CS106B
Spring 2016 Cynthia Lee

Section 2 (Week 3) Handout

Section problem authors include Marty Stepp and Jerry Cain.

Problem 1: Twice (Sets)

Write a function named **`twice`** that takes a vector of integers and returns a set containing all the numbers in the vector that appear exactly twice. Bonus: do the same thing, but you are not allowed to declare any kind of data structure other than sets.

```
Set<int> twice(Vector<int>& numbers) { ...
```

Problem 2: Reverse Map (Maps)

Write a function named **`reverseMap`** that takes a map from `int`s to `string`s, and returns a map with the associations reversed.

For example, if a Map variable named map stores `{ 1: "a", 2: "b", 3: "c" }`, the call of `reverseMap(map)` should return `{ "a": 1, "b": 2, "c": 3 }`. If there are duplicate values `(k1, v)` and `(k2, v)` in the original map, your returned map may contain either `(v, k1)` or `(v, k2).`

```
Map<string, int> reverseMap(Map<int, string>& map) { ...
```

Problem 3: Cannonballs (Recursion)

Write a function named **`cannonballs`** that returns the number of cannonballs in a square pyramid of cannonballs of the given height. For example, in a square pyramid of height 3, the bottom layer has 9 balls, the middle layer has 4, and there is one ball on top, so `cannonballs(3)` returns 14. (You can assume that height won't be negative).

```
int cannonballs(int height);
```

Problem 4: Reverse String (Recursion)

Write a function named **`reverseStr`** that takes a string and reverses it. For example, reversing "Hello World" returns "dlroW olleH".

```
string reverseStr(string s);
```

Problem 5: Twiddles (Recursion)

Two English words are considered *twiddles* if the letters at each position are either the same, neighboring letters, or next-to-neighboring letters. For instance, **sparks** and **snarls** are twiddles. Their second and second-to-last characters are different, but **p** is just two past **n** in the alphabet, and **k** comes just before **l**. A more dramatic example: **craggy** and **eschew**. They have no letters in common, but **craggy**'s **c**, **r**, **a**, **g**, **g**, and **y** are **-2**, **-1**, **-2**, **-1**, **2**, and **2** away from the **e**, **s**, **c**, **h**, **e**, and **w** in **eschew**. And just to be clear, **a** and **z** are **not** next to each other in the alphabet—there's no wrapping around at all.

Write a recursive procedure called **listTwiddles**, which accepts a string **str** and a reference to an English language **Lexicon**, and prints out all those English words that just happen to be **str**'s twiddles. You'll probably want to write a wrapper function. (Note: any word is considered to be a twiddle of itself, so it's okay to print it.)

```
static void listTwiddles(const string& str, const Lexicon& lex);
```

Problem 6: Making Change (Recursion)

For this problem, implement the following function:

```
static int countWaysToMakeChange(const Vector<int>& denominations, int amount)
```

The **countWaysToMakeChange** routine recursively computes the number of ways to make change for the specified amount given an unlimited number of coins of the specified denominations. Download the lab starter code to work with the small test harness to exercise your implementation. The test harness includes the following **main** function:

```
int main() {
    Vector<int> denominations;
    denominations += 25, 10, 5;
    cout << "Number of ways to make change for a dollar using " << denominations
         << ": " << countWaysToMakeChange(denominations, 100) << endl;
    denominations += 1;
    cout << "Number of ways to make change for a dollar using " << denominations
         << ": " << countWaysToMakeChange(denominations, 100) << endl;
    return 0;
}
```

Once properly implemented, the above **main** function should output the following:

```
Number of ways to make change for a dollar using {25, 10, 5}: 29
Number of ways to make change for a dollar using {25, 10, 5, 1}: 242
```

Of course, you're free to cannibalize the test harness in any way you'd like if it'll help confirm your implementation is solid.