Midterm Exam 1  
CS 455, Spring 2007

February 28, 2007

There are 6 problems on the exam, with 77 points total available. There are 8 pages to the exam, including this one; make sure you have all of them. There is also a separate one-page code handout. If you need additional space to write any answers, you may use the backs of pages (just direct us to look there).

Put your name and loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1AB</td>
<td>4 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 1CDE</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>15 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td>77 pts.</td>
</tr>
</tbody>
</table>

Problem 1 [12 pts. total]

For this problem please refer to the Home class definition is given on the code handout. For parts A and B assume the following definition for the 3-argument Home class constructor.

// Create a home with the given address, square footage, // and price // PRECONDITION: squareFeet and price must be non-negative Home::Home(string address, int squareFeet, int price)
    : mySquareFeet(squareFeet), myPrice(price)
{
}

Part A [2]. Consider the following call to the constructor:

Home myhouse("30200 N. Sepulveda", 1000, 245000);
cout << myhouse.getAddress() << " " << myhouse.getPrice();

Is the value of myAddress known after a call to this constructor? If so, what value does it have?

Part B [2]. Is the value of myPrice known after a call to this constructor? If so, what value does it have?
**Part C [3].** A corrected version of the constructor appears below. Add an assert statement to it that checks the precondition of the constructor.

```cpp
// Create a home with the given address, square footage,
// and price
// PRECONDITION: squareFeet and price must be non-negative
Home::Home(string address, int squareFeet, int price)
    : myAddress(address),
      mySquareFeet(squareFeet),
      myPrice(price)
{
```

**Part D [2].** Assuming we have the version of the constructor after doing Part C, what happens if we do the following call below (you can just describe the behavior and result of the call):

```cpp
Home myhouse("302 N. Sepulveda", 1000, -500);
```

**Part E [3].** Write an implementation of the default constructor for `Home` that uses a member initializer list. (Hint: there is more than one right answer).

```cpp
Home::Home()
```
Problem 2 [5 pts.]

Implement the non-member function costsLessThan, which tells if its first Home argument costs less than its second Home argument. See code handout for Home class definition. Here are some example calls

```cpp
Home a("1110 Broadway Ave.", 1200, 625000);
Home b("501 E. Maple St.", 3000, 995000);
Home c("1350 E. Maple St.", 900, 625000);
cout << lessThan(a, b) << endl;   // true
cout << lessThan(c, a) << endl;   // false
```

Here's the header:

```cpp
// returns true if and only if h1 costs less than h2
bool costsLessThan(Home h1, Home h2) {
```
Problem 3 [5 pts.]

Consider a function `cheapestHome`, whose header appears below, which returns the Home that’s the least expensive of all the ones in the `homeList`. The function will only work with a `homeList` with at least one home in it.

```cpp
// Return the cheapest home on the list
// PRE: homeList must have at least one home on it
Home cheapestHome(vector<Home> homeList);
```

Devise test cases for this function so that it thoroughly tests the function. (Recall this is called a black-box test, where we devise test cases without seeing the internals of the function.) You don’t have to give the exact test data values or output that would be produced, but can just describe the data, so it’s clear how each case differs from the other cases.
Problem 4 [15 pts.]

Write the function `cheapestHome`, which returns the `Home` that’s the least expensive of all the ones in the `homeList`. The function will only work with a `homeList` with at least one home in it.

For full credit your function must use `costsLessThan`. The interface for `costsLessThan` and the `Home` class, and a reminder of the vector member functions are given on the code handout. You may not use any other vector functions besides the ones listed.

Hint: to help verify that your function is correct, hand trace it with the test cases you devised for the previous problem.

```cpp
// Return the cheapest home on the list
// PRE: homeList must have at least one home on it
Home cheapestHome(vector<Home> homeList) {
```
Problem 5 [20 pts.]

Write the function scoreFreq, which takes a vector of student scores in the range [0-100], inclusive, and returns the number of times each score appears in the vector. The return value will be a vector itself. For example, given the following code fragment that includes a call to this function,

\[
\text{vector<int> scores;}
\]
\[
\text{\ldots}
\]
\[
\text{vector<int> freq;}
\]
\[
\text{freq = scoreFreq(scores);}
\]

freq[0] is the number of students that got 0’s,
freq[1] is the number of students that got 1’s,
freq[2] is the number of students that got 2’s, etc.

A reminder of the vector member functions is given on the code handout. You may not use any other vector functions besides the ones listed.

\[
// \text{PRE: each scores[i] value must be in range [0-100] inclusive}
\]
\[
\text{vector<int> scoreFreq(vector<int> scores) {}
\]
Problem 6 [20 pts.]

Write the function removeMiddle, which removes the middle element from its vector parameter, such that all the remaining elements are in the same order as before the call. By middle, we mean the one at the middle location, not the one with the median value. To keep the problem simpler, you may assume that the vector has an odd number of elements and that it has at least one element. Here are some examples of its use:

<table>
<thead>
<tr>
<th>v before</th>
<th>v after</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 9 2 7 9</td>
<td>3 9 7 9</td>
</tr>
<tr>
<td>5</td>
<td>[empty]</td>
</tr>
</tbody>
</table>

Please refer to the accompanying code handout for reminder of vector member functions. You may not use any other vector functions besides the ones listed.

```cpp
// PRE: vector is non-empty and has an odd number of elements
void removeMiddle (vector<int> &v) {
```