# Submission Report

* Submission generated at 09/06/2025 at 04:05:54
* 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/p1'  
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  
/usr/bin/ld: /usr/lib/gcc/x86\_64-linux-gnu/13/../../../x86\_64-linux-gnu/Scrt1.o: in function `\_start':  
(.text+0x1b): undefined reference to `main'  
collect2: error: ld returned 1 exit status  
make[1]: \*\*\* [Makefile:55: build/debug/myapp\_d] Error 1  
make[1]: Leaving directory '/home/runner/work/p1/p1'  
make[1]: Entering directory '/home/runner/work/p1/p1'  
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   
/usr/bin/ld: /usr/lib/gcc/x86\_64-linux-gnu/13/../../../x86\_64-linux-gnu/Scrt1.o: in function `\_start':  
(.text+0x1b): undefined reference to `main'  
collect2: error: ld returned 1 exit status  
make[1]: \*\*\* [Makefile:55: build/release/myapp] Error 1  
make[1]: Leaving directory '/home/runner/work/p1/p1'  
make[1]: Entering directory '/home/runner/work/p1/p1'  
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/p1'  
make[1]: Entering directory '/home/runner/work/p1/p1'  
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/p1'  
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:206:test\_create\_and\_destroy:PASS  
tests/lab-test.c:207:test\_append\_and\_get:PASS  
tests/lab-test.c:208:test\_insert:PASS  
tests/lab-test.c:209:test\_remove:PASS  
tests/lab-test.c:210:test\_size\_and\_empty:PASS  
tests/lab-test.c:211:test\_out\_of\_bounds:PASS  
tests/lab-test.c:212:test\_alloc\_hook\_smoke:PASS  
tests/lab-test.c:213:test\_list\_append\_alloc\_failure:PASS  
tests/lab-test.c:214:test\_list\_create\_alloc\_failure:PASS  
tests/lab-test.c:215:test\_destroy\_calls\_destructor:PASS  
tests/lab-test.c:216:test\_list\_insert\_alloc\_failure:PASS  
tests/lab-test.c:217:test\_get\_last\_index\_and\_oob:PASS  
tests/lab-test.c:218:test\_insert\_head\_tail\_and\_remove\_to\_empty:PASS  
tests/lab-test.c:219:test\_null\_list\_guards:PASS  
  
-----------------------  
14 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
tests/lab-test.c:206:test\_create\_and\_destroy:PASS  
tests/lab-test.c:207:test\_append\_and\_get:PASS  
tests/lab-test.c:208:test\_insert:PASS  
tests/lab-test.c:209:test\_remove:PASS  
tests/lab-test.c:210:test\_size\_and\_empty:PASS  
tests/lab-test.c:211:test\_out\_of\_bounds:PASS  
tests/lab-test.c:212:test\_alloc\_hook\_smoke:PASS  
tests/lab-test.c:213:test\_list\_append\_alloc\_failure:PASS  
tests/lab-test.c:214:test\_list\_create\_alloc\_failure:PASS  
tests/lab-test.c:215:test\_destroy\_calls\_destructor:PASS  
tests/lab-test.c:216:test\_list\_insert\_alloc\_failure:PASS  
tests/lab-test.c:217:test\_get\_last\_index\_and\_oob:PASS  
tests/lab-test.c:218:test\_insert\_head\_tail\_and\_remove\_to\_empty:PASS  
tests/lab-test.c:219:test\_null\_list\_guards:PASS  
  
-----------------------  
14 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 82 82 100%  
------------------------------------------------------------------------------  
TOTAL 82 82 100%  
------------------------------------------------------------------------------

## Address Sanitizer Report

tests/lab-test.c:206:test\_create\_and\_destroy:PASS  
tests/lab-test.c:207:test\_append\_and\_get:PASS  
tests/lab-test.c:208:test\_insert:PASS  
tests/lab-test.c:209:test\_remove:PASS  
tests/lab-test.c:210:test\_size\_and\_empty:PASS  
tests/lab-test.c:211:test\_out\_of\_bounds:PASS  
tests/lab-test.c:212:test\_alloc\_hook\_smoke:PASS  
tests/lab-test.c:213:test\_list\_append\_alloc\_failure:PASS  
tests/lab-test.c:214:test\_list\_create\_alloc\_failure:PASS  
tests/lab-test.c:215:test\_destroy\_calls\_destructor:PASS  
tests/lab-test.c:216:test\_list\_insert\_alloc\_failure:PASS  
tests/lab-test.c:217:test\_get\_last\_index\_and\_oob:PASS  
tests/lab-test.c:218:test\_insert\_head\_tail\_and\_remove\_to\_empty:PASS  
tests/lab-test.c:219:test\_null\_list\_guards:PASS  
  
