Overview

• Test 3 will be held **Friday, April 15, 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 13-19, Labs 7, 8 and 9 (checkpoint 1), and HW 6; material from earlier in the semester may be covered as well.

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

• 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.

• Nothing on linked lists (Lecture 20) will be covered on the test.

Practice Problems

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

   ```
   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 `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:

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

2. You are given two maps of type

    typedef map< string, list<string> > CoursesType;

Think of these as representing two maps associating course ids with lists of student ids. Your problem is to write a function to merge two CoursesType maps into a single map. When a course id is in both maps, the two lists associated with it, one from each map, must be merged into a single list. A pair associating the course id and the merged student id list should be placed in the final map. When a course id is in only one of the maps, the course id / student id list pair should be copied into the final map unchanged. You may assume that the following function has already been written and works correctly:

    list<string> merge( list<string> const& a, list<string> const& b );

3. 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

    | 0 1 2 3 4 5 6 7 8 |
    | 1 16 4 -3 2 76 9 3 6 |

and n==9, the segment should add 1 + 4 + 2 + 9 + 6 to get 22. Store the result in a variable called sum.

4. Show the output from the following code segment.

    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;
5. Write a `Vec<T>` class member function that creates a new `Vec<T>` from the current `Vec<T>` that stores the same values as the original vector but in reverse order. The function prototype is

```cpp
template <class T>
Vec<T> Vec<T>::reverse() const;
```

Recall that `Vec<T>` class objects have three member variables:

- `T* data_start; // first element in the 'Vec'
- `T* data_end; // (one past) the last element in the 'Vec'
- `T* array_bound; // (one past) the allocated memory`

6. (This problem may be a bit harder than what will be on the exam because we haven’t emphasized passing pointers by reference, 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

```
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
```

Then the resulting array containing the negative values would be

```
0  1  2
-1.3 -4.5 -9.1
```

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

```
0  1  2  3  4  5
5.2 8.7 0.0 7.8 3.5 6.6
```

(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.

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

7. Here is a class to represent an \((x,y,z)\) point location:

```cpp
class Point {
public:
    Point( double x=0.0, double y=0.0, double z=0.0 )
        : x_(x), y_(y), z_(z) {}
```
Point( const Point& p )
  : x_(p.x_), y_(p.y_), z_(p.z_) {}

void set_x( double x ) { x_ = x; }
void set_y( double y ) { y_ = y; }
void set_z( double z ) { z_ = z; }
double x() const { return x_; }
double y() const { return y_; }
double z() const { return x_; }

private:
  double x_, y_, z_; 
};

Write operator+= as a member function of the Point class. Specify its
declaration and indicate the location where it should go in the class declara-
tion above. Then give the function definition (the actual function). As
an example to show how the operator should work, the code

Point p(1.2, 3.4, -5 );
Point q(2.1, 4.0, 6.2 );
p += q;
cout << "(" << p.x() << "," << p.y() << "," << p.z() << ")\n";

should output
(3.3, 7.4, 1.2)

8. What is the output of the following code?

int * a = new int[4];
cout << "A: ";
for( unsigned int i=0; i<4; ++i ) cout << a[i] << " ";
cout << endl;
for( int * b = a; b != a+4; b += 2 ) *b = b-a;
cout << "B: ";
for( unsigned int i=0; i<4; ++i ) cout << a[i] << " ";
cout << endl;

int * c = a;
c[3] = 14;
c[1] = -2;
cout << "C: ";
for( unsigned int i=0; i<4; ++i ) cout << a[i] << " ";
cout << endl;
9. This is a 3-part question, with the parts indicated within the code below.

