Lab 2 - Linked List

Generated by Doxygen 1.8.8

Sat Sep 23 2017 00:32:20

Contents

1 Specification

This is the Linked List City Program. The user will see upon starting the program will see a list of cities being displayed. Then two cities will be inserted into the list and then displayed. The list will be destroyed (deleted) and then rebuilt Two new cities will then be inserted into the list and be displayed.

Freatures:

1) As many cities can be entered into the list as the user wants

2) The ability to add words into the dictionary/program for later use.

3) Delete/destroy list when done so one can recreate within the same program

2 Analysis

When the program runs the first/initial list will be displayed. From there two cities will be added in a certain order and then displayed. The list will then be destroyed and then rebuilt into its original state. Then the program will add two different cities and then rebuild the list. The list destroys itself when the program ends.

3 Design

There are 7 parts to this program. The insert and insert in order parts do exactly as they sound. Both of those fuctions insert into the list. One inserts the city is a specifc place relative to the spelling. The display and destroy functions display the list and delete the list. The build list directly function builds the list so that the list can be resued once deleted. a header file will be created to simplify function prototypes.

4 Test

This test shows that the program functions

```
Milpitas
Dublin
Livermore
Fremont
Hayward

Insert in order Sacramento & Newark
Milpitas
Dublin
Livermore
Fremont
Hayward
Newark
Sacramento

Destroying and rebuilding list
deleting: Milpitas
deleting: Dublin
deleting: Livermore
deleting: Fremont
deleting: Hayward
deleting: Newark
deleting: Sacramento

Insert Sacramento & Newark
Newark
Davis
Hayward
Fremont
Livermore
Dublin
Milpitas
```

5 Class Index

5.1 Class List

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

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Class Documentation

7.1 NODE Struct Reference

City Structure.

`#include <lab.h>`

Collaboration diagram for NODE:

Public Attributes

- std::string city
- NODE * next

7.1.1 Detailed Description

City Structure.

This structure is is used to create each node of the linked list of cities.

7.1.2 Member Data Documentation

7.1.2.1 std::string NODE::city

7.1.2.2 NODE* NODE::next

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

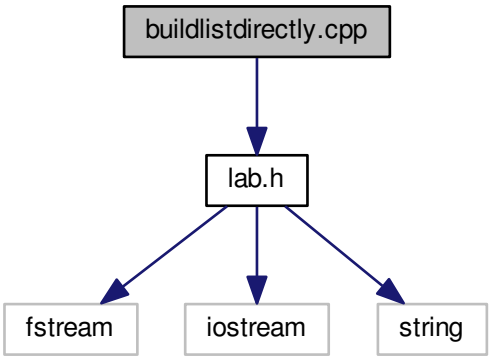
- lab.h

8 File Documentation

8.1 buildlistdirectly.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for buildlistdirectly.cpp:



Functions

- STATUS builddirectly (NODE *&head, string cities)

8.1.1 Function Documentation

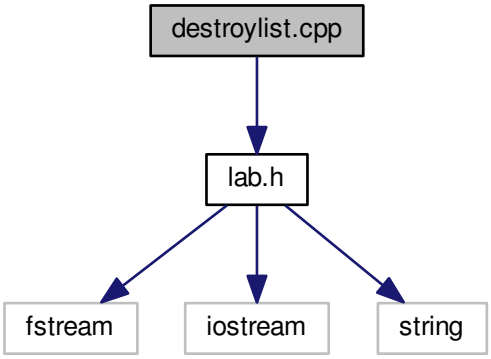
8.1.1.1 STATUS builddirectly (NODE *& *head,* string *cities*)

```
5 {
6     string city;
7     NODE* tail;
8     ifstream ifs("cities");
9     if (! ifs)
10     return FAILED;
11
```

```
12 while(ifs >> city) {
13     NODE *newnode = new NODE;
14     if(!newnode)
15     return FAILED;
16
17     newnode->city = city;
18
19     newnode->next = 0;
20
21      if(!tail) {
22         head = newnode;}
23     else {
24         tail->next = newnode;
25     }
26     tail = newnode;
27
28 }
29 }
```

8.2 destroylist.cpp File Reference

`#include "lab.h"`
Include dependency graph for destroylist.cpp:



Functions

- void destroylist (NODE *head)

8.2.1 Function Documentation

8.2.1.1 void destroylist (NODE * *head*)

Parameters

in	*head*	This is the only parameter the function takes.