-----------------------  
14 Tests 0 Failures 0 Ignored   
OK

## Source File: lab.c

#include "lab.h"  
#include <stdio.h>  
#include <stdlib.h>  
  
#ifndef ALLOC  
#define ALLOC(size) malloc(size)  
#endif  
#ifndef DESTROY  
#define DESTROY(ptr) free(ptr)  
#endif  
  
/\*\*  
 \* Global function pointer for custom allocation/deallocation. Set to NULL to use default ALLOC.  
 \* AI Use: Written By AI  
 \*/  
AllocFn lab\_alloc\_fn = NULL;  
FreeFn lab\_free\_fn = NULL;  
  
/\*\*  
 \* Node structure for the circular, doubly linked list.  
 \* AI Use: AI Assisted  
 \*/  
typedef struct Node {  
 void \*data;  
 struct Node \*prev;  
 struct Node \*next;  
} Node;  
  
/\*\*  
 \* List structure definition for the circular, doubly linked list with sentinel node.  
 \* AI Use: AI Assisted  
 \*/  
struct List {  
 size\_t size;  
 ListType type;  
 Node \*sentinel;  
};  
  
/\*\*  
 \* Creates a new circular, doubly linked list with a sentinel node.  
 \* AI Use: AI Assisted  
 \*/  
List \*list\_create(ListType type) {  
 // Allocate memory for the list  
 List \*list = ALLOC(sizeof(List));  
 if (list == NULL) {  
 return NULL; // allocation failed  
 }  
  
 // Allocate memory for the sentinel node  
 Node \*sentinel = ALLOC(sizeof(Node));  
 if (sentinel == NULL) {  
 DESTROY(list);  
 return NULL;  
 }  
  
 // Initialize the sentinel node (circular self-links)  
 sentinel->data = NULL;  
 sentinel->next = sentinel;  
 sentinel->prev = sentinel;  
  
 // Initialize the list  
 list->size = 0;  
 list->type = type;  
 list->sentinel = sentinel;  
  
 return list;  
}  
  
/\*\*  
 \* Destroys the list and frees all associated memory. Calls free\_func on each data element if provided.  
 \* AI Use: AI Assisted  
 \*/  
void list\_destroy(List \*list, FreeFunc free\_func) {  
 if (!list) return;  
 Node \*sentinel = list->sentinel;  
 Node \*curr = sentinel->next;  
 while (curr != sentinel) {  
 Node \*next = curr->next;  
 if (free\_func && curr->data) {  
 free\_func(curr->data);  
 }  
 DESTROY(curr);  
 curr = next;  
 }  
 DESTROY(sentinel);  
 DESTROY(list);  
}  
  
/\*\*  
 \* Appends a new element to the end of the list.  
 \* AI Use: AI Assisted  
 \*/  
bool list\_append(List \*list, void \*data) {  
 if (!list || !list->sentinel) return false;  
 Node \*new\_node = ALLOC(sizeof(Node));  
 if (!new\_node) return false;  
 new\_node->data = data;  
  
 Node \*sentinel = list->sentinel;  
 Node \*last = sentinel->prev;  
  
 // Insert new\_node before sentinel  
 new\_node->next = sentinel;  
 new\_node->prev = last;  
 last->next = new\_node;  
 sentinel->prev = new\_node;  
  
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* Inserts a new element at the specified index in the list.  
 \* AI Use: AI Assisted  
 \*/  
bool list\_insert(List \*list, size\_t index, void \*data) {  
 if (!list || !list->sentinel) return false;  
 if (index > list->size) return false; // index out of bounds  
  
 Node \*sentinel = list->sentinel;  
 Node \*curr = sentinel->next;  
 for (size\_t i = 0; i < index; ++i) {  
 curr = curr->next;  
 }  
  
 Node \*new\_node = ALLOC(sizeof(Node));  
 if (!new\_node) return false;  
 new\_node->data = data;  
  
