

# Doubly Linked List - Lee Beckermeyer

0.1.0

Generated by Doxygen 1.8.17



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 DLL Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 DLL()	6
3.1.3 Member Function Documentation	6
3.1.3.1 addHead()	6
3.1.3.2 addMiddle()	7
3.1.3.3 addToTail()	7
3.1.3.4 get()	8
3.1.3.5 printList()	8
3.1.3.6 removeHead()	9
3.1.3.7 removeTail()	9
3.1.4 Member Data Documentation	10
3.1.4.1 head	10
3.1.4.2 n	10
3.1.4.3 tail	10
3.2 Node Class Reference	11
3.2.1 Detailed Description	11
3.2.2 Constructor & Destructor Documentation	11
3.2.2.1 Node()	11
3.2.3 Member Data Documentation	11
3.2.3.1 data	12
3.2.3.2 nextNode	12
3.2.3.3 prevNode	12
3.3 SLL Class Reference	12
3.3.1 Detailed Description	13
3.3.2 Constructor & Destructor Documentation	13
3.3.2.1 SLL()	13
3.3.3 Member Function Documentation	13
3.3.3.1 addHead()	13
3.3.3.2 addMiddle()	14
3.3.3.3 addToTail()	15
3.3.3.4 get()	15
3.3.3.5 printList()	16
3.3.3.6 removeHead()	16
3.3.3.7 removeTail()	16

3.3.4 Member Data Documentation . . . . .	17
3.3.4.1 head . . . . .	17
3.3.4.2 n . . . . .	17
3.3.4.3 tail . . . . .	17
<b>4 File Documentation</b>	<b>19</b>
4.1 /home/lee/Leecmake/Project8/src/main.cpp File Reference . . . . .	19
4.1.1 Detailed Description . . . . .	20
4.1.2 Function Documentation . . . . .	20
4.1.2.1 main() . . . . .	20
<b>Index</b>	<b>23</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">DLL</a>	.....	<a href="#">5</a>
<a href="#">Node</a>	.....	<a href="#">11</a>
<a href="#">SLL</a>	.....	<a href="#">12</a>



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

/home/lee/Leecmake/Project8/src/ <a href="#">main.cpp</a>	
This is a class assignment for a <a href="#">DLL</a> . . . . .	19



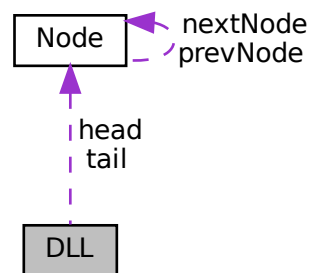


## Chapter 3

# 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.

```
218     {
219         head = NULL;
220         tail = NULL;
221         n = 0;
222     }
```

### 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

<i>d</i>	pointer to integer to return value
----------	------------------------------------

Returns

true if successful

Definition at line 372 of file main.cpp.

```
372     {
373         Node* curNode;
374         Node* old;
375         Node* newNode = new Node(d);
376         if(head == NULL) { // the list is empty
377             return(false);
378         } else {
379             old = head;
380             head = newNode;
381             curNode = head;
382             curNode->nextNode = newNode->nextNode;
383             curNode->nextNode = old;
```

```

384         old->prevNode = head;
385         n++;
386         return(true);
387     }
388 };

```

### 3.1.3.2 addMiddle()

```

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

```

done Adds node after the iith node

#### Parameters

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

#### Returns

true if successful

Definition at line 281 of file main.cpp.

```

281     {
282         Node* curNode;
283         Node* oldNode;
284         Node* newNode = new Node(d);
285         if(head == NULL) { // the list is empty
286             return(false);
287         } else if(ii >= n-1) {
288             cout << "ERROR: Asked for node beyond tail, check your .addMiddle() format" << endl;
289             return(false);
290         } else if(ii < 0) {
291             cout << "ERROR: Asked for negative index" << endl;
292             return(false);
293         } else {
294             curNode = head;
295             // traverse list to desired node
296             for(int jj = 0; jj < ii; jj++) {
297                 curNode = curNode->nextNode;
298             }
299             oldNode = curNode->nextNode;
300             cout << curNode->data;
301             // At this point curNode points to the node we want to add after
302             newNode->prevNode = curNode;
303             newNode->nextNode = curNode->nextNode;
304             oldNode->prevNode = newNode;
305             curNode->nextNode = newNode;
306
307             n++;
308             return(true);
309         }
310     }

