LAB 2: LINKED LISTS BY:AMULDEEP DHILLON

Implement the following three ways to build the list:

1. Call the insert(...) function within a loop.
2. Call the insertInOrder(...) function within a loop.
3. Build the list directly.

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

Write code in a separate file for each of these functions (and name file using function name):

1. loadListUsingInsert
2. loadListUsinginsertInOrder
3. loadListDirectly
4. displayList
5. insert
6. insertInOrder
7. destroyList

Create a diagram illustrating the insert (fig 14-3), and insert in order (fig 14-4) processes.

Put an Image in the pdf that shows that you have tested that your code runs correctly.

Lab.h

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

Contains all shared variables and functions

Main

The main function will test each of the linked list functions

```
#include "lab.h"
int main(){
    NODE* head = nullptr;
        if(loadListUsingInsert("cities",head)){
            displayList(head);
            destroyList(head);}
    else
        {std::cout << "List not loaded successfully" << std::endl;}
        std::cout << "\n";
        head = nullptr;
     if(loadListDirectly("cities",head)){
         displayList(head);
         destroyList(head);}
    else
        {std::cout << "List not loaded successfully" << std::endl;}
        std::cout << "\n";
        head = nullptr;
     if(loadListUsingInsertInOrder("cities",head)){
         displayList(head);
         destroyList(head);}
    else
        {std::cout << "List not loaded successfully" << std::endl;}
}
```

- The list in the Cities file
  - Lucknow
  - Kochi
  - Surat
  - Pune
  - Pune
  - Ahmedabad
  - Hyderabad
  - Kolkata
  - Chennai
  - Bengaluru
  - Mumbai

Load List Using Insert

return true if list is loaded correctly with the cities in the file

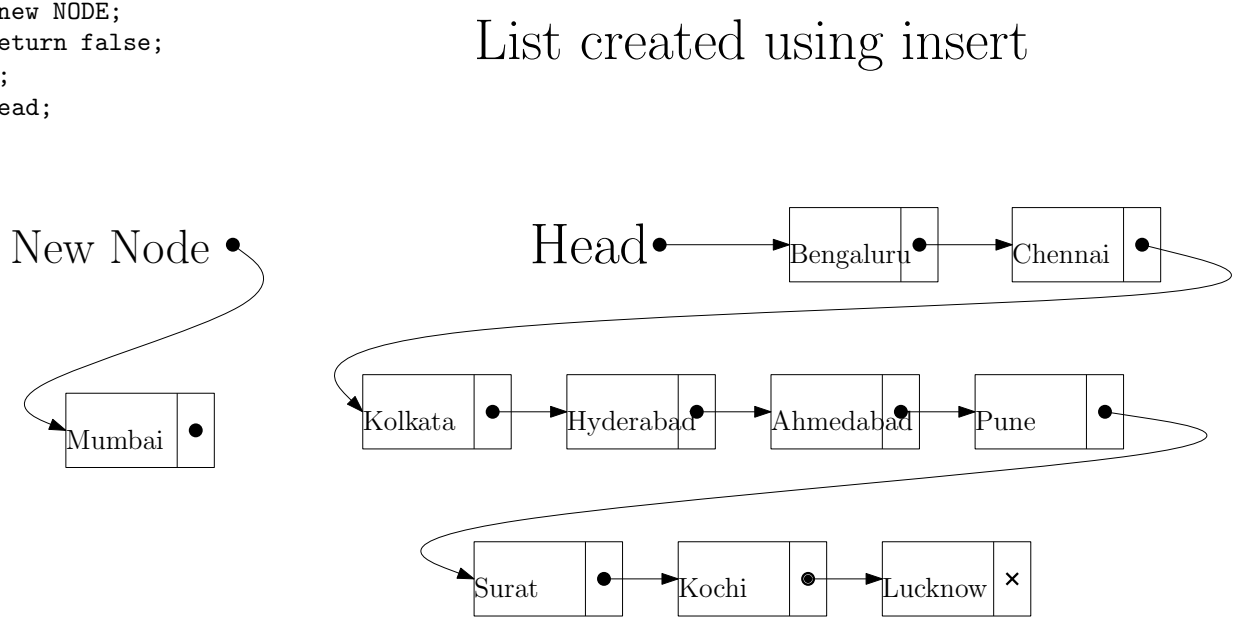
```
#include "lab.h"
bool loadListUsingInsert(std::string f, NODE*& head)
{
    std::ifstream ifs(f); if (not ifs) return false;
    std::string s;
    while(ifs >> s){insert(s,head);}

    //head = new NODE
    return true;
}
```

