CS124 Lab2 - Linked List Program

Generated by Doxygen 1.8.8

Sat Sep 23 2017 00:02:34

Contents

1 Specification 1

2 Analysis 1

3 Design 1

4 1 2

5 Class Index 3

5.1 Class List . . . 3

6 File Index 4

6.1 File List . . . 4

7 Class Documentation 4

7.1 NODE Struct Reference . . . 4

7.1.1 Member Data Documentation . . . 5

8 File Documentation 5

8.1 BuildListDirectly.cpp File Reference . . . 5

8.1.1 Function Documentation . . . 5

8.2 destroyList.cpp File Reference . . . 6

8.2.1 Function Documentation . . . 7

8.3 displayList.cpp File Reference . . . 8

8.3.1 Function Documentation . . . 8

8.4 doc.dox File Reference . . . 9

8.5 Insert.cpp File Reference . . . 9

8.5.1 Function Documentation . . . 9

8.6 InsertInOrder.cpp File Reference . . . 11

8.6.1 Function Documentation . . . 11

8.7 lab2.h File Reference . . . 12

8.7.1 Enumeration Type Documentation . . . 13

8.7.2 Function Documentation . . . 13

8.8 loadList.cpp File Reference . . . 18

8.8.1 Function Documentation . . . 19

8.9 loadList2.cpp File Reference . . . 20

8.9.1 Function Documentation . . . 20

8.10 loadList3.cpp File Reference . . . 21

8.10.1 Function Documentation 22
8.11 main.cpp File Reference 23
8.11.1 Function Documentation 23

Index 25

1 Specification

This program lets the user see a linked list printed out in three different versions. It is also a program that helps practice the programmer's pointer skills.

2 Analysis

This program first runs with multiple functions, one function that displays the list, another which destroys the list, and the other three functions switches the format of how the program's list is printed for the user.

3 Design

The program first prints out and tells the user it is printing out the first output of the program which is the basic insert function with the list of cities. The program then destroys the list and prints out again which tells the user it is empty. Then it prints out the second version of the list which is in alphbetical order. It then repeats the cycle which destroys the list again and tries to print out the list, but tells the user it is empty. Lastly, the third print out of the list prints the list backwards as it written. Then ends the program.

4 1

```
debian@debian:~/lab2$ ./lab
Displaying List with Insert

 I
List:
Newark
Hayward
Fremont
Dublin
Milpitas

Deleting: Newark
Deleting: Hayward
Deleting: Fremont
Deleting: Dublin
Deleting: Milpitas

List:
List is empty

Displaying List with InsertInOrder


List:
Dublin
Fremont
Hayward
Milpitas
Newark
```

```
Deleting: Dublin
Deleting: Fremont
Deleting: Hayward
Deleting: Milpitas
Deleting: Newark

List:
List is empty

Displaying List with BuildDirectly

Error on Insert
Error on Insert
Error on Insert
Error on Insert
Error on Insert

List:
Newark

Deleting: Newark

List:
List is empty
```

5 Class Index

5.1 Class List

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

NODE 4

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

BuildListDirectly.cpp 5

destroyList.cpp 6

displayList.cpp 8

Insert.cpp 9

InsertInOrder.cpp 11

lab2.h 12

loadList.cpp 18

loadList2.cpp 20

loadList3.cpp 21

main.cpp 23

7 Class Documentation

7.1 NODE Struct Reference

`#include <lab2.h>`

Collaboration diagram for NODE:

Public Attributes

- std::string city
- NODE * next
- NODE * prev

7.1.1 Member Data Documentation

7.1.1.1 std::string NODE::city

7.1.1.2 NODE* NODE::next

7.1.1.3 NODE* NODE::prev

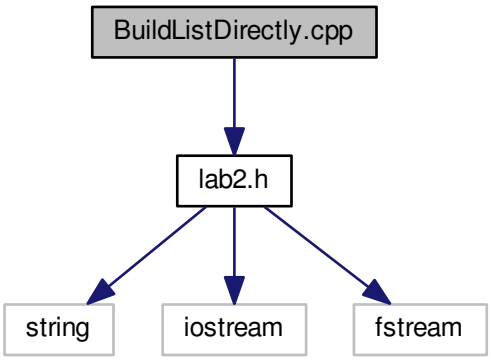
The documentation for this struct was generated from the following file:

- lab2.h

8 File Documentation

8.1 BuildListDirectly.cpp File Reference

```
#include "lab2.h"
```
Include dependency graph for BuildListDirectly.cpp:



Functions

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

  Insert the list directly.

