Programming Assignment I

Due on Wednesday, September 29th, 2021 @ 11:59pm

50 points

This is a warm-up simple programming assignment using only Unix system calls `fork()`, `wait()` and `exit()`.

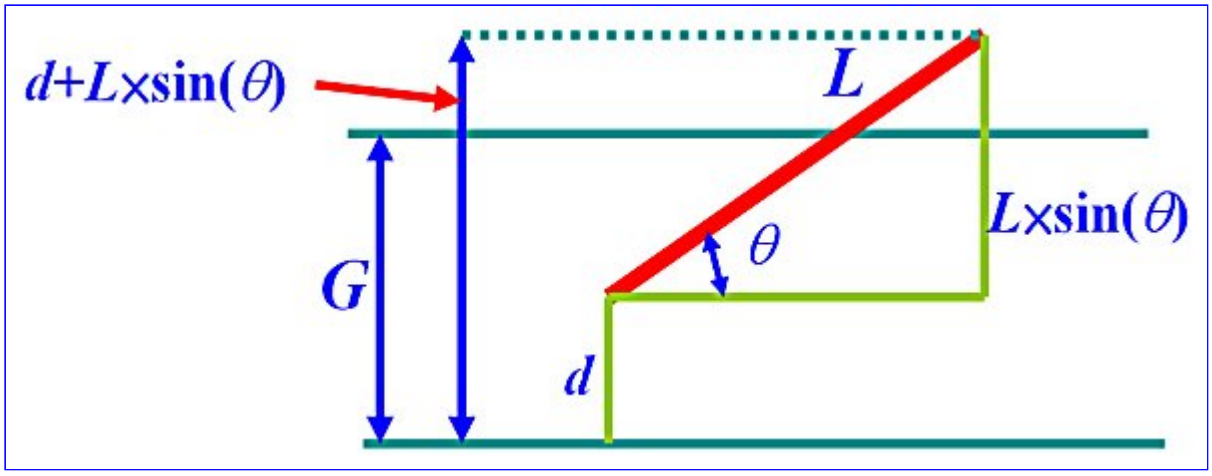
Running Four Independent Processes

Write a program that takes seven positive integers n, r, a, b, s, x and y from its commend line arguments, creates four child processes, waits for them to complete, and exits. The first process computes the n-th Fibonacci number f_n using recursion, the second process finds the solution of Buffon's needle problem by throwing a needle r times, the third process computes the area of an ellipse, and the fourth process simulates a simple pinball game. Here are the details:

1. The n-th Fibonacci number is defined recursively as follows:

   $f_1 = f_2 = 1$
   $f_n = f_{n-1} + f_{n-2}$ if $n > 2$

   Use **THIS** recursive formula to compute the n-th Fibonacci number.
2. The **Buffon's Needle** Problem. The problem was first stated by the French naturalist and mathematician George-Louis Leclerc, Comte de Buffon in 1733, and reproduced with solution by Buffon in 1777. Suppose the floor is divided into infinite number of parallel lines with a constant gap G. If we throw a needle of length L to the floor randomly, what is the probability of the needle crossing a dividing line? The answer is $(2/\pi)*(L/G)$, where π is 3.1415926.... **Use `acos(-1.0)` to obtain the value of π.**



We may use a program to simulate this needle throwing process. For simplicity, let $L = G = 1$.

We need two random numbers:

1. d, a random number in [0,1), represents the distance from one (fixed) tip of the needle to the lower dividing line.
2. a, a random number in [0,2*π), represents the angle between the needle and a dividing line.

Thus, if $d + L \times \sin(a)$ is less than 0 or larger than G, the needle crosses a dividing line.

In this way, your program loops r times. In each iteration, your program uses the above formula to throw a needle, and checks whether the needle crosses a dividing line. If the needle crosses a dividing line t times, t/r is an approximation of the exact probability. In fact, if r is very large, the simulated result would be very close to the exact result. In our case, since $L = G = 1$, the result for large r should be close to $2/\pi$ = 0.63661..... **Use `acos(-1.0)` to obtain the value of π.**

