Linked List

Sprint Report

Team: Partigiano

1 Introduction

The following article is the report of a Software Engineering project, that took place in 2025. The source code and file files can also be found in Github,

1.1 Purpose

The purpose of the project is to develop a robust ***Linked List***. The project will be designed in such a way that it is maintainable and extendable, as well fully documented. This shall result in fast and correct development and deployment of new features in the future. Another target of the project was to implement agile methods to develop software. In our case, I used **scrum.**

1.2 Document Structure

The rest of this document is structured as follows:

- Section 2 specifies the technologies used during the development of the project.
- Section 3 describes our Scrum team and specifies this Sprint's backlog.
- Section 4 dives into the Use Cases, which derive from User Stories.
- Section 5 explains the project's Architecture.
- Section 6 has an example run of application and its tests.

2 Tools and Technologies

- **C**: A high-level, general-purpose **programming language**. By design, C's features cleanly reflect the capabilities of the targeted **CPUs**.
- **CLion**: An IDE for developing computer software written in C.
- **CUnit**: A **unit testing framework** for C.
- **GitHub**: A **developer platform** that allows development teams to create, store and manage their code. It uses and extends the Git software.

3 Scrum team and Sprint Backlog

3.1 Scrum team

Product Owner	Alex Pournaras
Scrum Master	Alex Pournaras
Development Team	Alex Pournaras

3.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories	Description
1	21/3/2025	22/3/2025	0.2	-	Determine the tools and technologies to be used to build the project. Create use cases, domain classes.
2	22/3/2024	27/3/2025	0.7	US1, US2, US3, US4, US5	Implement core functionalities of LinkedList, Node, ErrorHander.
3	27/03/2025	30/03/2025	0.5	-	Perform tests for each and every source file.

4 Use Cases

4.1 Create Linked List

Use case ID	UC1
Actors	User
Preconditions	-
Main flow of events	1. The use case starts when the user calls the **create()** function.
Post conditions	2. A new **Linked List** has been **created** with empty contents.
Alternative flow 1	1. There is no space in the *heap* to allocate memory.
Post conditions	The programs exits, throwing an error message.

Add Node

Use case ID	UC2
Actors	User
Pre conditions	The user has created a Linked List.
Main flow of events	3. The use case starts when the user calls an **add_node** function. 3.1. add_at_head() 3.2. add_at_tail() 3.3. add_at_index() 3.3.1. The user will define the index of the insertion, default: head.
Post conditions	4. A new **node** has been **inserted** to the Linked List.
Alternative flow 1	5. There is **no space** in the ***heap*** to allocate memory .
Post conditions	6. The programs exits, throwing an error message.
Alternative flow 2	7. The user-provided **index is out of bounds**. 7.1. The software asks the user to dictate its next action 7.1.1. Ignore and do nothing. 7.1.2. Insert at head. 7.1.3. Insert at tail.
Post conditions	8. The software throws out of bounds message. 9. The node is inserted or not, depending on user's choice.

Delete Node

Use case ID	UC3
Actors	User
Pre	The user has created a Linked List. For the sake of the main flow, it will have to

conditions	be non-empty.
Main flow of events	10. The use case starts when the user calls a delete function. 10.1. delete_at_head(). 10.2. delete_at_tail(). 10.3. delete_at_index(). 10.3.1. It's matter mandatory for the user to define the index.
Post conditions	11. The corresponding node has been deleted.
Alternative flow 1	12. The user-provided **index is out of bounds**.
Post conditions	13. No effect to the data structure, a message is thrown, informing the user about that.

Get Node

Use case ID	UC4
Actors	User
Pre conditions	The user has created a Linked List. For the sake of the main flow, it will have to be non-empty.
Main flow of events	14. The user calls the **get_node()** function. It is mandatory for the user to define the index.
Post conditions	15. The corresponding **node** is being **returned**.
Alternative flow 1	16. The user-provided **index is out of bounds**.
Post conditions	17. Void is being returned. A message is thrown, informing the user about that

