Extra Credit 3

Due: June 6, 2024; late due date is June 11 **Points:** 50

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 four error conditions.

1. If the argument is not an integer, print this error message on the standard error: "`%s not an integer\n`" where "%s" is the name of the program (that is, `argv[0]`) and exit with exit code 1.
2. If the argument is a non-positive integer, print the following error message on the standard error: "`Argument must be a positive integer\n`" and exit with exit code 1.
3. 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): "`Usage:  %s number\n`" where "%s" is the name of the program (that is, `argv[0]`) and exit with exit code 1.
4. If the argument is a positive integer that is too large to be represented in the program, print this error message on the standard error: "`Magnitude of %s too big\n`" where "%s" is the argument `argv[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 */home/bishop/ex3/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 */home/bishop/ex3/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 */home/bishop/ex3/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.

The return value should be 0 on success and 1 if there is no argument, more than one argument, or the argument is not an integer.

Please call your file *fib.c*. Do *not* turn in *timeit.o*.

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:".