# Submission Report

* Submission generated at 09/30/2025 at 17:08:59
* Machine info: Linux runnervm3ublj 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-Simple-Linked-List/P1-Simple-Linked-List'  
mkdir -p build/debug  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address -c src/lab.c -o build/debug/lab.c.o  
mkdir -p build/debug  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address -c src/main.c -o build/debug/main.c.o  
cc -g -O0 -DDEBUG -fno-omit-frame-pointer -fsanitize=address build/debug/lab.c.o build/debug/main.c.o -o build/debug/myapp\_d -fsanitize=address  
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/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  
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  
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/lab.c.o build/release/main.c.o -o build/release/myapp   
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/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 src/main.c -o build/tests/main.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  
mkdir -p build/tests/  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage -c tests/lab-test.c -o build/tests/lab-test.c.o  
cc -g -O0 -DTEST -fprofile-arcs -ftest-coverage build/tests/lab.c.o build/tests/main.c.o build/tests/harness/unity.c.o build/tests/lab-test.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 src/main.c -o build/debug-test/main.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  
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  
cc -g -O0 -DDEBUG -DTEST -fno-omit-frame-pointer -fsanitize=address build/debug-test/lab.c.o build/debug-test/main.c.o build/debug-test/harness/unity.c.o build/debug-test/lab-test.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'  
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

Setting up tests...  
Tearing down tests...  
tests/lab-test.c:518:test\_list\_create:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:519:test\_list\_append:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:520:test\_list\_insert:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:521:test\_list\_insert\_and\_remove:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:522:test\_list\_size\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:523:test\_list\_is\_empty\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:524:test\_list\_get\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:525:test\_insert\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:526:test\_append\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:527:test\_list\_remove\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:528:test\_compare\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:529:test\_compare\_str:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:530:test\_sort\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:531:test\_sort\_str:PASS  
Setting up tests...  
merged[0] = 10  
merged[1] = 9  
merged[2] = 8  
merged[3] = 7  
Tearing down tests...  
tests/lab-test.c:532:test\_merge\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:533:test\_merge\_str:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:534:test\_is\_sorted\_descending\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:535:test\_split:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:536:test\_split\_sort\_merge:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:537:test\_sort\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:538:test\_split\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:539:test\_split\_internal\_middle:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:540:test\_split\_index\_zero\_moves\_all\_nodes:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:541:test\_is\_sorted\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:542:test\_is\_sorted\_detects\_unsorted\_pair:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:543:test\_generate\_random\_string\_chars\_and\_length:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:544:test\_generate\_list\_ints:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:545:test\_generate\_list\_uses\_random\_string:PASS  
  
-----------------------  
28 Tests 0 Failures 0 Ignored   
OK  
./build/tests/myapp\_t  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:518:test\_list\_create:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:519:test\_list\_append:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:520:test\_list\_insert:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:521:test\_list\_insert\_and\_remove:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:522:test\_list\_size\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:523:test\_list\_is\_empty\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:524:test\_list\_get\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:525:test\_insert\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:526:test\_append\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:527:test\_list\_remove\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:528:test\_compare\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:529:test\_compare\_str:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:530:test\_sort\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:531:test\_sort\_str:PASS  
Setting up tests...  
merged[0] = 10  
merged[1] = 9  
merged[2] = 8  
merged[3] = 7  
Tearing down tests...  
tests/lab-test.c:532:test\_merge\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:533:test\_merge\_str:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:534:test\_is\_sorted\_descending\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:535:test\_split:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:536:test\_split\_sort\_merge:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:537:test\_sort\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:538:test\_split\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:539:test\_split\_internal\_middle:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:540:test\_split\_index\_zero\_moves\_all\_nodes:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:541:test\_is\_sorted\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:542:test\_is\_sorted\_detects\_unsorted\_pair:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:543:test\_generate\_random\_string\_chars\_and\_length:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:544:test\_generate\_list\_ints:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:545:test\_generate\_list\_uses\_random\_string:PASS  
  
-----------------------  
28 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 169 169 100%  
------------------------------------------------------------------------------  
TOTAL 169 169 100%  
------------------------------------------------------------------------------

