# University of Wollongong

**School of Computing and Information Systems**

**CSCI251/851** **Advanced Programming** **Autumn 2017**

**Assignment 3** **(Due: 11.59pm Week 8, Wednesday 26 April)** **8 marks**

## Aim:

This assignment is to familiarise you with the use of container classes in your programs.

**On completion you should know how to:**

- Write container class objects with overloaded operators.
- Implement programs incrementally to minimise debugging.
- Custom design container class objects for software applications.

## Requirements:

Complete the following class object by writing the code needed in the provided files. Much of this is explained in the week-4 lecture notes "4b C++ Classes - book.pdf". After completing each step, test that your code works by writing appropriate code in main.cpp.

## Step 1 - 4 marks

Implement all public member functions of the linked list class declared below in files list.h and list.cpp. Note: RemoveHead() returns true if an item is successfully removed from the list or false if unsuccessful (i.e. the list is empty). Also, implement the main() function given below in main.cpp. Modify the main() function by inserting code for testing all the list's member functions. Your main() should print on the screen the test being performed and produce output showing that the test case works. (e.g. "Testing AddTail()", Testing RemoveHead()", "Testing destructor")

```
struct Node
{
    int Item;
    Node *Next;
};

class LinkedList
{
    public:
        LinkedList();
        ~LinkedList();
        void AddTail(int Item);     // adds item to tail
        bool RemoveHead(int &Item);// removes item from head
        void Print();               // prints list. eg 12 34 21 26
    private:
        Node *Head;
};

int main()
{
    LinkedList L1 . . .

    // Put your code for testing your list functions here

    return 0;
}
```

## Step 2 (2 marks)

Also implement AddHead() and RemoveTail() public member functions. AddHead() adds an item to the head of the list. RemoveTail() removes an item from the tail of the list and returns true if successful, or false if the list is empty. Modify the main() function by adding code for testing these functions. Make sure you print on the screen the test being performed and produce output that shows that the test worked.

## Step 3 (1 mark)

In your linked list class, also implement a copy constructor that makes a deep copy of the list argument. Add code for testing the copy constructor on empty and non-empty lists. Make sure you print on the screen the test being performed and produce output that shows that the test worked.

## Step 4 (1 mark)

Implement an assignment operator in your LinkedList class. The assignment operator should ensure multiple assignments are possible. eg the following statements should make A and B identical to C.

```
List A, B, C;
for(int i=0; i<10; i++)
      C.AddHead(i);
A = B = C;
```

Ensure there are no memory leaks and all lists have their own separate memory. You should not assume the left-hand operand is always empty. Add code in the main for testing the assignment operator on both empty and non-empty lists.

## Step 5 (for CSCI851 students. 1 mark deduction if not done.)

Implement the addition operator (+) in your LinkedList class. Your addition operator should enable a concatenated list to be assigned to another list. E.g.

```
LinkedList L1,L2,L3;

... // code for adding data to lists

L3 = L1 + L2; // makes L3 a concatenated list of L1 and L2
```

Add code to the main() for testing the addition operator on both empty and non-empty lists.

## Submit:

Submit your files using the submit facility on UNIX as shown below:

```
$ submit -u login -c CSCI251 –a3 main.cpp list.h list.cpp output.txt
```

**where 'login' is your UNIX login ID**
Note: CSCI851 should also submit to –c CSCI251.

**<u>You must demonstrate your program in the lab on week 9. Failure to demo on time will result in a 1 mark deduction for each week late.</u>**

Deductions will be made for untidy work or for failing to comply with the submission instructions. Requests for alternative submission arrangements will only be considered before the due date. An extension of time for the assignment submission may be granted in certain circumstances. Any request for an extension of the submission deadline must be made to the Subject Coordinator before the submission deadline. Supporting documentation must accompany the request for any extension. Late assignment submissions without granted extension will be marked but the marks awarded will be reduced by 1 mark for each day late. Assignments will not be accepted if more than three days late.