**Lab Description:**

\*In README File

\*Not on lab write up

**Main.cpp:**

#include <iostream> // cout, endl

/\*

\* Node containing the metadata for a bare linked list

\*/

struct Node {

int value;

Node\* next;

Node();

Node(int);

};

/\*

\* Default Constructor:

\* Initializes the node to default values;

\*/

Node::Node() {

value = 0;

next = nullptr;

};

/\*

\* Non-Default Constructor:

\* Initializes number to what is passed and sets pointers to default values

\*/

Node::Node(int n) {

value = n;

next = nullptr;

};

void print(Node\*); // Recursive print

int length(Node\*); // Recursively computes the length of the list

void reversePrint(Node\*); // Recursive reverse print

Node\* insertSorted(Node\*, int); // Recursively inserts a node in place in sorted position

Node\* remove(Node\*&, int); // Recursively removes one node of the specified value if found in list

Node\* appendList(Node\*, Node\*); // Appends two lists together recursively

Node\* reverse(Node\*); // Recursivesly reverses the pointers in the list

void deleteList(Node\*); // Recursively de-allocates all memory in each list

int main() {

std::cout << "Length Function Tests: " << std::endl; // Length function tests

Node\* head = new Node();

Node\* nextOne = new Node(1);

Node\* secondOne = new Node(2);

head->next = nextOne;

nextOne->next = secondOne;

print(head);

int len = length(head);

std::cout << "The length is: " << len << std::endl;

std::cout << "Print Function Tests: " << std::endl; // Print function tests

print(head);

Node\* head2 = nullptr;

std::cout << "Second list initially set to nullptr!" << std::endl;

print(head2);

head2 = insertSorted(head2, 12);

print(head2);

head2 = insertSorted(insertSorted(insertSorted(head2, 16), 2), -9);

print(head2);

std::cout << "ReversePrint Function Tests: " << std::endl; // Reverse print function tests

std::cout << "Normal list: " << std::endl;

print(head);

std::cout << "Reversed list: " << std::endl;

reversePrint(head);

std::cout << "Normal list 2: " << std::endl;

print(head2);

std::cout << "Reversed list 2: " << std::endl;

reversePrint(head2);

std::cout << "InsertSorted Function Tests: " << std::endl; // Insert sorted function tests

Node\* newList = new Node();

newList = insertSorted(insertSorted(insertSorted(insertSorted(newList, 10), -5), 4), 12);

newList = insertSorted(insertSorted(insertSorted(newList, 10), 10), 13);

print(newList);

std::cout << "Remove Function Tests: " << std::endl; // Remove function tests

std::cout << "Deleting a 10: " << std::endl;

Node\* rtn = remove(newList, 10);

print(newList);

std::cout << "The removed value was " << rtn->value << std::endl;

delete rtn;

std::cout << "Deleting a -5: " << std::endl;

rtn = remove(newList, -5);

print(newList);

std::cout << "The removed value was " << rtn->value << std::endl;

delete rtn;

std::cout << "Deleting a 13: " << std::endl;

rtn = remove(newList, 13);

print(newList);

std::cout << "The removed value was " << rtn->value << std::endl;

delete rtn;

std::cout << "Deleteing a 25: " << std::endl;

rtn = remove(newList, 25);

print(newList);

if (rtn == nullptr) {

std::cout << "The return value was nullptr\n" << std::endl;

}

std::cout << "AppendList Function Tests: " << std::endl; // Append list function tests

std::cout << "Pairing two empty lists: " << std::endl;

Node\* head3 = nullptr;

Node\* head4 = nullptr;

head3 = appendList(head3, head4);

if (head3 == nullptr) {

std::cout << "The list is empty" << std::endl;

}

head3 = nullptr;

head4 = nullptr;

head3 = insertSorted(insertSorted(insertSorted(insertSorted(head3, 1), 0), 4), 3);

head4 = insertSorted(insertSorted(insertSorted(head4, 5), 6), 12);

std::cout << "First list: " << std::endl;

print(head3);

std::cout << "Second list: " << std::endl;

print(head4);

std::cout << "Appended list: " << std::endl;

head3 = appendList(head3, head4);

print(head3);

std::cout << "Reverse Function Tests: " << std::endl; // Reverse function tests

std::cout << "First list: " << std::endl;

print(head3);

std::cout << "Reversed list: " << std::endl;