## Address Sanitizer Report

Setting up tests...  
Tearing down tests...  
tests/lab-test.c:518:test\_list\_create:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:519:test\_list\_append:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:520:test\_list\_insert:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:521:test\_list\_insert\_and\_remove:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:522:test\_list\_size\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:523:test\_list\_is\_empty\_null:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:524:test\_list\_get\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:525:test\_insert\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:526:test\_append\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:527:test\_list\_remove\_error:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:528:test\_compare\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:529:test\_compare\_str:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:530:test\_sort\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:531:test\_sort\_str:PASS  
Setting up tests...  
merged[0] = 10  
merged[1] = 9  
merged[2] = 8  
merged[3] = 7  
Tearing down tests...  
tests/lab-test.c:532:test\_merge\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:533:test\_merge\_str:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:534:test\_is\_sorted\_descending\_int:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:535:test\_split:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:536:test\_split\_sort\_merge:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:537:test\_sort\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:538:test\_split\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:539:test\_split\_internal\_middle:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:540:test\_split\_index\_zero\_moves\_all\_nodes:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:541:test\_is\_sorted\_guard\_conditions:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:542:test\_is\_sorted\_detects\_unsorted\_pair:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:543:test\_generate\_random\_string\_chars\_and\_length:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:544:test\_generate\_list\_ints:PASS  
Setting up tests...  
Tearing down tests...  
tests/lab-test.c:545:test\_generate\_list\_uses\_random\_string:PASS  
  
-----------------------  
28 Tests 0 Failures 0 Ignored   
OK

## Source File: lab.c

#include "lab.h"  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
/\*\*  
 \* Represents a circular list with a sentinel node.   
 \* AI Use: No AI   
 \*/  
struct List {   
 struct Node\* head; // sentinel node   
 size\_t 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;  
  
// typedef int Compare {  
  
// }  
  
/\*\*  
 \* 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;  
}  
  
/\*\*  
 \* Sorts the circular linked list in descending order using insertion sort.  
 \* AI Use: Assisted by AI  
 \*/  
void sort(List \*list, size\_t start, size\_t end, Compare cmp) {  
 if(!list || start >= end || end > list->size || !cmp) {  
 return;  
 }  
  
 // starting in second position, assuming first is sorted   
 for (size\_t i = start + 1; i < end; ++i) {  
 // current element being sorted   
 void \*key = list\_get(list, i);  
 size\_t j = i;  
  
 while (j > start && cmp(list\_get(list, j - 1), key) < 0) {  
 Node \*curr = list->head->next;  
 for (size\_t k = 0; k < j; ++k) {  
 curr = curr->next;  
 }  
 Node \*prev = list->head->next;   
 for (size\_t k = 0; k < j - 1; ++k) {  
 prev = prev->next;  
 }  
 curr->data = prev->data;  
 j--;  
 }  
 // place key in correct position   
 Node \*val = list->head->next;  
 for (size\_t k = 0; k < j; ++k) {  
 val = val->next;  
 }  
 val->data = key;  
 }  
}  
  
/\*\*  
 \* Merges two sorted circular linked lists into a new sorted circular linked list.  
 \* AI Use: Assisted by AI  
 \*/  
List\* merge(List \*list1, List \*list2, Compare cmp) {  
 List \*merged = list\_create(LIST\_LINKED\_SENTINEL);  
  
 while (list1->size > 0 && list2->size > 0) {  
 Node \*n1 = list1->head->next; // first node in list1  
 Node \*n2 = list2->head->next; // first node in list2  
  
 if (cmp(n1->data, n2->data) > 0) { // n1 is greater  
 list\_append(merged, n1->data);  
 n1->prev->next = n1->next;  
 n1->next->prev = n1->prev;  
 free(n1);  
 list1->size--;  
 } else { // n2 is greater or equal  
 list\_append(merged, n2->data);  
 n2->prev->next = n2->next;  
 n2->next->prev = n2->prev;  
 free(n2);  
 list2->size--;  
 }  
 }  
  
 while (list1->size > 0) {  
 Node \*node = list1->head->next;  
 list\_append(merged, node->data);  
 node->prev->next = node->next;  
 node->next->prev = node->prev;  
 free(node);  
 list1->size--;  
 }  
  
 // move remaining nodes from list2  
 while (list2->size > 0) {  
 Node \*node = list2->head->next;  
 list\_append(merged, node->data);  
 node->prev->next = node->next;  
 node->next->prev = node->prev;  
 free(node);  
 list2->size--;  
 }  
  
 return merged;  
}  
  
/\*\*  
 \* Splits the circular linked list into two lists at the specified index.  
 \* AI Use: Assisted by AI  
 \*/  
List\* split(List \*list, size\_t index) {  
 if (!list || index >= list->size) {  
 return NULL;   
 }  
  
