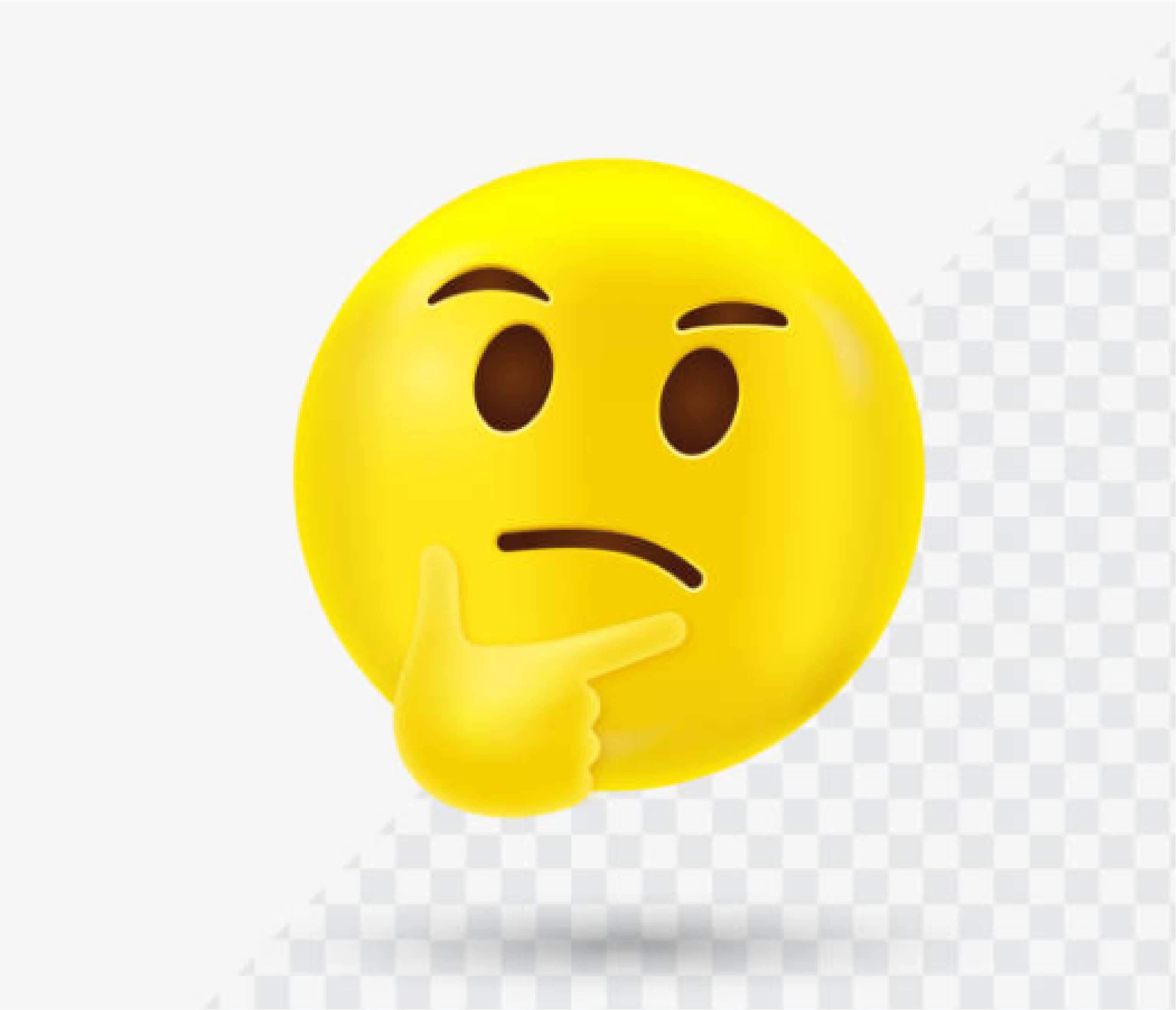
WCE acm Chapter

STACKS & QUEUES

WHAT IS STACK??

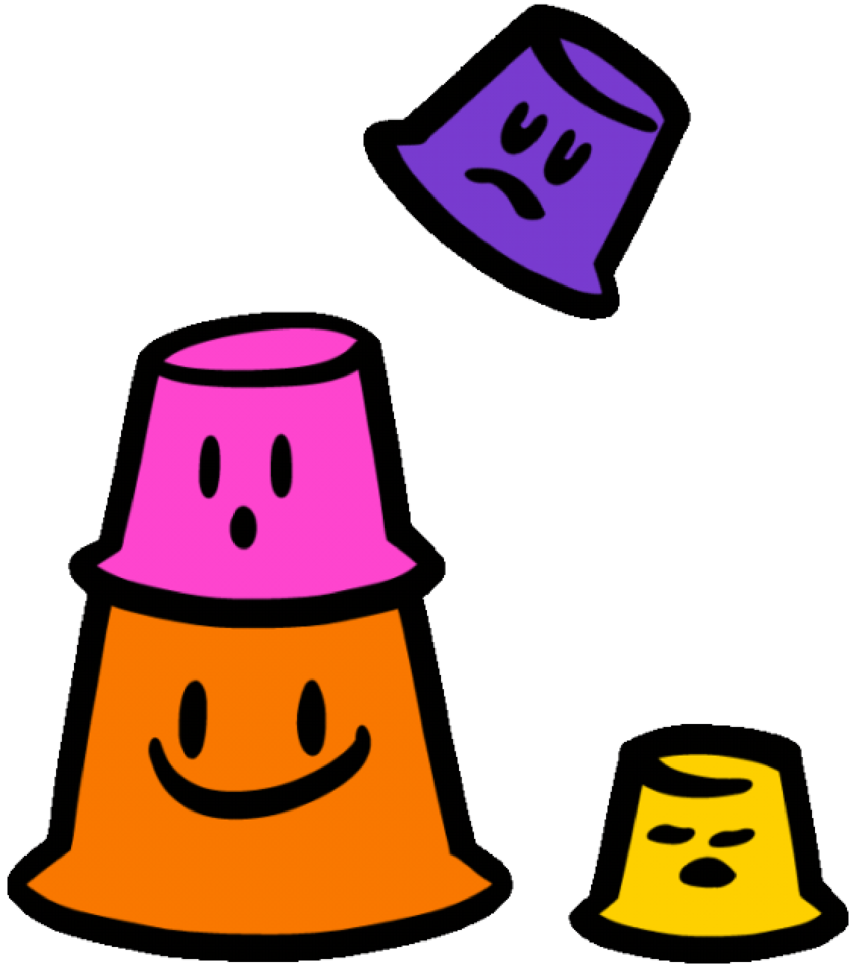


Let's See

- Stack is a linear data structure that follows a particular order in which the operations are performed.

