Overview

- Thursday, April 4, 2002, 2:00-3:50pm, in class

- Closed-book and closed-notes. Photocopies of Appendix B of the text will be provided, so be sure you familiarize yourself with its contents.

- The emphasis will be material covered since Test 2. You are still responsible for material covered before Test 1, however, because we have continued to build-on and use this material.

- The format will be similar to Test 1.

- Syntax is not a major concern except for the basic definition of maps and the syntax of class declarations and member-function definitions. See below.

Important Topics

- Maps:
  - How they work: declaration, ordering, pairs, operator[], iterators, insert, erase, find.
  - How they are used: class examples, lab exercises, and MP3 project.

- Generic functions: writing them using templates and iterators

- C++ classes:
  - Declaration, public vs. private, member variables and member functions, class scope, constructors, const.
  - Use in programs: rewrite of the week 4 homework.

- Low-level memory: memory, pointers, arrays, char arrays, pointers vs. arrays, pointers as array iterators, dynamic memory (new and delete). Equivalence between a[i] and *(a+i).

- Command-line arguments, opening files, and reading from / writing to files.

- Operators and their use in making classes behave like basic types:
– operators as member functions, operators as friends, operators as ordinary functions
– returning objects vs. returning references to objects (this pointer)

• Classes and templated classes with dynamic memory: copy constructors, assignment operators, and destructors.

Practice Problems

After reviewing the lecture notes and text based on the outline above, you should do practice problems. Work on problems from the notes, from lab, from homework, and from projects. Below are a few additional practice problems. Do not consider this list complete. You might even make-up specific examples of the types problems that you think may be on the test and work them yourself.

1. Create an Appointment class that has three private member variables: one representing the day of the appointment, one representing the hour of the appointment, and one describing the appointment. To keep things simple, for the day simply represent it as a Julian day, which is an integer in the range 1..365 (1 represents January 1, 32 represents February 1, 60 represents March 1 (for non-leap years)), for the hour simply represent it as an integer in the range 0..23 (military time) rather than worrying about am or pm, and for the appointment just use a string.

(a) Give the class declaration, including the member variables and the prototypes for the following functions: (i) a constructor from a Julian day, an hour and a description, (ii) a copy constructor, (iii) an operator== as a member function, and (iv) a friend function operator<<. (Of course, more functions than these would be needed, but this is enough for this problem.)

(b) Give the function definitions (the implementations) for the four functions you declared in part (a). You can assume that the values provided to the constructor are in the proper ranges.

2. Given an array of integers, intarray, and a number of array elements, n, write a short code segment that uses pointer arithmetic and dereferencing to add every second entry in the array. For example, when intarray is
and \( n=9 \), the segment should add \( 1 + 4 + 2 + 9 + 6 \) to get 22. Store the result in a variable called \( \text{sum} \).

3. (10 points) Clearly show the output from the following code segment.

```cpp
int x = 45;
int y = 30;
int *p = &x;
*p = 20;
cout << "a: x = " << x << endl;

int *q = &y;
int temp = *p;
*p = *q;
*q = temp;
cout << "b: x = " << x << ", y = " << y << endl;

int * r = p;
p = q;
q = r;
cout << "c: *p = " << *p << ", *q = " << *q << endl;
cout << "d: x = " << x << ", y = " << y << endl;
```

4. Write a \texttt{str} class member function that creates a new string from the current string and has the same characters but in reverse order. The function prototype is

\[
\text{str str::reverse( ) const;}
\]

Recall that \texttt{str} class objects have three member variables:

\[
\text{char * arr_;}
\text{unsigned int size_;}
\text{unsigned int alloc_;}
\]

5. (This problem may be a bit harder than what will be on the exam because our emphasis has been different this semester, but it should still be a good practice problem.) Write a function that takes an
array of floating point numbers and copies its values into two new arrays that must be allocated in the function, one containing only the negative numbers from the original array, and the other containing the non-negative numbers from the original array. For example, if the original array is

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
-1.3 & 5.2 & 8.7 & 0.0 & -4.5 & 7.8 & -9.1 & 3.5 & 6.6 \\
\end{array}
\]

Then the resulting array containing the negative values would be

\[
\begin{array}{ccccccc}
0 & 1 & 2 \\
-1.3 & -4.5 & -9.1 \\
\end{array}
\]

and the resulting array containing the non-negative values would be

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
5.2 & 8.7 & 0.0 & 7.8 & 3.5 & 6.6 \\
\end{array}
\]

(a) Start by writing the function prototype. Think about what parameters (6 of them) you need, what their types should be, and how they should be passed.

(b) Now write the code of the actual function. You do NOT need to write the prototype over again. Do not allocate any more space for the new arrays than is necessary. You may use the back of this page or the back of the preceding page if you need more room.

(c) Compare this to a version that is based on vectors or lists.

6. Our word counting program, discussed in class, created a map of the form

\[
\text{map< string, int > wc;}
\]

This map is an association between a word and the number of times it occurs in an input file. When we iterate through \( \text{wc} \), we access the map entries in alphabetical order. Suppose instead we wanted the entries sorted by the number of times they occur, with words occurring the fewest times first and words occurring the most times last. One way to do this is to create another map:

\[
\text{map< int, list<string> > word_order;}
\]
where in each entry of the map the int is the number of occurrences and the list<string> gives the words that occurred that many times. For example, if "hello", "never", and "once" are the only words that occurred exactly 5 times in the input file then there should be an entry in the map that contains the integer 5 and a list containing "hello", "never" and "once".

Write function to create word_order from wc. Here is the prototype:

```cpp
void alpha_to_occurrence( const map< string, int >& wc,
                          map< int, list<string> >& word_order);
```

7. Reconsider your MP3 program. Write an MP3 class. What member variables and member functions should this class have? Write several of the member functions.