Spring 2017 CS 32

Homework 4

Time due: 11:00 PM Tuesday, May 30

1. The files Set.h and Set.cpp contain the definition and implementation of Set implemented using a doubly-linked list. A client who wants to use a Set has to change the typedef in Set.h, and within one source file, cannot have two Sets containing different types.

   Eliminate the typedef, and change Set to be a class template, so that a client can say

   ```
   #include "Set.h"
   #include <string>
   using std::string;
   ...
   Set<int> si;
   Set<string> ss;
   si.insert(7);
   ss.insert("7-Up");
   ...
   ```

   Also, change `unite` and `subtract` to be function templates.

   (Hint: Transforming the typedef-based solution is a mechanical task that takes only five minutes if you know what needs to be done. What makes this problem non-trivial for you is that you haven't done it before; the syntax for declaring templates is new to you, so you may not get it right the first time.)

   (Hint: Template typename parameters don't have to be named with single letters like `T`; they can be names of your choosing. You might find that by choosing the name `ItemType`, you'll have many fewer changes to make.)

   (Hint: The Node class nested in the Set class can talk about the template parameter of the Set class; it should not itself be a template class.)

   The definitions *and* implementations of your Set class template and the `unite` and `subtract` template functions should be in just one file, Set.h, which is all that you will turn in for this problem. Although the implementation of a non-template non-inline function should not be placed in a header file (because of linker problems if that header file were included in multiple source files), the implementation of a template function, whether or not it's declared inline, *can* be in a header file.

There's a C++ language technicality that relates to a type declared inside a class template, like `N` below:

```
template <typename T>
class S
{
  ...
  struct N
  {
    ...
  };
  N* f();
  ...
};
```

The technicality affects how we specify the return type of a function (such as `S<T>::f`) when that return type uses a type defined inside a template class (such as `S<T>::N`). If we attempt to implement `f` this way:

```
template <typename T>
S<T>::N* S<T>::f()        // Error!  Won't compile.
{
  ...
}
```

the technicality requires the compiler to not recognize `S<T>::N` as a type name; it must be announced as a type name this way:

```
template <typename T>
typename S<T>::N* S<T>::f()       // OK
{
  ...
}
```

2. Consider this program:

```
#include "Set.h"  // class template from problem 1

      class Coord
      {
        public:
          Coord(int r, int c) : m_r(r), m_c(c) {}
          Coord() : m_r(0), m_c(0) {}
            double r() const { return m_r; }
            double c() const { return m_c; }
          private:
            double m_r;
            double m_c;
        };

        int main()
        {
            Set<int> si;
            si.insert(4);                // OK
            Set<string> ss;
            ss.insert("4ever");          // OK
            Set<Coord> sc;
            sc.insert(Coord(4,-4));      // error!
        }
```

Explain in a sentence or two why the call to `Set<Coord>::insert` causes at least one compilation error. (Notice that the calls to `Set<int>::insert` and `Set<string>::insert` are fine.) Don't just transcribe a compiler error message; your answer must indicate you understand the ultimate root cause of the problem and why that is connected to the call to `Set<Coord>::insert`.

26.

a. Implement the `removeOdds` function; you must use `vector`'s `erase` member function:

```
    #include <vector>
    #include <algorithm>
    #include <iostream>
    #include <cassert>
    using namespace std;

      // Remove the odd integers from v.
      // It is acceptable if the order of the remaining even integers is not
      // the same as in the original vector.
    void removeOdds(vector<int>& v)
    {
    }

    void test()
    {
        int a[8] = { 2, 8, 5, 6, 7, 3, 4, 1 };
        vector<int> x(a, a+8);  // construct x from the array
        assert(x.size() == 8 && x.front() == 2 && x.back() == 1);
        removeOdds(x);
        assert(x.size() == 4);
        sort(x.begin(), x.end());
        int expect[4] = { 2, 4, 6, 8 };
        for (int k = 0; k < 4; k++)
            assert(x[k] == expect[k]);
    }

            int main()
            {
                test();
                cout << "Passed" << endl;
            }
```