- The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Examples of Stack

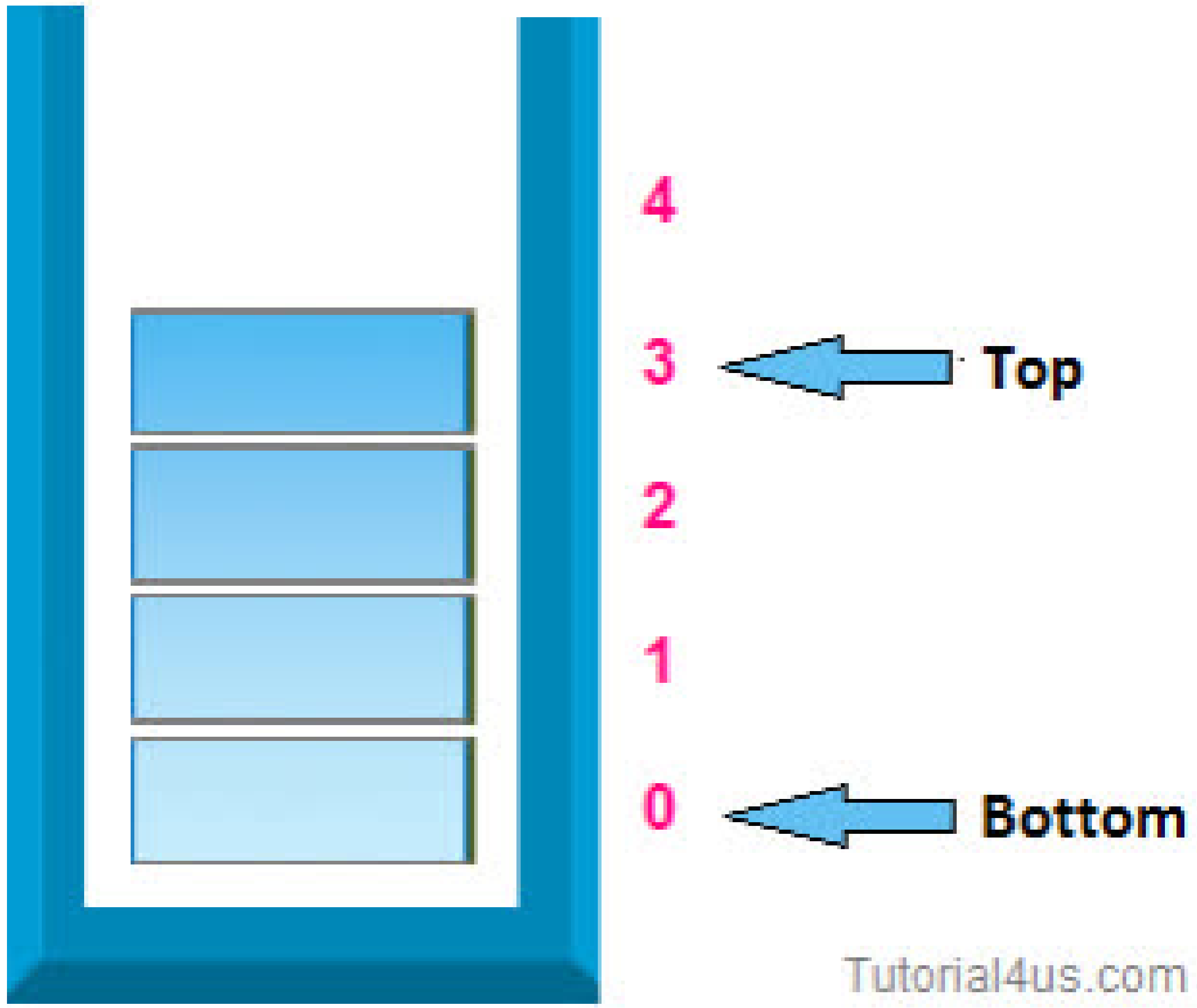


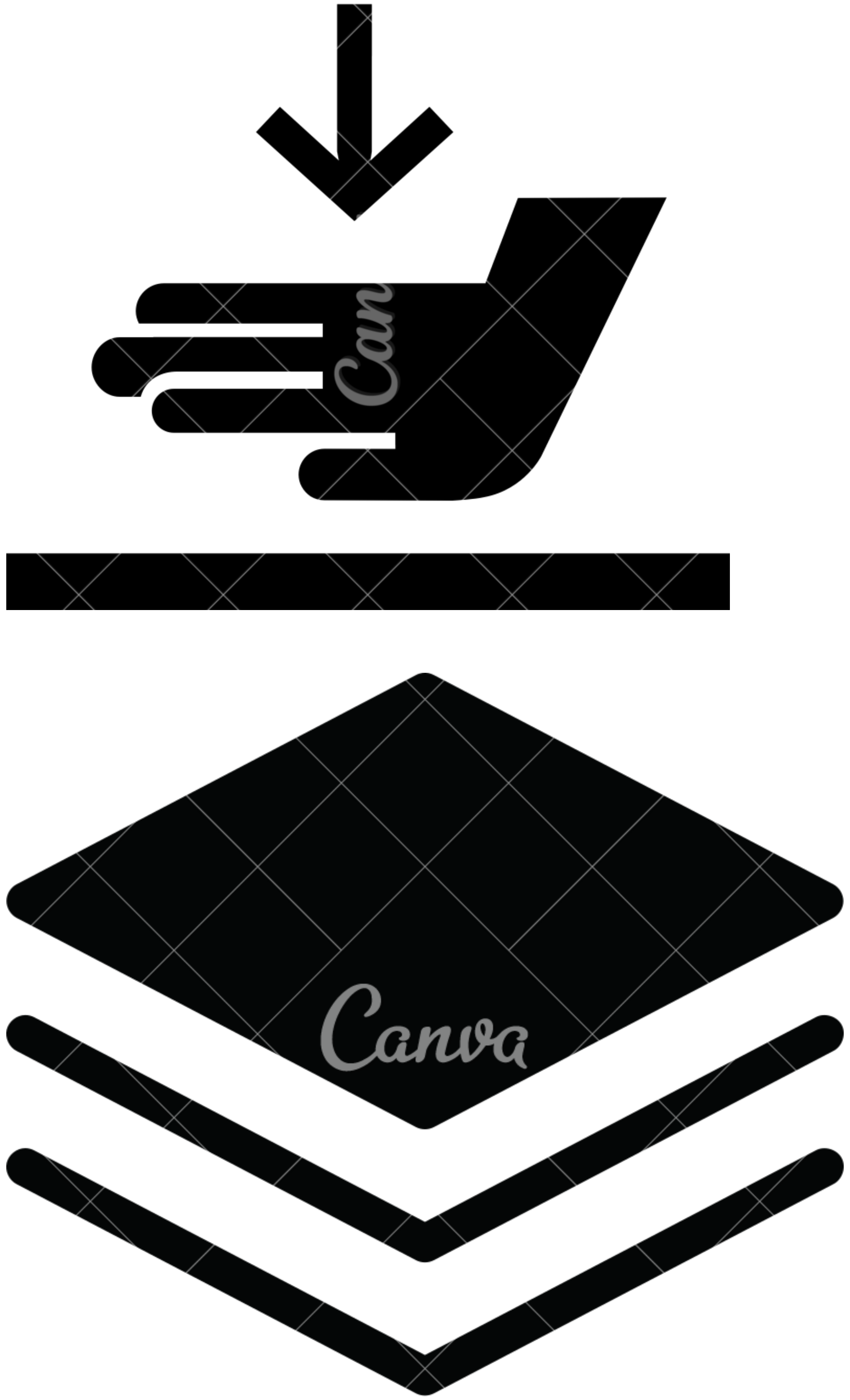
- *Stack of Plates*
- *Stack of Coins*
- *Stack of Books*

