# Submission Report

* Submission generated at 09/07/2025 at 22:19:16
* Machine info: Linux pkrvm7jw40e0xgp 6.11.0-1018-azure #18~24.04.1-Ubuntu SMP Sat Jun 28 04:46:03 UTC 2025 x86\_64 x86\_64 x86\_64 GNU/Linux

## Build Output

No main.c found in src. Skipping debug and release builds.  
make[1]: Entering directory '/home/runner/work/P1-Simple-Linked-List/P1-Simple-Linked-List'  
mkdir -p build/tests  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c src/lab.c -o build/tests/lab.c.o  
mkdir -p build/tests/  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c tests/lab-test.c -o build/tests/lab-test.c.o  
mkdir -p build/tests/harness/  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c tests/harness/unity.c -o build/tests/harness/unity.c.o  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage build/tests/lab.c.o build/tests/lab-test.c.o build/tests/harness/unity.c.o -o build/tests/myapp\_t -fprofile-arcs -ftest-coverage  
make[1]: Leaving directory '/home/runner/work/P1-Simple-Linked-List/P1-Simple-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Simple-Linked-List/P1-Simple-Linked-List'  
mkdir -p build/debug-test  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c src/lab.c -o build/debug-test/lab.c.o  
mkdir -p build/debug-test/  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c tests/lab-test.c -o build/debug-test/lab-test.c.o  
mkdir -p build/debug-test/harness/  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c tests/harness/unity.c -o build/debug-test/harness/unity.c.o  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address build/debug-test/lab.c.o build/debug-test/lab-test.c.o build/debug-test/harness/unity.c.o -o build/debug-test/myapp\_td -fsanitize=address  
make[1]: Leaving directory '/home/runner/work/P1-Simple-Linked-List/P1-Simple-Linked-List'  
Test builds completed. You can run the test build with: ./build/tests/myapp\_t  
You can run the debug-test build with: ./build/debug-test/myapp\_td

## Coverage Report

Setting up tests...  
Tearing down tests...  
tests/lab-test.c:167:test\_list\_create:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:168:test\_list\_append:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:169:test\_list\_insert:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:170:test\_list\_insert\_and\_remove:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:171:test\_list\_size\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:172:test\_list\_is\_empty\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:173:test\_list\_get\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:174:test\_insert\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:175:test\_append\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:176:test\_list\_remove\_error:PASS  
  
-----------------------  
10 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:167:test\_list\_create:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:168:test\_list\_append:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:169:test\_list\_insert:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:170:test\_list\_insert\_and\_remove:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:171:test\_list\_size\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:172:test\_list\_is\_empty\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:173:test\_list\_get\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:174:test\_insert\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:175:test\_append\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:176:test\_list\_remove\_error:PASS  
  
-----------------------  
10 Tests 0 Failures 0 Ignored   
OK  
mkdir -p ./build/report/html  
mkdir -p ./build/report/txt  
gcovr -r . --html --html-details --exclude-directories build/tests/harness --exclude '.\*main\.c$' --exclude '.\*test\.c$' -o ./build/report/html/coverage\_report.html  
(INFO) Reading coverage data...  
  
(INFO) Writing coverage report...  
  
gcovr -r . --txt --exclude-directories build/tests/harness --exclude '.\*main\.c$' --exclude '.\*test\.c$'  
(INFO) Reading coverage data...  
  
(INFO) Writing coverage report...  
  
------------------------------------------------------------------------------  
 GCC Code Coverage Report  
Directory: .  
------------------------------------------------------------------------------  
File Lines Exec Cover Missing  
------------------------------------------------------------------------------  
src/lab.c 73 73 100%  
------------------------------------------------------------------------------  
TOTAL 73 73 100%  
------------------------------------------------------------------------------

## Address Sanitizer Report

=================================================================  
==2183==ERROR: LeakSanitizer: detected memory leaks  
  
