CSci 1200
Computer Science II
Test 2 November 6, 2001

Name: ____________________________________________
Section: __________________________________________

1 (15 points): ________________
2 (10 points): ________________
3 (5 points): ________________
4 (20 points): ________________
5 (10 points): ________________
6 (20 points): ________________
7 (20 points): ________________

Total (100 points): ________________

Instructions:

• You have until 11:50 to complete this test.

• You may use nothing to aid you, not even a calculator.

• Put away ALL papers, books, and electronic devices.

• There are no syntax errors anywhere in the code on this exam.

• Please state clearly any assumptions that you have to make in interpreting a question.
1. (15 points) Write a recursive function to multiply two positive integers using only addition and comparison operations. No loops are allowed in the function. The function prototype should be

\[
\text{int Multiply( int m, int n)}
\]

**Solution:**

int Multiply( int m, int n)
{
    if ( n == 0 )
        return 0;
    else
        return m + Multiply( m, n-1 );
}
2. (10 points) Below is the `MergeSort` function given in class. Suppose this function is called with `low==0` and `high==6`. Specify the EXACT recursive calls that are made and the EXACT order in which they are made. You need only show the values of `low` and `high` each time.

```c
void MergeSort( double * pts, int low, int high )
{
    // Base case: an interval of size 1 is already sorted.
    if ( low == high ) return;

    // Compute the middle of the interval and
    // recursively sort the array.
    int mid = (low + high) / 2;
    MergeSort( pts, low, mid );
    MergeSort( pts, mid+1, high );

    // At this point the lower and upper subintervals of "pts"
    // are sorted. All that remains is to merge them
    // into a single sorted list.

    // Allocate scratch space
    double * temp = new double[ high-low+1 ];

    // Merge into the scratch space, use address arithmetic
    // to make the MergeArrays function think it is getting
    // two separate arrays to merge.
    MergeArrays( &(pts[low]), mid-low+1,
                 &(pts[mid+1]), high-mid,
                 temp );

    // Copy back into the pts array.
    for ( int i=low; i<=high; ++i )
        pts[i] = temp[i-low];

    // Delete the scratch space.
    delete [] temp;
}
```
**Solution:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. (5 points) For this question you will need to look at the code for MergeArrays in addition to the code on the previous page for MergeSort.

    void MergeArrays( double a[], int m, double b[], int n,
                      double result[] )
    {
        int i=0, j=0;
        int k=0;

        while ( i<m && j<n ) {
            if ( a[i] < b[j] ) {
                result[k] = a[i];
                ++ i;
            } else {
                result[k] = b[j];
                ++ j;
            }
            ++ k;
        }

        for ( ; i<m; ++i, ++k ) result[k] = a[i];
        for ( ; j<n; ++j, ++k ) result[k] = b[j];
    }

The question is simply this: If we made both MergeSort and MergeArrays templated by replacing double with a template class T, then what functions and operators would need to be defined for any type (class) used in place of T?

**Solution:**
Default constructor,
operator=,
operator<
4. **(20 points)** Write a function that finds the next word in an input stream and stores it in a C++ string object. You can assume that a word is defined as one or more alphabetic characters, possibly with hyphens in the middle or at the end (but not at the beginning). The function should leave capital letters unchanged. The function should return true if and only if a word was input. The function prototype should be

```cpp
bool next_word( istream& str, string& word )
```

As an example, for a stream containing the single line of input

```plaintext
can't can-Can- -good *-&
```

The function should produce strings containing the following on four consecutive calls

- can
- t
- can-Can-
- good

On the fifth call, it should return false.