- read cities file
- save each word in s one-at-a-time

Insert

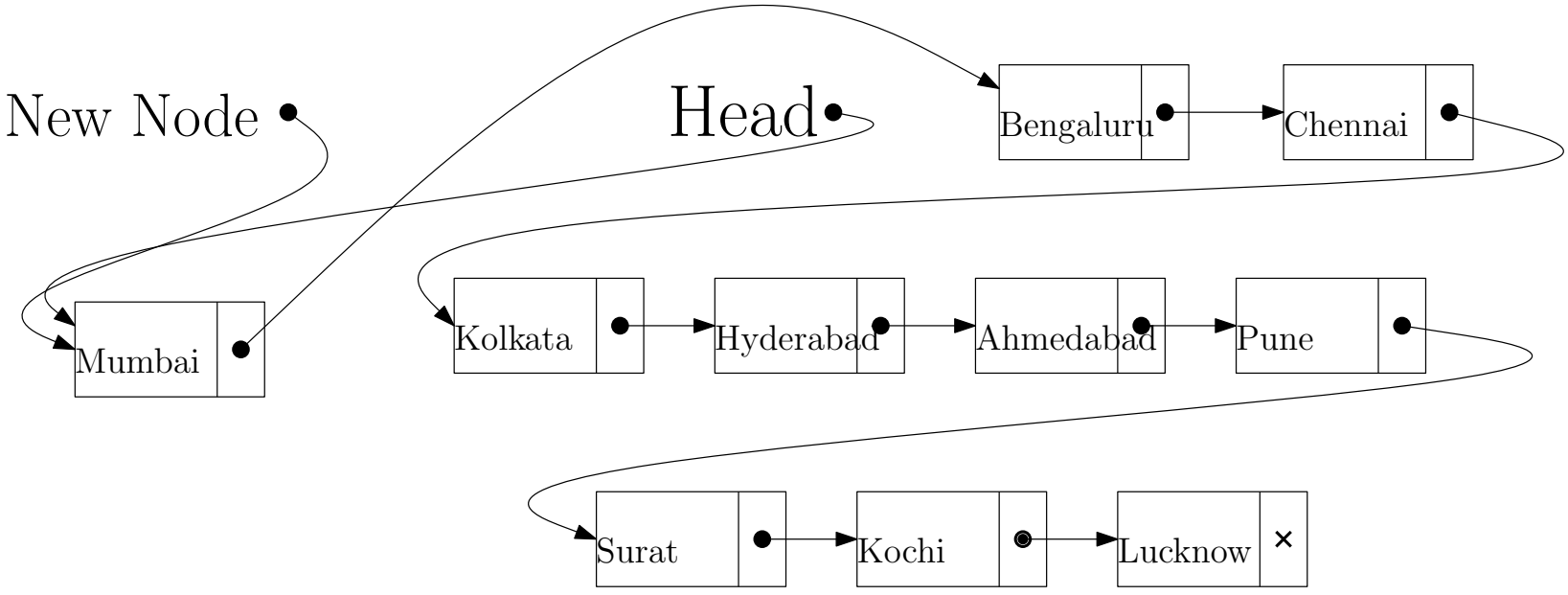
```
#include "lab.h"
bool insert(std::string s, NODE* &head)
{
    NODE * newnode = new NODE;
    if(not newnode) return false;
    newnode->data = s;
    newnode->next = head;
    head = newnode;
    return true;
}
```