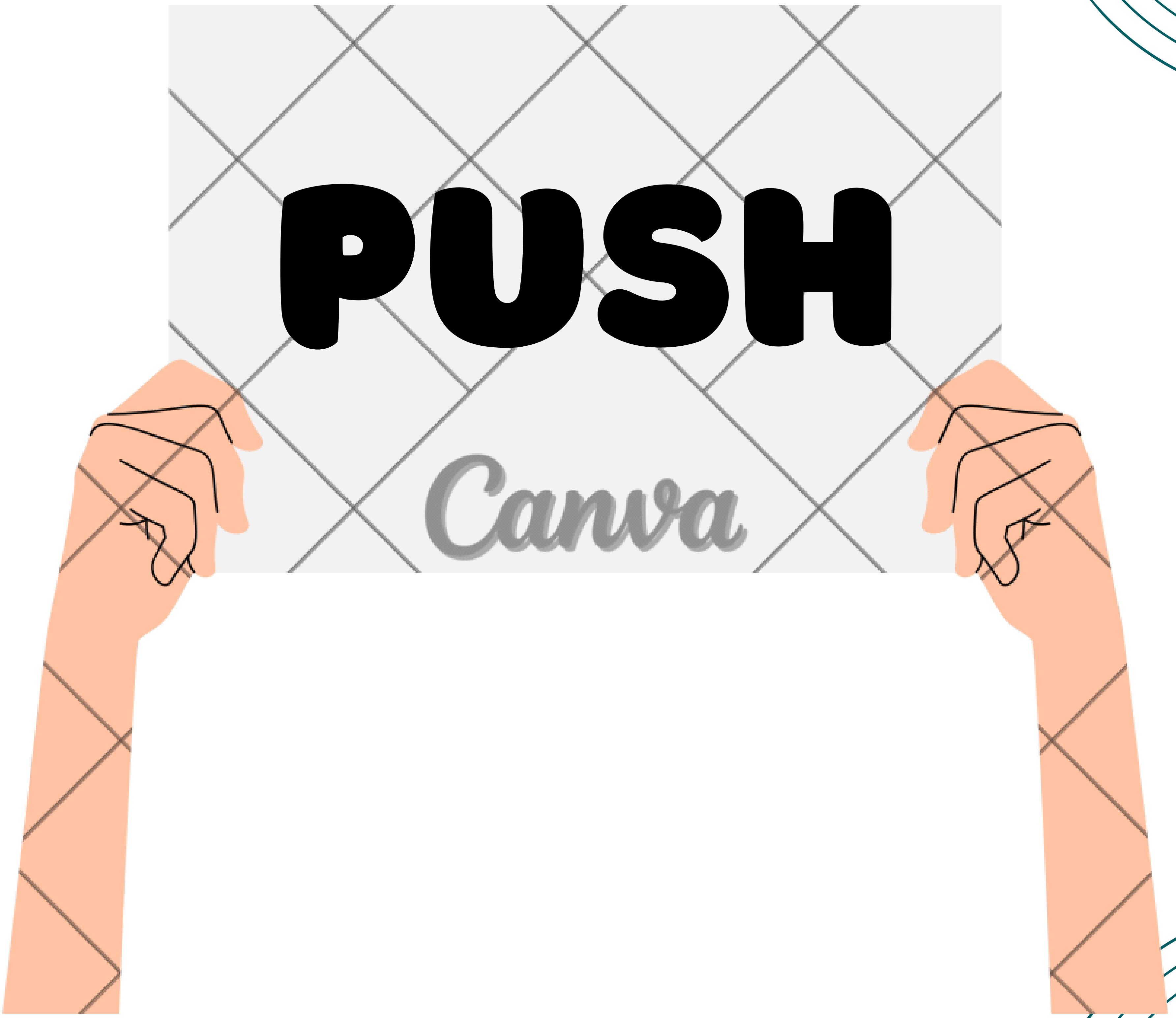
operations on stack

TOP

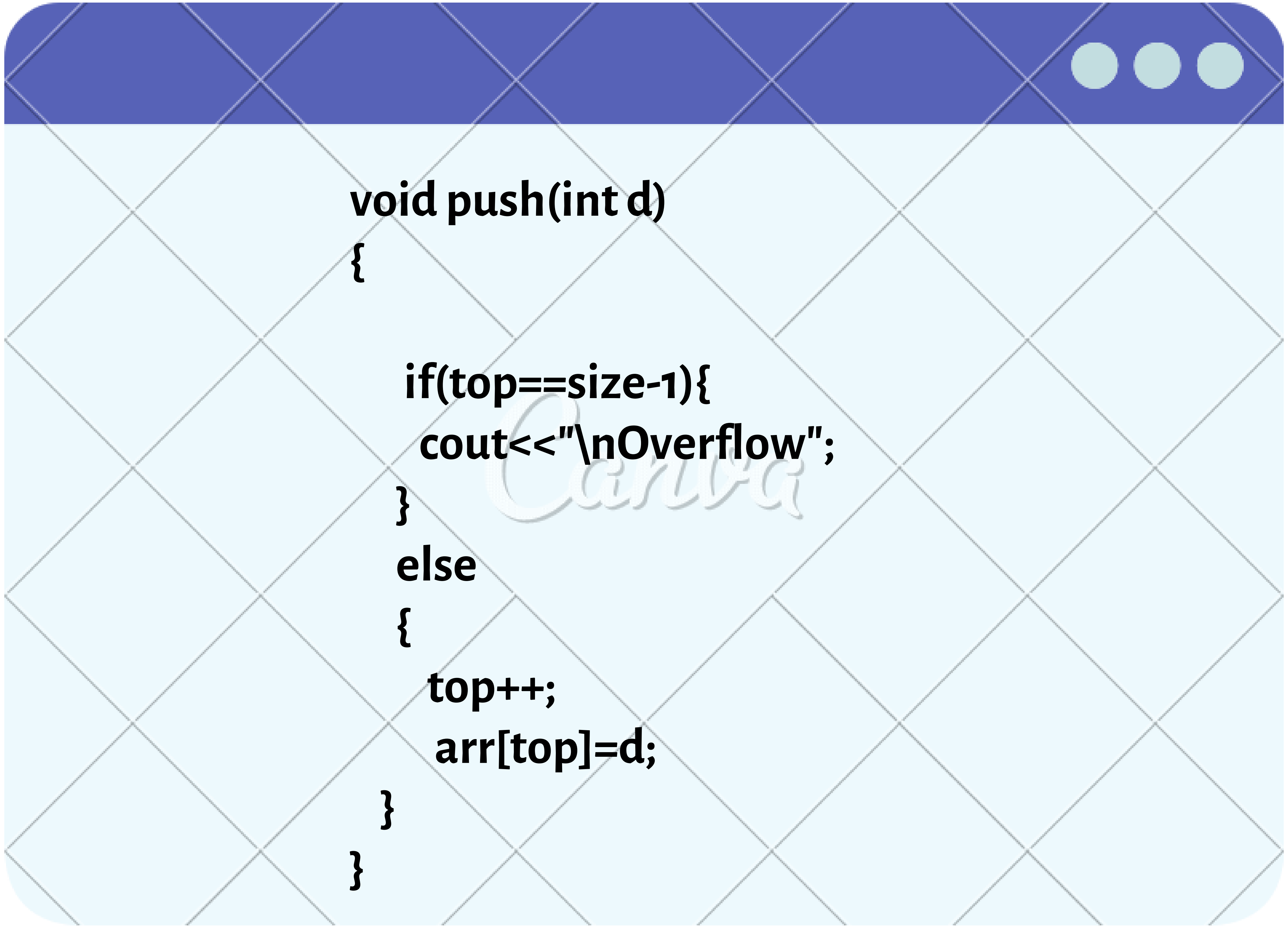
Returns the top element of the stack





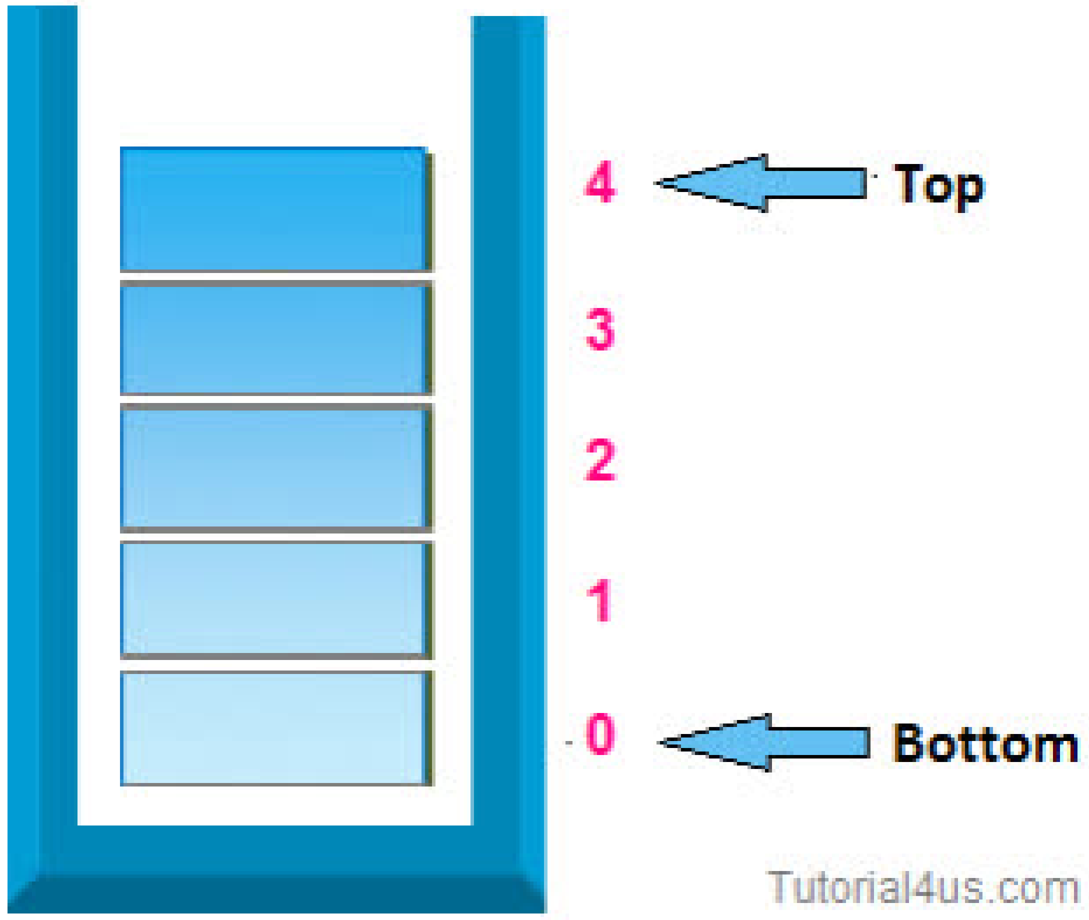

PUSH

push() to insert an element into the stack



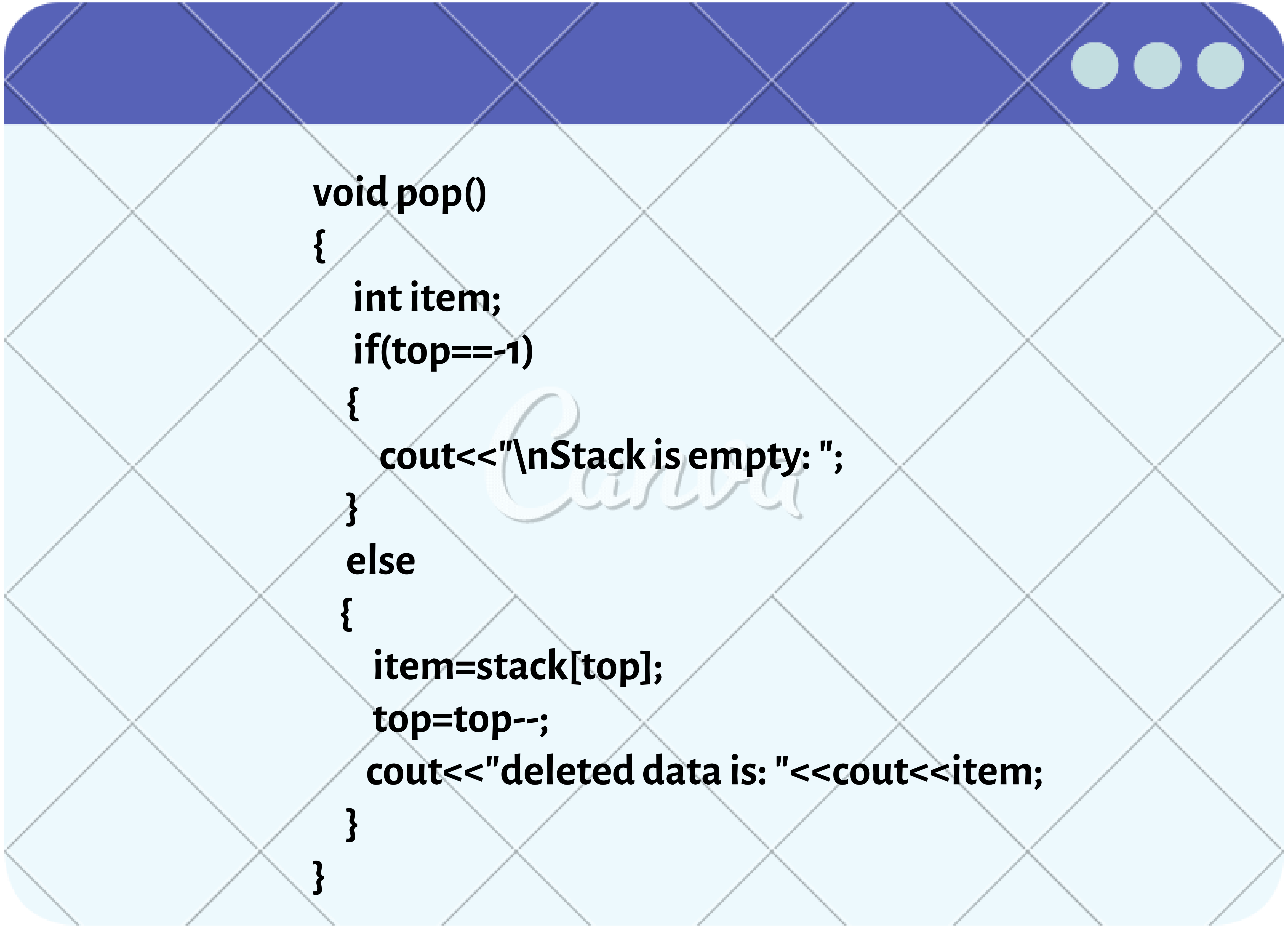
```
void push(int d)
{

    if(top==size-1){
     cout<<"\nOverflow";
    }
    else
    {
       top++;
        arr[top]=d;
   }
}
```

POP