This function completely destroys the list.

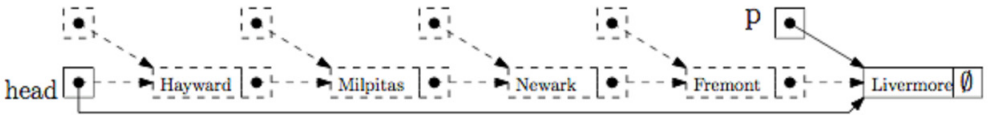


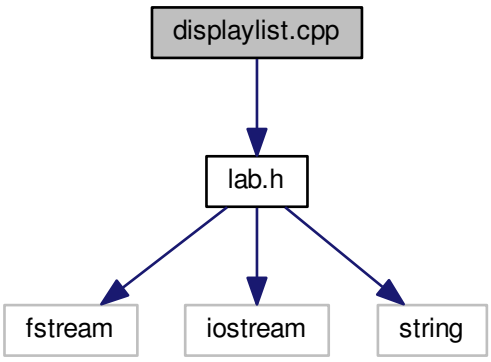
Figure 1: Linked List

As you can see that this function points the head at the node and then deletes it severing the tie to the list

```
14 {
15     NODE* node;
16     for(node = head; node; node = node->next)
17     {
18         cout << "deleting: "
19             << node->city << std::endl;
20         NODE* tmp = head->next;
21         delete head;
22         head = tmp;
23
24     }
25
26 }
```

8.3 displaylist.cpp File Reference

`#include "lab.h"`
Include dependency graph for displaylist.cpp:



Functions

- void displaylist (NODE *head)

8.3.1 Function Documentation

8.3.1.1 void displaylist (NODE * *head*)

Parameters

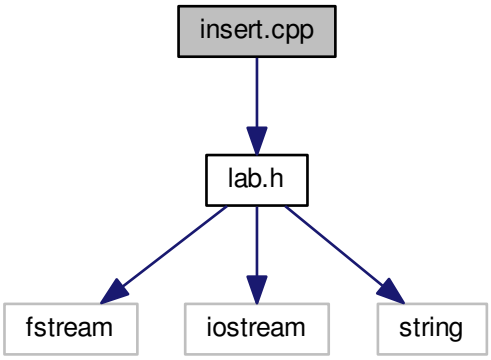
in	*head*	This fucntion only takes a pointer as an argument.

This function displays/traverses the list This is what a linked list looks like to a user It traverses the list without manipulating any pointers. See the comments for further deatils

```
11 {
12     //first we set the node pointer equal to the head pointer
13     //Then while node exists this loop continues
14     //Then we set the pointer equal to the next pointer
15
16     for(NODE* node = head; node; node = node->next)
17     //This displays the data within the node
18     cout << node->city << endl;
19 }
```

8.4 insert.cpp File Reference

`#include "lab.h"`
Include dependency graph for insert.cpp:



Functions

- STATUS insert (NODE ∗&head, std::string city)

  Insert inserts a new node at the beginning of the list.

8.4.1 Function Documentation

8.4.1.1 STATUS insert (NODE ∗& *head*, std::string *city*)

Insert inserts a new node at the beginning of the list.

Parameters

in,out	*head*	The head of the linked list
in	*city*	The data in the node being inserted