8.1.1 Function Documentation

8.1.1.1 STATUS BuildListDirectly (NODE *& *head*, std::string *city*)

Insert the list directly.

Parameters

in,out	*head,Inserting*	the head of linked list, but also called the header //they the same
in	*city,The*	data of the NODE is inserted

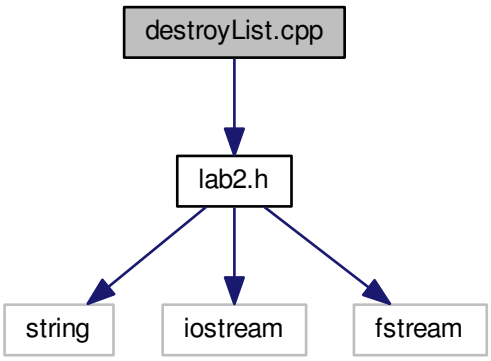
Returns

A STATUS indicating if BuildListDirectly was successful or not

```
4 {
5     NODE *tail=0, *newnode; // calls the pointers
6     head = 0;
7     newnode = new NODE; // allocates a new node
8     while (head != NULL) //while loop to use link list
9     {
10         newnode->city = city; //copy list information to newnode
11
12     }
13         newnode->next = 0; //puts node of next pointer to null
14         if (!tail) //if it is not tail then head is null
15             head = newnode;
16         else
17             tail->next = newnode; //else connect tail with newnode
18         tail = newnode; //tail equals to newnode
19     return OK;
20 } //end of function
```

8.2 destroyList.cpp File Reference

`#include "lab2.h"`
Include dependency graph for destroyList.cpp:



Functions

- void destroyList (NODE *head)

  Destroys the list.

8.2.1 Function Documentation

8.2.1.1 void destroyList (NODE * *head*)

Destroys the list.

Parameters

filename	Name of the file

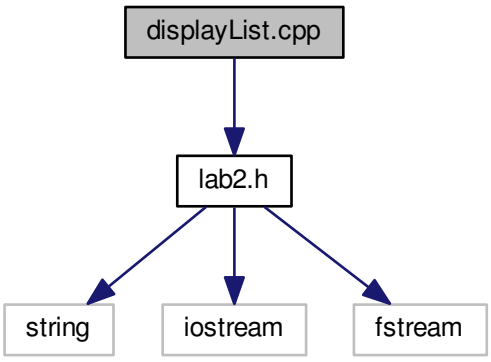
Returns

a pointer 'head; to the link list

```
4 {
5     std::cout <<  std::endl; //to space out lists
6     NODE* node; //creating node
7     for(node = head; node; node = node->next) //loop to delete list
8     {
9             std::cout << "Deleting: "
10             << node->city << std::endl; // to tell user its deleted
11             NODE* tmp = head->next; //assigns head pointer to tmp
12             //delete head;
13             head = tmp; //head becomes tmp which is deleted
14
15     }
16
17 }
```

8.3 displayList.cpp File Reference

`#include "lab2.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*)

This function display th link list

Parameters

head	is a pointer to beginning node of linklist

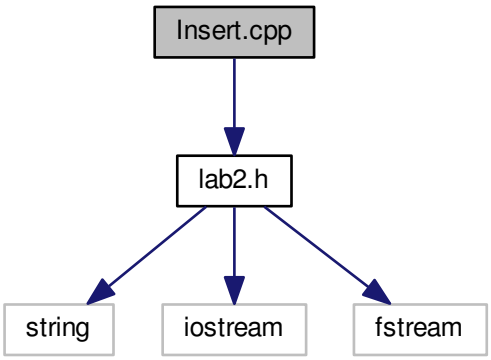
for Node equals head, and node equals node pointing to next, it will print out the city and will continue until it prints out all the list

```
4 {
5     std::cout << "\nList: " << std::endl; //tells user the list is built
6     NODE* node;
7     if(head)
12          for(NODE* node = head; node; node = node->next)
13          {
14              std::cout << node->city << std::endl;
15          }
16          else
17              std::cout << "List is empty\n" << std::endl;
18              // if there is nothing in the list, then it will
19              //print out this statement
20
21 }
```

8.4 doc.dox File Reference

8.5 Insert.cpp File Reference

`#include "lab2.h"`
Include dependency graph for Insert.cpp:



Functions

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

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

8.5.1 Function Documentation

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

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

Parameters

in,out	*head,Inserting*	the head of linked list, but also called the header //they the same
in	*city,The*	data of the NODE is inserted

