# Submission Report

* Submission generated at 09/06/2025 at 00:55:06
* 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/cs452\_p1/cs452\_p1'  
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/cs452\_p1/cs452\_p1'  
make[1]: Entering directory '/home/runner/work/cs452\_p1/cs452\_p1'  
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/cs452\_p1/cs452\_p1'  
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:311:test\_list\_create:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:312:test\_list\_destroy:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:313:test\_append:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:314:test\_insert:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:315:test\_insert\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:316:test\_get\_element:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:317:test\_get\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:318:test\_remove:PASS  
  
Setting up tests...  
  
Removed: (nil)  
  
Tearing down tests...  
tests/lab-test.c:319:test\_remove\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:320:test\_list\_is\_empty:PASS  
  
-----------------------  
10 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:311:test\_list\_create:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:312:test\_list\_destroy:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:313:test\_append:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:314:test\_insert:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:315:test\_insert\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:316:test\_get\_element:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:317:test\_get\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:318:test\_remove:PASS  
  
Setting up tests...  
  
Removed: (nil)  
  
Tearing down tests...  
tests/lab-test.c:319:test\_remove\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:320:test\_list\_is\_empty: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 107 107 100%  
------------------------------------------------------------------------------  
TOTAL 107 107 100%  
------------------------------------------------------------------------------

## Address Sanitizer Report

Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:311:test\_list\_create:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:312:test\_list\_destroy:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:313:test\_append:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:314:test\_insert:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:315:test\_insert\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:316:test\_get\_element:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:317:test\_get\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:318:test\_remove:PASS  
  
Setting up tests...  
  
Removed: (nil)  
  
Tearing down tests...  
tests/lab-test.c:319:test\_remove\_out\_of\_bounds:PASS  
  
Setting up tests...  
  
Tearing down tests...  
tests/lab-test.c:320:test\_list\_is\_empty:PASS  
  
-----------------------  
10 Tests 0 Failures 0 Ignored   
OK

## Source File: lab.c

#include "lab.h"  
#include <stdio.h>  
#include <stdlib.h>  
/\*  
 \* =====  
 \* TYPES  
 \* =====  
 \*/  
  
/\*\*  
 \* @enum NodeType  
 \* @brief Enumeration for selecting Node type  
 \*  
 \* AI Use: Assisted by AI  
 \* Needed a way to check (void\* data) parameter is a node  
 \*/  
typedef enum { NODE, SENTINEL } NodeType;  
  
/\*\*  
 \* @struct Node  
 \* @brief a Node struct that is able to point to other nodes both ways  
 \*/  
typedef struct Node {  
 NodeType type;  
 struct Node \*next, \*prev;  
} Node;  
  
/\*\*  
 \* @struct SentinelLinkedList  
 \* @brief SentinelLinkedList struct that is one implementation for a List struct  
 \*/  
typedef struct SentinelLinkedList {  
 Node \*head, \*tail;  
 size\_t size;  
} SentinelLinkedList;  
  
typedef struct List {  
 ListType type;  
  
 // AI Use: Assisted by AI  
 // I needed a form of inheritence to keep this type generic  
 union {  
 struct SentinelLinkedList \*sentinel\_list;  
 } lists;  
} List;  
  
/\*  
 \* ================  
 \* HELPER FUNCTIONS  
 \* ================  
 \*/  
  
/\*\*  
 \* @brief frees a passed in pointer to a node  
 \* @param Node object pointer  
 \*/  
void free\_node(void \*data\_ptr) { // GCOVR\_EXCL\_START  
 Node \*node = (Node \*)data\_ptr;  
 free(node);  
} // GCOVR\_EXCL\_STOP  
  
bool index\_in\_bounds(size\_t size, size\_t index) {  
 return (index < size && index >= 0);  
}  
  
size\_t sentinel\_list\_size(SentinelLinkedList \*sentinel\_list) {  
 return sentinel\_list->size;  
}  
  
void \*sentinel\_list\_get(SentinelLinkedList \*sentinel\_list, size\_t index) {  
  
 // Start at tail or head based on which is closest  
 // (unnecessary for small sets of data)  
 bool nodeIsCloseToTail = index > (sentinel\_list->size / 2); // index > middle  
 int currIdx = (nodeIsCloseToTail) ? sentinel\_list->size - 1  
 : -1; // (-1) accounts for sentinel  
 Node \*currNode =  
 (nodeIsCloseToTail) ? sentinel\_list->tail : sentinel\_list->head;  
 Node \*nextNode = currNode->next;  
  
