DTS Lab 4  
doubly linked list and iterator

Contents

[Objective 2](#_Toc358034584)

[Prototypes 2](#_Toc358034585)

[Desired Output 5](#_Toc358034586)

[Submission 6](#_Toc358034587)

# Objective

You will implement a templated, double-ended, doubly-linked list class named DLList and an iterator class named DLLIter to access the DLList.

Place all your code for both classes in a file named *DLList.h*.

# Data Members

The DLList class will have the following private data members:

Node\* head = nullptr;  
Node\* tail = nullptr;  
unsigned int Size = 0;

The DLLIter class will have the following private data members:

DLList<Type>\* ptrList;  
typename DLList<Type>::Node\* curr;

# Prototypes

The DLList class will have the following public interface:

/////////////////////////////////////////////////////////////////////////////  
// Function : Constructor  
// Notes : Does nothing  
/////////////////////////////////////////////////////////////////////////////  
DLList()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Destructor  
// Notes : Destroys a list (call clear())  
/////////////////////////////////////////////////////////////////////////////  
~DLList()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Assignment Operator  
// Notes: Deep copy of that list. Can be done with a loop  
/////////////////////////////////////////////////////////////////////////////  
DLList<Type>& operator=(const DLList<Type>& that)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : Copy Constructor  
// Notes: Call assignment operator  
/////////////////////////////////////////////////////////////////////////////  
DLList(const DLList<Type>& that)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : addHead  
// Parameters : v - the item to add to the head of the list  
/////////////////////////////////////////////////////////////////////////////  
void addHead(const Type& v)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : addTail  
// Parameters : v - the item to add to the tail of the list  
/////////////////////////////////////////////////////////////////////////////  
void addTail(const Type& v)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : clear  
// Notes : clears the list, freeing any dynamic memory  
/////////////////////////////////////////////////////////////////////////////  
void clear()

/////////////////////////////////////////////////////////////////////////////  
// Function : insert  
// Parameters : index - an iterator to the location to insert at  
// v - the item to insert  
// Notes : do nothing if index current pointer is null  
/////////////////////////////////////////////////////////////////////////////  
void insert(DLLIter<Type>& index, const Type& v)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : remove  
// Parameters : index - an iterator pointing at the node to delete  
// Notes : do nothing if index current pointer is null  
/////////////////////////////////////////////////////////////////////////////  
void remove(DLLIter<Type>& index)

The DLLIter class will have the following public interface:

/////////////////////////////////////////////////////////////////////////////  
// Function : Constructor  
// Parameters : listToIterate - the list to iterate  
// Notes: Point list pointer at address of this list object  
/////////////////////////////////////////////////////////////////////////////  
DLLIter(DLList<Type>& listToIterate)  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : beginHead  
// Notes : moves the iterator to the head of the list  
/////////////////////////////////////////////////////////////////////////////  
void beginHead()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : beginTail  
// Notes : moves the iterator to the tail of the list  
/////////////////////////////////////////////////////////////////////////////  
void beginTail()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : end  
// Notes : returns true if current pointer is null, false otherwise  
/////////////////////////////////////////////////////////////////////////////  
bool end() const  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : operator++  
// Notes : move the current pointer to its own next  
/////////////////////////////////////////////////////////////////////////////  
DLLIter<Type>& operator++()  
  
/////////////////////////////////////////////////////////////////////////////  
// Function : operator--  
// Notes : move the current pointer to its own previous  
/////////////////////////////////////////////////////////////////////////////  
DLLIter<Type>& operator--()

/////////////////////////////////////////////////////////////////////////////  
// Function : current  
// Notes : returns the item at the current iterator location. Does not  
// perform error checking  
////////////////////////////////////////////////////////////////////////////  
Type& current() const

# Desired Output

Compile and run your code with the DTSLab4.cpp file provided via FSO. Your console output should match the following block identically:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\* LAB 4: DOUBLY LINKED LIST AND ITERATOR \*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
\*\*\* TEST 1 \*\*\*  
BWD { works this way }  
FWD { works that way }  
  
\*\*\* TEST 2 \*\*\*  
FWD { This is not the end }  
FWD { This is the end }  
BWD { }  
  
\*\*\* TEST 3 \*\*\*  
FWD { are they equal? }  
BWD { }  
  
\*\*\* Test 4 \*\*\*  
FWD { the next two printouts should be empty lists }  
BWD { }  
FWD { }  
BWD { even I'm not sure why this is here }  
BWD { have you covered all your bases? }  
FWD { 24 20 16 12 8 4 0 }  
BWD { }

# Submission

To submit the lab assignment:

1. Clean, build, and run DTSLab1.cpp with your DynArray.h file in Visual Studio (debug mode).
   1. clear up any warnings you encounter.
   2. verify that your output is correct by comparing it to the lab document's Desired Output section, line-by-line.
   3. ensure there are no memory leaks.
2. Submit the 'DLList.h' file via FSO.