APS 105 — Computer Fundamentals

Lab #4: Loops and Functions

Winter 2022

You must use `examify.ca` to electronically submit your solution by 11:59 pm on **Saturday, February 12, 2022**.

Objective

The goal of this laboratory is to practice the material on loops and functions.

Preparation

Read through this entire document carefully, and do the work to create the programs that are described below. You are encouraged to:

- Read the class bulletin board on Piazza, to see if others had similar problems. If you do not see anything helpful there, ask a question. Do not ask for, or ever give a full or partial solution to the lab assignment.
- During the lab, ask for assistance from the lab TA assigned to you in your lab subsection.

Start and follow the suggested steps below:

- Write your pseudocode that is in a plain language describing and planning the steps in solving the problem at hand.
- Follow up your plan by writing a code on paper, i.e., stay away from typing on your computer at this time.
- Try to review your code and identify any mistakes that you can identify, i.e., walk through your code.
- Use your computer, type and run your code.
- Run the test cases provided to ensure proper functionality of your code.

Part 1 — Pascal's Triangle

Implement a C function, named `triangle`, that outputs Pascal's triangle. An example is provided below.

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
```

```
      1     10    45    120   210   252   210   120   45    10    1
   1     11    55    165   330   462   462   330   165   55    11    1
1     12    66    220   495   792   924   792   495   220   66    12    1
```

In general, the Pascal's triangle can be represented as:

$$
\begin{gathered}
0\mathbf{C}0 \\
1\mathbf{C}1 \quad 1\mathbf{C}0 \\
2\mathbf{C}2 \quad 2\mathbf{C}1 \quad 2\mathbf{C}0 \\
3\mathbf{C}3 \quad 3\mathbf{C}2 \quad 3\mathbf{C}1 \quad 3\mathbf{C}0 \\
4\mathbf{C}4 \quad 4\mathbf{C}3 \quad 4\mathbf{C}2 \quad 4\mathbf{C}1 \quad 4\mathbf{C}0 \\
\ldots
\end{gathered}
$$

where $n\mathbf{C}r$ represents how many ways there are to choose r from n, not counting duplicates.

In mathematics, it is usually presented as $\binom{n}{r}$. The formula used to calculate $n\mathbf{C}r$ can be written as:

$$n\mathbf{C}r = \frac{n!}{r!(n-r)!},$$

where $n!$ is the factorial of n.

The function `triangle`:

- is called by value where exactly one parameter (the number of rows of the Pascal's triangle) is passed to it.
- returns `void` (i.e., no value).
- prints out the Pascal's triangle to the standard output.
- Employs two other functions, namely:
  - `int choose(int n, int r);`
    That chooses r from n
  - `int factorial(int n);`
    That calculates factorial of n.

Note: You are allowed — in fact encouraged — to design and use any other functions as needed.

A `main` function will be supplied to you in this part of the laboratory.

An example of the required output from your program is as follows:

```
Enter the number of rows: 0
Enter the number of rows: 1
1
Enter the number of rows: 2
   1
1     1
Enter the number of rows: 3
      1
   1     1
```

```
1     2     1
Enter the number of rows: 4
         1
      1     1
   1     2     1
1     3     3     1
Enter the number of rows: 5
            1
         1     1
      1     2     1
   1     3     3     1
1     4     6     4     1
Enter the number of rows: 6
               1
            1     1
         1     2     1
      1     3     3     1
   1     4     6     4     1
1     5     10    10    5     1
Enter the number of rows: 7
                  1
               1     1
            1     2     1
         1     3     3     1
      1     4     6     4     1
   1     5     10    10    5     1
1     6     15    20    15    6     1
Enter the number of rows: 8
                     1
                  1     1
               1     2     1
            1     3     3     1
         1     4     6     4     1
      1     5     10    10    5     1
   1     6     15    20    15    6     1
1     7     21    35    35    21    7     1
Enter the number of rows: -1
```

Note:

- The number of the rows is limited to integers between **0** and **13** inclusive.
- If **0** is entered by the user, no triangle will be printed.
- If a negative integer is entered by the user, the program terminates.
- The number displayed in the last column **must** always be displayed at the start of the line.
- Carefully calculate the spaces used between the numbers so that the numbers are properly aligned across rows.

Part 2 — Erdős-Woods Numbers

In this part, you are asked to write a complete C program that finds an *Erdős-Woods number*.

In number theory, a positive integer k is said to be an *Erdős–Woods number* if there exists a positive integer a such that each of the consecutive integers $a + i$ for $0 < i < k$ shares at least one prime factor with either a or $a + k$. In other words, if for a k there is an a such that each evaluation of $\gcd(a, a + i) > 1$ or $\gcd(a + k, a + i) > 1$ returns true, then k is an Erdős–Woods number.

For example, one a for $k = 16$ is 2184. 2184 is $2^3 \times 3 \times 7 \times 13$, while $2184 + 16 = 2200 = 2^3 \times 5^2 \times 11$. We can easily verify that all the integers between 2185 and 2199 share at least a prime factor with either 2184 or 2200.

Your C program should begin by prompting the user to enter four integers:

- The number to start searching for k. It needs to be greater than 2. Your program will start searching for Erdős-Woods number from this starting point (inclusive).
- The number to stop searching for k (inclusive). It needs to be greater than the starting point above searching for k.
- The number to start searching for a (inclusive). It needs to be positive (greater than 0).
- The number to stop searching for a (inclusive). It needs to be greater than the starting point above searching for a.

When the stated requirements for these integers are not satisfied (for example, the starting point searching for k is not greater than 2), the user will be repeatedly prompted to enter new values for the same integer until the requirements are satisfied.

Once valid values for these four integers have been entered, your program should start from the starting point of searching for k, and use the definition of Erdős–Woods number to see if k is an Erdős–Woods number. To save some computation time, your program only needs to search in the range specified by the user for a.

Your C program needs to define and use at least one function that computes the greatest common divisor between two integers, a and b:

```
int gcd(int a, int b);
```

An example run of your program is as follows:

```
Enter the number to start searching for k (> 2, inclusive): 1
Enter the number to start searching for k (> 2, inclusive): 2
Enter the number to start searching for k (> 2, inclusive): 3
Enter the number to stop searching for k (inclusive): 2
Enter the number to stop searching for k (inclusive): 1
Enter the number to stop searching for k (inclusive): 20
Enter the number to start searching for a (> 0, inclusive): 0
Enter the number to start searching for a (> 0, inclusive): 1
Enter the number to stop searching for a (inclusive): 10000
Trying k = 3...
Trying k = 4...
Trying k = 5...
Trying k = 6...
```

```
Trying k = 7...
Trying k = 8...
Trying k = 9...
Trying k = 10...
Trying k = 11...
Trying k = 12...
Trying k = 13...
Trying k = 14...
Trying k = 15...
Trying k = 16...
Erdos-Woods number: 16
a = 2184
```

Another example run of your program is as follows:

```
Enter the number to start searching for k (> 2, inclusive): 17
Enter the number to stop searching for k (inclusive): 30
Enter the number to start searching for a (> 0, inclusive): 3000000
Enter the number to stop searching for a (inclusive): 4000000
Trying k = 17...
Trying k = 18...
Trying k = 19...
Trying k = 20...
Trying k = 21...
Trying k = 22...
Erdos-Woods number: 22
a = 3521210
```

A third example run of your program is as follows:

```
Enter the number to start searching for k (> 2, inclusive): 3
Enter the number to stop searching for k (inclusive): 22
Enter the number to start searching for a (> 0, inclusive): 3000000
Enter the number to stop searching for a (inclusive): 4000000
Trying k = 3...
Trying k = 4...
Trying k = 5...
Trying k = 6...
Trying k = 7...
Trying k = 8...
Trying k = 9...
Trying k = 10...
Trying k = 11...
Trying k = 12...
Trying k = 13...
Trying k = 14...
Trying k = 15...
Trying k = 16...
Erdos-Woods number: 16
a = 3000800
```

A final example run of your program is as follows, when no Erdős–Woods number is found:

```
Enter the number to start searching for k (> 2, inclusive): 3
Enter the number to stop searching for k (inclusive): 15
Enter the number to start searching for a (> 0, inclusive): 1
Enter the number to stop searching for a (inclusive): 3000
Trying k = 3...
Trying k = 4...
Trying k = 5...
Trying k = 6...
Trying k = 7...
Trying k = 8...
Trying k = 9...
Trying k = 10...
Trying k = 11...
Trying k = 12...
Trying k = 13...
Trying k = 14...
Trying k = 15...
Erdos-Woods number not found.
```

Hint: Since *examify.ca* has a timeout mechanism when running your code on the test cases, try to use an efficient algorithm for computing the greatest common divisor of two numbers, such as the Euclidean algorithm.

Marking

This lab will be marked out of 10, with 4 marks allocated to each part. 2 marks will be assigned as an evaluation of your coding style (introduced and discussed at length in Plenary Lecture 2). The coding style will be marked in your lab section by your lab TA. Here are the basic requirements for your code to earn the coding style marks:

- Use meaningful identifiers for the variables in your C program. Use either snake case or camel case, but be consistent throughout your entire C program.
- Use named constants when needed, rather than literal constants.
- Use consistent indentation throughout your entire C program.
- Use comments at the beginning of your program before the `main()` function. You may also add comments throughout your program as necessary, but they are not strictly enforced.
- Define and call functions as required in this laboratory.

The automarker may use multiple test cases for each part, and if this is the case, the marks for this part will be evenly split among the test cases. The test cases used for marking may or may not be the same as the test cases that are made available to you. The deadline of **(11:59 p.m. on Saturday, February 12, 2022)** is strictly enforced, so avoid last minute submissions.

Stay Safe and Good Luck!