 // Find node at given index  
 // (List is empty || index within bounds)  
 while (currIdx != index) {  
 // ? shift backward : shift forward  
 currNode = nextNode;  
 currIdx += (nodeIsCloseToTail) ? -1 : 1;  
 nextNode = currNode->next;  
  
 // Index bounds check  
 if (!index\_in\_bounds(sentinel\_list\_size(sentinel\_list), currIdx))  
 break;  
 }  
  
 // Check index was found  
 if (currIdx != index) { // GCOVR\_EXCL\_START  
 return NULL;  
 } // GCOVR\_EXCL\_STOP  
  
 return currNode;  
}  
  
/\*\*  
 \* @brief Create a new list of the specified type.  
 \* @return Pointer to the newly created list, or NULL on failure.  
 \*/  
List \*sentinel\_list\_create(void) {  
 List \*list = malloc(sizeof(List));  
 list->type = LIST\_LINKED\_SENTINEL;  
  
 // Creates SentinelList Pointer  
 // NOTE: allocates memory separately to optimize sizeof List  
 // (avoid unnecessary allocation to unused implementations in the future)  
 list->lists.sentinel\_list = malloc(sizeof(SentinelLinkedList));  
  
 // Sentinel node will always be the head -- we want tail on the first appended  
 // element later  
 Node \*sentinelNode = malloc(sizeof(Node));  
 sentinelNode->type = SENTINEL;  
 // Sets all meta data pointers to sentinel node initially  
 list->lists.sentinel\_list->head = list->lists.sentinel\_list->tail =  
 sentinelNode->next = sentinelNode->prev = sentinelNode;  
  
 // Sentinel node should not count toward size  
 list->lists.sentinel\_list->size = 0;  
  
 return list;  
}  
  
/\*\*  
 \* @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 sentinel\_list\_destroy(List \*list, FreeFunc free\_func) {  
 SentinelLinkedList \*sentinel\_list = list->lists.sentinel\_list;  
  
 // User must pass (non-null) FreeFunc to destroy elements  
 // NOTE: If skipped, manually cleanup nodes in the list later  
 if (free\_func) {  
 Node \*sentinelNode = sentinel\_list->head;  
 Node \*currNode = sentinelNode;  
 Node \*nextNode = currNode->next;  
  
 while (currNode != NULL) { // GCOVR\_EXCL\_START  
 free\_func(currNode); // GCOVR\_EXCL\_STOP  
 // Reduce size  
 sentinel\_list->size = list\_size(list) - 1;  
 // Update currNode until loops back to sentinel node  
 currNode = (nextNode == sentinelNode) ? NULL : nextNode;  
 // Update next node  
 nextNode = (currNode) ? currNode->next : NULL;  
 }  
 }  
  
 // Finally cleanup lists  
 free(sentinel\_list); // GCOVR\_EXCL\_START  
 sentinel\_list = NULL;  
 free(list);  
 list = NULL; // GCOVR\_EXCL\_STOP  
}  
  
/\*\*  
 \* @brief Append an element to the end of the list.  
 \* @param sentinel\_list Pointer to the sentinel list.  
 \* @param newTail Pointer to the Node struct to append.  
 \* @return true on success, false on failure.  
 \*/  
bool sentinel\_list\_append(SentinelLinkedList \*sentinel\_list, Node \*newTail) {  
 Node \*currTail = sentinel\_list->tail;  
  
 // Previous Tail <---> New Tail  
 currTail->next = newTail;  
 newTail->prev = currTail;  
  
