NYU, Tandon School of Engineering
Bridge to CS

# Homework #4

**Submission instructions:**

1. You should submit your homework in the Gradescope system.

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

3. For this assignment, you should turn in 9 files:
   - Eight '.cpp' files, one for each question 1 to 8.
     Name your files 'YourNetID_hw4_q1.cpp', 'YourNetID_hw4_q2.cpp', etc.
     If a question contains two sections, make a `cout` statement before the implementation of each section (`cout<<`"section a"`<<endl;`, `cout<<`"section b"`<<endl;`, etc.)
   - A '.pdf' file with your answers for questions 9-11.
     Name your file 'YourNetID_hw4_q9to11.pdf'

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, 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:**

Write **two versions** of a program that reads a positive integer `n`, and prints the first `n` even numbers.

a) In the first, use a `while` loop.
b) In the second, use a `for` loop.

Each section should interact with the user **exactly** as it shows in the following example:

Please enter a positive integer: 3
2
4
6

**Question 2:**

In this question we will use a **simplified** version of the Roman Numerals System to represent positive integers.

The digits in this system are I, V, X, L, C, D and M. Each digit corresponds to a decimal value, as showed in the following table:

| **Roman digit** | I | V | X | L | C | D | M |
|---|---|---|---|---|---|---|---|
| **Decimal value** | 1 | 5 | 10 | 50 | 100 | 500 | 1000 |

A number in the *simplified Roman numerals system* is a sequence of Roman digits, which follow these 2 rules:

1. The digits form a monotonically non-increasing sequence. That is the value of each digit is less than or equal to the value of the digit that came before it.
   For example, DLXXVI is a monotonically non-increasing sequence of Roman digits, but XIV is **not**.
2. There is no limit on the number of times that 'M' can appear in the number.
   'D', 'L' and 'V' can each appear at most one time in the number.
   'C', 'X' and 'I' can each appear at most four times in the number.

For example: IIII, XVII and MMMMMMDCCLXXXXVII are legal numbers in our simplified Roman numeral system, but IIIII, XIV, VVI and CCXLIII are **not**.

Write a program that reads from the user a (decimal) number, and prints it's representation in the simplified Roman numerals system.

Your program should interact with the user **exactly** as it shows in the following example:

Enter decimal number:
147
147 is CXXXXVII

**Question 3:**

Write a program that reads from the user a positive integer (in a decimal representation), and prints its binary (base 2) representation.

Your program should interact with the user **exactly** as it shows in the following example:

Enter decimal number:
76
The binary representation of 76 is 1001100

Implementation Requirements:

1. You are supposed to implement the algorithm that converts to base 2. You should not use any `string` or special `cout` functionalities to make the conversion.
2. You are not allowed to use arrays.

**Question 4:**

Write two versions of a program that **reads a sequence of positive integers from the user**, calculates their **geometric mean**, and print the geometric mean.

Notes:

1. In mathematics, geometric mean of a dataset $\{a_1, a_2, a_3 \dots, a_n\}$ containing positive numbers, is given by: $\sqrt[n]{a_1 \cdot a_2 \cdot a_3 \cdots a_n}$.
   For example, the geometric mean of 2, 9 and 12 is equal to 6 ($\sqrt[3]{2 \cdot 9 \cdot 12} = 6$).
2. In order to calculate the n-th root of a number, you should call the **`pow`** function, located in the **`cmath`** library.

Your two versions should read the integer sequence in two ways:

a) First read the length of the sequence.

For example, an execution would look like:

Please enter the length of the sequence: 3
Please enter your sequence:
1
2
3
The geometric mean is: 1.8171

b) Keep reading the numbers until -1 is entered.

For example, an execution would look like:

Please enter a non-empty sequence of positive integers, each one in a separate line. End your sequence by typing -1:
1
2
3
-1
The geometric mean is: 1.8171

**Question 5:**

Write a program that asks the user to input a positive integer n, and prints a textual image of an hourglass made of $2n$ lines with asterisks.

For example if n=4, the program should print:

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

**Question 6:**

Write a program that asks the user to input a positive integer n, and print all of the numbers from 1 to n that have more even digits than odd digits.

