linkedlist

0.0.1

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

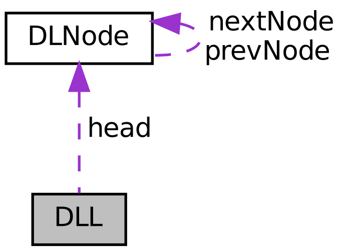
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 DLL Class Reference

Collaboration diagram for DLL:



Public Member Functions

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

Public Attributes

- DLNode ∗ head
- int n

3.1.1 Detailed Description

Definition at line 28 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DLL()

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

Definition at line 33 of file main.cpp.

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

3.1.3 Member Function Documentation

3.1.3.1 addHead()

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

Definition at line 38 of file main.cpp.

```
                        {
        DLNode* newNode = new DLNode(d);
        if (n == 0) { // the list is empty
            head = newNode;
        }
        else {
            newNode->nextNode = head;
            head->prevNode = newNode;
            head = newNode;


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

3.1.3.2 addMiddle()

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

Definition at line 113 of file main.cpp.

```
                                {
        DLNode* curNode;
        DLNode* newNode = new DLNode(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;
            }
            newNode->nextNode = curNode->nextNode;
            newNode->prevNode = curNode;
            curNode->nextNode = newNode;
            if(newNode->nextNode != NULL){
                newNode->nextNode->prevNode = newNode;
            }
            n++;
            return(true);
        }
    }
```

3.1.3.3 addToTail()

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

Definition at line 54 of file main.cpp.

```
                                {
        DLNode* newNode = new DLNode(d);
        if (n == 0) { // the list is empty
            head = newNode;
        }
        else {
            DLNode* value = head;
            while (value->nextNode != NULL) {
                value = value->nextNode;
            }
            newNode->prevNode = value;
            newNode->nextNode = NULL;
            value->nextNode = newNode;
        }
        n++;
        return(true);
    }
```

3.1.3.4 get()

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

Definition at line 90 of file main.cpp.

```
                      {
          DLNode* 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);
           }
       }
```

3.1.3.5 printList()

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

Definition at line 162 of file main.cpp.

```
                        {
           DLNode* 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;
           }
       }
```

3.1.3.6 removeHead()

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

Definition at line 144 of file main.cpp.

```
                               {
           int val;
           DLNode* 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
               head->prevNode = NULL;
               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 (
             int d )  [inline]
```

Definition at line 72 of file main.cpp.

```
                        {
        DLNode* var = head->nextNode;

        if (head->nextNode == NULL) {
            return(false);
        }
        while (var->nextNode->nextNode != NULL) {
            var = var->nextNode;
        }
        d = var->nextNode->data;
        delete var->nextNode;
        var->nextNode = NULL;

        n--;
        return(true);

    }
```

3.1.4 Member Data Documentation

3.1.4.1 head

```
DLNode* DLL::head
```

Definition at line 30 of file main.cpp.

3.1.4.2 n

```
int DLL::n
```

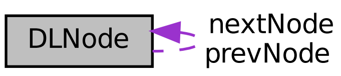
Definition at line 31 of file main.cpp.

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

- /home/addis/linkedlist/linked-list/src/main.cpp

3.2 DLNode Class Reference

Collaboration diagram for DLNode:



Public Member Functions

- DLNode (int d)

Public Attributes

- int data
- DLNode ∗ nextNode
- DLNode ∗ prevNode

3.2.1 Detailed Description

Definition at line 15 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DLNode()

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

Definition at line 21 of file main.cpp.

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

3.2.3 Member Data Documentation

3.2.3.1 data

```
int DLNode::data
```

Definition at line 17 of file main.cpp.

3.2.3.2 nextNode

```
DLNode* DLNode::nextNode
```

Definition at line 18 of file main.cpp.

3.2.3.3 prevNode

```
DLNode* DLNode::prevNode
```

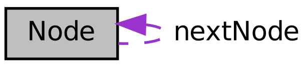
Definition at line 19 of file main.cpp.

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

- /home/addis/linkedlist/linked-list/src/main.cpp

3.3 Node Class Reference

Collaboration diagram for Node:



Public Member Functions

- Node (int d)

Public Attributes

- int data
- Node * nextNode

3.3.1 Detailed Description

Definition at line 4 of file SLL.cpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Node()

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

Definition at line 9 of file SLL.cpp.

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

3.3.3 Member Data Documentation

3.3.3.1 data

