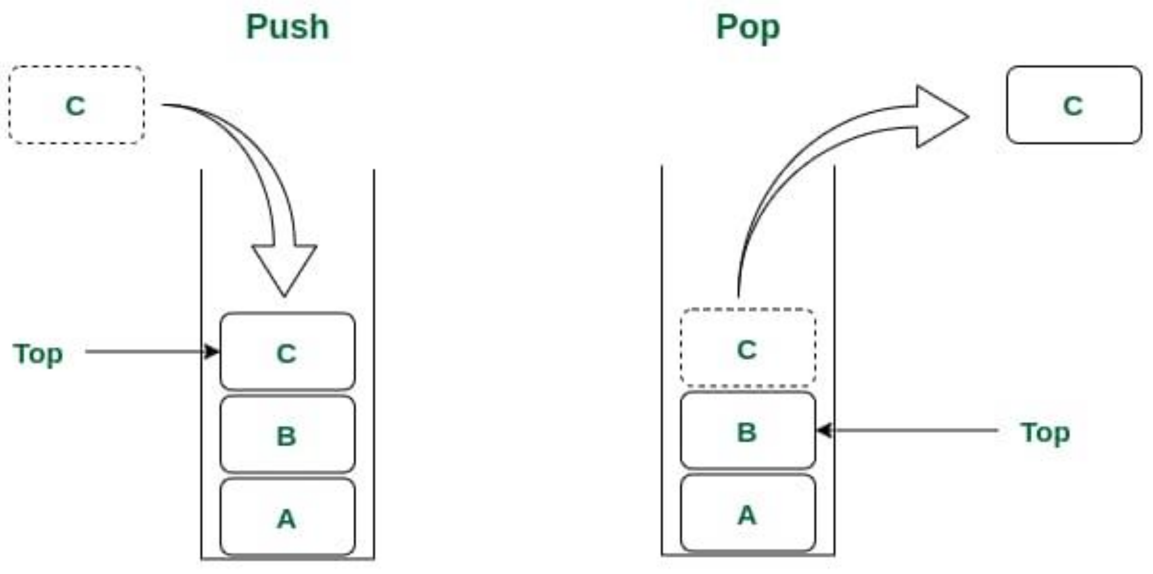
CASE STUDIES-STACK

Faculty Name: Manasa Mam



Stack Data Structure

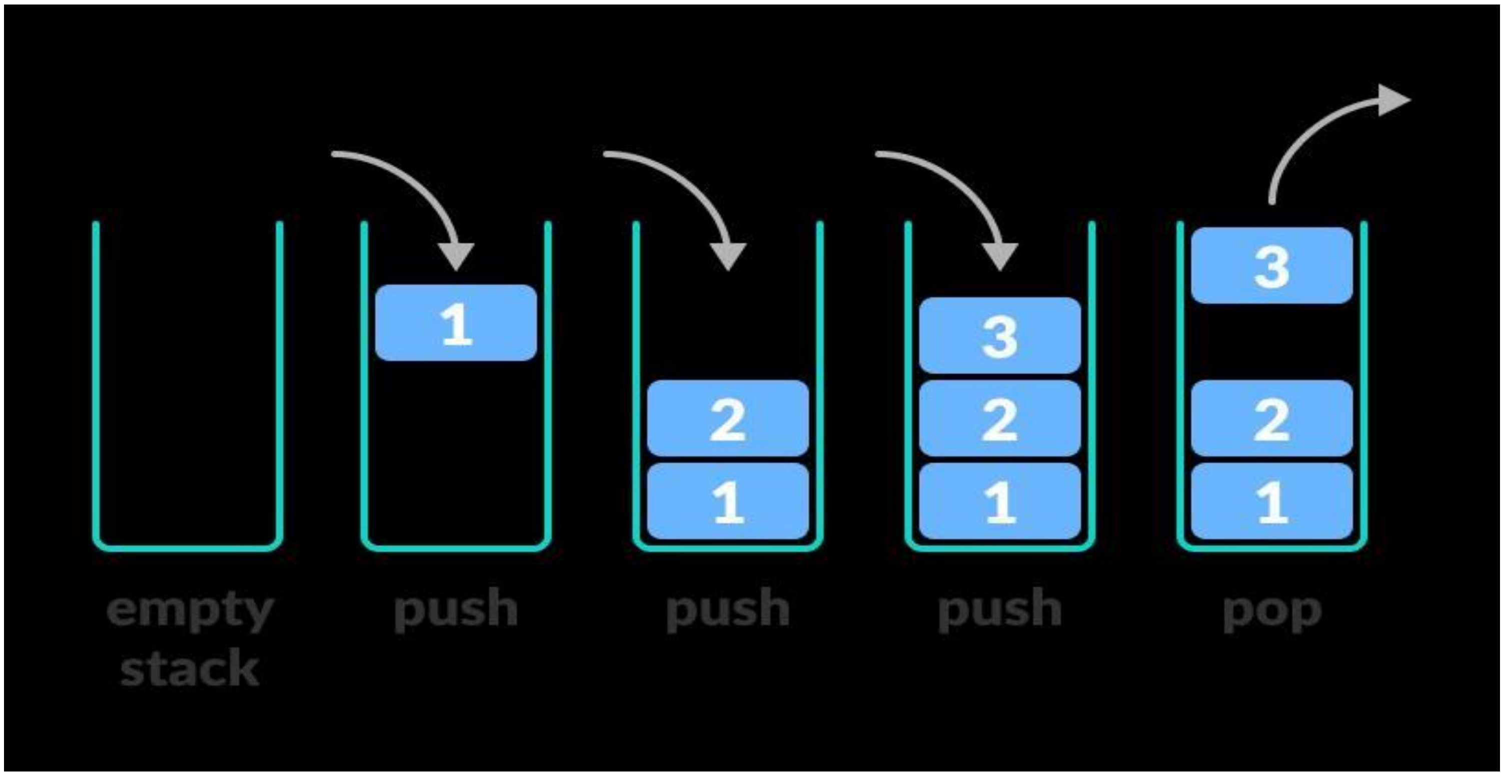
Team names:

Abhishek kasture(22-501)
A.Prabodh Obadiah(22-503)
B. Srikanth(22-512)
B.Naveen(22-515)
Vamsi(22-563)

DATA STRUCTURES - STACK

DEFINITION

A stack is a fundamental data structure characterized by Last In, First Out (LIFO) behaviour. It allows elements to be added or removed only from the top. Key operations include push (adding an element) and pop (removing the top element). Stacks are often used for function call management, expression evaluation, and undo mechanisms in programming. Implementation can be done using arrays or linked lists.



OPERATION

- Push(item): Adds the item to the top of the stack.
- Pop(): Removes and returns the item from the top of the stack.
- Peek() or Top(): Returns the top item without removing it.
- IsEmpty(): Checks if the stack is empty.size(): Returns the number of elements in the stack.

ADVANTAGES

Stacks offer several advantages in data structures, making them a valuable tool in various applications:

1. Simple Implementation: Stacks are easy to implement and understand, making them accessible for programmers and aiding in code readability.

2. Efficient Memory Usage:- Stacks use a fixed amount of memory since they follow the Last In, First Out (LIFO) principle. This simplicity can lead to more efficient memory usage compared to other data structures.

3. Function Call Management:- Stacks are commonly used in programming languages to manage function calls. The call stack keeps track of the order in which functions are called and allows for easy return to the calling function.

4. Undo Mechanism:- Stacks are employed in undo mechanisms in applications. Each action that can be undone is pushed onto the stack, allowing for a straightforward way to revert changes.

5. Expression Evaluation:- Stacks are useful in evaluating expressions, especially in converting between infix, postfix, and prefix notations. They simplify the process of parsing and evaluating mathematical expressions.

6. Backtracking:- In algorithms and search problems, stacks can be used to implement backtracking. Storing states on the stack allows for easy exploration of different possibilities and backtracking when needed.

7. Resource Management:- Stacks are employed in managing resources such as memory allocation and deallocation. The Last In, First Out nature ensures that the most recently allocated resource is the first one to be deallocated.

