Name : Jenibelle Wievin

NIM :2602105632

Major : Data Science

Subject : Data Structure

1. **A. Array:** An array is a collection of elements stored at contiguous memory locations. Implementation is straightforward, involving declaring a fixed size and accessing elements using indices. Arrays provide constant time complexity O(1) for access but resizing them is costly O(n) as it involves copying elements to a new array.

**Linked List:** A linked list consists of nodes, each containing data and a reference to the next node. Unlike arrays, linked lists do not require contiguous memory allocation. Implementation involves defining a node structure and managing pointers for insertion, deletion, and traversal. Linked lists provide O(1) complexity for insertions and deletions if the pointer is known, but access time is linear O(n).

**B. Single Linked List:** Nodes in a singly linked list have references to the next node only. This makes traversal unidirectional. Implementation is simpler but reversal operations are complex.

**Double Linked List:** Each node in a doubly linked list has references to both the next and the previous nodes, allowing bidirectional traversal. Although it uses more memory, the implementation is versatile, making insertion and deletion operations easier as they can be performed from both ends.

**C. Stack:** A stack is a LIFO (Last In, First Out) data structure. Implementation involves using an array or linked list with operations restricted to the top of the stack, providing O(1) time complexity for push and pop operations.

**Queue:** A queue is a FIFO (First In, First Out) data structure. It can be implemented using arrays or linked lists, with enqueuing at the rear and dequeuing at the front. Operations are O(1) for both enqueue and dequeue with a linked list.

**D. Static Memory Allocation:** Memory size is determined at compile time. This method is straightforward but inflexible, as the size cannot be changed during runtime.

**Dynamic Memory Allocation:** Memory is allocated during runtime using functions like malloc() and free() in C. This provides flexibility, allowing programs to allocate memory as needed, but requires careful management to avoid memory leaks and fragmentation.

1. Postfix : A B % C / D E F \* + G - H \* + I –

Prefix : - + / % A B C \* - + D \* E F G H I

1. 

A screenshot of a computer program

Description automatically generated