5 Design

5.1 Architecture

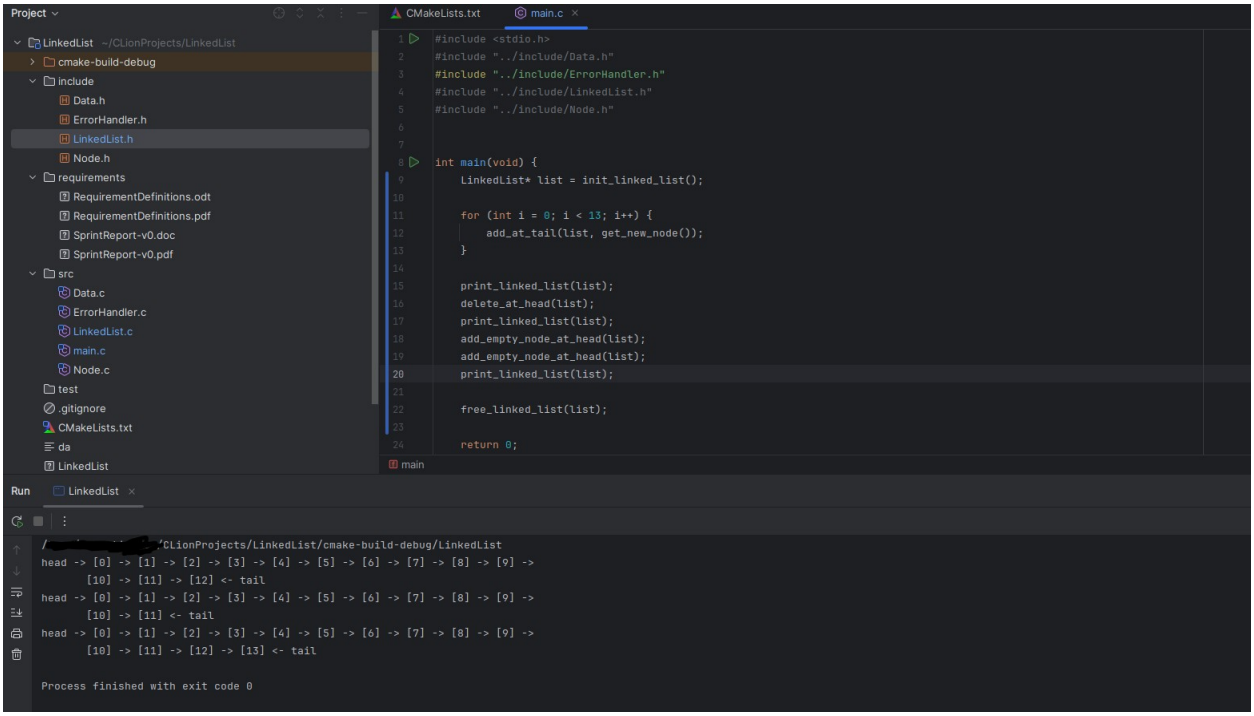
The project consists mainly of **4 source files.**

1. **Data**: This source file (currently only defines the Data struct), has the data to be attached to the nodes.
2. **Node**: This source file, is responsible for the creation of Nodes. Creates empty node, attaches data to it, and, of course, points to another node.
3. **LinkedList**: The core source file of this project. Includes all the functionality of creating and maintaining a one-way linked list.
4. **ErrorHandler**: To flexibly handle all kind of **RunTimeErrors** or **BadUserInputs** that might occur.

6 Execution Example

6.1 Program Run

Attach of a simple use of the linked list


Project
LinkedList ~/CLionProjects/LinkedList
cmake-build-debug
include
Data.h
ErrorHandler.h
LinkedList.h
Node.h
requirements
RequirementDefinitions.odt
RequirementDefinitions.pdf
SprintReport-v0.doc
SprintReport-v0.pdf
src
Data.c
ErrorHandler.c
LinkedList.c
main.c
Node.c
test
.gitignore
CMakeLists.txt
da
LinkedList
CMakeLists.txt
main.c
#include <stdio.h>
#include "../include/Data.h"
#include "../include/ErrorHandler.h"
#include "../include/LinkedList.h"
#include "../include/Node.h"
int main(void) {
LinkedList* list = init_linked_list();
for (int i = 0; i < 13; i++) {
add_at_tail(list, get_new_node());
}
print_linked_list(list);
delete_at_head(list);
print_linked_list(list);
add_empty_node_at_head(list);
add_empty_node_at_head(list);
print_linked_list(list);
free_linked_list(list);
return 0;
main
Run
LinkedList
/CLionProjects/LinkedList/cmake-build-debug/LinkedList
head -> [0] -> [1] -> [2] -> [3] -> [4] -> [5] -> [6] -> [7] -> [8] -> [9] ->
[10] -> [11] -> [12] <- tail
head -> [0] -> [1] -> [2] -> [3] -> [4] -> [5] -> [6] -> [7] -> [8] -> [9] ->
[10] -> [11] <- tail
head -> [0] -> [1] -> [2] -> [3] -> [4] -> [5] -> [6] -> [7] -> [8] -> [9] ->
[10] -> [11] -> [12] -> [13] <- tail
Process finished with exit code 0

6.2 Test Run

Two (2) attachments of the tests run

```
Suite: node_tests
  Test: test_get_new_node ...passed
  Test: test_is_not_null ...passed
  Test: test_attach_data_to_node ...passed
Suite: linked_list
  Test: test_init_linked_list ...passed
  Test: test_is_empty ...passed
  Test: test_add_at_head ...passed
  Test: test_add_at_tail ...passed
  Test: test_add_at_index ...passed
  Test: test_delete_at_head ...passed
  Test: test_delete_at_tail ...passed
  Test: test_delete_at_index ...The node you are trying to delete is out of bounds. Nothing to delete.
passed
  Test: test_add_at_index for out_of_bounds handle ...passed
Suite: error_handler
  Test: test_set_out_of_bounds_handler ...passed
  Test: test_handle_null_object_error ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      3      3    n/a      0        0
               tests     14     14     14      0        0
             asserts     40     40     40      0      n/a

Elapsed time =    0.000 seconds

Process finished with exit code 0
```

```
 Run Summary:    Type  Total    Ran Passed Failed Inactive
               suites      3      3    n/a      0        0
                tests     14     14     14      0        0
              asserts     40     40     40      0      n/a

 Elapsed time =    0.000 seconds
```