COSC 2P91 – Assignment 1 – Let's get this party started...

Due: Thursday Feb 5th @5pm

Objective:
To gain some basic practice with C, including variables, functions, and compiling and execution.

This assignment is broken up into four tasks; each of which is to be written as a separate program. Outside of being allowed to use `stdlib` and `stdio`, you are to implement everything yourself.

Task 1:
Calculating factorial or Fibonacci numbers.
Filename: *calculate.c*

- Decision between factorial or Fibonacci, as well as number, are based solely on command-line arguments. e.g.:

```
./calculate factorial X
```

or

```
./calculate fibonacci X
```

- If the parameters do not follow that convention, print the following error message and exit:

```
Invalid Parameters
```

- Don't worry about negative values; you're just verifying that two parameters were provided, and that the first was `factorial` or `fibonacci`
- On both 32 and 64 bit systems, ensure it will work on at least domain [1..20] for factorial and [1..75] for fibonacci (both within a reasonable time)
- Output for successful factorial must be in style of:

```
5!=120
```

- Output for successful Fibonacci must be in style of:

```
fib(4)=3
```

- Each output is terminated by a newline, but has no additional blank spaces afterwards

Task 2:
Calculating (prime) factors.
Filename: *factorize.c*

Interaction is driven by prompts while executing:

```
1. Factors
2. Prime Factors
0. Stop
```

For either factor choice, next line is:
`Number to factorize:` *(there's a space after the colon)*
After number is entered, a single line is printed, containing the factors (for option 1) or prime factors (for option 2) in ascending sequence, separated by tabs (with no tab before the first number).
Refer to the end for a sample execution.

- The program simply ends when the user chooses to stop – i.e. nothing is printed
- Don't worry about negative values, zero, decimals, etc.
- To be clear, (depending on the user choice) you're producing either all of the factors of the specified target, or the prime factors (including repetition) of said target

Task 3:
Fraction reduction.
Filename: *fractions.c*

- Based on keyboard input
- Repeats until user chooses to quit
- Prompt is as follows:

`Enter fraction (0 to end):` *(there's a space after the colon)*

- Fraction is reduced, and then simplified fraction is printed
- Entered fraction will be either improper fraction, or just a whole number (i.e. no denominator)
- Output will always consist solely of the reduced fraction (expressed as improper fraction)
- Don't worry about negative values, or numbers equivalent to zero

e.g.

```
Enter fraction (0 to end): 18/12
3/2
Enter fraction (0 to end): 2/6
1/3
Enter fraction (0 to end): 24
24/1
Enter fraction (0 to end): 0
```

Task 4:
Towers of Hanoi
Filename: *hanoi.c*

- Solve recursively
- Solve in optimal number of moves
  - Refer to the 'Tips' section
- Program must accept a single number as a command-line argument to indicate the number of discs on the starting peg

e.g.

```
./hanoi 3
```

- Discs always start on the first (leftmost) peg, and end on the last (rightmost) peg
- All output must match precise style of sample execution (see last page)
- Use an array (or arrays)

Important:

- Remember that you're implementing these tasks entirely by yourself, so stick to just `stdio` and `stdlib` for `includes`
- All filenames are mandatory
  - Compiled names must be the same as the source files (except minus the `.c`)
- All filenames and sample output are case-sensitive
- All sample output requires specific whitespace/newlines
- Put all four source files into the same folder when submitting
- Don't include any additional files
- For full marks, use **only** *local* variables – i.e. no globals at all!

Tips:

- For a reminder on how to use gcc, you could use google, but `man` is also good
- Include at least basic commenting and reasonable programming conventions

    - But including at least your name and student number at the beginning of each file is mandatory
    - You're expected to know by now when it's appropriate to abstract behaviours into separate functions, etc.
- Don't intentionally obfuscate your code

Submission:
Both electronic and physical submission are required.
For electronic, upload all files to sandcastle, SSH in, and run the `submit2p91` script.
For physical, print out all source files, as well as sample executions; attach a departmental cover sheet and deposit into the appropriate dropbox.
Due dates apply to both physical *and* electronic submission.
Refer to the department's (and university's) pages on academic integrity and plagiarism.

Factorization sample execution:

```
1. Factors
2. Prime Factors
0. Stop
1
Number to factorize: 18
1    2    3    6    9    18
1. Factors
2. Prime Factors
0. Stop
2
Number to factorize: 18
2    3    3
1. Factors
2. Prime Factors
0. Stop
0
```

Bolded values are user-input, rather than actual output.

Towers of Hanoi sample executions:

```
$ ./hanoi 2
Initial
 [|]     |      |
[ | ]    |      |
=================

Move 1 from A to B
  |      |      |
[ | ]   [|]     |
=================

Move 1 from A to C
  |      |      |
  |     [|]   [ | ]
=================
```

```
Move 1 from B to C
  |      |     [|]
  |      |    [ | ]
=================

3 moves

$ ./hanoi 3
Initial
  [|]       |         |
 [ | ]      |         |
[  |  ]     |         |
=======================

Move 1 from A to C
    |        |         |
 [ | ]      |         |
[  |  ]     |       [|]
=======================

Move 1 from A to B
    |        |         |
    |        |         |
[   |  ]   [ | ]     [|]
=======================

Move 1 from C to B
    |        |         |
    |       [|]        |
[   |  ]   [ | ]      |
=======================

Move 1 from A to C
    |        |         |
    |       [|]        |
    |      [ | ]   [   |  ]
=======================

Move 1 from B to A
    |        |         |
    |        |         |
   [|]     [ | ]   [   |  ]
=======================

Move 1 from B to C
    |        |         |
    |        |      [ | ]
   [|]       |     [   |  ]
=======================
```

```
Move 1 from A to C
    |         |        [|]
    |         |       [  |  ]
    |         |     [   |   ]
=======================

7 moves
```

Note that the base is appropriately sized, according to the problem size, and that the posts have a single space separating their maximum widths (i.e. even if you were to copy the largest disc and place one on each post, they still wouldn't touch).

The basic recursive algorithm is simple:

move(*n*, *from*, *to*)→

if *n*=1	move disc from *from* to *to*
else	move *n*-1 from *from* to *other*
	move 1 from *from* to *to*
	move *n*-1 from *other* to *to*