ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Data Structures and Algorithms

Lecture 3

© Fall 2025- **Dr. Hesham Sakr**
Building, office No. 12
Hesham.sakr@sut.edu.eg

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Data Structures

Data Structures:

- Data may be organized in many different ways logical or mathematical model of a program particularly organization of data. This organized data is called “Data Structure”.

Or

- The organized collection of data is called a ‘Data Structure’.

$$\boxed{\textbf{Data Structure = Organized data + Allowed operations}}$$

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Data Structure Goals

Data Structure Goals:

- Data Structure involves two complementary goals:

The first goal:

is to identify and develop useful, mathematical entities and operations and to determine what class of problems can be solved by using these entities and operations.

Data Structure Goals:

- Data Structure involves two complementary goals:

The second goal:

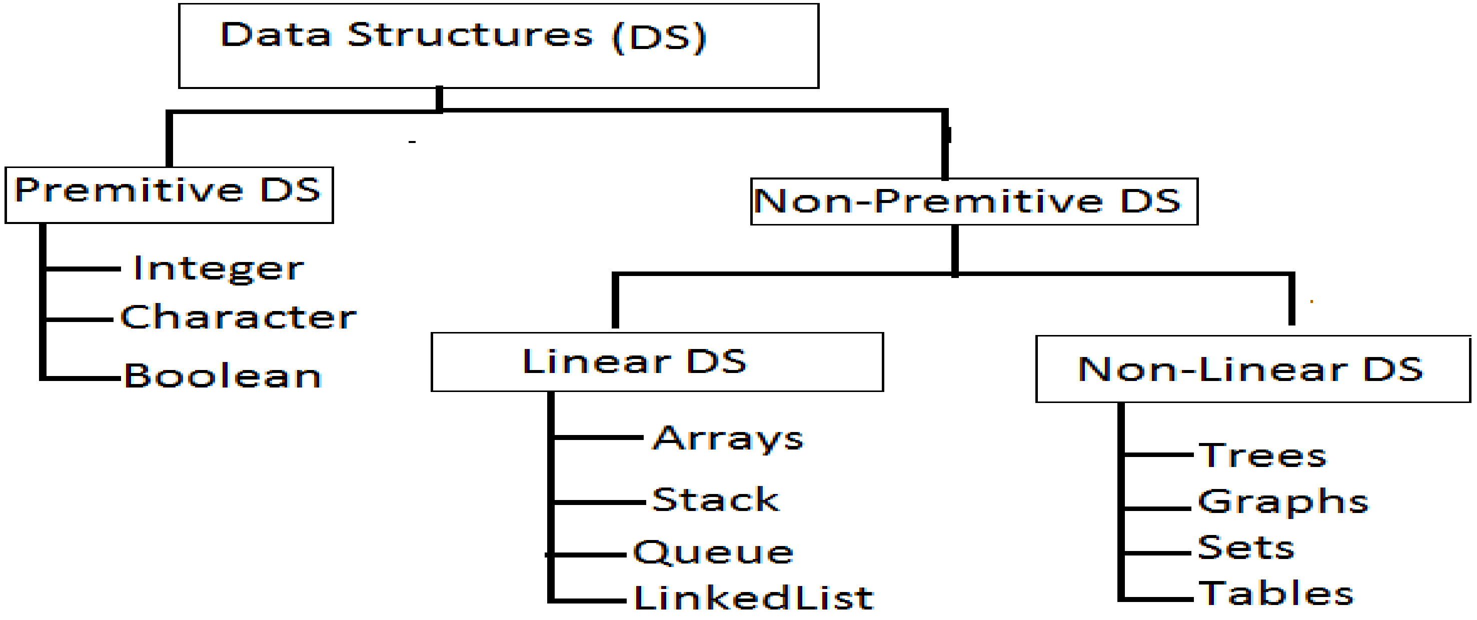
is to determine representation for those abstract entities to

<u>implement</u>

<u>abstract operations</u> on this concrete representation.

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Data Structure Categories



Important for all other branches of computer science

Primitive Data structures:

- Primitive Data structures are directly supported by the programming language.
- i.e; any operation is directly performed in these data items.
- Ex: integer, Character, Real numbers etc.

