Name: ______________________________

Write all of your responses on these exam pages. If you need extra space please use the backs of the pages. The Short Answer questions are worth 5 points each. The Coding questions are worth 35 points each.

1 Short Answer

1. Write a recursive function that will compute the double factorial. The double factorial is defined as

$$n!! = n \cdot (n-2) \cdot (n-4) \cdots 1$$

and $0!! = 1$. For example, $3!! = 3$, $4!! = 8$, $5!! = 15$, $6!! = 48$, $7!! = 105$, $\ldots$.

2. Write a recursive function to solve the Towers of Hanoi problem. The setup and initial function call are below.

```
const int FROM_PEG = 1;  // Initial "from" peg
const int TO_PEG = 3;    // Initial "to" peg
const int TEMP_PEG = 2;  // Initial "temp" peg
moveDiscs(NUM_DISCS, FROM_PEG, TO_PEG, TEMP_PEG);
```

3. What do LIFO and FIFO mean and what data structures use each of these types of access?

4. In each case below state what is faster to execute.

   (a) Recursion or iteration?

   (b) A dynamic stack implemented with an STL vector or with an STL list?

   (c) A static queue implemented with an STL vector or a circular array?

   (d) Accessing the 100^{th} element out of an STL vector or the 100^{th} element out of a linked list?

   (e) Removing the first element out of an STL vector or removing the first element of a linked list?

5. If the numbers 2, 7, 4, 5, 9, 10, 4 were pushed on to a stack and then popped off and printed to the screen display the output.

6. Write a function that accepts an STL vector of integers. The function should recursively calculate the sum of all the numbers in the vector. No loops are needed or allowed. As an example, the following code will produce an output of 224.

```
vector<int> v;
v.push_back(5);
v.push_back(21);
v.push_back(36);
v.push_back(42);
v.push_back(54);
v.push_back(66);
cout << sum(v) << endl;
```

2 Coding Exercise #1

This exercise is to code a general templated linked list class structure. The specification for the class is below.

```
template<class T>
class ListNode {
public:
    T value;
    ListNode<T> *next;

    ListNode(T nodeValue) {
        value = nodeValue;
        next = nullptr;
    }
};

template<class T>
class LinkedList {
protected:
    ListNode<T> *head;
    void displayListRec(ListNode<T>*) const;
    void displayListReverseRec(ListNode<T>*) const;
    int numNodes(ListNode<T>*) const;

public:
    LinkedList();
    virtual ~LinkedList();
    void appendNode(T);
    void insertNode(T);
    void deleteNode(T);
    void clear();
    bool isEmpty();
    void displayList() const {
        displayListRec(head);
    }
    void displayListReverse() const {
        displayListReverseRec(head);
    }
    int length() const {
        return numNodes(head);
    }
};
```

The description of the functions are below, code each function. In all cases there are to be no memory leaks in the implementations.

- The constructor and destructor should do their obvious tasks.
- appendNode will add the new node to the end of the linked list.
- insertNode inserts the new item so that all the items before it are less then it and the next item in the list is greater then or equal to the one inserted.
- deleteNode removes the first occurrence of the node with the specified value.
- clear removes all the nodes from the list.
- isEmpty returns true if the linked list is empty and false otherwise.
- displayList simply calls the function displayListRec with the parameter head. The displayListRec will display the contents of the linked list to the screen. This function is to be recursive, no loops.
- displayListReverse simply calls the function displayListReverseRec with the parameter head. The displayListReverseRec will display the contents of the linked list in reverse order to the screen. This function is to be recursive, no loops.
- length simply calls the function numNodes with the parameter head. The numNodes will return the number of nodes in the list. This function is to be recursive, no loops.

The code for a test program is below along with the output of the code.

Sample Code

```
int main() {
    LinkedList<int> iList;
    iList.appendNode(3);
    iList.appendNode(5);
    iList.appendNode(-1);
    iList.appendNode(17);
    iList.displayList();
    cout << endl;
    iList.displayListReverse();
    cout << endl;
    cout << iList.length() << endl;
    iList.clear();
    cout << iList.length() << endl;
    cout << iList.isEmpty() << endl;

    for (int i = 0; i < 10; i++)
        iList.insertNode(rand() % 100);

    iList.displayList();
    cout << endl;
    iList.displayListReverse();
    cout << endl;

    iList.deleteNode(49);
    iList.deleteNode(86);
    iList.deleteNode(26);
    iList.displayList();
    cout << endl;

    return 0;
}
```

Output

```
3 5 -1 17
17 -1 5 3
4
0
1
15 21 35 49 77 83 86 86 92 93
93 92 86 86 83 77 49 35 21 15
15 21 35 77 83 86 92 93
```

3 Coding Exercise #2

Create a class named StackQueue that inherits off of the LinkedList class from the first exercise. This class will have functions for both the stack and queue data structures, hence, like structures in the STL it could be used for both.

Write both the specification and implementation for the class. There should be a constructor, default, and a destructor. The class also needs to implement push and pop for the stack and enqueue and dequeue for the queue. Push and pop should add and remove elements from the front of the linked list with pop returning the value of the node. Enqueue should add nodes to the back and dequeue remove elements from the front of the linked list, and as with pop it should return the value of the node. If the list is empty then pop and dequeue will return the default value of the templated data type.

Once this is complete construct a main program that uses this derived class. The main will declare two objects of StackQueue type, called stack and queue. The stack object will only use stack operations and queue will only use queue operations. The program will determine if an input string is a palindrome or not. Here is the way it will work, this is a standard method for determining a palindrome. Take a string from the user, you may assume that there are no punctuation or white space and that the case of all the characters is the same. Put each character on both a stack and a queue. Then while the stack is not empty, pop the stack and dequeue the queue, compare the two letters that were removed. If all the letters are the same for each pop and dequeue the phrase was a palindrome, if any pair are different then the phrase was not a palindrome.