Programming and Data Structures - I

Lecture 15

Kripabandhu Ghosh

CDS, IISER Kolkata

STACK

Stack

Features

- Dynamic data structure (Abstract Data Type)
- Insertion and deletion at **one end**
- Deletion based on **Last In First Out** (LIFO) policy
- Insertion called **Push** operation
- Deletion called **Pop** operation
- Usually **Top** attribute is used to point to tle last inserted element

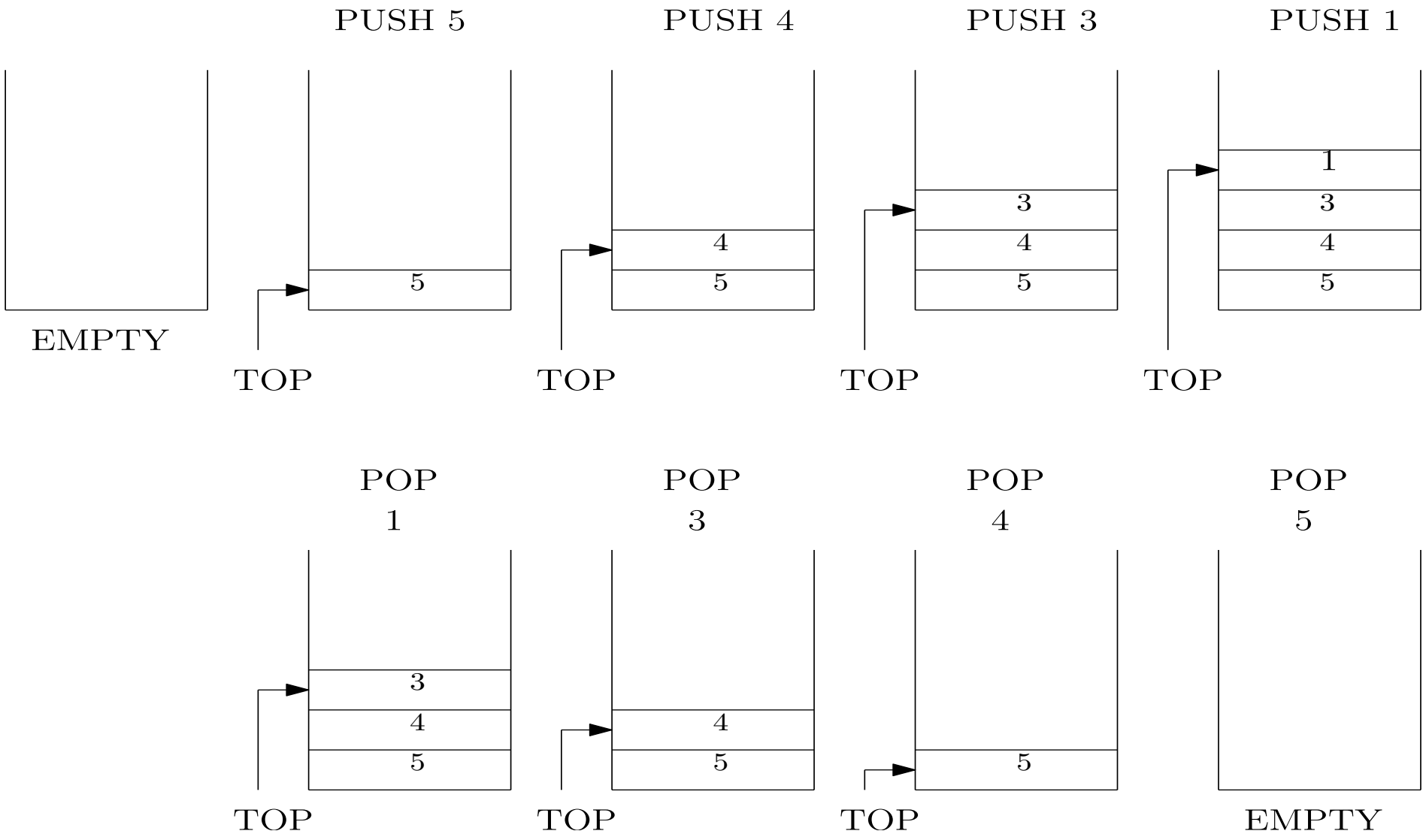
Applications

- Function (subroutine) call and return
- Backtracking
- Syntax analysis
- Depth First Search