 // Insert new\_node before curr  
 new\_node->prev = curr->prev;  
 new\_node->next = curr;  
 curr->prev->next = new\_node;  
 curr->prev = new\_node;  
  
 list->size++;  
 return true;  
}  
  
/\*\*  
 \* Removes the element at the specified index from the list and returns its data pointer.  
 \* AI Use: AI Assisted  
 \*/  
void \*list\_remove(List \*list, size\_t index) {  
 if (!list || !list->sentinel) return NULL;  
 if (index >= list->size) return NULL;  
 Node \*sentinel = list->sentinel;  
 Node \*curr = sentinel->next;  
 for (size\_t i = 0; i < index; ++i) {  
 curr = curr->next;  
 }  
 void \*data = curr->data;  
 curr->prev->next = curr->next;  
 curr->next->prev = curr->prev;  
 DESTROY(curr);  
 list->size--;  
 return data;  
}  
  
/\*\*  
 \* Returns the data pointer at the specified index in the list.  
 \* AI Use: AI Assisted  
 \*/  
void \*list\_get(const List \*list, size\_t index) {  
 if (!list || !list->sentinel) return NULL;  
 if (index >= list->size) return NULL;  
 Node \*curr = list->sentinel->next;  
 for (size\_t i = 0; i < index; ++i) {  
 curr = curr->next;  
 }  
 return curr->data;  
}  
  
/\*\*  
 \* Returns the number of elements in the list.  
 \* AI Use: AI Assisted  
 \*/  
size\_t list\_size(const List \*list) {  
 if (!list) return 0;  
 return list->size;  
}  
  
/\*\*  
 \* Returns true if the list is empty, false otherwise.  
 \* AI Use: AI Assisted  
 \*/  
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>  
#include <stdlib.h> // for malloc, free.   
 #include <stdlib.h> // malloc, free  
  
//alloc/free hook types + externs  
typedef void \*(\*AllocFn)(size\_t); //function pointers  
typedef void (\*FreeFn)(void \*); //Function pointers  
extern AllocFn lab\_alloc\_fn; //golbal function pointer for custom allocation  
extern FreeFn lab\_free\_fn; //golbal function pointer for custom free  
  
/\*is a macro wrapper around allocation.  
If lab\_alloc\_fn is set, call it (test hook).  
Otherwise call malloc(sz).\*/  
#ifndef ALLOC  
# define ALLOC(sz) (lab\_alloc\_fn ? lab\_alloc\_fn(sz) : malloc(sz))  
#endif  
#ifndef DESTROY  
# define DESTROY(p) do { if (lab\_free\_fn) lab\_free\_fn(p); else free(p); } while (0)  
#endif  
  
/\*\*  
 \* @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

//Don't need main.c

## Test Files

### lab-test.c

#include <stdlib.h>  
#include <stdio.h>  
#include "harness/unity.h"  
  
//All tests written by AI  
  
// --- ALLOC/DESTROY mocking for allocation failure testing ---  
static int alloc\_fail\_after = -1;  
static int alloc\_call\_count = 0;  
void \*test\_alloc(size\_t size) {  
 alloc\_call\_count++;  
 if (alloc\_fail\_after > 0 && alloc\_call\_count == alloc\_fail\_after) {  
 return NULL;  
 }  
 return malloc(size);  
}  
void test\_destroy(void \*ptr) {  
 free(ptr);  
}  
  
#include "../src/lab.h"  
  
void setUp(void) {  
 alloc\_fail\_after = -1;  
 alloc\_call\_count = 0;  
  
 // Redirect allocations in lab.c to our test hooks  
 lab\_alloc\_fn = test\_alloc;  
 lab\_free\_fn = test\_destroy;  
}  
void tearDown(void) {}  
  
static void test\_create\_and\_destroy(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
 list\_destroy(list, NULL);  
}  
  
static void test\_alloc\_hook\_smoke(void) {  
 alloc\_fail\_after = -1;  
 alloc\_call\_count = 0;  
  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
  