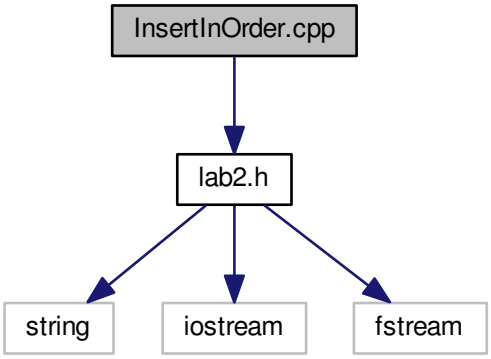
Returns

A STATUS indicating if Insert was successful or not A function to insert the linked list into the program which is then printed out for the user

```
15 {
16     NODE* newnode = new NODE; //allocating new node
17     newnode->city = city; //copying information into node
18     newnode->next = head; //pointing to head
19     head = newnode; //node equals head
20     return OK;
21 }
```

8.6 InsertInOrder.cpp File Reference

`#include "lab2.h"`
Include dependency graph for InsertInOrder.cpp:



Functions

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

  Insert the list in order.

8.6.1 Function Documentation

8.6.1.1 STATUS InsertInOrder (NODE *& *head,* std::string *city*)

Insert the list in order.

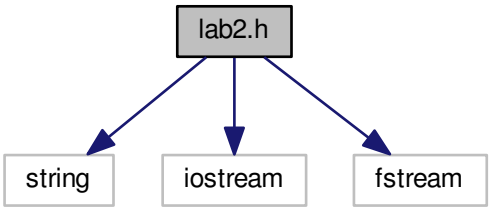
Code that inserts the information of the linked list, but in alphabetical order.

```
7 {
8
9     NODE *newnode; //creating a new node called newnode
10
11     newnode = new NODE; //allocating a new node
12     if(!newnode)
13         return FAILED; //debugger if it fails
14
15     newnode->city=city; //to copy information into list
16
17     NODE *node = head, *prev = 0;
18     //While node and nodepointing to city is less than
19     //or equal to city, prev equals node and node equals
20     // node points to next
21     while (node && node->city <=city)
22     {
23             prev = node;
24             node = node->next;
25     }
26
27     newnode->next = node;
28     if(prev)
29     prev->next = newnode;
30     else
31         head = newnode;
32
33     return OK;
34
35 }
```

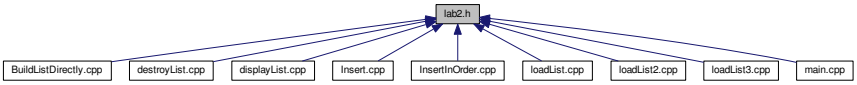
8.7 lab2.h File Reference

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

Include dependency graph for lab2.h:



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



Classes

- struct NODE

Enumerations

- enum STATUS { FAILED, OK }

  City Structure.

Functions

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

  Insert puts a new node at the beginning of the list.
- NODE * loadList (std::string filename)

  This function loads the data from a file.
- void displayList (NODE *head)
- void destroyList (NODE *head)

  Destroys the list.
- NODE * loadList2 (std::string filename)

  This function loads the list in alphabetical order.
- STATUS InsertInOrder (NODE *&head, std::string city)

  Insert the list in order.
- NODE * loadList3 (std::string filename)

  This function builds the list directly.
- STATUS BuildListDirectly (NODE *&head, std::string city)

  Insert the list directly.

8.7.1 Enumeration Type Documentation

8.7.1.1 enum STATUS

City Structure.

This is a structue used to create each node of the linked list of cities enumerate something, number it statuses of new type(string, int, double)

Enumerator

FAILED

OK

```
16 {FAILED, OK};
```

8.7.2 Function Documentation

8.7.2.1 STATUS BuildListDirectly (NODE *& *head*, std::string *city*)

Insert the list directly.

Parameters

in,out	*head,Inserting*	the head of linked list, but also called the header //they the same
in	*city,The*	data of the NODE is inserted

Returns

A STATUS indicating if BuildListDirectly was successful or not

```
4 {
5     NODE *tail=0, *newnode; // calls the pointers
6     head = 0;
7     newnode = new NODE; // allocates a new node
8     while (head != NULL) //while loop to use link list
9     {
10         newnode->city = city; //copy list information to newnode
11
12     }
13         newnode->next = 0; //puts node of next pointer to null
14         if (!tail) //if it is not tail then head is null
15             head = newnode;
16         else
17             tail->next = newnode; //else connect tail with newnode
18         tail = newnode; //tail equals to newnode
19     return OK;
20 } //end of function
```

8.7.2.2 void destroyList (NODE * *head*)

Destroys the list.

