John Gill

Dr. Tala Talaei Khoei

CS 5008

8 June 2024

Assignment 1

**Question 1**: Explain how memory is allocated and deallocated in a linked list. Also, provide examples of real-world applications where linked lists are preferred over arrays.

Linked lists use dynamic memory allocation, meaning memory is evenly allocated to all nodes in a linked list during its runtime, not before the program is created. The memory for the linked list is deallocated once the process calling upon it has finalized, meaning the list ceases to exist. Linked lists can be preferable to arrays in programs that use a lot of memory, and don’t have extra memory to spare for an array. Because of that, a linked list would be preferable to an array if we don’t know how much memory is necessary for the program before going into it. Say we wanted to write a large program that processes student data, and we want a small portion of it to be an algorithm that prints out all the students in a graduating class; a linked list would be useful because we could add all the students to it while the program is running, before we know the exact amount of memory needed for such a list, so that we don’t take away from the memory required for the other parts of the program.

**Question 2**: Discuss why stack is recursive data structure?

Stack is a recursive data structure because it adds new elements, and removes existing elements from the top – i.e. it follows “FILO”: first-in-last-out. If someone using stack wanted a specific element located somewhere in the middle of it, the stack would have to pop elements off the top recursively until the specified element appeared on top.

**Question 3**: Are linked lists considered linear or non-linear Data Structures? Explain your response.

Linked lists are linear data structures because they are arranged linearly. When accessing data within them, the compiler iterates through each value by looking at the node, then the pointer to the next node until it reaches the desired value.

**Question 4**: Define a priority queue. How is it different from a regular queue? How would you

handle situations where a queue becomes full or empty?

A priority queue arranges elements according to the priority each element carries. Elements are assigned priority values and placed in the queue according to them. A regular queue, on the other hand, follows a first-come-first-serve model, otherwise known as “FIFO”: first-in-first-out. In the case that a priority queue is full, we could either resize the queue, remove elements from the queue, or reject new elements being added to it. If the queue is empty, then we can block operations that call upon it, return null from it, or return an exception to indicate that it’s empty.

**Question 5**: Create a program that uses pointers for enqueue and dequeue operations in a queue.

Program is in the .zip file.

**Question 6**:Write a C function to detect and remove duplicates from an unsorted linked list.

Program is in the .zip file.