# Submission Report

* Submission generated at 09/07/2025 at 22:07:47
* 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

make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
mkdir -p build/debug  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address -c src/main.c -o build/debug/main.c.o  
mkdir -p build/debug  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address -c src/lab.c -o build/debug/lab.c.o  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address build/debug/main.c.o build/debug/lab.c.o -o build/debug/myapp\_d -fsanitize=address  
make[1]: Leaving directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
mkdir -p build/release  
cc -Wall -Wextra -O2 -fPIE -MMD -MP -Wformat -Wformat=2 -Wconversion -Wsign-conversion -Wimplicit-fallthrough -fstack-protector-strong -Werror=format-security -Werror=implicit -Werror=incompatible-pointer-types -Werror=int-conversion -c src/main.c -o build/release/main.c.o  
mkdir -p build/release  
cc -Wall -Wextra -O2 -fPIE -MMD -MP -Wformat -Wformat=2 -Wconversion -Wsign-conversion -Wimplicit-fallthrough -fstack-protector-strong -Werror=format-security -Werror=implicit -Werror=incompatible-pointer-types -Werror=int-conversion -c src/lab.c -o build/release/lab.c.o  
cc -Wall -Wextra -O2 -fPIE -MMD -MP -Wformat -Wformat=2 -Wconversion -Wsign-conversion -Wimplicit-fallthrough -fstack-protector-strong -Werror=format-security -Werror=implicit -Werror=incompatible-pointer-types -Werror=int-conversion build/release/main.c.o build/release/lab.c.o -o build/release/myapp   
make[1]: Leaving directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
mkdir -p build/tests  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c src/main.c -o build/tests/main.c.o  
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/main.c.o 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-Linked-List/P1-Linked-List'  
make[1]: Entering directory '/home/runner/work/P1-Linked-List/P1-Linked-List'  
mkdir -p build/debug-test  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address -c src/main.c -o build/debug-test/main.c.o  
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/main.c.o 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-Linked-List/P1-Linked-List'  
Builds completed. You can run the application with: ./build/release/myapp  
You can run the debug build with: ./build/debug/myapp\_d  
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

tests/lab-test.c:211:test\_list\_create\_destroy:PASS  
tests/lab-test.c:212:test\_list\_append\_and\_get:PASS  
tests/lab-test.c:213:test\_list\_insert:PASS  
tests/lab-test.c:214:test\_list\_remove:PASS  
tests/lab-test.c:215:test\_list\_get\_invalid:PASS  
tests/lab-test.c:216:test\_null\_list\_operations:PASS  
tests/lab-test.c:217:test\_large\_list\_and\_circular\_integrity:PASS  
tests/lab-test.c:218:test\_list\_create\_invalid\_type:PASS  
tests/lab-test.c:219:test\_list\_destroy\_null:PASS  
tests/lab-test.c:220:test\_list\_get\_null:PASS  
  
-----------------------  
10 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
tests/lab-test.c:211:test\_list\_create\_destroy:PASS  
tests/lab-test.c:212:test\_list\_append\_and\_get:PASS  
tests/lab-test.c:213:test\_list\_insert:PASS  
tests/lab-test.c:214:test\_list\_remove:PASS  
tests/lab-test.c:215:test\_list\_get\_invalid:PASS  
tests/lab-test.c:216:test\_null\_list\_operations:PASS  
tests/lab-test.c:217:test\_large\_list\_and\_circular\_integrity:PASS  
tests/lab-test.c:218:test\_list\_create\_invalid\_type:PASS  
tests/lab-test.c:219:test\_list\_destroy\_null:PASS  
tests/lab-test.c:220:test\_list\_get\_null: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 91 85 93% 34,55,60-61,114,144  
------------------------------------------------------------------------------  
TOTAL 91 85 93%  
------------------------------------------------------------------------------

