CS106X Midterm Examination

This is an open-book, open-note, closed-electronic-device exam. You have 90 minutes to complete the exam. If you’re taking the exam remotely, you can telephone Jerry at 415-205-2242 to ask questions.

Good luck!

Section Leader: ___________________

Last Name: ___________________

First Name: ___________________

SUNet ID: ___________________@stanford.edu

I accept the letter and spirit of the honor code.

(signed) __________________________________________________________

Score  Grader

2. Up-Down Words [10]  ____  ____
3. NEWS Mazes [10]  ____  ____

Total [30]  ____  ____
Summary of Relevant Data Types

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

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

class Map {
    bool isEmpty() const;
    int size() const;
    void put(const Key& key, const Value& value);
    bool containsKey(const Key& key) const;
    Value get(const Key& key) const;
    Value& operator[](const Key& key);
};

class Set {
    bool isEmpty() const;
    int size() const;
    void add(const Type& elem); // operator+= also adds elements
    bool contains(const Type& elem) const;
};

class Lexicon {
    bool contains(const string& str);
    bool containsPrefix(const string& str);
    void add(const string& word);
};

class Grid {...} // only operator[][] is relevant
Problem 1: Campaign Donations and Kickbacks [10 points]
When a politician’s supporters donate money to help support their campaign, it’s important to ensure their supporters aren’t compensated for their donations in any way.

Consider the case of a county’s tax assessor: an elected official who, among his or her many responsibilities, manages property tax assessments and the property owners’ appeals to have those taxes reduced. Let’s assume that a preliminary investigation of this tax assessor suggests that a suspiciously large fraction of his or her financial donors have received tax reductions.

A more complete investigation would entail a comparison of all those who donated money to those whose property tax liabilities were reduced. If the percentage of financial donors receiving reductions is statistically greater than the percentage of the general applicant pool receiving reductions, that might be grounds for a more thorough investigation.

For the purposes of this problem, assume we have access to two key collections of information:

- a Set<string> called donors, which stores the names of all those who donated money to help support the election of a county’s tax assessor, and
- a Map<string, bool> called appeals, which stores the names of all county residents who appealed to have their property taxes reduced, where the keys are the names of those who filed appeals, and the values indicated whether a particular person’s appeal was approved (true) or denied (false). Note some donors may not have appealed their property tax assessments and would not be present in the supplied map. Assume property tax reductions only result because of an appeal process—that is, no one’s property taxes are reduced unless an appeal is filed.

Implement a function called analyze, which accepts references to donors, appeals, and two doubles called f1 and f2, and populates f1 and f2 with the fraction of those receiving tax reductions who donated money to the assessor’s campaign and the general fraction of those receiving tax reductions, respectively. Use the next page for your implementation.
void analyze(const Set<string>& donors, const Map<string, bool>& appeals, double& f1, double& f2) {

[Problem 1 continued]
Problem 2: Up-Down Words [10 points]

An up-down word is an English word where all letters at odd indices are alphabetically larger than their neighbors. So, "prevalence", for instance, is an up-down word, because the alternating letters are 'r', 'v', 'l', 'n', and 'e' are further along in the alphabet that their immediate neighbors—that is, 'r' is greater than both 'p' and 'e', 'v' is greater than 'e' and 'a', etc. A down-up word is defined similarly, except that all letters at odd indices are alphabetically smaller than their neighbors. And to be clear, words with double letters—"ebb", "jammer", "gross", "stiff", etc—are neither down-up nor up-down.

Write the recursive list function that exhaustively generates and prints all of the up-down (or down-up) words—one per line—beginning with the supplied letter and direction. You may not iterate over the Lexicon, but you may use containsPrefix and contains to prune the recursion and confirm a string is a word.

Sample calls:

- `list(english, 'p', true)` would print all of the up-down words beginning with 'p'. "praline", "prevalence", "purveyor", and "pyosis" would be among the words generated.
- `list(english, 'b', false)` would print all of the down-up words beginning with 'b'. "balanced", "bananas", "barnyards", and "bawdy" would be among the words generated.
- `list(english, 'z', false)` would print all of the down-up words beginning with 'z'. "zenith", "zero", "zilch", and "ziti" would be among the words generated.
- `list(english, 'z', true)` and `list(english, 'a', false)` would print absolutely nothing.

Use the next page to present your implementation.
Problem 2: Up-Down Words [continued]

    void list(const Lexicon& english, char letter, bool up) {


Problem 3: NEWS Puzzles [10 points]

NEWS puzzles are maze-like constructions where the goal is to navigate from one location to another. You’d be free to wander up, down, left, and right as you please, except that moves are constrained by a set of rules. Some are obvious, but some aren’t. Here they are:

- If you find yourself in a location labeled with a N, E, W, or S, then you must move north, east, west, or south, respectively.
- If you find yourself in a location labeled with a +, then you can move in any of the four major compass directions.
- If you find yourself in a location that isn’t labeled, then that location is considered a bridge, and you must pass through that location in the same direction you entered—that is, if you move east into an unlabeled location, you’re moving east again.

The puzzle itself is modeled as a `Grid<char>`, where the `char` is either 'N', 'E', 'W', 'S', '+', ' ' (the space character), or 'X' (which means the location isn’t part of the maze).

Pair the above with the definition of a `coord`:

```c
struct coord {
    int row, col;
};
```

and you’re more than outfitted to implement a recursive backtracking routine which decides whether a path from one location to a second exists. For simplicity, you can assume that:
• the `coord` type magically works with all relational operators, including `<` and `==`. That means you can store a set of `coords` to keep track of what locations you’ve visited. That way, you can ensure you never get trapped in any infinite loops.

• the initial location houses either an N, E, W, S, or a +, so there’s no confusion as to how you can initiate a search.

Using the next two pages, implement the `solve` routine, which returns `true` if and only if one can travel from `start` to `finish` while respecting the rules. You are required to use recursive backtracking. You needn’t return the path—just the `true` or `false`.

Here are three helper functions you’re encouraged to use for your answer:

```cpp
static coord next(coord curr, char rule) { // assume (0,0) is upper left
    switch (rule) {
        case 'N': curr.row--; break;
        case 'E': curr.col++; break;
        case 'W': curr.col--; break;
        case 'S': curr.row++; break;
    }
    return curr;
}

static bool isValid(const Grid<char>& maze, const coord& c) {
    return maze.inBounds(c.row, c.col) && maze[c.row][c.col] != 'X';
}

static string getOptions(char prev, char curr) {
    switch (curr) {
        case ' ': return charToString(prev); // prev determines next move
        case '+': return "NEWS"; // four major compass directions
        default: return charToString(curr); // only option is supplied as curr
    }
}
```
Problem 3: NEWS Puzzles [continued]

    bool solve(const Grid<char>& maze, const coord& start, const coord& finish) {
Problem 3: NEWS Puzzles [continued]