Linked List program, written by Mohammed Abbas under David Topham

Create a file containing a few names of cititles (atleast 10) in random order. Place each name on a seperate line. Write a program that reads the names from the file and puts them into a linked list.

Implement the following three ways to build the list.

Use struct to create a node to store each city and its next city pointer.

Write code in a seperate file for each of these on a seperate line.

- Call the insert(...) function within a loop.
- Call the insertInOrder(...) function within a loop.
- Build the list directly.

Functions to call:

- loadListUsingInsert
- loadListUsingInsertInOrder
- loadListDirectly
- displayList
- insert
- insertInOrder
- destroyList

lab2.h Source Code

```
#include <iostream>
#include <fstream>
#include <cstdlib>
struct NODE {std::string data; NODE* next;};
bool loadListUsingInsert(std::string,NODE*&);
bool insert(std::string,NODE*&);
void displayList(NODE*);
void destroyList(NODE*&, NODE*&);
bool insertAtEnd(std::string, NODE*&);
bool loadListDirectly(std::string, NODE*&);
bool insertInOrder(NODE *&, std::string);
bool loadListUsinginsertInOrder(std::string, NODE *&);
```

The header file will: include any library files needed such as vector, fstream, and iostream create a struct named NODE that will create a new node for pointers to point to of the NODE type. The pointer 'next' and data of type string that will store data passed by insert functions listed here.

main.cpp Source Code

main will: the main function will test all of the linked lists and destory the lists after the function is executed to allocate memory and prevent memory leak.

```
#include "lab2.h"
int main()
{
    NODE* head = nullptr;
    NODE* tail = nullptr;
    if(loadListUsingInsert("cities.dat", head))
    {
        std::cout << "--Load List Using Insert--" << std::endl;
        displayList(head);
        destroyList(head, tail);
    }
    else
        std::cout << "list not loaded" << std::endl;
    if(loadListDirectly("cities.dat", head))
    {
        std::cout << "--Load List Directly--" << std::endl;
        displayList(head);
        destroyList(head, tail);
    }
    else
        std::cout << "list not loaded" << std::endl;
    if(loadListUsinginsertInOrder("cities.dat", head))
    {
        std::cout << "--Load List Using Insert In Order--" << std::endl;
        displayList(head);
```

loadListUsingInsert.cpp Source Code

```
#include "lab2.h"
bool loadListUsingInsert(std::string f,NODE* &head)
{
    std::ifstream ifs(f);
    if(not ifs)
        return false;
    std::string s;
    while(ifs >> s)
    {
        insert(s,head);
        }
    return true;
}
```

load list using insert will: include the declaration from the header file created and create an object call ifs to read the data file containing the cities. This load the data from the file and stores into variable f. simultaeniously the node is storing the data into the variable s and head.

loadListUsingInsertInOrder.cpp Source Code

```
#include "lab2.h"
bool loadListUsinginsertInOrder(std::string f, NODE *&head)
{
    std::ifstream ifs(f);
    if (not ifs) return false;
    std::string s;
    while(ifs >> s) {insertInOrder(head,s);}
    return true;
}
```

load list using insert in order will: include declarations from the header file and do almost the same process as loadListUsingInsert but, exacute the data from the file in a different manner. the function insertInOrder will be described in the source code.

loadListDirectly.cpp Source Code

```
#include "lab2.h"
bool loadListDirectly(std::string f, NODE*& head)
{
    std::ifstream ifs(f);
    if(not ifs)
        return false;
    std::string s;
    while(ifs >> s)
    {
        insertAtEnd(s,head);
        }
    return true;
}
```

load list directly will: this performs the same as the last two functions except when the data is fed into the insertAtEnd function the list is displayed as it is written in the data file.

displayList.cpp Source Code

```
#include "lab2.h"
void displayList(NODE* head)
{
    NODE* newnode = new NODE;
    for(NODE* node = head; node ; node = node->next)
        std::cout << node->data << std::endl;
}
```

display list will: This function creates a node pointer called newnode. This will point to data of type node. The for loop creates a loop that prints the data stored in the data file line by line until the node is NOT null.

insert.cpp Source Code

```
#include "lab2.h"
bool insert(std::string s,NODE*& head)
{
    NODE* newnode = new NODE;
    if(not newnode)
        return false;
    newnode->data = s;
    newnode->next = head;
    head = newnode;
    return true;
}
```

insert will: insert function creates a new node and assigns the data from the data file to the new node that has been created. it returns true if the node has been successfully created.

