CISP 430 Data Structures
Project 3: List

We will implement, as a **C++** class Abstract Data Type, doubly linked list: **DLList**. Our **DLList** will be implemented as a class using an **ItemType** node with doubly linked structures. This list must work for any type that we declare to be **ItemType**. Test your **DLList** with **STRING**
It will have the following functionality:

```
Post: an empty DLList is created
DLList ( void );

Post: A DLList is destroyed.
~DLList ( void );

Post: RETVAL[1] == The list is empty
bool  IsEmpty ( void ) const;

Post: RETVAL == Current is after last item in the list or
before the first item in the list
bool EndOfList ( void ) const;

Pre: !IsEmpty()
Post: The cursor is moved to the first item in the list
void Reset ( void );

Pre: !IsEmpty() && !EndOfList()
Post: the cursor is moved to the next item in the list
void Advance ( void );

Pre: !IsEmpty() && !EndOfList()
Post: the cursor is moved to the previous item
void Retreat ( void );

Pre: !IsEmpty() && !EndOfList()
Post: RETVAL == Item at the cursor
ItemType CurrentItem( void );

Pre: !IsEmpty()&& !EndOfList()
Post: Item at the cursor is deleted && the cursor points to the
successor of deleted item or IsEmpty() is true if the item
deleted was the last item in the list
void Delete( void );
```

1 RETVAL: Function return value

Pre: None
Post: If the list was empty then *Inserted* is the only item in the list. If EndOfList was true then *Inserted* is the new first item in the list. Otherwise, *Inserted* is the predecessor of the item that was current when the function was called. *Inserted* is the new current item.
void InsertBefore (/*in*/ const ItemType& *Inserted*);

Pre: None
Post: If the list was empty then *Inserted* is the only item in the list. If EndOfList was true then *Inserted* is the new last item in the list. Otherwise, *Inserted* is the successor of the item that was current when the function was called. *Inserted* is the new current item.
void InsertAfter (/*in*/ const ItemType& *Inserted*);

Post: List is displayed to standard out. This is used for debugging only.
void Display (void) const;

You are not being asked to provide a test driver, however you must carefully test your classes to ensure that they work correctly. When working on a large project, it is important that naming conventions are understood and followed by the groups responsible for the separate units. For this assignment I will be providing a test driver. It is important that you name your functions **EXACTLY** the same as those in these specifications so that my test driver can use your classes. You will be marked-down if you vary from the specifications.

(Remember: **C++** is case sensitive!!!!!)

File names:

- DLList.h
- DLList.cpp

Due: **See the drop box**