Write a program that creates a sorted linked list from a data file. Each line of data should be stored in a class object called **node** that contains a First Name, Last Name, Age, and a pointer to the next node in the list.

Create a container class called **list** to store the nodes in a sorted linked list. As nodes are created, insert them into the list in alphabetical order. The **list** class should clean up memory when it is destroyed. Test your program with the code and text files below.

<https://markbowman.org/231/Program03.zip>

You may not change the Main.cpp or Node.h file.

Node

The **node** class should have the following private data elements:

* First name
* Last name
* Age
* Next - a pointer to another node

The node class should have the following functions:

* Constructor - use default values
* Destructor - deletes the next node
* Put - write last name, first name, and age to an output stream
* Fore - put the current node and any following nodes to an output stream
* Back - put any following nodes and the current node to an output stream
* Insert - given a pointer to a node, use recursion to add it to the list (this function is optional)

The node class should have the following operators:

* **==** compare the node to another node, using Last name, then First name, and return a bool
* **> <**  compare the node to another node, using Last name, then First name, then Age, and return a bool

List

The **list** class should have the following data element:

* Head - a pointer to a node, the start of the list

The **list** class should have the following functions:

* Constructor
* Destructor - deletes all the nodes in the list
* Insert - given a first name, last name, and age, add a node to the list in sorted order
* Find - given a first and last name, locate a node in the list. Return a pointer to the node, or NULL.
* Remove - given a first and last name, delete a node from the list
* Forwards - put the entire list to an output stream, forwards
* Backwards - put the entire list to an output stream, in reverse
* Length - number of nodes in list

What to Hand In

Hand in a copy of the code, and runs to demonstrate that your program and functions work. Sample runs are shown on the next page.

Sample Run – Find in list

Enter file name: ***People.txt***

10 nodes

Enter First and Last name:

***Mark Bowman***

Find: Found!

Bowman Mark 13

Remove: Success!

9 nodes

Forwards List

--------------

Bowman David 45

Bowman Frank 37

Bowman John 30

Bowman Mark 42

Bowman Richard 47

Christensen Ann 70

Cox Susan 36

Gueller Kathleen 34

Morales Carlos 68

Backwards List

--------------

Morales Carlos 68

Gueller Kathleen 34

Cox Susan 36

Christensen Ann 70

Bowman Richard 47

Bowman Mark 42

Bowman John 30

Bowman Frank 37

Bowman David 45

Sample Run – Find at head

Enter file name: ***People.txt***

10 nodes

Enter First and Last name:

***David Bowman***

Find: Found!

Bowman David 45

Remove: Success!

9 nodes

Forwards List

--------------

Bowman Frank 37

Bowman John 30

Bowman Mark 13

Bowman Mark 42

Bowman Richard 47

Christensen Ann 70

Cox Susan 36

Gueller Kathleen 34

Morales Carlos 68

Backwards List

--------------

Morales Carlos 68

Gueller Kathleen 34

Cox Susan 36

Christensen Ann 70

Bowman Richard 47

Bowman Mark 42

Bowman Mark 13

Bowman John 30

Bowman Frank 37

Sample Run – Not found

Enter file name: ***People.txt***

10 nodes

Enter First and Last name:

***Bob Smith***

Find: Not found

Remove: Fail

Forwards List

--------------

Bowman David 45

Bowman Frank 37

Bowman John 30

Bowman Mark 13

Bowman Mark 42

Bowman Richard 47

Christensen Ann 70

Cox Susan 36

Gueller Kathleen 34

Morales Carlos 68

Backwards List

--------------

Morales Carlos 68

Gueller Kathleen 34

Cox Susan 36

Christensen Ann 70

Bowman Richard 47

Bowman Mark 42

Bowman Mark 13

Bowman John 30

Bowman Frank 37

Bowman David 45

Extra Credit

Add another **list** function:

* Find All - given a first and last name, locate all nodes that match in the list. Return an array of pointers to the nodes.

Used the commented out code in Main.cpp to test your function. Use the data file Extra.txt for your test runs. Hand in a copy of the code, and runs to demonstrate that your function works.

**Output – Not Found:**

**Text

Description automatically generated**

**Code:**

**Node.cpp:**

//Implementation of Node class

//Implementation of Node class

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

#include <cctype>

#include "Node.h"

//Constructor

node::node(string argf, string argl, int arga)

{

first = argf;

last = argl;

age = arga;

next = NULL;

}

//Destructor

node::~node()

{

delete this;

}

// Put

void node::put(ostream& out)

{

out << last << " " << first << " " << age << endl;

}

// Output list forwards

void node::fore(ostream& out)

{

out << last << " " << first << " " << age << endl;

//put(out);

}

// Output list backwards

void node::back(ostream& out)

{

out << last << " " << first << " " << age << endl;

//put(out);

}

// Recursive insert

void node::insert(node\* p)

{

//optional

}

// Equal

bool node::operator == (const node& n)

{

return last == n.last && first == n.first && age == n.age;

}

// Less than

bool node::operator < (const node& n)

{

if (last < n.last)

{

return true;

}

else if (first < n.first)

{

return true;

}

else if (age < n.age)

{

return true;

}

return false;

}

// Greater than

bool node::operator > (const node& n)

{

if (last > n.last)

{

return true;

}

else if (first > n.first)

{

return true;

}

else if (age > n.age)

