Doubly Linked List - Lee Beckermeyer

0.1.0

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

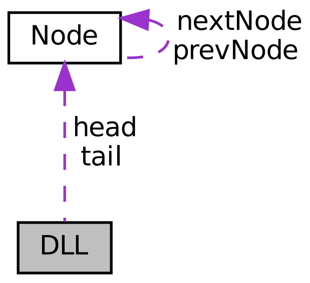
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 DLL Class Reference

Collaboration diagram for DLL:



Public Member Functions

- DLL ()
- bool addToTail (int d)
- int get (int ii)
- bool addMiddle (int ii, int d)
- bool removeHead (int &d)
- bool removeTail ()
- bool addHead (int d)
- void printList ()

Public Attributes

- Node ∗ head
- Node ∗ tail
- int n

3.1.1 Detailed Description

Definition at line 209 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DLL()

```
DLL::DLL ( )  [inline]
```

Constructor

Definition at line 218 of file main.cpp.

```
        {
    head = NULL;
    tail = NULL;
    n = 0;
}
```

3.1.3 Member Function Documentation

3.1.3.1 addHead()

```
bool DLL::addHead (
             int d )  [inline]
```

done Adds a new node to the Head of the list

Parameters

d	pointer to integer to return value

Returns

true if successful

Definition at line 372 of file main.cpp.

```
                        {
      Node* curNode;
      Node* old;
       Node* newNode = new Node(d);
       if(head == NULL) { // the list is empty
           return(false);
       } else {
           old = head;
           head = newNode;
           curNode = head;
           curNode->nextNode = newNode->nextNode;
           curNode->nextNode = old;
```

```
            old->prevNode = head;
            n++;
            return(true);
        }
    };
```

3.1.3.2 addMiddle()

```
bool DLL::addMiddle (
            int ii,
            int d )  [inline]
```

done Adds node after the iith node

Parameters

ii	the node to insert after, note that currently you can't go beyond n-1 as it will reference the head.
d	the data in the new node

Returns

true if successful

Definition at line 281 of file main.cpp.

```
                                   {
        Node* curNode;
        Node* oldNode;
        Node* newNode = new Node(d);
        if(head == NULL) { // the list is empty
            return(false);
        } else if(ii >= n-1) {
            cout « "ERROR: Asked for node beyond tail, check your .addMiddle() format" « endl;
            return(false);
        } else if(ii < 0) {
            cout « "ERROR: Asked for negative index" « endl;
            return(false);
        } else {
            curNode = head;
            // traverse list to desired node
            for(int jj = 0; jj < ii; jj++) {
                curNode = curNode->nextNode;
            }
            oldNode = curNode->nextNode;
            cout « curNode->data;
            // At this point curNode points to the node we want to add after
            newNode->prevNode = curNode;
            newNode->nextNode = curNode->nextNode;
            oldNode->prevNode = newNode;
            curNode->nextNode = newNode;

            n++;
            return(true);
        }
    }
```

3.1.3.3 addToTail()

```
bool DLL::addToTail (
            int d )  [inline]
```

done Adds node to tail of list, done

Definition at line 228 of file main.cpp.

```
                            {
        Node* newNode = new Node(d);
        Node* old;
        Node* curNode;

        if(n == 0) { // the list is empty
            head = newNode;
            tail = newNode;
        } else {
            curNode = head;
            tail->nextNode = newNode;
            head->prevNode = newNode; // update the last node’s next node to newNode
            newNode->prevNode = tail;
            tail = newNode; // update the tail pointer to newNode
        }
        n++;
        return(true);
    }
```

3.1.3.4 get()

```
int DLL::get (
              int ii )  [inline]
```

done Returns the data from the iith node

Parameters

ii	the number of the node to collect data from, note that currently it can't go beyond n-1.