Non-primitive data structures:

- Non-primitive data types are not defined by the programming language, but are instead created by the programmer.

Non-primitive data structures subcategories:

1. Linear data structures.

2. Non-Linear data structures.

Linear Data structures:

- Linear data structures organize their data elements in a linear fashion, where data elements are attached one after the other.
- Linear data structures are very easy to implement, since the memory of the computer is also organized in a linear fashion.
- Some commonly used linear data structures are arrays, linked lists, stacks and queues.

Nonlinear Data structures:

- In nonlinear data structures, data elements are not organized in a sequential fashion.
- Data structures like multidimensional arrays, trees, graphs, tables and sets are some examples of widely used nonlinear data structures.

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Operations on the Data Structures

Operations on the Data Structures :

Following operations can be performed on the data structures:

1. Traversing
2. Searching
3. Inserting
4. Deleting
5. Sorting
6. Merging

Operations on the Data Structures :

1. Traversing عبور: It is used to access each data item exactly once so that it can be processed.

2. Searching: It is used to find out the location of the data item if it exists in the given collection of data items.

Operations on the Data Structures :

3. Inserting: It is used to add a new data item in the given collection of data items. queues.

4. Deleting: It is used to delete an existing data item from the given collection of data items.

Operations on the Data Structures :

5. **Sorting:** It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

6. **Merging:** It is used to combine the data items of two sorted files into single file in the sorted form.

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

STACKS AND QUEUES

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

STACKS AND QUEUES

STACKS:

- A Stack is linear data structure.
- A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack.

STACKS:

- Stack principle is LIFO (last in, first out). Which element inserted last on to the stack that element deleted first from the stack.
- As the items can be added or removed only from the top i.e. the last item to be added to a stack is the first item to be removed.

Real life examples of stacks are:

Stack of books

Stack of Plates

Stack of Toys

Operations on stack:

- The two basic operations associated with stacks are:
  1. Push
  2. Pop

Operations on stack:

- While performing push and pop operations the following test must be conducted on the stack.

  a. Stack is empty or not
  b. Stack is full or not

1. Push:

Push:

Push operation is used to add new elements in to the stack. At the time of addition first check the stack is full or not. If the stack is full it generates an error message "stack overflow".

2. Pop:

Pop:

Pop operation is used to delete elements from the stack. At the time of deletion first check the stack is empty or not. If the stack is empty it generates an error message "stack underflow“.

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Representation of Stack (or) Implementation of stack

The stack should be represented in two ways:

1. Stack using array

2. Stack using linked list

1. Stack using array:

- Let us consider a stack with 6 elements capacity.
- This is called as the size of the stack.
- The number of elements to be added should not exceed the maximum size of the stack.

1. Stack using array:

- Let us consider a stack with 6 elements capacity.
- This is called as the size of the stack.
- The number of elements to be added should not exceed the maximum size of the stack.

Stack Overflow :

- Similarly, you cannot remove elements beyond the base of the stack.
- If such is the case, we will reach a *stack underflow condition.*

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Operation Performed in Stack

push & pop & display

1.push():

push():

When an element is added to a stack, the operation is performed by

push().

Below Figure shows the creation of a stack and addition of elements using push()

