Homework 4

Due: November 23, 2015 **Points:**100

Note about points. If you submit this by Friday, November 20, at 11:55pm, we will add 10 points to whatever score you receive.

Linux

All these questions are to be answered using the CSIF systems. If you use some other system, your answers may differ, and we will grade based on the CSIF systems.

You ***must*** put your answers in a file called "Linux.txt" or "Linux.pdf". If you need to submit pictures also, put them in files called "Linux*n*.*ext*", where *n* is a digit and *ext* is an appropriate extension. ***If you call your file(s) anything else, or submit something other than text or a PDF file, we will not grade it and you will get 0!***

1. (*5 points*) I have a file in my directory named `-r` (that is a "`-`" followed by the letter "`r`"). I want to rename it `dashr`. What command should I type to do this? Give the full command, not just the name of the command.
2. (*5 points*) I want to make the file *libprog.a* in my home directory available to everyone so they can copy it, but I do not want anyone to be able to see any other files in my home directory. How should I set the "other" bits of my directory's protection to do this?
3. (*10 points*) The file */usr/share/dict/words* contains a list of English words and abbreviations, one per line. How many words and abbreviations does it have? How many have the trigram "abc" in them? Remember, you must show the command(s) you use to get this information.

C Programming Language

Please do either of the two questions. You must pick one; you cannot do part of one and part of the other. In your submission, state which one you have done.

If you do question 5, you ***must*** submit a file called "MyProgLab.*ext*", where *ext* is any 3-letter extension. We will not look at the contents of this file; its presence tells us you did the MyProgrammingLab exercises.

If you do question 6, you ***must*** put your answers in a file called "CPL.txt" or "CPL.pdf". ***If you call your file(s) anything else, or submit something other than text or a PDF file, we will not grade it and you will get 0!***

4. (*20 points*) In MyProgrammingLab, please do the following questions: 10224 (second question of 9.2) and questions 10289–10293 (all questions of 10.1) . Please click on "Submit" for each answer, so you can see if your answer is correct. If it isn't, try again. We will consider only the last answer you submit. Also, do not submit your answers to SmartSite; simply say you have completed them, and we will get your score directly from MyProgrammingLab.
5. (*20 points*) In the textbook, please do questions 4 and 5 in the Review Questions section of chapter 10.

Programming

You ***must*** put your program in a file called "fibs.c". Do not include the timing function library. ***If you call your file anything else, or submit something other than a C program (for example, a text or PDF file containing the code), we will not grade it and you will get 0!***

6. (*50 points*) In homework #2, you wrote a program to print the Fibonacci numbers. Most of you did it iteratively; a few, recursively. This program has you compare the two approaches.

   Write a program that takes a single integer command-line argument, *n*. It prints the first *n* Fibonacci numbers in two ways. First, it prints them by repeatedly calling a function that computes the Fibonacci numbers iteratively. Second, it prints them by repeatedly calling a function that computes the Fibonacci numbers recursively. For example, to print the first 20 numbers, and time their computation, you would type

   ```
   fibs 20
   ```

   The output looks like this:

   ```
   Iterative:  0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
   Recursive:  0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
   Iterative timing:     0.000017
   Recursive timing:     0.000084
   ```

   Put one blank after the word "timing:" in the last two lines, and print the difference in times using the format "%12.6f".

   You need to check for three error conditions.

   (a) If the argument is not an integer, or does not begin with an integer (so "`hello`" is not a valid argument, but "`19hello`" is, and should be treated as the integer 19) use the following line to print an error message (here, `argv[1]` is the first argument after the command):

   ```
   fprintf(stderr, "%s not an integer\n", argv[1]);
   ```

   and exit with exit code 1.

   (b) If the argument is a non-positive integer, print the following error message on the standard error:

   ```
   Argument must be a positive integer
   ```

   and exit with exit code 1.

