Spring 2018 CS 32

Homework 4

Time due: 11:00 PM Tuesday, May 29

1. The files Sequence.h and Sequence.cppcontain the definition and implementation of Sequence implemented using a doubly-linked list. A client who wants to use a Sequence has to change the type alias declaration in Sequence.h, and within one source file, cannot have two Sequences containing different types.

   Eliminate the `using` statement defining the type alias, and change Sequence to be a class template, so that a client can say

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

   Also, change `subsequence` and `interleave`to be function templates.

   (Hint: Transforming the solution based on a type alias is a mechanical task that takes 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 Sequence class can talk about the template parameter of the Sequence class; it should not itself be a template class.)

   The declarations *and* implementations of your Sequence class template and the `subsequence` and `interleave` template functions must be in just one file, Sequence.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 without causing linker problems.

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

   ```
   template <typename T>
   class M
   {
     ...
     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
{
  ...
}
```

For you to not get a score of zero for this problem, this test program that we will try with your `Sequence.h` **must** build and execute successfully under both g32 and either Visual C++ or clang++, with no `Sequence.cpp` file:

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

void test()
{
    Sequence<int> si;
    Sequence<string> ss;
    assert(ss.empty());
    assert(ss.size() == 0);
    assert(ss.insert("Hello") == 0);
    assert(si.insert(0, 20));
    assert(si.insert(10) == 0);
    assert(si.remove(10) == 1);
    assert(si.erase(0));
    assert(ss.remove("Goodbye") == 0);
    assert(ss.find("Hello") == 0);
    string s;
    assert(ss.get(0, s));
    assert(ss.set(0, "Hello"));
    Sequence<string> ss2(ss);
    ss2.swap(ss);
    ss2 = ss;
    assert(subsequence(ss,ss2) == 0);
    assert(subsequence(si,si) == -1);
    interleave(ss,ss2,ss);
    interleave(si,si,si);
}

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

2. Consider this program:

```
#include "Sequence.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()
{
    Sequence<int> si;
    si.insert(50);               // OK
    Sequence<Coord> sc;
    sc.insert(0, Coord(50,20));  // OK
    sc.insert(Coord(40,10));     // error!
}
```

Explain in a sentence or two why the call to the one-argument form of `Sequence<Coord>::insert` causes at least one compilation error. (Notice that the call to the one-argument form of `Sequence<int>::insert` is fine, as is the call to the two-argument form of `Sequence<Coord>::insert`.) 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 `Sequence<Coord>::insert`.

3.

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.

b. 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.

4. A file has a name. A file is either a plain file (like a text file, an image file, a C++ source program, etc.) or a directory. Directories contain zero or more files. The following program reflects this structure:

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

using namespace std;

class File
{
  public:
    File(string nm) : m_name(nm) {}
    virtual ~File() {}
    string name() const { return m_name; }
    virtual bool add(File* f) = 0;
    virtual const vector<File*>* files() const = 0;
  private:
    string m_name;
};

class PlainFile : public File
{
  public:
    PlainFile(string nm) : File(nm) {}
    virtual bool add(File* f) { return false; }
    virtual const vector<File*>* files() const { return nullptr; }
};

class Directory : public File
{
  public:
    Directory(string nm) : File(nm) {}
    virtual ~Directory();
    virtual bool add(File* f) { m_files.push_back(f); return true; }
    virtual const vector<File*>* files() const { return &m_files; }
  private:
    vector<File*> m_files;
};

Directory::~Directory()
{
    for (int k = 0; k < m_files.size(); k++)
        delete m_files[k];
}

void listAll(string path, const File* f)  // two-parameter overload
```

```
{
    You will write this code.
}

void listAll(const File* f)  // one-parameter overload
{
    if (f != nullptr)
        listAll("", f);
}

int main()
{
    Directory* d1 = new Directory("Fun");
    d1->add(new PlainFile("party.jpg"));
    d1->add(new PlainFile("beach.jpg"));
    d1->add(new PlainFile("skitrip.jpg"));
    Directory* d2 = new Directory("Work");
    d2->add(new PlainFile("seaslab.jpg"));
    Directory* d3 = new Directory("My Pictures");
    d3->add(d1);
    d3->add(new PlainFile("me.jpg"));
    d3->add(new Directory("Miscellaneous"));
    d3->add(d2);
    listAll(d3);
    delete d3;
}
```

When the listAll function is called from the main routine above, the following output should be produced (the first line written is `/My Pictures`, not an empty line):

```
/My Pictures
/My Pictures/Fun
/My Pictures/Fun/party.jpg
/My Pictures/Fun/beach.jpg
/My Pictures/Fun/skitrip.jpg
/My Pictures/me.jpg
/My Pictures/Miscellaneous
/My Pictures/Work
/My Pictures/Work/seaslab.jpg
```

This is a list, one per line, of the complete pathname for `listAll`'s argument and, if the argument is a directory, everything in that directory. A pathname starts with / and has / separating pathname components.

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?

5.

a. Suppose we have a list of N world cities, numbered from 0 to N-1. The two-dimensional array of doubles `dist` holds the airline distance between each pair of cities: `dist[i][j]` is the distance between city i and city j.

Now, for every pair of cities i and j, we'd like to consider all the flights between the two that make one stop in a third city k, and record which city k yields the shortest distance traveled in a one-stop flight between city i and city j that passes through city k. Here's the code:

```
const int N = some value;
assert(N > 2);  // algorithm fails if N <= 2
double dist[N][N];
...
int bestMidPoint[N][N];
for (int i = 0; i < N; i++)
{
    bestMidPoint[i][i] = -1;  // one-stop trip to self is silly
    for (int j = 0; j < N; j++)
    {
        if (i == j)
            continue;
        int minDist = maximum possible integer;
        for (int k = 0; k < N; k++)
        {
            if (k == i  ||  k == j)
                continue;
            int d = dist[i][k] + dist[k][j];
            if (d < minDist)
            {
                minDist = d;
                bestMidPoint[i][j] = k;
            }
        }
    }
}
```

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 distances: for every pair of cities i and j, dist[i][j] == dist[j][i]. We can skip a lot of operations and compute the best midpoints more quickly with this algorithm:

```
const int N = some value;
assert(N > 2);  // algorithm fails if N <= 2
double dist[N][N];
...
int bestMidPoint[N][N];
for (int i = 0; i < N; i++)
{
    bestMidPoint[i][i] = -1;  // one-stop trip to self is silly
    for (int j = 0; j < i; j++)  // loop limit is now i, not N
    {
        int minDist = maximum possible integer;
        for (int k = 0; k < N; k++)
        {
            if (k == i  ||  k == j)
                continue;
            int d = dist[i][k] + dist[k][j];
            if (d < minDist)
            {
                minDist = d;
                bestMidPoint[i][j] = k;
                bestMidPoint[j][i] = k;
            }
        }
    }
}
```

What is the time complexity of this algorithm? Why?

6.

a. Here again is the non-member `interleave` function for Sequences from Sequence.cpp:

```
void interleave(const Sequence& seq1, const Sequence& seq2, Sequence& result)
{
    Sequence res;

    int n1 = seq1.size();
    int n2 = seq2.size();
    int nmin = (n1 < n2 ? n1 : n2);
    int resultPos = 0;
    for (int k = 0; k < nmin; k++)
    {
        ItemType v;
        seq1.get(k, v);
        res.insert(resultPos, v);
        resultPos++;
        seq2.get(k, v);
        res.insert(resultPos, v);
        resultPos++;
    }

    const Sequence& s = (n1 > nmin ? seq1 : seq2);
    int n = (n1 > nmin ? n1 : n2);
    for (int k = nmin ; k < n; k++)
    {
        ItemType v;
        s.get(k, v);
        res.insert(resultPos, v);
        resultPos++;
    }

    result.swap(res);
}
```

Assume that `seq1`, `seq2`, and the old value of `result` each have N elements. In terms of the number of ItemType objects visited (in the linked list nodes) during the execution of this function, what is its time complexity? Why?

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

```
s3.interleave(s1,s2);
```

sets `s3` to the result of interleaving `s1`and `s2`. The implementation is

```
void Sequence::interleave(const Sequence& seq1, const Sequence& seq2)
{
    Sequence res;

    Node* p1 = seq1.m_head->m_next;
    Node* p2 = seq2.m_head->m_next;
    for ( ; p1 != seq1.m_head  &&  p2 != seq2.m_head;
                                          p1 = p1->m_next, p2 = p2->m_next)
    {
        res.insertBefore(res.m_head, p1->m_value);
        res.insertBefore(res.m_head, p2->m_value);
    }

    Node* p    = (p1 != seq1.m_head ? p1 : p2);
    Node* pend = (p1 != seq1.m_head ? seq1 : seq2).m_head;
    for ( ; p != pend; p = p->m_next)
        res.insertBefore(res.m_head, p->value);

      // Swap *this with res
    swap(res);
      // Old value of *this (now in res) is destroyed when function returns.
}
```

Assume that `seq1`, `seq2`, and the old value of `*this` each have about N elements. In terms of the number of ItemType objects visited during the execution of this function, what is its time complexity? Why? Is it the same, better, or worse, than the implementation in part a?

7. 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. You won't get any result for insertion sort, 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 `compareStudentPtr`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:

```
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 partly because of not getting the same sequence of random numbers each time, but also because of factors like caching by the operating system.

Turn it in

By Monday, May 28, 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 six files:

- `Sequence.h`, a C++ header file with your declaration and implementation of the class and function templates for problem 1.
- `odds.cpp`, a C++ source file with your solution to problem 3a.
- `bad.cpp`, a C++ source file with your solution to problem 3b.
- `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.
- `hw.docx`, `hw.doc`, or `hw.txt`, a Word document or a text file with your solutions to problems 2, 4b, 5a, 5b, 6a, and 6b.