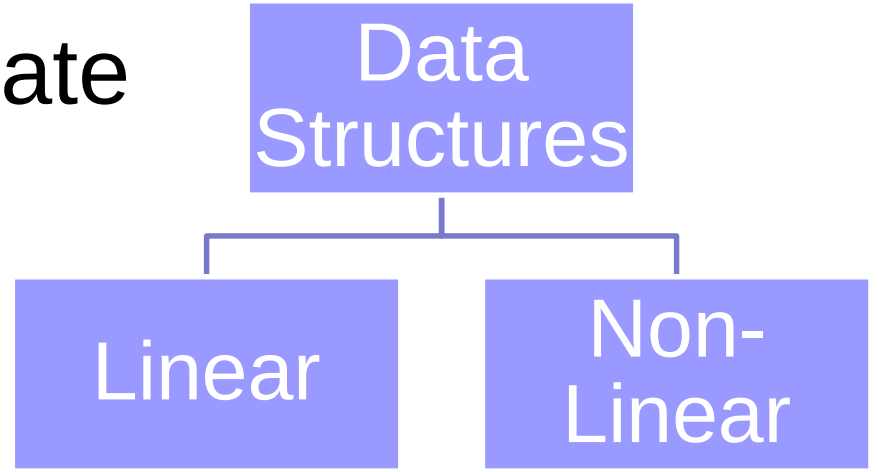
Stack

Unit-2

Created By: Prof. Aparna

Data Structures Types

- Combination of data structures along with algorithm makes the program cost efficient and accurate