```cpp
int main()
{
    int * p = new int[6];
    for ( unsigned int i=0; i<6; ++i )
        p[i] = 2*i;

    // (a) Write code using pointers to output the contents
    // of the array pointed by p. You may not using indexing.

    int * q = p+2;
    *q = 20;
    ++q;
    *q = 22;
    *(q+2) = 30;

    // (b) Show the state of the pointers p and q and the
    // contents of the dynamically allocated array. Do this
    // by drawing a diagram, labeling clearly the pointers p
    // and q, where they point to in the array, and then the
    // contents.

    q = p;
    p = new int[4];
    q[2] = p[0];
    q[3] = p[1];
    p[1] = q[0];

    // (c) Show the output of the following


    return 0;
}
```

10. Here is the declaration for a `TimeOfDay` class in a file called `TimeOfDay.h`:

```cpp
class TimeOfDay {
public:
    TimeOfDay( ) : hour_(0), minute_(0) {}
    TimeOfDay( int h, int m ) : hour_(h), minute_(m) {}
    int hour() const { return hour_; }
    int minute() const { return minute_; }
};
```
private:
    int hour_, minute_;  
};

An operator+ on two TimeOfDay objects should behave as follows:

    TimeOfDay t1( 14, 25 ); // 14 hours and 25 minutes, or 2:25 pm
    TimeOfDay t2( 10, 49 ); // 10 hours and 49 minutes, or 10:49 am
    TimeOfDay t3 = t1 + t2; // 1 hour and 14 minutes, 1:14 am

Notice that computing the addition to form t3 involved wrap-around in both the minute and hours.

(a) Show the declarations above for operator+ as a member function of TimeOfDay and as a non-member function. Indicate clearly where they would appear in the class declaration.

(b) Write the implementation of operator+ as a non-member function.

11. Just like in our own implementation of the vector class (Vec<T>), we can also create an implementation of the string class. It has a similar structure, with a dynamically allocated array of characters (it is not templated).
We will represent it somewhat differently, however. The class has three member variables: a pointer, arr_, to the start of the allocated array; an unsigned integer, size_, indicating the number of characters stored in the array; an unsigned integer, alloc_, indicating the number of characters in the allocated array. The following is part of our declaration of the str class.

    class str {
    public:
        typedef char* iterator;
        typedef const char* const_iterator;
    private:
        char * arr_;  
        unsigned int size_;  
        unsigned int alloc_;  
    public:
        str( );
        str( char c );
        str( const char * cstr );
        str( const str& old );
        str( str::iterator beg, str::iterator end );
        ~str();
        char& operator[] ( int i ) { return arr_[i]; }  
        const char& operator[] ( int i ) const { return arr_[i]; }
        void resize(int n, char c);
        str splice( int i, const str& second ) const;
    }
(a) Write the constructor:

```cpp
str :: str( str::iterator beg, str::iterator end );
```

This takes two string iterators and creates an appropriate-sized string from them. For example

```cpp
str a( "capricorn" );
str::iterator p = a.begin(); ++p; ++p;
str::iterator q = p+3;
str b( p, q );  // Now b == string("pri") and b.size() == 3;
```

(b) Write the member function,

```cpp
void str :: resize(int n, char c);
```

This makes the size of the str equal to \( n \). If \( n \) is less than the current size, \texttt{resize} just reduces the size, but does no reallocation. If \( n \) is greater than the current size, the function fills the new spaces with the value stored in \( c \). It reallocates only if necessary. Here are two examples:

```cpp
str s = "ab";
s.resize( 5, 'c' );
// Now, s == string("abccc")

s.resize( 1, 'd' );
// Now, s == string("a")
```

(c) The job of the \texttt{splice} member function is to create a new \texttt{str} object having the contents of \texttt{second} inserted into the middle of the current \texttt{str}, starting at location \( i \). For example

```cpp
str s1("abcd");  // uses constructor not shown
str s2("efghij"); // uses constructor not shown
str s3;
str s3 = s1.splice( 2, s2 );  // s3 should now have 10 chars: abefghijcd
```

Write the \texttt{splice} function. You may use any of the \texttt{str} member functions shown, but you may not assume the existence of any other \texttt{str} member functions.