Practice Problems

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.)

   Solution:

   ```cpp
class Appointment {
public:
    Appointment( int d, int h, string des );
    Appointment( const Appointment& old );
    bool operator==( const Appointment& other );
    friend ostream& operator<<( ostream& ostr, const Appointment& appt );
private:
    int day;
    int hour;
    string description;
};
```
(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.

Solution:

Appointment::Appointment( int d, int h, string des )
    : day(d), hour(h), description(des)

{}

Appointment::Appointment( const Appointment& old )
    : day(old.day), hour(old.hour), description(old.description)

{}

bool Appointment::operator==( const Appointment& other )
{
    return day == other.day && hour == other.hour
    && description == other.description;
}

ostream& operator<< ( ostream& ostr, const Appointment& appt )
{
    ostr << "Day: " << appt.day << ", Hour: " << appt.hour << ", Description: " << appt.description;
    return ostr;
}
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

```
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`.

**Solution:**

```c
sum = 0;
for ( int *p = intarray; p < intarray+n; p+=2 )
    sum += *p;
```
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;
```

Solution:

a:  x = 20
b:  x = 30, y = 20
c:  *p = 20, *q = 30
d:  x = 30, y = 20
4. Write a `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

```cpp
str str::reverse( ) const;
```

Recall that `str` class objects have three member variables:

```cpp
char * arr_;
unsigned int size_;
unsigned int alloc_;
```

Solution:

```cpp
str str::reverse( ) const
{
    str new_string;
    new_string.size_ = size_;  
    new_string.alloc_ = alloc_;  
    new_string.arr_ = new char[ alloc_ ];

    for ( unsigned int i=0; i<size_; ++i )
    { 
        new_string.arr_[i] = arr_[ size_-1-i ];
    }

    return new_string;
}
```
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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.3</td>
<td>5.2</td>
<td>8.7</td>
<td>0.0</td>
<td>-4.5</td>
<td>7.8</td>
<td>-9.1</td>
<td>3.5</td>
<td>6.6</td>
</tr>
</tbody>
</table>

Then the resulting array containing the negative values would be

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.3</td>
<td>-4.5</td>
<td>-9.1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>8.7</td>
<td>0.0</td>
<td>7.8</td>
<td>3.5</td>
<td>6.6</td>
</tr>
</tbody>
</table>

(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.
Solution: Here’s the whole thing, including the prototype. Note that the pointers must be passed by reference and a separate count variable must be passed by reference for each of the array. See below for a simpler solution based on vectors.

```c
void split( float floatarr[], int n,
        float*& negatives, int& neg_count,
        float*& positives, int& pos_count )
{
    int i;
    neg_count = 0;
    for ( i=0; i<n; ++i )
        if ( floatarr[ i ] < 0 )
            neg_count ++ ;
    pos_count = n - neg_count;

    negatives = new float[ neg_count ];
    positives = new float[ pos_count ];
    int ni = 0, pi = 0;
    for ( i=0; i<n; ++i )
        if ( floatarr[ i ] < 0 )
            {  
                negatives[ ni ] = floatarr[ i ];
                ++ ni;
            }
        else
            {  
                positives[ pi ] = floatarr[ i ];
                pi ++ ;
            }
}
```
(c) Compare this to a version that is based on vectors or lists.

**Solution:** A solution based on vectors (or lists) will create two vectors, *negatives* and *positives* and use the `push_back` member function to store the floats in the appropriate vector.

```c++
void split( const vector<float>& in_floats, 
            vector<float>& negatives, 
            vector<float>& positives )
{
    negatives.clear();
    positives.clear();
    for ( unsigned int i=0; i<in_floats.size(); ++i )
        if ( in_floats[i] < 0 )
            negatives.push_back( in_floats[i] );
        else
            positives.push_back( in_floats[i] );
}
```
6. Our word counting program, discussed in class, created a map of the form

```cpp
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:

```cpp
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);
```

**Solution:**

```cpp
void alpha_to_occurrence( const map<string, int>& wc,
                          map<int, list<string>>& word_order)
{
    word_order.clear();
    for ( map<string, int>::const_iterator wc_p = wc.begin();
         wc_p != wc.end(); ++wc_p )
        word_order[ wc_p->second ] . push_back( wc_p->first );
}
```

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.