CS106B Instructor: Cynthia Lee

Autumn 2017 November 2, 2017

MIDTERM EXAM

NAME (LAST, FIRST): ____

SUNET ID: ____@stanford.edu

Problem	1	2	3	4	5	TOTAL
Topic	ADTs	Pointers, Memory	Recursion	Linked List Nodes	Big-O	
Score						
	16	12	12	8	12	60

Instructions:

- The time for this exam is **2 hours**, or 120 minutes. There are 60 points possible, so about 2 minutes per point. This should give you a general sense of pacing, though each person will have different problems that challenge them more or less than others.
- Use of anything other than a pencil, eraser, pen, one 8.5x11 page (two sides) of notes, and the official textbook is prohibited. In particular, **no** computers or digital devices of any kind are permitted. Blank scratch paper may be provided by proctors, but NOT be turned in with exam.
- PLEASE rip off the pages of **library reference in the back of the exam** and do not turn them in.
- Please do NOT staple or otherwise insert new pages into the exam. Changing the number of pages in the exam confuses our automatic scanning system. Thanks.
- For coding problems, you do not need to #include or use function prototypes.

Please sign *before* you begin:

I agree to abide by the spirit and letter of the Honor Code, and to follow the instructions above.

____ ______________

(Signature) (Date)

1. **ADTs (16pts).**

(a) (6pts) Consider the following code, written using Stanford library ADT implementations:

```
void collectionMystery(Queue<int>& q) {
   Stack<int> s1;
   Stack<int> s2;
   while (!q.isEmpty()) {
      s1.push(q.dequeue());
      if (!q.isEmpty()) {
         s2.push(q.dequeue());
      }
   }
   while (!s1.isEmpty()) {
      s2.push(s1.pop());
   }
   while (!s2.isEmpty()) {
      q.enqueue(s2.pop());
   }
   cout << q << endl;
}
```

Now write the output of the above code, given the following inputs. In the inputs shown, the leftmost entry is least recently added item, and the rightmost entry is the most recently added item.

Input: Output:

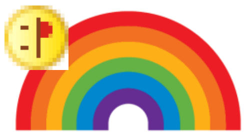
`q = {4, 5, 7, 6, 3, 2}` ____________________________________

`q = {1, 3, 5, 7, 9, 11, 13}` ____________________________________

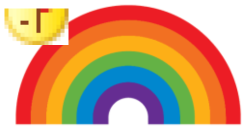
(b) (10pts) For this problem, you'll write a new Fauxtoshop filter. The function takes a `Grid<int>` representing the background image, a second `Grid<int>` that represents an image that you want to place on top of the background image, and a row, column pair that indicates where the second image should be placed (*i.e.*, identifies the top left corner of placement). The trick is that when you place the second image, you should place it *rotated left 90°*. You should make the changes to the background image Grid, which is passed by reference. If the row/col specified would put the sticker image partially or completely off of the background image, simply make the changes to the background that are visible, if any (so make no changes to the background in the case of row/col that would put the sticker completely out of view). If you need extra space to write your solution, use the back of a page.

Examples:

`background (before):  sticker:  row=0, col=0          background (after):`



`background (before):  sticker:  row=-20, col=0        background (after):`



```
void rotateAndPlaceImg(Grid<int>& background, const Grid<int>& sticker,
                       int row, int col) {





}
```

2. **Pointers and Memory (12pts).** Draw the state of memory at the end of the execution of this code.
   - Be careful in showing where your pointers originate and terminate (outer box vs. inner box).
   - Leave uninitialized or unspecified areas blank, and clearly mark NULL values by writing NULL or drawing a slash through the box.
   - Draw the components in the appropriate stack and heap areas marked for you.
   - Draw struct fields adjacent to each other (held in one subdivided box), and label each field with the field name for clarity.

```
int eleven = 11;
int season = 2;
Things* stranger = new Things;
stranger->el = eleven;
stranger->barb = &season;
stranger->joyce = new Things;
stranger->joyce->el = stranger->el;
stranger->joyce->barb = stranger->barb;
int* eggos = new int[3];
eggos[2] = eleven;
```

```
struct Things {
   int el;
   int* barb;
   Things* joyce;
};
```

DRAWING:

Stack:

Heap:

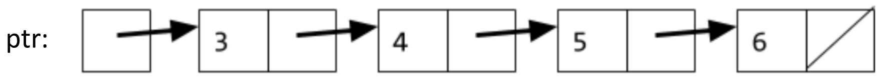
3. **Recursion (12pts).** Write a recursive function **gossipCheck** that assesses the risk that gossip you might want to share with your immediate friends might make it back to the person the gossip is about.
    - The function takes a string that is your name (you are the gossiper), the name of the person you want to gossip about (victim), a maximum distance maxDist, and a friendships Map that maps each person's name to the list of names of people they might share gossip with. For example, friendships[gossiper] is the list of names of your immediate friends that you might share gossip with. Note that friendship in this definition is not necessarily mutual—a person who is on your list of those you would share gossip with may or may not share gossip with you (*i.e.*, you may or may not be on their list).
    - If you share a piece of gossip with all your immediate friends, and they share it with all their immediate friends, and so on, the gossip may eventually reach back to the person you gossiped about! Your function should return true if such a gossiper to victim chain is possible in no more than maxDist steps (include gossiper, victim, and each person in between in the count), otherwise return false.
    - Your solution must not use any global variables, and must be fundamentally recursive in approach. You must use the function signature provided below, but you are free to use this as a "wrapper" and define a separate recursive function, if you wish.
    - If you need extra space to write your solution, use the back of a page.

```
bool gossipCheck(string gossiper, string victim, int maxDist,
                 Map<string, Vector<string>>& friendships) {









}
```