 // Sentinel <---> New Tail  
 newTail->next = sentinel\_list->head;  
 sentinel\_list->head->prev = newTail;  
  
 // Update tail and list data  
 sentinel\_list->tail = newTail;  
 sentinel\_list->size += 1;  
  
 return newTail == sentinel\_list->tail;  
}  
  
/\*\*  
 \* @brief Insert an element at a specific index.  
 \* @param sentinel\_list Pointer to the sentinel list.  
 \* @param index Index at which to insert the element.  
 \* @param newNode Pointer to the Node struct to insert.  
 \* @return true on success, false on failure (e.g., index out of bounds).  
 \*/  
bool sentinel\_list\_insert(SentinelLinkedList \*sentinel\_list, size\_t index,  
 Node \*newNode) {  
 // Index is out of bounds  
 // GCOVR\_EXCL\_START  
 if (index < 0 || index > sentinel\_list\_size(sentinel\_list))  
 return false;  
 // GCOVR\_EXCL\_STOP  
 //  
 Node \*nodeAtGivenIndex = sentinel\_list\_get(sentinel\_list, index);  
 // Node was not found  
 // GCOVR\_EXCL\_START  
 if (!nodeAtGivenIndex)  
 return false;  
 // GCOVR\_EXCL\_STOP  
  
 Node \*oldPrev = nodeAtGivenIndex->prev;  
 // Found Node Prev <-> New Node  
 newNode->prev = oldPrev;  
 oldPrev->next = newNode;  
 // New Node <-> Found Node  
 newNode->next = nodeAtGivenIndex;  
 nodeAtGivenIndex->prev = newNode;  
  
 // Update list data as needed  
 sentinel\_list->tail =  
 (sentinel\_list\_size(sentinel\_list) == 0) ? newNode : sentinel\_list->tail;  
 sentinel\_list->size += 1;  
  
 // GCOVR\_EXCL\_START  
 return true;  
 // GCOVR\_EXCL\_STOP  
}  
  
/\*\*  
 \* @brief Remove an element at a specific index.  
 \* @param sentinel\_list Pointer to the sentinel list.  
 \* @param index Index of the element to remove.  
 \* @return Pointer to the element, or NULL if index is out of bounds.  
 \*/  
void \*sentinel\_list\_remove(SentinelLinkedList \*sentinel\_list, size\_t index) {  
 // Index is out of bounds  
 // GCOVR\_EXCL\_START  
 if (!index\_in\_bounds(sentinel\_list\_size(sentinel\_list), index)) {  
 return NULL;  
 } // GCOVR\_EXCL\_STOP  
  
 Node \*nodeAtGivenIndex = sentinel\_list\_get(sentinel\_list, index);  
 // Node was not found  
 if (!nodeAtGivenIndex) { // GCOVR\_EXCL\_START  
 return NULL;  
 } // GCOVR\_EXCL\_STOP  
  
 // "Remove" node at given index  
 Node \*prevOfFoundNode = nodeAtGivenIndex->prev;  
 Node \*nextOfFoundNode = nodeAtGivenIndex->next;  
 // Prev of Node to Remove <-> Next of Node to Remove  
 prevOfFoundNode->next = nextOfFoundNode;  
 nextOfFoundNode->prev = prevOfFoundNode;  
  
 // Update list data as needed  
 sentinel\_list->size -= 1;  
 if (nodeAtGivenIndex == sentinel\_list->tail)  
 sentinel\_list->tail = prevOfFoundNode;  
  
 // NOTE: Function returns pointer, so maybe don't clean  
 // (avoids dangling pointer)  
 return nodeAtGivenIndex;  
}  
  
/\*  
 \* =================  
 \* PRIMARY FUNCTIONS  
 \* =================  
 \*/  
  
List \*list\_create(ListType type) {  
 List \*list = NULL; // could remain null  
  
 // List may have multiple implementations -- assume find by type  
 switch (type) {  
 case LIST\_LINKED\_SENTINEL:  
 list = sentinel\_list\_create();  
 break;  
 }  
  
 return list;  
}  
  
void list\_destroy(List \*list, FreeFunc free\_func) {  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 sentinel\_list\_destroy(list, free\_func);  
 break;  
 }  
  
