NYU, Tandon School of Engineering
Extended Bridge to CS — Winter 2019

Homework #11
Due by Friday 3/29 11:55pm

Submission instructions:

1. For this assignment you should turn in 5 files, one for each question:
   - Four '.cpp' files (for questions 1, 2, 3, and 4). Name these files: 'YourNetID_hw8_q1.cpp', 'YourNetID_hw8_q2.cpp', etc.
     For each of these questions, create a main function that calls the other functions you were asked to implement in the question.
   - One '.pdf' file (for questions 5 and 6).
     Each question should start on a new page!
     Name this file 'YourNetID_hw8_q5to6.pdf'.

2. **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.

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

4. 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:

Give a **recursive** implement to the following functions:

a. `void printTriangle(int n)`

This function is given a positive integer `n`, and prints a textual image of a right triangle (aligned to the left) made of n lines with asterisks.

For example, `printTriangle(4)`, should print:

```
*
**
***
****
```

b. `void printOpositeTriangles(int n)`

This function is given a positive integer `n`, and prints a textual image of a two opposite right triangles (aligned to the left) with asterisks, each containing n lines.

For example, `printOpositeTriangles(4)`, should print:

```
****
***
**
*
*
**
***
****
```

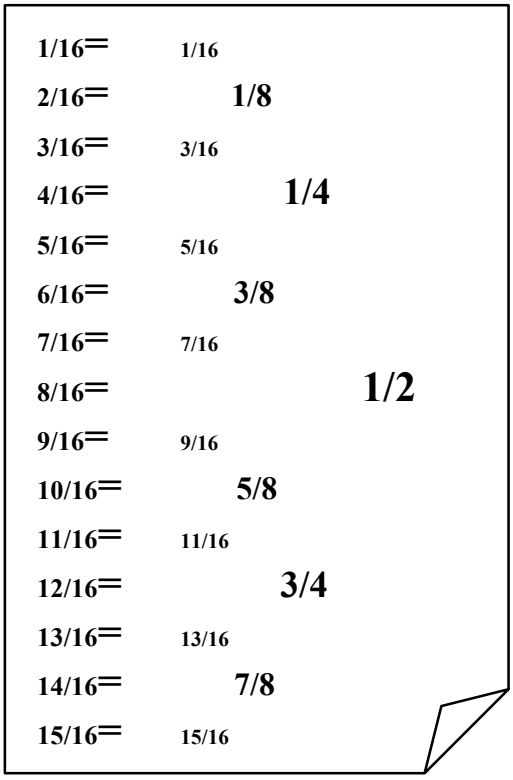
c. `void printRuler(int n)`

This function is given a positive integer `n`, and prints a vertical ruler of $2^n - 1$ lines. Each line contains '-' marks as follows:

- The line in the middle ($\frac{1}{2}$) of the ruler contains *n* '-' marks
- The lines at the middle of each half ($\frac{1}{4}$ and $\frac{3}{4}$) of the ruler contains (*n*-1) '-' marks
- The lines at the $\frac{1}{8}$, $\frac{3}{8}$, $\frac{5}{8}$ and $\frac{7}{8}$ of the ruler contains (*n*-2) '-' marks
- And so on …
- The lines at the $\frac{1}{2^k}$, $\frac{3}{2^k}$, $\frac{5}{2^k}$, …, $\frac{2^k-1}{2^k}$ of the ruler contains *1* '-' mark

For example, `printRuler(4)`, should print (only the blue marks):

```
-
- -
-
- - -
-
- -
-
- - - -
-
- -
-
- - -
-
- -
-
```



Hints:

1. Take for n=4: when finding `print_ruler(4)`, try to think first what `print_ruler(3)` does, and how you can use it to print a ruler of size 4.
   Then, generally identify what `print_ruler(n-1)` is **supposed** to print, and use that in order to define how to print the ruler of size n.
2. You may want to have more than one recursive call
3. It looks much scarier than it actually is

Question 2:

Give a **recursive** implement to the following functions:

a. `int sumOfSquares(int arr[], int arrSize)`
   This function is given `arr`, an array of integers, and its logical size, `arrSize`. When called, it returns the sum of the squares of each of the values in `arr`.
   For example, if `arr` is an array containing [2, -1, 3, 10], calling `sumOfSquares(arr, 4)` will return 114 (since, $2^2 + (-1)^2 + 3^2 + 10^2 = 114$).

b. `bool isSorted(int arr[], int arrSize)`
   This function is given `arr`, an array of integers, and its logical size, `arrSize`. When called, it should return `true` if the elements in `arr` are sorted in an ascending order, or `false` if they are not.

Question 3:

Write two **recursive** versions of the function `minInArray`. The function will be given a sequence of elements and should return the minimum value in that sequence. The two versions differ from one another in the technique we use to pass the sequence to the function.

In version 1 – The prototype of the function should be:
`int minInArray1(int arr[], int arrSize)`
Here, the function is given `arr`, an array of integers, and its logical size, `arrSize`.
The function should find the minimum value out of all the elements in positions:
0, 1, 2, …, arrSize-1.

In version 2 – The prototype of the function should be:
`int minInArray2(int arr[], int low, int high)`
Here, the function is given `arr`, an array of integers, and two additional indices: `low` and `high` (`low` ≤ `high`), which indicate the range of indices that need to be considered.
The function should find the minimum value out of all the elements in positions:
low, low+1, …, high.

Use the following main to check your program:

```
int main() {
    int arr[10] = {9, -2, 14, 12, 3, 6, 2, 1, -9, 15};
    int res1, res2, res3, res4;

    res1 = minInArray1(arr, 10);
    res2 = minInArray2(arr, 0, 9);
    cout<<res1<<" "<<res2<<endl;  //should both be -9

    res3 = minInArray2(arr, 2, 5);
    res4 = minInArray1(arr+2, 4); //arr+2 is equivalent to &(arr[2])
    cout<<res3<<" "<<res4<<endl;  //should both be 3

    return 0;
}
```

Question 4 (taken from Ch14 problem 6 in text):

The game of "Jump It" consists of a board with *n* positive integers in a row, except for the first column, which always contains zero. These numbers represent the cost to enter each column. Here is a sample game board where *n* is 6:

0	3	80	6	57	10

The object of the game is to move from the first column to the last column with the lowest total cost.

The number in each column represents the cost to enter that column. You always start the game in the first column and have two types of moves. You can either move to the adjacent column or jump over the adjacent column to land two columns over. The cost of a game is the sum of the costs of the visited columns.

In the board shown above, there are several ways to get to the end. Starting in the first column, our cost so far is 0. We could jump to 80, then jump to 57, and then move to 10 for a total cost of 80 + 57 + 10 = 147. However, a cheaper path would be to move to 3, jump to 6, then jump to 10, for a total cost of 3 + 6 + 10 = 19.

Write a **recursive** function that solves this problem and returns the lowest cost of a game board represented and passed as an array.

Note: your function shouldn't output the actual sequence of jumps, only the lowest cost of this sequence.

Question 5:

a. Use mathematical induction to prove that for any positive integer n, 3 divide $n^3 + 2n$ (leaving no remainder).
   Hint: you may want to use the formula: $(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$.
b. Use strong induction to prove that any positive integer n ($n \geq 2$) can be written as a product of primes.

Question 6:

Solve the following questions from the Discrete Math zyBook:

a) Exercise 7.4.1, sections a-g
b) Exercise 7.4.3, section c
   Hint: you may want to use the following fact: $\frac{1}{(k+1)^2} \leq \frac{1}{k(k+1)}$
c) Exercise 7.5.1, section a