National University of Singapore
School of Continuing and Lifelong Education

TIC1001: Introduction to Computing and Programming I
Semester I, 2019/2020

Problem Set 3 Functional Abstraction

Release date: 5th September 2019, 8:00 pm
Due: 13th October 2019, 6:00 pm

There are several sub-questions in each task. Make sure you answer every one of them in the **template file** provided before finalizing your submission on Coursemology.

Task 1: Pascal's Triangle (5 marks)

In mathematics, Pascal's triangle is a triangular array of the binomial coefficients. In much of the Western world, it is named after French mathematician Blaise Pascal, although other mathematicians studied it centuries before him in India, Persia (Iran), China, Germany, and Italy.

To construct a Pascal's triangle, we begin by numbering the first row as $n = 0$. The second row would be $n = 1$, the third row would be $n = 2$, etc. Each row contains $n + 1$ elements with the first element numbered as $k = 0$. the second element $k = 1$, and so on until the last element, which is $k = n$.

On row 0, we simply write the number one. To derive the elements for the subsequent rows, we use this algorithm—add the number directly above and to the left with the number directly above and to the right of a specific element to find the new number for that element. If either the number to the left or right is not present, we will substitute a zero in its place.

$$\begin{array}{c}
1 \\
1 \quad 1 \\
1 \quad 2 \quad 1 \\
1 \quad 3 \quad 3 \quad 1 \\
1 \quad 4 \quad 6 \quad 4 \quad 1 \\
1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1
\end{array}$$

Mathematically, the number of each element is calculated with the same formula for calculating combinations:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

For example, 5 choose 3. i.e., the sixth row ($n = 5$) fourth element ($k = 3$) in the figure would be:

$$\binom{5}{3} = \frac{5!}{3!(5-3)!} = \frac{5!}{3! \times 2!} = \frac{5 \times 4 \times 3 \times 2 \times 1}{(3 \times 2 \times 1) \times (2 \times 1)} = \frac{5 \times 4}{2 \times 1} = 10$$

Note that you can simplify the computation by cancelling out $(3 \times 2 \times 1)$ in the computation above.

Questions

1. Implement the function `int factorial(int n)` that takes in a positive integer n and returns n factorial.
2. What happens to your function when $n \geq 13$? Explain why.
3. Implement the function `int choose(int n, int k)` that takes as inputs two integers, where $0 \leq n < 30$ and $0 \leq k \leq n$, and returns the value of $\binom{n}{k}$. For example, `choose(13, 2)` returns 78.
4. Are you able to make use of the `factorial` function in `choose`? Explain why.
5. Finally, implement the function `void pascal_triangle(int row)` that takes one integer as input and **prints** (i.e., using `printf` or `cout`) the requested row of the Pascal's Triangle with one space between the numbers . For example, `pascal_triangle(15)` prints:

   ```
   1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
   ```

Assumption: Your functions are only required to work for $0 \leq n < 30$ and $0 \leq row < 30$.

Hint: Take note of the size limits of the types.

Hint 2: Because `pascal_triangle` is required to **print** the output instead of returning as a value, do remove all debugging `printf` before submitting on Coursemology, lest they interfere with the output.

Task 2: Calendar (10 marks)

Given the month and the year, you are required to print the Georgian Calendar[1] for that month. For example, the calendar for the month of September 2017 is:

```
 S  M  T  W  T  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

The solution to this problem can be rather complex, so we will use functional abstraction to break it down into smaller parts. You are advised to implement and test each part (by defining a suitable `main` function) before moving on the the next part.

You are to implement the following functions:

1. `bool is_leap_year(int year)` to determine if a given year is leap.

   Note that not all years which are multiples of 4 are leap. Centuries are not leap unless they are multiples of 400.
2. `int days_in_month(int month, int year)` that takes the month (an integer from 1 to 12) and year, and returns number of days in the given month (an integer from 1 to 31).

   Note that you should use the `is_leap_year` function to determine if February has 28 or 29 days.

[1]This is the calendar that we currently use.

3. `int days_from_epoch(int day, int month, int year)` that computes and returns the number of days from the *epoch*, 1 January 1970[2], to the input date.

   For example. `days_from_epoch(10, 1, 1970)` returns 9 because it is nine days away from the epoch.

4. `int day_of_week(int day, int month, int year)` that returns the day of the week of the given input date. Days are represented by integers ranging from 0 for Sunday, to 6 for Saturday. Note that the epoch, 1 January 1970, falls on a Thursday.

5. Finally, `void display_month(int month, int year)` to print the specific month of the given year. The output should contain a header for the day of week `" S M T W T F S"` and the numbers should be right aligned under their respective day of the week.

   For example, the output for September 2017 is

```
  S  M  T  W  T  F  S
                 1  2
  3  4  5  6  7  8  9
 10 11 12 13 14 15 16
 17 18 19 20 21 22 23
 24 25 26 27 28 29 30
```

   and the output for August 1965 is

```
  S  M  T  W  T  F  S
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
 15 16 17 18 19 20 21
 22 23 24 25 26 27 28
 29 30 31
```

   Hint: you may use the conversion specifier `"%3d"` to print an integer right justified within 3 text spaces.

As this problem set is on functional abstraction, you should reuse these functions whenever possible.

[2]Unix systems count time based on the number of seconds elapsed since 1 January 1970. iPhones were known to crash if you set the date to epoch due to suspected integer underflow. Make sure your program doesn't!