CS 106B Practice Midterm Exam #5

(based on CS 106B Winter 2014 midterm)

This sample exam is intended to demonstrate an example of some of the kinds of problems that will be asked on the actual midterm exam. We do not guarantee that the number of questions in this sample exam will match the number on the real exam, nor that every kind of problem shown here will exactly match the kinds of problems shown on the final exam, though the real exam will be generally similar overall. Also, to save paper, this exam does not provide blank pages for writing space. The real exam will provide more blank space for you to write your answers.

In general, problem solutions are not graded on "style" as long as they work properly, though many questions do have some rules and constraints. For example, you should not use global variables, and some problems require recursion or a particular data structure.

1. C++ Basics and Parameters (read)

The following code C++ uses parameters and produces four lines of output. What is the output?
For the purposes of this problem, assume that the variables in `main` are stored at the following memory addresses:

- `main`'s `b` variable is stored at address `0xbb00`
- `main`'s `d` variable is stored at address `0xdd00`
- `main`'s `e` variable is stored at address `0xee00`
- `main`'s `f` variable is stored at address `0xff00`

```
int parameterMystery5(int* d, int e, int& f) {
    f += 10;
    *d = e + 2;
    e--;
    cout << e << " " << d << " " << *d << " " << f << endl;
    return e + f;
}

int main() {
    int b = 0;
    int d = -1;
    int e = 5;
    int f = 2;

    b = parameterMystery5(&d, e, f);
    parameterMystery5(&f, d, e);
    parameterMystery5(&b, f, d);

    cout << d << " " << e << " " << f << " " << b << endl;
    cout << endl;

    return 0;
}
```

2. File I/O and Strings (write)

Write a function named **printBox** that accepts two parameters: a string holding a file name, and an integer for a width. Your function should open that file and reads its contents as a sequence of lines, and display the lines to the console with a **'box' border of # characters** around them on all four sides. The second parameter to your function indicates the total width of the box including its border. You must also convert each line to **"title case"** by capitalizing the first letter of the line and lowercasing all subsequent letters. For example, suppose the file `poem.txt` contains the following text:

```
roses ARE red
VIOLETS Are blUE

All my BASE
ARE belong To YOU
```

Then the following are outputs from two calls to your function:

```
printBox("poem.txt", 19);
####################
#Roses are red     #
#Violets are blue  #
#                  #
#All my base       #
#Are belong to you#
####################
```

```
printBox("poem.txt", 30);
##############################
#Roses are red               #
#Violets are blue            #
#                            #
#All my base                 #
#Are belong to you           #
##############################
```

You may assume that the width value passed is non-negative and that no line in the file is too wide to fit into a box of that width. Notice that the file might contain blank lines.

If the input file does not exist or is not readable, your function should print no output. For full credit, your solution should read the file only once, not make multiple passes over the file data.

3. ADTs / Collections (read)

Consider the following function:

```
void collectionMystery5(Map<string, int>& m1, Map<int, string>& m2) {
    Map<string, string> result;
    for (string s : m1) {
        if (m2.containsKey(m1[s])) {
            result[s] = m2[m1[s]];
        }
    }
    cout << result << endl;
}
```

Write the output produced by the function when passed each of the following pairs of maps:

a) Maps

```
m1: {"bar":1, "cat":2, "foo":3, "mum":4}
m2: {1:"earth", 2:"wind", 3:"air", 4:"fire"}
```

Output

b) Maps

```
m1: {"five":105, "four":104, "one":101, "six":106, "three":103, "two":102}
m2: {99:"uno", 101:"dos", 103:"tres", 105:"cuatro"}
```

Output

c) Maps

```
m1: {"a":42, "b":9, "c":7, "d":15, "e":11, "f":24, "g":7}
m2: {1:"four", 3:"score", 5:"and", 7:"seven", 9:"years", 11:"ago"}
```

Output

4. ADTs / Collections (write)

Write a function named **rarest** that accepts a reference to a Map from strings to strings as a parameter and returns the value that occurs least frequently in the map. If there is a tie, return the value that comes earlier in ABC order. For example, if a map variable called map containing the following elements:

```
{"Alyssa":"Harding", "Char":"Smith", "Dan":"Smith", "Jeff":"Jones", "Kasey":"Jones",
 "Kim":"Smith", "Morgan":"Jones", "Ryan":"Smith", "Stef":"Harding"}
```

Then a call of rarest(map) would return "Harding" because that value occurs 2 times, fewer than any other. Note that we are examining the values in the map, not the keys. If the map passed is empty, throw a string **exception**.

