**Node Based List Implementation File**

class List {

  public:

    List(int ignored = 0);

    List(const List& other);

    List& operator=(const List& other);

    ~List();

    void insert(const char& newDataItem) throw (logic\_error);

    void remove() throw (logic\_error);

    void replace(const char& newDataItem) throw (logic\_error);

    void clear();

    bool isEmpty() const;

    bool isFull() const;

    void gotoBeginning() throw (logic\_error);

    void gotoEnd() throw (logic\_error);

    bool gotoNext() throw (logic\_error);

    bool gotoPrior() throw (logic\_error);

    char getCursor() const throw (logic\_error);

  private:

    class ListNode {

      public:

ListNode(const char& nodeData, ListNode\* nextPtr);

char dataItem;

ListNode\* next;

    };

    ListNode\* head;

    ListNode\* cursor;

};

**Insert Function**

Void List::insertAfter(const char& data)

{

if (head == NULL)

{

head = new Node (data, NULL);

cursor = head;

return;

}

cursor -> next = new Node (data, cursor -> next );

cursor = cursor -> next;

}

List::ListNode::ListNode (const char& nodeData, ListNode\* nextPtr)

{

dataItem = nodeData;

next = nextPtr;

}

**Overloaded Assignment Operator**

List& List::operator=(const List& other){

if (this == &other)

{

return \*this;

}

Node\* ptr = other.head;

clear();

head = new Node (ptr->data, NULL);

Node\* tmp = head;

if(ptr = other->cursor)

{

cursor = tmp;

}

while{ptr -> next != NULL)

{

ptr = ptr->next;

tmp->next = new Node(ptr->data, NULL);

tmp = tmp->next;

if(ptr = other.cursor)

{

cursor = tmp;

}

}

return \*this;

}



 