insertInOrder.cpp Source Code

```
#include "lab2.h"
bool insertInOrder(NODE *&head, std::string data){
NODE *newnode;
newnode = new NODE;
if(not newnode){return false;}
newnode->data = data;
NODE *node = head, *prev = 0;
while(node && node->data <= data)
{prev = node; node = node->next;}
newnode->next = node;
if(prev)
{prev->next = newnode;}
else
head = newnode;
    return false;
}
```

insert in order will: this will perform almost the same as the insert function except with a while loop that searches the link list created until the node is Not null. this will display the lists of cities in order.

destoryList.cpp Source Code

```
#include "lab2.h"
void destroyList(NODE*& head, NODE*& tail)
{
    NODE *current = head;
    while (current != NULL)
    {
        head = head->next;
        delete current;
        current = head;
}
head = NULL;
tail = NULL;
std::cout << "----------------------------------------" << std::endl;
std::cout << "Hmmm..." << std::endl;
std::cout << "These arent the nodes im looking for..." << std::endl;
std::cout << "[List Terminated]" << std::endl;
std::cout << "----------------------------------------" << std::endl;
}
```

destroy list will: this function will loop with a new node created called current to assume the value of the head of the node for future deletion. untill the current node is not pointing to or is null the nodes are deleted to allocate memory.

Image of program running using the loadListUsingInsert:

```
debian@debian:~/lab2$ ./lab
--Load List Using Insert--
santaclara
oakland
pleasonton
santacruz
sanjose
milpitas
unioncity
hollister
sanfrancisco
fremont
newark
Cities:
----------------------------------------
Hmmm...
These arent the nodes im looking for...
[List Terminated]
----------------------------------------
```

Image of program running using the loadListDirectly:

```
--Load List Directly--
Cities:
newark
fremont
sanfrancisco
hollister
unioncity
milpitas
sanjose
santacruz
pleasonton
oakland
santaclara
----------------------------------------
Hmmm...
These arent the nodes im looking for...
[List Terminated]
----------------------------------------
```

Image of program running using the loadListUsinginsertInOrder:

```
--Load List Using Insert In Order--
Cities:
fremont
hollister
milpitas
newark
oakland
pleasonton
sanfrancisco
sanjose
santaclara
santacruz
unioncity
----------------------------------------
Hmmm...
These arent the nodes im looking for...
[List Terminated]
----------------------------------------
```

Image of program the memory leak checker:

```
LEAK SUMMARY:
   definitely lost: 24 bytes in 3 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 0 bytes in 0 blocks
        suppressed: 0 bytes in 0 blocks
```

there was a little memory leak that i was not able to find however, the program runs as it should.

Diagram of list created using insert:

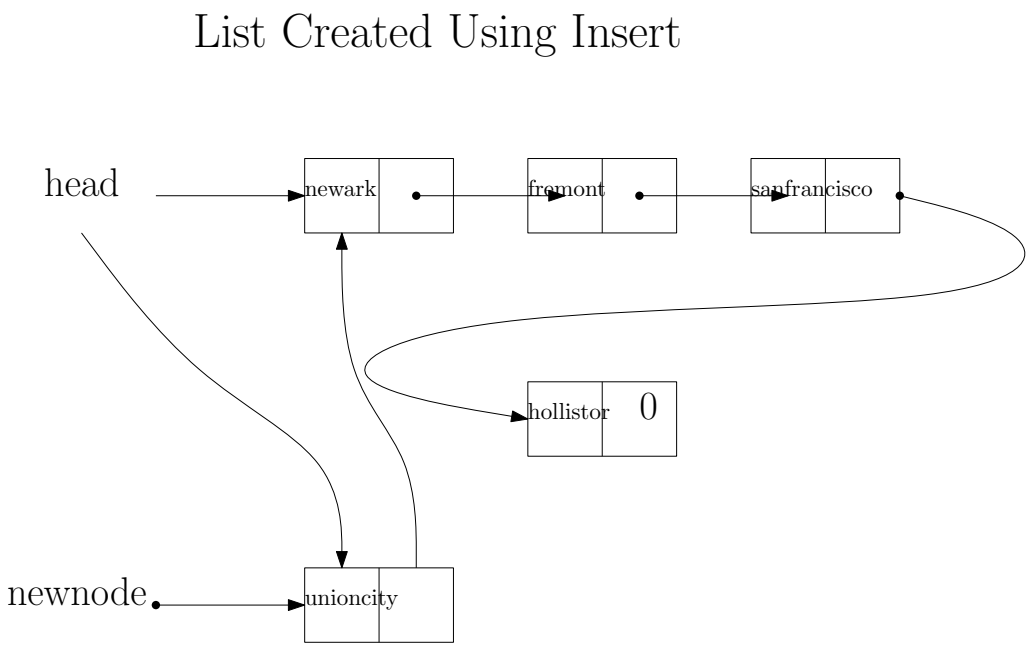

Diagram of list created using insert in order: