

CS106X Midterm Examination

This is an open-note exam. You can refer to any course handouts, handwritten lecture notes, and printouts of any code relevant to a CS106X assignment. You may not use any laptops, cell phones, or handheld devices of any sort.

Anyone taking the exam remotely can call in to ask questions: 415-205-2242. Once remote students are done, they should fax all pages to 650-723-6092.

Good luck!

Section Leader: _____

Last Name: _____

First Name: _____

I accept the letter and spirit of the honor code. I've neither given nor received aid on this exam. I pledge to write more neatly than I ever have in my entire life.

(signed) _____

	Score	Grader
1. Publishing Stories	(8) _____	_____
2. Numeric Palindromes	(12) _____	_____
3. Character Swaps	(10) _____	_____
4. Linked List Fun	(15) _____	_____
Total	(45) _____	_____

SCPD students who want their exams sent back through regular mail, check here: _____

Problem 1: Publishing Stories (8 points)

Social networking sites like Facebook, LinkedIn, Orkut, and MySpace typically record and publish stories about actions taken by you and your friends. Stories such as:

John Dixon accepted your friend request.
 Jeff Barbose is no longer in a relationship.
 Scott James wrote a note called "The Two Percent Solution".
 Arlene Heitner commented on Melodie Bowsher's video.
 Antonio Melara gave The French Laundry a 5-star review.

are created from story templates like

{name} accepted your friend request.
 {name} is no longer in a relationship.
 {name} wrote a note called "{title}".
 {name} commented on {target}'s video.
 {actor} gave {restaurant} a {rating}-star review.

The specific story is generated from the skeletal one by replacing the tokens—substrings like "**{name}**", "**{title}**", and "**{rating}**"—with event-specific values, like "**John Dixon**", "**The Two Percent Solution**", and "**5**". The token-value pairs can be packaged in a **Map<string>**, and given a story template and a data map, it's possible to generate an actual story.

Write the **SubstituteTokens** function, which accepts a story template (like "**{actor} gave {restaurant} a {rating}-star review.**") and a **Map<string>** (which might map "**actor**" to "**Antonio Melara**", "**restaurant**" to "**The French Laundry**", and "**rating**" to "**5**"), and builds a string just like the story template, except that the tokens have been replaced by the text they map to.

Place your implementation on the next page, and feel free to rip this page out so you can easily refer to it. Assume the following is true:

- '**{**' and '**}**' exist to delimit token names, but won't appear anywhere else. In other words, if you encounter the '**{**' character, you can assume it marks the beginning of a token that ends with a '**}**'.
- We guarantee that all tokens are in the **Map<string>**. You don't need to do any error checking.
- Note that the '**{**' and the '**}**' aren't included in the map. "**{name}**", for instance, is replaced with whatever "**name**" identifies in the map.

Problem 1 (continued)

```
string SubstituteTokens(string storyTemplate, Map<string>& data)
{
```

Problem 2: Numeric Palindromes (12 points)

Write a function called **CountPalindromes(n, multiple)**, which returns the number of **n**-digit palindromes (numbers that read the same forwards and backwards) that are a multiple of **multiple**. For instance, **CountPalindromes(3, 6)** returns 13, because 606, 414, 222, 828, 636, 444, 252, 858, 666, 474, 282, 888, and 696 are all three-digit palindromes that just happen to be divisible by 6. (000, 030, 060, and 090 don't count, because we exclude numbers with leading zeroes.)

The brute force approach would just for loop over all digit numbers and count all multiples of multiple that just happen to be palindromes. But that's a very time consuming approach, since a vast majority of numbers will not be palindromes. A smarter approach—and one that you **must use**—is to recursively generate all numeric palindromes from the center out, and count those and only those that are **n** digits and a multiple of **multiple**. By taking this approach, you only deal with palindromes and nothing else, and it's much, much faster.

Another example: **CountPalindromes(6, 211)** returns 4, because 522225, 496694, 807708, and 259952 are the only six-digit numbers that just happen to be multiples of 211.