   (c) If you have more than one argument (not including the command name, of course), use the following line to print an error message (here, `argv[0]` is the zeroth argument, which is the command name):

   ```
   fprintf(stderr, "Usage: %s number\n", argv[0]);
   ```

   and exit with exit code 1.

   To do this program, you must write two Fibonacci functions, the first computing a Fibonacci number iteratively and the second, recursively. Here is a prototype for the functions:

   ```
   int iterfib(int n);   /* compute the n-th Fibonacci number iteratively */
   int recfib(int n);    /* compute the n-th Fibonacci number recursively */
   ```

   These both take an integer argument *n* and return the *n*th Fibonacci number. *They do not print anything — your main routine should do that!*

   The comparison of the two routines is to be done based on their time. Timing a routine requires that you obtain the time before calling the routine, then call it, and then obtain the time after the routine returns. Now subtract the first time from the second. To help, we have written two functions. The first function obtains the current time. Its prototype is:

   ```
   struct timeval *gettime(void);
   ```

   and it returns a structure containing the number of seconds (field `tv_sec`) and microseconds (field `tv_usec`) since time 0 (called "the epoch", it is January 1, 1970, at 12:00:00am). To use this function, you must put this line where you have the other includes:

   ```
   #include <sys/time.h>
   ```

The pointer that `gettime` returns points to a static area in the library. So, each time you call `gettime`, that changes. This means that if you need to refer to a previous value returned by `gettime`, be sure to copy it somewhere!

The second function returns the difference between two times in seconds as a double. Its prototype is:

```
double timediff(struct timeval *t1, struct timeval *t2);
```

and it returns the difference between the second and the first arguments in seconds as a floating point number.

The timing functions are in the object file "timeit.o", available at *~bishop/ecs30/timeit.o* on the CSIF. It is a binary file, and so will *only* work on the CSIF.

There is also a sample program using the two timing functions. It is available at *~bishop/ecs30/timeex.c* on the CSIF. This may help you use those functions. It shows you one way to save the value returned by `gettime`.

Finally, there is an executable program, *reffibs*, that will produce the proper output. It is available at *~bishop/ecs30/reffibs* on the CSIF. You can compare your output against it. Remember, though, that the times you get may differ from the times this program gets. Gradebot takes this into account when grading your program.

Hint: Here is the recommended approach. Say you want to print the first n Fibonacci numbers. First, get the time (call it `TimeIterBegin`) using `gettime`. Then in a for loop, call `iterfib(i)` with $i = 1, 2, \ldots, n$, and have the function return the i-th Fibonacci number. Print it while in the for loop, so you don't have to store it in an array. When done, get the final time (call it `TimeIterEnd`), again using `gettime`. Then subtract `TimeIterBegin` from `TimeIterEnd` using `timediff` (call this result `TimeIter`). That's the time to print the first n Fibonacci numbers iteratively. Repeat, but using `recfib(i)` rather than `iterfib(i)`. Then print `TimeIter` and `TimeRec`, the equivalent of `TimeIter` but for the recursive version.

Also, note the output spacing. There are 2 spaces after the colon in the lines that begin with "Iterative:" and "Recursive:", and one space after the colon in the lines with "timings:".

Debugging

You ***must*** put your corrected program in a file called "buggy.c". ***If you call your file anything else, or submit something other than a C program (for example, a text or PDF file containing the code), we will not grade it and you will get 0!***

7. (*10 points*) The program *buggy.c*, available at *~bishop/ecs30/buggy.c* on the CSIF, is to read in up to 1023 lines of input, the maximum input line length being 1023 characters. It will then print the lines in reverse order. But it doesn't; it crashes. Please identify the problem, and say how to fix it. Give the specific line(s) where the problem arises, and the code that you would fix it with.

   Hint: The fixed program has to do the same thing as the buggy program is supposed to do, so you *must* print the lines in reverse order without changing any characters, you *must* allocate space for the lines (that is, you cannot eliminate the `malloc`), and you *must* free the space allocated for each line before exiting.