 // AI Use: Assisted by AI  
 // Used to find how to skip brackets marked as "untested" line  
 // NOTE: tests for brackets with no code behind it  
  
} // GCOVR\_EXCL\_LINE  
  
bool list\_append(List \*list, void \*data) {  
 Node \*dataNode = data;  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 // Check a node was passed in as data  
 // GCOVR\_EXCL\_START  
 if (dataNode->type != SENTINEL && dataNode->type != NODE)  
 return false;  
 // GCOVR\_EXCL\_STOP  
 return sentinel\_list\_append(list->lists.sentinel\_list, data);  
 }  
} // GCOVR\_EXCL\_LINE  
  
bool list\_insert(List \*list, size\_t index, void \*data) {  
 Node \*dataNode = data;  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 // Check a node was passed in as data  
 // GCOVR\_EXCL\_START  
 if (dataNode->type != SENTINEL && dataNode->type != NODE)  
 return false;  
 // GCOVR\_EXCL\_STOP  
  
 // Insert node into list  
 return sentinel\_list\_insert(list->lists.sentinel\_list, index, dataNode);  
 }  
} // GCOVR\_EXCL\_LINE  
  
void \*list\_remove(List \*list, size\_t index) {  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 return sentinel\_list\_remove(list->lists.sentinel\_list, index);  
 }  
} // GCOVR\_EXCL\_LINE  
  
void \*list\_get(const List \*list, size\_t index) {  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 return sentinel\_list\_get(list->lists.sentinel\_list, index);  
 }  
} // GCOVR\_EXCL\_LINE  
  
size\_t list\_size(const List \*list) {  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 return sentinel\_list\_size(list->lists.sentinel\_list);  
 }  
} // GCOVR\_EXCL\_LINE  
  
bool list\_is\_empty(const List \*list) {  
 switch (list->type) {  
 case LIST\_LINKED\_SENTINEL:  
 return sentinel\_list\_size(list->lists.sentinel\_list) == 0;  
 }  
} // GCOVR\_EXCL\_LINE

## 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 "../src/lab.h"  
#include "harness/unity.h"  
#include "harness/unity\_internals.h"  
#include <stdio.h>  
#include <stdlib.h>  
  
// AI Use: Assisted by AI  
// Needed a way to check (void\* data) parameter is a node  
typedef enum { NODE, SENTINEL } NodeType;  
  
typedef struct Node {  
 NodeType type;  
 struct Node \*next, \*prev;  
} Node;  
  
typedef struct SentinelLinkedList {  
 Node \*head, \*tail;  
 size\_t size;  
} SentinelLinkedList;  
  
typedef struct List {  
 ListType type;  
  
 // AI Use: Assisted by AI  
 // I needed a form of inheritence to keep this type generic  
 union {  
 struct SentinelLinkedList \*sentinel\_list;  
 } lists;  
} List;  
  
int test\_element = 4; // int simply for testing  
  
extern void free\_node(void \*data\_ptr);  
  
void setUp(void) { printf("\nSetting up tests...\n"); }  
  
void tearDown(void) { printf("\nTearing down tests...\n"); }  
  
void test\_list\_create(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 SentinelLinkedList \*sentinel\_list\_ptr = list->lists.sentinel\_list;  
  
 TEST\_ASSERT\_NOT\_NULL(list);  
 TEST\_ASSERT\_NOT\_NULL(sentinel\_list\_ptr);  
 TEST\_ASSERT\_NOT\_NULL(sentinel\_list\_ptr->head);  
 TEST\_ASSERT\_NOT\_NULL(sentinel\_list\_ptr->tail);  
 TEST\_ASSERT\_EQUAL(sentinel\_list\_ptr->head, sentinel\_list\_ptr->tail);  
  
