UCLA Computer Science Department TA: Kung-Hua Chang

Student Name and ID _____

CS 32, WINTER 2015, PRACTICE MIDTERM I.

Problem #	Maximal Possible Points	Received
1.1	3	
1.2	5	
1.3	5	
1.4	5	
1.5	5	
2	3	
3.1	4	
3.2	5	
Total	35	

Problem #1: Below is a definition of a LinkedList class and a test main program:

```
#include <iostream>
using namespace std;
class LinkedList
{
public:
	LinkedList(): head(nullptr) { }
	~LinkedList();
	void addToList(int value);    // add to the end of the linked list
	void reverse();                    // Reverse the linked list
	void output();
private:
	struct Node
	{
		int num;
		Node *next;
	};
	Node *head;
};
void LinkedList::output()
{
	Node *ptr = head;
	cout << "The elements in the list are: ";
	while(ptr!=nullptr) {
		cout << ptr->num << " ";
		ptr = ptr->next;
	}
	cout << endl;
}
int main()
{
	LinkedList list;
	for(int i=1;i<=10;i++)	list.addToList(i);
	list.output();
	list.reverse();
	list.output();
}
```

Problem #1.1: Please complete the implementation of the **destructor**.

```
LinkedList::~LinkedList()
{
	Node *temp;
	while(head != nullptr) {
		temp = head;
		head = head->next;
		delete temp;
	}
}
```

Problem #1.2: Please write an implementation of the **addToList** member function.

```
void LinkedList::addToList(int value)
{
	Node *nodeToAdd = new Node;
	nodeToAdd->num = value;
	nodeToAdd->next = nullptr;
	if( head == nullptr)
		head = nodeToAdd;
	else {
		Node *ptr = head;
		while(ptr->next != nullptr)
			ptr = ptr->next;
		ptr->next = nodeToAdd;
	}
}
```

Problem #1.3: Please complete the implementation of the **reverse** member function.

```
void LinkedList::reverse()
{
    Node *nextNode = nullptr,*prevNode = nullptr, *current = head;
    while(current) {
    // Hint: Only 4 lines of codes are needed inside the while loop
        nextNode = current->next;
        current->next = prevNode;
        prevNode = current;
        current = nextNode;
    }
    head = prevNode;
}
```

Problem #1.4: Suppose we add a new member function called findNthFromLast() to find the N-th node from the end of the list, where N being 1 means the last node, N being 2 the second-to-last, etc. Use the reverse member function to complete the implementation of findNthFromLast() member function. If the list has at least N nodes, then assign to the variable value the number that is stored in that node and return true; otherwise, leave the variable value unchanged and return false. Don't forget to call the reverse function to restore the linked list to its original state.

```
bool LinkedList::findNthFromLast(int N, int &value)
{
    reverse();
    Node *ptr = head;
    int n = 0;
    while(ptr != nullptr) {
        n++;
        if(n == N) {
            value = ptr->num;
            reverse();
            return true;
        }
        ptr = ptr->next;
    }
    reverse();
    return false;
}
```

Problem #1.5: There is a more efficient way to solve problem #1.4 without using reverse() function. The idea is to scan first to count the total number of nodes in the linked list. Once we know how many nodes are in the linked list, the second time we scan the list, the Nth node from the end of list can be easily calculated. Please use this idea to implement this member function findNthFromLast().

```
bool LinkedList::findNthFromLast(int N, int &value)
{
	if( N <= 0 ) return false;

	Node *ptr = head;
	int i, M = 0;
	// M is the number of nodes in the linked list.
	while(ptr != nullptr)
	{
		M++;
		ptr = ptr->next;
	}

	if( N > M ) return false;

	for( i=1, ptr = head; i < (M-N+1); i++)
		ptr = ptr->next;
	value = ptr->num;

	return true;
}
// Your implementation should pass the following test program
int main()
{
	int value = 999;
	LinkedList list;
	for(int i=1;i<=10;i++)
		list.addToList(i);
	assert( ! list.findNthFromLast(11,value) && value == 999);
	assert( list.findNthFromLast(1,value) && value == 10);
	assert( list.findNthFromLast(2,value) && value == 9);
	assert( list.findNthFromLast(10,value) && value == 1);
}
```

Problem #2: Please complete the missing blocks of codes below to make the program generate the following output:

1+2i
6+7i
6+7i
6+7i

```
#include <iostream>
using namespace std;
class Complex {
private:
	double r,i;
public:
	// Please complete the missing codes below.
	Complex(): r(0),i(0) {}
	Complex(int c_r,int c_i): r(c_r),i(c_i) {}

	void output() {
		cout << r << "+" << i << "i" << endl;
	}
};

int main() {
	Complex a,b;

	a = Complex(1,2);	a.output();
	b = Complex(6,7);	b.output();

	a = b;

	a.output();
	b.output();

	return 0;
}
```

Problem #3: The implementation below will CRASH because it's missing a copy constructor and an assignment operator.

```
#include <iostream>
using namespace std;

class Triangle    {
public:
      Triangle() {
            p = new Point[3];
      }
      Triangle(int x1,int y1,int x2, int y2,int x3,int y3) {
            p = new Point[3];
            p[0].x = x1;    p[0].y = y1;
            p[1].x = x2;    p[1].y = y2;
            p[2].x = x3;    p[2].y = y3;
      }
      Triangle::~Triangle() {      delete [] p;       }
private:
      struct Point {
            int x,y;
            Point(int px=0,int py=0): x(px), y(py) { }
      };
      Point *p;
};

int main()    {
      Triangle *array[3];
      array[0] = new Triangle(1,1,1,3,3,1);
      array[1] = new Triangle(2,2,2,6,6,2);
      array[2] = new Triangle(3,3,3,9,9,3);
      Triangle c2 = *array[0];
      c2 = *array[1];

      for(int i=0;i<3;i++)
            delete array[i];
}
```

Problem #3.1: Please write an implementation of the copy constructor.

```
Triangle::Triangle(Triangle &t)
{
	p = new Point[3];

	for(int i=0;i<3;i++)
		p[i] = t.p[i];
}
```

Problem #3.2: Please write an implementation of the assignment operator.

```
Triangle& Triangle::operator=(const Triangle &t)
{
	if(&t == this)
		return *this;

	delete [] p;

	p = new Point[3];

	for(int i=0;i<3;i++)
		p[i] = t.p[i];

	return *this;
}
```