Direct leak of 24 byte(s) in 1 object(s) allocated from:  
 #0 0x7f9cd1cfd9c7 in malloc ../../../../src/libsanitizer/asan/asan\_malloc\_linux.cpp:69  
 #1 0x558d3ab5f6b8 in list\_append src/lab.c:86  
 #2 0x558d3ab60c96 in test\_append\_error tests/lab-test.c:138  
 #3 0x558d3ab65218 in UnityDefaultTestRun tests/harness/unity.c:2202  
 #4 0x558d3ab61122 in main tests/lab-test.c:175  
 #5 0x7f9cd182a1c9 (/lib/x86\_64-linux-gnu/libc.so.6+0x2a1c9) (BuildId: 282c2c16e7b6600b0b22ea0c99010d2795752b5f)  
 #6 0x7f9cd182a28a in \_\_libc\_start\_main (/lib/x86\_64-linux-gnu/libc.so.6+0x2a28a) (BuildId: 282c2c16e7b6600b0b22ea0c99010d2795752b5f)  
 #7 0x558d3ab5f314 in \_start (/home/runner/work/P1-Simple-Linked-List/P1-Simple-Linked-List/build/debug-test/myapp\_td+0x3314) (BuildId: f4ce82102cfba38b741dab8c98c2b6ff62c7f708)  
  
Direct leak of 24 byte(s) in 1 object(s) allocated from:  
 #0 0x7f9cd1cfd9c7 in malloc ../../../../src/libsanitizer/asan/asan\_malloc\_linux.cpp:69  
 #1 0x558d3ab5f8e2 in list\_insert src/lab.c:106  
 #2 0x558d3ab60a43 in test\_insert\_error tests/lab-test.c:121  
 #3 0x558d3ab65218 in UnityDefaultTestRun tests/harness/unity.c:2202  
 #4 0x558d3ab61104 in main tests/lab-test.c:174  
 #5 0x7f9cd182a1c9 (/lib/x86\_64-linux-gnu/libc.so.6+0x2a1c9) (BuildId: 282c2c16e7b6600b0b22ea0c99010d2795752b5f)  
 #6 0x7f9cd182a28a in \_\_libc\_start\_main (/lib/x86\_64-linux-gnu/libc.so.6+0x2a28a) (BuildId: 282c2c16e7b6600b0b22ea0c99010d2795752b5f)  
 #7 0x558d3ab5f314 in \_start (/home/runner/work/P1-Simple-Linked-List/P1-Simple-Linked-List/build/debug-test/myapp\_td+0x3314) (BuildId: f4ce82102cfba38b741dab8c98c2b6ff62c7f708)  
  
SUMMARY: AddressSanitizer: 48 byte(s) leaked in 2 allocation(s).  
make: \*\*\* [Makefile:117: leak-test] Error 1

## Source File: lab.c

#include "lab.h"  
#include <stdio.h>  
#include <stdlib.h>  
  
/\*\*  
 \* Represents a circular list with a sentinel node.   
 \* AI Use: No AI   
 \*/  
struct List {   
 struct Node\* head; // sentinel node   
 int size; // size of linked list   
};  
  
/\*\*  
 \* Represents a node in the circular linked list.   
 \* AI Use: No AI  
 \*/  
typedef struct Node {  
 void \*data;   
  
 // pointer to next node in list   
 struct Node\* next;   
  
 // pointer to previous node in list   
 struct Node\* prev;   
} Node;  
  
/\*\*  
 \* Creates a new circular linked list with a sentinel node.   
 \* AI Use: Assisted by AI  
 \*/  
List \*list\_create(ListType type) {  
 if (type != LIST\_LINKED\_SENTINEL) { // GCOVR\_EXCL\_START  
 return NULL;   
 } // GCOVR\_EXCL\_STOP  
  
 List \*list = (List \*)malloc(sizeof(List));   
  
 if (list == NULL) { // GCOVR\_EXCL\_START  
 return NULL;   
 } // GCOVR\_EXCL\_STOP  
  
 Node \*sentinel = (Node \*)malloc(sizeof(Node));   
 if (sentinel == NULL) { // GCOVR\_EXCL\_START  
 free(list);   
 return NULL;   
 } // GCOVR\_EXCL\_STOP  
  
 sentinel->data = 0;   
 sentinel->next = sentinel; // circular  
 sentinel->prev = sentinel; // circular  
 list->head = sentinel;   
 list->size = 0;   
  
 return list;   
}  
  
/\*\*  
 \* Destroys the circular linked list and frees all associated memory.  
 \* AI Use: Assisted by AI  
 \*/  
void list\_destroy(List \*list, FreeFunc free\_func) {  
 Node \*curr = list->head->next;  
  
 while (curr != list->head) {  
 Node \*next = curr->next;  
 if (free\_func != NULL && curr->data != NULL) {  
 free\_func(curr->data);  
 free(curr);   
 }  
 curr = next;  
 }  
 free(list->head);  
 free(list);  
}  
  
