Written by Gabriel Nepomuk Hoyer

Introduction

This report is made to explain the process of solving an assignment for INF1100. The assignment is to implement a linked list in a program and to finish a garbage collector using AI. The report includes a technical explanation of linked lists and garbage collectors as well as my experiences with implementing them in this particular system.

Technical background

The assignment is to implement a liked list in a program. The program is a simulation of bacterial mutation and antibiotic resistance, but that is outside the scope of this report. The assignment also includes finishing a garbage collector that the simulation makes use of through the help of AI.

Linked list implementation

The file for the assignment includes a header file for a liked list that is completed to make the implementation of the linked list easier. It predefines the names used in the code for the simulation so that I only had to create the logic part of the functions and not implement them into the simulation code.

What is a linked list?

A linked list is a sequence of “nodes” which contain a data point and a pointer to the next node. Superficially it is similar to an array, but it differs in memory allocation. An array allocated a block of memory that is stored sequentially in the RAM. As a result of this one must allocate a block of memory that fits the whole array before you can fill it. A linked list is not limited by any initial allocation of memory because each node points to the next node in the sequence independent of its location in memory. Some of the other perks of a linked list compared to an array is that you can add a new node anywhere in the sequence by changing the pointer of the previous node and following node.

What are the components of a linked list?

A linked list is made up of a sequence of nodes that are split into two parts, one part contains a stored data point and the other contains a pointer to the next node’s data point.

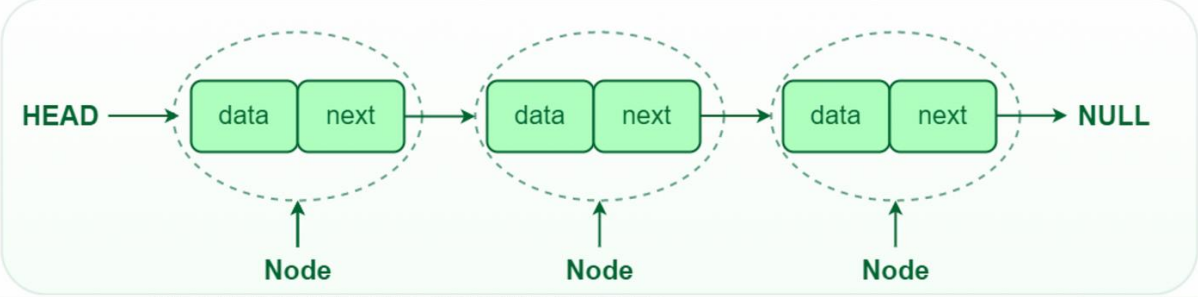

Figure 1, https://blog.stackademic.com/single-linked-lists-unveiled-a-step-by-step-guide-to-understanding-and-implementation-309587911bd6

In the picture above you can see a visual representation of the basics of how a linked list works.

Some other important components of a linked list are:

- Head/start: A variable you save that represents the start of the list, needs to be updated as you add more nodes to the start of the list.
- Tail/end: Technically not a required part of a linked list, but very useful when you want to add something to the end of the list
- Length variable: A useful variable that contains the number of nodes in the list.

Garbage collector

A garbage collector manages the memory that a program no longer needs. It takes in the garage data that the program sends it and deallocates it from the memory. In this assignment the garbage collector is mostly finished but is lacking one part that is supposed to be finished with the help of artificial intelligence.

Design and implementation

Here I will discuss the design and implementation of both the linked list and the garbage collector. The linked list is mostly made and designed by me, whilst the garbage collector is exclusively premade and finished by AI.

Implementation of the linked list

As mentioned earlier in the report, the implementation into the actual simulation is premade and the only part that is made by me is the functions that the linked list is built upon. All the function calls to use the list are untouched by me.

What is the structure of my linked list?

I have made a singly linked list that makes use of a head, tail and a variable containing the length of the list. Due to the way the header file is set up the creation of the linked

list is split up into several different parts. I will now explain the steps the program goes through to make a list.

