

# VE 280 Lab 9

**Out:** 00:01 am, July 15, 2021; **Due:** 11:59 pm, July 22, 2021.

The range of `int` is -2147483648 ~ 2147483647; the range of `long int` is -2147483648 ~ 2147483647 on 32-bit platforms and is -9223372036854775808 ~ 9223372036854775807 on 64-bit platforms; the range of `long long int` is -9223372036854775808 ~ 9223372036854775807. What if we want to store an integer with value 1000000000000000000000?

We could create an ADT to represent large numbers! The number is represented by a linked list and each element stored in the linked list is an integer in the range 0 ~ 9.

## Ex. 1 Templated Singly-Linked List

Related topics: *template, linked list, deep copy*

To get you familiar with templates, the singly-linked list you are going to implement is a templated linked list, which is given in `mylist.h` and shown below:

```
// an exception class
class EmptyList{};

template <class T>
struct Node_t{
    Node_t* next;
    T val;
};

// singly-linked list
template <class T>
class List{
private:

    Node_t<T>* first;
    Node_t<T>* last;

    void removeAll();
    // EFFECTS: called by destructor/operator= to remove and destroy
    //           all list elements

    void copyFrom(const List &l);
    // MODIFIES: this
    // EFFECTS: called by copy constructor/operator= to copy elements
    //           from a source list l to this list;
    //           if this list is not empty originally,
    //           removes all elements from it before copying

public:
    bool isEmpty() const;
    // EFFECTS: returns true if list is empty, false otherwise

    void insertBack(T val);
    // MODIFIES: this
```

```

// EFFECTS: inserts val at the back of the list

T removeFront();
// MODIFIES: this
// EFFECTS: removes the first element from
//           non-empty list and returns its value
//           throws an instance of EmptyList if empty

const Node_t<T>* returnFront() const;
// EFFECTS: returns first

void print();
// EFFECTS: print the elements in the list

List(); // constructor
List(const List &l); // copy constructor
List &operator=(const List &l); // assignment operator
~List(); // destructor
};

```

Since this linked list needs to support `insertBack` and `removeFront` for later functions, it contains both a pointer that points to the first node `first` and a pointer that points to the last node `last`.

`returnFront` is used to return the pointer that points to the first node `first` in the list, so that you can use it to iterate through the whole list. We don't want the value of `first` to be changed outside the class, so there is a `const` before `Node_t` in the function declaration; we also don't want this function itself to change any member in this class, so there is another `const` at the end of the function declaration.

`print` is already implemented in `mylist_impl.h`, which prints elements in the list in order. Please do not modify it.

Since dynamically allocated storage occurs in this class, you must also provide a destructor, a copy constructor, and an assignment operator.

Here is a demo for `insertBack` and `removeFront`:

```

List<int> a;
a.insertBack(4); // 4(first & last)
a.insertBack(1); // 4(first) -> 1(last)
a.insertBack(3); // 4(first) -> 1 -> 3(last)
a.removeFront(); // 1(first) -> 3(last)

```

## Ex. 2 Which one is larger?

As mentioned above, you are going to use this linked list to store a large integer. And you want to provide a function to compare two integers stored in two linked lists. In order to make the implementation of this function to be easier, an integer is represented "reversely". For example, integer `415` is represented by:

```
List<int> a;
a.insertBack(5);
a.insertBack(1);
a.insertBack(4); // 5 -> 1 -> 4
```

The comparison function you need to implement is

```
bool isLarger(const List<int> &a, const List<int> &b);
// EFFECTS: returns true if the number represented by a
//           is larger than the number represented by b;
//           otherwise, returns false.
//           returns false if both a and b are empty
```

Example:

```
List<int> a, b;
a.insertBack(5);
a.insertBack(4);
a.insertBack(3); // a = 345
b.insertBack(2);
b.insertBack(4);
b.insertBack(1);
b.insertBack(3); // b = 3142
isLarger(a, b);  // false
```

## Ex. 3 Addition

Addition is a basic operation on integers. You want to implement this for `List<int>`. The representation of an integer by a `List<int>` is the same as in **Ex. 2**.

```
List<int> add(const List<int> &a, const List<int> &b);
// EFFECTS: adds the numbers represented by a and b; returns the result
```

Example:

```
List<int> a, b;
a.insertBack(5);
a.insertBack(3); // a = 35
b.insertBack(2);
b.insertBack(4); // b = 42
List<int> ab_sum = add(a, b); // ab_sum = 77
```

## Submission

`mylist.h` and `mylist_impl.h` can be found in `lab9_starter_files` on Canvas. Please implement the linked list methods and another two functions in `mylist_impl.h`. Submit it as a tar or zip file via online judgement system.

There is also a `simpletest.cpp` for you to test the linked list you implemented. Please check and make sure there is no memory leak.

---

Created by Yaxin Chen.  
Updated by Yanjun Chen.  
Last update: July 13, 2021  
@UM-SJTU Joint Institute