```
int Node::data
```

Definition at line 6 of file SLL.cpp.

3.3.3.2 nextNode

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

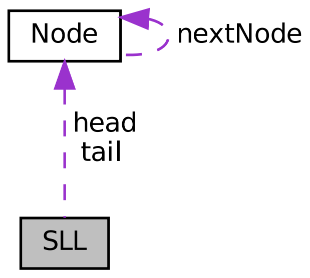
Definition at line 7 of file SLL.cpp.

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

- /home/addis/linkedlist/linked-list/src/SLL.cpp

3.4 SLL Class Reference

Collaboration diagram for SLL:



Public Member Functions

- SLL ()
- bool addHead (int d)
- bool addToTail (int d)
- bool removeTail (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

3.4.1 Detailed Description

Definition at line 15 of file SLL.cpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 SLL()

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

Definition at line 21 of file SLL.cpp.

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

3.4.3 Member Function Documentation

3.4.3.1 addHead()

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

Definition at line 27 of file SLL.cpp.

```
                          {
        Node* newNode = new Node(d);
        if (n == 0) { // the list is empty
            head = newNode;
        }
        else {
            newNode->nextNode = head;
            head = newNode;
        }
        n++;
        return(true);
    }
```

3.4.3.2 addMiddle()

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

Definition at line 91 of file SLL.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);
         }
     }
```

3.4.3.3 addToTail()

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

Definition at line 40 of file SLL.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);
    }
```

3.4.3.4 get()

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

Definition at line 71 of file SLL.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);
        }
    }
```

3.4.3.5 printList()

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

Definition at line 132 of file SLL.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;
        }
    }
```

3.4.3.6 removeHead()

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

Definition at line 116 of file SLL.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);
        }
    }
```

3.4.3.7 removeTail()

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

Definition at line 53 of file SLL.cpp.

```
                              {
          Node* var = head->nextNode;

          if (head->nextNode == NULL) {
              return(false);
          }
          while (var->nextNode->nextNode != NULL) {
              var = var->nextNode;
          }
          d = var->nextNode->data;
          delete var->nextNode;
          var->nextNode = NULL;

          n--;
          return(true);

      }
```

3.4.4 Member Data Documentation

3.4.4.1 head

```
Node* SLL::head
```

Definition at line 17 of file SLL.cpp.

3.4.4.2 n

```
int SLL::n
```

Definition at line 19 of file SLL.cpp.

3.4.4.3 tail

```
Node* SLL::tail
```

Definition at line 18 of file SLL.cpp.

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

- /home/addis/linkedlist/linked-list/src/SLL.cpp

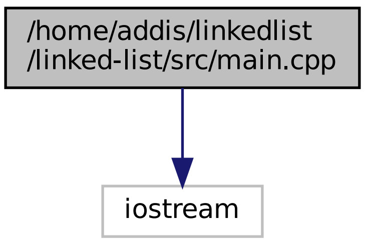
Chapter 4

File Documentation

4.1 /home/addis/linkedlist/linked-list/src/main.cpp File Reference

This is a test of CMake, doxygen, and GitHub.

```
#include <iostream>
```
Include dependency graph for main.cpp:



Classes

- class DLNode
- class DLL

Functions

- int main (int, char ∗∗)

4.1.1 Detailed Description

This is a test of CMake, doxygen, and GitHub.

This is the long brief at the top of main.cpp.

Author

Seth McNeill

Date

1/28/2021

4.1.2 Function Documentation

4.1.2.1 main()

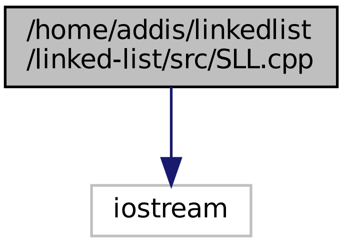
```
int main (
            int ,
            char ** )
```

Definition at line 179 of file main.cpp.

```
                        {
    DLL 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(5);
    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.printList();
    if (myList.addHead(retData))
        cout « "addedtohead " « retData « endl;
    else
        cout « "list was empty" « endl;

    if (myList.addToTail(retData))
        cout « "RemovedTail " « 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.removeTail(retData))
        cout « "Removed Tail " « retData « 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();
}
```

4.2 /home/addis/linkedlist/linked-list/src/SLL.cpp File Reference

`#include <iostream>`
Include dependency graph for SLL.cpp:



Classes

- class Node
- class SLL

Functions

- int main (int, char ∗∗)

4.2.1 Function Documentation

4.2.1.1 main()

```
int main (
              int ,
              char ∗∗  )
```

Definition at line 148 of file SLL.cpp.

```
                    {
    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(5);
    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();


    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();
}
```

Index