 List \*newList = list\_create(LIST\_LINKED\_SENTINEL);  
 if (!newList) { // GCOVR\_EXCL\_START  
 return NULL;   
 } // GCOVR\_EXCL\_STOP  
  
 Node \*curr = list->head->next;   
 for (size\_t i = 0; i < index; i++) {  
 curr = curr->next;   
 }  
  
 // re-link the original list to exclude the split portion  
 Node\* prev = list->head->prev;  
 list->head->prev = curr->prev;  
 list->head->prev->next = list->head;   
 //curr->prev->next = list->head;   
  
  
 // set up the new list  
 newList->head->next = curr;   
 newList->head->prev = prev;   
 curr->prev = newList->head;   
 prev->next = newList->head;  
  
 // update sizes  
 newList->size = list->size - index;   
 list->size = index;   
  
 return newList;   
}  
  
/\*\*  
 \* Compares two integers.  
 \* AI Use: No AI  
 \*/  
int compare\_int(void \*a, void \*b) {  
 return (\*(int\*)a - \*(int\*)b);  
}  
  
/\*\*  
 \* Compares two strings lexicographically.  
 \* AI Use: No AI  
 \*/  
int compare\_str(void \*a, void \*b) {  
 return strcmp(b, a);   
}  
  
/\*\*  
 \* Checks if the circular linked list is sorted in descending order.  
 \* AI Use: No AI  
 \*/  
bool is\_sorted(List\* list, Compare cmp) {  
 if (!list || list->size < 2) {  
 return true;   
 }  
  
 Node \*curr = list->head->next;   
 while (curr->next != list->head) {  
 if (cmp(curr->data, curr->next->data) < 0) {  
 return false;   
 }  
 curr = curr->next;   
 }  
 return true;  
}  
  
  
/\*\*  
 \* Generates a random string of lowercase letters.  
 \* AI Use: Assisted by AI  
 \*/  
char \*generate\_random\_string() {  
 int length = rand() % 11 + 5; // random length between 5 and 15.   
 char \*str = malloc((size\_t)(length + 1));  
 for (int i = 0; i < length; i++) {  
 str[i] = (char)('a' + rand() % 26); // random lowercase letter  
 }  
 str[length] = '\0';  
 return str;  
}  
  
/\*\*  
 \* Generates a list of random integers or strings based on the specified data type and length.  
 \* AI Use: Assisted by AI  
 \*/  
void generate\_list(List \*list, const char \*data\_type, size\_t length) {  
 for (size\_t i = 0; i < length; ++i) {  
 void \*data\_ptr;   
 if (strcmp(data\_type, "int") == 0) {  
 int \*data = (int \*)malloc(sizeof(int));  
 \*data = rand() % 100; // random integer between 0 and 99  
 data\_ptr = data;  
 } else {  
 data\_ptr = generate\_random\_string();  
 }  
 list\_append(list, data\_ptr);  
 }  
}

## 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 \*);  
  
/\*\*  
 \* @typedef Compare   
 \* @brief. Function pointer type for comparing two elements in a list. If a < b,   
 \* a negative integer is returned and vice versa.   
 \*/  
typedef int (\*Compare)(void\* a, void\* b);   
  
  
/\*\*  
 \* @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 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.  
 \*/  
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);  
  
/\*\*  
 \* @brief Sorts the list.   
 \* @param list Pointer to the list.  
 \* @param start Starting index of the list.   
 \* @param end Ending index of the list.   
 \* @param cmp Pointer to the Compare function.   
 \*/  
void sort(List \*list, size\_t start, size\_t end, Compare cmp);  
  
/\*\*  
 \* @brief Merges two lists.  
 \* @param list1 Pointer to the first list.   
 \* @param list2 Pointer to the second list.   
 \* @param cmp Pointer to the Compare function.  
 \*/  
List\* merge(List \*list1, List \*list2, Compare cmp);  
  
/\*\*  
 \* @brief Splits a list into two at the specified index.  
 \* @param list Pointer to the original list.  
 \* @param index Index at which to split the list.  
 \* @return Pointer to the new list containing elements from index to end, or NULL on failure.  
 \*/  
List\* split(List \*list, size\_t index);  
  