Tutorial4us.com

pop() to remove an element from the stack



```
void pop()
{
    int item;
    if(top==-1)
   {
      cout<<"\nStack is empty: ";
   }
   else
  {
     item=stack[top];
     top=top--;
    cout<<"deleted data is: "<<cout<<item;
  }
}
```

Implementation of stack using arrays

Applications of Stack

- *Function calls and recursion*
- *Undo/Redo operations*
- *Expression Evaluation*

PROBLEMS

1.Valid Parenthesis

i) [( ) [{ ( ) }]]

ii) [(( )]

iii) [( )

iv) [ ) ( ]

PROBLEMS

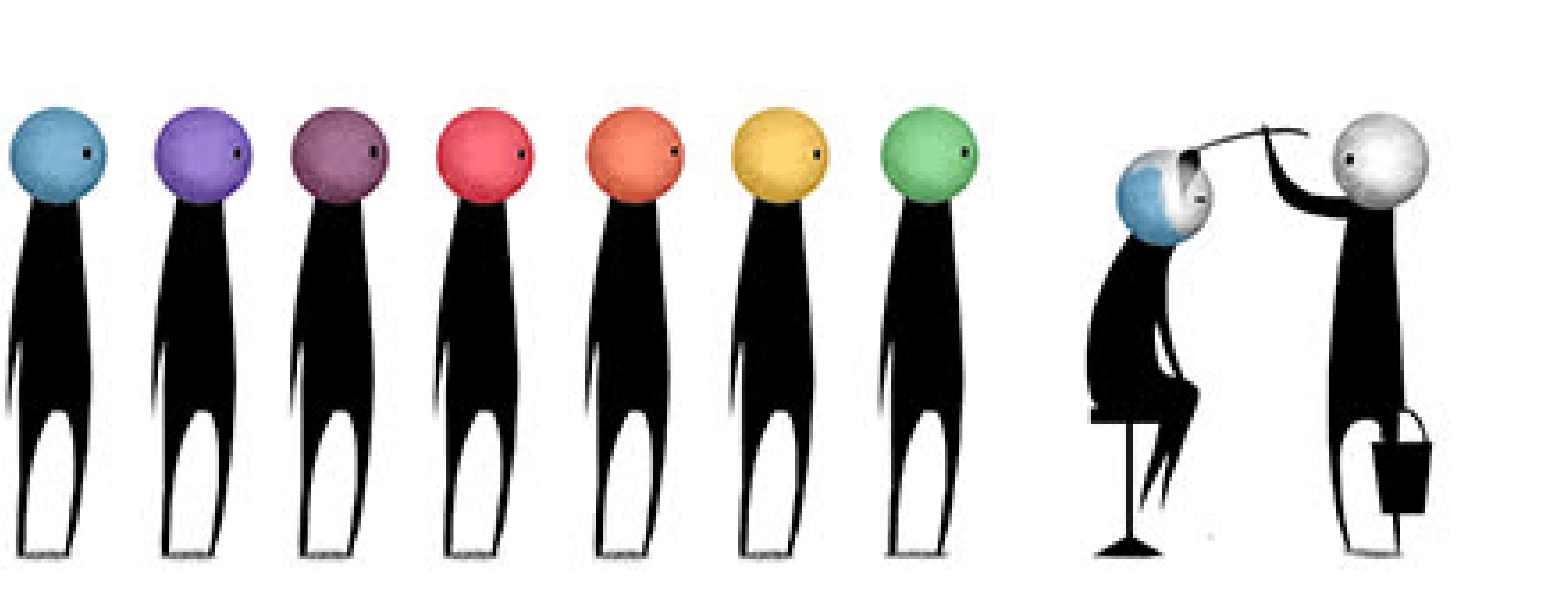
2. Baseball Game

RULES:

An integer x Record a new score of x.
'+' Record a new score that is the sum of the previous two scores.
'D' Record a new score that is the double of the previous score.
'C'. Invalidate the previous score, removing it from the record.

i) ["5","2","C","D","+"]
ii) ["5","-2","4","C","D","9","+","+"]