Returns

A STATUS indicating if Insert was successful of not

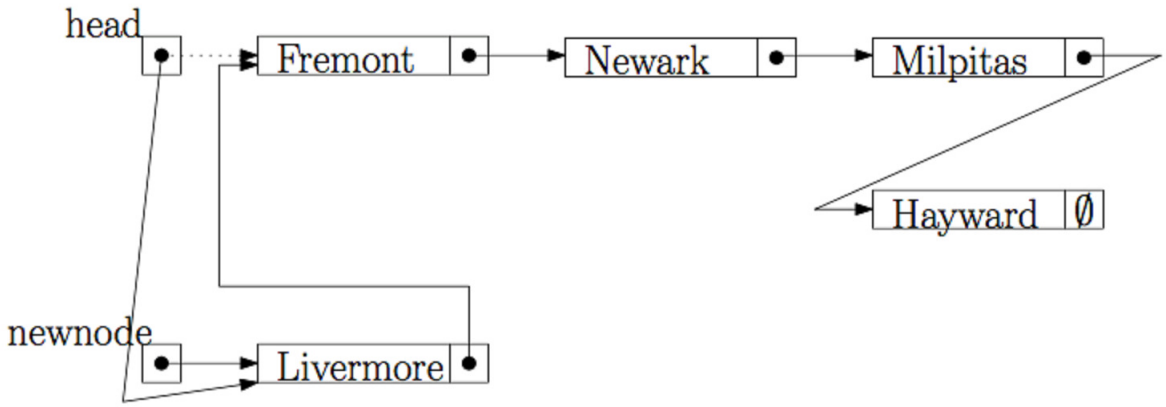
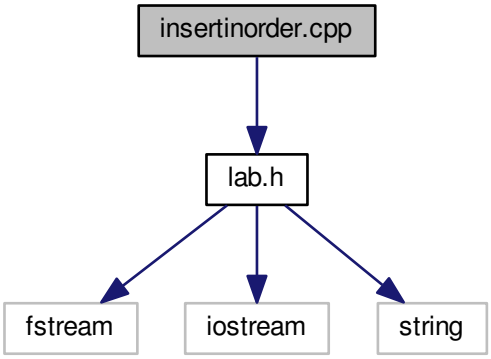


Figure 2: Linked List

```
11 {
12     //city = "Dublin"
13     NODE *newnode;
14 
15     // Allocate a new node:
16 
17     newnode = new NODE;
18     if (!newnode)
19         return FAILED;
20     //copy the info into the new node:
21 
22     newnode->city = city;
23 
24     //Link the new node to the list:
25 
26     newnode->next = head;
27     head = newnode;
28 
29     return OK;
30 }
```

8.5 insertinorder.cpp File Reference

`#include "lab.h"`
Include dependency graph for insertinorder.cpp:



Functions

- STATUS insertinorder (NODE ∗&head, std::string city)

8.5.1 Function Documentation

8.5.1.1 STATUS insertinorder (NODE ∗& *head,* std::string *city*)

Parameters

in	*head,string*	This functions takes in the head and a string which is whatever city the user defines

This function inserts a city into the list in a specific order.

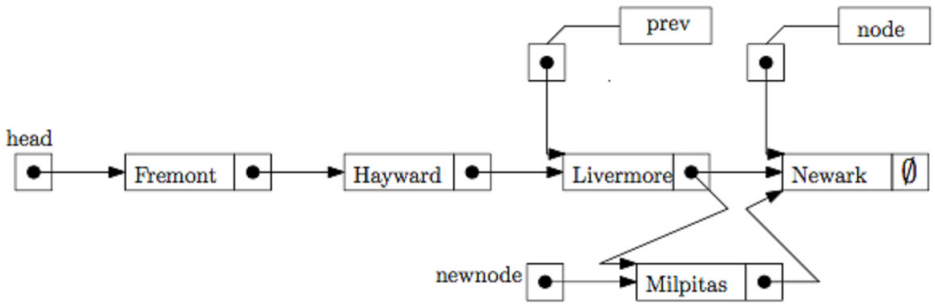


Figure 3: Linked List

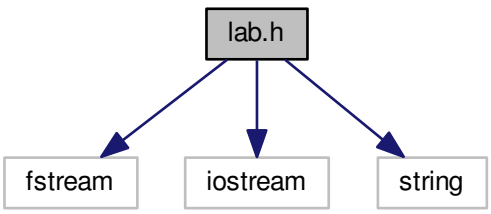
This shows how the pointers previous and next work as they traverse the list looking for a place to insert. Once it does it changes the pointers to add the value into the list

```
16 {
17     NODE *newnode;
18      // Allocate a new node:
19
20     newnode = new NODE;
21     if (!newnode)
22         return FAILED;
23
24     // Copy the info into newnode:
25
26     newnode->city = city;
27
28     // LINK newnode to the list:
29     // a) Find the right place to insert newnode (between "prev" and "node":
30
31     NODE *NODE = head, *prev = 0;
32     while (NODE && NODE->city <= city) {
33         //node->city <= city
34     prev = NODE;         //advance node and prev
35     NODE = NODE->next;
36 }
37     // b) Link newnode between prev and node
38
39     newnode->next = NODE; //append node to newnode
40     if (prev)
41         prev->next = newnode; //Insert after "prev"
42     else
43         head = newnode; //No prev: make new node the new head
44
45         return OK;
46     }
```

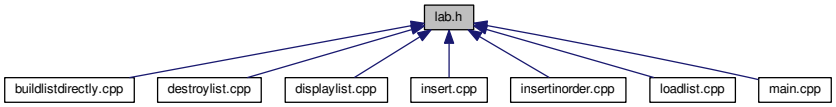
8.6 lab.h File Reference

```
#include <fstream>
#include <iostream>
#include <string>
```

Include dependency graph for lab.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct NODE

  City Structure.