/\*\*  
 \* @brief Sorts integers in descending order.  
 \* @param a Integer to be compared.   
 \* @param b Integer to be compared.   
 \*/  
int compare\_int(void \*a, void \*b);   
  
/\*\*  
 \* @brief Sorts strings in lexicographical order.  
 \* @param a String to be compared.   
 \* @param b String to be compared.   
 \*/  
int compare\_str(void \*a, void \*b);   
  
/\*\*  
 \* @brief Returns if the list is sorted or not.  
 \* @return true is the list is sorted, false otherwise.   
 \*/  
bool is\_sorted(List\* list, Compare cmp);   
  
/\*\*  
 \* @brief Generates a random string of length between 5 and 15 characters.  
 \* @return Pointer to the newly allocated string. Caller is responsible for freeing it.  
 \*/  
char \*generate\_random\_string();  
  
/\*\*  
 \* @brief Generates a list of random integers or strings based on the specified data type and length.  
 \* @param list Pointer to the list to populate.  
 \* @param data\_type "int" for integers, "string" for strings.  
 \* @param length Number of elements to generate.  
 \*/  
void generate\_list(List \*list, const char \*data\_type, size\_t length);  
  
#endif // LAB\_H

## Source File: main.c

#include "lab.h"  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <pthread.h>  
  
#ifdef TEST  
#define main main\_exclude  
#endif  
  
typedef struct ThreadArgs {  
 List \*list;   
 size\_t start;   
 size\_t end;   
 Compare cmp;   
} ThreadArgs;  
  
// helper threadsafe function to call sort  
void \*threaded\_sort(void \*args) {  
 ThreadArgs \*targs = (ThreadArgs \*)args;   
 sort(targs->list, targs->start, targs->end, targs->cmp);   
 return NULL;   
}  
  
int main(int argc, char \*argv[]) {  
 if (argc != 3) {  
 fprintf(stderr, "Usage: %s <int|string> <length>\n", argv[0]);  
 return EXIT\_FAILURE;  
 }  
  
 const char \*data\_type = argv[1];  
 size\_t length = (size\_t)atol(argv[2]);  
  
 // populate list   
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 generate\_list(list, data\_type, length);  
  
 Compare cmp = (strcmp(data\_type, "int") == 0) ? compare\_int : compare\_str;  
  
 size\_t mid = length / 2;  
  
 pthread\_t thread1, thread2;  
 ThreadArgs args1 = {list, 0, mid, cmp};  
 ThreadArgs args2 = {list, mid, length, cmp};  
  
 pthread\_create(&thread1, NULL, threaded\_sort, &args1);  
 pthread\_create(&thread2, NULL, threaded\_sort, &args2);  
  
 pthread\_join(thread1, NULL);  
 pthread\_join(thread2, NULL);  
  
 List \*second\_half = split(list, mid);  
 List \*merged = merge(list, second\_half, cmp);  
 // print sorted list  
 for (size\_t i = 0; i < list\_size(merged); ++i) {  
 void \*data = list\_get(merged, i);  
 if (strcmp(data\_type, "int") == 0) {  
 printf("%d ", \*(int \*)data);  
 } else {  
 printf("%s ", (char \*)data);  
 }  
 }  
 printf("\n");  
  
 list\_destroy(merged, strcmp(data\_type, "int") == 0 ? free : free);  
 list\_destroy(second\_half, NULL); // already merged  
}

## Test Files

### lab-test.c

#include <stdlib.h>  
#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
#include <stddef.h>  
#include "harness/unity.h"  
#include "../src/lab.h"  
  
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);  
}  
  
void test\_compare\_int(void) {  
 int a = 10, b = 5, c = 10;  
 TEST\_ASSERT\_TRUE(compare\_int(&a, &b) > 0); // 10 > 5 → descending  
 TEST\_ASSERT\_TRUE(compare\_int(&b, &a) < 0); // 5 < 10  
 TEST\_ASSERT\_EQUAL(0, compare\_int(&a, &c)); // equal  
}  
  
void test\_compare\_str(void) {  
 char\* a = "apple";  
 char\* b = "banana";  
 char\* c = "apple";  
 TEST\_ASSERT\_TRUE(compare\_str(a, b) > 0); // "apple" < "banana"  
 TEST\_ASSERT\_TRUE(compare\_str(b, a) < 0); // "banana" > "apple"  
 TEST\_ASSERT\_EQUAL(0, compare\_str(a, c)); // equal  
}  
  