{

return true;

}

return false;

}

**List.cpp:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#include "List.h"

list::list() {

head = NULL;

cnt = 0;

}

void list::insert(string first, string last, int age)

{

node\_ptr newNode = new node();

newNode->first = first;

newNode->last = last;

newNode->age = age;

newNode->next = NULL;

if (head == NULL || head >= newNode)

{

newNode->next = head;

head = newNode;

cnt++;

return;

}

//Locate the node before insertion

struct node\* current = head;

while (current->next != NULL && current->next < newNode)

current = current->next;

newNode->next = current->next;

current->next = newNode;

cnt++;

}

int list::length()

{

return cnt;

}

node\* list::find(string first, string last)

{

node\_ptr curr = head;

while (curr != NULL)

{

if (curr->first == first && curr->last == last)

{

return curr;

}

curr = curr->next;

}

return NULL;

}

bool list::remove(string first, string last)

{

node\* temp = head;

node\* prev = NULL;

if (temp != NULL && temp->first == first && temp->last == last)

{

head = temp->next; // Changed head

delete temp; // free old head

return true;

}

else

{

while (temp != NULL)

{

if (temp->first != first && temp->last != last)

{

prev = temp;

temp = temp->next;

}

}

// If key was not present in linked list

if (temp == NULL)

return false;

// Unlink the node from linked list

prev->next = temp->next;

// Free memory

delete temp;

return true;

}

}

void list::forwards(ostream& out)

{

node\_ptr curr = head;

while (curr != NULL) {

curr->fore(out);

curr = curr->next;

}

}

void list::backwards(ostream& out)

{

struct node\* temp = NULL;

struct node\* prev = NULL;

struct node\* current = head;

while (current != NULL) {

temp = current->next;

current->next = prev;

prev = current;

current = temp;

}

while (prev != NULL) {

while (prev != NULL) {

prev->back(out);

prev = prev->next;

}

}

}

int list::findall(string first, string last, node\_ptr\* arr, int sz)

{

node\_ptr curr = head;

int i = 0;

while (curr != NULL && i < sz) {

if (curr->first == first && curr->last == last) {

arr[i++] = curr;

}

curr = curr->next;

}

return sz;

}

**List.h:**

#include "Node.h"

//class node;

class list

{

private:

node\* head;

int cnt;

public:

list();

void insert(string, string, int);

int length();

node\* find(string, string);

bool remove(string, string);

void forwards(ostream& out);

void backwards(ostream& out);

int findall(string, string, node\_ptr\* arr, int sz);

};

**Node.h:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Node declaration

\* DO NOT CHANGE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//include guards - person.h included multiple times within the file

class list;

class node

{

friend list;

public:

node(string argf = "", string argl = "", int arga = 0); // Default constructor

~node(); // Destructor

void put(ostream& out); // Put

void fore(ostream& out); // Output list forwards

void back(ostream& out); // Output list backwards

void insert(node\* p); // Recursive insert

bool operator == (const node&); // Equal

bool operator < (const node&); // Less than

bool operator > (const node&); // Greater than

private:

string first, last;

int age;

node\* next;

};

typedef node\* node\_ptr;

**Main.h:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Program 03

\* Written by Mark M Bowman

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#include "List.h"

void extra(list&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Main

\* Test function - DO NOT CHANGE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

int age;

list a;

node\* p;

string fname, first, last;

fstream infile;

// Get file name

cout << "Enter file name: ";

cin >> fname;

// Open file

infile.open(fname, ios::in);

// Loop through file

while (!infile.eof())

{

infile >> first >> last >> age;

// Process if okay

if (infile.good())

a.insert(first, last, age);

};

// Close

infile.close();

cout << endl << a.length() << " nodes" << endl;

// Find node

cout << endl;

cout << "Enter First and Last name: ";

cin >> first >> last;

cout << endl << "Find: ";

p = a.find(first, last);

if (p != NULL)

{

cout << "Found!" << endl;

p->put(cout);

}

else

cout << "Not found" << endl;

// Remove from list

cout << endl << "Remove: ";

if (a.remove(first, last))

{

cout << "Success!" << endl;

cout << a.length() << " nodes" << endl;

}

else

cout << "Fail" << endl;

// Display forwards

cout << endl;

cout << "Forwards List\n--------------\n";

a.forwards(cout);

cout << endl;

// Display backwards

cout << "Backwards List\n--------------\n";

a.backwards(cout);

cout << endl;

// Extra credit

// extra(a);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Extra Credit

\* Test function - DO NOT CHANGE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

void extra(list &a)

{ int i,n;

node\_ptr map[4];

string first,last;

// Find node

cout << endl;

cout << "Enter First and Last name: ";

cin >> first >> last;

n = a.findall(first,last,map,4);

// Display forwards

cout << endl;

cout << "Find List\n--------------\n";

for(i=0;i<n;i++)

map[i]->put(cout);

}

\*/

Sample Run – Two matches

Enter file name: ***Extra.txt***

⁞

Enter First and Last name:

***Mark Bowman***

Found List

--------------

Bowman Mark 13

Bowman Mark 42

Sample Run – Four matches

Enter file name: ***Extra.txt***

⁞

Enter First and Last name:

***Jango Fett***

Found List

--------------

Fett Jango 1

Fett Jango 2

Fett Jango 3

Fett Jango 4