Enumerations

- enum STATUS { FAILED, OK }

Functions

- STATUS insert (NODE *&head, std::string city)

  Insert inserts a new node at the beginning of the list.
- STATUS insertinorder (NODE *&head, std::string city)
- STATUS builddirectly (NODE *&head, string cities)
- void destroylist (NODE *head)
- void displaylist (NODE *head)
- NODE * loadlist (std::string filename)

8.6.1 Enumeration Type Documentation

8.6.1.1 enum STATUS

Enumerator

FAILED

OK

```
7 {FAILED, OK};
```

8.6.2 Function Documentation

8.6.2.1 STATUS builddirectly (NODE *& *head,* string *cities*)

```
5 {
6     string city;
7     NODE* tail;
8     ifstream ifs("cities");
9     if (! ifs)
10     return FAILED;
11
12 while(ifs >> city) {
13     NODE *newnode = new NODE;
14     if(!newnode)
15     return FAILED;
16
17     newnode->city = city;
18
19     newnode->next = 0;
20
21      if(!tail) {
22         head = newnode;}
23     else {
24         tail->next = newnode;
25     }
26     tail = newnode;
27
28 }
29 }
```

8.6.2.2 void destroylist (NODE * *head*)

Parameters

in	*head*	This is the only parameter the function takes.

This function completely destroys the list.

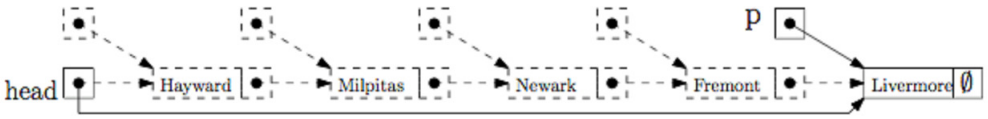


Figure 4: Linked List

As you can see that this function points the head at the node and then deletes it severing the tie to the list

```
14 {
15     NODE* node;
16     for(node = head; node; node = node->next)
17     {
18         cout << "deleting: "
19             << node->city << std::endl;
20         NODE* tmp = head->next;
21         delete head;
22         head = tmp;
23
24     }
25
26 }
```

8.6.2.3 void displaylist (NODE * *head*)

Parameters

in	*head*	This fucntion only takes a pointer as an argument.

This function displays/traverses the list This is what a linked list looks like to a user It traverses the list without manipulating any pointers. See the comments for further deatils

```
11 {
12     //first we set the node pointer equal to the head pointer
13     //Then while node exists this loop continues
14     //Then we set the pointer equal to the next pointer
15
16     for(NODE* node = head; node; node = node->next)
17     //This displays the data within the node
18     cout << node->city << endl;
19 }
```

8.6.2.4 STATUS insert (NODE *& *head*, std::string *city*)

Insert inserts a new node at the beginning of the list.

Parameters

in,out	*head*	The head of the linked list
in	*city*	The data in the node being inserted

Returns

A STATUS indicating if Insert was successful of not

Parameters

in,out	*head*	The head of the linked list
in	*city*	The data in the node being inserted