**Solution:**

```cpp
bool next_word( istream& str, string& word )
{
    char c;
    while ( str >> c && !isalpha(c) )
    {
        // continue
    }
    if ( str.eof() ) return false;
    word = c;
    while ( str.get(c) && (isalpha(c) || c == '-' )
    {
        word += c;
    }
    return true;
}
```
5. **(10 points)** Write a function that takes a doubly-linked, circular list, with a dummy head node, and removes the last node in the list. If the list is empty, the function should do nothing. The templated definition of the linked-list node type is

```cpp
template <class T>
class Node {
    public:
        T data;
        Node<T> * prev;
        Node<T> * next;
};
```

The function prototype is

```cpp
template <class T>
DeleteLast( Node<T>* head )
```

Where `head` is the pointer to the dummy head node.
Solution:

template <class T>
DeleteLast( Node<T>* head )
{
    if ( head -> next == head )
        return;
    Node<T>* p = head -> prev;
    p -> prev -> next = head;
    head -> prev = p -> prev;
    delete p;
}
6. (20 points) Write a function to create a new singly-linked list (non-circular, no dummy node) that is a copy of a sublist of an existing list. The prototype is

\[
\text{DNode* Sublist( DNode* head, int low, int high )}
\]

The DNode class is:

```cpp
class DNode {
public:
    double value;
    DNode* next;
};
```

The new list will contain \( \text{high-low+1} \) nodes, which are copies of the values in the nodes occupying positions \( \text{low} \) up through and including \( \text{high} \) of the list pointed to by \( \text{head} \). The function should return the pointer to the first node in the new list. For example, in the following drawing the original list is shown on top and the new list created by the function when \( \text{low==2} \) and \( \text{high==4} \) is shown below.

**Original list**

```
head ----> 3.1 ----> 2.4 ----> 8.7 ----> 9.4 ----> 14.2 ----> 0.9 ----> 0
```

**New list**

```
nhead ----> 2.4 ----> 8.7 ----> 9.4
```

A pointer to the first node of this new list should be returned. (In the drawing this would be the value of \( \text{nhead} \).) You may assume the original list contains at least \( \text{low} \) nodes. If it contains fewer than \( \text{high} \) nodes, then stop copying at the end of the original list.

(The next page is blank to give you room to write your solution.)
Solution:

```c
DNode* Sublist( DNode* head, int low, int high )
{
    // skip over the first low-1 nodes
    DNode*p = head;
    int i;
    for ( i=1; i<low; ++i ) p = p->next;

    DNode* new_head = new DNode;
    new_head -> value = p -> value;
    DNode * last = new_head;

    for ( ++i, p = p->next; i<=high && p; ++i, p = p->next ) {
        last -> next = new DNode;
        last -> next -> value = p -> value;
        last = last -> next;
    }
    last -> next = 0;
}
```
7. **(20 points)** Suppose that the only data structure you have to work with is a queue of integers. You don’t have a linked list or an array. Suppose further that for your application, you need to read a sequence of integers and output them in increasing order. The key to doing this is to insert values into the queue in the correct order.

Write a function that takes a queue of integers that is already in increasing order (or empty) and adds a new integer in such a way as to maintain the ordering property. Thus, if the queue contains 3, 5, 9, 11 and a 7 is to be added, then after your function the queue will contain 3, 5, 7, 9, 11, in order. The function prototype is

```cpp
void insert_in_order( int x, QueueOfInt& q )
```

You may ONLY use the public queue interface (member functions) specified as follows.

```cpp
class QueueOfInt {
public:
    Queue(); // create an empty queue
    Queue( const QueueOfInt& old); // copy an existing queue
    ~Queue(); // destroy a queue
    void enqueue( int item ); // add item to back of queue
    void dequeue(); // remove front of queue
    void dequeue( int& item ); // remove front of queue and place in item
    int getFront( ) const; // return front of queue
    bool isEmpty( ) const; // is the queue empty?
};
```

You will need to use a second queue. It is possible to solve this problem with only one queue, but it is not easy (if the values are unique it becomes easier).
Solution:

```c
void insert_in_order( int x, QueueOfInt& q )
{
    if ( q.isEmpty() )
        q.enqueue( x );
    else {
        QueueOfInt temp( q ); // copy q;
        while ( !q.isEmpty() ) q.dequeue(); // empty q out

        while ( !temp.isEmpty() && x > temp.getFront() ) {
            int item;
            temp.dequeue( item );
            q.enqueue( item );
        }

        q.enqueue( x );
        while ( !temp.isEmpty() )
            int item;
            temp.dequeue( item );
            q.enqueue( item );
    }
}
```