void test\_sort\_int(void) {  
 List\* list = list\_create(LIST\_LINKED\_SENTINEL);  
 int\* a = malloc(sizeof(int)); \*a = 5;  
 int\* b = malloc(sizeof(int)); \*b = 10;  
 int\* c = malloc(sizeof(int)); \*c = 7;  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
  
 sort(list, 0, list\_size(list), compare\_int);  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 list\_destroy(list, free);  
}  
  
void test\_sort\_str(void) {  
 List\* list = list\_create(LIST\_LINKED\_SENTINEL);  
 list\_append(list, strdup("banana"));  
 list\_append(list, strdup("apple"));  
 list\_append(list, strdup("cherry"));  
  
 sort(list, 0, list\_size(list), compare\_str);  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_str));  
  
 list\_destroy(list, free);  
}  
  
void test\_merge\_int(void) {  
 List\* list1 = list\_create(LIST\_LINKED\_SENTINEL);  
 List\* list2 = list\_create(LIST\_LINKED\_SENTINEL);  
  
int\* a = malloc(sizeof(int)); \*a = 10;  
int\* b = malloc(sizeof(int)); \*b = 8;  
int\* c = malloc(sizeof(int)); \*c = 9;  
int\* d = malloc(sizeof(int)); \*d = 7;  
  
list\_append(list1, a);  
list\_append(list1, b);  
list\_append(list2, c);  
list\_append(list2, d);  
  
sort(list1, 0, list\_size(list1), compare\_int);  
sort(list2, 0, list\_size(list2), compare\_int);  
  
List\* merged = merge(list1, list2, compare\_int);  
 size\_t msize = list\_size(merged);  
 for (size\_t i = 0; i < msize; ++i) {  
 int \*val = (int \*)list\_get(merged, i);  
 printf("merged[%zu] = %d\n", i, val ? \*val : 0);  
 }  
 TEST\_ASSERT\_EQUAL(4, list\_size(merged));  
  
 TEST\_ASSERT\_TRUE(is\_sorted(merged, compare\_int));  
  
 list\_destroy(list1, free);  
 list\_destroy(list2, free);  
 list\_destroy(merged, free);  
}  
  
void test\_merge\_str(void) {  
 List\* list1 = list\_create(LIST\_LINKED\_SENTINEL);  
 List\* list2 = list\_create(LIST\_LINKED\_SENTINEL);  
 list\_append(list1, "apple");  
 list\_append(list1, "banana");  
 list\_append(list2, "avocado");  
 list\_append(list2, "blueberry");  
  
 List\* merged = merge(list1, list2, compare\_str);  
 TEST\_ASSERT\_EQUAL(4, list\_size(merged));  
 TEST\_ASSERT\_TRUE(is\_sorted(merged, compare\_str));  
  
 list\_destroy(list1, NULL);  
 list\_destroy(list2, NULL);  
 list\_destroy(merged, NULL);  
}  
  
void test\_is\_sorted\_descending\_int(void) {  
 List\* list = list\_create(LIST\_LINKED\_SENTINEL);  
 int\* a = malloc(sizeof(int)); \*a = 10;  
 int\* b = malloc(sizeof(int)); \*b = 8;  
 int\* c = malloc(sizeof(int)); \*c = 5;  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));   
  
 list\_destroy(list, free);  
}  
  
void test\_split(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int\* a = malloc(sizeof(int)); \*a = 10;  
 int\* b = malloc(sizeof(int)); \*b = 20;  
 int\* c = malloc(sizeof(int)); \*c = 30;  
 int\* d = malloc(sizeof(int)); \*d = 40;  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
 list\_append(list, d);  
  
 List\* newList = split(list, 2);  
 TEST\_ASSERT\_NOT\_NULL(newList);  
 TEST\_ASSERT\_EQUAL(2, list\_size(list));  
 TEST\_ASSERT\_EQUAL(2, list\_size(newList));  
 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(newList, 0));  
 TEST\_ASSERT\_EQUAL\_PTR(d, list\_get(newList, 1));  
  
 list\_destroy(list, free);  
 list\_destroy(newList, free);  
}  
  
void test\_split\_sort\_merge(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int\* a = malloc(sizeof(int)); \*a = 30;  
 int\* b = malloc(sizeof(int)); \*b = 10;  
 int\* c = malloc(sizeof(int)); \*c = 40;  
 int\* d = malloc(sizeof(int)); \*d = 20;  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
 list\_append(list, d);  
  