 // Free manually to avoid premature stacktraces (tests destroy separately)  
 free(list->lists.sentinel\_list->head); // frees sentinel  
 free(list->lists.sentinel\_list);  
 free(list);  
 sentinel\_list\_ptr = NULL;  
 list = NULL;  
}  
  
void test\_list\_destroy(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 list\_destroy(list, free\_node);  
 list = NULL;  
  
 TEST\_ASSERT\_NULL(list);  
}  
  
void test\_append(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
  
 SentinelLinkedList \*sentinel\_list = list->lists.sentinel\_list;  
 Node \*newNode = malloc(sizeof(Node));  
 Node \*newNode2 = malloc(sizeof(Node));  
 newNode->type = newNode2->type = NODE;  
  
 // Append an item  
 bool item\_appended = list\_append(list, newNode);  
 TEST\_ASSERT\_TRUE(item\_appended);  
 // Check node addresses are different  
 TEST\_ASSERT\_NOT\_EQUAL(sentinel\_list->head, sentinel\_list->tail);  
 // Sentinel -> New node  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->next, sentinel\_list->tail);  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->prev, sentinel\_list->tail);  
 // New node -> Sentinel  
 TEST\_ASSERT\_EQUAL(sentinel\_list->tail->prev, sentinel\_list->head);  
 TEST\_ASSERT\_EQUAL(sentinel\_list->tail->next, sentinel\_list->head);  
 TEST\_ASSERT\_EQUAL(1, list->lists.sentinel\_list->size);  
  
 // Append second item (testing shifting and pointers)  
 bool item\_appended2 = list\_append(list, newNode2);  
 TEST\_ASSERT\_TRUE(item\_appended2);  
 // Sentinel -> New node  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->prev, sentinel\_list->tail);  
 // New node -> Sentinel  
 TEST\_ASSERT\_EQUAL(sentinel\_list->tail->next, sentinel\_list->head);  
 // Sentinel !(->) New Node  
 TEST\_ASSERT\_NOT\_EQUAL(sentinel\_list->head->next, sentinel\_list->tail);  
 // New Node !(->) Sentinel  
 TEST\_ASSERT\_NOT\_EQUAL(sentinel\_list->tail->prev, sentinel\_list->head);  
 TEST\_ASSERT\_EQUAL(2, list->lists.sentinel\_list->size);  
  
 // Append a non-node item  
 bool non\_node\_appended = list\_append(list, &test\_element);  
 TEST\_ASSERT\_FALSE(non\_node\_appended);  
 TEST\_ASSERT\_EQUAL(2, list->lists.sentinel\_list->size);  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 list = NULL;  
 sentinel\_list = NULL;  
 newNode = NULL;  
}  
  
void test\_insert(void) {  
 int test\_element\_2 = 2;  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 SentinelLinkedList \*sentinel\_list = list->lists.sentinel\_list;  
 Node \*node1 = malloc(sizeof(Node));  
 node1->type = NODE;  
  
 // Test insert stores the node at the given index  
 bool insertedElement = list\_insert(list, 0, node1);  
 TEST\_ASSERT\_TRUE(insertedElement);  
 // Sentinel -> New node  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->next, sentinel\_list->tail);  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->prev, sentinel\_list->tail);  
 // New node -> Sentinel  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->next, sentinel\_list->tail);  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->prev, sentinel\_list->tail);  
  
 // Test insert shifts the list correctly and stores the node  
 // sentinel->node2->node1  
 Node \*node2 = malloc(sizeof(Node));  
 node2->type = NODE;  
 bool insertedElement2 = list\_insert(list, 0, node2);  
 TEST\_ASSERT\_TRUE(insertedElement2);  
 TEST\_ASSERT\_NOT\_EQUAL(sentinel\_list->head->next, sentinel\_list->tail);  
 TEST\_ASSERT\_EQUAL(sentinel\_list->head->next, sentinel\_list->tail->prev);  
  