Parameters

filename	Name of the file

Returns

a pointer 'head; to the link list

```
4 {
5     std::cout <<  std::endl; //to space out lists
6     NODE* node; //creating node
7     for(node = head; node; node = node->next) //loop to delete list
8     {
9             std::cout << "Deleting: "
10             << node->city << std::endl; // to tell user its deleted
11             NODE* tmp = head->next; //assigns head pointer to tmp
12             //delete head;
13             head = tmp; //head becomes tmp which is deleted
14
15     }
16
17 }
```

8.7.2.3 void displayList (NODE * *head*)

This function display th link list

Parameters

head	is a pointer to beginning node of linklist

for Node equals head, and node equals node pointing to next, it will print out the city and will continue until it prints out all the list

```
4 {
```

```
5      std::cout << "\nList: " << std::endl; //tells user the list is built
6      NODE* node;
7      if(head)
12          for(NODE* node = head; node; node = node->next)
13          {
14              std::cout << node->city << std::endl;
15          }
16          else
17              std::cout << "List is empty\n" << std::endl;
18              // if there is nothing in the list, then it will
19              //print out this statement
20
21  }
```

8.7.2.4 STATUS Insert (NODE *& *head*, std::string *city*)

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

Parameters

in,out	*head,Inserting*	the head of linked list, but also called the header //they the same
in	*city,The*	data of the NODE is inserted

Returns

A STATUS indicating if Insert was successful or not

Parameters

in,out	*head,Inserting*	the head of linked list, but also called the header //they the same
in	*city,The*	data of the NODE is inserted

Returns

A STATUS indicating if Insert was successful or not A function to insert the linked list into the program which is then printed out for the user

```
15 {
16      NODE* newnode = new NODE; //allocating new node
17      newnode->city = city; //copying information into node
18      newnode->next = head; //pointing to head
19      head = newnode; //node equals head
20      return OK;
21  }
```

8.7.2.5 STATUS InsertInOrder (NODE *& *head*, std::string *city*)

Insert the list in order.

Parameters

in,out	*head,Inserting*	the head of linked list, but also called the header //they the same
in	*city,The*	data of the NODE is inserted

Returns

A STATUS indicating if Insert was successful or not

Code that inserts the information of the linked list, but in alphabetical order.

```
7  {
```

```
8
9     NODE *newnode; //creating a new node called newnode
10
11     newnode = new NODE; //allocating a new node
12     if(!newnode)
13         return FAILED; //debugger if it fails
14
15     newnode->city=city; //to copy information into list
16
17     NODE *node = head, *prev = 0;
18     //While node and nodepointing to city is less than
19     //or equal to city, prev equals node and node equals
20     // node points to next
21     while (node && node->city <=city)
22     {
23             prev = node;
24             node = node->next;
25     }
26
27     newnode->next = node;
28     if(prev)
29     prev->next = newnode;
30     else
31         head = newnode;
32
33     return OK;
34
35 }
```

8.7.2.6 NODE* loadList (std::string *filename*)

This function loads the data from a file.

The file contains a list of cities, each on a seperate line.