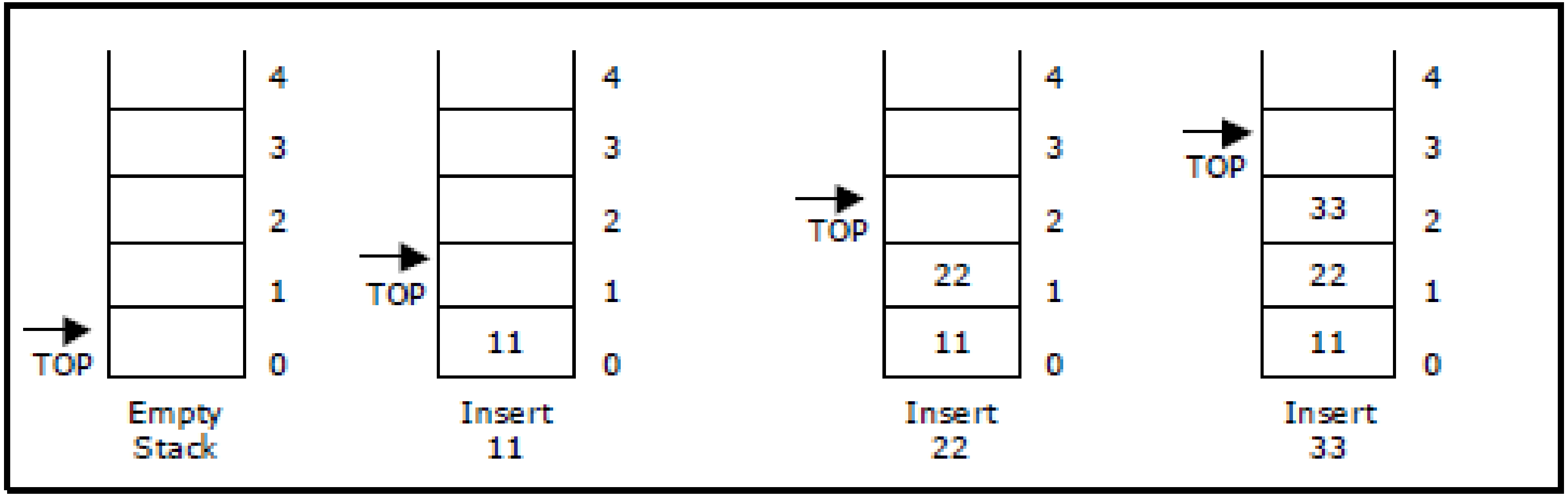


Figure . Push operations on stack

2.pop():

pop():

When an element is taken off from the stack, the operation is performed by

pop().

Below figure shows a stack initially with three elements and shows the deletion of elements using pop()