## Address Sanitizer Report

tests/lab-test.c:211:test\_list\_create\_destroy:PASS  
tests/lab-test.c:212:test\_list\_append\_and\_get:PASS  
tests/lab-test.c:213:test\_list\_insert:PASS  
tests/lab-test.c:214:test\_list\_remove:PASS  
tests/lab-test.c:215:test\_list\_get\_invalid:PASS  
tests/lab-test.c:216:test\_null\_list\_operations:PASS  
tests/lab-test.c:217:test\_large\_list\_and\_circular\_integrity:PASS  
tests/lab-test.c:218:test\_list\_create\_invalid\_type:PASS  
tests/lab-test.c:219:test\_list\_destroy\_null:PASS  
tests/lab-test.c:220:test\_list\_get\_null:PASS  
  
-----------------------  
10 Tests 0 Failures 0 Ignored   
OK

## Source File: lab.c

#include "lab.h"  
#include <stdlib.h>  
#include <stdio.h>  
  
/\*\*  
 \* @brief structure for the doubly linked list  
 \* AI Use: No AI  
 \*/  
typedef struct Node {  
 void \*data;  
 struct Node \*next;  
 struct Node \*prev;  
} Node;  
  
/\*\*  
 \* @brief structure containing the sentinel node and metadata  
 \* AI Use: No AI  
 \*/  
struct List {  
 Node \*sentinel;  
 size\_t size;  
 ListType type;  
};  
  
/\*\*  
 \* @brief Create a new node with the given data  
 \* @param data Pointer to the data to store in the node  
 \* @return Pointer to the newly created node, or NULL on failure  
 \* AI Use: Assisted AI  
 \*/  
static Node \*node\_create(void \*data) {  
 Node \*node = (Node \*)malloc(sizeof(Node));  
 if (node == NULL) {  
 return NULL;  
 }  
 node->data = data;  
 node->next = NULL;  
 node->prev = NULL;  
 return node;  
}  
  
/\*\*  
 \* @brief Create a new list of the specified type  
 \* @param type The type of list to create  
 \* @return Pointer to the newly created list, or NULL on failure  
 \* AI Use: Assisted AI  
 \*/  
List \*list\_create(ListType type) {  
 if (type != LIST\_LINKED\_SENTINEL) {  
 return NULL;   
 }  
   
 List \*list = (List \*)malloc(sizeof(List));  
 if (list == NULL) {  
 return NULL;  
 }  
   
 list->sentinel = node\_create(NULL);  
 if (list->sentinel == NULL) {  
 free(list);  
 return NULL;  
 }  
   
 // sentinel points to itself  
 list->sentinel->next = list->sentinel;  
 list->sentinel->prev = list->sentinel;  
   
 list->size = 0;  
 list->type = type;  
   
 return list;  
}  
  
/\*\*  
 \* @brief the list and free all associated memory  
 \* @param list Pointer to the list to destroy  
 \* @param free\_func Function to free individual elements (can be NULL)  
 \* AI Use: Assisted AI  
 \*/  
void list\_destroy(List \*list, FreeFunc free\_func) {  
 if (list == NULL) {  
 return;  
 }  
   
 // Remove all nodes except sentinel  
 Node \*current = list->sentinel->next;  
 while (current != list->sentinel) {  
 Node \*next = current->next;  
 if (free\_func != NULL && current->data != NULL) {  
 free\_func(current->data);  
 }  
 free(current);  
 current = next;  
 }  
   
 free(list->sentinel);  
 free(list);  
}  
  
/\*\*  
 \* @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  
 \* AI Use: Assisted AI  
 \*/  
bool list\_append(List \*list, void \*data) {  
 if (list == NULL) {  
 return false;  
 }  
   
 Node \*new\_node = node\_create(data);  
 if (new\_node == NULL) {  
 return false;  
 }  
   
 // Insert new node before sentinel (at the end)  
 Node \*last = list->sentinel->prev;  
   
 new\_node->next = list->sentinel;  
 new\_node->prev = last;  
 last->next = new\_node;  
 list->sentinel->prev = new\_node;  
   