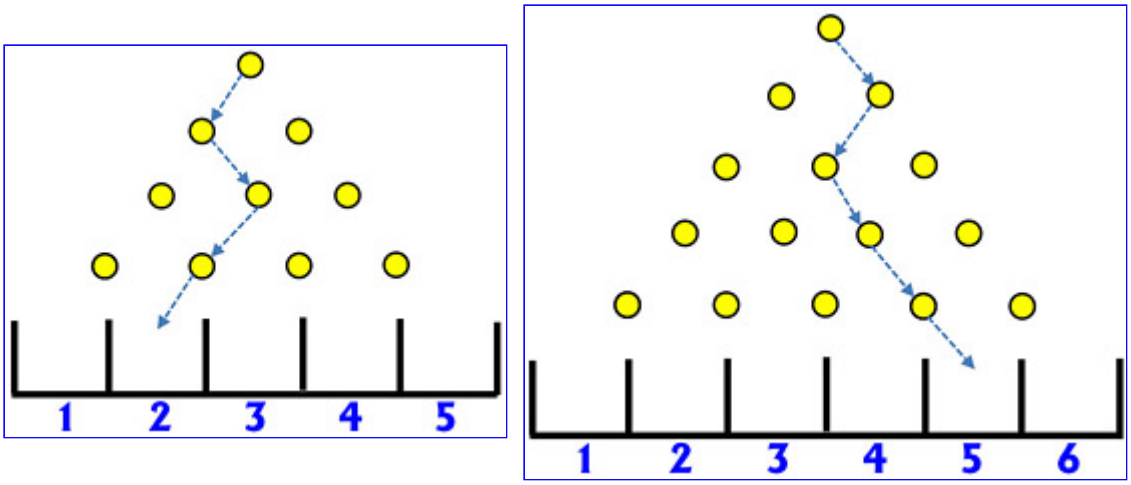
3. The area of ellipse $x^2/a^2 + y^2/b^2 = 1$ is πab, where a is the length of semi-major axis, b is the length of semi-minor axis, and π = 3.1415926.... Because of symmetry, the area of ellipse $x^2/a^2 + y^2/b^2 = 1$ is four times of the its area in the first quadrant where $x \geq 0$ and $y \geq 0$.

   If we randomly pick s points in the rectangle bounded by the x-axis, y-axis, the vertical line of $x = a$, and the horizontal line $y = b$, and find out t of them being in the area of the given ellipse (*i.e.*, $x^2/a^2 + y^2/b^2 \leq 1$), then the ratio t/s suggests that the area of the given ellipse in the first quadrant is approximately t/s of that of the rectangle with length a and height b (*i.e.*, $(t/s)ab$). Therefore, $((t/s)ab)\times 4$ should be close to πab. Here is what your program must do.

   1. Generate two random numbers x and y where $0 \leq x < a$; and $0 \leq y < b$. This (x,y) represents a point in the rectangle.
   2. If $x^2/a^2 + y^2/b^2 \leq 1$, (x,y) is in the given ellipse.
   3. Let the total number of points in the given ellipse be t.
   4. Then, $((t/s)ab)\times 4$ should be close to πab (*i.e.*, $(t/s)\times 4$) being approximately π), the desired result. This is especially true for very large s.

   Use `acos(-1.0)` to obtain the value of π.

4. Let us use random numbers to simulate a very **simple pinball game**. At the bottom of this game has a number of equal size bins, numbered as 1, 2, 3, ..., x, where x is an input positive integer larger than 1. Above these bins are rows of pins as shown in the diagrams below. If there are x bins, then the last row has x-1 pins as shown in the diagrams above. Above this there is a row of x-2 pins, etc. Continue this process, the top row has only 1 pin. In this way, we have exactly x-1 rows of pins.



Suppose we drop a ball vertically to the top pin. In the idealized case, when the ball hits each pin, it may go left or right with equal probability (*i.e.*, 0.5). In this way, when the ball hits the top bin, it goes either to the left or to the right. Also assume that when a ball is falling from the row above this row of pins it will always hit either the left pin or the right pin below it. As a result, a ball can only go left or right until it

finally falls into a bin. In the left diagram above, thr balls goes to left, right, left and left, and finally falls into bin 2. In the right6 diagram, the ball goes right, left, right, right and right, and finally falls into bin 5. Here is what your program must do:

1. Get x, the number of bins, and y, the number of balls to be dropped, from command line arguments.
2. Drop y balls as discussed earlier and tally the number of balls that end up in each bin.
3. Print out the number of balls in each bin, the percentage of each bin, and a histogram.