```

### 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.

```

228         {
229             Node* newNode = new Node(d);
230             Node* old;
231             Node* curNode;
232
233             if(n == 0) { // the list is empty
234                 head = newNode;
235                 tail = newNode;
236             } else {
237                 curNode = head;
238                 tail->nextNode = newNode;
239                 head->prevNode = newNode; // update the last node's next node to newNode
240                 newNode->prevNode = tail;
241                 tail = newNode; // update the tail pointer to newNode
242             }
243             n++;
244             return(true);
245         }

```

### 3.1.3.4 get()

```

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

```

done Returns the data from the iith node

#### Parameters

<i>ii</i>	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.

```

253         {
254             Node* curNode;
255             if(head == NULL) { // the list is empty
256                 return(-999999);
257             } else if(ii >= n-1) {
258                 cout << "ERROR: Asked for node beyond tail" << endl;
259                 return(-999998);
260             } else if(ii < 0) {
261                 cout << "ERROR: Asked for negative index" << endl;
262                 return(-999997);
263             } else {
264                 curNode = head;
265                 // traverse list to desired node
266                 for(int jj = 0; jj < ii; jj++) {
267                     curNode = curNode->nextNode;
268                 }
269                 return(curNode->data);
270             }
271         }

```

### 3.1.3.5 printList()

```

void DLL::printList ( ) [inline]

```

Prints the list to stdout, done

Definition at line 392 of file main.cpp.

```

392         {

```

```

393     Node* curNode;
394     if(head == NULL) { // the list is empty
395         cout << "Empty list" << endl;
396     } else { // the list is not empty
397         curNode = head; // start at the beginning
398         cout << "Forward: \n";
399         while(curNode->nextNode != NULL) {
400             cout << curNode->data << " -> ";
401             curNode = curNode->nextNode; // update to next node
402         }
403         cout << curNode->data;
404         cout << endl;
405         curNode = tail; // reverse list for DLL(Double Linked List)
406         cout << "Backward: \n";
407         for(int jj = 0; jj < n-1; jj++) {
408             cout << curNode->data << " -> ";
409             curNode = curNode->prevNode;
410         }
411         cout << curNode->data;
412         cout << endl;
413     }
414 }

```

### 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

<i>d</i>	pointer to integer to return value
----------	------------------------------------

#### Returns

true if successful

Definition at line 319 of file main.cpp.

```

319     {
320         int val;
321         Node* old; // save off the old node
322         if(head != NULL) {
323             val = head->data; // collect the data from node to be removed
324             old = head; // save off pointer to node we are removing
325             head = head->nextNode; // update head to new node
326             delete old; // release the memory from the removed node
327             n--; // decrement n to show shorter list
328             d = val;
329             return(true);
330         } else { //list is empty
331             return(false);
332         }
333     }

```

### 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.

```

342         {
343             int val;
344             Node* curNode;
345             Node* secondNode = head;
346             if (head != NULL) {
347                 //points to the node right before your tail. please note that prevNode makes this a lot
348                 simpler.
349                 secondNode = tail->prevNode;
350                 val = secondNode->nextNode->data; //saves old data
351
352                 //deletion and redoing pointers
353                 delete (secondNode->nextNode); //deletes the old node in memory
354                 head->prevNode = secondNode; //points head to its new node
355                 secondNode->nextNode = NULL; //remove the previous node's pointer
356                 tail = secondNode; //set the tail equal to the previous node
357                 std::cout << "Here is the number you removed: " << val << "\n";
358                 n--; //decrements the counter
359                 return(true);
360             } else { //list is empty
361                 return(false);
362             }
363         }

```

### 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.

```
22         {
23             data = d;
24             nextNode = NULL;
25             prevNode = NULL;
26         }
```

### 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
```

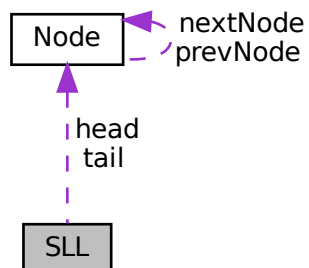
Definition at line 17 of file main.cpp.

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.