 // list\_create should allocate 2 blocks: List + sentinel  
 TEST\_ASSERT\_TRUE(alloc\_call\_count >= 2);  
  
 list\_destroy(list, NULL);  
}  
  
  
static void test\_append\_and\_get(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a = 1, b = 2, c = 3;  
 TEST\_ASSERT\_TRUE(list\_append(list, &a));  
 TEST\_ASSERT\_TRUE(list\_append(list, &b));  
 TEST\_ASSERT\_TRUE(list\_append(list, &c));  
 TEST\_ASSERT\_EQUAL\_PTR(&a, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(&b, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL\_PTR(&c, list\_get(list, 2));  
 list\_destroy(list, NULL);  
}  
  
static void test\_insert(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a = 1, b = 2, c = 3;  
 list\_append(list, &a);  
 list\_append(list, &c);  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, &b));  
 TEST\_ASSERT\_EQUAL\_PTR(&a, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(&b, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL\_PTR(&c, list\_get(list, 2));  
 list\_destroy(list, NULL);  
}  
  
static void test\_remove(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a = 1, b = 2, c = 3;  
 list\_append(list, &a);  
 list\_append(list, &b);  
 list\_append(list, &c);  
 void \*removed = list\_remove(list, 1);  
 TEST\_ASSERT\_EQUAL\_PTR(&b, removed);  
 TEST\_ASSERT\_EQUAL\_PTR(&a, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(&c, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL\_UINT32(2, list\_size(list));  
 list\_destroy(list, NULL);  
}  
  
static void test\_size\_and\_empty(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 int a = 1;  
 list\_append(list, &a);  
 TEST\_ASSERT\_FALSE(list\_is\_empty(list));  
 TEST\_ASSERT\_EQUAL\_UINT32(1, list\_size(list));  
 list\_remove(list, 0);  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 list\_destroy(list, NULL);  
}  
  
static void test\_out\_of\_bounds(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a = 1;  
 TEST\_ASSERT\_FALSE(list\_insert(list, 1, &a));  
 TEST\_ASSERT\_NULL(list\_get(list, 0));  
 TEST\_ASSERT\_NULL(list\_remove(list, 0));  
 list\_destroy(list, NULL);  
}  
  
void test\_list\_create\_alloc\_failure(void) {  
 alloc\_fail\_after = 1; // fail on first ALLOC (List allocation)  
 alloc\_call\_count = 0;  
 TEST\_ASSERT\_NULL(list\_create(LIST\_LINKED\_SENTINEL));  
  
 alloc\_fail\_after = 2; // fail on second ALLOC (sentinel allocation)  
 alloc\_call\_count = 0;  
 TEST\_ASSERT\_NULL(list\_create(LIST\_LINKED\_SENTINEL));  
  
 alloc\_fail\_after = -1; // reset  
}  
  
void test\_list\_append\_alloc\_failure(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a = 1;  
 alloc\_fail\_after = 1; // fail on first ALLOC in append  
 alloc\_call\_count = 0;  
 TEST\_ASSERT\_FALSE(list\_append(list, &a));  
 list\_destroy(list, NULL);  
 alloc\_fail\_after = -1; // reset  
}  
  
static int free\_count = 0;  
static void dummy\_free(void \*ptr) { (void)ptr; free\_count++; }  
  
void test\_destroy\_calls\_destructor(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a=1, b=2;  
 list\_append(list, &a);  
 list\_append(list, &b);  
  
 free\_count = 0;  
 list\_destroy(list, dummy\_free);  
 TEST\_ASSERT\_EQUAL\_INT(2, free\_count);  
}  
  
void test\_list\_insert\_alloc\_failure(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a = 1, b = 2;  
 list\_append(list, &a);  
  
 alloc\_fail\_after = 1; // fail on allocation inside insert  
 alloc\_call\_count = 0;  
 TEST\_ASSERT\_FALSE(list\_insert(list, 1, &b));  
  
 // state unchanged  
 TEST\_ASSERT\_EQUAL\_UINT32(1, list\_size(list));  
 TEST\_ASSERT\_EQUAL\_PTR(&a, list\_get(list, 0));  
  
 list\_destroy(list, NULL);  
 alloc\_fail\_after = -1;  
}  
  
static void test\_get\_last\_index\_and\_oob(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a=1,b=2;  
 list\_append(list,&a);  
 list\_append(list,&b);  
 TEST\_ASSERT\_EQUAL\_PTR(&b, list\_get(list, list\_size(list)-1));  
 TEST\_ASSERT\_NULL(list\_get(list, list\_size(list))); // OOB  
 list\_destroy(list,NULL);  
}  
  
static void test\_insert\_head\_tail\_and\_remove\_to\_empty(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int a=1,b=2,c=3;  
 TEST\_ASSERT\_TRUE(list\_insert(list, 0, &a)); // head insert into empty  
 TEST\_ASSERT\_TRUE(list\_insert(list, list\_size(list), &b)); // tail insert  
 TEST\_ASSERT\_TRUE(list\_insert(list, 1, &c)); // middle  
 // a, c, b  
 TEST\_ASSERT\_EQUAL\_PTR(&a, list\_get(list, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(&c, list\_get(list, 1));  
 TEST\_ASSERT\_EQUAL\_PTR(&b, list\_get(list, 2));  
  