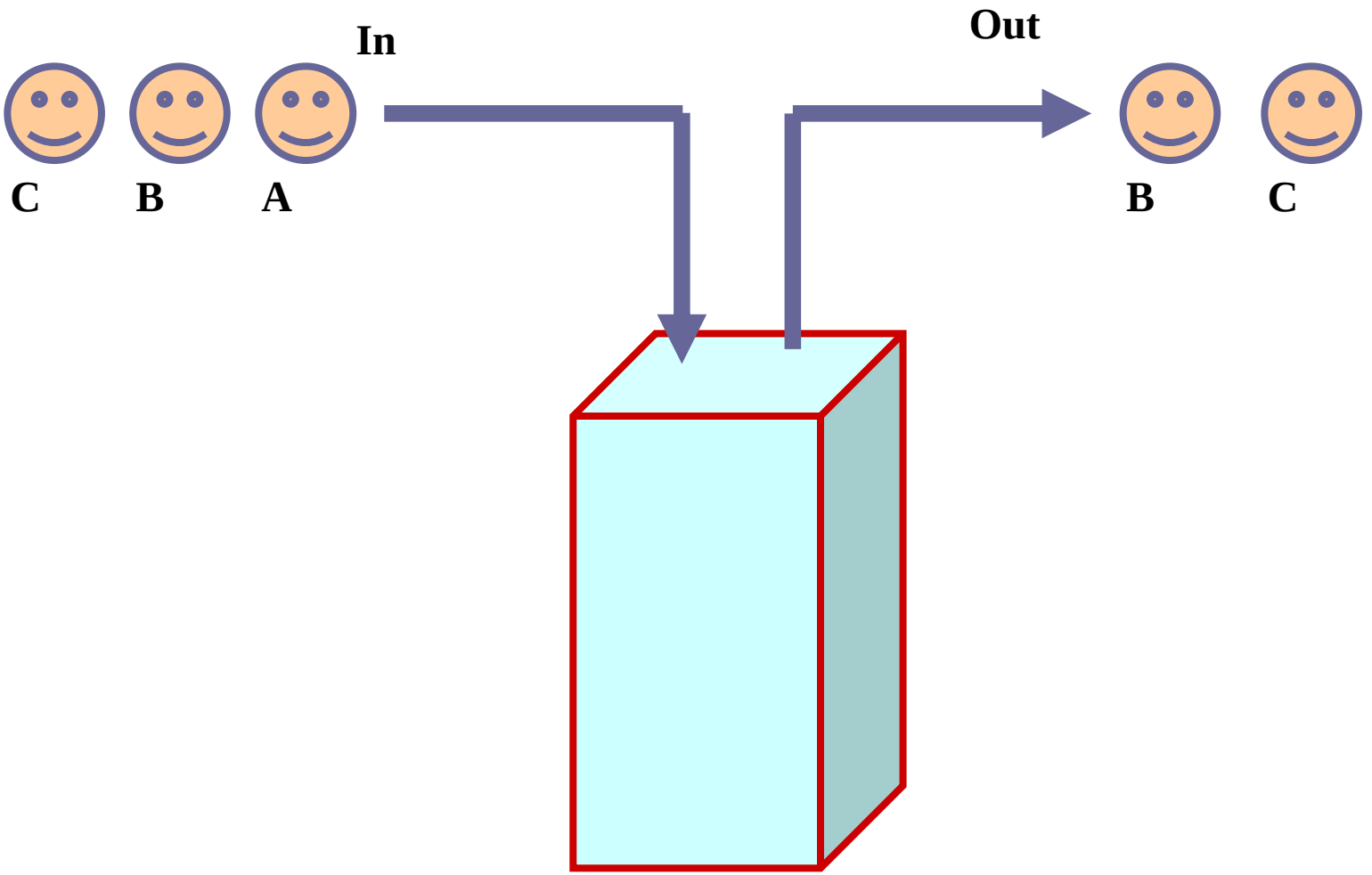


Stack

- A stack is a **linear data structure**, collection of items of the same type (sequentially connected).
- Stack follows the **Last In First Out (LIFO)** fashion wherein the last element entered is the first one to be popped out.

Stack

Data structure with Last-In First-Out (LIFO) behavior



Operations on Stack

isempty: determines if the stack has no elements

isfull: determines if the stack is full in case of a bounded sized stack

top: returns the top element in the stack

push: inserts an element into the stack

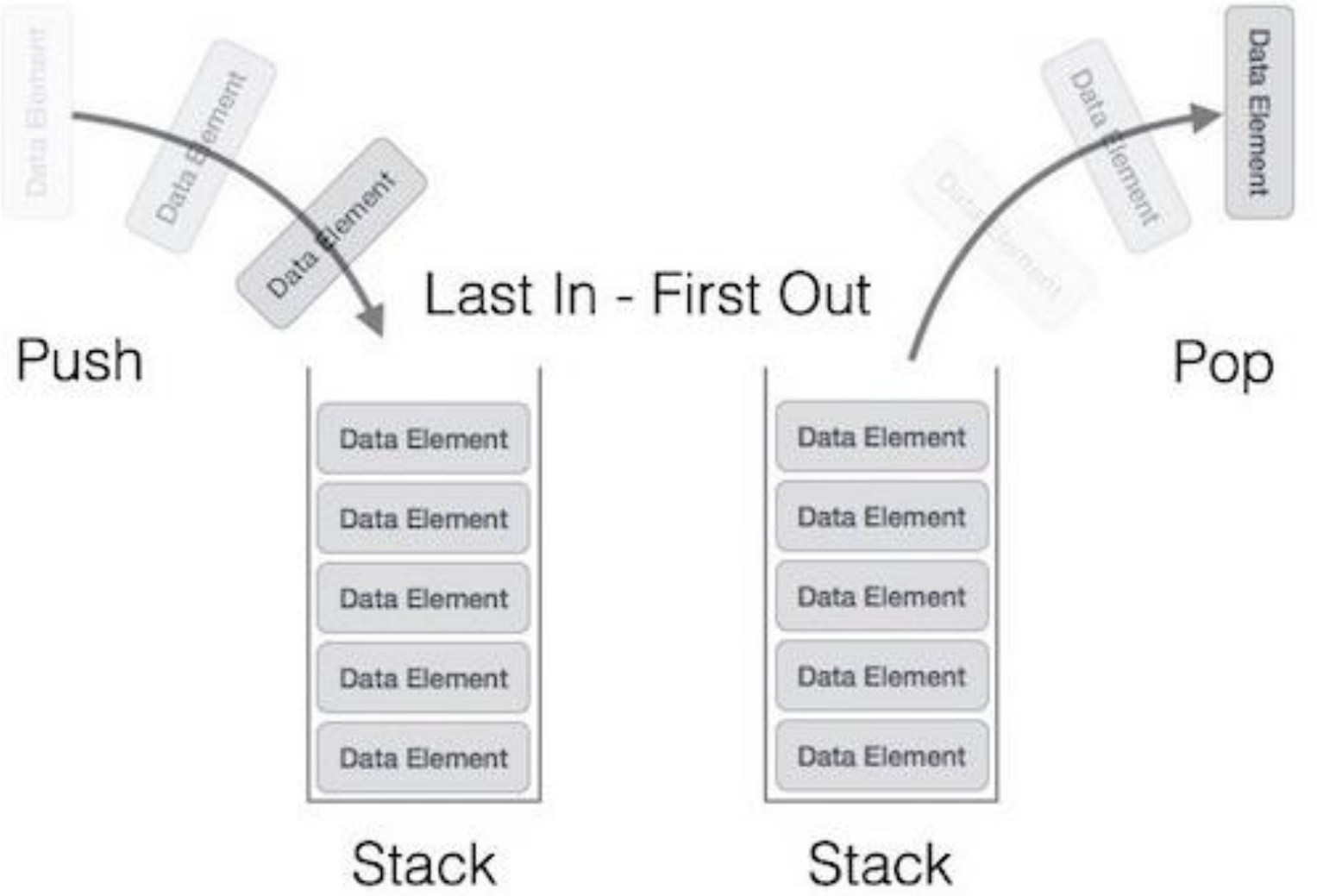
pop: removes the top element from the stack