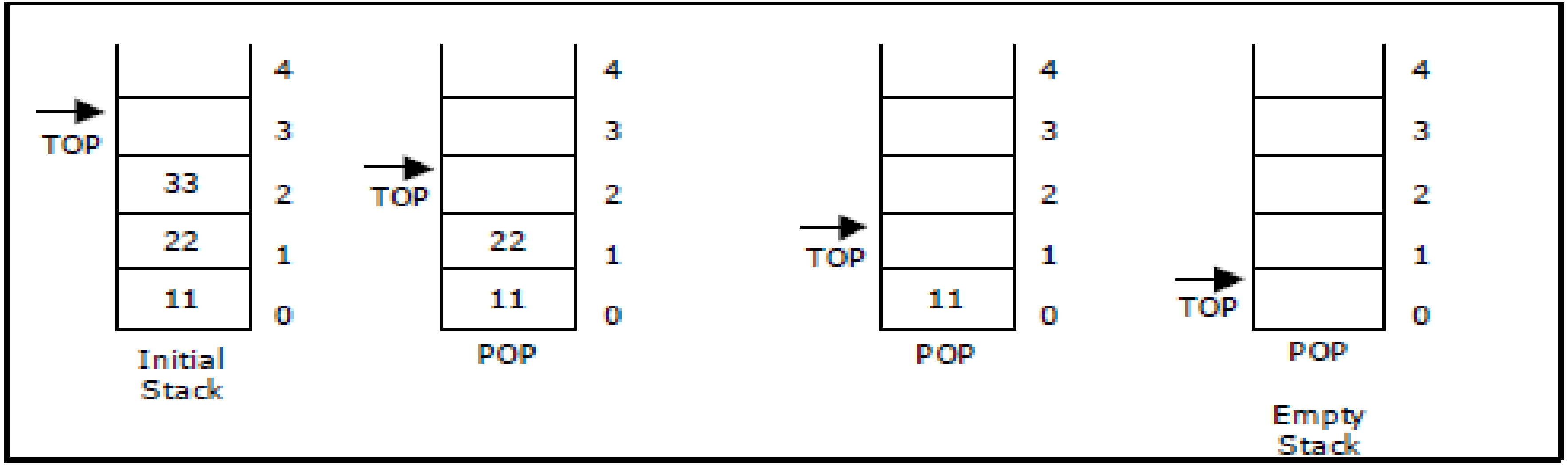


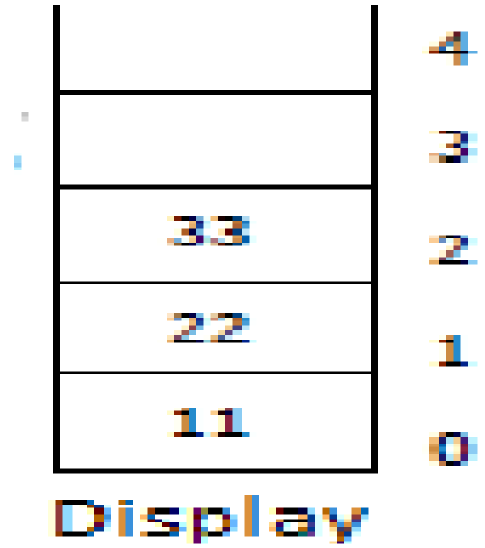
Figure Pop operations on stack

3. display():

display():

- This operation performed display the elements in the stack.
- We display the element in the stack check the condition is stack is empty or not, otherwise display the list of elements in the stack.

Display the elements in the stack



Example1-Push operation

```
#include <iostream>
#include <stack>
int main() {
   std::stack<int> myStack;

   // Push elements onto the stack
   myStack.push(10);
   myStack.push(20);
   myStack.push(30);

   // Display and pop elements from the stack
   while (!myStack.empty()) {
      std::cout << "Top element: " << myStack.top() << std::endl;
      myStack.pop();
   }
  // Check if the stack is empty
   if (myStack.empty()) {
      std::cout << "The stack is now empty!" << std::endl;
   }
   return 0;
}
```

Example1-Push operation -Explanation

? **#include <iostream>**

- This includes the iostream library, which provides basic input/output operations such as std::cout and std::endl.
- **Explanation**: It enables us to print information to the console using std::cout.

? **#include <stack>**

- This includes the stack library from the C++ Standard Template Library (STL).
- **Explanation**: We need this library to work with the stack data structure, which provides operations like push(), pop(), top(), and empty().

? **std::stack<int> myStack;**

- Declares a stack named myStack that will store integers (int type).
- **Explanation**: This initializes an empty stack that will hold integers, allowing us to perform stack operations like push() and pop().

? **myStack.push(10); myStack.push(20); myStack.push(30);**

- **push()** adds elements to the stack. We push the numbers 10, 20, and 30 onto the stack.
- **Explanation**:
    - The stack now contains {10, 20, 30}.
    - 10 was pushed first, so it is at the bottom. 30 was pushed last, so it is at the top (last-in, first-out behavior).

Stack using Arrays Push operation-Example 1

```
#include <iostream>
#define MAX 5  // Define the maximum size of the stack
class Stack {
private:
   int arr[MAX];  // Array to store stack elements
   int top;       // Variable to track the top of the stack
public:
   // Constructor to initialize the stack
   Stack() {
      top = -1;  // Top is set to -1, indicating an empty stack
   }
   // Push operation to add an element onto the stack
   void push(int value) {
      if (top >= MAX - 1) {
         std::cout << "Stack Overflow! Cannot push " << value << ".\n";
      } else {
         top++;          // Move the top pointer up
         arr[top] = value; // Add the new value at the top position
         std::cout << value << " pushed to stack.\n";
      }
   }
   // Display the stack content
   void displayStack() {
      if (top < 0) {
         std::cout << "Stack is empty.\n";
      } else {
         std::cout << "Stack elements are: ";
         for (int i = 0; i <= top; i++) {
            std::cout << arr[i] << " ";
         }
         std::cout << std::endl;
      }
   }
};
int main() {
   Stack stack;
   stack.push(10);  // Push 10
   stack.push(20);  // Push 20
   stack.push(30);  // Push 30
   stack.displayStack();  // Display the stack

   return 0;
}
```

Stack using Arrays-Example 1-Explaination

Explanation of Push Command:

- **Functionality**: The push operation adds a new value at the top of the stack.
- **Commands**:
    - top++: Increment the top index to point to the new top.
    - arr[top] = value: Assigns the new value to the top of the stack.
- **Illustration**:
    - Initial state: top = -1, arr = []
    - After push(10): top = 0, arr = [10]
    - After push(20): top = 1, arr = [10, 20]

Example 2-Pop operation

```
#include <iostream>
#include <stack>    // Required to use stack data structure
int main() {
    std::stack<int> myStack;  // Creating a stack of integers

    // Push elements onto the stack
    myStack.push(10);
    myStack.push(20);
    myStack.push(30);
    std::cout << "Stack before popping elements:" << std::endl;

    // Display the stack's top element without removing it
    std::cout << "Top element: " << myStack.top() << std::endl;

    // Pop the top element from the stack
    myStack.pop();  // Removes 30 (the top-most element)

    std::cout << "Stack after one pop:" << std::endl;
    std::cout << "Top element: " << myStack.top() << std::endl;

    // Pop another element
    myStack.pop();  // Removes 20 (the next top element)

    std::cout << "Stack after two pops:" << std::endl;
    std::cout << "Top element: " << myStack.top() << std::endl;
    return 0;
}
```

Example 2-Pop operation -Explanation

- **#include <iostream>**
  - This includes the iostream library, which provides basic input/output operations such as std::cout and std::endl.
  - **Explanation**: It enables us to print information to the console using std::cout.
- **#include <stack>**
  - This includes the stack library from the C++ Standard Template Library (STL).
  - **Explanation**: We need this library to work with the stack data structure, which provides operations like push(), pop(), top(), and empty().
- **std::stack<int> myStack;**
  - Declares a stack named myStack that will store integers (int type).
  - **Explanation**: This initializes an empty stack that will hold integers, allowing us to perform stack operations like push() and pop().
- **myStack.push(10); myStack.push(20); myStack.push(30);**
  - **push()** adds elements to the stack. We push the numbers 10, 20, and 30 onto the stack.
  - **Explanation**:
    - The stack now contains {10, 20, 30}.
    - 10 was pushed first, so it is at the bottom. 30 was pushed last, so it is at the top (last-in, first-out behavior).
- **std::cout << "Top element: " << myStack.top() << std::endl;**
  - **top()** returns the element at the top of the stack without removing it.
  - **Explanation**:
    - The stack's current top element is 30, which is printed without modifying the stack.
    - **Output**: Top element: 30.

Example 2-Pop operation -Explanation

myStack.pop();

- **pop()** removes the top element from the stack.
- **Explanation**:
  - The top element, 30, is removed.
  - After this operation, the stack contains {10, 20}, and the new top element becomes 20.

std::cout << "Stack after one pop:" << std::endl;

- This outputs a message indicating that one element has been popped from the stack.
- **Explanation**: It helps to show the state of the stack after the pop() operation.

std::cout << "Top element: " << myStack.top() << std::endl;

- **top()** retrieves the new top element (now 20) after the first pop.
- **Explanation**:
  - Since 30 was removed, 20 is now the top of the stack.
  - **Output**: Top element: 20.

myStack.pop();

- **pop()** removes the current top element (20) from the stack.
- **Explanation**: After this second pop() operation, the stack now contains only {10}.

std::cout << "Stack after two pops:" << std::endl;

Outputs a message showing the state of the stack after the second pop.

Explanation: It makes it clear that two elements have been removed from the stack.

std::cout << "Top element: " << myStack.top() << std::endl;

top() retrieves the new top element (now 10) after the second pop.

Explanation:

- After popping 20, the remaining top element is 10.
- **Output**: Top element: 10.