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* @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  
 \* AI Use: Assisted AI  
 \*/  
bool list\_insert(List \*list, size\_t index, void \*data) {  
 if (list == NULL || index > list->size) {  
 return false;  
 }  
   
 Node \*new\_node = node\_create(data);  
 if (new\_node == NULL) {  
 return false;  
 }  
   
 // Find the position to insert  
 Node \*current = list->sentinel;  
 for (size\_t i = 0; i < index; i++) {  
 current = current->next;  
 }  
   
 // Insert after current  
 Node \*next = current->next;  
 new\_node->next = next;  
 new\_node->prev = current;  
 current->next = new\_node;  
 next->prev = new\_node;  
   
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* @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 data, or NULL if index is out of bounds  
 \* AI Use: Assisted AI  
 \*/  
void \*list\_remove(List \*list, size\_t index) {  
 if (list == NULL || index >= list->size) {  
 return NULL;  
 }  
   
 // Find the node to remove  
 Node \*current = list->sentinel->next;  
 for (size\_t i = 0; i < index; i++) {  
 current = current->next;  
 }  
   
 // Remove the node from the list  
 void \*data = current->data;  
 current->prev->next = current->next;  
 current->next->prev = current->prev;  
   
 free(current);  
 list->size--;  
   
 return data;  
}  
  
/\*\*  
 \* @brief Get a pointer to 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  
 \* AI Use: Assisted AI  
 \*/  
void \*list\_get(const List \*list, size\_t index) {  
 if (list == NULL || index >= list->size) {  
 return NULL;  
 }  
   
 // Find the node at the specified index  
 Node \*current = list->sentinel->next;  
 for (size\_t i = 0; i < index; i++) {  
 current = current->next;  
 }  
   
 return current->data;  
}  
  
/\*\*  
 \* @brief Get the current size of the list  
 \* @param list Pointer to the list  
 \* @return The number of elements in the list  
 \* AI Use: Written by AI  
 \*/  
size\_t list\_size(const List \*list) {  
 if (list == NULL) {  
 return 0;  
 }  
 return list->size;  
}  
  
/\*\*  
 \* @brief Check if the list is empty  
 \* @param list Pointer to the list  
 \* @return true if the list is empty, false otherwise  
 \* AI Use: Written by AI  
 \*/  
bool list\_is\_empty(const List \*list) {  
 if (list == NULL) {  
 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

## Source File: main.c

#include "lab.h"  
#include <stdio.h>  
#include <stdlib.h>  
  
#ifdef TEST  
#define main main\_exclude  
#endif  
  
  
  
int main(void)  
{  
   
}

## Test Files

### lab-test.c

#include "../tests/harness/unity.h"  
#include "../src/lab.h"  
#include <stdlib.h>  
#include <string.h>  
  
/\* === Required by Unity === \*/  
void setUp(void) {}  
void tearDown(void) {}  
  
/\* === Helper structures and functions === \*/  
typedef struct {  
 int id;  
 char name[50];  
} TestObject;  
  
static TestObject \*create\_test\_object(int id, const char \*name) {  
 TestObject \*obj = (TestObject \*)malloc(sizeof(TestObject));  
 if (obj) {  
 obj->id = id;  
 strncpy(obj->name, name, sizeof(obj->name) - 1);  
 obj->name[sizeof(obj->name) - 1] = '\0';  
 }  
 return obj;  
}  
  
static void free\_test\_object(void \*obj) {  
 free(obj);  
}  
  
/\* === Tests === \*/  
  
void test\_list\_create\_destroy(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
 TEST\_ASSERT\_EQUAL\_UINT32(0, list\_size(list));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 list\_destroy(list, NULL);  
  
 /\* Create and destroy with free\_func \*/  
 list = list\_create(LIST\_LINKED\_SENTINEL);  
 for (int i = 0; i < 5; i++) {  
 list\_append(list, create\_test\_object(i, "tmp"));  
 }  
 list\_destroy(list, free\_test\_object);  
}  
  
void test\_list\_append\_and\_get(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 TestObject \*obj1 = create\_test\_object(1, "One");  
 TestObject \*obj2 = create\_test\_object(2, "Two");  
 TestObject \*obj3 = create\_test\_object(3, "Three");  
  
