Department of EECS
York University

Term: Fall 2016
Instructor: Andy Mirzaian

EECS2011: Fundamentals of Data Structures

Assignment 2

Due: 10 pm, Wednesday, October 19, 2016

- **This is an individual assignment; you may not share code with other students.**
- **Read the course FAQ on how to submit assignments and the marking scheme.**
- **Print your name, eecs account and student ID number on top of every file you submit.**

Further Instructions:

- This assignment consists of 3 problems and is designed to let you practice on recursion versus iteration, linear structures such as lists, stacks, queues, and some Java programming concepts.
- You are expected to apply appropriate OOP principles in the design of your programs.
- Feel free to design additional classes or members as you deem necessary.
- Include your additional explanations and I/O test results in a file named **a2sol.pdf.**

Problem 1 (30 points): Enumeration of Coin Change Making:

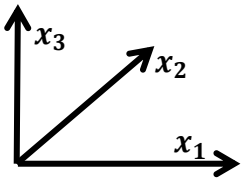
Develop a program in a class named ***Coins*** that includes a method with the signature ***ways(int money)*** that uses recursion (possibly through recursive helper methods) to enumerate the distinct ways in which the given amount of money in cents can be changed into quarters, dimes, nickels, and pennies. Test your program from the ***main()*** method for various amounts of change by prompting the user to enter an amount in cents. For example, suppose the entered amount is 17. Then there are 6 ways to make change for this amount. Here is what exactly the I/O printout should look like:

Enter an amount in cents:
17
This amount can be changed in the following ways:
1) 1 dime, 1 nickel, 2 pennies
2) 1 dime, 7 pennies
3) 3 nickels, 2 pennies
4) 2 nickels, 7 pennies
5) 1 nickel, 12 pennies
6) 17 pennies

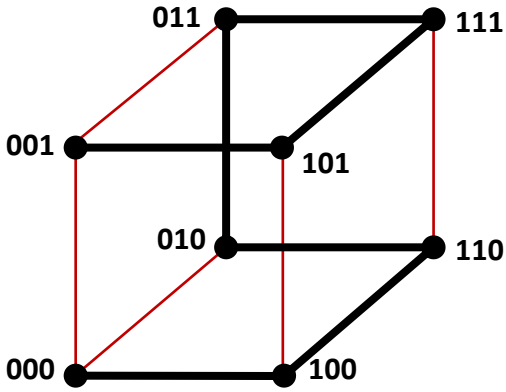
In your program you may use additional private helper methods, constructors, etc., as you deem necessary.
Restriction: Other than the I/O types, you may use only a constant amount of additional memory cells for your local variables and method parameters. So, you are not allowed to use elaborate structures such as arrays, lists, stacks or queues that may hold more than a constant number of elements.

Problem 2 (40 points): A Walk on the Hypercube:

The unit hypercube in the n-dimensional space is the set of all points $(x_1, x_2, \dots, x_n)$ in that space such that its coordinates are restricted to be $0 \le x_i \le 1$ for $i = 1..n$. This hypercube has 2^n corners. Each coordinate of a corner is either 0 or 1. Each edge of the hypercube connects a pair of corners that differ in exactly one of their coordinates. A walk on the hypercube starts at some corner and walks along a sequence of edges of the hypercube so that it visits each corner exactly once. There are many such walks possible. The following figure shows an example for $n = 3$. The edges of the (hyper)cube along the walk are thickened.



A Walk:
000
100
110
010
011
111
101
001



The problem: for any given n, output the sequence of corners visited along a walk of the n-dimensional hypercube.

We want to study recursive and iterative solutions to this problem. In order to do that, design a ***Hypercube*** class that (among possibly other members) includes the following:

a) A nested class named ***Corner.*** An instance of this type represents a corner of the hypercube and occupies O(n) memory cells.

b) A recursive method named ***recursiveWalk()*** to solve the above problem.
The parameters and local variables of this method can only be primitives or Corner types.

c) An iterative method named ***iterativeWalk()*** to solve the above problem using a single **queue.** The elements of this queue, as well as the other local variables and method parameters, can only be primitives or Corner types.

(Note: We know that any recursive algorithm can be converted to an equivalent iterative one by simulating a recursion stack. However, here you are forbidden to use a stack or any structure that may act as one such as an array or a list.)

d) Explain the running times of your solutions in parts (b) and (c) in a2sol.pdf.

e) The ***main()*** method of Hypercube class should test recursiveWalk and iterativeWalk for at least the dimensions $n = 3, 4, 5$. Report the I/O results in a2sol.pdf.

Problem 3 (30 points): Augmented Stack with getMin:

An **AugmentedStack** is an ADT that maintains any generic stack S with comparable element type under the following operations, each requiring **O(1) worst-case** time:

push(x,S): Insert element x on top of stack S.

pop(S): Remove and return the top element of S (if not empty).
Return null if S is empty.

getMin(S): Return the minimum element on stack S (if not empty).
Return null if S is empty.
(Upon return from this method, stack S should be in the same state as it was before this method was called. In particular, the minimum element is not removed from the stack.)

Also include the usual stack operations ***isEmpty(S)*** and ***top(S)*** with the same meaning as for conventional stacks that run in O(1) worst-case time.

Design and analyze a data structure that implements this ADT with the specified running times.

Hint: maintain additional private auxiliary fields in your augmented structure.