 // sentinel->node2->node3->node1  
 Node \*node3 = malloc(sizeof(Node));  
 node3->type = NODE;  
 bool insertedElement3 = list\_insert(list, 1, node3);  
 TEST\_ASSERT\_TRUE(insertedElement2);  
 TEST\_ASSERT\_NOT\_EQUAL(sentinel\_list->head->next, node3);  
 TEST\_ASSERT\_EQUAL(node2->next, node3);  
 TEST\_ASSERT\_EQUAL(node3->next, node1);  
  
 // Append a non-node item  
 bool non\_node\_appended = list\_append(list, &test\_element);  
 TEST\_ASSERT\_FALSE(non\_node\_appended);  
 TEST\_ASSERT\_EQUAL(3, list->lists.sentinel\_list->size);  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 sentinel\_list = NULL;  
 list = NULL;  
}  
  
void test\_insert\_out\_of\_bounds(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 Node \*newNode = malloc(sizeof(Node));  
 newNode->type = NODE;  
  
 // Expected to fail (due to out of bounds)  
 bool insertedElement = list\_insert(list, 1, newNode);  
 TEST\_ASSERT\_FALSE(insertedElement);  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 free(newNode);  
 list = NULL;  
 newNode = NULL;  
}  
  
void test\_get\_element(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 Node \*node1 = malloc(sizeof(Node));  
 Node \*node2 = malloc(sizeof(Node));  
 node1->type = node2->type = NODE;  
 list\_append(list, node1);  
 list\_append(list, node2);  
  
 // Test item retrieval  
 void \*foundNode1 = list\_get(list, 0);  
 void \*foundNode2 = list\_get(list, 1);  
 TEST\_ASSERT\_NOT\_NULL(foundNode1);  
 TEST\_ASSERT\_NOT\_NULL(foundNode2);  
 TEST\_ASSERT\_EQUAL(node1, foundNode1);  
 TEST\_ASSERT\_EQUAL(node2, foundNode2);  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 node1 = node2 = NULL;  
 list = NULL;  
}  
  
void test\_get\_out\_of\_bounds(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 Node \*newNode = malloc(sizeof(Node));  
 newNode->type = NODE;  
 list\_append(list, newNode);  
  
 // Test item retrieval  
 void \*outOfBoundsNode = list\_get(list, 1);  
 TEST\_ASSERT\_NULL(outOfBoundsNode);  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 newNode = NULL;  
 list = NULL;  
}  
  
void test\_remove(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 Node \*node1 = malloc(sizeof(Node));  
 Node \*node2 = malloc(sizeof(Node));  
 Node \*node3 = malloc(sizeof(Node));  
 node1->type = node2->type = node3->type = NODE;  
 list\_append(list, node1);  
 list\_append(list, node2);  
 list\_append(list, node3);  
  
 // Test removing items + list shifts accordingly  
 void \*first\_item\_removed = list\_remove(list, 0); // removes node 1  
 void \*second\_item\_removed = list\_remove(list, 1); // removes node 3  
 void \*third\_item\_removed = list\_remove(list, 0); // removes node 2  
  
 TEST\_ASSERT\_NOT\_NULL(first\_item\_removed);  
 TEST\_ASSERT\_NOT\_NULL(second\_item\_removed);  
 TEST\_ASSERT\_NOT\_NULL(third\_item\_removed);  
 TEST\_ASSERT\_EQUAL(node1, first\_item\_removed);  
 TEST\_ASSERT\_EQUAL(node3, second\_item\_removed);  
 TEST\_ASSERT\_EQUAL(node2, third\_item\_removed);  
  
 // Cleanup  
 free(node1);  
 free(node2);  
 free(node3);  
 list\_destroy(list, free\_node);  
 list = NULL;  
 node1 = node2 = node3 = first\_item\_removed = second\_item\_removed =  
 third\_item\_removed = NULL;  
}  
  
void test\_remove\_out\_of\_bounds(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 SentinelLinkedList \*sentinel\_list = list->lists.sentinel\_list;  
 Node \*newNode = malloc(sizeof(Node));  
 newNode->type = NODE;  
 list\_append(list, newNode);  
  