QUEUES



Queue Based On Principle

LIFO

FIFO

Queue Based On Principle

☒ LIFO

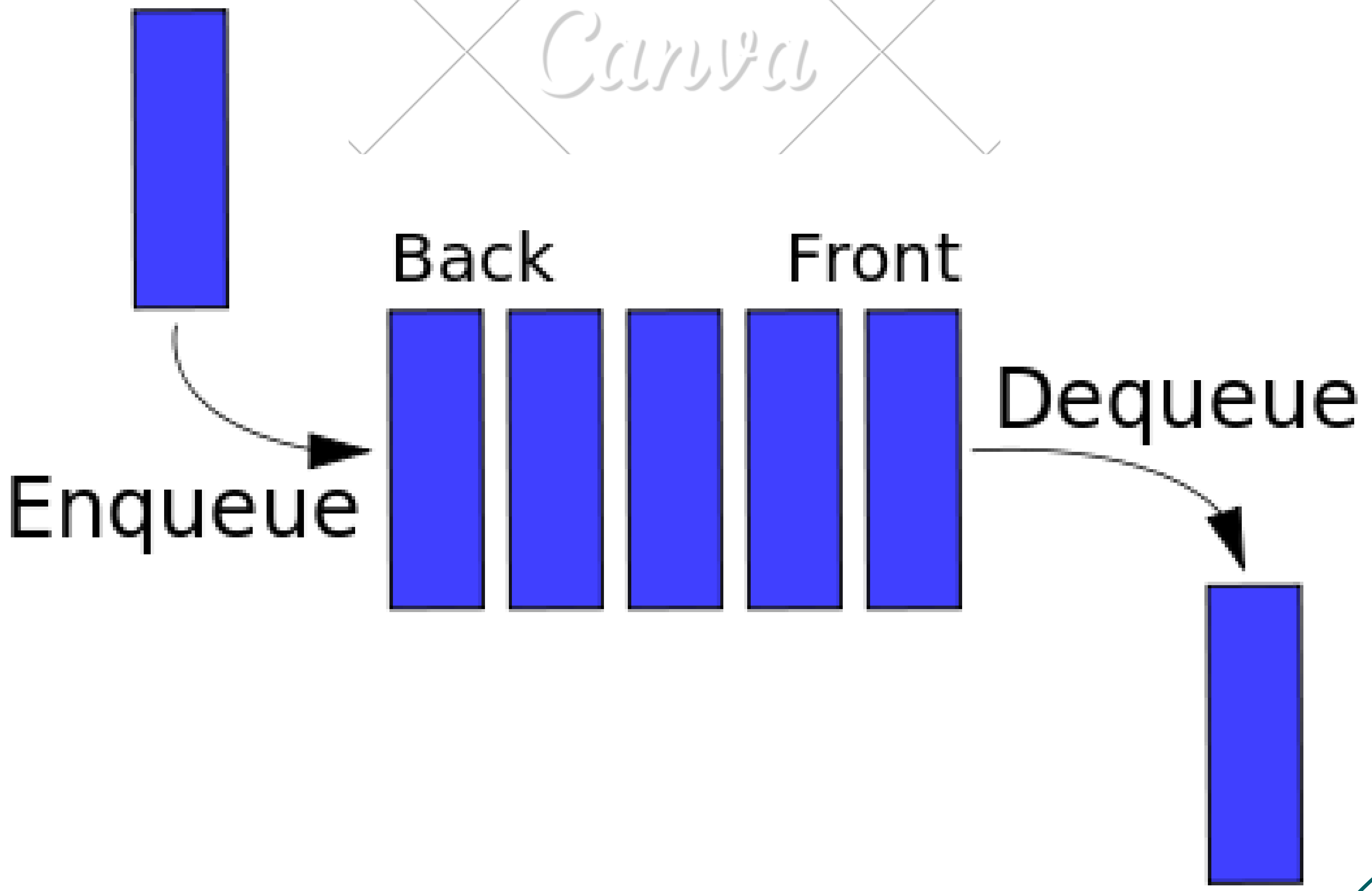
✓ FIFO

QUEUE

- Queue: A Linear data structure
- Add: New elements at the rear end
- Remove: Existing elements from the front end
- Operations: Performed in First In First Out (FIFO) order
- eg. Employees waiting in a line, cashier line in a store , a car wash line.