Parameters

filename	Name of file

Returns

a pointer 'head' to the link list

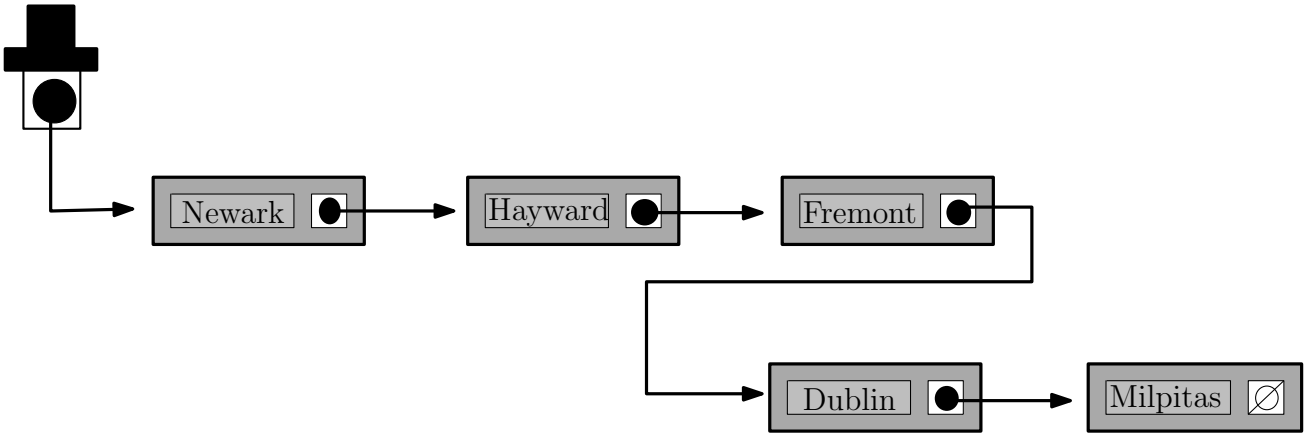


Figure 1: Linked List

loads the linked list into the program which lets the function DisplayList print it out for the user

```
10 {
```

```
11     NODE* head = 0; //puts head to null
12     std::ifstream inputFile(filename.c_str()); //taking input
13     std::string city; //to use string city for list
14     while (inputFile >> city) // loop to grab information
15         if (Insert(head, city) == FAILED)
16             std::cerr << "Error on Insert\n";
17         //if statements that catches error if it cannot
18         //get linked list information
19     return head; //returns head
20 }
```

8.7.2.7 NODE* loadList2 (std::string *filename*)

This function loads the list in alphabetical order.

The file contains a list of cities, each on a seperate line.

Parameters

filename	Name of file

Returns

a pointer 'head' to the link list

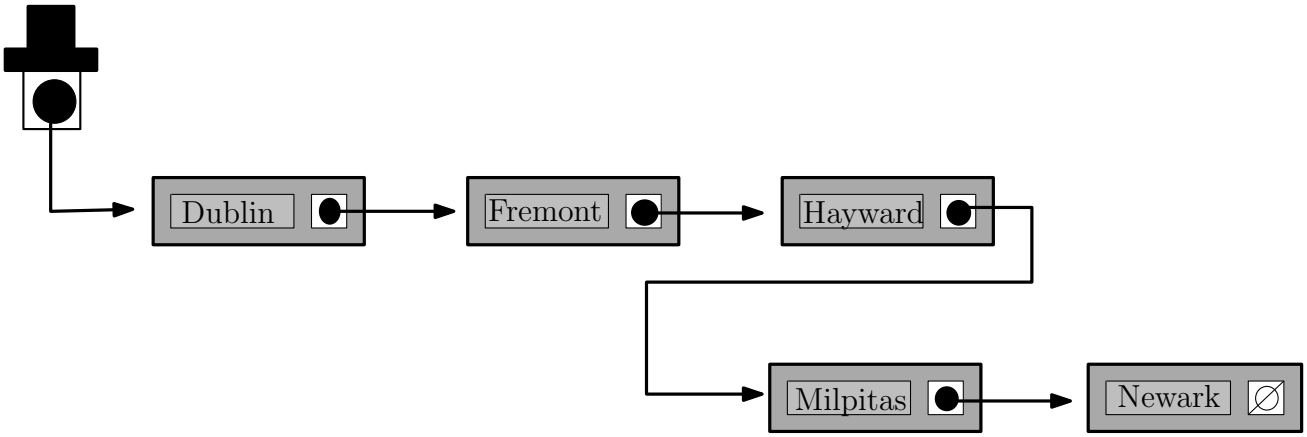


Figure 2: Linked List

```
6 {
7     NODE* head = 0; //puts head to null
8     std::ifstream inputFile(filename.c_str()); //taking input
9     std::string city; //to use string city for list
10     while (inputFile >> city) // loop to grab information
11         if (InsertInOrder(head, city) == FAILED)
12             std::cerr << "Error on Insert\n";
13             //if statements that catches error if it cannot
14         //get linked list information
15     return head; //returns head
16 }
```

8.7.2.8 NODE* loadList3 (std::string *filename*)

This function builds the list directly.

The file contains a list of cities, each on a seperate line.