 List\* newList = split(list, 2);  
 TEST\_ASSERT\_NOT\_NULL(newList);  
  
 sort(list, 0, list\_size(list), compare\_int);  
 sort(newList, 0, list\_size(newList), compare\_int);  
  
 List\* merged = merge(list, newList, compare\_int);  
 TEST\_ASSERT\_NOT\_NULL(merged);  
 TEST\_ASSERT\_EQUAL(4, list\_size(merged));  
 TEST\_ASSERT\_TRUE(is\_sorted(merged, compare\_int));  
  
 list\_destroy(list, free);  
 list\_destroy(newList, free);  
 list\_destroy(merged, free);  
}  
  
void test\_sort\_guard\_conditions(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*x = malloc(sizeof(int)); \*x = 1;  
 TEST\_ASSERT\_TRUE(list\_append(list, x));  
  
 size\_t before = list\_size(list);  
  
 /\* should return immediately (no crash / no modification) \*/  
 sort(NULL, 0, 1, compare\_int); // !list  
 sort(list, 1, 1, compare\_int); // start >= end  
 sort(list, 0, before + 1, compare\_int); // end > list->size  
 sort(list, 0, before, NULL); // !cmp  
  
 /\* verify list unchanged \*/  
 TEST\_ASSERT\_EQUAL(before, list\_size(list));  
 int \*val = (int \*)list\_get(list, 0);  
 TEST\_ASSERT\_NOT\_NULL(val);  
 TEST\_ASSERT\_EQUAL(1, \*val);  
  
 list\_destroy(list, free);  
}  
  
void test\_split\_guard\_conditions(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*a = malloc(sizeof(int)); \*a = 1;  
 int \*b = malloc(sizeof(int)); \*b = 2;  
 TEST\_ASSERT\_TRUE(list\_append(list, a));  
 TEST\_ASSERT\_TRUE(list\_append(list, b));  
  
 // NULL list should return NULL  
 TEST\_ASSERT\_NULL(split(NULL, 0));  
  
 // index == size (out of range) should return NULL  
 TEST\_ASSERT\_NULL(split(list, list\_size(list)));  
  
 // index > size should return NULL  
 TEST\_ASSERT\_NULL(split(list, list\_size(list) + 1));  
  
 // valid split for control: index within range should return a new list  
 List \*newList = split(list, 1);  
 TEST\_ASSERT\_NOT\_NULL(newList);  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
 TEST\_ASSERT\_EQUAL(1, list\_size(newList));  
  
 list\_destroy(list, free);  
 list\_destroy(newList, free);  
}  
  
void test\_split\_internal\_middle(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*a = malloc(sizeof(int)); \*a = 1;  
 int \*b = malloc(sizeof(int)); \*b = 2;  
 int \*c = malloc(sizeof(int)); \*c = 3;  
 int \*d = malloc(sizeof(int)); \*d = 4;  
  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
 list\_append(list, d);  
  
 /\* split in the middle: index 2 -> newList gets elements 2..end (0-based) \*/  
 List \*newList = split(list, 2);  
 TEST\_ASSERT\_NOT\_NULL(newList);  
  
 /\* original should have first two elements \*/  
 TEST\_ASSERT\_EQUAL(2, list\_size(list));  
 int \*v0 = list\_get(list, 0); TEST\_ASSERT\_NOT\_NULL(v0); TEST\_ASSERT\_EQUAL(1, \*v0);  
 int \*v1 = list\_get(list, 1); TEST\_ASSERT\_NOT\_NULL(v1); TEST\_ASSERT\_EQUAL(2, \*v1);  
  
 /\* newList should have remaining elements in order \*/  
 TEST\_ASSERT\_EQUAL(2, list\_size(newList));  
 int \*n0 = list\_get(newList, 0); TEST\_ASSERT\_NOT\_NULL(n0); TEST\_ASSERT\_EQUAL(3, \*n0);  
 int \*n1 = list\_get(newList, 1); TEST\_ASSERT\_NOT\_NULL(n1); TEST\_ASSERT\_EQUAL(4, \*n1);  
  
 list\_destroy(list, free);  
 list\_destroy(newList, free);  
}  
  
void test\_split\_index\_zero\_moves\_all\_nodes(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*a = malloc(sizeof(int)); \*a = 5;  
 int \*b = malloc(sizeof(int)); \*b = 6;  
 int \*c = malloc(sizeof(int)); \*c = 7;  
  
 list\_append(list, a);  
 list\_append(list, b);  
 list\_append(list, c);  
  
