

Class Demo Singly Linked List

0.1.0

Generated by Doxygen 1.8.17

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Node Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 Node()	6
3.1.3 Member Data Documentation	6
3.1.3.1 data	6
3.1.3.2 nextNode	6
3.2 SLL Class Reference	6
3.2.1 Detailed Description	7
3.2.2 Constructor & Destructor Documentation	7
3.2.2.1 SLL()	7
3.2.3 Member Function Documentation	7
3.2.3.1 addMiddle()	7
3.2.3.2 addToTail()	8
3.2.3.3 get()	8
3.2.3.4 printList()	9
3.2.3.5 removeHead()	9
3.2.4 Member Data Documentation	10
3.2.4.1 head	10
3.2.4.2 n	10
3.2.4.3 tail	10
4 File Documentation	11
4.1 /home/drseth/CPTR227/20210208-SLLClassDemo/src/main.cpp File Reference	11
4.1.1 Detailed Description	12
4.1.2 Function Documentation	12
4.1.2.1 main()	12
Index	15

Chapter 1

Class Index

1.1 Class List

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

Node	5
SLL	6

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

<code>/home/drseth/CPTR227/20210208-SLLClassDemo/src/main.cpp</code>	
This is a demo of making a singly linked list	11

Chapter 3

Class Documentation

3.1 Node Class Reference

Collaboration diagram for Node:



Public Member Functions

- [Node](#) (int d)

Public Attributes

- int [data](#)
- [Node](#) * [nextNode](#)

3.1.1 Detailed Description

Definition at line 13 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Node()

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

Constructor

Definition at line 21 of file main.cpp.

```
21     {
22         data = d;
23         nextNode = NULL;
24     }
```

3.1.3 Member Data Documentation

3.1.3.1 data

```
int Node::data
```

Definition at line 15 of file main.cpp.

3.1.3.2 nextNode

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

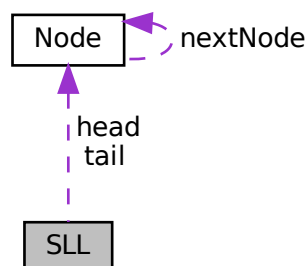
Definition at line 16 of file main.cpp.

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

- [/home/drseth/CPTR227/20210208-SLLClassDemo/src/main.cpp](#)

3.2 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)
- void [printList](#) ()

Public Attributes

- [Node](#) * [head](#)
- [Node](#) * [tail](#)
- int [n](#)

3.2.1 Detailed Description

Definition at line 27 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 SLL()

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

Constructor

Definition at line 36 of file main.cpp.

```
36     {
37         head = NULL;
38         tail = NULL;
39         n = 0;
40     }
```

3.2.3 Member Function Documentation

3.2.3.1 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 90 of file main.cpp.

```

90         {
91             Node* curNode;
92             Node* newNode = new Node(d);
93             if(head == NULL) { // the list is empty
94                 return(false);
95             } else if(ii >= n) {
96                 cout << "ERROR: Asked for node beyond tail" << endl;
97                 return(false);
98             } else if(ii < 0) {
99                 cout << "ERROR: Asked for negative index" << endl;
100                return(false);
101            } else {
102                curNode = head;
103                // traverse list to desired node
104                for(int jj = 0; jj < ii; jj++) {
105                    curNode = curNode->nextNode;
106                }
107                // At this point curNode points to the node we want to add after
108                newNode->nextNode = curNode->nextNode;
109                curNode->nextNode = newNode;
110                n++;
111                return(true);
112            }
113        }

```

3.2.3.2 addToTail()

```

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

```

Adds node to tail of list

Definition at line 45 of file main.cpp.

```

45         {
46             Node* newNode = new Node(d);
47             if(n == 0) { // the list is empty
48                 head = newNode;
49                 tail = newNode;
50             } else {
51                 tail->nextNode = newNode; // update the last node's next node to newNode
52                 tail = newNode; // update the tail pointer to newNode
53             }
54             n++;
55             return(true);
56         }

```

3.2.3.3 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 63 of file main.cpp.

```

63     {
64         Node* curNode;
65         if(head == NULL) { // the list is empty
66             return(-999999);
67         } else if(ii >= n) {
68             cout << "ERROR: Asked for node beyond tail" << endl;
69             return(-999998);
70         } else if(ii < 0) {
71             cout << "ERROR: Asked for negative index" << endl;
72             return(-999997);
73         } else {
74             curNode = head;
75             // traverse list to desired node
76             for(int jj = 0; jj < ii; jj++) {
77                 curNode = curNode->nextNode;
78             }
79             return(curNode->data);
80         }
81     }

```

3.2.3.4 printList()

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

Prints the list to stdout

Definition at line 140 of file main.cpp.

```

140     {
141         Node* curNode;
142         if(head == NULL) { // the list is empty
143             cout << "Empty list" << endl;
144         } else { // the list is not empty
145             curNode = head; // start at the beginning
146             while(curNode->nextNode != NULL){
147                 cout << curNode->data << " -> ";
148                 curNode = curNode->nextNode; // update to next node
149             }
150             cout << curNode->data;
151             cout << endl;
152         }
153     }