Definition at line 253 of file main.cpp.

```
                      {
        Node* curNode;
        if(head == NULL) { // the list is empty
            return(-999999);
        } else if(ii >= n-1) {
            cout « "ERROR: Asked for node beyond tail" « endl;
            return(-999998);
        } else if(ii < 0) {
            cout « "ERROR: Asked for negative index" « endl;
            return(-999997);
        } else {
            curNode = head;
            // traverse list to desired node
            for(int jj = 0; jj < ii; jj++) {
                curNode = curNode->nextNode;
            }
            return(curNode->data);
        }
    }
```

3.1.3.5 printList()

```
void DLL::printList ( )  [inline]
```

Prints the list to stdout, done

Definition at line 392 of file main.cpp.

```
                       {
```

```
        Node* curNode;
        if(head == NULL) { // the list is empty
            cout « "Empty list" « endl;
        } else { // the list is not empty
            curNode = head; // start at the beginning
            cout « "Forward: \n";
            while(curNode->nextNode != NULL){
                cout « curNode->data « " -> ";
                curNode = curNode->nextNode; // update to next node
            }
            cout « curNode->data;
            cout « endl;
            curNode = tail; // reverse list for DLL(Double Linked List)
            cout « "Backward: \n";
            for(int jj = 0; jj < n-1; jj++) {
                cout « curNode->data « " -> ";
                curNode = curNode->prevNode;
            }
            cout « curNode->data;
            cout « endl;
        }
    }
```

3.1.3.6 removeHead()

```
bool DLL::removeHead (
               int & d )   [inline]
```

done Removes the head node and returns the data value from the removed node

Parameters

d	pointer to integer to return value

Returns

true if successful

Definition at line 319 of file main.cpp.

```
                             {
        int val;
        Node* old; // save off the old node
        if(head != NULL) {
            val = head->data; // collect the data from node to be removed
            old = head; // save off pointer to node we are removing
            head = head->nextNode; // update head to new node
            delete old; // release the memory from the removed node
            n--; // decrement n to show shorter list
            d = val;
            return(true);
        } else { //list is empty
            return(false);
        }
    }
```

3.1.3.7 removeTail()

```
bool DLL::removeTail ( )   [inline]
```

done function to remove the tail of the DLL inspiration taken from here: `https://www.geeksforgeeks.org/remove-last-node-of-the-linked-list/`

Definition at line 342 of file main.cpp.

```
                      {
        int val;
        Node* curNode;
        Node* secondNode = head;
        if (head != NULL){
            //points to the node right before your tail. please note that prevNode makes this a lot
   simpler.
            secondNode = tail->prevNode;

            val = secondNode->nextNode->data;//saves old data

            //deletion and redoing pointers
            delete (secondNode->nextNode);//deletes the old node in memory
            head->prevNode = secondNode;//points head to its new node
            secondNode->nextNode = NULL;//remove the previous node's pointer
            tail = secondNode;//set the tail equal to the previous node
            std::cout « "Here is the number you removed: " « val « "\n";
            n--;//decrements the counter
            return(true);
        }else { //list is empty
            return(false);
        }
    }
```

3.1.4 Member Data Documentation

3.1.4.1 head

`Node* DLL::head`

Definition at line 211 of file main.cpp.

3.1.4.2 n

`int DLL::n`

Definition at line 213 of file main.cpp.

3.1.4.3 tail

`Node* DLL::tail`

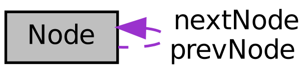
Definition at line 212 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/lee/Leecmake/Project8/src/main.cpp

3.2 Node Class Reference

Collaboration diagram for Node:



Public Member Functions

- Node (int d)

Public Attributes

- int data
- Node ∗ nextNode
- Node ∗ prevNode

3.2.1 Detailed Description

Definition at line 13 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node()

```
Node::Node (
                int d )  [inline]
```

Constructor

Definition at line 22 of file main.cpp.

```
                {
        data = d;
        nextNode = NULL;
        prevNode = NULL;
    }
```

3.2.3 Member Data Documentation

3.2.3.1 data

```
int Node::data
```

Definition at line 15 of file main.cpp.

3.2.3.2 nextNode

```
Node* Node::nextNode
```

Definition at line 16 of file main.cpp.