For full credit, obey the following restrictions in your solution. A solution that disobeys them can get partial credit.

- You may create **one additional data structure** (stack, queue, set, map, etc.) as auxiliary storage. (You can have as many simple variables as you like.)
- Your solution should run in O(N log N) time or faster, where N is the number of pairs in the map.
- You should not modify the contents of the map passed to your function. Declare your function in such a way that any caller can be sure that this will not happen.

---

5. Algorithm Analysis and Big-Oh (read)

Give a tight bound of the nearest runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of the variable N. In other words, write the code's growth rate as N grows. Write a simple expression that gives only a power of N, such as $O(N^2)$ or $O(\log N)$, *not* an exact calculation like $O(2N^3 + 4N + 14)$. Write your answer on the right side in the blanks provided.

Question	Answer

a)
`int sum = 0;`
`for (int i = 1; i <= N; i++) {`
`  for (int j = 1; j <= 2 * N; j++) {`
`  sum++;`
`  }`
`}`
`cout << sum << endl;`
	O(_____)

b)
`int sum = 0;`
`for (int i = 1; i <= 100000; i++) {`
`  for (int j = 1; j <= i; j++) {`
`  for (int k = 1; k <= N; k++) {`
`  sum++;`
`  }`
`  }`
`}`
`for (int x = 1; x <= N; x += 2) {`
`  sum++;`
`}`
`cout << sum << endl;`
	O(_____)

c)
`Vector<int> v;`
`for (int i = 1; i <= N; i++) {`
`  v.insert(0, i);`
`}`
`HashSet<int> s;`
`for (int k : v) {`
`  s.add(k);`
`}`
`cout << "done!" << endl;`
	O(_____)

d)
`Queue<int> q;`
`for (int i = 1; i <= 2*N; i++) {`
`  q.enqueue(i);`
`}`
 	
`Map<int, int> map;`
`while (!q.isEmpty()) {`
`  int k = q.dequeue();`
`  map[k] = -k;`
`}`
`cout << "done!" << endl;`
	O(_____)

e)
`HashMap<int, int> map;`
`for (int i = 1; i <= N * N; i++) {`
`  map.put(i, i*i);`
`}`
`HashSet<int> set;`
`for (int k : map) {`
`  set.add(map.get(k));`
`}`
`cout << "done!" << endl;`
	O(_____)

6. Recursion (read)

For each of the calls to the following recursive function below, indicate what value is returned:

```
int recursionMystery5(int a, int b) {
    if (a < 10 || b < 10) {
        return a + b;
    } else if (a > b) {
        int x = recursionMystery5(a / 2, b / 2);
        int y = recursionMystery5(b, a - b);
        return x + y;
    } else {
        return recursionMystery5(a, b / 2);
    }
}
```

Call	Returns
a) `recursionMystery5(11, 18)`	
b) `recursionMystery5(26, 12)`	
c) `recursionMystery5(32, 48)`	

7. Recursion (write)

Write a recursive function named **sequence** that accepts an integer *k* as its parameter and prints out the numbers 1 through *k* inclusive in a particular pattern separated by plus signs and wrapped in parentheses. The order of the numbers should begin with all of the evens in downward order, followed by all of the odds upward from 1. Each time a number is added to the pattern, a new set of parentheses and a plus sign are added to the pattern. If the value for *k* is 0 or negative, throw an integer **exception**.

The table below shows several calls and their expected output. You must reproduce this output format exactly.

Call	Output
`sequence(1);`	`1`
`sequence(2);`	`(2 + 1)`
`sequence(3);`	`((2 + 1) + 3)`
`sequence(4);`	`(4 + ((2 + 1) + 3))`
`sequence(5);`	`((4 + ((2 + 1) + 3)) + 5)`
`sequence(6);`	`(6 + ((4 + ((2 + 1) + 3)) + 5))`
`sequence(7);`	`((6 + ((4 + ((2 + 1) + 3)) + 5)) + 7)`
`sequence(8);`	`(8 + ((6 + ((4 + ((2 + 1) + 3)) + 5)) + 7))`
`sequence(9);`	`((8 + ((6 + ((4 + ((2 + 1) + 3)) + 5)) + 7)) + 9)`
`sequence(10);`	`(10 + ((8 + ((6 + ((4 + ((2 + 1) + 3)) + 5)) + 7)) + 9))`

Do not construct any data structures (no array, vector, set, map, etc.), and do not declare any **global variables**. Also, **do not use loops**; you must use recursion. You are allowed to define other **"helper"** functions if you like.