Stack using Arrays Pop operation-Example 2

```
#include <iostream>
#define MAX 5  // Define the maximum size of the stack
class Stack {
private:
   int arr[MAX];  // Array to store stack elements
   int top;       // Variable to track the top of the stack
public:
   // Constructor to initialize the stack
   Stack() {
      top = -1;  // Top is set to -1, indicating an empty stack
   }
   // Pop operation to remove the top element from the stack
   void pop() {
      if (top < 0) {
         std::cout << "Stack Underflow! Cannot pop.\n";
      } else {
         std::cout << arr[top] << " popped from stack.\n";
         top--;  // Decrease the top pointer to remove the top element
      }
   }
   // Display the stack content
   void displayStack() {
      if (top < 0) {
         std::cout << "Stack is empty.\n";
      } else {
         std::cout << "Stack elements are: ";
         for (int i = 0; i <= top; i++) {
            std::cout << arr[i] << " ";
         }
         std::cout << std::endl;
      }
   }
};
int main() {
   Stack stack;
   stack.push(10);  // Push 10
   stack.push(20);  // Push 20
   stack.push(30);  // Push 30
   stack.pop();     // Pop the top element (30)
   stack.displayStack(); // Display the stack after pop
   return 0;
}
```

Stack using Arrays Pop operation-Example 2 -Exp.

Explanation of Pop Command:

- **Functionality**: The pop operation removes the top element from the stack.
- **Commands**:
  - top--: Decrement the top index to remove the top element.
- **Illustration**:
  - Initial state: top = 2, arr = [10, 20, 30]
  - After pop(): top = 1, arr = [10, 20]

Example 3-top() operation

```
#include <iostream>
#include <stack>

int main() {
   std::stack<std::string> myStack;  // Creating a stack to store strings

   // Push some string elements onto the stack
   myStack.push("apple");
   myStack.push("banana");
   myStack.push("cherry");

   // Use top() to access the current top element
   std::cout << "Current top element: " << myStack.top() << std::endl;

   // Pop the top element and check the new top
   myStack.pop();
   std::cout << "After pop, new top element: " << myStack.top() << std::endl;

   return 0;
}
```

Example 3-Top() operation -Explanation

- **std::stack<std::string> myStack;**
  - This declares a stack named myStack that stores strings (std::string type).
  - **Explanation**: A stack is initialized to hold strings, and operations like push(), pop(), and top() will act on this string stack.
- **myStack.push("apple"); myStack.push("banana"); myStack.push("cherry");**
  - **push()** adds the strings "apple", "banana", and "cherry" to the stack.
  - **Explanation**: The stack now contains {"apple", "banana", "cherry"}, with "cherry" at the top.
- **std::cout << "Current top element: " << myStack.top() << std::endl;**
  - **top()** retrieves the element at the top of the stack without removing it.
  - **Explanation**:
    - The top element is "cherry", which is printed.
    - **Output**: Current top element: cherry.
- **myStack.pop();**
  - **pop()** removes the top element from the stack, which is "cherry".
  - **Explanation**: After this operation, the stack contains {"apple", "banana"}, and the top element is now "banana".
- **std::cout << "After pop, new top element: " << myStack.top() << std::endl;**
  - **top()** is called again to retrieve the new top element ("banana"), which is now at the top after the pop.
  - **Explanation**: This shows how the top of the stack changes after an element is removed.
  - **Output**: After pop, new top element: banana.

Stack using Arrays Top operation-Example 3

```
#include <iostream>
#define MAX 5  // Define the maximum size of the stack
class Stack {
private:
   int arr[MAX];  // Array to store stack elements
   int top;       // Variable to track the top of the stack
public:
   // Constructor to initialize the stack
   Stack() {
      top = -1;  // Top is set to -1, indicating an empty stack
   }
   // Top operation to return the current top element of the stack
   int getTop() {
      if (top < 0) {
         std::cout << "Stack is empty.\n";
         return -1;
      } else {
         return arr[top];  // Return the element at the top position
      }
   }
   // Display the stack content
   void displayStack() {
      if (top < 0) {
         std::cout << "Stack is empty.\n";
      } else {
         std::cout << "Stack elements are: ";
         for (int i = 0; i <= top; i++) {
            std::cout << arr[i] << " ";
         }
         std::cout << std::endl;
      }
   }
};
int main() {
   Stack stack;
   stack.push(10);  // Push 10
   stack.push(20);  // Push 20
   stack.push(30);  // Push 30
   std::cout << "Current top element: " << stack.getTop() << std::endl;  // Check top element
   return 0;
}
```

Stack using Arrays Top operation-Example 3 -Exp.