BASIC OPERATIONS ON QUEUE

- Enqueue
- Dequeue
- Peek
- isEmpty



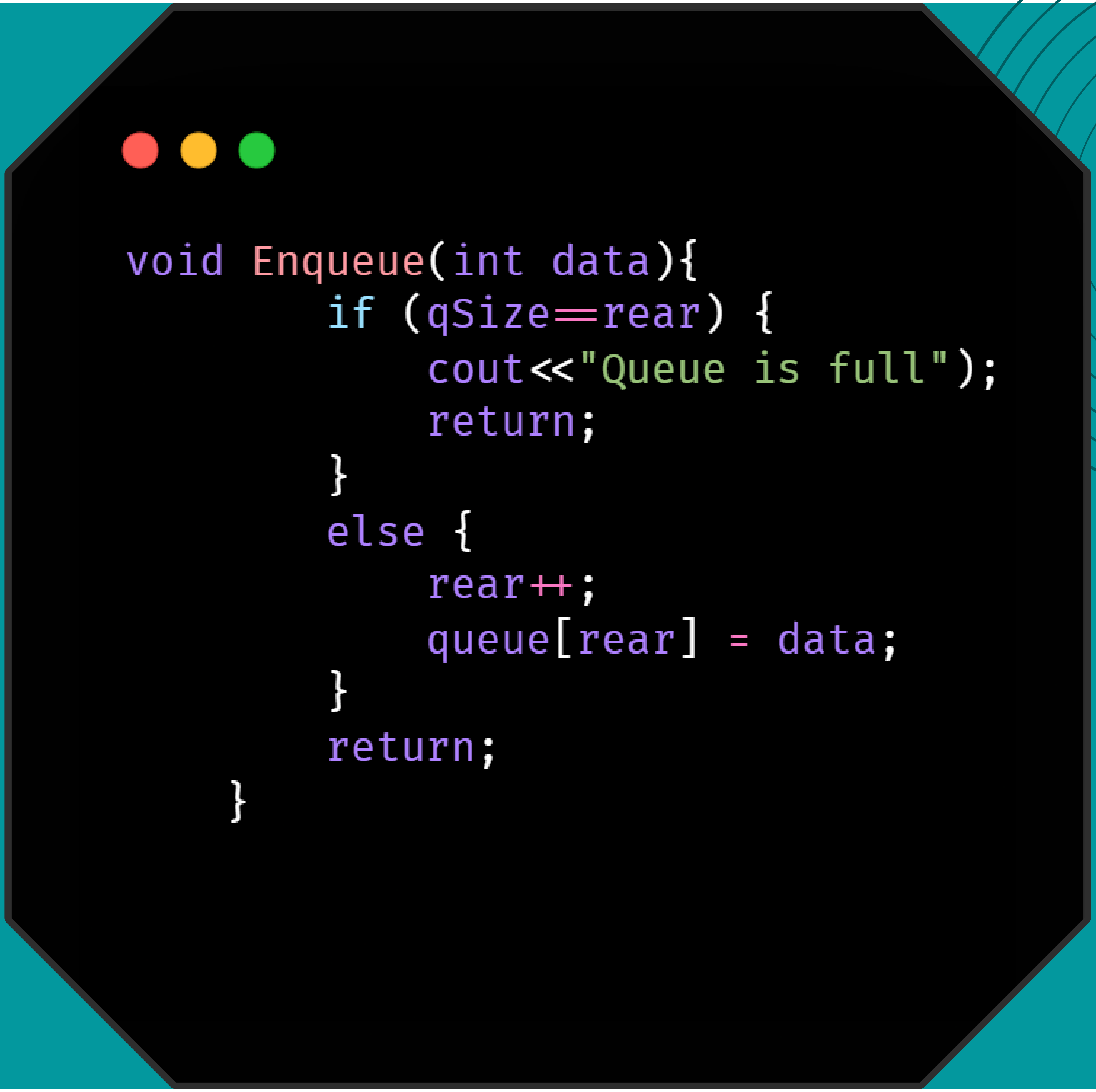
Enqueue

- It is used to **insert** an element to the front of the queue

Steps of Algorithm:

1. Check if the Queue is full.
2. Set the front as 0 for the first element.
3. Increase rear by 1.
4. Add the new element at the rear index

Code



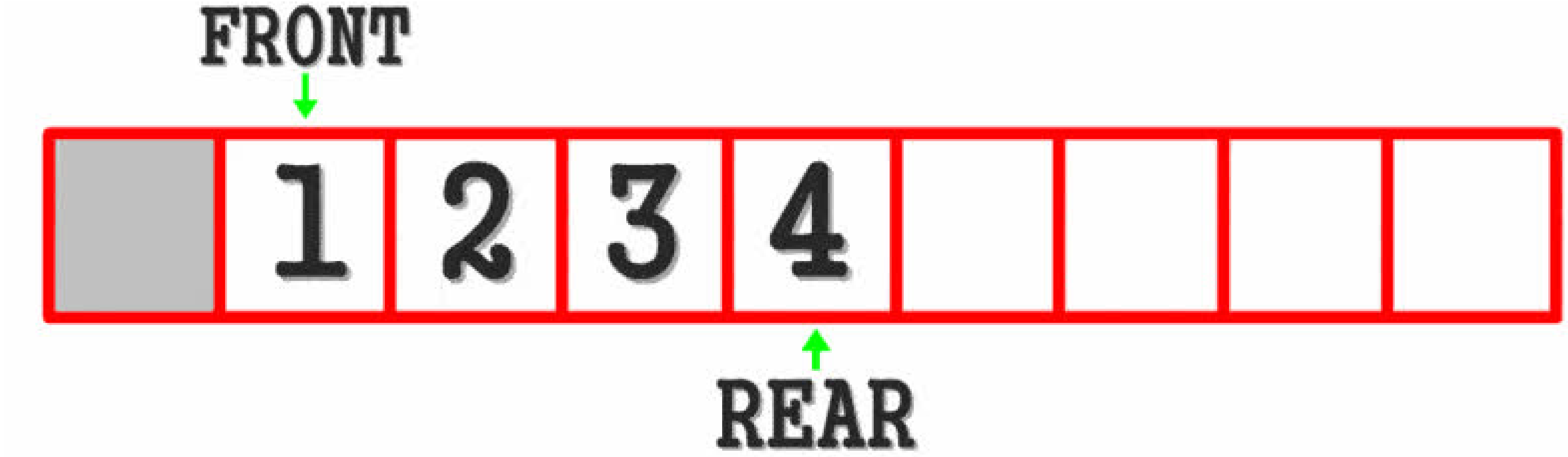
```
void Enqueue(int data){
        if (qSize==rear) {
            cout<<"Queue is full");
            return;
        }
        else {
            rear++;
            queue[rear] = data;
        }
        return;
    }
```

Dequeue

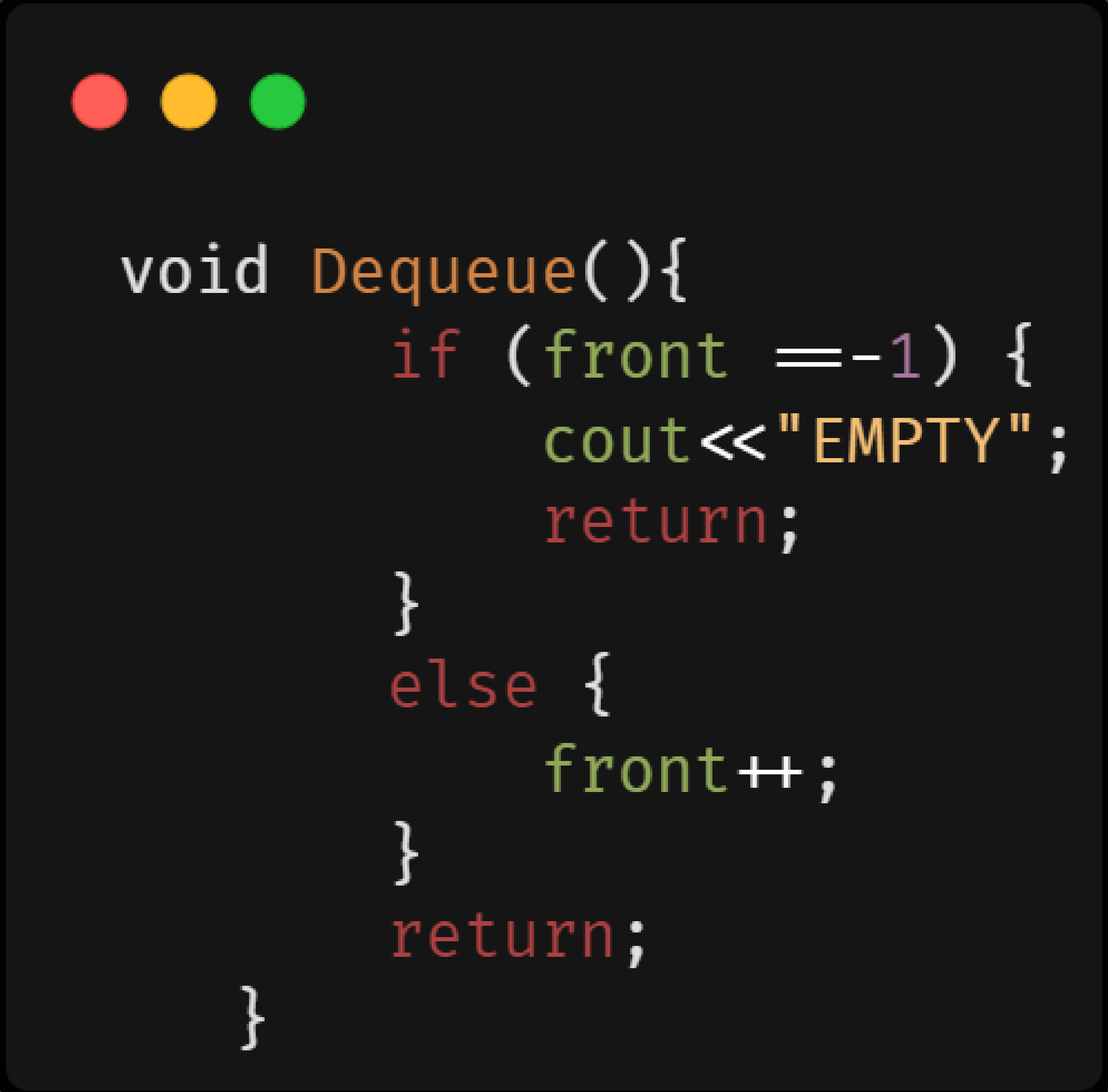
The Dequeue opearation is used to **remove** an element from the rear of the queue