 /\* split at 0 should move all nodes to the new list, leaving original empty \*/  
 List \*newList = split(list, 0);  
 TEST\_ASSERT\_NOT\_NULL(newList);  
  
 TEST\_ASSERT\_EQUAL(0, list\_size(list));  
 TEST\_ASSERT\_TRUE(list\_is\_empty(list));  
  
 TEST\_ASSERT\_EQUAL(3, list\_size(newList));  
 int \*n0 = list\_get(newList, 0); TEST\_ASSERT\_NOT\_NULL(n0); TEST\_ASSERT\_EQUAL(5, \*n0);  
 int \*n1 = list\_get(newList, 1); TEST\_ASSERT\_NOT\_NULL(n1); TEST\_ASSERT\_EQUAL(6, \*n1);  
 int \*n2 = list\_get(newList, 2); TEST\_ASSERT\_NOT\_NULL(n2); TEST\_ASSERT\_EQUAL(7, \*n2);  
  
 list\_destroy(list, free);  
 list\_destroy(newList, free);  
}  
  
void test\_is\_sorted\_guard\_conditions(void) {  
 /\* NULL list -> should return true (guard) \*/  
 TEST\_ASSERT\_TRUE(is\_sorted(NULL, compare\_int));  
  
 /\* empty list -> should return true \*/  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 /\* single-element list -> should return true \*/  
 int \*v = malloc(sizeof(int)); \*v = 42;  
 TEST\_ASSERT\_TRUE(list\_append(list, v));  
 TEST\_ASSERT\_EQUAL(1, list\_size(list));  
 TEST\_ASSERT\_TRUE(is\_sorted(list, compare\_int));  
  
 list\_destroy(list, free);  
}  
  
void test\_is\_sorted\_detects\_unsorted\_pair(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 int \*a = malloc(sizeof(int)); \*a = 8;  
 int \*b = malloc(sizeof(int)); \*b = 10;  
  
 TEST\_ASSERT\_TRUE(list\_append(list, a));  
 TEST\_ASSERT\_TRUE(list\_append(list, b));  
  
 /\* cmp(a,b) = 8 - 10 = -2 < 0 -> should trigger the branch and return false \*/  
 TEST\_ASSERT\_FALSE(is\_sorted(list, compare\_int));  
  
 list\_destroy(list, free);  
}  
  
void test\_generate\_random\_string\_chars\_and\_length(void) {  
 for (int iter = 0; iter < 50; ++iter) {  
 char \*s = generate\_random\_string();  
 TEST\_ASSERT\_NOT\_NULL(s);  
  
 size\_t len = strlen(s);  
 TEST\_ASSERT\_TRUE(len >= 5 && len <= 15);  
  
 for (size\_t i = 0; i < len; ++i) {  
 TEST\_ASSERT\_TRUE(s[i] >= 'a' && s[i] <= 'z');  
 }  
 TEST\_ASSERT\_EQUAL\_CHAR('\0', s[len]);  
  
 free(s);  
 }  
}  
  
void test\_generate\_list\_ints(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
  
 /\* generate 10 integer entries \*/  
 generate\_list(list, "int", 10);  
 TEST\_ASSERT\_EQUAL(10, list\_size(list));  
  
 for (size\_t i = 0; i < list\_size(list); ++i) {  
 int \*v = (int \*)list\_get(list, i);  
 TEST\_ASSERT\_NOT\_NULL(v);  
 /\* basic sanity: dereference must be a valid int (no further range assumptions) \*/  
 (void)\*v;  
 }  
  
 list\_destroy(list, free);  
}  
  
void test\_generate\_list\_uses\_random\_string(void) {  
 List \*list = list\_create(LIST\_LINKED\_SENTINEL);  
 TEST\_ASSERT\_NOT\_NULL(list);  
  