Returns

A STATUS indicating if Insert was successful of not

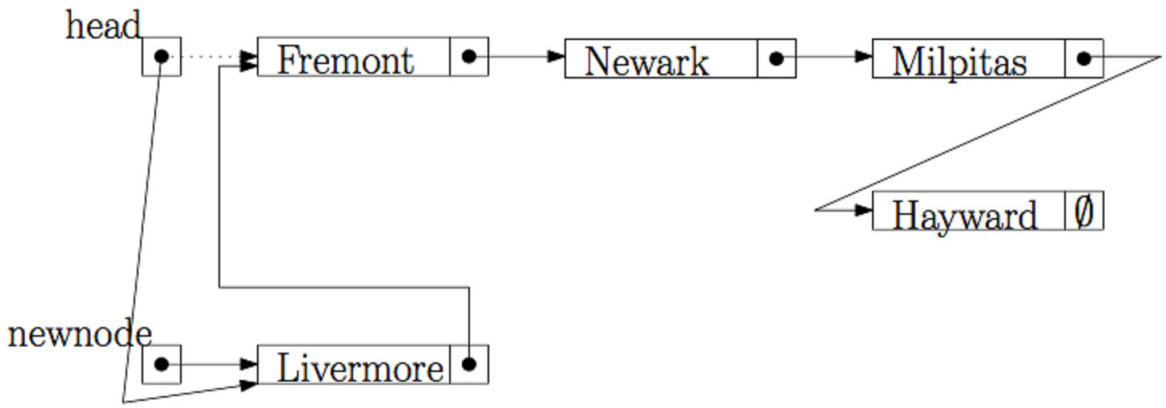


Figure 5: Linked List

```
11 {
12     //city = "Dublin"
13     NODE *newnode;
14
15     // Allocate a new node:
16
17     newnode = new NODE;
18     if (!newnode)
19         return FAILED;
20     //copy the info into the new node:
21
22     newnode->city = city;
23
24     //Link the new node to the list:
25
26     newnode->next = head;
27     head = newnode;
28
29     return OK;
30 }
```

8.6.2.5 STATUS insertinorder (NODE *& *head*, std::string *city*)

Parameters

in	*head,string*	This functions takes in the head and a string which is whatever city the user defines

This function inserts a city into the list in a specific order.

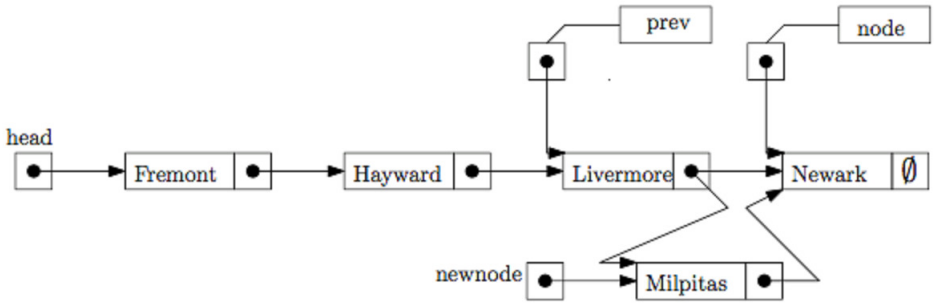


Figure 6: Linked List

This shows how the pointers previous and next work as they traverse the list looking for a place to insert. Once it does it changes the pointers to add the value into the list

```
16 {
17     NODE *newnode;
18      // Allocate a new node:
19
20     newnode = new NODE;
21     if (!newnode)
22         return FAILED;
23
24     // Copy the info into newnode:
25
26     newnode->city = city;
27
28     // LINK newnode to the list:
29     // a) Find the right place to insert newnode (between "prev" and "node":
30
31     NODE *NODE = head, *prev = 0;
32     while (NODE && NODE->city <= city) {
33         //node->city <= city
34     prev = NODE;         //advance node and prev
35     NODE = NODE->next;
36 }
37     // b) Link newnode between prev and node
38
39     newnode->next = NODE; //append node to newnode
40     if (prev)
41         prev->next = newnode; //Insert after "prev"
42     else
43         head = newnode; //No prev: make new node the new head
44
45         return OK;
46     }
```

8.6.2.6 NODE* loadlist (std::string *filename*)

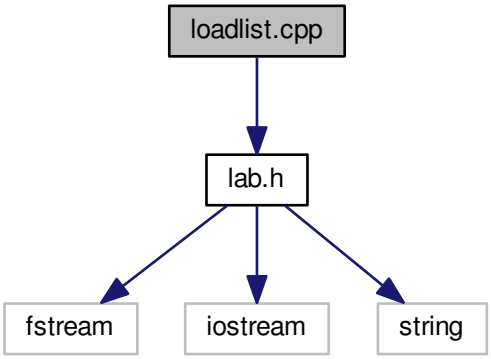
This function loads the dictionary into a linked list it takes the filename "cities" and loads that as cities the variable Then it uses the insert function to create the list after the head pointer. Then at the end it returns head so that we can use it in other functions.

```
10 {
11     NODE* head = 0; // this is where we declare the head as null to create the list
12     std::ifstream ifs(filename.c_str());
13     string city;
14     while(ifs >> city)
15         if(insert(head,city) == FAILED)
16          cerr << "error on insert\n";
17
18     return head;
19 }
```