Parameters

filename	Name of file

Returns

a pointer 'head' and also a pointer 'tail' to the link list

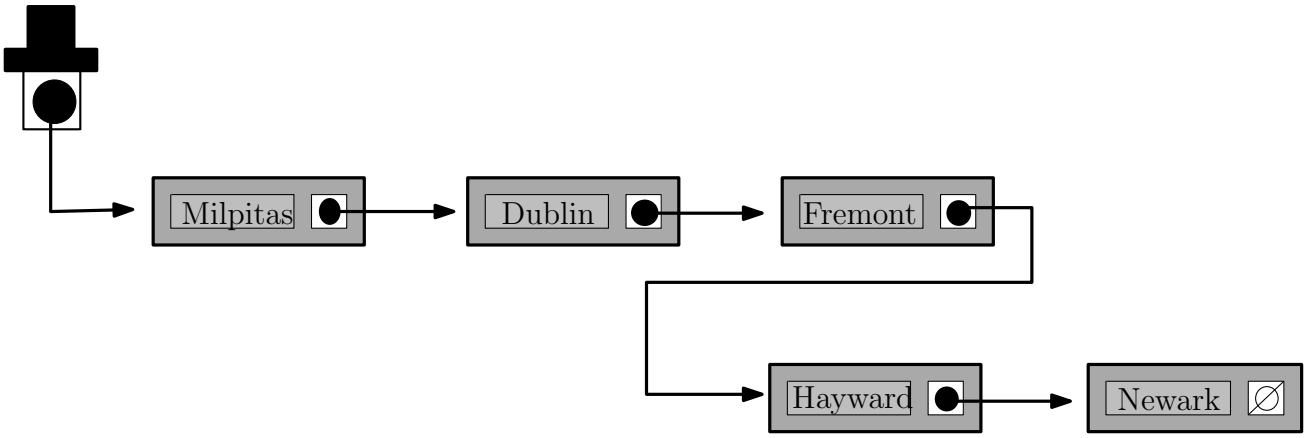


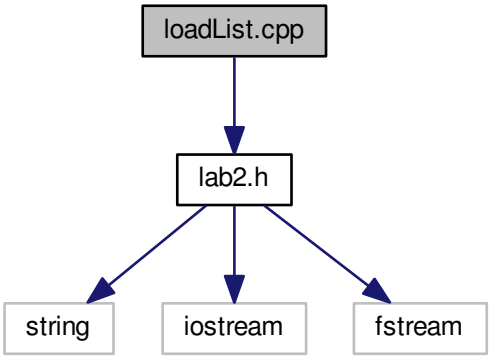
Figure 3: Linked List

```
6 {
7     NODE* head = 0;  //puts head to null
8     std::ifstream inputFile(filename.c_str()); //taking input
9     std::string city; //to use string city for list
10     while (inputFile >> city) // loop to grab information
11         if (BuildListDirectly(head, city) == FAILED)
12             std::cerr << "Error on Insert\n";
13             //if statements that catches error if it cannot
14         //get linked list information
15     return head; //returns head
16 }
```

8.8 loadList.cpp File Reference

```
#include "lab2.h"
```

Include dependency graph for loadList.cpp:



Functions

- NODE * loadList (std::string filename)

  This function loads the data from a file.

8.8.1 Function Documentation

8.8.1.1 NODE* loadList (std::string *filename*)

This function loads the data from a file.