For this problem, you will turn a file named `odds.cpp` with the body of the `removeOdds` function, from its "void" to its "}", no more and no less. Your function must compile and work correctly when substituted into the program above.

gg. Implement the `removeBad` function:

```
            #include <list>
            #include <vector>
            #include <algorithm>
            #include <iostream>
            #include <cassert>
            using namespace std;

            vector<int> destroyedOnes;

            class Movie
            {
              public:
                Movie(int r) : m_rating(r) {}
                ~Movie() { destroyedOnes.push_back(m_rating); }
                int rating() const { return m_rating; }
              private:
                int m_rating;
            };

              // Remove the movies in li with a rating below 50 and destroy them.
              // It is acceptable if the order of the remaining movies is not
              // the same as in the original list.
            void removeBad(list<Movie*>& li)
            {
            }

            void test()
            {
                int a[8] = { 85, 80, 30, 70, 20, 15, 90, 10 };
                list<Movie*> x;
                for (int k = 0; k < 8; k++)
                    x.push_back(new Movie(a[k]));
                assert(x.size() == 8 && x.front()->rating() == 85 && x.back()->rating() == 10);
                removeBad(x);
                assert(x.size() == 4 && destroyedOnes.size() == 4);
                vector<int> v;
                for (list<Movie*>::iterator p = x.begin(); p != x.end(); p++)
```

```
    {
        Movie* mp = *p;
        v.push_back(mp->rating());
    }
      // Aside:  In C++11, the above loop could be
      //     for (auto p = x.begin(); p != x.end(); p++)
      //     {
      //         Movie* mp = *p;
      //         v.push_back(mp->rating());
      //     }
      // or
      //     for (auto p = x.begin(); p != x.end(); p++)
      //     {
      //         auto mp = *p;
      //         v.push_back(mp->rating());
      //     }
      // or
      //     for (Movie* mp : x)
      //         v.push_back(mp->rating());
      // or
      //     for (auto mp : x)
      //         v.push_back(mp->rating());
    sort(v.begin(), v.end());
    int expect[4] = { 70, 80, 85, 90 };
    for (int k = 0; k < 4; k++)
        assert(v[k] == expect[k]);
    sort(destroyedOnes.begin(), destroyedOnes.end());
    int expectGone[4] = { 10, 15, 20, 30 };
    for (int k = 0; k < 4; k++)
        assert(destroyedOnes[k] == expectGone[k]);
    for (list<Movie*>::iterator p = x.begin(); p != x.end(); p++)
        delete *p;  // Deallocate remaining movies.
}

int main()
{
    test();
    cout << "Passed" << endl;
}
```

For this problem, you will turn a file named `bad.cpp` with the body of the `removeBad` function, from its "void" to its "}", no more and no less. Your function must compile and work correctly when substituted into the program above.

27. An Internet domain has a *label* (e.g., `edu`) and zero or more *subdomains* (e.g., the subdomain labeled `ucla` and the subdomain

labeled `caltech` are subdomains of the domain labeled `edu`). The root domain has the empty string as its label. The following program reflects this structure:

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Domain
{
  public:
    Domain(string lbl) : m_label(lbl) {}
    string label() const { return m_label; }
    const vector<Domain*>& subdomains() const { return m_subdomains; }
    void add(Domain* d) { m_subdomains.push_back(d); }
    ~Domain();
  private:
    string m_label;
    vector<Domain*> m_subdomains;
};

Domain::~Domain()
{
    for (size_t k = 0; k < m_subdomains.size(); k++)
        delete m_subdomains[k];
}

void listAll(string path, const Domain* d) // two-parameter overload
{
    You will write this code.
}

void listAll(const Domain* d)  // one-parameter overload
{
    if (d != nullptr)
        listAll(d->label(), d);
}

int main()
{
    Domain* d1 = new Domain("ucla");
    d1->add(new Domain("cs"));
    d1->add(new Domain("ee"));
    d1->add(new Domain("math"));
    Domain* d2 = new Domain("caltech");
    d2->add(new Domain("math"));
    d2->add(new Domain("cs"));
    Domain* d3 = new Domain("edu");
    d3->add(d1);
    d3->add(d2);
    Domain* d4 = new Domain("com");
    d4->add(new Domain("microsoft"));
    d4->add(new Domain("apple"));
    Domain* d5 = new Domain("biz");
    Domain* root = new Domain("");
    root->add(d3);
    root->add(d4);
    root->add(d5);
    listAll(root);
    cout << "====" << endl;
    listAll(d2);
    cout << "====" << endl;
    listAll(d5);
    delete root;
}
```

When the listAll function is called from the main routine above, the following output should be produced (the first line written is `cs.ucla.edu`, not an empty line):

