Doubly Linked Lists

- Sequence of items, each item connected to the **next and previous** by links
- **Node = data value + link to next item + link to previous item**
- Facilitates traversal in either direction (one direction sufficient for some operations)
- Functionality: pretty much same as singly linked list

Access to Doubly Linked List

- *head* and *tail* reference first and last nodes in list, respectively
- For empty list, *head* and *tail* are NULL
- *head* and *tail* are NOT part of a node (they go in DLINKED_LIST class)

Simple Pointer-Based Implementation

```
#ifndef DNODE_H
#define DNODE_H

class DNODE {
private:
  int data;
  DNODE *next, *previous;

public:
  // Constructor
  DNODE(int initData, DNODE* initNext, DNODE* initPrev);

  // Accessors
  int getData( ) const;
  DNODE* getNext( ) const;
  DNODE* getPrev( ) const;

  // Mutators
  void setData(const int x);
  void setNext(DNODE* newLink);
  void setPrev(DNODE* newLink);
};

#endif  // DNODE_H



class DLINKED_LIST {
private:
  // Member variables
  NODE *headPtr, *tailPtr;

…
};
```

```
void DLINKED_LIST::insertNode(DNODE* ptrToPrevNode,
                              const int newData) {
  if ((headPtr == NULL) || (ptrToPrevNode == NULL))  // insert at front
      insertHeadNode(newData);
  else {
        DNODE *ptrToNewNode = new DNODE(newData, NULL, NULL);
        DNODE *ptrToNextNode = ptrToPrevNode->getNext();
        ptrToNewNode->setNext(ptrToNextNode);
        ptrToNewNode->setPrevious(ptrToPrevNode);
        ptrToPrevNode->setNext(ptrToNewNode);
        if (ptrToNextNode == NULL)
           tailPtr = ptrToNewNode;
        else ptrToNextNode->setPrevious(ptrToNewNode);
  }
}

void DLINKED_LIST::removeNode(DNODE* ptrToNode) {
  if (ptrToNode->getPrevious() == NULL)
    removeHeadNode();  // we must be removing head node
  else if (ptrToNode->getNext() == NULL)
         removeTailNode();  // we must be removing tail node
       else {
             DNODE *ptrToPrevNode = ptrToNode->getPrevious();
             DNODE *ptrToNextNode = ptrToNode->getNext();
             ptrToPrevNode->setNext(ptrToNextNode);
             ptrToNextNode->setPrevious(ptrToPrevNode);
             delete ptrToNode;
            }
}
```

Note:

Preconditions for linked list functions should mention what is assumed/expected of input parameters, plus what is expected of headPtr and tailPtr.

Postconditions for linked list functions should mention what has been changed because of the function, as well as anything that might have changed about the status of headPtr and/or tailPtr.