Node\* reversedList = reverse(head3);

print(reversedList);

// De-allocate all memory in each list

deleteList(head);

deleteList(newList);

deleteList(head2);

deleteList(reversedList);

return 0;

}

/\*

\* deleteList Function:

\* Recursive destructor that de-allocates each node

\*/

void deleteList(Node\* head) {

if (head == nullptr) {

return;

}

if (head->next == nullptr) {

delete head;

return;

}

Node\* destroy = head;

head = head->next;

delete destroy;

deleteList(head);

}

/\*

\* print Function:

\* Recursively prints out each node in succession

\*/

void print(Node\* head) {

if (head) {

std::cout << head->value << " "<< std::endl;

print(head->next);

} else {

std::cout << "\n";

}

}

/\*

\* length Function:

\* Recursively computes the length of the entire list

\*/

int length(Node\* head) {

if (head == nullptr) {

return 0;

}

return length(head->next) + 1;

}

/\*

\* reversePrint Function:

\* Recursively prints the list backwards

\*/

void reversePrint(Node\* head) {

if (head == nullptr) {

return;

}

reversePrint(head->next);

std::cout << head->value << " " << std::endl;

}

/\*

\* insertSorted Function:

\* Recursively inserts a value into a list sorted

\*/

Node\* insertSorted(Node\* head, int val) {

if (head == nullptr) {

Node\* rtnMe = new Node(val);

return rtnMe;

}

if (val < head->value) {

Node\* rtnMe = new Node(val);

rtnMe->next = head;

return rtnMe;

} else if (head->next == nullptr || val < head->next->value) {

Node\* appendMe = new Node(val);

appendMe->next = head->next;

head->next = appendMe;

return head;

}

insertSorted(head->next, val);

return head;

}

/\*

\* remove Function:

\* Recursively removes a specified node from the list

\*/

Node\* remove(Node\* &head, int removeNum) {

if (head == nullptr) {

std::cout << "List is empty" << std::endl;

return head;

}

if (head->next == nullptr && head->value == removeNum) {

Node\* rtnMe = head;

head = nullptr;

return rtnMe;

}

if (head->next == nullptr) {

return nullptr;

}

if (head->value == removeNum) {

Node\* rtnMe = head;

head = head->next;

return rtnMe;

}

if (head->next->value == removeNum) {

Node\* rtnMe = head->next;

head->next = head->next->next;

return rtnMe;

}

return remove(head->next, removeNum);

}

/\*

\* appendList Function:

\* Recursively appends two lists together

\*/

Node\* appendList(Node\* head1, Node\* head2) {

if (head1 == nullptr) {

head1 = head2;

return head1;

}

if (head1->next == nullptr) {

head1->next = head2;

return head1;

}

appendList(head1->next, head2);

return head1;

}

/\*

\* reverse Function:

\* Recursively reverses the entire list

\*/

Node\* reverse(Node\* head) {

if (head == nullptr) {

return nullptr;

}

if (head->next == nullptr) {

return head;

}

Node\* store = reverse(head->next);

head->next->next = head;

head->next = nullptr;

return store;

}

**Sample Output:**

Length Function Tests:

0

1

2

The length is: 3

Print Function Tests:

0

1

2

Second list initially set to nullptr!

12

-9

2

12

16

ReversePrint Function Tests:

Normal list:

0

1

2

Reversed list:

2

1

0

Normal list 2:

-9

2

12

16

Reversed list 2:

16

12

2

-9

InsertSorted Function Tests:

-5

0

4

10

10

10

12

13

Remove Function Tests:

Deleting a 10:

-5

0

4

10

10

12

13

The removed value was 10

Deleting a -5:

0

4

10

10

12

13

The removed value was -5

Deleting a 13:

0

4

10

10

12

The removed value was 13

Deleteing a 25:

0

4

10

10

12

The return value was nullptr

AppendList Function Tests:

Pairing two empty lists:

The list is empty

First list:

0

1

3

4

Second list:

5

6

12

Appended list:

0

1

3

4

5

6

12

Reverse Function Tests:

First list:

0

1

3

4

5

6

12

Reversed list:

12

6

5

4

3

1

0