|  |  |  |
| --- | --- | --- |
|  | **School of Engineering & Technology** | |
| **Department: SOET** | **Session: 2023-2024** |
| **Program: B.Tech Computer Science Engineering** | **Semester: III** |
| **Course Code:** | **Number of students:** |
| **Course Name:** - **Data Structure** | **Faculty: Ms. Suman** |
| **Lab Date** | **Batch:** |

**Lab Assignment 1**

**Learning Outcomes:**

* Gain insight into the stack's characteristics, including its Last In First Out (LIFO) behavior and its role in data management.
* Learn to define and implement a stack Abstract Data Type (ADT) with basic operations like push and pop, fostering programming proficiency.
* Apply object-oriented principles by creating stack objects and invoking member.
* Explore real-world applications through exercises.

**Aim**:

Exploring Stack Data Structure: Understanding, Implementation, and Practical Applications

**Objectives:**

* **Comprehend Stack Data Structure and its fundamental operations.**
* **Master the process of defining Stack ADT and executing basic operations.**
* **Demonstrate proficiency in creating objects from ADT and utilizing member functions.**
* **Apply stack concepts to implement various practical applications.**

**Introduction:**

**In Stacks are fundamental data structures in computer science with a simple yet powerful concept: Last In First Out (LIFO). Imagine a stack of books placed one on top of the other — the last book added is the first one removed. This characteristic makes stacks incredibly useful in a wide range of applications, from programming languages and algorithmic problem-solving to everyday scenarios like managing browser history.**

**In programming, stacks serve as a vital tool for managing function calls, evaluating expressions, and implementing undo mechanisms. They provide an efficient way to store and access data, facilitating tasks such as backtracking algorithms and syntax parsing.**

**Understanding stacks and their operations, such as push (adding an item) and pop (removing an item), is crucial for any programmer. By mastering the principles of stacks, programmers gain a valuable skillset that can be applied to various computational tasks and problem-solving scenarios.**

A Stack is an ordered list in which insertion and deletion are made at one end called the Top. The restrictions on the stack imply that if we add the elements 1,2,3,4,5 to the stack in that order, then 5 is the first element we delete from the stack. The add operation is called “Push”, and the delete operation is called “Pop”. Since the last element pushed into the stack is the first element popped, a stack is also known as a Last In First Out (LIFO) List

**In this lab, we delve into the world of stacks, exploring their structure, operations, and practical applications. Through hands-on exercises and implementations, participants will not only grasp the concepts of stacks but also understand their significance in software development and beyond.**

**Algorithm:**

**Algorithm Stack\_PUSH(item)**

Step-1: If TOP = Max-1

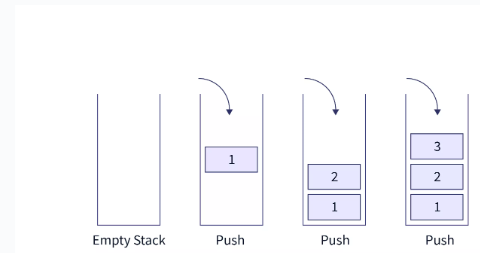
Print “Overflow”

Goto Step 4

Step-2: Set TOP= TOP + 1

Step-3: Set Stack[TOP]= ELEMENT

Step-4: END



**Algorithm Stack\_POP(item)**

Step 1: START

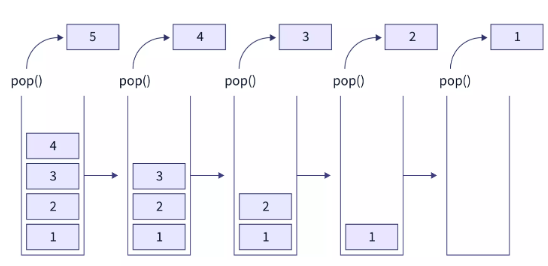
Step 2: if top==-1

then Write “Stack is Underflow”

Step 3: otherwise

print “deleted element”

top=top-1;

Step 4: END

**Sample Code:**

#include <iostream>

using namespace std;

// Define the maximum size of the stack

#define MAX\_SIZE 100

class Stack {

private:

int top; // Index of the top element

int items[MAX\_SIZE];// Array to store stack elements

public:

Stack() {

top = -1; // Initialize top to -1 indicating an empty stack

}

// Function to check if the stack is empty

bool isEmpty() {

return top == -1;