Write a program `prog1.c` in C to perform the following tasks

1. It reads in six command line arguments and converts them to six integers. This means `prog1.c` is run with the following command line

   ```
   ./prog1 n r a b s x y
   ```

   where `n`, `r`, `a`, `b`, `s`, `x` and `y` are seven positive integers.
2. Then, the main program forks four child processes, waits for their completion, and exits.
3. The first child process computes the n-th Fibonacci number f_n with recursion. (Yes, you must use recursion to kill time!) This process does its task in the following order:
   1. Prints the value of `n`
   2. Uses recursion to compute f_n
   3. Prints the result.

   Use **`long`** for computation.
4. The second child process simulates throwing a needle r times and estimates an approximation of the exact probability. This process does this simulation in the following order:
   1. Prints the value of `r`
   2. Iterates r times. Note that $L = G = 1$.
   3. Prints the computed result.

   Since `rand()` only generates random integers in the range from 0 to `RAND_MAX`, you should convert an integer random number to a real one in the range of 0 and 1 using `(float rand())/RAND_MAX`. If necessary, you may scale this real random number to other range.
5. The third child process estimates the area of ellipse $x^2/a^2 + y^2/b^2 = 1$. This process does the following:
   1. Prints the value of `s`, `a` and `b`.
   2. Iterates s times and count the number of points in the area of the ellipse in the first quadrant. The estimated area is scaled four times larger to obtain the area of the given ellipse. **You have to scale the obtained random number in the range of [0,1) to [0, a) and [0, b) by multiplying it with a and b, respectively.**
   3. Prints the computed area.
6. The fourth child process simulates dropping y balls into one of the x bins. This process does the following:
   1. Prints the values of x and y.
   2. Iterates y times and counts the number of balls that will eventually fall into each bin.
   3. Prints the results and a histogram.

You must use recursion to compute f_n. Moreover, the parent and its four child processes must run concurrently. Violating these rules you will receive zero point for this programming assignment.

Input and Output

The input to your program should be taken from command line arguments. Your source program must be named as `prog1.c`. The command line looks like the following:

```
./prog1  n r a b s x y
```

Here, `n`, `r`, `a`, `b`, `s`, `x` and `y` are seven positive integers. There are always seven arguments and they are always correct. Moreover, `r` , `s` and `y` are usually very large, say 1,000,000 or even larger.

Suppose the command line is

```
./prog1  10 100000 6 2 200000 6 3000000
```

Then, your program should (1) compute the 10-th Fibonacci number, (2) simulate throwing a needle 100000 times, (3) use 200000 points to estimate the area of ellipse $x^2/6^2 + y^2/2^2 = 1$, and (4) drop 3000000 balls and count the number of balls that will fall into each one of the 6 bins.

The Fibonacci process should have its output as follows:

```
Fibonacci Process Started
Input Number 10
Fibonacci Number f(10) is 55
Fibonacci Process Exits
```

All output from this process starts on column 4 (*i.e.*, 3 leading spaces). Since the computed Fibonacci number may be very large, use `%ld` to print the computed result.

The Buffon's Needle process should have its output as follows:

```
Buffon's Needle Process Started
Input Number 100000
Estimated Probability is 0.63607
Buffon's Needle Process Exits
```

Since the output value is always in the range of 0 and 1, use 5 positions after the decimal point for printing. All output from this process starts on column 7 (*i.e.*, 6 leading spaces).

The ellipse area (*i.e.* fourth) process should have its output as follows:

```
Ellipse Area Process Started
Total random Number Pairs 200000
Semi-Major Axis Length 6
Semi-Minor Axis Length 2
Total Hits 156899
Estimated Area is 37.65576
Actual Area is 37.69911
Ellipse Area Process Exits
```

All output from this process starts on column 10 (*i.e.*, 9 leading spaces).

The simple pinball game process should have its output as follows:

```
Simple Pinball Process Started
Number of Bins 6
Number of Ball Droppings 3000000
  1-(  94188)-( 3.14%)|*****
  2-( 468531)-(15.62%)|************************
  3-( 937169)-(31.24%)|************************************************
  4-( 937427)-(31.25%)|************************************************
  5-( 468535)-(15.62%)|************************
```

```
  6-(  94150)-( 3.14%)|*****
Simple Pinball Process Exits
```

In the output above, the first column shows the bin numbers printed with `%3d`; the second has the number of balls falling to each bin, because the number of balls can be very large these numbers are printed using `%7d` or `%7ld` depending on the data type you choose to use; and the "percentage" column uses `%5.2f` as the numbers are percentages. The histogram requires some explanation. The length (or the number of asterisks) of each horizonal bar is scaled so that the largest percentage is printed using 50 positions (*i.e.*, 50 asterisks). Note that it is possible that if a percentage is so small, there may not be any asterisks shown as after scaling the number of asterisks to be used is less than 1. All output from this process starts on column 1 (*i.e.*, leading spaces).