 // Test remove fails  
 void \*non\_existent\_item\_removed = list\_remove(list, 1);  
 void \*item\_removed = list\_remove(list, 0);  
 void \*non\_existent\_item\_removed\_2 = list\_remove(list, 0);  
 printf("\nRemoved: %p\n", non\_existent\_item\_removed\_2);  
  
 TEST\_ASSERT\_NULL(non\_existent\_item\_removed);  
 TEST\_ASSERT\_NOT\_NULL(item\_removed);  
 TEST\_ASSERT\_NULL(non\_existent\_item\_removed\_2);  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 free(item\_removed);  
 list = NULL;  
 item\_removed = non\_existent\_item\_removed\_2 = non\_existent\_item\_removed =  
 newNode = NULL;  
}  
  
void test\_list\_size(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 SentinelLinkedList \*sentinel\_list = list->lists.sentinel\_list;  
 Node \*node1 = malloc(sizeof(Node));  
 Node \*node2 = malloc(sizeof(Node));  
  
 TEST\_ASSERT\_EQUAL(0, list\_size(list));  
 list\_append(list, node1);  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
 list\_append(list, node2);  
 TEST\_ASSERT\_EQUAL(2, list\_size(list));  
 list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
 list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL(0, list\_size(list));  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 free(node1);  
 free(node2);  
 node1 = node2 = NULL;  
 sentinel\_list = NULL;  
 list = NULL;  
}  
  
void test\_list\_is\_empty(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 Node \*newNode = malloc(sizeof(Node));  
 newNode->type = NODE;  
  
 // Test remove fails  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 list\_append(list, newNode);  
 TEST\_ASSERT\_FALSE(list\_is\_empty(list));  
  
 // Cleanup  
 list\_destroy(list, free\_node);  
 list = NULL;  
}  
  
int main(void) {  
 UNITY\_BEGIN();  
 RUN\_TEST(test\_list\_create);  
 RUN\_TEST(test\_list\_destroy);  
 RUN\_TEST(test\_append);  
 RUN\_TEST(test\_insert);  
 RUN\_TEST(test\_insert\_out\_of\_bounds);  
 RUN\_TEST(test\_get\_element);  
 RUN\_TEST(test\_get\_out\_of\_bounds);  
 RUN\_TEST(test\_remove);  
 RUN\_TEST(test\_remove\_out\_of\_bounds);  
 RUN\_TEST(test\_list\_is\_empty);  
 return UNITY\_END();  
}

## README

# Project 1 - Simple Linked List  
  
- Name: Chris Tudela  
- Email: christudela@u.boisestate.edu  
- Class: 452-002  
  
## Known Bugs or Issues  
  
No known bugs, but it is possible that I missed some edge cases I didn't think to test for.  
  
## Experience  
  
This project was a great refresher for pointers and the dynamics of managing memory. Some of my breakthroughs involved re-learning what exactly pointers are (references or copies?) and managing the memory (addresses). This definitely introduced some struggles, especially with dynamic behavior like the list\_destroy function. Cleaning all the nodes and the list(s) was challenging to do without memory leaks. It was mostly because I decided to manage a tail, which slightly paid off if the list ever gets large, but will hardly make a difference and should've probably excluded that functionality. Not to say that it works any differently, the implementation still respects the sentinel node -- it just introduced, probably, unnecessary struggle.   
  
Not necessarily a struggle, but I've been working in Typescript for the past year and it was interesting to see all the cross-over when typing. I noticed that Typescript inherited a lot of principles of C, where interfaces are basically structs, and many more elements. In other words, there were new discoveries for me when dealing with type checks and creations in C. I thought it was a great intro project and is definitely one that got me back up to speed -- hopefully enough to be ready for the next projects.

## End of Report

Report generated on 09/06/2025 at 00:55:08

## GitHub Info

* GitHub repo name: ctudel/cs452\_p1
* The repository visibility is public.
* The workflow was triggered by ctudel

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

a83987a9a14e1dab60137b68454568e4d0cdd1e3ced9ba36772b95656f2f0d24 submission-report.md