3.2.3.3 prevNode

```
Node* Node::prevNode
```

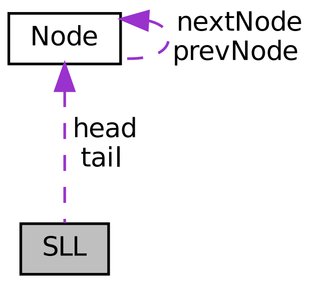
Definition at line 17 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/lee/Leecmake/Project8/src/main.cpp

3.3 SLL Class Reference

Collaboration diagram for SLL:



Public Member Functions

- SLL ()
- bool addToTail (int d)
- int get (int ii)
- bool addMiddle (int ii, int d)
- bool removeHead (int &d)
- bool removeTail ()
- bool addHead (int d)
- void printList ()

Public Attributes

- Node ∗ head
- Node ∗ tail
- int n

3.3.1 Detailed Description

Definition at line 29 of file main.cpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 SLL()

```
SLL::SLL ( )  [inline]
```

Constructor

Definition at line 38 of file main.cpp.

```
            {
        head = NULL;
        tail = NULL;
        n = 0;
    }
```

3.3.3 Member Function Documentation

3.3.3.1 addHead()

```
bool SLL::addHead (
             int d )  [inline]
```

Adds a new node to the Head of the list

Parameters

d	pointer to integer to return value

Returns

true if successful

Definition at line 173 of file main.cpp.
```
                        {
        Node* curNode;
        Node* old;
         Node* newNode = new Node(d);
         if(head == NULL) { // the list is empty
             return(false);
         } else {
             old = head;
             head = newNode;
             curNode = head;
             // At this point curNode points to the node we want to add after
             curNode->nextNode = newNode->nextNode;
             curNode->nextNode = old;
             n++;
             return(true);
         }
    };
```

3.3.3.2 addMiddle()

```
bool SLL::addMiddle (
              int ii,
              int d )   [inline]
```

Adds node after the iith node

Parameters

ii	the node to insert after
d	the data in the new node

Returns

true if successful

Definition at line 92 of file main.cpp.
```
                                 {
        Node* curNode;
        Node* newNode = new Node(d);
        if(head == NULL) { // the list is empty
            return(false);
        } else if(ii >= n) {
            cout « "ERROR: Asked for node beyond tail" « endl;
            return(false);
         } else if(ii < 0) {
             cout « "ERROR: Asked for negative index" « endl;
             return(false);
         } else {
             curNode = head;
             // traverse list to desired node
             for(int jj = 0; jj < ii; jj++) {
                 curNode = curNode->nextNode;
             }
             // At this point curNode points to the node we want to add after
             newNode->nextNode = curNode->nextNode;
             curNode->nextNode = newNode;
             n++;
             return(true);
         }
      }
```

3.3.3.3 addToTail()

```
bool SLL::addToTail (
              int d )  [inline]
```

Adds node to tail of list

Definition at line 47 of file main.cpp.

```
                              {
        Node* newNode = new Node(d);
        if(n == 0) { // the list is empty
            head = newNode;
            tail = newNode;
        } else {
            tail->nextNode = newNode; // update the last node’s next node to newNode
            tail = newNode; // update the tail pointer to newNode
        }
        n++;
        return(true);
    }
```

3.3.3.4 get()

```
int SLL::get (
              int ii )  [inline]
```

Returns the data from the iith node

Parameters

ii	the number of the node to collect data from

Definition at line 65 of file main.cpp.

```
                       {
        Node* curNode;
        if(head == NULL) { // the list is empty
            return(-999999);
        } else if(ii >= n) {
            cout « "ERROR: Asked for node beyond tail" « endl;
            return(-999998);
        } else if(ii < 0) {
            cout « "ERROR: Asked for negative index" « endl;
            return(-999997);
        } else {
            curNode = head;
            // traverse list to desired node
            for(int jj = 0; jj < ii; jj++) {
                curNode = curNode->nextNode;
            }
            return(curNode->data);
        }
    }
```

3.3.3.5 printList()

```
void SLL::printList ( )  [inline]
```

Prints the list to stdout

Definition at line 193 of file main.cpp.

```
                    {
        Node* curNode;
        if(head == NULL) { // the list is empty
            cout « "Empty list" « endl;
        } else { // the list is not empty
            curNode = head; // start at the beginning
            while(curNode->nextNode != NULL){
                cout « curNode->data « " -> ";
                curNode = curNode->nextNode; // update to next node
            }
            cout « curNode->data;
            cout « endl;
        }
    }
```

3.3.3.6 removeHead()

```
bool SLL::removeHead (
              int & d )  [inline]
```