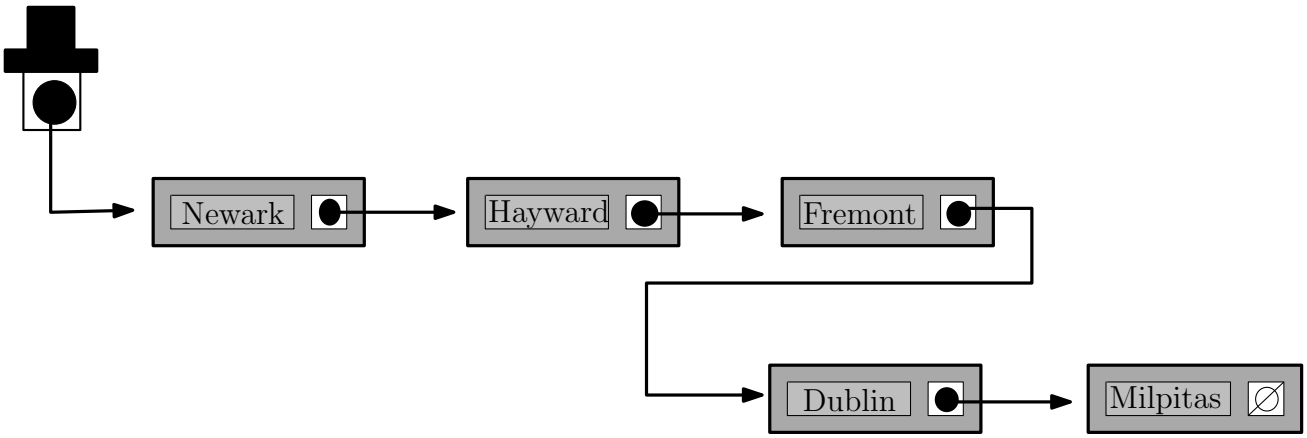


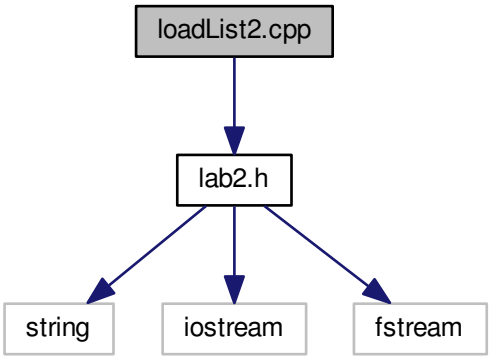
Figure 4: Linked List

loads the linked list into the program which lets the function DisplayList print it out for the user

```
10 {
11     NODE* head = 0; //puts head to null
12     std::ifstream inputFile(filename.c_str()); //taking input
13     std::string city; //to use string city for list
14     while (inputFile >> city) // loop to grab information
15         if (Insert(head, city) == FAILED)
16             std::cerr << "Error on Insert\n";
17         //if statements that catches error if it cannot
18         //get linked list information
19     return head; //returns head
20 }
```

8.9 loadList2.cpp File Reference

`#include "lab2.h"`
Include dependency graph for loadList2.cpp:



Functions

- NODE * loadList2 (std::string filename)

  This function loads the list in alphabetical order.

8.9.1 Function Documentation

8.9.1.1 NODE* loadList2 (std::string *filename*)

This function loads the list in alphabetical order.

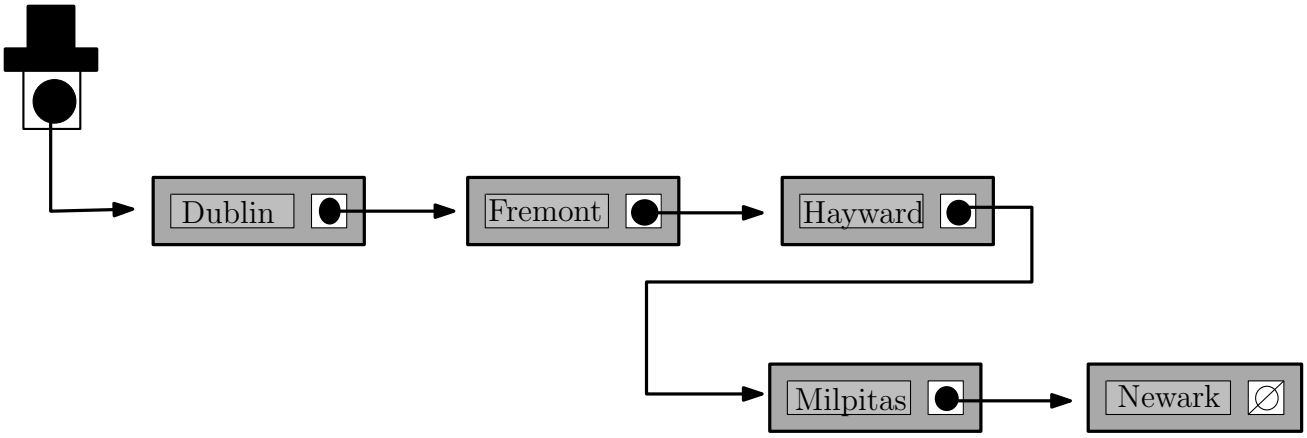


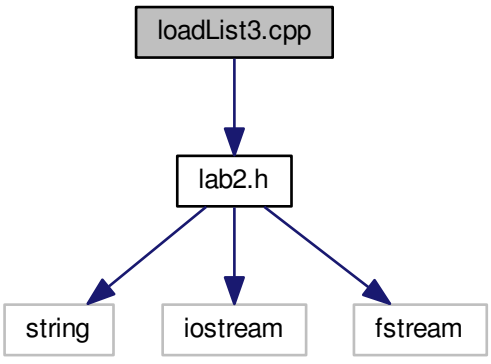
Figure 5: Linked List

```
6 {
7     NODE* head = 0; //puts head to null
8     std::ifstream inputFile(filename.c_str()); //taking input
9     std::string city; //to use string city for list
10     while (inputFile >> city) // loop to grab information
11         if (InsertInOrder(head, city) == FAILED)
12             std::cerr << "Error on Insert\n";
13             //if statements that catches error if it cannot
14         //get linked list information
15     return head; //returns head
16 }
```

8.10 loadList3.cpp File Reference

`#include "lab2.h"`

Include dependency graph for loadList3.cpp:



Functions

- NODE * loadList3 (std::string filename)

  This function builds the list directly.