Note that **the results of Buffon's Needle problem, the ellipse area problem and the pinball problem may be slightly different from the output shown here even with the same *r*, *s* and *y* because random numbers are used.**

The main program should print the following output:

```
Main Process Started
Fibonacci Input            = 10
Buffon's Needle Iterations = 100000
Total random Number Pairs  = 200000
Semi-Major Axis Length     = 6
Semi-Minor Axis Length     = 2
Number of Bins             = 6
Number of Ball Droppings   = 3000000

Fibonacci Process Created
Buffon's Needle Process Created
Ellipse Area Process Created
Pinball Process Created
Main Process Waits
Main Process Exits
```

All output line from the main process should start on column 1 (*i.e.*, no leading spaces). Note that the main process and the pinball process all have their output on column 1.

It is very important to remember the following. All processes **must** run concurrently and may print their output lines in an unpredictable order. As a result, the output lines from a process may mix with output lines from other processes. This is the reason that proper indentation is required to know who prints what. **Also make sure that each line from a process will not contain the output from different processes.**

It is very important to remember the following.

> **All processes must run concurrently and may print their output lines in an unpredictable order. As a result, the output lines from a process may mix with output lines from other processes. This is the reason that proper indentation is required to know who prints what. Also make sure that each line will not contain the output from different processes.**

Submission Guidelines

General Rules

1. All programs must be written in C.
2. Use `Canvas` to submit your work as a zip file. This archive will contain both your program as well as your README (see below).
3. Your program should be named as `prog1.c`. Since Unix filename is case sensitive, `PROG1.c`, `prog1.C`, `pROG1.c`, etc are ***not*** acceptable.
4. We will use the following approach to compile and test your programs:

```
gcc  prog1.c -lm -o prog1                    <-- compile your program
./prog1 10 100000 6 2 200000 6 3000000  <-- test your program
```

   This procedure may be repeated a number of times with different input to see if your program works correctly.

> **You must use the above command lines to compile and run your program. Otherwise, our grader may not be able to grade your program, and you will risk a low or even a 0 score.**
>
> **Our grader uses the default install of C on the CS lab machines and remote services. Therefore, your program must compile and run in that environment and may not compile otherwise. If this happens, you will receive a 0 because your program does not compile.**

5. Your implementation should fulfill the program specification as stated. Any deviation from the specification will cause you to receive **zero** point.
6. A `README` file is always required.
7. **No late submission will be graded.**
8. **Programs submitted to wrong class and/or wrong section will not be graded.**

Compiling and Running Your Programs

This is about the way of compiling and running your program. If we cannot compile your program due to any syntax errors, wrong file names, etc, **we cannot test your program, and, as a result, you receive 0 point**. If your program compiles successfully but fails to run, **we cannot test your program, and, again, you receive 0 point**. **Therefore, before submitting your work, make sure your program can compile and run properly.**

1. **Not-compile programs receive 0 point.** By **not-compile**, I mean any reason that could cause an unsuccessful compilation, including missing files, incorrect filenames, syntax errors in your programs, and so on. Double check your files before you submit, since I will **not** change your program. Note again: Unix filenames are ***case sensitive***.
2. **Compile-but-not-run programs receive 0 point. Compile-but-not-run** usually means you have attempted to solve the problem to some degree but you failed to make it working properly.
3. **A meaningless or vague program receives 0 point even though it compiles successfully.** This usually means your program does not solve the problem but serves as a placeholder or template just making it to

compile and run.

Program Style and Documentation

This section is about program style and documentation.

1. For each file, the first piece should be a program header to identify yourself like this:

```
/* ---------------------------------------------------------- */
/* NAME : John Smith                          User ID: xxxxxxxx */
/* DUE DATE : mm/dd/yyyy                                       */
/* PROGRAM ASSIGNMENT #                                        */
/* FILE NAME : xxxx.yyyy.zzzz (your unix file name)            */
/* PROGRAM PURPOSE :                                           */
/*    A couple of lines describing your program briefly        */
/* ---------------------------------------------------------- */
```

   Here, **User ID** is the one you use to login. It is ***not*** your social security number nor your M number.