 TEST\_ASSERT\_TRUE(list\_append(list, obj1));  
 TEST\_ASSERT\_TRUE(list\_append(list, obj2));  
 TEST\_ASSERT\_TRUE(list\_append(list, obj3));  
 TEST\_ASSERT\_EQUAL\_UINT32(3, list\_size(list));  
  
 TestObject \*r1 = (TestObject \*)list\_get(list, 0);  
 TestObject \*r2 = (TestObject \*)list\_get(list, 1);  
 TestObject \*r3 = (TestObject \*)list\_get(list, 2);  
  
 TEST\_ASSERT\_NOT\_NULL(r1);  
 TEST\_ASSERT\_NOT\_NULL(r2);  
 TEST\_ASSERT\_NOT\_NULL(r3);  
 TEST\_ASSERT\_EQUAL\_INT(1, r1->id);  
 TEST\_ASSERT\_EQUAL\_INT(2, r2->id);  
 TEST\_ASSERT\_EQUAL\_INT(3, r3->id);  
  
 list\_destroy(list, free\_test\_object);  
}  
  
void test\_list\_insert(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 TestObject \*o1 = create\_test\_object(1, "First");  
 TestObject \*o2 = create\_test\_object(2, "Second");  
 TestObject \*o3 = create\_test\_object(3, "Third");  
 TestObject \*o4 = create\_test\_object(4, "Fourth");  
  
 /\* Insert at index 0 (empty list) \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, o1));  
  
 /\* Insert at end \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, o3));  
  
 /\* Insert in middle \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, o2));  
  
 /\* Insert at beginning \*/  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, o4));  
  
 /\* Verify order: 4,1,2,3 \*/  
 TestObject \*r0 = list\_get(list, 0);  
 TestObject \*r1 = list\_get(list, 1);  
 TestObject \*r2 = list\_get(list, 2);  
 TestObject \*r3 = list\_get(list, 3);  
 TEST\_ASSERT\_EQUAL\_INT(4, r0->id);  
 TEST\_ASSERT\_EQUAL\_INT(1, r1->id);  
 TEST\_ASSERT\_EQUAL\_INT(2, r2->id);  
 TEST\_ASSERT\_EQUAL\_INT(3, r3->id);  
  
 /\* Invalid insert \*/  
 TestObject \*bad = create\_test\_object(99, "Bad");  
 TEST\_ASSERT\_FALSE(list\_insert(list, 10, bad));  
 free\_test\_object(bad);  
  
 list\_destroy(list, free\_test\_object);  
}  
  
void test\_list\_remove(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
  
 TestObject \*objs[4];  
 for (int i = 0; i < 4; i++) {  
 objs[i] = create\_test\_object(i + 1, "X");  
 list\_append(list, objs[i]);  
 }  
  
 /\* Remove from middle \*/  
 TestObject \*r = list\_remove(list, 1);  
 TEST\_ASSERT\_EQUAL\_INT(2, r->id);  
 free\_test\_object(r);  
  
 /\* Remove from beginning \*/  
 r = list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL\_INT(1, r->id);  
 free\_test\_object(r);  
  
 /\* Remove from end \*/  
 r = list\_remove(list, 1);  
 TEST\_ASSERT\_EQUAL\_INT(4, r->id);  
 free\_test\_object(r);  
  
 /\* Remove last element \*/  
 r = list\_remove(list, 0);  
 TEST\_ASSERT\_EQUAL\_INT(3, r->id);  
 free\_test\_object(r);  
  
