Name [redacted]

CS 32
Winter 2012
Midterm 1
February 1, 2012

Problem #	Possible Points	Actual Points
1	24	24
2	11	10
TOTAL	35	[illegible]

35

STUDENT ID #: [redacted]

INSTRUCTOR: [X] Nachenberg [ ] Smallberg

SIGNATURE: [redacted]

OPEN BOOK, OPEN NOTES
NO ELECTRONIC DEVICES

ENJOY!

Name [redacted]

1. [24 points in all]

Recall this from the Map class you wrote; this is a Map from strings to ints:

```
typedef string KeyType;
typedef int ValueType;

class Map
{
  public:
    Map();
    bool insert(const KeyType& key, const ValueType& value);
    bool update(const KeyType& key, const ValueType& value);
    bool insertOrUpdate(const KeyType& key,
                                    const ValueType& value);
    bool erase(const KeyType& key);
    bool contains(const KeyType& key) const;
    bool get(const KeyType& key, ValueType& value) const;
    // other functions not shown
};
```

Consider this excerpt from a class representing a photo. For this problem, all we need to know is that every photo has a string indicating the subject of the photo and and int indicating the year the photo was taken:

```
class Photo
{
  public:
    Photo(string subj, int yr);
    string subject() const { return m_subject; }
    int year() const { return m_year; }
    // other functions not shown
  private:
    string m_subject;
    int m_year;
};
```

Assume that we have this global variable, accessible from anywhere in the program:

```
Map scm;  // short for subjectCountMap
```

This map is supposed to keep a count of the subject strings for every Photo object currently in existence in the program. If the first four Photo objects created were

```
Photo p1("Royce Hall", 2008);
Photo p2("bruin statue", 2012);
Photo p3("Royce Hall", 2008);
Photo p4("Royce Hall", 2011);
```

then scm would map ("bruin statue" to 1 and "Royce Hall" to 3. This, then, could be part of the implementation of the Photo constructor shown above:

```
Photo::Photo(string subj, int yr)
...
   int count = 0;
   scm.get(m_subject, count);  // count remains 0 if
                               // m_subject is not in map
   scm.insertOrUpdate(m_subject, count+1);
...
```

If we let the compiler generate the destructor, copy constructor, and assignment operator for the Photo class, the contents of scm would not always be accurate. (For example, consider this function:

```
void h()
{
    Photo p("inverted fountain", 1998);
}
```

If just before we called this function, scm contained no occurrences of "inverted fountain", then after returning from the function, scm would incorrectly contain one occurrence of that string; that's incorrect, because *after* returning from the function there are *no* existing Photo objects with that subject string, since the object p went away.) Therefore, you must declare and implement the destructor, copy constructor, and assignment operator.

In parts a, b, and c below, you may implement additional helper functions if you like. Write any helper function implementations in whichever of parts a, b, or c first uses it; you don't have to repeat its implementation if a later part also calls it. The scm map must never contain a string that maps to 0; if no Photo exists with a particular subject string, then that subject string should not be in the map at all.

a. [5 points]

Complete the implementation of the destructor for the Photo class:

don't need to check if get returns false since this photo should be mapped in scm

```
Photo::~Photo()
{
   int count=0;
   scm.get(m_string, count);
   if (count == 1) // the only photo w/that subject
      scm.erase(m_string);
   else
      scm.update(m_string, count-1);
}
```

5

b. [5 points]

Implement the copy constructor for the Photo class:

```
Photo:: Photo (const Photo & src)
{
    m_year= src.m_year;
    m_string= src.m_string;
    int count=0;
    if (scm.get(m_string, count))  // if such a Photo subject exists
       scm.Update (m_string, count+1);
    else
       scm.insert(m_string, 1);

}
```

(5)

c. [5 points]

Implement the assignment operator for the Photo class:

```
Photo& Photo:: operator = (const Photo & src)
{ if (&src ==this)
    return *this;
  int count = 0;
  scm.get(m_string,count)
  if (count== 1)
    scm.erase (m_string);
  else
    scm.update(m_string, count-1);
  m_year= src.m_year;
  m_string= src.m_string;
  scm.update(m_string, count+1); // not insert, since src map should
                                    already be mapped

  return *this;
}
```

(5)

d. [5 points]