Stack example[1]

[1] Source: https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

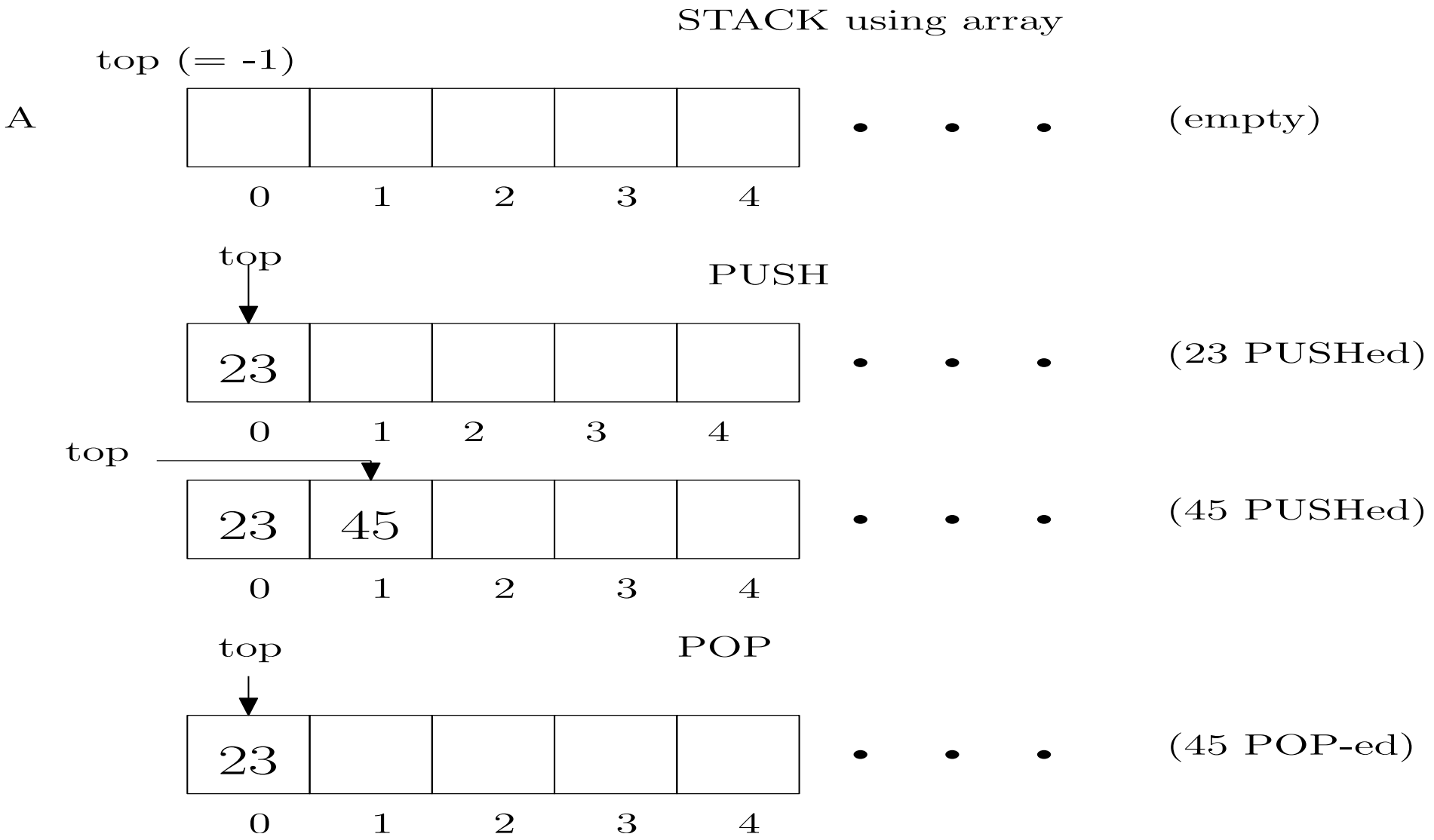
Stack : pictorial representation



Stack : operations

- **Push**: Insertion
- **Pop**: Deletion
- **isEmpty**: Check if the stack is empty
- **isFull**: Check if the stack is full
- **Peek**: Returns the top element (element eligible for deletion)