---

8. Recursive Backtracking (write)

(thanks to Keith Schwarz for this problem)

Write a recursive function named **`isElementSpellable`** that checks whether a given string can be spelled out using just element symbols from the Periodic Table. For example, the word "began" can be spelled out as BeGaN (beryllium, gallium, nitrogen), and the word "feline" can be spelled out as FeLiNe (iron, lithium, neon). Not all words have this property, though; the word "interesting" cannot be made out of element letters, nor can the word "chemistry" (though, interestingly, the word "physics" can be made as PHYSICS (phosphorous, hydrogen, yttrium, sulfur, iodine carbon, sulfur).

You don't need to know anything about chemistry or have the periodic table memorized to solve this problem; you are given a `Lexicon` containing all the element symbols in the periodic table. Write a function:

```
bool isElementSpellable(string text, Lexicon& symbols)
```

that accepts a string, then returns whether that string can be written using only element symbols. If you like, you may assume the fact that all element symbols are at most three letters. If passed the empty string, you should return `true`.

You may use a loop in your solution if you like, but the overall algorithm must use recursion and backtracking.

---

9. Implementing a Collection Class (write)

Write a member function **filter** to be added to the ArrayList class from lecture. Your function should accept a reference to a Set of integers as its parameter and should remove from your list any integers that appear in the set. For example, suppose that an ArrayList called list stores the following sequence of eleven values:

{3, 14, 5, -1, 7, 14, 7, 7, 29, 3, 7}

And suppose we create a set of integers called set that stores {3, 5, 7}. The call of list.filter(set); would change the list to store the following sequence of four values:

{14, -1, 14, 29}

If the set passed is empty, or if none of its values occur in your list, then the list is unchanged by your function. You should return a result of true if the list is modified and false if it is not modified by the call.

You should <u>not</u> call any other public member functions of ArrayList; all traversing and modifying of the list should be done in your own code. Your function should not modify the state of the set that is passed in; its header should be declared in such a way as to promise this to the client. For full credit, your solution must run in $O(N^2)$ time or better, where N is the number of elements in the list.

You should write the member function's body as it would appear in ArrayList.cpp. You do not need to write the function's header as it would appear in ArrayList.h. Write only your member function, not the rest of the class. Remember that you are adding this as a member function to the ArrayList class from lecture:

```
class ArrayList {
public:
    void add(int value);
    void clear();
    int get(int index) const;
    void insert(int index, int value);
    bool isEmpty() const;
    void remove(int index);
    void set(int index, int value);
    int size() const;
    string toString() const;
}
```

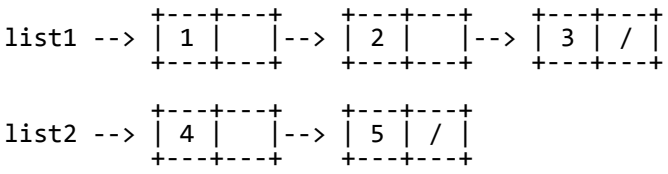
```
private:
    int* elements;
    int mysize;
    int capacity;
    void checkCapacity();
    void checkIndex(int i, int min, int max);
```

10. Pointers and Linked Nodes (write)

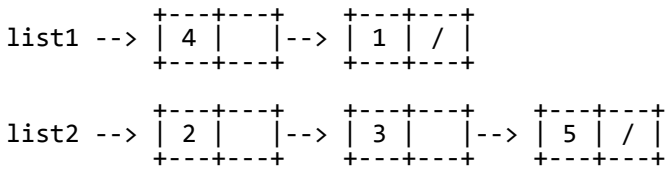
Write the code that will turn the "before" picture into the "after" picture by modifying links between the nodes shown. There may be more than one way to write the code, but you are NOT allowed to change any existing node's `data` field value. You also should not create new `ListNode` objects unless necessary to add new values to the chain, but you may create **a single `ListNode*` pointer** variable (in addition to `list2`) to point to any existing node if you like.

If a pointer variable does not appear in the "After" picture, it doesn't matter what value it has after the changes are made. If a *node* does not appear in the "After" picture, you must **free its memory** to avoid a memory leak.

Before



After



To help maximize partial credit in case you make mistakes, we suggest that you include optional comments with your code that describe the links you are trying to change, as shown in the solution code for the linked list section handout.

Assume that you are using the `ListNode` structure as defined in lecture and section:

```
struct ListNode {
    int data;         // data stored in this node
    ListNode* next;   // a link to the next node in the list

    ListNode(int data, ListNode* next) { ... }   // constructor
};
```