Removes the head node and returns the data value from the removed node

Parameters

d	pointer to integer to return value

Returns

true if successful

Definition at line 123 of file main.cpp.

```
                           {
        int val;
        Node* old; // save off the old node
        if(head != NULL) {
            val = head->data; // collect the data from node to be removed
            old = head; // save off pointer to node we are removing
            head = head->nextNode; // update head to new node
            delete old; // release the memory from the removed node
            n--; // decrement n to show shorter list
            d = val;
            return(true);
        } else { //list is empty
            return(false);
        }
    }
```

3.3.3.7 removeTail()

```
bool SLL::removeTail ( )  [inline]
```

function to remove the tail of the LL

inspiration taken from here: https://www.geeksforgeeks.org/remove-last-node-of-the-linked-list/

Definition at line 145 of file main.cpp.

```
                       {
        int val;
        Node* curNode;
        Node* newNode;
        Node* second_last = head;
        if (head != NULL){
            while (second_last->nextNode->nextNode != NULL){
                second_last = second_last->nextNode;
            };
            // Delete last node
            val = second_last->nextNode->data;
            delete (second_last->nextNode);

            // Change next of second last
            second_last->nextNode = NULL;
            std::cout « "Here is the number you removed: " « val « "\n";
            return(true);
        }else { //list is empty
            return(false);
        }
    }
```

3.3.4 Member Data Documentation

3.3.4.1 head

`Node* SLL::head`

Definition at line 31 of file main.cpp.

3.3.4.2 n

`int SLL::n`

Definition at line 33 of file main.cpp.

3.3.4.3 tail

`Node* SLL::tail`

Definition at line 32 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/lee/Leecmake/Project8/src/main.cpp

Chapter 4

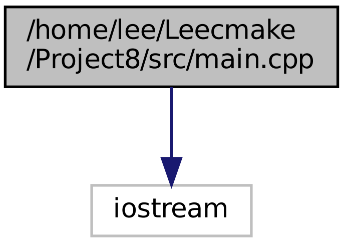
File Documentation

4.1 /home/lee/Leecmake/Project8/src/main.cpp File Reference

This is a class assignment for a DLL.

```
#include <iostream>
```

Include dependency graph for main.cpp:



Classes

- class Node
- class SLL
- class DLL

Functions

- int main (int, char $**$)

4.1.1 Detailed Description

This is a class assignment for a DLL.

Based on stuff covered in class

Author

Lee Beckermeyer

Date

2021 February 14

4.1.2 Function Documentation

4.1.2.1 main()

```
int main (
            int ,
            char ** )
```

SLL myList; int retData; // for data from remove

myList.printList(); myList.addToTail(1); myList.printList(); myList.addToTail(2); myList.printList(); myList.addToTail(3); myList.printList(); myList.addToTail(4); myList.printList(); myList.addToTail(15); myList.printList();

cout << "get(0) = " << myList.get(0) << endl; cout << "get(1) = " << myList.get(1) << endl; cout << "get(4) = " << myList.get(4) << endl; cout << "get(5) = " << myList.get(5) << endl; cout << "get(7) = " << myList.get(7) << endl; cout << "get(-3) = " << myList.get(-3) << endl;

myList.addMiddle(3,10); myList.printList(); myList.addMiddle(3,11); myList.printList(); myList.addMiddle(6,12); myList.printList(); myList.addMiddle(0,13); myList.printList(); myList.addHead(5); myList.printList();

myList.removeTail(); myList.printList();

if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(my↩
List.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList(); if(myList.removeHead(retData)) cout << "Removed " << retData << endl; else cout << "list was empty" << endl; myList.printList();

Definition at line 417 of file main.cpp.

```
                     {
    DLL myList;
    int retData;

    myList.addToTail(1);
    myList.addToTail(2);
    myList.addToTail(3);
    myList.addToTail(4);
    myList.addToTail(5);
    myList.addToTail(6);
    myList.printList();
    myList.addMiddle(4,10);
    myList.printList();
    if(myList.removeHead(retData))
        cout « "Removed " « retData « endl;
    else
        cout « "list was empty." « endl;
    myList.printList();
    if(myList.removeHead(retData))
        cout « "Removed " « retData « endl;
    else
        cout « "list was empty." « endl;
    myList.printList();

    myList.removeTail();
    myList.printList();

    myList.addHead(1);
    myList.printList();

}
```

Index