Stack: PUSH and POP



Stack implementation

- Arrays
- Linked lists

Stack implementation

- **Arrays**
- Linked lists

PUSH: array implementation

Code

```
#include<stdio.h>
#include<stdlib.h>

int stack_arr[MAX];
int top = -1;

void push(int item)
{
   if(isFull())
   {
      printf("Stack Overflow\n");
      return;
   }
   top = top + 1;
   stack_arr[top] = item;
}
```

POP: array implementation

Code

```
int pop()
{
   int item;
   if(isEmpty())
   {
      printf("Stack Underflow\n");
      return -999;
   }
   item = stack_arr[top];
   top = top - 1;
   return item;
}
```

isEmpty(): array implementation

Code

```
int isEmpty() //Returns 1 if the stack in empty, else returns 0
{
   if(top == -1)
      return 1;
   else
      return 0;
}
```

isFull(): array implementation

Code

```
int isFull() //Returns 1 if the stack in full, else returns 0
{
   if(top == MAX - 1)
      return 1;
   else
      return 0;
}
```

peek(): array implementation

Code

```
int peek() //Returns top element
{
   if(isEmpty())
      return -999;
   else
      return stack_array[top];
}
```

Reversing a string using stack

Code

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#define MAX 20

void push(char* stack, int* top, char item)
{
   int i;
   if(*top == (MAX - 1))
   {
      printf("Stack Overflow\n");
      return;
   }
   *top = *top + 1;
   stack[*top] = item;
}
```

Reversing a string using stack (contd.)

Code

```
char pop(char* stack, int* top)
{
   char item;
   if(*top == -1)
   {
      printf("Stack Underflow\n");
      exit(1);
   }
   item = stack[*top];
   *top = *top - 1;
   return item;
}
```

Reversing a string using stack (contd.)

Code