Write the **CountPalindromes** function on the next page. Feel free to tear this page out so you can easily refer to it. But, be aware of the following:

- Your implementation must use recursion to build out all possible palindromes from the center digit or digits.
- You should use **StringToInteger** and **IntegerToString** to convert back and forth between numbers and their string forms. It's easier to built palindromes using string concatenation, but it's easier to do modulo math with integers.
- Don't allow numbers with leading zeroes to be considered palindromes.
- Don't worry about the maximum value an **int** can store. Just assume an **int** can store arbitrarily large numbers.
- You must handle all positive values of **n** and **multiple**, both even and odd.

Problem 2 (continued)

```
int CountPalindromes(int n, int multiple)
{
```

Problem 3: Recursive Backtracking and Character Exchanges (10 points)

Write a function **Morph**, which returns **true** if and only if the **first** word can be morphed into a **second** through a series of at most **n** character exchanges, where all intervening strings are also words in the English language. To be clear, a character exchange is an exchange of two characters in different positions within a given word to form another word. So, **Morph("traces", "carets", english, 3)** should return **true**, as the following path demonstrates:

```
traces
crates
cartes
carets
```

"crates" is formed from **"traces"** by swapping the **'c'** and the **'t'**, **"cartes"** (it's really a word) is formed from **"crates"** by swapping the **'a'** and the **'r'**, and **"carets"** is formed from **"cartes"** by swapping the **'t'** and the **'e'**.

As it turns out, that's the shortest such path between the two words, so that **Morph("traces", "carets", english, 2)** would return **false**.

Use the next page to implement the **Morph** function, which uses recursive backtracking to decide if two words are connected via no more than **n** character exchanges. Your function only needs to return **true** or **false**. It does **not** need to disclose what the actual path between the two words is. Feel free to rip this page out so you can easily refer to it.

Problem 3: Continued

```
bool Morph(string first, string second, Lexicon& english, int n)
{
```

Problem 4: Fun With Linked Lists (15 points)

Use the following data structure for both parts of this problem:

```
struct node {
    int value;
    node *next;
};
```

- a. (5 points) Write the **Intermingle** function, which takes a linked list of integers and inserts a new node in between every one of the originals, where the value in each new node is the difference between the numbers on either side of it.

Here are some examples:

list	list after call Intermingle(list)
1 → 4 → 2	1 → -3 → 4 → 2 → 2
3	3
6 → 1 → 1 → 1 → 2	6 → 5 → 1 → 0 → 1 → 0 → 1 → -1 → 2

```
void Intermingle(node *list)
{
```

Problem 4 (continued)

- b. (10 points) Write the **Cluster** function, which updates a linked list of integers so that all instances of the same number are grouped together while retaining the order in which each number first appears. For instance, the list:

Before: $4 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 5 \rightarrow 2 \rightarrow 2 \rightarrow 1 \rightarrow 5$

would become

After: $4 \rightarrow 4 \rightarrow 1 \rightarrow 1 \rightarrow 5 \rightarrow 5 \rightarrow 5 \rightarrow 2 \rightarrow 2$

Note that the original list introduces 4, then 1, then 5, and then 2. That means the transformed list brings all 4s to the front, followed by all 1s, followed by all 5s, followed by all 2s.

Some more examples:

list	list after call Cluster(list)
$8 \rightarrow 7 \rightarrow 14 \rightarrow 8 \rightarrow 7$	$8 \rightarrow 8 \rightarrow 7 \rightarrow 7 \rightarrow 14$
7	7
$4 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 4$	$4 \rightarrow 4 \rightarrow 4 \rightarrow 4 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow 5 \rightarrow 3$

Use the next page for your implementation (and feel free to rip this page out so you can easily refer to it.) You can take as many passes over the list as you need to to get the desired transformation.

Problem 4 (continued)

```
void Cluster(node *list)
{
```