   For each function in your program, include a simple description like this:

```
/* ---------------------------------------------------------- */
/* FUNCTION  xxyyzz : (function name)                          */
/*     the purpose of this function                            */
/* PARAMETER USAGE :                                           */
/*    a list of all parameters and their meaning               */
/* FUNCTION CALLED :                                           */
/*    a list of functions that are called by this one          */
/* ---------------------------------------------------------- */
```

2. Your programs must contain enough concise and to-the-point comments. Do not write a novel!
3. Your program should have good indentation.
4. Do not use global variables!

Program Specification

Your program must follow exactly the requirements of this programming assignment. Otherwise, you receive 0 point even though your program runs and produces correct output.
The following is a list of potential problems.

1. Your program does not use the indicated algorithms/methods to solve this problem.
2. Your program does not follow the structure given in the specification. For example, your program is not divided into functions and files, etc when the specification says you should.
3. Any other significant violation of the given program specification.
4. **Incorrect output format.** This will cost you some points depending on how serious the violations are. The grader will make a decision. Hence, carefully check your program output against the required one.

Program Correctness

If your program compiles and runs, we will check its correctness. We will run your program with at least two sets of input data, one posted on this programming assignment page (the public one) and the other prepared by the grader (the private ones). You program must deliver correct results for **all** data sets to be considered as a correct one. Depending on the seriousness of the problem(s), significant deduction may be applied. For example, if your program delivers all wrong results for the public data set, you receive 0 point for that component.

The `README` File

A file named `README` is required to answer the following questions:

1. **Question:** Draw a diagram showing the parent-child relationship if the following program is run with command line argument 4. How many processes are created? Explain step-by-step how these processes are created, especially who is created by whom.

```
void main(int argc, char **argv)
{
     int i, n = atoi(argv[1]);

     for (i = 1; i < n; i++)
          if (fork())
               break;
     printf("Process %ld with parent %ld\n", getpid(), getppid());
     sleep(1);
}
```

2. **Question:** Draw a diagram showing the parent-child relationship if the following program is run with command line argument 4. How many processes are created? Explain step-by-step how these processes are created, especially who is created by whom.

```
void main(int argc, char **argv)
{
     int i, n = atoi(argv[1]);

     for (i = 0; i < n; i++)
          if (fork() <= 0)
               break;
     printf("Process %ld with parent %ld\n", getpid(), getppid());
     sleep(1);
}
```

3. **Question:** Draw a diagram showing the parent-child relationship if the following program is run with command line argument 3. How many processes are created? Explain step-by-step how these processes are created, especially who is created by whom.

```
void main(int argc, char **argv)
{
     int i, n = atoi(argv[1]);

     for (i = 0; i < n; i++)
          if (fork() == -1)
               break;
     printf("Process %ld with parent %ld\n", getpid(), getppid());
     sleep(1);
}
```

4. **Question:** The histogram you obtained from the simple pinball game is always symmetric, even though the number of balls in each bin may be slightly different. However, if the histogram is significantly not symmetric, your program is definitely incorrect. Actually, this is a distribution you may have learned in your statistics and probability course. What is this distribution called? What is the reason you believe the histogram is the named distribution by you? Answer this question with a good logic reasoning. Without doing so (*e.g.*, *only writing done the answer with a vague reason*), you will lose point for this portion.

You should elaborate your answer and provide details. **When answering the above questions, make sure each answer starts with a new line and has the question number (*e.g.*, Question X:) clearly shown. Separate two answers with a blank line. Also make sure that the diagrams in your `README` file will PRINT correctly. Do not judge the results on screen.**

Note that the filename has to be `README` rather than `readme` or `Readme`. Note also that there is ***no*** filename extension, which means a filename such as `README.TXT` is ***NOT*** acceptable.

`README` must be a plain text file. We do not accept files produced by any word processor. Moreover, watch for very long lines. More precisely, limit the length of each line to no more than 80 characters with the Return/Enter key for line separation. Missing this file, submitting non-text file, file with long lines, or providing incorrect and/or vague answers will cost you many points. Suggestion: Use a Unix text editor to prepare your `README` rather than a word processor.

Final Notes

1. Your submission should include two files, namely: `prog1.c` and `README`. Please note the way of spelling filenames.
2. **Always start early, because I will not grant any extension if your home machine, network connection, your phone line or the department machines crash in the last minute, etc.**
3. **Since the rules are all clearly stated, no leniency will be given and none of the above conditions is negotiable. So, if you have anything in doubt, please ask for clarification.**
4. **Click here to see how your program will be graded.**