NYU, Tandon School of Engineering
Extended Bridge to CS — Summer 2021

Homework #7
Due by Friday 8/27 11:59pm

Submission instructions:

1. For this assignment, you should turn in 5 files:
   - Two '.cpp' files, one for each question 1-2.
     Name your files 'YourNetID_hw7_q1.cpp', and 'YourNetID_hw7_q2.cpp'.
   - One '.pdf' file with your answers for questions 3-7.
     Each question should start on a new page!
     Name your file 'YourNetID_hw7_q3to7.pdf'

2. You must type all your solutions. We will take off points for submissions that are handwritten.

3. **You should submit your homework in the Gradescope system.**
   Note that when submitting the pdf file, you would be asked to assign the pages from your file to their corresponding questions.

4. **You can work and submit in groups of up to 4 people. If submitting as a group, make sure to associate all group members to the submission on gradescope.**

5. Pay special attention to the style of your code. Indent your code correctly, choose meaningful names for your variables, define constants where needed, choose the most appropriate control flow statements, break down your solutions by defining functions, etc.

6. For the math questions, you are expected to justify all your answers, not just to give the final answer (unless explicitly asked to).
   As a rule of thumb, for questions taken from zyBooks, the format of your answers, should be like the format demonstrated in the sample solutions we exposed.

Question 1:

a. Implement a function:
   `int printMonthCalender(int numOfDays, int startingDay)`
   This function is given two parameters:
   - `numOfDays` - The number of days in the month
   - `startingDay` – a number 1-7 that represents the day in the week of the first day in that month (1 for Monday, 2 for Tuesday, 3 for Wednesday, etc.).

   The function should:
   - Print a formatted monthly calendar of that month
   - Return a number 1-7 that represents the day in the week of the **last day** in that month.

Formatting Notes:

- The output should include a header line with the days' names.
- Columns should be spaced by a Tab.

Example: when calling `printMonthCalender(31, 4)` it should return 6, and should print:

```
Mon   Tue   Wed   Thr   Fri   Sat   Sun
                  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
```

b. A method for determining if a year is a leap year in the Gregorian calendar system is to check if it is divisible by 4 but not by 100, unless it is also divisible by 400.
For example, 1896, 1904, and 2000 were leap years but 1900 was not.
Write a function that takes in a year as input and return true if the year is a leap year, return false otherwise.
Note: background on leap year https://en.wikipedia.org/wiki/Leap_year

c. Implement a function:
`void printYearCalender(int year, int startingDay)`
This function is given two parameters:
   - `year` – an integer that represents a year (e.g. 2016)
   - `startingDay` – a number 1-7 that represents the day in the week of 1/1 in that year (1 for Monday, 2 for Tuesday, 3 for Wednesday, etc.).

   The function should use the functions from sections (a) and (b) in order to print a formatted yearly calendar of that year.

   Formatting Note: As the header for each month you should print the months' name followed by the year (e.g. March 2016).

   Example: Appendix A shows the expected output of the call `printYearCalender(2016, 5)`.

d. Write program that interacts with the user and your function in (c).

Question 2:

Consider the following definitions:

a. A **proper divisors** of a positive integer (≥ 2) is any of its divisors excluding the number itself.
   For example, the proper divisors of 10 are: 1, 2 and 5.

b. A **perfect number** is a positive integer (≥ 2) that is equal to the sum of its proper divisors.
   For example, 6 and 28 are perfect numbers, since:
   6 = 1 + 2 + 3
   28 = 1 + 2 + 4 + 7 + 14
   Background of perfect numbers: https://en.wikipedia.org/wiki/Perfect_number

c. **Amicable numbers** are two different positive integer (≥ 2), so related that the sum of the proper divisors of each is equal to the other number.
   For example, 220 and 284 are amicable numbers, since:
   284 = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110

   220 = 1 + 2 + 4 + 71 + 142

   (proper divisors of 220)

   (proper divisors of 284)

   Background of amicable numbers: https://en.wikipedia.org/wiki/Amicable_numbers

a. Write a function:
   `void analyzeDividors(int num, int& outCountDivs, int& outSumDivs)`
   The function takes as an input a positive integer `num` (≥ 2), and updates two output parameters with the number of `num`'s proper divisors and their sum.
   For example, if this function is called with `num`=12, since 1, 2, 3, 4 and 6 are 12s proper divisors, the function would update the output parameters with the numbers 5 and 16.
   Note: Pay attention to the running time of your function. An efficient implementation would run in $\Theta(\sqrt{num})$.

b. Use the function you wrote in section (a), to implement the function:
   `bool isPerfect(int num)`
   This functions is given positive integer `num` (≥ 2), and determines if it is perfect number or not.

c. Use the functions you implemented in sections (a) and (b), to write a program that reads from the user a positive integer M (≥ 2), and prints:
   - All the perfect numbers between 2 and M.
   - All pairs of amicable numbers that are between 2 and M (both numbers must be in the range).

   Note: Pay attention to the running time of your implementation. An efficient algorithm for this part would call `analyzeDividors` $\Theta(M)$ times all together.

Appendix A.

The expected output of the call `printYearCalender(2016, 5)` is:

```
January 2016
Mon   Tue   Wed   Thr   Fri   Sat   Sun
                        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

February 2016
Mon   Tue   Wed   Thr   Fri   Sat   Sun
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

March 2016
Mon   Tue   Wed   Thr   Fri   Sat   Sun
      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

April 2016
Mon   Tue   Wed   Thr   Fri   Sat   Sun
                        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

May 2016
Mon   Tue   Wed   Thr   Fri   Sat   Sun
                                    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
```

June 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
		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			

July 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
				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

August 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
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				

September 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
			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		

October 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
					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						

November 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
	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				

December 2016

Mon	Tue	Wed	Thr	Fri	Sat	Sun
			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	