push is like inserting at the front of the list

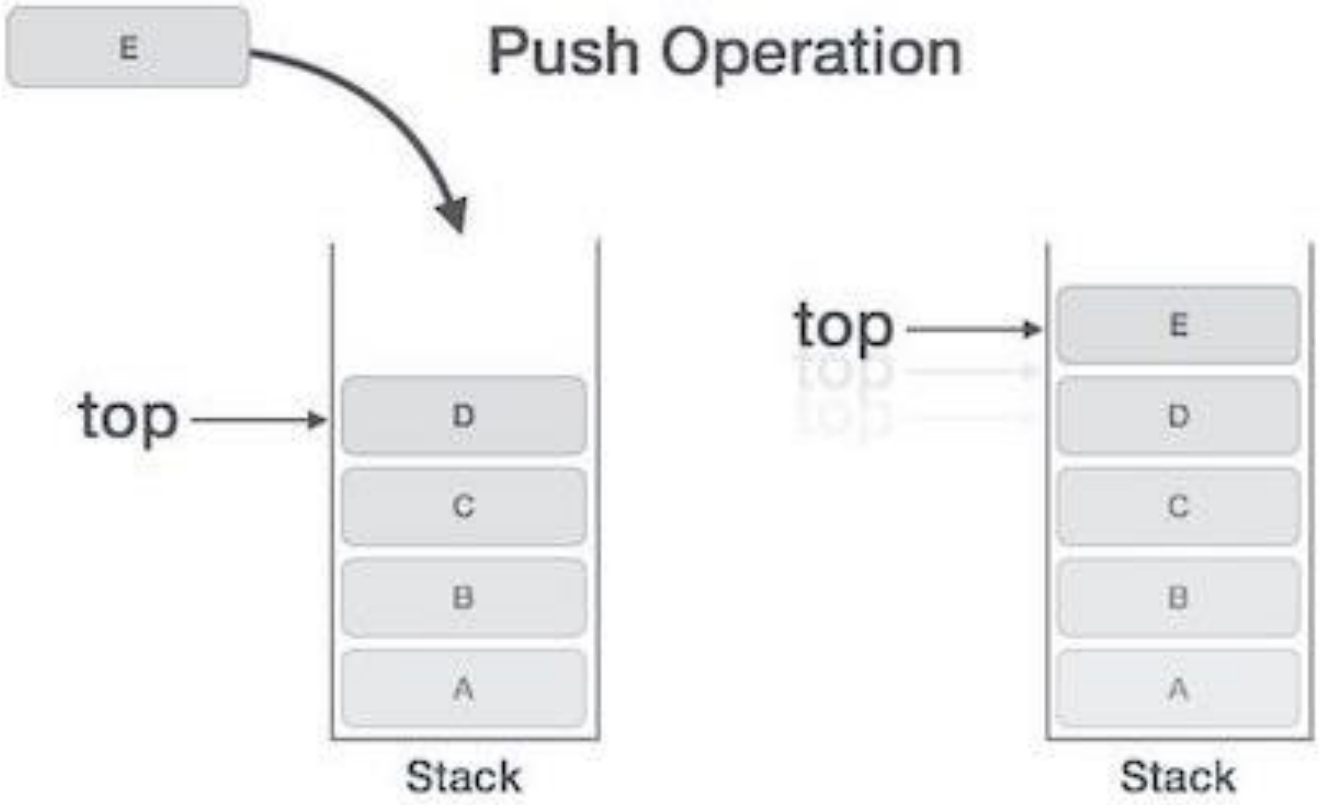
pop is like deleting from the front of the list

push(), pop(), isEmpty() and isfull() all take O(1) time.