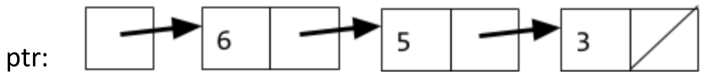
4. **Linked List Nodes (8pts).** Write code that will turn the "before" picture into the "after" picture by modifying links between the nodes shown and/or creating new nodes as needed. This is not a general algorithm—you are writing code for this example **only**, using the variable names shown. You are NOT allowed to change any existing node's data field value, nor create new ListNode objects, but you may create a single `ListNode*` pointer variable to point to existing nodes. **Your code should NOT leak memory.**

```
struct ListNode {
    int data;
    ListNode* next;
};
```

Before:



After:



5. **Big-O (12pts).** Give a tight bound of the nearest runtime complexity class (worst-case) for each of the following code fragments in Big-O notation, in terms of variable N. As a reminder, when doing Big-O analysis, we write a simple expression that gives only a power of N, such as O(N^2) or O(log N), *not* an exact calculation. Write your answer in the blanks on the right side. For each problem, the N that you use for your analysis is defined as the value of the variable **N** shown in the code.

Question (3pts each)	Answer
See code 1 below	O( )
See code 2 below	O( )
See code 3 below	O( )
See code 4 below	O( )

```
// assume numRows() == numCols()
void flipVertical(Grid<int>& img){
    int N = img.numRows();
    for(int row = 0; row < N/2; row++) {
        for (int col = 0; col < N; col++) {
            img[N - row - 1][col] = img[row][col];
        }
    }
}
```

O(   )

```
void printStuff(Vector<int>& vec) {
    int N = vec.size();
    recursePrint(vec, N);
}
void recursePrint(Vector<int>& vec, int size) {
    if (size == 0) return;
    for (int i = 0; i < size; i++) {
        cout << vec[i] << endl;
    }
    recursePrint(vec, size - 1);
}
```

O(   )

```
int addZeros(Vector<int>& vec) {
    int N = vec.size();
    for (int i = 0; i < N; i++) {
        vec.insert(0, 0);
    }
}
```

O(   )

```
int addOnes(Vector<int>& vec) {
    int N = vec.size();
    for (int i = 0; i < N; i++) {
        vec.add(1);
    }
}
```

O(   )

Please excuse the use of a one-letter, capitalized variable name, which we've told you is bad style. The naming N is to give clarity on our definition of "size" for each problem. ☺

Summary of Relevant Data Types

We tried to include the most relevant member functions and operators for the exam, but not all are listed. You are free to use ones not listed here that you know exist. ***You do <u>not</u> need to do #include for these.***

```
class string {
 bool empty() const; // O(1)
 int size() const; // O(1)
 int find(char ch) const; // O(N)
 int find(char ch, int start) const; // O(N)
 string substr(int start) const; // O(N)
 string substr(int start, int length) const; // O(N)
 char& operator[](int index);  // O(1)
 const char& operator[](int index) const; // O(1)
};

class Vector {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void add(const Type& elem); // operator+= used similarly - O(1)
 void insert(int pos, const Type& elem); // O(N)
 void remove(int pos); // O(N)
 Type& operator[](int pos); // O(1)
};

class Grid {
 int numRows() const; // O(1)
 int numCols() const; // O(1)
 bool inBounds(int row, int col) const; // O(1)
 Type get(int row, int col) const; // cascade of operator[] also works - O(1)
 void set(int row, int col, const Type& elem); // O(1)
};

class Stack {
 bool isEmpty() const; // O(1)
 void push(const Type& elem); // O(1)
 Type pop(); // O(1)
};

class Queue {
 bool isEmpty() const; // O(1)
 void enqueue(const Type& elem); // O(1)
 Type dequeue(); // O(1)
};
```

```
class Map {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void put(const Key& key, const Value& value); // O(logN)
 bool containsKey(const Key& key) const; // O(logN)
 Value get(const Key& key) const; // O(logN)
 Value& operator[](const Key& key); // O(logN)
};
```
Example for loop: `for (Key key : mymap){…}`

```
class HashMap {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void put(const Key& key, const Value& value); // O(1)
 bool containsKey(const Key& key) const; // O(1)
 Value get(const Key& key) const; // O(1)
 Value& operator[](const Key& key); // O(1)
};
```
Example for loop: `for (Key key : mymap){…}`

```
class Set {
 bool isEmpty() const; // O(1)
 int size() const; // O(1)
 void add(const Type& elem); // operator+= also adds elements – O(logN)
 bool contains(const Type& elem) const; // O(logN)
};
```
Example for loop: `for (Type elem : mymap){…}`

```
class Lexicon {
  int size() const; // O(1)
  bool isEmpty() const; // O(1)
  void clear(); // O(N)
  void add(string word); // O(W) where W is word.length()
  bool contains(string word) const; // O(W) where W is word.length()
  bool containsPrefix(string pre) const; // O(W) where W is pre.length()
};
```
Example for loop: `for (string str : english){…}`