```
38     {  
39         head = NULL;  
40         tail = NULL;  
41         n = 0;  
42     }
```

### 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**

<i>d</i>	pointer to integer to return value
----------	------------------------------------

**Returns**

true if successful

Definition at line 173 of file main.cpp.

```

173         {
174             Node* curNode;
175             Node* old;
176             Node* newNode = new Node(d);
177             if(head == NULL) { // the list is empty
178                 return(false);
179             } else {
180                 old = head;
181                 head = newNode;
182                 curNode = head;
183                 // At this point curNode points to the node we want to add after
184                 curNode->nextNode = newNode->nextNode;
185                 curNode->nextNode = old;
186                 n++;
187                 return(true);
188             }
189     };

```

**3.3.3.2 addMiddle()**

```

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

```

Adds node after the iith node

**Parameters**

<i>ii</i>	the node to insert after
<i>d</i>	the data in the new node

**Returns**

true if successful

Definition at line 92 of file main.cpp.

```

92         {
93             Node* curNode;
94             Node* newNode = new Node(d);
95             if(head == NULL) { // the list is empty
96                 return(false);
97             } else if(ii >= n) {
98                 cout << "ERROR: Asked for node beyond tail" << endl;
99                 return(false);
100             } else if(ii < 0) {
101                 cout << "ERROR: Asked for negative index" << endl;
102                 return(false);
103             } else {
104                 curNode = head;
105                 // traverse list to desired node
106                 for(int jj = 0; jj < ii; jj++) {
107                     curNode = curNode->nextNode;

```

```

108         }
109         // At this point curNode points to the node we want to add after
110         newNode->nextNode = curNode->nextNode;
111         curNode->nextNode = newNode;
112         n++;
113         return(true);
114     }
115 }

```

### 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.

```

47     {
48         Node* newNode = new Node(d);
49         if(n == 0) { // the list is empty
50             head = newNode;
51             tail = newNode;
52         } else {
53             tail->nextNode = newNode; // update the last node's next node to newNode
54             tail = newNode; // update the tail pointer to newNode
55         }
56         n++;
57         return(true);
58     }

```

### 3.3.3.4 get()

```

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

```

Returns the data from the iith node

#### Parameters

<i>ii</i>	the number of the node to collect data from
-----------	---

Definition at line 65 of file main.cpp.

```

65     {
66         Node* curNode;
67         if(head == NULL) { // the list is empty
68             return(-999999);
69         } else if(ii >= n) {
70             cout << "ERROR: Asked for node beyond tail" << endl;
71             return(-999998);
72         } else if(ii < 0) {
73             cout << "ERROR: Asked for negative index" << endl;
74             return(-999997);
75         } else {
76             curNode = head;
77             // traverse list to desired node
78             for(int jj = 0; jj < ii; jj++) {
79                 curNode = curNode->nextNode;
80             }
81             return(curNode->data);
82         }
83     }

```

### 3.3.3.5 printList()

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

Prints the list to stdout

Definition at line 193 of file main.cpp.

```
193     {
194         Node* curNode;
195         if(head == NULL) { // the list is empty
196             cout << "Empty list" << endl;
197         } else { // the list is not empty
198             curNode = head; // start at the beginning
199             while(curNode->nextNode != NULL){
200                 cout << curNode->data << " -> ";
201                 curNode = curNode->nextNode; // update to next node
202             }
203             cout << curNode->data;
204             cout << endl;
205         }
206     }
```

### 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

<i>d</i>	pointer to integer to return value
----------	------------------------------------

#### Returns

true if successful

Definition at line 123 of file main.cpp.