 /\* generate 8 random strings (non-"int" branch) \*/  
 generate\_list(list, "str", 8);  
 TEST\_ASSERT\_EQUAL(8, list\_size(list));  
  
 for (size\_t i = 0; i < list\_size(list); ++i) {  
 char \*s = (char \*)list\_get(list, i);  
 TEST\_ASSERT\_NOT\_NULL(s);  
  
 size\_t len = strlen(s);  
 TEST\_ASSERT\_TRUE(len >= 5 && len <= 15);  
  
 for (size\_t j = 0; j < len; ++j) {  
 TEST\_ASSERT\_TRUE(s[j] >= 'a' && s[j] <= 'z');  
 }  
 }  
  
 list\_destroy(list, free);  
}  
  
  
  
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);  
 RUN\_TEST(test\_compare\_int);  
 RUN\_TEST(test\_compare\_str);  
 RUN\_TEST(test\_sort\_int);  
 RUN\_TEST(test\_sort\_str);  
 RUN\_TEST(test\_merge\_int);  
 RUN\_TEST(test\_merge\_str);  
 RUN\_TEST(test\_is\_sorted\_descending\_int);  
 RUN\_TEST(test\_split);  
 RUN\_TEST(test\_split\_sort\_merge);  
 RUN\_TEST(test\_sort\_guard\_conditions);  
 RUN\_TEST(test\_split\_guard\_conditions);  
 RUN\_TEST(test\_split\_internal\_middle);  
 RUN\_TEST(test\_split\_index\_zero\_moves\_all\_nodes);  
 RUN\_TEST(test\_is\_sorted\_guard\_conditions);  
 RUN\_TEST(test\_is\_sorted\_detects\_unsorted\_pair);  
 RUN\_TEST(test\_generate\_random\_string\_chars\_and\_length);  
 RUN\_TEST(test\_generate\_list\_ints);  
 RUN\_TEST(test\_generate\_list\_uses\_random\_string);  
 return UNITY\_END();  
}```  
  
  
  
---  
  
## README  
```markdown  
# Project 2 - Multithreaded Sorting Application  
  
- Name: Emily Newton  
- Email: emilynewton@u.boisestate.edu  
- Class: 452-002  
  
## Known Bugs or Issues  
  
No known bugs or issues!   
  
## Experience  
  
This project was such a doozy for me. I was struggling to understand what the instructions wanted. I couldn't understand how to split the list, and if that was even the right thing to do. The second task said that there was only supposed to be one list the threads were fired on, but then the merge function took in two list pointers as the parameters. I just decided to make a split function in my lab.c and split after generating a random list. The threading made sense to me though. Luckily, I have worked with threads before. I chose to do an insertion sort without any rhyme or reason, but it worked out well. I also added a function to generate a random string for data in the randomized linked list and then a function to generate a linked list with random data whether it be integers or strings. This made it really easy to work with in regards to the command arguments and cleaned up my main.c. My favorite part was learning about function pointers. I didn't know that it was possible to pass in one function but have it mean two different things depending on the context. I had the compare\_int and compare\_str under my Compare function pointer and it was cool that I could just use that function pointer without having to switch between the two functions.

## End of Report

Report generated on 09/30/2025 at 17:09:01

## GitHub Info

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

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

0d9be9303ff6cc0ae81b5c1fd6cafd6a781942c0bc705a6d9942e4e995bb78da submission-report.md 94d105ef865785ff7c79933a1a8d5ad366ccba08932f55bac5ea5952ba666fc8 submission-report.md cb7e2d01bdb8ca6b1e479cd305fe5e4d8bb61809066c86976b3140afdeee24d3 submission-report.md e9400931b27189893c1ebffa0ce7cdb578606a67d26203ee57ed57d02d2c622a submission-report.md a70a94cbb8cf7ab6bf0613ae841b4ac3d429f66b612e0792d1c62939ca14b9a7 submission-report.md 5574ce089be2e71e90586cc239335c54865b6542515da37e4ba9f50c75c12339 submission-report.md 88677025c3b9d7ec42f6b05b1df56885b5ee198b28bc1b2909f573f0017bb425 submission-report.md 3a9a2694f178321eb8e0478394f547be47eb8f9fdc093656279ab0dc9aa90b94 submission-report.md cd6a5c65aff87e2f8320806cab9d50c4583cce2bb87c01763d9a821a492ecba6 submission-report.md 2f4122fa098261e817910dfe58712542ea4a7614b4ed1cae87a2f2ed61c832a7 submission-report.md