Consider the following program. Correct answers to parts a, b, and c will result in the assertions being true. Assuming those correct answers, the program produces four lines of output.

```
Map scm;

int howmany(const Map& m, string s)
{
   int ct = 0;
   m.get(s, ct);  // ct remains 0 if s is not a key in m
   return ct;
}

void f(Photo p)
{
   cout << "Line 2: " << howmany(scm, "David") << " "
                      << howmany(scm, "Carey") << endl;
}

void g()
{
   Photo p1("David", 2012);
   Photo p2("Carey", 1995);
   Photo p3(p2);
   if (howmany(scm, "David") == 1)
   {
      Photo p4("David", 2008);
      p2 = p4;
      Map checkScm;
      checkScm.insert(p1.subject());
      checkScm.insert(p2.subject());
      checkScm.insert(p3.subject());
      checkScm.insert(p4.subject());
      assert(howmany(scm, "David") == howmany(checkScm, "David"));
      assert(howmany(scm, "Carey") == howmany(checkScm, "Carey"));
   }
   cout << "Line 1: " << howmany(scm, "David") << " "
                      << howmany(scm, "Carey") << endl;
   f(p1);
   cout << "Line 3: " << howmany(scm, "David") << " "
                      << howmany(scm, "Carey") << endl;
}

int main()
{
   g();
   cout << "Line 4: " << howmany(scm, "David") << " "
                      << howmany(scm, "Carey") << endl;
}
```

On the next page, write the four lines of output this program produces.

Complete the four lines of output the program on the previous page produces:

Line 1: 2 1
Line 2: 3 1
Line 3: 2 1
Line 4: 0 0

(5)

e. [4 points]

Here is a declaration for a Magazine class:

```
class Magazine
{
  public:
    Magazine(string t, string i, int pg, int yr, string subj);
    // other functions not shown
  private:
    string m_title;
    string m_issue;
    Photo m_coverPhoto;
    int m_pages;
};
```

Every magazine has a title, an issue, a cover photo, and a number of pages. We can construct magazines like this:

```
Magazine m1("Nerd World", "Feb 2012", 128, 1995, "Carey");
Magazine m2("The Chic Geek", "May 2011", 64, 2011, "David");
```

The constructor takes the magazine's title, its issue, its number of pages, the year its cover photo was taken, and the subject of its cover photo.

Write the implementation of the Magazine constructor below:

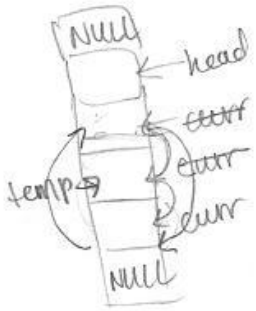
```
Magazine::Magazine(string t, string i, int pg, int yr, string subj)
  : m_coverPhoto(subj, yr), m_title(t), m_issue(i), m_pages(pg)
{
}
```

(4)

2. [11 points]

Here is an excerpt from the definition of a doubly-linked list class. A LinkedList object represents a doubly-linked list of integers. The implementation uses no dummy node. The first node in the list has NULL as its `prev` data member, and the last node has NULL as its `next` data member.

```
class LinkedList
{
  public:
    ...
    void eraseSecondToLast();
  private:
    struct Node
    {
      int   value;
      Node* next;
      Node* prev;
    };
    Node* head;   // points to first Node in the list
};
```



The `eraseSecondToLast` function properly deletes one node from a linked list that has at least three nodes; it removes the one just before the last node in the list. You may assume that it will be called only for lists with at least three nodes. (In other words, the code you produce doesn't have to work for a list with two or fewer nodes.) The code **must** take the following form, with no additional lines and the blanks indicating code from the listed choices. Your options are limited to those shown.

```
void LinkedList::eraseSecondToLast()
{
    Node* curr = ____1____;
    while (____2____ != ____3____)
        ____4____ = ____5____;
    Node* temp = ____6____;
    ____7____ = ____8____;
    ____9____ = ____10____;
    delete ____11____;
}
```

A `curr`
B `curr->next`
C `curr->next->next->next`
D `curr->next->next->prev`
E `curr->prev`
F `curr->prev->next`
G `curr->prev->prev`
H `curr->prev->prev->next`
I `head`
J `head->next`
K `NULL`
L `temp`
M `temp->next->next`

Write the letters corresponding to the filled-in code here; we'll look at these, not the code.

1	2	3	4	5	6	7	8	9	10	11
J	B	K	A	B	E	H	A	E	G	L

10