# CSC 232: Data Structures and Algorithms

**Asn6: List ADT**

**Due: 12/01 @ 5pm**

*Submit your program in one file in BB by midnight on the due date. Email your programs to me as a last resort if you experience problems with BB.*

**Objectives**

* Implement the List ADT
* Implement the Position ADT
* Fully understand iterators in the STL by implementing them
* Explore the nested class relationship and implementation details

**Requirements**

* Program structure – the requirements to get nested template classes to work is fairly strict, meaning that certain code components must go in specific locations and/or in a specific order. See the provided code below.
* Update your SLinkedList class from Asn4 to match the declaration in the provided code.
* Use the updated SLinkedList to derive the List class. Add and test the following methods first because they do not rely on an iterator:

back(): Return the last element

pop\_back(): Remove and return the last element from L

push\_back(elem): Append element to L

* Add the Iterator class and associated methods and fields as shown in the provided code. Test these methods, then continue.
* Complete the List class by adding the following methods:

begin(): Return an iterator to the first element of the list

end(): Return an iterator to the end of the list\*

insert(itr, elem): Insert element into the list before itr

erase(itr): Remove the element at itr

\*See Limitations.

* Implement the Iterator class as a nested class of List per the provided code.

operator\*: Dereference the iterator and return an element value

Iterator<E> operator++: operator++: increment the iterator

Iterator<E> operator--: decrement the iterator

bool operator==: test equality of two iterators

bool operator!=: test inequality two of iterators

* The following exceptions must be handled wherever they might occur:
  1. itr was never initialized or was set to a position in a different list
  2. itr results from an illegal operation, such as ++L.end(), i.e. attempting to access a position outside of the list.
* Include a completed comment template at the top of your program and use good coding style per the coding style cheat sheet.

**Limitations**

There are a few limitations to this implementation due to the fact that there are no beginning or ending sentinels.

* 1. Cannot get to the first node in list by decrementing begin()
  2. Cannot have end() return an iterator to the last node in the list.
  3. Must treat the following as separate cases when using iterators to insert into a list with only one node to get correct behavior:
     1. Insert at begin()
     2. Insert at end()

**Extra Credit**

These two opportunities for extra credit are each worth 3% of the total available points. They can be independent of each other.

1. Implement a recursive merge sort function (not method) and execute on a list of 1000 random integers in [0..10000]. The sort will move the values between nodes via subscripting – it will not modify the structure of the list. See [www.sorting-algorithms.com/merge-sort](http://www.sorting-algorithms.com/merge-sort) for more info.
2. Implement a binary search function (not method) on a previously sorted list of 1000 random integers in [0..10000]. You can do #1 or grab your insertionSort() method from Asn4 to sort the list.

**Legal**

* You may use only language features discussed in class or presented in the book up to the date the assignment is due.
* Your submission must be your own work. You may not utilize any code outside of that provided in lecture or from the book nor may you post any provided code on publicly accessible websites.

**Nested Class Structure**

/// All the stuff at the beginning here. ///

/\*\* This class implements a singly-linked list ADT. \*\*/

template <typename E>

class SLinkedList {

public:

/\* Constructors & destructor \*/

SLinkedList(); //default

SlinkedList(const SlinkedList<E>& sll); //copy

~SLinkedList(); //destructor

/\* ADT methods \*/

bool empty() const; //list empty?

E front() const; //return 1st element

E pop\_front(); //return 1st element & delete node

void push\_front(const E e); //insert at front

int size(); //return size of SLL

/\* For overloads \*/

void swap(SLinkedList<E> rhs); //for copy & swap

SLinkedList& operator= (SLinkedList<E> rhs);

E operator[] (const int index);

friend ostream& operator<< <E>(ostream& out, const SLinkedList<E>& sll);

protected:

Node<E>\* head;

int sllsize;

}; /\* end SlinkedList class declaration \*/

// Definitions for SLinkedList go here as usual.

/\*\* End SLinkedList class \*\*/

/\*\* List class \*\*/

template <typename E>

class List : public SlinkedList<E> {

public:

// NOTE THE DIFFERENT LETTER – IT IS ONLY USED HERE!

// Use E everywhere else! m

// For a nested class, methods are declared and defined \*INSIDE\*

// the class declaration.

template <typename I>

class Iterator {

public:

// Give List access to Iterator private fields.

friend class List<E>;

// These are the minimum methods needed.

E operator\* { } //dereference the iterator and return a value

Iterator<E> operator++ { } //increment the iterator

Iterator<E> operator-- { } //decrement the iterator

bool operator== { } //test equality of iterators

bool operator!= { } //test inequality of iterators

private:

// Constructors & destructor here since only want List class to access.

// List constructor called from List::begin(). Use initializer list or

// create class copy constructor and assignment overload.

Iterator(const List<E>\* sl) : llist(sl) {

nodePosition = sl->head;

}

// Class fields.

const List<E>\* llist; //give Iterator class a handle to the list

Node<E>\* nodePosition; //abstracted position is a pointer to a node

}; /\*\* end Iterator class \*\*/

/\* The Iterator class is now fully defined. The rest of these

statements must go AFTER the Iterator class or the compiler

won’t have complete information about their data types.

\*/

// REQUIRED: While not necessary for the code to work, my test suite needs

// this defined. Create a less cumbersome name for Iterator<E>. Use

// anywhere you would have used List<E>::Iterator<E> in class List. Allows

// this syntax in main() -- List<int>::iterator instead of List<int>::Iterator<int>.

typedef typename List<E>::Iterator<E> iterator;

/\*\*\* All method declarations and fields for the List class go here.

Any method that returns an iterator must be defined here.

\*\*\*/

iterator begin() const { //return an iterator of beginning of list

// Call iterator constructor with pointer to List that begin() was

// called with.

return iterator(this);

}

}; /\*\* end List class declaration \*\*/

// Remaining definitions for List class

/\*\* End List class \*\*/