1. Creating an empty list
   Through the list_create function the program creates an empty list that has a length of 0, a head pointing to NULL and a tail also pointing to NULL.
2. Populating the empty list
   Through the functions list_addfirst and list_addlast the program takes input in the form of a list that data should be added into and a data point that the list is supposed to store.
3. Deleting elements from the list
   Through the function list_remove the program takes input in the form of a list that data should be removed from, and a data point used to locate the element that should be removed.

The program also contains a function that returns the length of any given list once it's called upon.

The iterator

The program contains an iterator that is used to access the different elements in the list. The iterator in this program is quite basic and can only move from one object to another in chronological order. It starts of at the head of the list and moves one step towards the end of the list when a function called "list_next" is called upon. This function is also the only part of the iterator that returns a datapoint from the list.

The iterator also includes two other functions that resets the iterator to the head of the list and frees the memory the allocator uses.

Garbage collection

For the garbage collector I copied the code in gc.c and asked it to complete it. When I was debugging using GDB I was lead down a rabbit hole of thinking my segmentation fault originated from the code that ChatGPT made, and I spent quite some time giving different prompts trying to make the code work. In hindsight the problem was with my own code and the original code that ChatGPT had made was okay.

Important details of the process of implementing my system

Due to my own limited knowledge of c as a programming language my initial code contained a lot of problematic code. The initial code compiled, but when I ran the program, it crashed from a segmentation fault. To start my troubleshooting I had to learn what a segmentation fault was. Once I researched a bit, I started looking over my memory allocation in search of a bug. Through the help of ChatGPT I cleaned up some sloppy mistakes where I defined nodes the opposite way around, added checks for incorrect input of empty lists and checks for malloc failing to allocate memory. Through

my troubleshooting using ChatGPT I also cleaned up some logic faults I had overlooked during my manual debugging.

After this I still got segmentation faults when I tried to run the code. I tried to comment out different parts of the code to locate in which function the segmentation fault was located. When I had tried commenting out every function and running the program, still getting segmentation faults with every attempt, I tried using GDB to debug. This probably should have been higher up on my list for potential solutions, but in this case, it led me down a rabbit hole of trying to correct the ”gc_free” function in the garbage collector. I’m still not quite certain where the segmentation faults originated from, but I suspect there were several different ones.

Through the many changes I made to the code I managed to make it close enough to correct that I were able to locate my bug by manually exchanging every function for code I knew for a fact worked and eventually finding the function that contained the last (to my knowledge) major bug.

Discussion

The perks and downsides of the way I chose to solve the assignment can be separated into four different categories:

Time management

I’ll be the first to admit that my time management for this assignment was a mess. I overestimated my own ability and ended up missing the deadline due to my own arrogance. It most definitely had an impact on the quality of my work, and due to cramming the work into a few days instead of a few weeks it most likely affected my retention of new information.

On the bright side it was a good reminder that I should start working as assignments as soon as possible instead of waiting until the deadline is closing in. It also helped me remember that I shouldn’t overestimate my own ability.

Debugging

The way that I chose to debug was suboptimal. I should have created a simple program where I tested the functions practically to see if they were working as intended. I ended up brute forcing until the program ran without any obvious errors.

Using AI to learn

Using AI to learn can be an incredibly powerful tool, but it can easily be abused so that you end up with a working result without having learned why it works. I believe the way that I used AI falls under the first category, but I could also have benefited from diversifying the external tools I made use of during the solving process. A very basic tool I didn’t make as much use of as I should have done is the GDB debugger. I hadn’t really

sat down to learn how it works and was therefore not able to use it in a way that benefited my debugging.

Research

One of the things I believe I did quite well for this assignment was my research prior to starting the actual coding. I spent quite a lot of time studying up on linked lists and how to implement them into programs. I did however skip an essential step during my research phrase. I overly focused on the theoretical implementation and didn't spend enough time on implementing linked lists in simple programs and learning the practical side of the implementation before I started coding in the assignment.

Conclusion

Throughout this assignment I learnt a lot about linked lists and a little bit about garbage collectors. Through putting my newly learnt knowledge of linked lists into writing in this report and implementing one into a system I broadened my understanding and crystalized my knowledge.

One of the most important experiences I gained throughout the assignment was within debugging a system. I did a lot of things wrong during the solving process and through reflecting on my mistakes I realized better ways to implement code into a premade system and debugging my own code.