```
cs.ucla.edu
ee.ucla.edu
math.ucla.edu
math.caltech.edu
cs.caltech.edu
microsoft.com
apple.com
biz
====
math.caltech
cs.caltech
====
biz
```

Each call to the one-parameter overload of `listAll` produces a list, one per line, of all the leaf domain names in the domain tree rooted at `listAll`'s argument. A domain name is a sequence of labels separated by periods. There is no period before the (empty) label of the root domain.

a. You are to write the code for the two-parameter overload of `listAll` to make this happen. You must not use any additional container (such as a stack), and the two-parameter overload of `listAll` must be recursive. You must not use any global variables or variables declared with the keyword `static`, and you must not modify any of the code we have already written or add new functions. You may use a loop to traverse the vector; you must not use loops to avoid recursion.

Here's a useful function to know: The standard library string class has a + operator that concatenates strings and/or characters. For example,

```
string s("Hello");
string t("there");
string u = s + ", " + t + '!';
// Now u has the value "Hello, there!"
```

It's also useful to know that if you choose to traverse an STL container using some kind of iterator, then if the container is const, you must use a `const_iterator`:

```
    void f(const list<int>& c)  // c is const
    {
        for (list<int>::const_iterator it = c.begin(); it !=
c.end(); it++)
            cout << *it << endl;
    }
```

(Of course, a vector can be traversed either by using some kind of iterator, or by using operator[] with an integer argument).

For this problem, you will turn a file named `list.cpp` with the body of the two-parameter overload of the `listAll` function, from its "void" to its "}", no more and no less. Your function must compile and work correctly when substituted into the program above.

b. We introduced the two-parameter overload of `listAll`. Why could you not solve this problem given the constraints in part a if we had only a one-parameter `listAll`, and you had to implement *it* as the recursive function?

91.

a. In conducting its investigation of contacts between a presidential campaign and agents of an unfriendly foreign power, the FBI maintains, for N people numbered 0 through N-1, a two-dimensional array of bool `hasCommunicatedWith` that records which people have been in direct communication with others:`hasCommunicatedWith[i][j]` is true if and only if person i and person j have been in direct communication. If person i has directly communicated with person k, and person k has directly communicated with person j, we call person k a *direct intermediary* between person i and person j.

The agency has an algorithm that, for every pair of people i and j, determines how many direct intermediaries they have between them. Here's the code:

```
    const int N = some value;
    bool hasCommunicatedWith[N][N];
    ...
    int numIntermediaries[N][N];
    for (int i = 0; i < N; i++)
    {
        numIntermediaries[i][i] = -1;  // the concept of
intermediary
                                      // makes no sense in
this case
        for (int j = 0; j < N; j++)
        {
            if (i == j)
                continue;
            numIntermediaries[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                if (k == i  ||  k == j)
                    continue;
               if (hasCommunicatedWith[i][k]  &&
hasCommunicatedWith[k][j])
                   numIntermediaries[i][j]++;
            }
        }
    }
```

What is the time complexity of this algorithm, in terms of the number of basic operations (e.g., additions, assignments, comparisons) performed: Is it O(N), O(N log N), or what? Why? (Note: In this homework, whenever we ask for the time complexity, we care only about the high order term, so don't give us answers like O(N^2+4N).)

b. The algorithm in part a doesn't take advantage of the symmetry of communication: for every pair of persons i and j, hasCommunicatedWith[i][j] == hasCommunicatedWith[j][i]. One can skip a lot of operations and compute the number of direct intermediaries more quickly with this algorithm:

```
c.    const int N = some value;
d.    bool hasCommunicatedWith[N][N];
e.    ...
f.    int numIntermediaries[N][N];
g.    for (int i = 0; i < N; i++)
h.    {
i.        numIntermediaries[i][i] = -1;  // the concept of
   intermediary
```

```
                                            // makes no sense in this case
        for (int j = 0; j < i; j++)  // loop limit is now i, not N
        {
            numIntermediaries[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                if (k == i  ||  k == j)
                    continue;
                if (hasCommunicatedWith[i][k]  &&  hasCommunicatedWith[k][j])
                    numIntermediaries[i][j]++;
            }
            numIntermediaries[j][i] = numIntermediaries[i][j];
        }
    }
```

What is the time complexity of this algorithm? Why?

92.

Here again is the (non-template) non-member `unite` function for Sets from [Set.cpp]:

```
void unite(const Set& set1, const Set& set2, Set& result)
{
    const Set* sp = &set2;
    if (&result == &set1)
    {
        if (&result == &set2)
                return;
    }
    else if (&result == &set2)
        sp = &set1;
    else
    {
        result = set1;
        if (&set1 == &set2)
            return;
    }
    for (int k = 0; k < sp->size(); k++)
    {
        ItemType v;
        sp->get(k, v);
        result.insert(v);
    }
}
```

Assume that `set1`, `set2`, and the old value of `result` each have N elements. In terms of the number of linked list nodes visited during the execution of this function, what is its worst case time complexity? Why?

y. Here is an implementation of a related member function. The call

```
z. s3.unite(s1,s2);
```

sets s3 to the set union of s1 and s2. The implementation is

```
void Set::unite(const Set& set1, const Set& set2)
{
    vector<ItemType> v;

      // copy all items into v;
    for (Node* p1 = set1.m_head->m_next; p1 != set1.m_head;
p1 = p1->m_next)
        v.push_back(p1->m_value);
    for (Node* p2 = set2.m_head->m_next; p2 != set2.m_head;
p2 = p2->m_next)
        v.push_back(p2->m_value);

      // sort v using an O(N log N) algorithm
    sort(v.begin(), v.end());

      // delete result nodes (other than the dummy node)
    while (m_head->next != m_head)
        doErase(m_head->m_next);

      // copy unique items from v into result
    for (size_t k = 0; k < v.size(); k++)
    {
        if (k == 0  ||  v[k] != v[k-1])  // add non-
duplicates
       insertBefore(m_head, v[k]);
    }

      // v is destroyed when function returns
}
```

Assume that `set1`, `set2`, and the old value of `*this` each have N elements. In terms of the number of ItemType objects visited (either in linked list nodes or the vector) during the execution of this function, what is its time complexity? Why?

aa. Here is a different implementation of the member function of part b:

```
bb.    void Set::unite(const Set&set1, const Set& set2)
cc.    {
dd.        const Set* sp = &set2;
ee.        if (this == &set1)
ff.        {
gg.            if (this == &set2)
hh.                    return;
ii.        }
jj.        else if (this == &set2)
kk.            sp = &set1;
ll.        else
mm.       {
nn.           *this = set1;
oo.           if (&set1 == &set2)
pp.               return;
qq.       }
rr.       Node* p1 = m_head->m_next;
ss.       Node* p2 = sp->m_head->m_next;
tt.       while (p1 != m_head  &&  p2 != sp->m_head)
uu.       {
vv.           if (p1->m_value < p2->m_value)
ww.               p1 = p1->m_next;
xx.           else
yy.           {
zz.               if (p1->m_value > p2->m_value)
aaa.                  insertBefore(p1, p2->m_value);
bbb.              else
ccc.                  p1 = p1->m_next;
ddd.              p2 = p2->m_next;
eee.          }
fff.      }
ggg.      for ( ; p2 != sp->m_head; p2 = p2->m_next)
hhh.          insertBefore(m_head, p2->m_value);
iii.  }
```

Assume that `set1`, `set2`, and the old value of `*this` each have N elements. In terms of the number of linked list nodes visited during the execution of this function, what is its time complexity? Why?

93. The file sorts.cpp contains an almost complete program that creates a randomly ordered array, sorts it in a few ways, and reports on the elapsed times. Your job is to complete it and experiment with it.

    You can run the program as is to get some results for the STL sort algorithm. The insertion sort result will be meaningless (and you'll get an assertion violation), because the insertion sort function right now doesn't do anything. That's one thing for you to write.

    The objects in the array are not cheap to copy, which makes a sort that does a lot of moving objects around expensive. Your other task will be to create a vector of *pointers*to the objects, sort the pointers using the same criterion as was used to sort the objects, and then make one pass through the vector to put the objects in the proper order.

    Your two tasks are thus:

    a. Implement the `insertion_sort` function.

b. Implement the `compareStorePtr` function and the code in `main` to create and sort the array of pointers.

The places to make modifications are indicated by `TODO:` comments. You should not have to make modifications anywhere else. (Our solution doesn't.)

When your program is correct, build an optimized version of it to do some timing experiments. On cs32.seas.ucla.edu, build the executable and run it this way:

```
  /usr/local/cs/bin/g32fast -o sorts sorts.cpp
  ./sorts
```

(You don't have to know this, but this command omits some of the runtime error checking compiler options that our g32 command supplies, and it adds the -O2 compiler option that causes the compiler to spend more time optimizing the machine language translation of your code so that it will run faster when you execute it.)

Under Xcode, select Product / Scheme / Edit Scheme.... In the left panel, select Run, then in the right panel, select the Info tab. In the Build Configuration dropdown, select Release. For Visual C++, it's a little trickier.

Try the program with about 10000 items. Depending on the speed of your processor, this number may be too large or small to get meaningful timings in a reasonable amount of time. Experiment. Once you get insertion sort working, observe the O(N^2) behavior by sorting, say, 10000, 20000, and 40000 items.

To further observe the performance behavior of the STL sort algorithm, try sorting, say, 100000, 200000, and 400000 items, or even ten times as many. Since this would make the insertion sort tests take a long time, skip them by setting the TEST_INSERTION_SORT constant at the top of sorts.cpp to false.

Notice that if you run the program more than once, you may not get exactly the same timings each time. This is *not* because you're not getting the same sequence of random numbers each time; you are. Instead, factors like caching by the operating system are the cause.

Turn it in

By Monday, May 29, there will be a link on the class webpage that will enable you to turn in this homework. Turn in one zip file that contains your solutions to the homework problems. The zip file must contain four files:

- `Set.h`, a C++ header file with your definition and implementation of the class and function templates for problem 1. For you to not get a score of zero for this problem, this test program that we will try with your header **must** build and execute successfully under both g32 and either Visual C++ or clang++:

```
#include "Set.h"
#include <iostream>
#include <string>
#include <cassert>
using namespace std;

void test()
{
    Set<int> si;
    Set<string> ss;
    assert(si.empty());
    assert(ss.empty());
    assert(si.size() == 0);
    assert(ss.size() == 0);
    assert(si.insert(10));
    assert(ss.insert("hello"));
    assert(si.contains(10));
    assert(ss.contains("hello"));
    int i;
    assert(si.get(0, i)  &&  i == 10);
    string s;
    assert(ss.get(0, s)  &&  s == "hello");
    assert(si.erase(10));
    assert(ss.erase("hello"));
    Set<int> si2(si);
    Set<string> ss2(ss);
    si.swap(si2);
    ss.swap(ss2);
    si = si2;
    ss = ss2;
    unite(si,si2,si);
    unite(ss,ss2,ss);
    subtract(si,si2,si);
    subtract(ss,ss2,ss);
}

int main()
{
    test();
```

- `        cout << "Passed all tests" << endl;`
- `    }`
- `odds.cpp`, a C++ source file with your solution to problem 3a.
- `bad.cpp`, a C++ source file with your solution to problem 3b.
- `hw.doc`, `hw.docx`, or `hw.txt`, a Word document or a text file with your solutions to problems 2, 4b, 5a, 5b, 6a, 6b, and 6c.
- `list.cpp`, a C++ source file with the implementation of the two-parameter overload of the `listAll` function for problem 4a.
- `sorts.cpp`, a C++ source file with your solution to problem 7.