/\*\*  
 \* Appends a new node with the given data to the end of the circular linked list.  
 \* AI Use: No AI   
 \*/  
bool list\_append(List \*list, void \*data) {  
 if (!list || !data) {   
 return false;   
 }   
  
 Node \*new = (Node \*)malloc(sizeof(Node));   
  
 new->data = data;   
 new->next = list->head; // circular list points back to sentinel node   
 new->prev = list->head->prev; // sentinel's previous node is the curr last node before appending  
 list->head->prev->next = new;   
 list->head->prev = new;   
 list->size++;   
 return true;   
}  
  
/\*\*  
 \* Inserts a new node with the given data at the specified index in the circular linked list.  
 \* AI Use: Assisted by AI   
 \*/  
bool list\_insert(List \*list, size\_t index, void \*data) {  
 if (!list || !data || index > list->size) {   
 return false;   
 }   
  
 Node \*new = (Node \*)malloc(sizeof(Node));   
  
 new->data = data;   
  
 Node \*curr = list->head;  
  
 for (size\_t i = 0; i < index; i++) {  
 curr = curr->next;   
 }  
  
 new->next = curr->next;  
 new->prev = curr;  
 curr->next->prev = new;  
 curr->next = new;  
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* Removes the node at the specified index from the circular linked list and returns its data.  
 \* AI Use: No AI   
 \*/  
void \*list\_remove(List \*list, size\_t index) {  
 if (!list || index >= list->size) {   
 return NULL;   
 }  
  
 Node \*curr = list->head->next;   
 for (size\_t i = 0; i < index; i++) {  
 curr = curr->next;   
 }  
  
 if (!curr) return NULL;  
   
 curr->prev->next = curr->next;   
 curr->next->prev = curr->prev;  
 void \*data = curr->data;  
 free(curr);  
 list->size--;  
 return data;  
}  
  
/\*\*  
 \* Retrieves the data at the specified index in the circular linked list.  
 \* AI Use: No AI   
 \*/  
void \*list\_get(const List \*list, size\_t index) {  
 if (!list || index >= list->size) {   
 return NULL;   
 }   
  
 Node \*curr = list->head->next;   
 for (size\_t i = 0; i < index; i++) {  
 curr = curr->next;   
 }  
 return curr->data;   
}  
  
/\*\*  
 \* Returns the number of elements in the circular linked list.  
 \* AI Use: No AI  
 \*/  
size\_t list\_size(const List \*list) {  
 if (!list) {  
 return 0;   
 }  
 return list->size;  
}  
  
/\*\*  
 \* Checks if the circular linked list is empty.  
 \* AI Use: No AI  
 \*/  
bool list\_is\_empty(const List \*list) {  
 if (!list) {  
 return true;   
 }  
 return list->size == 0;  
}

## Source File: lab.h

#ifndef LAB\_H  
#define LAB\_H  
  
#include <stdbool.h>  
#include <stddef.h>  
  
/\*\*  
 \* @file lab.h  
 \* @brief Header file for a generic list data structure supporting multiple implementations.  
 \*/  
typedef struct List List;  
  
/\*\*  
 \* @enum ListType  
 \* @brief Enumeration for selecting the list implementation type.  
 \*/  
typedef enum {  
 LIST\_LINKED\_SENTINEL  
} ListType;  
  
/\*\*  
 \* @typedef FreeFunc  
 \* @brief Function pointer type for freeing elements. If NULL, no action is taken.  
 \* Must be provided by the user when destroying the list or removing elements.  
 \*  
 \*/  
typedef void (\*FreeFunc)(void \*);  
  
  
/\*\*  
 \* @brief Create a new list of the specified type.  
 \* @param type The type of list to create (e.g., LIST\_LINKED\_SENTINEL).  
 \* @return Pointer to the newly created list, or NULL on failure.  
 \*/  
List \*list\_create(ListType type);  
  
/\*\*  
 \* @brief Destroy the list and free all associated memory.  
 \* @param list Pointer to the list to destroy.  
 \* @param free\_func Function to free individual elements. If NULL, elements are not freed.  
 \*/  
void list\_destroy(List \*list, FreeFunc free\_func);  
  
/\*\*  
 \* @brief Append an element to the end of the list.  
 \* @param list Pointer to the list.  
 \* @param data Pointer to the data to append.  
 \* @return true on success, false on failure.  
 \*/  
bool list\_append(List \*list, void \*data);  
  
