Enter the C

CMS 230, Fall 2016

Due Thursday, September 1, 11:59:59 PM

Description

The following problems will give you practice writing basic programs in C, as well as reviewing some fundamental programming concepts, like loops and recursion.

Please submit a single .zip archive containing one .c source file for each problem along with a `Makefile` that builds all of the source files into separate executables. The five problem files must be named `problem1.c`, `problem2.c`, etc.

Rubric

I will grade your project using the included grading script, `grade`. To run it, type `grade` at the command prompt. It will first build your project using `make`, then run each executable and evaluate its output. Each problem in this project has an unambiguously correct solution — if your code prints the same solution, you will get full credit.

Make sure your code compiles before you submit it!

Makefile and Compilation (20 points)

- No Makefile – 0 points
- Makefile exists, but contains an error that prevents it from running – 5 points
- Makefile exists and runs correctly – 20 points

Problems (5 problems for 14 points each)

- Program compiles successfully – 2 points
- Program produces correct output when run – 10 points

The Problems

The Basics

Use a loop to print all the positive integers less than 200 that are divisible by 7.

Area of a Circle

Write a simple program that calculates the area of a circle with radius 5.0, then prints it using

```
printf("%f\n", area);
```

For the value of π, use the constant `M_PI` defined in the header `math.h`.

And Then You Slide Down the Flagpole and Fireworks Explode

At the end of most levels of the original *Super Mario Brothers*, Mario jumps up a staircase like this:

```
       ##
      ###
     ####
    #####
   ######
  #######
 ########
#########
```

Write a C program that can produce such a staircase, with the height controlled by a variable. Print a staircase that is 12 steps high.

Hint: if the height is h, the top level has $h-1$ spaces followed by two `#` characters. The next row has $h-2$ spaces followed by three `#` characters, and so forth.

Everyone Needs a Hobby

I enjoy building stone ziggurats in my backyard. To build an N-level ziggurat, I first build an $N \times N$ square of stones on the ground. Then I build an $N-1 \times N-1$ square of stones for the second level, then an $N-2 \times N-2$ square of stones for the third level, and so forth, until I finally place a single stone on the top level.

Write a **recursive** C program that calculates the number of stones in a ten-level ziggurat.

Hint: The number of stones in a ten-level ziggurat is the number in a nine-level ziggurat plus 10^2. In general,

$$stones(N) = stones(N-1) + N^2$$

Binet's Formula and Linking with Libraries

Recall the famous Fibonacci sequence, where each term is the sum of the two previous terms:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Binet's Formula (named after the mathematician Jacques Philippe Marie Binet) is an explicit formula for finding terms in the Fibonacci sequence. The nth Fibonacci number, F_n, is given by

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

The special number

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618033\ldots$$

is the famous *golden ratio*, the most aesthetically pleasing of all proportions.

Write a C program that uses Binet's Formula to calculate the first 20 Fibonacci numbers. Use `sqrt` and `pow` to perform the calculations; both functions are defined in `math.h`. Look up both commands to see how they're used.

To use the `pow` function, you'll need to *link* your code with the math library. *Libraries* are pre-compiled collections of useful routines. The linking process merges this pre-compiled code into your executable.

By convention, all libraries start with the prefix `lib-`, followed by the name of the library. The math library is called `libm.a` and lives in a subdirectory of `/usr/lib`.

The `-l` flag instructs `gcc` to link in a library. The appropriate command is

```
gcc -o problem5 problem5.c -lm
```

`gcc` processes the `-l` flag by taking the rest of the flag (in this case, the letter `m`), which it interprets as the name of the library. It uses that name to generate the name of the library file in standard format: `libm.a`. The linker then looks up `libm.a` in the standard directory and links its code into the executable.