CS106B Instructor: Cynthia Bailey

PRACTICE MIDTERM EXAM #2

Name: SUID: *(numeric)*

Problem	0	1	2	3	4	TOTAL
Topic	Write Sunet on every page	ADTs I (short answer)	ADTs II (coding)	Recursion (coding)	Big-O (short answer)	
Points	1	6	15	14	12	56

Instructions:

- **The time for this exam is 2 hours**, or 120 minutes. **Time limits are strict**, and you are expected to do *all* writing (including, e.g., writing SUID on each page), *before* time is called, or lose all points.
- Use of anything other than a pencil, eraser, pen, and one 8.5x11 page (two sides) of notes, is prohibited. In particular, no digital devices of any kind are permitted.
- Scratch writing areas are included in the exam, and **no additional scratch paper is permitted**.
- **Write your SUID on each page**, in case pages become separated during scanning.
- Please **write in boxes provided**, or the automated grading software in Gradescope will not see your response.
- For coding problems, you may assume `#include` and function prototypes already exist for you as needed.

SUID:

1. **ADTs I (6pts).** Consider the following code, written using Stanford library ADT implementations:

```
void collectionMystery(Queue<int>& q) {
    Vector<int> v;
    while (!q.isEmpty()) {
        v.add(q.dequeue());
    }
    cout << q << endl;  // output 1
    Stack<int> s;
    for (int i = 0; i < v.size(); i++) {
        q.enqueue(v[s.size()]);
        s.push(q.peek());
    }
    while (!q.isEmpty()) {
        s.push(q.dequeue());
    }
    cout << s << endl;  // output 2
    while (s.peek() % 3 != 0) {
        s.pop();
    }
    while (!s.isEmpty()) {
        q.enqueue(s.pop());
    }
    cout << q << endl;  // output 3
}
```

Now write the output of the above code, given the following inputs. As a reminder, this is what the Stanford Library Documentation webpages say about the format when you `cout <<` these structures:

- `Queue`: The elements are listed left-to-right from the front of the queue to the back, such as `{value1, value2, value3}`.
- `Stack`: The output is in the form `{value1, value2, value3}` where elements are listed left-to-right from the bottom of the stack to the top.
- `Vector`: The output is in the form `{value1, value2, value3}` where elements are listed in order of ascending index.

Input: `q = {1, 3, 9, 7, 5}`

Output 1:

Output 2:

Output 3:

SUID:

---

2. **ADTs II (15pts).** Our CS106B homework submission system (Paperless) allows you to re-submit as often as you like, and only your most recent submission is graded. *Pro Tip: submit your code as soon as it is partially working, and resubmit it improves, so you'll have a "safety" if something terrible were to happen and you lost all your work on your personal computer!* Now imagine if we made Paperless keep track of each student's submissions in a way that allows students to go back to earlier versions, using a trio of functions as follows (the **TYPE** placeholder is explained below):
   - `void submitCode(`**TYPE**`& storage, const string& sunet, const string& code);`
     - Adds the submission code to the submissions for the student identified by `sunet`.
   - `string seeMostRecent(const` **TYPE**`& storage, const string& sunet);`
     - Retrieves the code of the most recent submission for that student.
   - `bool discardMostRecent(`**TYPE**`& storage, const string& sunet);`
     - Discards the most recent submission for the student, reverting to the most recent of any previous submission(s) made by that student.

   The role of the `storage` parameter in each of the functions above is to hold all the submissions (except those that have been discarded) for all the students. You'll notice that `storage`'s type is a placeholder **TYPE**. This is because we want you to demonstrate that you can *select the best ADT(s) for the given task*, so you will define what **TYPE** should be, and then implement the three functions accordingly.