/\*\*  
 \* @brief Insert an element at a specific index.  
 \* @param list Pointer to the list.  
 \* @param index Index at which to insert the element.  
 \* @param data Pointer to the data to insert.  
 \* @return true on success, false on failure (e.g., index out of bounds).  
 \*/  
bool list\_insert(List \*list, size\_t index, void \*data);  
  
/\*\*  
 \* @brief Remove an element at a specific index.  
 \* @param list Pointer to the list.  
 \* @param index Index of the element to remove.  
 \* @return Pointer to the element, or NULL if index is out of bounds.  
 \*/  
void \*list\_remove(List \*list, size\_t index);  
  
/\*\*  
 \* @brief Get a pointer the element at a specific index.  
 \* @param list Pointer to the list.  
 \* @param index Index of the element to retrieve.  
 \* @return Pointer to the element, or NULL if index is out of bounds.  
 \*/  
void \*list\_get(const List \*list, size\_t index);  
  
/\*\*  
 \* @brief Get the current size of the list.  
 \* @param list Pointer to the list.  
 \* @return The number of elements in the list.  
 \*/  
size\_t list\_size(const List \*list);  
  
/\*\*  
 \* @brief Check if the list is empty.  
 \* @param list Pointer to the list.  
 \* @return true if the list is empty, false otherwise.  
 \*/  
bool list\_is\_empty(const List \*list);  
  
#endif // LAB\_H

## Test Files

### lab-test.c

#include <stdlib.h>  
#include <stdio.h>  
#include "harness/unity.h"  
#include "../src/lab.h"  
  
/\*\*  
 \* TODO:   
 \* write unit tests to ensure list works, minimum of four?   
 \*/  
  
void setUp(void) {  
 printf("Setting up tests...\n");  
}  
  
void tearDown(void) {  
 printf("Tearing down tests...\n");  
}  
  
void test\_list\_create(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
 TEST\_ASSERT\_EQUAL(0, list\_size(list));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 list\_destroy(list, free);  
}  
  
void test\_list\_append(void) {   
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*data1 = malloc(sizeof(int));  
 int \*data2 = malloc(sizeof(int));  
 \*data1 = 10;  
 \*data2 = 20;  
  
 TEST\_ASSERT\_TRUE(list\_append(list, data1));  
 TEST\_ASSERT\_TRUE(list\_append(list, data2));  
 TEST\_ASSERT\_EQUAL(2, list\_size(list));  
 TEST\_ASSERT\_EQUAL\_PTR(data1, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(data2, list\_get(list, 1));  
 TEST\_ASSERT\_FALSE(list\_is\_empty(list));  
  
 list\_destroy(list, free);  
}  
  
void test\_list\_insert(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*data1 = malloc(sizeof(int));  
 int \*data2 = malloc(sizeof(int));  
 int \*data3 = malloc(sizeof(int));   
 \*data1 = 5;  
 \*data2 = 10;  
 \*data3 = 15;   
  
 // Insert first element at index 0  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, data1));  
 TEST\_ASSERT\_EQUAL\_PTR(data1, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
  
 // Insert second element at index 1 (end)  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, data2));  
 TEST\_ASSERT\_EQUAL\_PTR(data2, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL(2, list\_size(list));  
  
 // Insert third element in the middle   
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, data3));  
 TEST\_ASSERT\_EQUAL\_PTR(data3, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL\_PTR(data2, list\_get(list, 2));  
 TEST\_ASSERT\_EQUAL(3, list\_size(list));  
  
 list\_destroy(list, free);  
}  
  
void test\_list\_insert\_and\_remove(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*data1 = malloc(sizeof(int));  
 int \*data2 = malloc(sizeof(int));  
 int \*data3 = malloc(sizeof(int));  
 \*data1 = 10;  
 \*data2 = 20;  
 \*data3 = 15;  
  
 TEST\_ASSERT\_TRUE(list\_append(list, data1));  
 TEST\_ASSERT\_TRUE(list\_append(list, data2));  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, data3)); // Insert in the middle  
  
 TEST\_ASSERT\_EQUAL(3, list\_size(list));  
 TEST\_ASSERT\_EQUAL\_PTR(data1, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(data3, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL\_PTR(data2, list\_get(list, 2));  
  
 int \*removed\_data = (int \*)list\_remove(list, 1); // Remove the middle element  
 TEST\_ASSERT\_EQUAL\_PTR(data3, removed\_data);  
 free(removed\_data);  
  