 /\* Now empty \*/  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 TEST\_ASSERT\_NULL(list\_remove(list, 0));  
  
 list\_destroy(list, NULL);  
}  
  
void test\_list\_get\_invalid(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NULL(list\_get(list, 0));  
 list\_destroy(list, NULL);  
}  
  
void test\_null\_list\_operations(void) {  
 TEST\_ASSERT\_FALSE(list\_append(NULL, (void \*)1));  
 TEST\_ASSERT\_FALSE(list\_insert(NULL, 0, (void \*)1));  
 TEST\_ASSERT\_NULL(list\_remove(NULL, 0));  
 TEST\_ASSERT\_NULL(list\_get(NULL, 0));  
 TEST\_ASSERT\_EQUAL\_UINT32(0, list\_size(NULL));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(NULL));  
}  
  
void test\_large\_list\_and\_circular\_integrity(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 for (int i = 0; i < 1000; i++) {  
 list\_append(list, create\_test\_object(i, "bulk"));  
 }  
 TEST\_ASSERT\_EQUAL\_UINT32(1000, list\_size(list));  
  
 /\* Remove some elements \*/  
 free\_test\_object(list\_remove(list, 500));  
 free\_test\_object(list\_remove(list, 0));  
 free\_test\_object(list\_remove(list, list\_size(list) - 1));  
  
 /\* Insert at front and back \*/  
 list\_insert(list, 0, create\_test\_object(2000, "front"));  
 list\_insert(list, list\_size(list), create\_test\_object(3000, "back"));  
 TEST\_ASSERT\_EQUAL\_UINT32(999, list\_size(list));  
  
 /\* Clear \*/  
 while (!list\_is\_empty(list)) {  
 free\_test\_object(list\_remove(list, 0));  
 }  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
  
 list\_destroy(list, NULL);  
}  
  
// Test creating list with invalid type  
void test\_list\_create\_invalid\_type(void) {  
 List \*list = list\_create(999); // unsupported type  
 TEST\_ASSERT\_NULL(list);  
}  
  
// Test destroying NULL list  
void test\_list\_destroy\_null(void) {  
 // Should not crash  
 list\_destroy(NULL, NULL);  
 TEST\_ASSERT\_TRUE(1); // dummy check just to satisfy Unity  
}  
  
// Test list\_get on NULL list  
void test\_list\_get\_null(void) {  
 TEST\_ASSERT\_NULL(list\_get(NULL, 0));  
 TEST\_ASSERT\_NULL(list\_get(NULL, 100)); // redundant but forces both args  
}  
  
  
/\* === Test Runner === \*/  
int main(void) {  
 UNITY\_BEGIN();  
 RUN\_TEST(test\_list\_create\_destroy);  
 RUN\_TEST(test\_list\_append\_and\_get);  
 RUN\_TEST(test\_list\_insert);  
 RUN\_TEST(test\_list\_remove);  
 RUN\_TEST(test\_list\_get\_invalid);  
 RUN\_TEST(test\_null\_list\_operations);  
 RUN\_TEST(test\_large\_list\_and\_circular\_integrity);  
 RUN\_TEST(test\_list\_create\_invalid\_type);  
 RUN\_TEST(test\_list\_destroy\_null);  
 RUN\_TEST(test\_list\_get\_null);  
  
 return UNITY\_END();  
}

## README

# Project 1 Linked list (sentenial)  
  
- Name: Ryan Wensmann  
- Email: Ryan Wensmann@u.boisestate.edu  
- Class: 452-001  
  
## Known Bugs or Issues  
  
There are no issues but I was unable to reach those edge cases for 100% coverage but I belive it has to do with malloc.  
  
## Experience  
  
Understanding the correct pointer manipulation for a circular, doubly linked list with a sentinel node. Off-by-one errors and forgetting to update both next and prev links caused tricky bugs at first.  
Achieving near-100% code coverage required thinking carefully about edge cases, like NULL lists and out-of-bounds indices.

## End of Report

Report generated on 09/07/2025 at 22:07:49

## GitHub Info

* GitHub repo name: rw139b/P1-Linked-List
* The repository visibility is public.
* The workflow was triggered by rw139b

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

0aeaac2e1268621f3df377fb0fc98c5d33a61cb24b68bc8470828290e5cc533c submission-report.md