}

// Function to check if the stack is full

bool isFull() {

return top == MAX\_SIZE - 1;

}

// Function to push an element onto the stack

void push(int element) {

if (isFull()) {

cout << "Stack Overflow. Cannot push element.\n";

return;

}

items[++top] = element; // Increment top and add element to the stack

cout << "Element " << element << " pushed onto the stack.\n";

}

// Function to pop an element from the stack

int pop() {

if (isEmpty()) {

cout << "Stack Underflow. Cannot pop element.\n";

return -1;

}

int poppedElement = items[top--]; // Get the top element and decrement top

cout << "Element " << poppedElement << " popped from the stack.\n";

return poppedElement;

}

};

int main() {

Stack stack;

// Pushing elements onto the stack

stack.push(10);

stack.push(20);

stack.push(30);

// Popping elements from the stack

stack.pop();

stack.pop();

stack.pop();

stack.pop(); // Trying to pop from an empty stack

return 0;

}

**Output of the Program:**

Element 10 pushed onto the stack.  
Element 20 pushed onto the stack.  
Element 30 pushed onto the stack.  
Element 30 popped from the stack.  
Element 20 popped from the stack.  
Element 10 popped from the stack.  
Stack Underflow. Cannot pop element.

**Analysis:**

* **Efficiency:** Both push and pop operations have O(1) time complexity, making them efficient for frequent operations.
* **Error Handling**: Proper management of stack limits prevents data structure overflow and underflow, critical for robust software.
* **Stack Behavior:** The LIFO nature of the stack is evident as the last pushed item is the first to be popped.

## Applications:

* Function Call Management: Stacks are extensively used in programming languages to manage function calls. Each function call adds a new stack frame, allowing for easy tracking of function execution and enabling functions to return to their respective callers.
* Expression Evaluation: Stacks play a crucial role in evaluating expressions, such as arithmetic expressions or postfix expressions. They facilitate the parsing and evaluation process by storing operands and operators in a structured manner.
* Undo Mechanisms: Many applications utilize stacks to implement undo functionalities. Each action performed is pushed onto the stack, allowing users to revert to previous states by popping actions off the stack.
* Backtracking Algorithms: Stacks are instrumental in backtracking algorithms, such as depth-first search (DFS). They store the state of the search, enabling the algorithm to backtrack efficiently when necessary.
* Syntax Parsing and Compilation: Stacks are utilized in syntax parsing and compilation processes, particularly in parsing context-free grammars. They help in tracking the structure of the code being parsed and ensuring syntactic correctness.
* Browser History Management: Stacks are employed in managing browser history. Each visited URL is pushed onto the stack, allowing users to navigate backward through their browsing history.
* Task Scheduling: Stacks are used in task scheduling algorithms, such as Depth-First Scheduling (DFS). They store the sequence of tasks to be executed, facilitating the scheduling process.
* Memory Management: Stacks are used in memory management systems, particularly for managing function call stacks and local variables. They allocate memory dynamically during function calls and deallocate it when functions return.
* Expression Conversion: Stacks are utilized in converting between different forms of expressions, such as infix to postfix or prefix notation. They facilitate the rearrangement of operands and operators according to specific rules.
* Algorithmic Problem Solving: Stacks serve as fundamental data structures in various algorithmic problems, such as tower of Hanoi, balanced parentheses checking, and finding connected components in graphs. They provide efficient solutions to a wide range of computational problems.

## Conclusion:

## Stacks are fundamental in many computational tasks, including function call management, undo mechanisms in applications, and algorithmic problem solving. This lab shows how to implement and utilize stacks in C++, highlighting their importance and practical applications

## Lab Exercise:

1. Enhance the stack class to handle different data types using C++ templates.
2. Implement a peek function that returns the top element without removing it.
3. Create a function in the Stack class that returns the current size of the stack.
4. Discuss and implement a method to reverse a string using a stack.
5. Implement a program to find the Value of the Top after performing 5 PUSH operations and 2 POP operations continuously in stack.
6. Write a program to check whether the given string is palindrome or not
7. Implement reverse of a String using Stack

**VIVA Questions:**

1. What is an abstract data type?

2. What are linear and nonlinear data Structures?

3. Define stack?

4. What are the various operations that can be performed on stack?

5. What are the applications of stacks?