Computer Science II — CSci 1200
Test 2 Overview and Review

Overview

• Test 2 will be held **Friday, March 11, 2005, 10:00-11:50pm, Sage 3303**. No make-ups will be given except as pre-arranged by written excuse from the Dean of Students office.

• Coverage: the emphasis is the material covered in Lectures 6-12, Labs 4-7 (not the material on maps from Lab 7), and HW 3-5; material from earlier in the semester may be covered as well.

• Closed-book and closed-notes. Photocopies of the “Details” sections from Chapters 0-5 and 9 of the text will be provided.

• Maps, covered in Lecture 13, will not be on the test.

• Below are sample questions. Solutions will be posted on-line.

• *How to study?*
  – Review the lecture notes
  – Review and re-do lecture exercises, lab and homework problems.
  – Do the practice problems. Practice writing solutions using pencil (or pen) and paper.

Practice Problems

1. Write a code segment that copies the contents of a string into a list of char in reverse order.

2. Write a code segment that removes all occurrences of the letter 'c' from a string. Consider both uppercase 'C' and lowercase 'c'. For example, the string

   Chocolate

   would become the string

   hoolate
3. Write a function that rearranges a list of doubles so that all the negative values come before all the non-negative values AND the order of the negative values is preserved AND the order of the positive values is preserved. For example, if the list contains

-1.3, 5.2, 8.7, 0.0, -4.5, 7.8, -9.1, 3.5, 6.6

Then the modified list should contain

-1.3, -4.5, -9.1, 5.2, 8.7, 0.0, 7.8, 3.5, 6.6

This is a challenging problem, but it is good practice.

4. Consider the following start to the declaration of a Course class.

```cpp
class Course {
public:
    Course( const string& id, unsigned int max_stu )
    : course_id( id ), max_students( max_stu ) {}  

private:
    string course_id;
    list<string> students;
    unsigned int max_students;
};
```

Use this in solving each of the following problems.

(a) Provide three member functions: one returns the maximum number of students allowed in the Course, a second returns the number of students enrolled, and a third returns a bool indicating whether or not any openings remain in the Course. Provide both the prototype in the declaration above and the member function implementation.

(b) Write a function that sorts course objects by increasing enrollment. In other words, the course having the fewest students should be first. If two courses have the same number of students, the course with the smaller maximum number of students allowed should be earlier in the sorted vector.
(c) Write a member function of `Course` called `merge`. This function should take another `Course` object as an argument. All students should be removed from the passed `Course` and placed in the current course (the one on which the function is called). You may assume (just for this problem) that no students are enrolled in both courses and there is enough room in the current course. As an example if `cs2` is of type `Course` and has 15 students and `baskets` is of type `Course` and has 5 students, then after the call

```cpp
    cs2.merge( baskets );
```

`baskets` will have no students and `cs2` will have 20.

5. Write a function that takes a vector of strings as an argument and returns the number of vowels that appear in the string. A vowel is defined as an 'a', 'e', 'i', 'o' or 'u'. For example, if the vector contains the strings

```
ap
abe
lincoln
went
to
the
white
house
```

Your function should return the value 12.

You may assume that all letters are lower case. Here is the function prototype:

```cpp
    int count_vowels( const vector<string>& strings )
```

6. Write a function that takes a list of doubles and copies its values into two lists of doubles, one containing only the negative numbers from the original list, the other containing only the positive numbers. Values that are 0 should not be in either list. For example, if the original list contains the values

```
-1.3, 5.2, 8.7, -4.5, 0.0, 7.8, -9.1, 3.5, 6.6
```

then the resulting list of negative numbers should contain
-1.3, -4.5, -9.1

and the resulting list of positive numbers should contain

5.2, 8.7, 7.8, 3.5, 6.6

Start this problem by writing the function prototype as you think it should appear and then write the code.

7. Write a program that reads a sequence of strings from cin (up to the end of file) and determines the fraction of strings for which the vowels appear in alphabetical order. For example if the strings are

Aardvark birthday anybody count car peppermint 
Ohio!! enough re-iterate baseball

Then the output should be

0.6

because there are 10 strings and 6 (Aardvark, anybody, count, car, peppermint and enough) contain their vowels in alphabetical order.

You do not need to write down any #include statements, and you are welcome to write additional functions.

8. Write a function that merges two lists of float, each of which is already in increasing order, to form a single list of floats. Try to do this in two ways. In the first way, copy the values into a new vector. In the second, insert the values from one list into the other. Use the list<T>::insert function.

9. Write a function that finds the mode of a vector of integers. This is the integer that occurs most often. If two (or more) integers occur the same number of times then you should return the smallest. More computationally efficient solutions are better. For example, if the vector contains the values

15, 45, 32, 16, 32, 83, 45, 89, 32, 15, 83

the function should return the value 32 because it occurs 3 times and no other value occurs more than twice.
10. Write a **recursive** function to add two non-negative integers using only a limited set of operations: adding 1 to a value, subtracting 1 from a value, and comparing a value to 0. No loops are allowed in the function. The function prototype should be

```c
int Add( int m, int n)
```

11. Write a **recursive** function to multiply two non-negative integers using only addition, subtraction and comparison operations. No loops are allowed in the function. The function prototype should be

```c
int Multiply( int m, int n)
```

12. Consider the **mergesort** function discussed in detail in lecture. Suppose the vector passed to **mergesort** has 7 items in it. Specify the EXACT set of function calls that are made and the exact order that they are made. In specifying this, you do not need to show the contents of the vector, just the values of low and high (and mid, for calls to merge). This is not a question that will appear on the test, but it is instructive about the behavior of merge sort.