For example, if n=30, the program should print:

2
4
6
8
20
22
24
26
28

**Question 7:**

Write a program that reads a positive integer n from the user and prints out a nxn multiplication table. The columns should be spaced by a tab.

Your program should interact with the user **exactly** as it shows in the following example:

Please enter a positive integer:
10

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

**Question 8:**

Implement a number guessing game. The program should randomly choose an integer between 1 and 100 (inclusive), and have the user try to guess that number.

Implementations guidelines:

1. The user can guess at most 5 times.
2. Before each guess the program announces:
    - An updated guessing-range, taking in to account previous guesses and responses.
    - The number of guesses that the user has left.
3. If the user guessed correctly, the program should announce that, and also tell how many guesses the user used.
4. If the user guessed wrong, and there are still guesses left, the program should tell the user if the number (it chose) is bigger or smaller than the number that the user guessed.
5. If the user didn't guess the number in all of the 5 tries, the program should reveal the number it chose.
6. Follow the execution examples below for the exact format.

In order to generate random numbers (of type **int**), you should first call the **srand** function (one time) to initialize a seed for the random number generator. Then, you can call the **rand** function repeatedly to generate random integers.

Follow the syntax as demonstrated in the code below:

```
1.  #include <iostream>
2.  #include <cstdlib>
3.  #include <ctime>
4.  using namespace std;
5.
6.  int main() {
7.      int x1, x2, x3, x4;
8.
9.      srand(time(0));
10.
11.     x1 = rand();
12.     x2 = rand();
13.     x3 = rand() % 100;
14.     x4 = (rand() % 100) + 1;
15.
16.     cout<<x1<<" "<<x2<<" "<<x3<<" "<<x4<<endl;
17.
18.     return 0;
19. }
```

Note that we first included the **cstdlib** and **ctime** libraries (lines 2 and 3).
In line 9 we called **srand(time(0))** to create the seed for the random number generator. Then, in lines 11-14, we created 4 random numbers: The first two were taken from the entire positive range of **int** variables. For **x3** and **x4** we used arithmetic manipulation to change the range. **x3** would be in the range 0-99 (since these are the possible remainders when dividing a number by 100), and **x4** would be in the range of 1-100 (shifting the range 0-99 by 1).
Further reading regarding random number generation can be found in the textbook on pages 188-189.

Your program should interact with the user **exactly** as it shows in the following two examples:

Execution example 1:

I thought of a number between 1 and 100! Try to guess it.
Range: [1, 100], Number of guesses left: 5
Your guess: 15
Wrong! My number is bigger.

Range: [16, 100], Number of guesses left: 4
Your guess: 34
Wrong! My number is smaller.

Range: [16, 33], Number of guesses left: 3
Your guess: 23
Congrats! You guessed my number in 3 guesses.

Execution example 2:

I thought of a number between 1 and 100! Try to guess it.
Range: [1, 100], Number of guesses left: 5
Your guess: 15
Wrong! My number is bigger.

Range: [16, 100], Number of guesses left: 4
Your guess: 50
Wrong! My number is smaller.

Range: [16, 49], Number of guesses left: 3
Your guess: 3
Wrong! My number is bigger.

Range: [16, 49], Number of guesses left: 2
Your guess: 34
Wrong! My number is smaller.

Range: [16, 33], Number of guesses left: 1
Your guess: 20
Out of guesses! My number is 25

**Question 9:**

Solve the following questions from the Discrete Math zyBook:

a) Exercise 4.1.3, sections b, c
b) Exercise 4.1.5, sections b, d, h, i, l

**Question 10:**

I. Solve the following questions from the Discrete Math zyBook:
   a. Exercise 4.2.2, sections c, g, k
   b. Exercise 4.2.4, sections b, c, d, g
II. Give an example of a function from the set of integers to the set of positive integers that is:
   a. one-to-one, but not onto.
   b. onto, but not one-to-one.
   c. one-to-one and onto.
   d. neither one-to-one nor onto

**Question 11:**

Solve the following questions from the Discrete Math zyBook:

a) Exercise 4.3.2, sections c, d, g, i
b) Exercise 4.4.8, sections c, d
c) Exercise 4.4.2, sections b-d
d) Exercise 4.4.6, sections c-e
e) **Extra Credit**: Exercise 4.4.4, sections c, d