## **LEC-6 Stack**

# **Stack: Overview**

A Stack is a fundamental data structure widely used in computer science and programming. It follows the Last-In-First-Out (LIFO) principle, meaning that the last element added to the stack is the first one to be removed. Stacks have various applications, ranging from expression evaluation to managing function calls.

# **Characteristics of a Stack**

Stacks possess several key characteristics that define their behavior:

* **LIFO Principle:** The last item added to the stack is the first one to be removed.
* **Two Main Operations:** Stacks support two primary operations: "push" to add an element and "pop" to remove the top element.
* **Peek Operation:** The "peek" or "top" operation retrieves the top element without removing it.

# **Stack Implementation**

Stacks can be implemented using various data structures, but two common approaches are:

* **Array Implementation:** Using a dynamic or fixed-size array to store stack elements.
* **Linked List Implementation:** Utilizing a linked list where each node holds an element and a pointer to the next node.

# **Stack Operations**

**1. Push Operation**

Adding an element to the top of the stack:

* Allocate memory (for linked list implementation) or check for overflow (for array implementation).
* Place the new element at the top of the stack.
* Update the stack's top pointer.

**2. Pop Operation**

Removing the top element from the stack:

* Check for underflow (empty stack).
* Remove the top element.
* Update the stack's top pointer.

**3. Peek Operation**

Viewing the top element without removal:

* Return the element at the top of the stack.
* No changes to the stack's structure.

# **Common Applications**

Stacks find applications in various domains:

* **Function Call Management:** Stacks manage function calls and their local variables in programming languages.
* **Expression Evaluation:** Stacks can evaluate mathematical expressions, converting infix notation to postfix or prefix.
* **Undo/Redo Mechanisms:** Stacks are used to implement undo and redo functionalities in applications.
* **Depth-First Search (DFS):** Stacks are employed in graph traversal algorithms like DFS.

# **Limitations**

Stacks have certain limitations:

* **Fixed Size (for array implementation):** Arrays have a predetermined size, potentially leading to overflow or underflow.
* **Dynamic Memory Allocation (for linked list implementation):** Linked list nodes require memory allocation, leading to potential memory fragmentation.

# **Conclusion**

In summary, a Stack is a crucial data structure that operates on the Last-In-First-Out principle. Its push, pop, and peek operations make it valuable for various applications, including function call management, expression evaluation, and graph traversal. However, its choice of implementation (array or linked list) comes with its own set of trade-offs and considerations.



Stack

#include <iostream>

using namespace std;

//Stack (last-in ---first-out)

class stack{

private:

    int maxsize;

    int currsize;

    int top;

    int\* arr;

public:

    stack(int n=10){

        maxsize=n;

        currsize=0;

        top=-1;

        arr=new int [maxsize];

    }

    ~stack(){

        if(arr!=0){

            delete [] arr;

            arr=0;

        }

    }

    bool isEmpty(){

        if(currsize==0){

            return 1;

        }

        return 0;

    }

    bool isFull(){

        if(currsize==maxsize){

            return 1;

        }

        return 0;

    }

    bool pop();

    bool push(int n);

};

    bool pop(){

        if(isEmpty()){

            return false;

        }

        else{

            top--;

            currsize--;

            return true;

        }

    } // delete from stack

    bool push(int n){

        if(isFull()){

            return false;

        }

        else{

            top++;

            arr[top]=n;

            currsize++;

            return true;

        }

    } // insert in stack