1. Download the following code and text from the website and create a workspace:

<https://markbowman.org/231/Lab14.zip>

2. Compile and run the program. Use Names.txt as the input file. You should see a listing of the file.

3. Replace the shaded code with this:

p = new node(name);

p->put(cout);

delete p;

4. Run the program again. You should see a listing of the nodes in triplicate.

5. What do each of the three copies of the node represent?

6. Add the following code to the beginning of the function, before the file is opened:

head = NULL;

7. Add this code at the end of the program, after the file is closed:

if(head!=NULL) delete head;

2. Compile and run your program. You should see each node as it is created in the input loop, then again as the output loop executes.

3. Update the code in node::add() that will use recursion to add the argument to the end of the list.

* If the next pointer is valid

Call next->add(argp)

* Otherwise

Set next = argp

3. Update the code in main() that adds nodes to the list. Replace the lines shown below at the left with those on the right.

|  |  |  |
| --- | --- | --- |
| if(infile.good())  { p = new node(name);  p->put(cout);  if(head==NULL)  head = p;  else  q->add(p);  q = p;  }; | ← | if(infile.good())  { p = new node(name);  p->put(cout);  if(head==NULL)  head = p;  else  head->add(p);  }; |

If head is NULL, then the list is empty and head is set to p. Otherwise, head is used to call the add() function.

4. Compile and run your program. You should see the same output as before.

**Sample Output – Numbers.txt:**

A screenshot of a computer

Description automatically generated with medium confidence

**Sample Output – Names.txt:**

**Text

Description automatically generated**

5. Replace your output loop in the main() function with the following code:

cout << "Forwards" << endl;

if(head!=NULL) head->forwards(cout);

cout << endl;

You will need to add the node::forwards() function to the Node.cpp file. You could do this by copying your loop into the function. Instead, implement the function using recursion. The node destructor is recursive, use that as an example.

6. Remove the call to put() in the input loop.

7. Test your updates again. Your output should display all the nodes in the same order they were added to the list.

8. Add the following code to the main() function, so that the program will also display the list backwards.

cout << "Backwards" << endl;

if(head!=NULL) head->backwards(cout);

cout << endl;

You will need to add the node::backwards() function to the Node.cpp file. The only way this can be done efficiently is with recursion.

9. Test your code. Your output should display the twice, forwards and backwards. Make sure that all the node are displayed in each list.

10. Compile and run the program with Names.txt and Numbers.txt, and save the outputs.

**Names.txt – Output:**

Text

Description automatically generated

**Numbers.txt – Output:**

Text

Description automatically generated

Lab 14 – Main.cpp:

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

\* Lab 14

\* Created by Safford, Twymun

\* Date: 11-Oct-2021

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

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#include "Node.h"

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

\* main()

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

void main()

{ string name;

fstream infile;

node \*head,\*p,\*q;

// Initialize

head = NULL;

// Open file

cout << "Enter file name: ";

cin >> name;

cout << left << endl;

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

// Loop through file

while(!infile.eof())

{ infile >> name;

// Process valid input

if(infile.good())

{ p = new node(name);

p->put(cout);

if (head == NULL)

{

head = p;

}

else

{

//replaced

head->add(p);

//q->add(p);

//q = p;

}

};

};

// Close file

infile.close();

cout << endl;

// Display

p = head;

while(p!=NULL)

{ //p->put(cout);

p = p->next;

};

//cout << endl;

cout << "Forwards" << endl;

if (head != NULL)

{

head->forwards(cout);

}

cout << endl;

//print backwards

cout << "Backwards" << endl;

if (head != NULL)

{

head->backwards(cout);

}

cout << endl;

// Clean up

if (head != NULL)

{

delete head;

}

}

Node.cpp:

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

\* Node.cpp

\* Created by Safford, Twymun

\* Date: 11-Oct-2021

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

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

#include "Node.h"

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

\* Constructor

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

node::node(string args)

{ name = args;

next = NULL;

}

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

\* Destructor

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

node::~node()

{

if (next != NULL)

{

delete next;

}

}

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

\* Implementation-Recursive Add Function

\* Created by Safford, Twymun

\* Date: 11-Oct-2021

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

//now this is the implemented recursive function

void node::add(node \*argp)

{

if (this->next == NULL)

{

this->next = argp;

return;

}

//recursively add

this->next->add(argp);

//next = argp;

}

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

\* Implementation-Node Forwards

\* Created by Safford, Twymun

\* Date: 11-Oct-2021

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

//now this is the implemented for forwards and needs to be recursive

void node::forwards(ostream& out)

{

//call to put

put(out);

//check if the list is empty

if (next != NULL)

{

//store next value and call forwards

next->forwards(out);

}

}

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

\* Implementation-Node Backwards

\* Created by Safford, Twymun

\* Date: 11-Oct-2021

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

//now this is the implemented for backwards and needs to be recursive

void node::backwards(ostream& out)

{

//check if list is non-empty

if (next != NULL)

{

// reverse current order in linked list

next->backwards(out);

}

//call to put for backwards list

put(out);

}

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

\* Output

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

void node::put(ostream &out)

{ out << setw(10) << this;

out << setw(8) << name;

out << setw(10) << next << endl;

}