 TEST\_ASSERT\_EQUAL(2, list\_size(list));  
 TEST\_ASSERT\_EQUAL\_PTR(data1, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(data2, list\_get(list, 1));  
  
 list\_destroy(list, free);  
}  
  
void test\_list\_size\_null(void) {  
 TEST\_ASSERT\_EQUAL(0, list\_size(NULL));  
}  
  
void test\_list\_is\_empty\_null(void) {  
 TEST\_ASSERT\_TRUE(list\_is\_empty(NULL));  
}  
  
void test\_list\_get\_error(void) {  
 TEST\_ASSERT\_NULL(list\_get(NULL, 0));   
 TEST\_ASSERT\_NULL(list\_get(NULL, 1000));   
}  
  
void test\_insert\_error(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
 int value = 5;  
 TEST\_ASSERT\_FALSE(list\_insert(NULL, 0, &value)); // list is NULL  
 TEST\_ASSERT\_FALSE(list\_insert(list, 0, NULL)); // data is NULL  
 TEST\_ASSERT\_FALSE(list\_insert(list, 999, &value)); // index too large  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, &value));  
 TEST\_ASSERT\_EQUAL\_PTR(&value, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
  
 list\_destroy(list, NULL);  
}  
  
void test\_append\_error(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
  
 int value = 10;  
  
 TEST\_ASSERT\_FALSE(list\_append(NULL, &value));  
  
 TEST\_ASSERT\_FALSE(list\_append(list, NULL));  
  
 TEST\_ASSERT\_TRUE(list\_append(list, &value));  
 TEST\_ASSERT\_EQUAL\_PTR(&value, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
  
 list\_destroy(list, NULL);  
}  
  
void test\_list\_remove\_error(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
  
 int value = 42;  
  
 TEST\_ASSERT\_NULL(list\_remove(NULL, 0));  
  
 TEST\_ASSERT\_NULL(list\_remove(list, 999));  
  
 TEST\_ASSERT\_TRUE(list\_append(list, &value));  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
  
 void \*removed = list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL\_PTR(&value, removed);  
 TEST\_ASSERT\_EQUAL(0, list\_size(list));  
  
 list\_destroy(list, NULL);  
}  
  
int main(void) {  
 UNITY\_BEGIN();  
 RUN\_TEST(test\_list\_create);  
 RUN\_TEST(test\_list\_append);  
 RUN\_TEST(test\_list\_insert);  
 RUN\_TEST(test\_list\_insert\_and\_remove);  
 RUN\_TEST(test\_list\_size\_null);  
 RUN\_TEST(test\_list\_is\_empty\_null);  
 RUN\_TEST(test\_list\_get\_error);  
 RUN\_TEST(test\_insert\_error);  
 RUN\_TEST(test\_append\_error);  
 RUN\_TEST(test\_list\_remove\_error);  
 return UNITY\_END();  
}

## README

# Project 1 - Simple Linked List   
  
- Name: Emily Newton  
- Email: emilynewton@u.boisestate.edu  
- Class: 452-002  
  
## Known Bugs or Issues  
  
TODO: No known bugs or issues!   
  
## Experience  
  
TODO: I feel like I was able to get a better understanding of the code coverage. In P0, it was like   
okay great it just sees how many lines execute but with this project I was able to understand which ones   
are infeasible in the sense that they are much harder to test. At first, I wanted to put a skip in every instance   
of returning null when really I could make tests and just assert them to null. The only places that did matter to skip were   
places where I attempt to allocate memory to a place that's null. The hardest part about this project was just getting   
back into understanding C. I haven't coded in C really since CS 253 which was a couple years ago so this was a good wake up   
call. I'm glad it was linked lists though because I do feel somewhat comfortable with that concept and the setinel node made   
some sense to me. I had trouble with segmentation faults just because I was forgetting null checks or not using the free function   
properly which is something I'll need more practice with. Honestly though, it was nice to have null checks actually matter   
because I get a little lazy in Java and I feel like the stakes are higher with that kind of stuff in C.

## End of Report

Report generated on 09/07/2025 at 22:19:17

## GitHub Info

* GitHub repo name: emilynewton/P1-Simple-Linked-List
* The repository visibility is private.
* The workflow was triggered by emilynewton

Hash is committed to repo as submission-report-hash.txt

0d9be9303ff6cc0ae81b5c1fd6cafd6a781942c0bc705a6d9942e4e995bb78da submission-report.md 94d105ef865785ff7c79933a1a8d5ad366ccba08932f55bac5ea5952ba666fc8 submission-report.md