San José State University
College of Science / Department of Computer Science
CS 146 Data Structures and Algorithms
Section 4/7, Spring 2015

Instructor: Dr. Angus Yeung

Assignment 2

Due Date: Monday, March 2, 11:59 pm

Submission:

1. Create a folder called `hw2` on your computer.
2. Create a WORD document (or any similar word processing document) and put all of your answers in the same document. Your word processing software must allow you to enter math equations.
3. Save the file as `hw2`, e.g., `hw2.doc` or `hw2.docx`. Export the Word document (or similar word processing document) to PDF file. For example, File -> Save As -> choose Format as "PDF".
4. Alternatively, you can write your answers on paper, scan and save your work as a PDF file.
5. Copy or move your PDF file, `hw2.pdf`, to folder `hw2`
6. For programming assignment, follow the instruction in the question to submit your `.java` files.
7. When you are ready for submitting your work, zip up the `hw2` folder and upload it to Assignment 2 on Canvas.

Only softcopy is acceptable. Do not hand in your solutions in hardcopy. Do not encrypt your zip file with a password.

Problems (Total: 100 Points):

Part A contains written assignment questions. Follow the instruction in each question carefully and show all of your work. (40 Points)

Part B contains both written and programming assignment questions. Follow the guidelines in each assignment for programming and submission requirements. (60 Points)

PART A Non-programming Questions (40 Points)

2.1 (5 Points) We have seen how dynamic arrays enable arrays to grow while still achieving constant-time amortized performance. This problem concerns extending dynamic arrays to let them both grow and shrink on demand.

1. Consider an underflow strategy that cuts the array size in half whenever the array falls below half full. Give an example sequence of insertions and deletions where this strategy gives a bad amortized cost.
2. Then, give a better underflow strategy than that suggested above, one that achieves constant amortized cost per deletion.

2.2 (5 Points) What is the running time of the following code? Give the estimate in O() and explain why you have reached such answer.

```
public static List<Integer> makeList( int N )
{
      ArrayList<Integer> lst = new ArrayList<>( );

      for( int i = 0; i < N; i++ )
      {
            lst.add( i );
            lst.trimToSize( );
      }
 }
```

2.3 (5 Points) If the recursive routine in Section 2.4 (our textbook) used to compute Fibonacci numbers is run for *N* = 50,

a) How many calls will be stored in stack?
b) Is stack space likely to run out? Why or why not?
c) Will the routine terminate in a reasonable amount of time? Please explain.

2.4 (10 Points) The following routine removes the first half of the list passed as a parameter:

```
public static void removeFirstHalf( List<?> lst )
{
      int theSize = lst.size( ) / 2;

      for( int i = 0; i < theSize; i++ )
            lst.remove( 0 );
}
```

a) Why is theSize saved prior to entering the for loop?
b) What is the running time of removeFirstHalf if lst is an ArrayList?
c) What is the running time of removeFirstHalf if lst is a LinkedList?
d) Does using an iterator make removeHalf faster for either type of List?

For each question, you must provide both answer and explanation to get full credit.

2.5 (15 Points) Follow our textbook example on page 88, perform the infix to postfix conversion of the following mathematical expression:

```
(600-203)*(143+21)/(87+17)
```

You must show the changes in both stack and output when you push each symbol/operator to your stack. Student must illustrate each step to get full credit.

PART B Programming Questions. (60 Points)

2.6 (20 Points) A common problem for compilers and text editors is determining whether the parentheses in a string are balanced and properly nested. For example, the string ((())())() contains properly nested pairs of parentheses, which the strings)()(and ()) do not.

Design a stack algorithm that returns true if a string contains properly nested and balanced parentheses, and false if otherwise. For full credit, identify the position of the first offending parenthesis if the string is not properly nested and balanced.

Write a Java program for your stack algorithm. Your program must read in a text file and process the entire file. It must print out the exact location of first offending parenthesis, e.g., line 3, character 4, if the parentheses are not balanced.

How to Submit:

1. Create the folder "`balanceSymbols`" that contains all required `.java` file(s).
2. The folder should be part of the `hw2.zip` file that you upload to Canvas.
3. Do not include any other file types inside the `balanceSymbols` folder except `.java` file(s).
4. Do not declare and use any "`package`" in your `.java` file(s).
5. Use "`balanceSymbols`" as the class name – so you can run as `%java balanceSymbols`.

There is one point penalty for each requirement that a student fails to follow (total penalty: 5 points).

2.7 (20 Points) Write a program which uses stack to check for balancing symbols in Java language (/* */, (), [], {}). Your Java program must be able to read any *.java file and prints out the sums of each pair of balancing symbols, e.g., */* */ = 14 pairs, () = 30 pairs, [] = 56 pairs, {} = 153 pairs*. If user doesn't provide any Java file, the program will read in `./countJavaPairs.java` by default and print out the summary of this specific Java source code file.

How to Submit:

1. Create the folder "`countJavaPairs`" that contains all required `.java` file(s).
2. The folder should be part of the `hw2.zip` file that you upload to Canvas.
3. Do not include any other file types inside the `countJavaPairs` folder except `.java` file(s).
4. Do not declare and use any "`package`" in your `.java` file(s).
5. Use "`countJavaPairs`" as the class name – so you can run as `%java countJavaPairs`.

There is one point penalty for each requirement that a student fails to follow (total penalty: 5 points).

2.8 (20 Points) Efficiently implement a queue class using a circular array.

How to Submit:

1. Create the folder "`cirArrayQueue`" that contains all required `.java` file(s).
2. The folder should be part of the `hw2.zip` file that you upload to Canvas.
3. Do not include any other file types inside the `cirArrayQueue` folder except `.java` file(s).
4. Do not declare and use any "`package`" in your `.java` file(s).
5. Use "`cirArrayQueue`" as the class name – so you can run as `%java cirArrayQueue`.

There is one point penalty for each requirement that a student fails to follow (total penalty: 5 points).