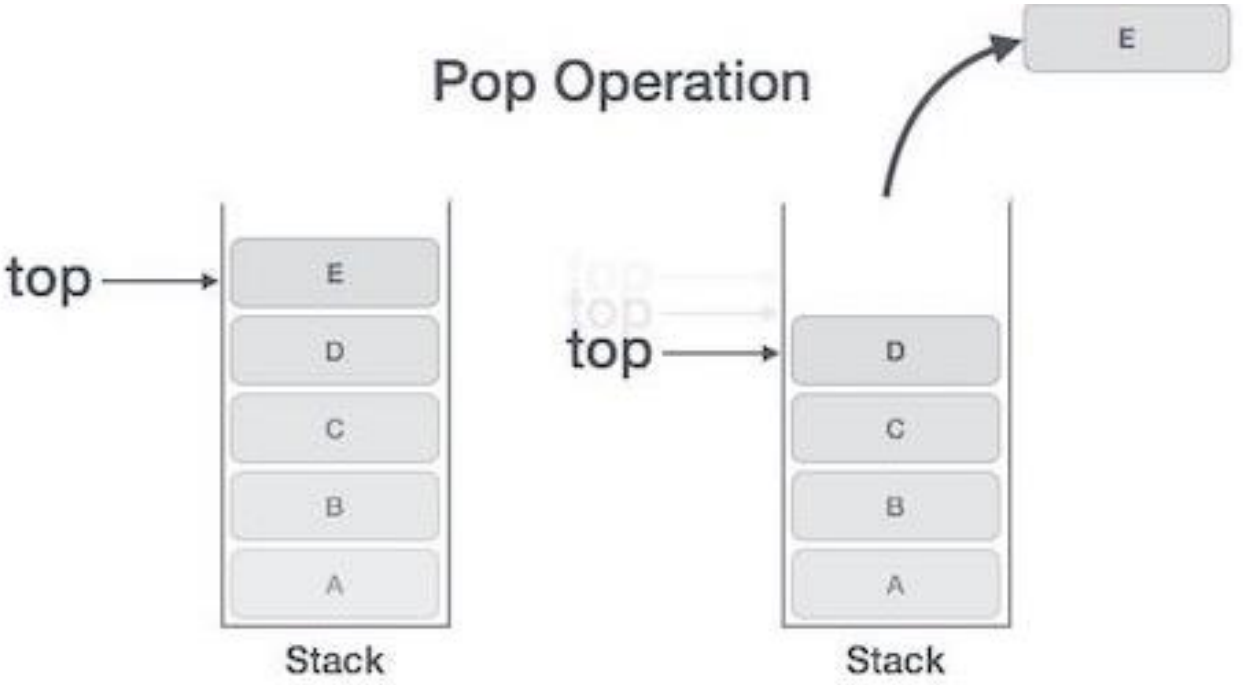
Operations on Stack



Push Operation



Pop Operation



Stack real-life example

For example, we can place or remove a card or plate from the top of the stack only.

Working of Stack

- Initially, set a pointer Top/Peek to keep the track of the topmost item in the stack.
- Initialize the stack to -1. Then, we check whether the stack is empty through the comparison of Peek to -1 i.e. **Top == -1**
- As we add the elements to the stack: Peek element position keeps updating every time.
- Delete item from the set of inputs: the top-most element gets deleted and thus the value of Peek/Top gets reduced.

Working of Stack

- Initially, set a pointer Top/Peek to keep the track of the topmost item in the stack.
- Initialize the stack to -1. Then, we check whether the stack is empty through the comparison of Peek to -1 i.e. **Top == -1**
- As we add the elements to the stack: Peek element position keeps updating every time.
- Delete item from the set of inputs: the top-most element gets deleted and thus the value of Peek/Top gets reduced.

Algorithm for PUSH Operation

```
begin procedure
{

if stack is full
     return null
endif
     top ← top + 1
     stack[top] ← data
}

end procedure
```

Algorithm for POP Operation

```
begin procedure
{
     if stack is empty
          return null
     endif
     data ← stack[top]
     top ← top - 1
     return data
}
end procedure
```

Implementing Stack in C

- Stacks can be represented using
  - Arrays
  - Linked lists.

- Here, We have implemented stacks using arrays in C.

Implementing Stack in C

```
int Top=-1, stack_array[Size];
void Push();
void Pop();
void show();
```

Implementing Stack in C

```
void Push()
{
    int x;

    if(Top==Size-1)
    {
        printf("\nOverflow!!");
    }
    else
    {
        printf("\nEnter element to be
inserted to the stack:");
        scanf("%d",&x);
        Top=Top+1;
        stack_array[Top]=x;
    }
```

Implementing Stack in C

```
void Pop()
{
    if(Top==-1)
    {
        printf("\nUnderflow!!");
    }
    else
    {
        printf("\nPopped
element:  %d",stack_array[Top]);
        Top=Top-1;
    }
}
```

Implementing Stack in C

```
void show()
{

    if(Top==-1)
    {
        printf("\nUnderflow!!");
    }
    else
    {
        printf("\nElements present in the
stack: \n");
        for(int i=Top;i>=0;--i)
            printf("%d\n",stack_array[i]);
    }
}
```

Thankyou!!!