CSE250, Spring 2014 Prelim Im Name:_____
Mar. 11, 2014 St.ID#:_____

Closed book, closed-notes-except-for-1-sheet, closed neighbors, 48 minutes. This question paper has FOUR pages. Do ALL THREE questions **on this exam paper**. Please *show all your work*; this may help for partial credit. The exam totals 67 pts., subdivided as shown.

(1) (24 pts. total)

Given the following declarations and parts of classes, say which of the following statements are legal. They are faithful to Assignments 2–4 and lecture notes.

```
class StringWrap {
   string st;
 public:
   explicit StringWrap(string s) : st(s) { }
   StringWrap() : st("") { }
   string str() const { return st; }
   void alter(int i, char c) { st[i] = c; }
   int size() const { return st.length(); }
   ...
};

string word;
StringWrap sw(word);
StringWrap sw2(word);
StringWrap* const swcp;
const StringWrap* swpcd;
vector<StringWrap*> vpsw;
```

For each line write "Yes" if legal, "No" if not legal. If you think a line is legal in syntax but would be a null-pointer de-reference, consider that the pointer was initialized on a previous legal line. Each is 1 pt. except the last, separate question.

_____(a) `sw = new StringWrap(word);`

_____(b) `sw = StringWrap(word);`

_____(c) `sw = word;`

_____(d) `swcp = sw.str();`

_____(e) `swcp = &sw2;`

_____(f) `word = swcp->str();`

_____(g) `sw2.alter(0,'J');`

_____(h) `swcp.alter(0,'H');`

_____(i) `swcp->alter(0,'B');`

Repeated for convenience:

```
class StringWrap {
   string st;
 public:
   explicit StringWrap(string s) : st(s) { }
   StringWrap() : st("") { }
   string str() const { return st; }
   void alter(int i, char c) { st[i] = c; }
   int size() const { return st.length(); }
   ...
};

string word;
StringWrap sw(word);
StringWrap sw2(word);
StringWrap* const swcp;
const StringWrap* swpcd;
vector<StringWrap*> vpsw;
```

_____(j) swpcd->alter(0,'M');

_____(k) swcp = swpcd;

_____(l) int i = swcp->size();

_____(m) vpsw.at(0) = new StringWrap();

_____(n) vpsw.at(0) = sw;

_____(o) vpsw->at(0) = new StringWrap("Mello");

_____(p) vpsw.at(0) = word;

_____(q) vpsw.at(0) = StringWrap();

_____(r) (*vpsw)[0] = StringWrap("Yello");

_____(s) vpsw.push_back(swp);

_____(t) swp->st[0] = 'Y';

_____(u) vpsw = new vector<StringWrap*>(100);

(v) Which, if any, answer(s) would change if the "explicit" were removed from the first constructor?

---

(2) ($4 \times 4 = 16$ **pts.)**

For each code sketch labeled (a)–(d), find a function $g(n)$ such that the code's running time is "$O(g(n))$ and no faster," i.e. is $\Theta(g(n))$. The four deque push/pop operations and the `isEmpty()` and `size()` methods are all coded to take constant time, i.e. $O(1)$ time. Assume the declarations

```
Deque<int>* dq1;
Deque<int>* dq2;
vector<Deque<int>*> ladders = ...//contains n deques, each with n items
```

and that each of the n deques currently holds exactly $\log n$ integers. Please write your answers to the right of each part, in the form "$\Theta(\cdots)$" filling in your $g(n)$, followed by *a brief justification*—or if the code gives an error, say that.

```
(a) for (int i = 0; i < n; i++) {                    (a):
       dq = ladders->at(i);
       if (i < dq->popFront()) { dq->pushRear(i); }
    }
```

```
(b) int sum = 0;                                     (b):
    for (int i = 0; i < n; i++) {
       while(!ladders->at(i)->isEmpty()) {
          sum += ladders->at(i)->popRear();
       }
    }
```

```
(c) int sum = 0;                                     (c):
    for (int i = 0; i < n; i++) {
       for (int j = 0; j < i; j++) { //note difference from (b)
          sum += ladders->at(i)->popRear();
       }
    }
```

```
(d) int count = 0;                                   (d):
    for (int i = 0; i < n; i++) {
       for (int j = 0; j < i; j++) { //compare deques i and j
          dq1 = ladders->at(i);
          dq2 = ladders->at(j);
          while(!(dq1->isEmpty() || dq2->isEmpty())) {
             if (dq1->popFront() != dq2->popFront()) {
                count++;
             }
          } //end while
       } //end for j
    } //end for i
    return count; //total # of mismatches between every pair of deques.
```

(3) (27 pts.)

Say a string x is a *transpose* of a string y if y can be obtained from x by interchanging two adjacent characters. For example, `sung` is a transpose of `snug`, but `sung` is not a transpose of `guns` because the `g` and the `s` are not adjacent. Nor is `sung` a transpose of `sang` even though they have edit-distace 1, and only words of the same length can be transposes. Finally, note that `meet` is a transpose of `mete` by switching the last two letters, and *is* considered a transpose of itself by switching the two `e`'s.

Write in C++ a function `bool transpose(const string& x, const string& y)` which returns `true` if and only if `x` is a transpose of `y`. You may use C++ `string` library functions such as `substr` and `length`, and may use either `at` or the `operator[]` to refer to characters. Your code will have at least one loop and at least one `if(...)` test—you must write a comment explaining what it means if and when the loop executes to termination without being interrupted (say by a return statement), and what the `if(...)` being true means. End of Exam