Winter 2016 CS 32

Project 3 Warmup

We anticipate that many people working on Project 3 will spend a lot of time debugging something that arises from a common novice misunderstanding. To save you that time later, we'll give you a chance to make that mistake in a simpler context, so you can work out that issue and how it manifests itself. (It may turn out that you don't have that misunderstanding, so you won't have any problems during this warmup. Still, keep this warmup in mind, because you may still make the mistake in Project 3.)

You will not turn in any of the code you write because of this warmup.

1. Implement the `removeOdds` function:

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

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

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

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

2. Implement the `removeOdds` 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;
}
```

3. 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]);
}

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

4. Implement the `removeBad` function:

```
#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 v 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 vector.
void removeBad(vector<Movie*>& v)
{
}
```

```
void test()
{
    int a[8] = { 85, 80, 30, 70, 20, 15, 90, 10 };
    vector<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 (int k = 0; k < 4; k++)
        v.push_back(x[k]->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]);
}

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