This is the linked list before we have added the tenth city



List created using insert

This is the linked list after adding the tenth city



Load List Directly

return true if list is loaded correctly with the cities in the file

```
#include "lab.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);}

    //head = new NODE
    return true;
}
```

Functions the exact same way as previous version

Insert At End

```
#include "lab.h"
bool insertAtEnd(std::string s, NODE* &head)
{
    NODE* tail = nullptr;
    for(tail = head; tail and tail->next; tail = tail->next);
    NODE * newnode = new NODE;
    if(not newnode) return false;
    newnode->data = s;
    newnode->next = nullptr;
    if (not tail) head = newnode;
    else tail->next = newnode;
    return true;
}
```

- point tail at end
- move head to place newnode at end

Load List Using Insert In Order

return true if list is loaded correctly with the cities in the file

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

    //head = new NODE
    return true;
}
```

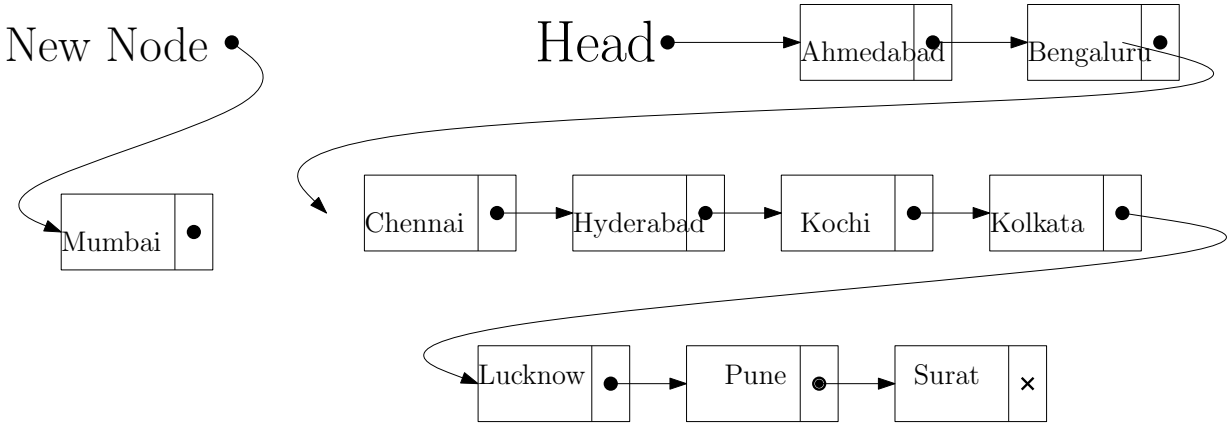
Functions the exact same way as previous version

Insert In Order

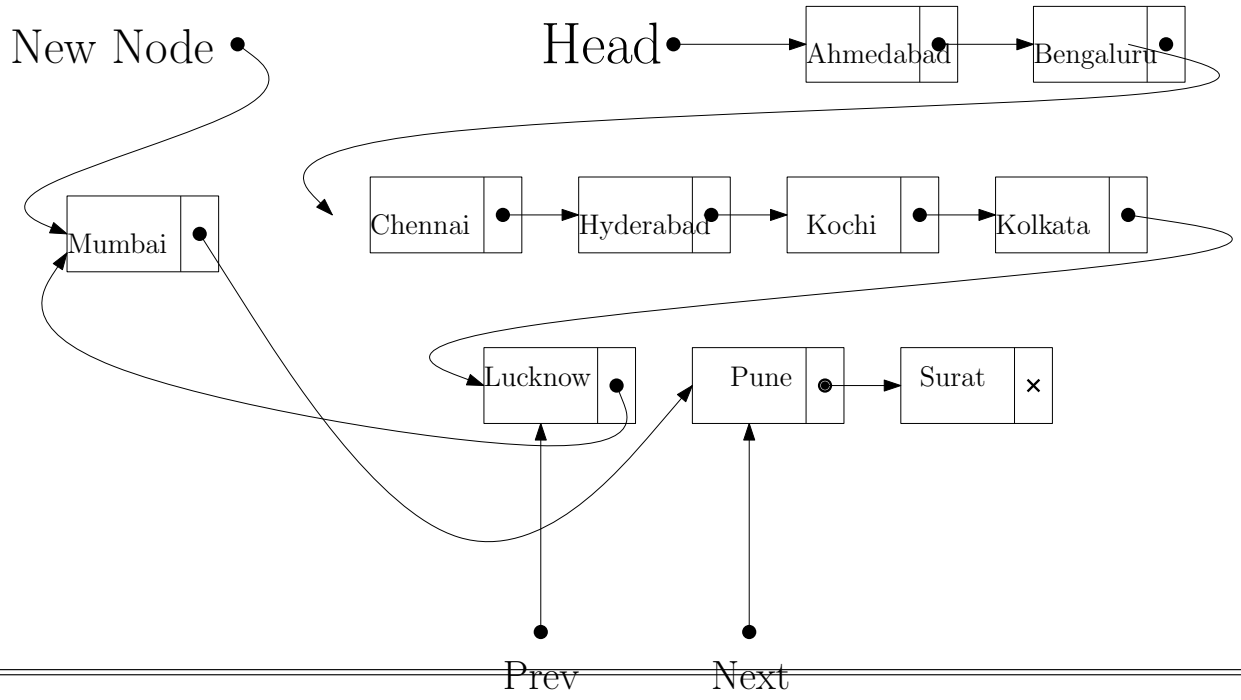
```
#include "lab.h"
bool insertInOrder(std::string s, NODE* &head)
{
    NODE * newnode = new NODE;
    if(not newnode) return false;
    newnode->data = s;
    NODE *node = head, *prev = nullptr;
    while(node && node->data <= s){
        prev = node;
        node = node->next;}
    newnode->next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