   (a) (3pts) Write your chosen **TYPE** for the variable `storage` in the box below. Your choice will apply to all places where **TYPE** appears, and must work without editing any of the provided code below or any of the provided function signatures. For full credit on part (a), your choice for **TYPE** should not only work, but also show good style and discernment in matching the concept the chosen ADT(s) represent to the task.

```
void runSubmitSystem() {
    [                                                    ] storage;
    while (true) {
        int choice = getInteger("Choose (1)submit, (2)see most recent, or (3)discard: ");
        string sunet = getLine("Enter sunet: ");
        if (choice == 1) {
            string code = getLine("Paste submission here: ");
            submitCode(storage, sunet, code);
        } else if (choice == 2) {
            cout << "Submission:" << endl << seeMostRecent(storage, sunet) << endl;
        } else if (choice == 3 && discardMostRecent(storage, sunet)) {
            cout << "Successfully discarded submission." << endl;
        } else {
            cout << "No action." << endl;
        }
        // output count of students with at least one submission currently stored
        cout << storage.size() << " students have a submission." << endl;
    }
}
```

SUID:

Complete the three functions below.

- Assume that whatever you wrote in the box in part (a) is now written in place of **TYPE** in all the function signatures below.
- Note the last `cout` line in the code in part (a). Be sure to maintain your `storage` such that its `size()` method will always produce the intended value for that print statement.

(b) (3pts) Adds the submission code to the submissions for the student identified by `sunet`.

```
void submitCode(TYPE& storage, const string& sunet, const string& code) {




}
```

(c) (4pts) Retrieves the code of the most recent submission for that student. If the student does not have any submissions, returns the empty string.

```
string seeMostRecent(const TYPE& storage, const string& sunet) {









}
```

SUID:

(d) (5pts) Discards the most recent submission for the student, reverting to the most recent of any previous submission(s) made by that student. If this was the only remaining submission for this student, make sure that this student will *not* be counted among those students who have a submission (see `cout` code in part (a)). Returns `true` if a submission was discarded, else `false`.

```
bool discardMostRecent(TYPE& storage, const string& sunet) {




}
```

[scratch space]

SUID:

---

3. **Recursion (14pts).** Write a recursive function `makeWords` that finds English words can be formed using only consonants from a provided subset of consonants, and unlimited vowels. The function takes as input a Lexicon containing all valid English words, and a set of characters you may use to form words (includes allowable consonants, and all vowels). Of the provided letters, each consonant may be used at most once, and each vowel may be used zero up to many times. The function takes an empty Lexicon `result` by reference, and when the function returns, `result` should contain all the English words that can be formed.
   - You may assume that everything given to you is all in all lower case.
   - Use Lexicon's `containsPrefix()` to **prune your search**.
   - Your solution must not use any global variables, and must be fundamentally recursive in approach.
   - There is no specific Big-O requirement, but excessively repetitious execution may lose points.
   - You must use the provided function signature, but you may want to make a recursive helper function.
   - For your convenience, assume this function exists for you to use: `bool isVowel(char c)`.
   - Ex: If `letters = {'a', 'e', 'i', 'm', 'o', 't', 'u'}`, then you can spell words like "meat", "team", "meet", and "too"; but you can't spell words like "mitt" (uses two t's) or "meant" (uses n).
   - Write you solution on the next page (including the recursive helper function). You may use the space below as scratch space to plan your solution.

[scratch space]

SUID:

```
void makeWords(const Lexicon& english, Set<char> letters, Lexicon& result) {
```

SUID:

4. **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

`void loopStuff(string& str) {`
`  int N = str.size();`
`  for(int i = 0; i < N; i++) {`
`  cout << str[i] << endl;`
`  }`
`  for(int i = N - 1; i >= N / 2; i--) {`
`  cout << str[i] << endl;`
`  }`
`}`
	O( )

`void printStuff(int N) {`
`  recurse1(N);`
`}`
`void recurse1(int level) {`
`  if (level < 0) {`
`  return;`
`  }`
`  recurse1(level – 1);`
`  cout << level << endl;`
`  recurse1(level – 1);`
`}`
	O( )

`int vectorStuff(Vector<int>& vec) {`
`  int N = vec.size();`
`  for (int i = 0; i < N * 2; i++) {`
`  vec.insert(0, vec[0]);`
`  }`
`}`
	O( )

`void moreRecursiveStuff(string& str) {`
`  int N = str.size();`
`  recurse2(str, N - 1);`
`}`
`void recurse2(string& str, int level) {`
`  if (level < 0) return;`
`  cout << str[level] << endl;`
`  recurse2(str, level – 1);`
`}`
	O( )

Hint for recursion questions: it may help to draw a call tree and/or trace using stack frames.

[scratch space]