# CSCE2014 Programming Foundations II
# Homework Five

By Wing Ning Li

## 1 Problem Description

In our previous homework, we have developed Mystring class that is similar to C++ standard library string class. In this homework, we will develop **Linkedlist** class that is similar to C++ standard library list class, where like vector class we used in homework three, list is a template class. However, our Linkedlist class is not a template class. Our `Linkedlist` class is basically the same as `list<int>`, that is a linked list of integers.

C++ standard library `list<>` is implemented using a doubly linked list. So we will use a doubly linked list to implement the **Linkedlist** class. We have gained some experiences in the linked list Labs to manipulate a linked list. The Node structure introduced in Lab 8 can be used as the basic node data structures to implement **Linkedlist**. Note that in the labs, we have an `element_type` that is introduced through typedef and is equivalent to int. We will keep the same set up here.

Since we are going to implement C++ library list class, we need to understand the list class better from the outside. So we will first use list class and see how each member functions that we will implement work, which is similar in spirit to what we have done in string labs (5 and 6) before we implement the Mystring class. Then we will replace `list<int>` by `Linkedlist`, and if we implement everything correctly, the code should compile and produce the same results.

## 2 Purpose

Understand class, class members and member functions (methods), constructor, destructor, overloading an operator as a member function, memory management, doubly linked list, and C++ `list<>` class. To gain experience and understanding related to how a complex and useful class such as C++ `list<>` may be designed and implemented using basic C++ language features, data structures, and algorithms.

## 3 Design

We have had experiences of using the .cpp and .h files in previous Labs and Homework. For this assignment we will do the same as well. Use `Linkedlist.h` file for class declaration, `Linkedlist.cpp` file for class implementation, and `homework5.cpp` for testing the implementation.

In the design, the **Node** structure (introduced by the keyword struct) should be part of the Linkedlist, that is, it is introduced inside the class as private type.

Since struct is a class where all its members are public, so ***we will have a class declared inside another class here***. To really mimic the list, we also need an iterator class inside Linkedlist class. However, the design and implementation of iterator class and related methods will **NOT** be part of this homework.

The Linkedlist class has the following public methods and types (these are also called interface), the behavior of which is the same as in the C++ library list class:

```
 typedef element_type& reference;
 typedef const element_type& const_reference;
 Linkedlist(); //default constructor for empty list
~Linkedlist(); //destructor to free nodes dynamically created to support the linklist
 bool empty() const;
 void clear();
 reference back();
 const_reference back() const;
 reference front();
 const_reference front() const;
 Linkedlist& operator=(const Linkedlist& l);
 void pop_back ( );
 void pop_front ( );
 void push_back ( const element_type& x );
 void push_front ( const element_type& x );
 void sort ( );
```

Since iterator class is not part of the homework, we will have the following additional public methods that are not in C++ list:

```
// constructor that initializes the linked list with n nodes,
// with elem value from 0 to n-1
explicit Linkedlist(unsigned int n);

// print the linked list in the forward direction,
// similar to the show function of lab7
void check() const;

// print the linked list in the backward direction,
// similar to the reverse_show function of lab8
void rcheck() const;

// insert a node with value specified by x after the node
// specified by pos. The first node has position 0.
// if the number of nodes in the linked list is less than
// pos, the node is inserted at the end.
void insert(unsigned int pos, const element_type& x);

// remove the node specified by pos.
```

```
    // if the number of nodes in the linked list is less than
    // pos, the node at the end if any is removed.
    void erase(unsigned int pos);
```

## 4 Implementation

We should have a `Linkedlist.h` file for the class declaration, a `Linkedlist.cpp` file for the class implementation, and the main function in the `homework5.cpp` file. In the main function, first use C++ `list<int>` member functions that are identical or similar to those member functions to be implemented (you are welcome to try other member functions as well), then test the member functions of Linkedlist in a similar way including member functions of Linkedlist not in C++ list class.

## 5 Test and evaluation

In the main program, demonstrate the functionality and correctness of each implemented member function or operator of the Linkedlist class.

## 6 Report and documentation

A short report about things observed and things learned and understood. The report should also describe the test cases used in the main program and the reasons for each test case selected. Properly document and indent the source code. The source code must include the author name and as well as a synopsis of the file.

## 7 Lab submission

Use Blackboard to submit the source files (Linkedlist.h, Linkedlist.cpp, homework5.cpp) and the report.