```
void main()
{
   int top = -1;
   char stack[MAX];
   char str[] = "stressed"; //Emordnilap
   int i;
   /*Push characters of the string str on the stack */
   for(i = 0; i < strlen(str); i++)
   {
      push(stack, &top, str[i]);
   }
   /*Pop characters from the stack and store in string str */
   for(i = 0; i < strlen(str); i++)
   {
      str[i] = pop(stack, &top);
   }
   printf("Reversed string is : %s\n", str);
}
```

QUEUE

Queue

Features

- Dynamic data structure (Abstract Data Type)
- Insertion at **rear** (tail) – Enqueue
- Deletion at **front** (head) – Dequeue
- Deletion based on **First In First Out** (FIFO) policy

Applications

- Scheduling
- Breadth First Search

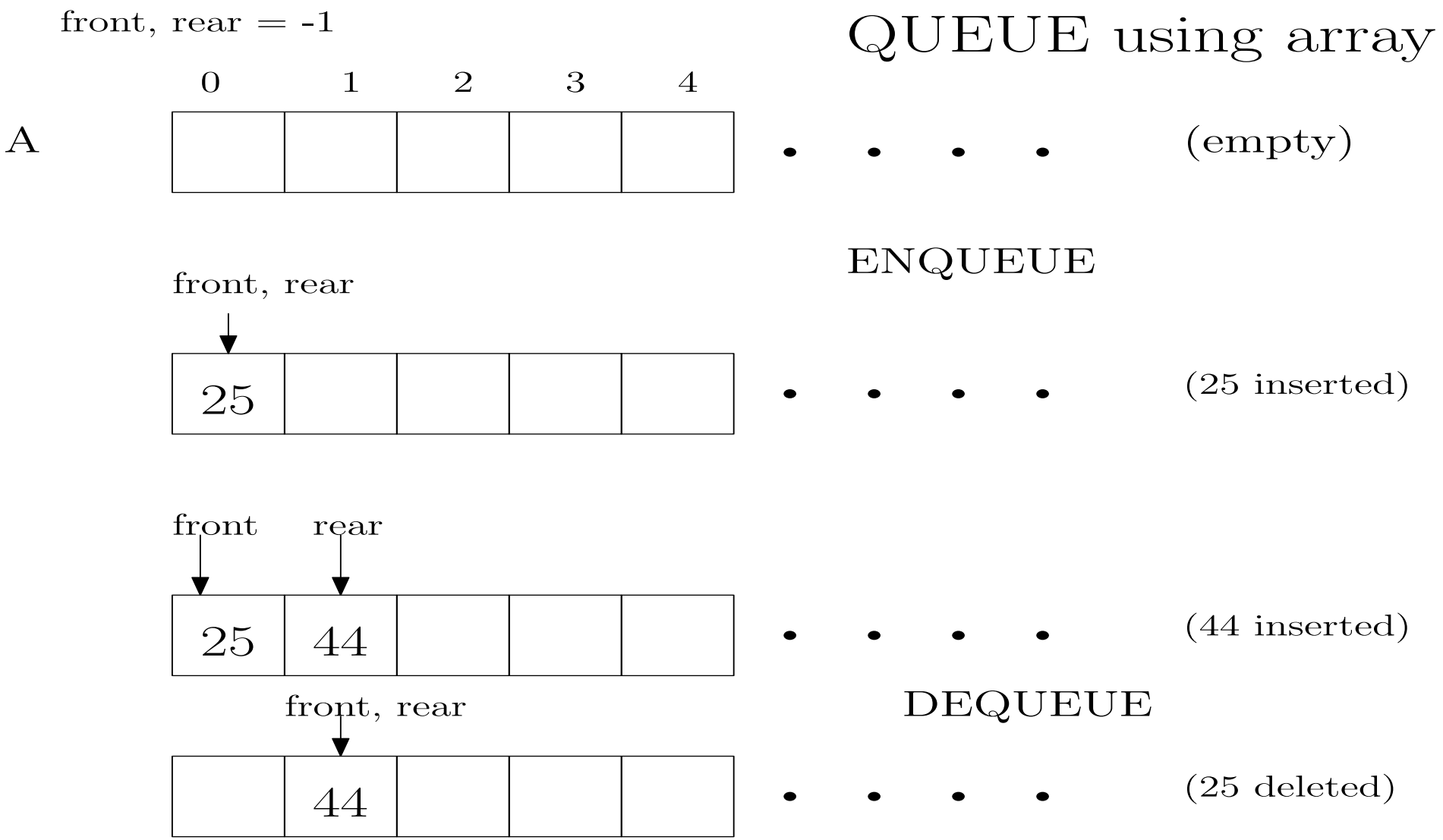
Queue : example[2]

[2] Source:Google

Queue : operations

- **Enqueue**: Insertion
- **Dequeue**: Deletion
- **isEmpty**: Check if the queue is empty
- **isFull**: Check if the queue is full
- **Peek**: Returns the front element (element eligible for deletion)

Queue: ENQUEUE and DEQUEUE



Queue implementation

- Arrays
- Linked lists

Queue implementation

- **Arrays**
- Linked lists

Enqueue : array implementation

Code

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int queue_arr[MAX];
int rear = -1;
int front = -1;

void enqueue(int item)
{
   if(isFull())
   {
      printf("Queue Overflow\n");
      return;
   }
   if(front == -1)
      front = 0;
   rear = rear + 1;
   queue_arr[rear] = item ;
}
```

Dequeue : array implementation

Code

```
int dequeue(int item)
{
   int item;
   if(isEmpty())
   {
      printf("Queue Underflow\n");
      return -999;
   }
   item = queue_arr[front];
   front = front + 1;
   return item;
}
```

isEmpty() : array implementation

Code

```
int isEmpty() //Returns 1 if the queue is empty, else returns 0
{
   if(front == -1 || front == rear + 1)
      return 1;
   else
      return 0;
}
```

isFull() : array implementation

Code

```
int isFull() //Returns 1 if the queue is full, else returns 0
{
   if(rear == MAX - 1)
      return 1;
   else
      return 0;
}
```

peek() : array implementation

Code

```
int peek() //Returns front element
{
   if(isEmpty())
      return -999;
   else
      return queue_arr[front];
}
```


Thank
you