```

3.2.3.5 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 121 of file main.cpp.

```
121         {
122             int val;
123             Node* old; // save off the old node
124             if(head != NULL) {
125                 val = head->data; // collect the data from node to be removed
126                 old = head; // save off pointer to node we are removing
127                 head = head->nextNode; // update head to new node
128                 delete old; // release the memory from the removed node
129                 n--; // decrement n to show shorter list
130                 d = val;
131                 return(true);
132             } else { //list is empty
133                 return(false);
134             }
135         }
```

3.2.4 Member Data Documentation

3.2.4.1 head

`Node* SLL::head`

Definition at line 29 of file main.cpp.

3.2.4.2 n

`int SLL::n`

Definition at line 31 of file main.cpp.

3.2.4.3 tail

`Node* SLL::tail`

Definition at line 30 of file main.cpp.

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

- </home/drseth/CPTR227/20210208-SLLClassDemo/src/main.cpp>

Chapter 4

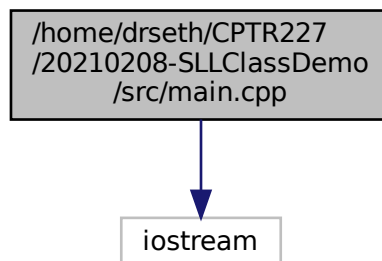
File Documentation

4.1 /home/drseth/CPTR227/20210208-SLLClassDemo/src/main.cpp File Reference

This is a demo of making a singly linked list.

```
#include <iostream>
```

Include dependency graph for main.cpp:



Classes

- class [Node](#)
- class [SLL](#)

Functions

- int [main](#) (int, char **)

4.1.1 Detailed Description

This is a demo of making a singly linked list.

Based on ODS book examples

Author

Seth McNeill

Date

2021 February 08

4.1.2 Function Documentation

4.1.2.1 main()

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

Definition at line 156 of file main.cpp.

```
156         {
157             SLL myList;
158             int retData; // for data from remove
159
160             myList.printList();
161             myList.addToTail(1);
162             myList.printList();
163             myList.addToTail(2);
164             myList.printList();
165             myList.addToTail(3);
166             myList.printList();
167             myList.addToTail(4);
168             myList.printList();
169             myList.addToTail(5);
170             myList.printList();
171
172             cout << "get(0) = " << myList.get(0) << endl;
173             cout << "get(1) = " << myList.get(1) << endl;
174             cout << "get(4) = " << myList.get(4) << endl;
175             cout << "get(5) = " << myList.get(5) << endl;
176             cout << "get(7) = " << myList.get(7) << endl;
177             cout << "get(-3) = " << myList.get(-3) << endl;
178
179             myList.addMiddle(3,10);
180             myList.printList();
181             myList.addMiddle(3,11);
182             myList.printList();
183             myList.addMiddle(6,12);
184             myList.printList();
185             myList.addMiddle(0,13);
186             myList.printList();
187
188
189             if(myList.removeHead(retData))
190                 cout << "Removed " << retData << endl;
191             else
192                 cout << "list was empty" << endl;
193             myList.printList();
194             if(myList.removeHead(retData))
195                 cout << "Removed " << retData << endl;
196             else
197                 cout << "list was empty" << endl;
198             myList.printList();
```



```
199     if(myList.removeHead(retData))
200         cout << "Removed " << retData << endl;
201     else
202         cout << "list was empty" << endl;
203     myList.printList();
204     if(myList.removeHead(retData))
205         cout << "Removed " << retData << endl;
206     else
207         cout << "list was empty" << endl;
208     myList.printList();
209     if(myList.removeHead(retData))
210         cout << "Removed " << retData << endl;
211     else
212         cout << "list was empty" << endl;
213     myList.printList();
214     if(myList.removeHead(retData))
215         cout << "Removed " << retData << endl;
216     else
217         cout << "list was empty" << endl;
218     myList.printList();
219     if(myList.removeHead(retData))
220         cout << "Removed " << retData << endl;
221     else
222         cout << "list was empty" << endl;
223     myList.printList();
224     if(myList.removeHead(retData))
225         cout << "Removed " << retData << endl;
226     else
227         cout << "list was empty" << endl;
228     myList.printList();
229 }
```


Index

/home/drseth/CPTR227/20210208-SLLClassDemo/src/main.cpp,
11

addMiddle
SLL, 7

addToTail
SLL, 8

data
Node, 6

get
SLL, 8

head
SLL, 10

main
main.cpp, 12

main.cpp
main, 12

n
SLL, 10

nextNode
Node, 6

Node, 5
data, 6
nextNode, 6
Node, 5

printList
SLL, 9

removeHead
SLL, 9

SLL, 6
addMiddle, 7
addToTail, 8
get, 8
head, 10
n, 10
printList, 9
removeHead, 9
SLL, 7
tail, 10

tail
SLL, 10