 TEST\_ASSERT\_EQUAL\_PTR(&a, list\_remove(list, 0)); // remove head  
 TEST\_ASSERT\_EQUAL\_PTR(&b, list\_remove(list, list\_size(list)-1)); // remove tail  
 TEST\_ASSERT\_EQUAL\_PTR(&c, list\_remove(list, 0)); // last one  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
 list\_destroy(list, NULL);  
}  
  
static void test\_null\_list\_guards(void) {  
 int x=42;  
 TEST\_ASSERT\_FALSE(list\_append(NULL,&x));  
 TEST\_ASSERT\_FALSE(list\_insert(NULL,0,&x));  
 TEST\_ASSERT\_NULL(list\_get(NULL,0));  
 TEST\_ASSERT\_NULL(list\_remove(NULL,0));  
 TEST\_ASSERT\_EQUAL\_UINT32(0, list\_size(NULL));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(NULL));  
}  
  
int main(void) {  
 UNITY\_BEGIN();  
 RUN\_TEST(test\_create\_and\_destroy);  
 RUN\_TEST(test\_append\_and\_get);  
 RUN\_TEST(test\_insert);  
 RUN\_TEST(test\_remove);  
 RUN\_TEST(test\_size\_and\_empty);  
 RUN\_TEST(test\_out\_of\_bounds);  
 RUN\_TEST(test\_alloc\_hook\_smoke);  
 RUN\_TEST(test\_list\_append\_alloc\_failure);  
 RUN\_TEST(test\_list\_create\_alloc\_failure);  
 RUN\_TEST(test\_destroy\_calls\_destructor);  
 RUN\_TEST(test\_list\_insert\_alloc\_failure);  
 RUN\_TEST(test\_get\_last\_index\_and\_oob);  
 RUN\_TEST(test\_insert\_head\_tail\_and\_remove\_to\_empty);  
 RUN\_TEST(test\_null\_list\_guards);  
 return UNITY\_END();  
}

## README

# Project 1  
  
- Name: Laura Loughmiller  
- Email: laurajones2@u.boisestate.edu  
- Class: 452-001  
  
## Known Bugs or Issues  
  
No known Bugs or Issues  
  
## Experience  
  
Well it's done. Hopefully I didn't booby trap p2. Here is a quick break down of what cost me the most time.  
  
### Problem: I needed to test allocation-failure branches  
but lab.c was compiled in a separate translation unit, so my test-side #define ALLOC=... didn’t affect it. Including lab.c in tests caused duplicate symbols because the Makefile already compiled it.  
  
### Solution: hooks  
In lab.h: define AllocFn/FreeFn and extern AllocFn lab\_alloc\_fn; extern FreeFn lab\_free\_fn;, then  
 #define ALLOC(sz) (lab\_alloc\_fn ? lab\_alloc\_fn(sz) : malloc(sz))  
 #define DESTROY(p) do { if (lab\_free\_fn) lab\_free\_fn(p); else free(p); } while (0)  
In lab.c: define the globals once:  
 AllocFn lab\_alloc\_fn = NULL;  
 FreeFn lab\_free\_fn = NULL;  
In tests (setUp): point hooks to the test doubles:  
 lab\_alloc\_fn = test\_alloc;  
 lab\_free\_fn = test\_destroy;  
  
#### Why it works: No Makefile changes.   
lab.c still compiles normally, but at runtime the tests can inject a failing allocator. In production, hooks are NULL → real malloc/free.  
  
Used a smoke test to confirm the hook is hit (alloc count ≥ 2 in list\_create), then forced failures (alloc\_fail\_after = 1/2) to cover error paths in list\_create, list\_append, and list\_insert.

## End of Report

Report generated on 09/06/2025 at 04:05:55

## GitHub Info

* GitHub repo name: laurajones2/p1
* The repository visibility is private.
* The workflow was triggered by laurajones2

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

90049789f254c2098f7417afde83b6f06687f6036c13f643b2dcc3df9efb61d9 submission-report.md b7cf47c5f144c55d355063b0ed687e1d59b60e58e74d83214b7db17df5002f78 submission-report.md