EECS2040 Data Structure Hw #1 (Chapter 1, 2 of textbook)
due date 4/8/2021

Format: Use a text editor to type your answers to the homework problem. You need to submit your HW in an HTML file or a DOC file named as **Hw1-SNo.doc** or **Hw1-SNo.html**, where SNo is your student number. Submit the **Hw1-SNo.doc or Hw1-SNo.html** file to eLearn. Inside the file, you need to put the **header and your student number, name (e.g., EECS2040 Data Structure Hw #1 (Chapter 1, 2 of textbook) due date 4/8/2021 by SNo, name)** first, and then the **problem** itself followed by your **answer** to that problem, one by one. The grading will be based on the correctness of your answers to the problems, and the **format**. Fail to comply with the aforementioned format (file name, header, problem, answer, problem, answer,…), will certainly degrade your score. If you have any questions, please feel free to ask.

Part 1 (40% of Hw1)

1. (20%) Using the ADT1.1 *NaturalNumber* in the textbook pp.10, add the following operations to the *NaturalNumber* ADT: Predecessor, *IsGreater*, *Multiply*, *Divide*.
2. (20%) Determine the frequency counts for all statements (by step table) in the following two program segments:

   code (a):

   ```
   1  for(i=1;i<=n;i++)
   2      for(j=1;j<=I;j++)
   3          for(k=1;k<=j;k++)
   4              x++;
   ```

   code (b)

   ```
   1  i=1;
   2  while(i<=n)
   3  {
   4      x++;
   5      i++;
   6  }
   ```

3. (20%) For the function Multiply() shown below,
   (a) Introduce statements to increment count at all appropriate points and compute the count
   (b) Simplify the resulting program by eliminating statement and compute the count
   (c) Obtain the step count for the function using the frequency method.

   ```
   void Multiply(int **a,int **b, int **c, int m, int n, int p)
   {
      for(int i=0;i<m;i++)
         for(int j=0; j<p; j++)
         {
            c[i][j] = 0;
   ```

```
                for(int k=0;k<n;k++)
                  c[i][j] += a[i][k] * b[k][j];
            }
    }
```

4. (20%) A complex-valued matrix X is represented by a pair of matrices (A, B) where A and B contains real values. Write a program that computes the product of two complex-valued matrices (A, B) and (C, D), where (A, B) * (C, D) = (A+iB)*(C+iD) = (AC-BD)+i(AD + BC). Determine the number of additions and multiplications if the matrices are all nxn.
5. (20%) The Tower of Hanoi is a classical problem which can be solved by recurrence. There are three pegs and N disks of different sizes. Originally, all the disks are on the left peg, stacked in decreasing size from bottom to top. Our goal is to transfer all the disks to the right peg, and the rules are that we can only move one disk at a time, and no disk can be moved onto a smaller one. We can easily solve this problem with the following recursive method: If $N = 1$, move this disk directly to the right peg and we are done. Otherwise ($N > 1$), first transfer the top $N - 1$ disks to the middle peg applying the method recursively, then move the largest disk to the right peg, and finally transfer the $N - 1$ disks on the middle peg to the right peg applying the method recursively. Let T(N) be the total number of moves needed to transfer N disks.
   (a) Prove that $T(N) = 2T(N - 1) + 1$ with $T(1) = 1$.
   (b) Unfold this recurrence relation to obtain a closed-form expression for T(N). (T(N) is expressed in terms of function of N.)

Part 2 Coding (60% of Hw1)

You should submit:
(a) All your source codes (C++ file).
(b) Show the execution trace of your program.

1. (30%) Write a C++ program to implement the **ADT2.3 Polynomial** (pp.88) using Representation 3 (dynamic array of (coef, exp) tuples). Implement the Mult(Polynomial p) and Eval(float x). Estimate the computing time for Mult and Eval function. Add two more functions to input and output polynomials via **overloading** the >> and << **operators**.
   You should try out at least two runs of your program (execution trace) to

demonstrate the Add, Mult, Eval and input, output functions.

2. (35%) Write a C++ program to implement the **ADT2.4 SparseMatrix** in textbook (pp.97) (with Transpose implemented by FastTranspose). You should build you program based on the example codes in the book and implement the Add function and functions to input, output a sparse matrix by **overloading** the >> and << **operators**.
   You should try out at least two runs of your program to demonstrate the Add, Mult, Eval and input, output functions.
3. (35%) Write a C++ program to implement the **ADT2.5 String** (pp.114) (with Find function implemented by FastFind). In addition, write two more functions:
   String::Delete(int start, int length); //remove length characters beginning at start
   String::CharDelete(char c); //returns the string with all occurrence of c removed.
   You should try out at least two runs of your program to demonstrate all those functions.