CS 106B Practice Midterm Exam #4

(based on CS 106B Autumn 2013 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 `a` variable is stored at address `0xaa00`
- `main`'s `b` variable is stored at address `0xbb00`
- `main`'s `c` variable is stored at address `0xcc00`
- `main`'s `d` variable is stored at address `0xdd00`

```
int parameterMystery4(int& a, int* b, int c) {
    a--;
    c = c * 2;
    cout << c << " " << a << " " << *b << endl;
    (*b)++;
    return a + c;
}

int main() {
    int a = 2;
    int b = 5;
    int c = -3;
    int d;

    parameterMystery4(a, &b, c);
    d = parameterMystery4(c, &a, b);
    parameterMystery4(b, &d, a);

    cout << a << " " << b << " " << c << " " << d << endl;

    return 0;
}
```

2. File I/O and Strings (write)

Write a function named **`inputStats`** that accepts a `string` parameter representing a file name, then opens/reads that file's contents and prints information to the console about the file's lines. Report the length of each line, the number of lines in the file, the length of the longest line, and the average characters per line. You may assume that the input file has at least one line of input. For example, if the input file contains the following data:

```
Beware the Jabberwock, my son,
the jaws that bite, the claws that catch,

Beware the JubJub bird and shun
the frumious bandersnatch.
```

Your function should produce the following console output:

```
Line 1 has 30 chars
Line 2 has 41 chars
Line 3 has 0 chars
Line 4 has 31 chars
Line 5 has 26 chars
5 lines; longest = 41, average = 25.6
```

You may assume that the file contains at least 1 line of input.

If the input file does not exist or is not readable, your function should print no output.

3. ADTs / Collections (read)

Consider the following function:

```
void collectionMystery4(Vector<int>& v) {
    for (int i = 1; i < v.size(); i += 2) {
        if (v[i - 1] >= v[i]) {
            v.remove(i);
            v.insert(0, 0);
        }
    }
    cout << v << endl;
}
```

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

Vector	Output
a) `{10, 20, 10, 5}`	
b) `{8, 2, 9, 7, -1, 55}`	
c) `{0, 16, 9, 1, 64, 25, 25, 14, 0}`	

4. ADTs / Collections (write)

Write a function named **`starters`** that accepts two parameters: a reference to a vector of strings, and an integer *k*. Your function should examine the strings in the vector passed and return a set of all first characters that occur at least *k* times. In other words, if *k* or more strings in the vector start with a particular character at index 0 of the string (case-insensitively), that character should be part of the set that you return. All elements of your set should be in lowercase.

Consider a vector variable called `v` containing the following elements:

```
{"hi", "how", "are", "He", "", "Marty!", "this", "morning?", "fine.", "?foo!", "", "HOW", "A"}
```

Two words in the vector start with `"a"`, one starts with `"f"`, four start with `"h"`, two start with `"m"`, one starts with `"t"`, and one starts with `"?"`. Therefore the call of `starters(v, 2)` should return a set containing:

```
{'a', 'h', 'm'}
```

The call of `starters(v, 3)` on the same vector should return a set containing:

```
{'h'}
```

If no start character occurs *k* or more times, return an empty set. The characters should appear in your set in alphabetical order. Note that some of the elements of the vector might be **empty strings**; empty strings have no first character, so your code should not consider them when counting. (But your code shouldn't crash on an empty string.)

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

- You will need to construct your set to be returned, and in addition to that, 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 strings in the vector.
- You should not modify the contents of the vector 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 = 0; i < N; i++) {`
`    sum++;`
`}`
`for (int i = 100*N; i >= 0; i--) {`
`    sum++;`
`}`
`cout << sum << endl;`
	O(__________)

b)
`int sum = 0;`
`for (int i = 1; i < N - 2; i++) {`
`    for (int j = 0; j < N * 3; j += 2) {`
`        for (int k = 0; k < 1000; k++) {`
`            sum++;`
`        }`
`    }`
`}`
`cout << sum << endl;`
	O(__________)

c)
`Vector<int> v;`
`for (int i = 0; i < N; i++) {`
`    v.add(i);`
`}`
`while (!v.isEmpty()) {`
`    v.remove(0);`
`}`
`cout << "done!" << endl;`
	O(__________)

d)
`Set<int> set;`
`for (int i = 0; i < N/2; i++) {`
`    set.add(i);`
`}`
`Stack<int> stack;`
`for (int i = 0; i < N/2; i++) {`
`    set.remove(i);`
`    stack.push(i);`
`}`
 	
`cout << "done!" << endl;`
	O(__________)

e)
`Queue<int> queue;`
`for (int i = 1; i <= N; i++) {`
`    queue.enqueue(i * i);`
`}`
`HashMap<int, int> map;`
`while (!queue.isEmpty()) {`
`    int k = queue.dequeue();`
`    map.put(k, N * N);`
`}`
`cout << "done!" << endl;`
	O(__________)

6. Recursion (read)

For each of the calls to the following recursive function below, indicate what output is printed:

```
void recursionMystery4(int n) {
    if (n <= 1) {
        cout << "*";
    } else if (n == 2) {
        recursionMystery4(n - 1);
        cout << "*";
    } else {
        cout << "(";
        recursionMystery4(n - 2);
        cout << ")";
    }
}
```

Call	Output
a) `recursionMystery4(2);`	
b) `recursionMystery4(3);`	
c) `recursionMystery4(4);`	
d) `recursionMystery4(6);`	
e) `recursionMystery4(9);`	

7. Recursion (write)

Write a recursive function `indexOf` that accepts two strings *s1* and *s2* as parameters and that returns the starting index of the first occurrence of the second string *s2* inside the first string *s1*, or `-1` if *s2* is not found in *s1*. The table below shows several calls to your function and their expected return values. If *s2* is the empty string, always return index `0` regardless of the contents of *s1*. If *s2* is longer than *s1*, it of course cannot be contained in *s1* and therefore your function would return `-1` in such a case. Notice that case matters; the last example returns `-1`.

