```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file list.cpp
* @assignment 1: Review; Linked Lists
*/

#include "list.h"

List::List() {}

List::~List() {
    auto deleteListNode = m_head;
    while (m_size > 0) {
        auto nextListNode = deleteListNode->right;
        delete deleteListNode;
        m_size--;
        deleteListNode = nextListNode;
    }
    m_head = nullptr;
    m_tail = nullptr;
}

void List::insertEnd(std::string input) {
    auto newListNode = new ListNode;
    newListNode->data = input;
    if (m_size > 0) {
       auto prevListNode = m_tail;
       prevListNode->right = newListNode;
       newListNode->left = prevListNode;
       m_tail = newListNode;
    }
    else {
        m_tail = newListNode;
        m_head = newListNode;
    }
    m_size++;
}

void List::insertStart(std::string input) {
    auto newListNode = new ListNode;
    newListNode->data = input;
    if (m_size > 0) {
        auto prevListNode = m_head;
        prevListNode->left = newListNode;
        newListNode->right = prevListNode;
        m_head = newListNode;
    }
    else {
        m_tail = newListNode;
        m_head = newListNode;
    }
    m_size++;
}

void List::removeStart() {
    if (m_size >= 1){
        if (m_size == 1) {
            auto deleteListNode = m_head;
            delete deleteListNode;
            m_tail = nullptr;
            m_head = nullptr;
        }
        else {
            auto deleteListNode = m_tail;
```



```
            auto adjustListNode = deleteListNode->right;
            adjustListNode->left = nullptr;
            delete deleteListNode;
            m_head = adjustListNode;
        }
        m_size--;
    }
}

void List::removeEnd() {
    if (m_size >= 1) {
        if (m_size == 1) {
            auto deleteListNode = m_tail;
            delete deleteListNode;
            m_tail = nullptr;
            m_head = nullptr;
        }
        else {
            auto deleteListNode = m_tail;
            auto adjustListNode = deleteListNode->left;
            delete deleteListNode;
            adjustListNode->right = nullptr;
            m_tail = adjustListNode;
        }
        m_size--;
    }
}

std::string List::getData(ListNode* node) {
    if (node != nullptr) {
        return node->data;
    }
    else {
        return "";
    }
}

size_t List::getSize() {
    return m_size;
}

List::ListNode* List::getTail() {
    return m_tail;
}

List::ListNode::ListNode() {}
```



```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file list.h
* @assignment 1: Review; Linked Lists
*/

#include <string>

/*
 * @brief ‘List‘ is a doubly linked list that stores ‘ListNodes‘. Elementary
 * methods are included for removing and adding ‘ListNodes‘ at the beginning
 * and end of the ‘List‘. There is a head, tail, and a size that vary as the
 * container changes. A ‘ListNode‘ cannot be instantiated * or modified
 * outside of a ‘List‘ so a method is provided to accsess the data associated
 * with a ‘ListNode‘ pointer, in case a derived class of ‘List‘ requires that
 * data.
*/
class List {
    // Forward declaration
    struct ListNode;
public:
    // ‘List‘ default constructor
    List();

    // ‘List‘ default destructor
    ~List();

    // Creates a new ‘ListNode‘ with string input attatched to ‘List‘ end
    void insertEnd(std::string input);

    // Creates a new ‘ListNode‘ with string input attatched to ‘List‘ start
    void insertStart(std::string input);

    // Removes ‘ListNode‘ at ‘List‘ start
    void removeStart();

    // Removes ‘ListNode‘ at ‘List‘ end
    void removeEnd();

    // Returns string data associated with a ‘ListNode‘
    std::string getData(ListNode* node);

    // Returns the number of elements in ‘List‘
    size_t getSize();

    // Returns a ‘ListNode‘ pointer to the ‘ListNode‘ at the end of the ‘List‘
    ListNode* getTail();
private:
    ListNode* m_tail = nullptr;
    ListNode* m_head = nullptr;
    size_t m_size = 0;

    /*
    * @brief ‘ListNode‘ is modular component that is used to build up a ‘List‘.
    * As a ‘List‘ follows a doubly linked convention, each ListNode contains
    * pointer variables that link the ‘ListNode‘ to adjacent ‘ListNodes‘. The
    * ‘ListNode‘ also contains a string that can be stored.
    * */
    struct ListNode {
        ListNode();
        ListNode* left = nullptr;
        ListNode* right = nullptr;
        std::string data;
    };
```



```
};
```

```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file main.cpp
* @assignment 1: Review; Linked Lists
*/

/*
* @description This program reads input strings and prints them to standard
* output with a stack data structure that is an extension of a custom
* implemented doubly linked list.
* @status The program compiles. It has been tested using typed standard
* input and redirection with over 100,000 inputs (no memory leaks found).
*/

#include <iostream>
#include "stack.h"

int main() {
    Stack stack;
    std::string buffer;
    while(std::getline(std::cin, buffer)) {
        stack.push(buffer);
    }
    while (!stack.isEmpty()) {
        auto msg = stack.pop();
        std::cout << msg << std::endl;
    }
}
```



```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file stack.cpp
* @assignment 1: Review; Linked Lists
*/

#include "stack.h"

Stack::Stack() {}

void Stack::push(std::string input) {
    m_list.insertEnd(input);
}

std::string Stack::pop() {
   std::string tail = m_list.getData(m_list.getTail());
   m_list.removeEnd();
   return tail;
}

bool Stack::isEmpty() {
    if (m_list.getSize() == 0) {
        return true;
    }
    return false;
}
```

awkward.

.give me something that I don't need/ shouldn't have and then you have to explain it to me.

could just return a string instead.


```
p1: main.o list.o stack.o
   g++ -o p1 main.o list.o stack.o

main.o: main.cpp
   g++ -c main.cpp

list.o: list.h list.cpp
   g++ -c list.cpp

stack.o: stack.h stack.cpp
   g++ -c stack.cpp

clean:
   rm -f p1 test *.o *~

debug: main.cpp stack.cpp list.cpp
   g++ -o p1 -g main.cpp stack.cpp list.cpp
```



```
/*
* @author Cole Van Verth
* @pengo cverth
* @email colevanverth@gmail.com
* @file stack.h
* @assignment 1: Review; Linked Lists
*/

#include "list.h"

/*
 * @brief ‘Stack‘ provides methods for interacting with a container with a stack
 * convention. Although this abstraction is hidden from a ‘Stack‘ user, its
 * container is actually built on a ‘List‘ object, a doubly linked list.
*/

class Stack {
public:
    // ‘Stack‘ default constructor
    Stack();

    // Places a string at the end of the ‘Stack‘ container
    void push(std::string input);

    // Removes and returns string from end of the ‘Stack‘ container
    std::string pop();

    // Returns true if the ‘Stack‘ is empty
    bool isEmpty();
private:
    List m_list;
};
```