8.7 loadlist.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for loadlist.cpp:



Functions

- NODE * loadlist (std::string filename)

8.7.1 Function Documentation

8.7.1.1 NODE* loadlist (std::string *filename*)

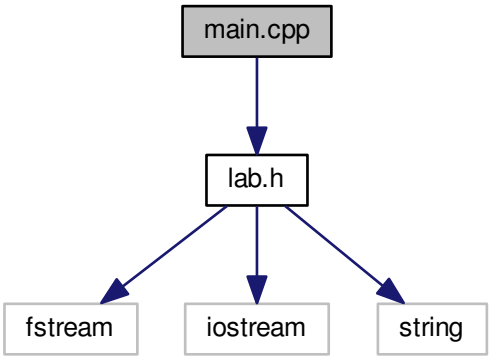
This function loads the dictionary into a linked list it takes the filename "cities" and loads that as cities the variable Then it uses the insert function to create the list after the head pointer. Then at the end it returns head so that we can use it in other functions.

```
10 {
11     NODE* head = 0; // this is where we declare the head as null to create the list
12     std::ifstream ifs(filename.c_str());
13     string city;
14     while(ifs >> city)
15         if(insert(head,city) == FAILED)
16          cerr << "error on insert\n";
17 
18     return head;
19 }
```

8.8 main.cpp File Reference

```
#include "lab.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

8.8.1 Function Documentation

8.8.1.1 int main ( )

This is the main function of the program. This function uses functions described in other files to build, display,insert, and destroy the list.

Parameters

in	*Nothing*	This program takes no inputs
out	*0*	The program exits succesfully

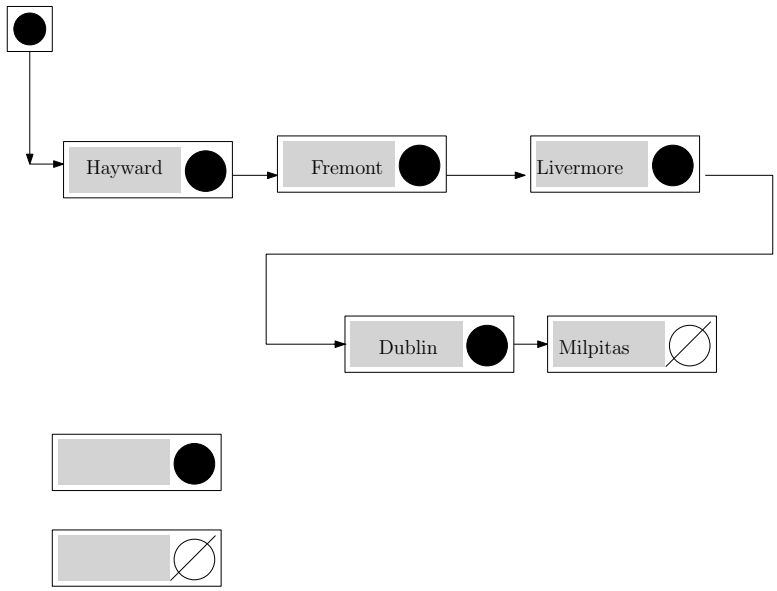


Figure 7: Linked List

This is what happens when the list is displayed, built or loaded.

```
13 {
14     NODE* head = loadlist("cities");
15     displaylist(head);
16
17
18
19     if(head)
20     {
21         cout << "\nInsert in order Sacramento & Newark" << endl;
22         insertinorder(head, "Sacramento");
23         insertinorder(head, "Newark");
24         displaylist(head);
25
26         cout << "\nDestroying and rebuilding list" << endl;
27         destroylist(head);
28         builddirectly(head, "cities");
29
30         cout << "\nInsert Sacramento & Newark" << endl;
31         insert(head, "Davis");
32         insert(head, "Newark");
33
34         displaylist(head);
35
36
37     }
38     return 0;
39 }
```

8.9 specification.dox File Reference

Index