Call	Value Returned
`indexOf("Barack Obama", "Bar")`	`0`
`indexOf("foo", "foo")`	`0`
`indexOf("Stanford CS", "ford")`	`4`
`indexOf("Barack Obama", "ack")`	`3`
`indexOf("Barack Obama", "a")`	`1`
`indexOf("sandwich", "")`	`0`
`indexOf("Barack Obama", "McCain")`	`-1`
`indexOf("Barack Obama", "ACK")`	`-1`

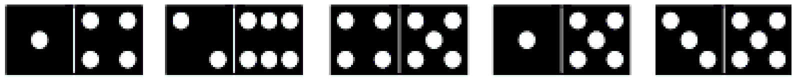
Strings have member functions named `find` and `rfind`, but you should not call them, because they allow you to get around using recursion. Similarly, the `replace` member is forbidden. You should limit yourself to using only the following string members *(see reference sheet for details of each member if necessary)*:

- `at`, `append`, `compare`, `erase`, `insert`, `length` or `size`, `substr`, `trim`, operators such as `[]`, `==`, `!=`, `<`, etc.

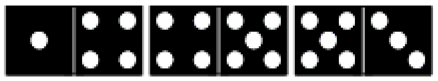
Do not construct any data structures (no array, vector, set, map, etc.), and do not declare any **global variables**. Also, **do not use loops**; you must use recursion. You are allowed to define other "**helper**" functions if you like.

8. Recursive Backtracking (write)

Write a recursive function named `chainExists` that looks for domino number chains. The game of Dominoes is played with rectangular pieces composed of two connected squares, each of which is marked with a certain number of dots. For example, each of the following five rectangles represents a domino:



Dominoes are connected end-to-end to form chains, subject to the condition that two dominoes can be linked together only if the numbers match, although it is legal to rotate dominoes 180° so that the numbers are reversed. For example, you could connect the first, third, and fifth dominoes in the above collection to form the following chain. Note that the 3-5 domino had to be rotated so that it matched up correctly with the 4-5.



Given a set of dominoes, an interesting question to ask is whether it is possible to form a chain starting at one number and ending with another. For example, the example chain shown earlier makes it clear that you can use the original set of five dominoes to build a chain starting with a 1 and ending with a 3. Similarly, if you wanted to build a chain starting with a 6 and ending with a 2, you could do so using only one domino. On the other hand, there is no way—using just these five dominoes—to build a chain starting with a 1 and ending with a 6.

Dominoes can be represented in C++ as a structure with a pair of integers. Assuming the type `Domino` is defined as:

```
struct Domino {
   int first;
   int second;
};
```

For this problem, write a function:

```
bool chainExists(const Vector<domino>& dominoes, int start, int end)
```

that returns `true` if it is possible to build a chain from start to finish using any subset of the dominoes in the dominoes vector. To simplify the problem, assume that `chainExists` always returns `true` if start is equal to finish, because you can trivially connect any number to itself with a chain of zero dominoes. (Don't worry about what the chain is; worry only about the yes or not that comes back in the form of a `bool`.) For example, if `dominoes` is the domino set illustrated above, `chainExists` should produce the following results:

```
   chainExists(dominoes, 1, 3) → true
   chainExists(dominoes, 5, 5) → true
   chainExists(dominoes, 1, 6) → false
```

If the value of *start* or *end* passed is negative, your function should throw an integer exception.

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 **`removeDuplicates`** to be added to the `ArrayList` class from lecture. Your function should remove any duplicate elements from the list, such that after a call to your function, the list will contain a set of unique elements. To simplify the problem, we will assume that the list is **sorted**, so that all duplicates occur consecutively.

Suppose an `ArrayList` variable named `list` stores the following values:

```
{7, 7, 18, 18, 18, 18, 21, 39, 39, 42, 42, 42}
```

After a call of `list.removeDuplicates();` , the list should store the following values:

```
{7, 18, 21, 39, 42}
```

If the list is empty or does not contain any duplicate values, calling your function should have no effect. You may call other private member functions of the `ArrayList` if you like, but not public ones.

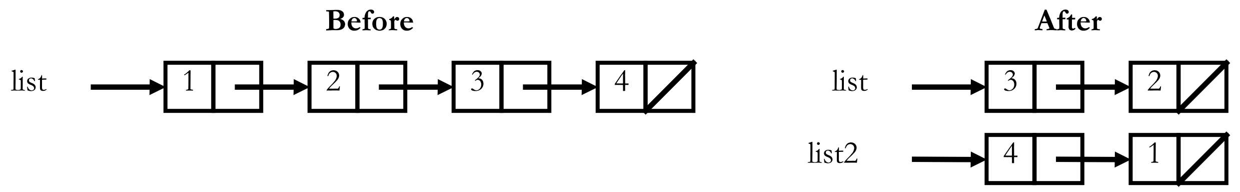
You should write the member function's body as it would appear in `ArrayList.cpp`. You do not need to write the function's header as it would appear in `ArrayList.h`. Write only your member function, not the rest of the class. Remember that you are adding this as a member function to the `ArrayList`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

```
class ArrayList {
public:                                      private:
    void add(int value);                         int* elements;
    void clear();                                int mysize;
    int get(int index) const;                    int capacity;
    void insert(int index, int value);           void checkCapacity();
    bool isEmpty() const;                        void checkIndex(int i, int min, int max);
    void remove(int index);
    void set(int index, int value);
    int size() const;
    string toString() const;
}
```

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.



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
};
```