Explanation of Top Command:

- **Functionality**: The getTop operation retrieves the current top element of the stack without removing it.
- **Commands**:
  - return arr[top]: Returns the value of the top element.
- **Illustration**:
  - Initial state: top = 2, arr = [10, 20, 30]
  - After getTop(): Returns 30, stack remains unchanged.

Example 4-Empty () operation

```
#include <iostream>
#include <stack>
int main() {
    std::stack<int> myStack;  // Creating a stack of integers
    // Check if the stack is empty initially
    if (myStack.empty()) {
        std::cout << "The stack is empty at the start." << std::endl;
    }
    // Push elements onto the stack
    myStack.push(100);
    myStack.push(200);
    // Check again if the stack is empty after pushing elements
    if (!myStack.empty()) {
        std::cout << "The stack is not empty after pushing elements." << std::endl;
    }
  // Pop all elements
    myStack.pop();  // Removes 200
    myStack.pop();  // Removes 100
    // Check if the stack is empty after popping all elements
    if (myStack.empty()) {
        std::cout << "The stack is now empty after popping all elements." <<
std::endl;
    }
    return 0;
}
```

Example 4-Empty () operation -Explanation

std::stack<int> myStack;

- Declares a stack named myStack that will store integers (int type).
- **Explanation**: This initializes an empty stack that can hold integers. The empty() function will be used to check its status.

if (myStack.empty())

- **empty()** checks if the stack is empty. Initially, the stack is empty since no elements have been added yet.
- **Explanation**: The stack is empty at the start, so this condition is true, and the message "The stack is empty at the start" is printed.
- **Output**: The stack is empty at the start.

myStack.push(100); myStack.push(200);

- **push()** adds two elements (100 and 200) to the stack.
- **Explanation**: The stack now contains {100, 200}, and the top element is 200.

if (!myStack.empty())

- **empty()** is used again, but this time it checks if the stack is **not** empty (!myStack.empty()).
- **Explanation**: Since the stack contains two elements (100 and 200), this condition is true, and the message "The stack is not empty after pushing elements" is printed.
- **Output**: The stack is not empty after pushing elements.

myStack.pop();

- **pop()** removes the top element from the stack. Initially, 200 is removed.
- **Explanation**: After this operation, the stack contains {100}, with 100 as the new top element.

Example 4-Empty () operation -Explanation

myStack.pop();

- **pop()** removes the next element from the stack, which is 100.
- **Explanation**: After this operation, the stack is empty again.

if (myStack.empty())

- **empty()** checks if the stack is now empty after popping all elements.
- **Explanation**: The stack is empty at this point, so the message "The stack is now empty after popping all elements" is printed.
- **Output**: The stack is now empty after popping all elements.

Stack using Arrays Empty () operation-Example 4

```
#include <iostream>
#define MAX 5  // Define the maximum size of the stack
class Stack {
private:
   int arr[MAX];  // Array to store stack elements
   int top;       // Variable to track the top of the stack
public:
   // Constructor to initialize the stack
   Stack() {
      top = -1;  // Top is set to -1, indicating an empty stack
   }

   // Empty operation to check if the stack is empty
   bool isEmpty() {
      return (top < 0);  // Returns true if the stack is empty, false otherwise
   }
};
int main() {
   Stack stack;
   stack.push(10);  // Push 10
   stack.push(20);  // Push 20
   stack.pop();     // Pop the top element
   std::cout << "Is the stack empty? " << (stack.isEmpty() ? "Yes" : "No") << std::endl;  // Check if
empty
   return 0;
}
```

Stack using Arrays Empty () operation-Example 4- Exp.

Explanation of Empty Command:

- **Functionality**: The isEmpty operation checks if the stack is empty.
- **Commands**:
  - return (top < 0): Returns true if the stack is empty.
- **Illustration**:
  - Initial state: top = -1, arr = [], isEmpty() returns true.
  - After push(10): top = 0, arr = [10], isEmpty() returns false.

Activity

Write the same programs for push- pop- top- empty operations but under user choice.

ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Thank You