8. Syntax Parsing:- Stacks are often used in parsing and interpreting the syntax of programming languages. They help ensure the correct order of opening and closing symbols (like parentheses or braces).

9. History Maintenance:- Stacks can be used to maintain a history of actions, facilitating the implementation of features like redo in addition to undo.

10. Task Scheduling:- In operating systems, stacks play a role in managing task scheduling and keeping track of the execution context of different processes.

DISADVANTAGES

While stacks offer several advantages, they also come with certain limitations and disadvantages:

1. Fixed Size:- In some implementations, stacks have a fixed size. This limitation can lead to overflow errors if the stack becomes full, and new elements cannot be accommodated.

2. Access Limitation:- Stacks provide access to only the topmost element. Retrieving or modifying elements in the middle of the stack requires popping elements off the top until reaching the desired position, which can be inefficient.

3. No Random Access:- Unlike arrays, stacks do not support direct access to arbitrary elements. Access is limited to the top of the stack, making it less suitable for scenarios where random access to elements is essential.

4. Limited Use Cases:- Stacks are well-suited for specific scenarios like function call management and expression evaluation, but they may not be the most efficient choice for other data manipulation tasks.

5. Dynamic Memory Management Overhead:- If implemented using dynamic memory allocation, stacks may incur overhead due to constant resizing operations when the number of elements fluctuates.

6. Not Suitable for All Problems:- The Last In, First Out (LIFO) nature of stacks may not be appropriate for certain problem scenarios where a different order of access is required.

7. Complexity in Undo Mechanism:- While stacks are commonly used in undo mechanisms, managing large amounts of undo data can become complex and resource-intensive.

8. Limited Search Functionality:- Stacks do not support searching for elements efficiently. To find a specific element, you may need to pop elements off the stack until you locate the desired one.

9. Recursive Function Limitations:- Stacks are used for managing function calls in recursion, but deep recursion can lead to a stack overflow, especially in languages with limited stack space.

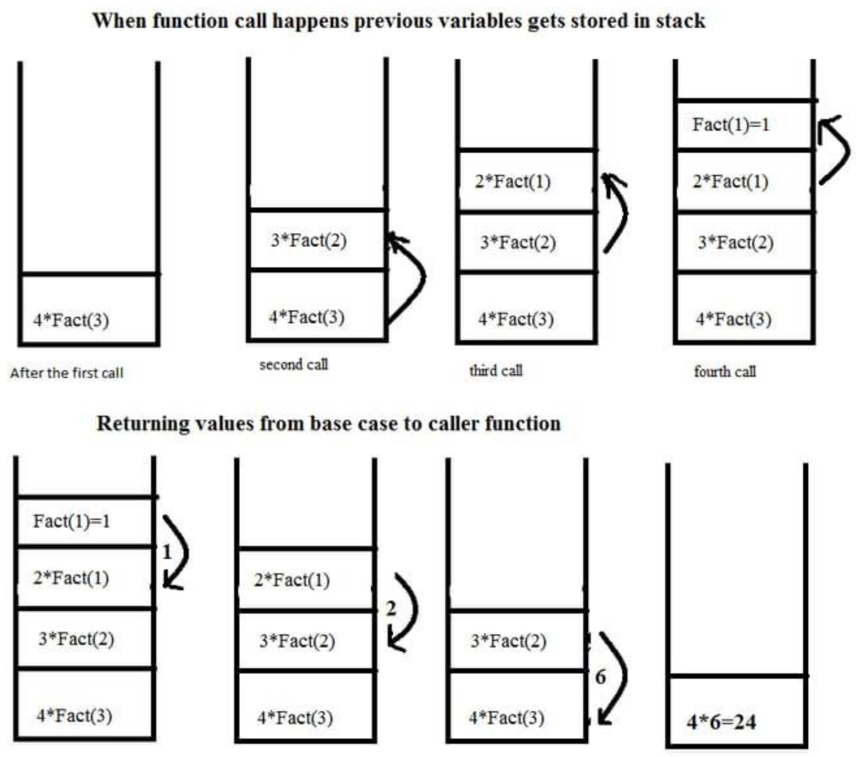
10. Potential for Resource Wastage:- If not managed properly, stacks can lead to resource wastage, especially in situations where a fixed-size stack is allocated but not fully utilized.