```
123     {
124         int val;
125         Node* old; // save off the old node
126         if(head != NULL) {
127             val = head->data; // collect the data from node to be removed
128             old = head; // save off pointer to node we are removing
129             head = head->nextNode; // update head to new node
130             delete old; // release the memory from the removed node
131             n--; // decrement n to show shorter list
132             d = val;
133             return(true);
134         } else { //list is empty
135             return(false);
136         }
137     }
```

### 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.

```

145         {
146             int val;
147             Node* curNode;
148             Node* newNode;
149             Node* second_last = head;
150             if (head != NULL) {
151                 while (second_last->nextNode->nextNode != NULL) {
152                     second_last = second_last->nextNode;
153                 };
154                 // Delete last node
155                 val = second_last->nextNode->data;
156                 delete (second_last->nextNode);
157
158                 // Change next of second last
159                 second_last->nextNode = NULL;
160                 std::cout << "Here is the number you removed: " << val << "\n";
161                 return(true);
162             } else { //list is empty
163                 return(false);
164             }
165         }

```

### 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

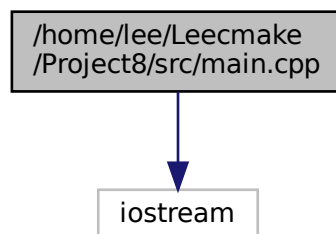
# File Documentation

### 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 Beckermeier

#### 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(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();
```

Definition at line 417 of file main.cpp.

```
417         {
418     DLL myList;
419     int retData;
420
421     myList.addToTail(1);
422     myList.addToTail(2);
```



```
423     myList.addToTail(3);
424     myList.addToTail(4);
425     myList.addToTail(5);
426     myList.addToTail(6);
427     myList.printList();
428     myList.addMiddle(4,10);
429     myList.printList();
430     if(myList.removeHead(retData))
431         cout << "Removed " << retData << endl;
432     else
433         cout << "list was empty." << endl;
434     myList.printList();
435     if(myList.removeHead(retData))
436         cout << "Removed " << retData << endl;
437     else
438         cout << "list was empty." << endl;
439     myList.printList();
440
441     myList.removeTail();
442     myList.printList();
443
444     myList.addHead(1);
445     myList.printList();
446
447 }
528 }
```



# Index

/home/lee/Leecmake/Project8/src/main.cpp, [19](#)

addHead

DLL, [6](#)  
SLL, [13](#)

addMiddle

DLL, [7](#)  
SLL, [14](#)

addToTail

DLL, [7](#)  
SLL, [15](#)

data

Node, [11](#)

DLL, [5](#)

addHead, [6](#)  
addMiddle, [7](#)  
addToTail, [7](#)  
DLL, [6](#)  
get, [8](#)  
head, [10](#)  
n, [10](#)  
printList, [8](#)  
removeHead, [9](#)  
removeTail, [9](#)  
tail, [10](#)

get

DLL, [8](#)  
SLL, [15](#)

head

DLL, [10](#)  
SLL, [17](#)

main

main.cpp, [20](#)

main.cpp

main, [20](#)

n

DLL, [10](#)  
SLL, [17](#)

nextNode

Node, [12](#)

Node, [11](#)

data, [11](#)  
nextNode, [12](#)  
Node, [11](#)  
prevNode, [12](#)

prevNode

Node, [12](#)

printList

DLL, [8](#)  
SLL, [15](#)

removeHead

DLL, [9](#)  
SLL, [16](#)

removeTail

DLL, [9](#)  
SLL, [16](#)

SLL, [12](#)

addHead, [13](#)  
addMiddle, [14](#)  
addToTail, [15](#)  
get, [15](#)  
head, [17](#)  
n, [17](#)  
printList, [15](#)  
removeHead, [16](#)  
removeTail, [16](#)  
SLL, [13](#)  
tail, [17](#)

tail

DLL, [10](#)  
SLL, [17](#)