}
```

List created using insert in order

Use while loop and comparison to travese list



List created using insert in order



Display List

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

}
```

```
Mumbai
Bengaluru
Chennai
Kolkata
Hyderabad
Ahmedabad
Pune
Surat
Kochi
Lucknow
```

(left) Notice that the cities are in reverse order from the original

```
Lucknow
Kochi
Surat
Pune
Ahmedabad
Hyderabad
Kolkata
Chennai
Bengaluru
Mumbai
```

(left) Notice that the cities are in the same order from the original

```
Ahmedabad
Bengaluru
Chennai
Hyderabad
Kochi
Kolkata
Lucknow
Mumbai
Pune
Surat
```

(left) Notice that the cities are in alphabetical order

Destroy List

return true if list is loaded correctly with the cities in the file

```
#include "lab.h"
void destroyList(NODE* &head){
    NODE* temp;
    while(head && head->next){

        temp = head;
        head = head->next;
        delete temp;
    }
    delete head;
}
```

- makes temp pointer
- temp points to where head points
- head moves up
- temp deletes previous
- loop until head equals null
- delete what head points at

TEST

```
debian@debian:~/cs124/lab2$ ./lab
Mumbai
Bengaluru
Chennai
Kolkata
Hyderabad
Ahmedabad
Pune
Surat
Kochi
Lucknow

Lucknow
Kochi
Surat
Pune
Ahmedabad
Hyderabad
Kolkata
Chennai
Bengaluru
Mumbai

Ahmedabad
Bengaluru
Chennai
Hyderabad
Kochi
Kolkata
Lucknow
Mumbai
Pune
Surat
```

- 1st list uses insert
- 2nd list is made directly
- 3rd list is alphabetized

TEST part 2

```
==2736==
==2736== HEAP SUMMARY:
==2736==     in use at exit: 0 bytes in 0 blocks
==2736==   total heap usage: 96 allocs, 96 frees, 26,874 bytes allocated
==2736==
==2736== All heap blocks were freed -- no leaks are possible
==2736==
==2736== For counts of detected and suppressed errors, rerun with: -v
==2736== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

No memory leaks