REALTIME EXAMPLES/APPLICATIONS

These are the real-time examples of how stacks are used:

1. -Function Call Management in Programming:-

- When a function is called, its execution context is pushed onto the call stack. When the function completes, its context is popped off, allowing the program to return to the previous state.



2. -Undo/Redo Functionality in Software:-

- Software applications use stacks to implement undo and redo functionalities. Each user action is pushed onto the stack, allowing the user to revert or redo actions.

3. -Text Editors and Word Processors:-

- Stacks are used to track the formatting and cursor positions as users apply styles, indentations, or perform other editing actions.

4. -Backtracking in Algorithms:-

- Algorithms that involve backtracking, like depth-first search, often use stacks to keep track of the current state and easily backtrack when needed.

5. -Managing Recursive Calls:-

- Stacks are crucial in managing recursive function calls, ensuring that each recursive call has its own space in memory.

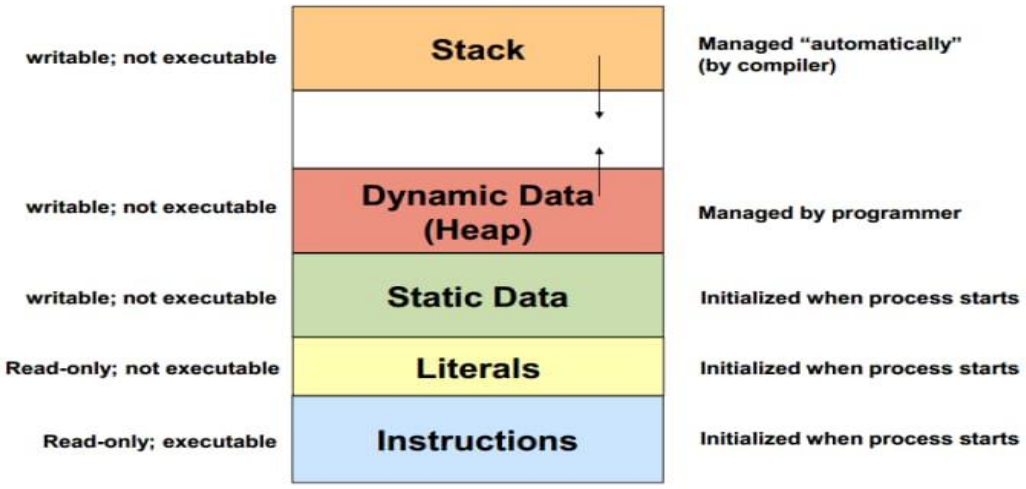
6. -Expression Evaluation in Compilers:-

- Stacks are employed in compilers to evaluate arithmetic expressions by maintaining operands and operators in the correct order.

7. -Browser Forward and Backward Navigation:-

- Web browsers use stacks to keep track of visited pages, enabling users to navigate backward and forward through their browsing history.

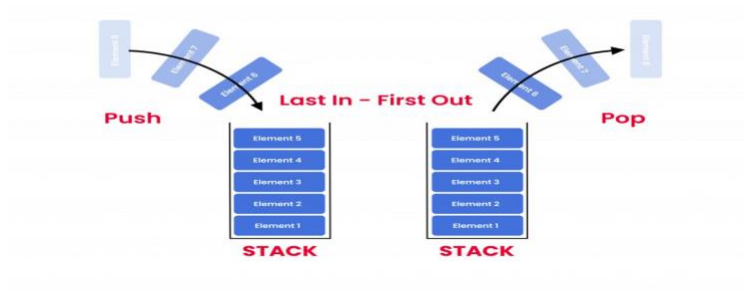
8. -Memory Management in Operating Systems:-



- The operating system uses a stack to manage the memory allocation and deallocation of function calls and local variables.

9. -Bracket Matching in Text Editors:-

- Text editors use stacks to check and highlight matching pairs of brackets,



parentheses, or braces in code.

10. -Game State Management:-

- In game development, stacks can be used to manage different states of the game, such as menus, levels, and pause screens.