8.10.1 Function Documentation

8.10.1.1 NODE* loadList3 (std::string *filename*)

This function builds the list directly.

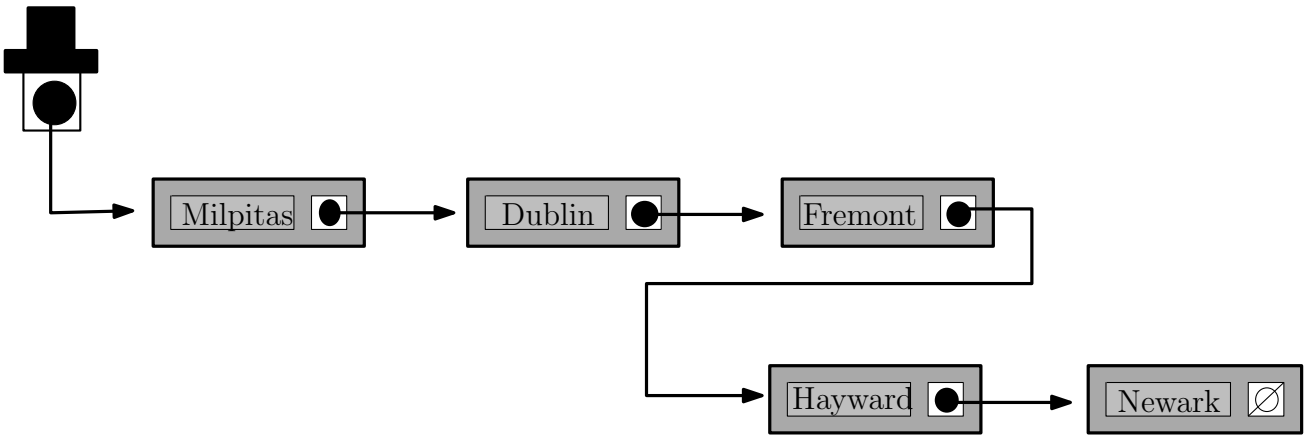
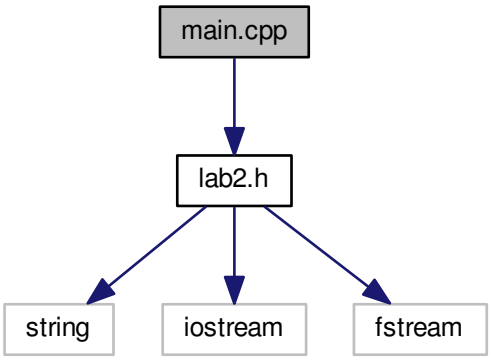


Figure 6: Linked List

```
6 {
7     NODE* head = 0;  //puts head to null
8     std::ifstream inputFile(filename.c_str()); //taking input
9     std::string city; //to use string city for list
10     while (inputFile >> city) // loop to grab information
11         if (BuildListDirectly(head, city) == FAILED)
12             std::cerr << "Error on Insert\n";
13             //if statements that catches error if it cannot
14         //get linked list information
15     return head; //returns head
16 }
```

8.11 main.cpp File Reference

`#include "lab2.h"`
Include dependency graph for main.cpp:



Functions

- int main ()

8.11.1 Function Documentation

8.11.1.1 int main ( )

main.cpp Displays the linked list in three different ways, one way is normal insertion, second is in alphabetical order, and third is directly. They all build, then get destroyed, and rebuild to make sure it is destroyed.

```
11          {
12 std::cout<< "Displaying List with Insert\n\n";
13     NODE* head = loadList("cities");
14     if(head)
15     {
16         displayList(head);
17         destroyList(head);
18         head =0;
19         displayList(head);
20     }
21 std::cout<< "Displaying List with InsertInOrder\n\n";
22     head = loadList2("cities");
23     if(head)
24     {
25         displayList(head);
26         destroyList(head);
27         head=0;
28         displayList(head);
29     }
30 std::cout<< "Displaying List with BuildDirectly\n\n";
31     head = loadList3("cities");
32     if(head)
33     {
34         displayList(head);
35         destroyList(head);
```

```
36          head=0;
37          displayList(head);
38      }
39      return 0;
40  }
```

Index

FAILED
    lab2.h, 13

lab2.h
    FAILED, 13
    OK, 13

OK
    lab2.h, 13