called mu.c that acts as a front-end command line parser, calling findtext, right, and count functions as directed. The command line arguments given to mu.c are arranged in groups, each beginning with -f, -r, or -c (which selects either findtext, right or count function to use), followed by the normal parameter(s) for that function.

For example, the command line:

```
mu -r 3 4 5 -f STAND_ALONE right.c
```

would perform the right triangle check on sides 3, 4, and 5, and then display the lines in right.c (with line numbers) that contain the string STAND_ALONE. If an invalid - argument is given (such as -x), display an error message with bad - argument and its parameter(s) and then continue with next - argument.

Compile your source files using: **`gcc mu.c findtext.c right.c count.c -o mu`**
This compiles all four source code programs (mu.c driver plus three other .c programs) into one output executable file called mu.

Test program with the following commands, which should be recorded using the Linux script command.

```
ls -l >tempfile
cat tempfile
mu -f .c tempfile -r 5 4 3 -x 1.234 alpha -c *.c -r 3 7 23
```

Part 3: Separate Compilations and Object Files

Without changing any of your source files, compile each of the four files to produce an object file.
Use the commands:

```
gcc -c mu.c
gcc -c findtext.c
gcc -c right.c
gcc -c count.c
```

to produce object files called mu.o, findtext.o, right.o, and count.o.

Then link these object files together using the command:

```
gcc -v -o multi mu.o findtext.o right.o count.o
```

Observe the ld line and see what is being done automatically for you by the loader (the -v does a verbose mode which shows all internal steps).

Record, using the script command, the compilation and linking above, as well as the running of the following set of commands that test your program:

```
ls -l >tempfile
cat tempfile
multi -f .c tempfile -r 5 4 3 -c *.c -r 3 7 23
```

Part 4: Building a Function Library

In this part, you will build a function library and investigate if only the needed functions or all the functions in the library are included in the executable program you link using the library. First, create an object file called extra.o from this source:

```
void extra() { }
```

Next, create the executable program extra using this command:
`gcc -o extra mu.o findtext.o right.o count.o extra.o`

Examine the output of **`nm -g extra`** which produces a list of all the public names in the file to see if you find extra. If so, its code is included in the executable program.

Now create a function library called mylib using this command:
`ar r mylib.a findtext.o right.o count.o extra.o`
`ranlib mylib.a` (this is not needed with some versions of ar)

Create a new executable form of your program called mu2 using this command:
`gcc -o mu2 mu.o mylib.a`

Use the **`nm -g mu2`** command to determine if extra is included or not in mu2.

Finally, test mu2 to see if it still works properly by using these commands, recording the output with the Linux script command:

```
ls -l >tempfile
cat tempfile
mu2 -f .c tempfile -r 5 4 3 -c *.c -r 3 7 23
```

Turn in the following:
Since there are a lot of parts to this assignment, it is especially important that you organize your work so it is easy to find what I will be looking for. Organize things in the order of the steps and the list below.

Line numbered listings of mu.c findtext.c right.c count.c and extra.c
(Using cat -n or pr -n are two ways of producing a line numbered listing)
The script captured from part 1
The script captured from part 2
The script captured from part 3
The script captured from part 4

See the instructions on the next page for submission. Do not submit this assignment to Blackboard!!!

Submission:

1. Each student submits one copy of each item listed on the previous page.

2. Create a folder and name it [first name]_[last name]_h2 ($ mkdir john_smith_h2)

3. Copy all your source code to the above folder (source code only, no binary files!).

4. Copy all your scripts captured from each of the four parts to the above folder.

5. scp the folder (e.g., john_smith_h2) to your account on grail (if it's not already there).

6. ssh log in grail, go to the parent directory of the folder you created, and run the following: (suppose that your folder is "john_smith_h2")

turnin22 proj2@cis545j john_smith_h2

You should see a list of files followed by a message like the following:

You are about to turnin X files for [XKB] for proj2 to cis545j

* * * Do you want to continue?

Answer y or yes to submit or answer n or no if you do not want to submit.

If you answered y or yes, you should now see a list of the files being submitted, followed by:

* * * TURNIN of proj2 to cis545j COMPLETE! * * *

7. You can submit more than one time. Your most recent submission will always automatically overwrite the previous one. Do note, however, that if your latest submission is sent after the deadline, it may lose points for being late, even if your earlier submission was on time.