**Stack Using Linked List Data Structure**

**Definition**

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. In a linked list implementation of a stack, each element is represented by a node, which is connected sequentially. The top of the stack is represented by the head of the linked list.

**Operations**

1. **Push**: Add an element to the top of the stack (i.e., insert a node at the beginning of the linked list).
2. **Pop**: Remove and return the top element from the stack (i.e., delete the head node of the linked list).
3. **Peek/Top**: Return the top element without removing it (i.e., return the data of the head node).
4. **isEmpty**: Check if the stack is empty (i.e., check if the head node is null).

**Pros and Cons**

**Pros:**

* **Dynamic Size**: The stack can grow and shrink as needed, so there is no need to define a fixed size.
* **Efficient Memory Usage**: Memory is allocated as needed, so no wasted space exists.
* **No Overflow**: There is no risk of overflow unless the system runs out of memory.

**Cons:**

* **Memory Overhead**: Each element in the stack requires additional memory for the pointer/reference.
* **Complexity**: Linked list operations are generally more complex than array operations due to the use of pointers/references.
* **Cache Performance**: Linked lists have poorer cache performance compared to arrays because nodes are not stored contiguously in memory.

**Applications**

* **Function Call Management**: Used to keep track of function calls and return addresses.
* **Expression Evaluation**: Used in parsing and evaluating expressions (e.g., converting infix to postfix).
* **Undo Mechanism**: Used in software applications to keep track of the history of operations for undo functionality.
* **Backtracking**: Useful in algorithms that require backtracking, such as solving mazes, puzzles, etc.
* **Syntax Parsing**: Used in compilers for syntax parsing of programming languages.