Steps of Algorithm:

1. Check if the Queue is empty.
2. Return the value at the front index.
3. Increase front by 1.
4. Set front and rear as -1 for the last element.


FRONT
1
2
3
4
REAR

Code



```
void Dequeue(){
       if (front ==-1) {
           cout<<"EMPTY";
           return;
       }
       else {
           front++;
       }
       return;
   }
```

Peek

- It is used to **return the front most** element of the queue.

- Steps of Algorithm:
  - 1.Check if the Queue is empty.
  - 2.Return the value at the front index.

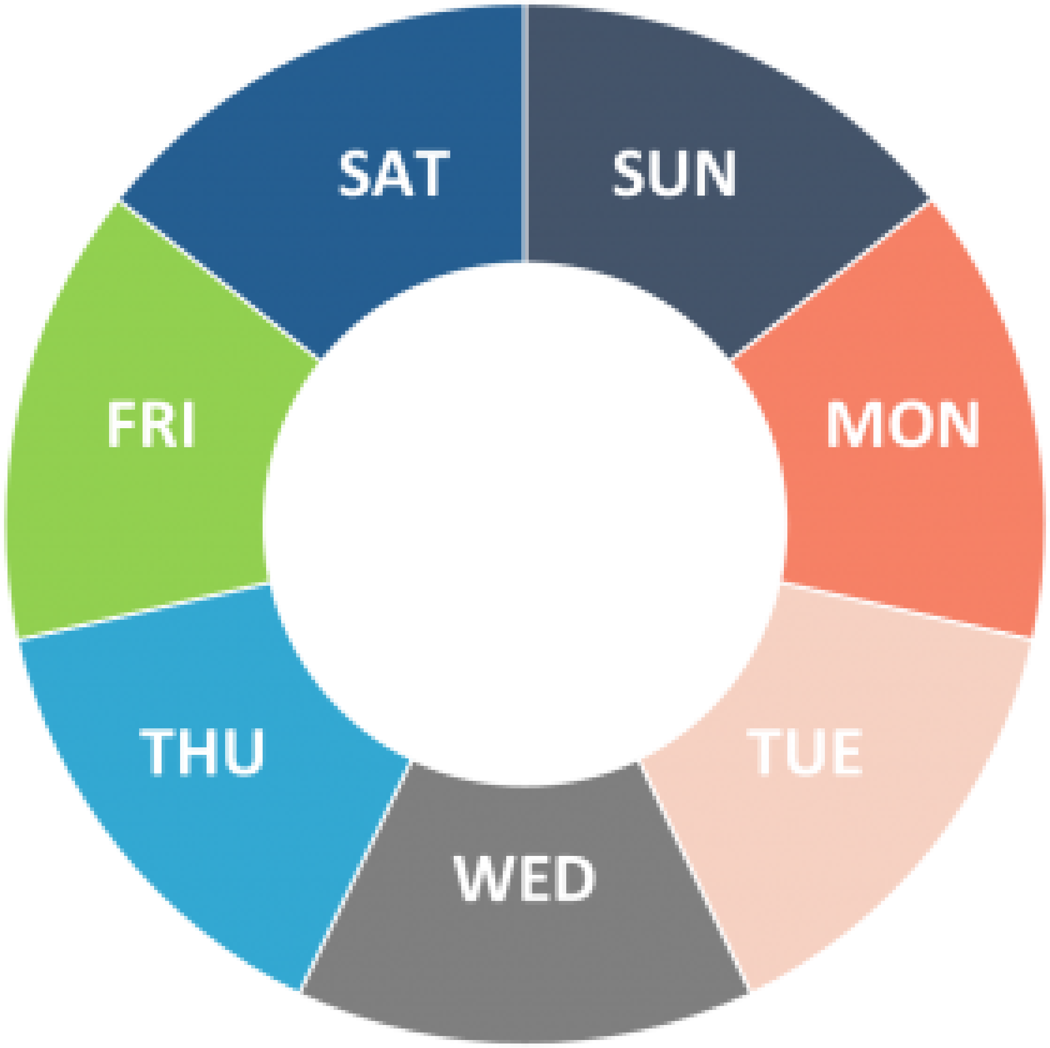
isEmpty

- It is used to check whether Queue is **empty or not.**

- Steps of Algorithm:
  - 1.Check if the number of elements in the queue(size) is equal to 0, if yes return True
  - 2.else Return False.

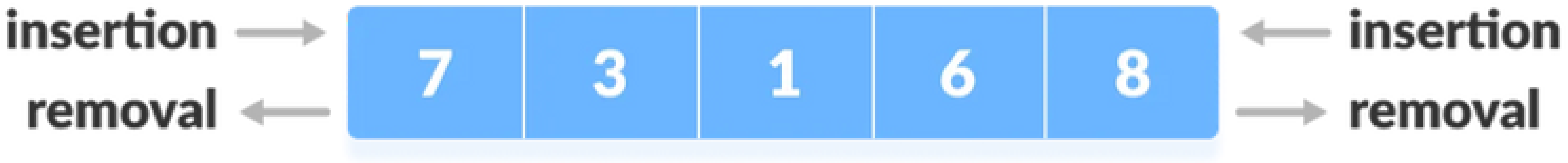
Circular Queue

- Circular Queue: A variation of a simple queue
- Last Member Linked to First: Fo
- Enables efficient use of space and continuous operation without shifting elements.



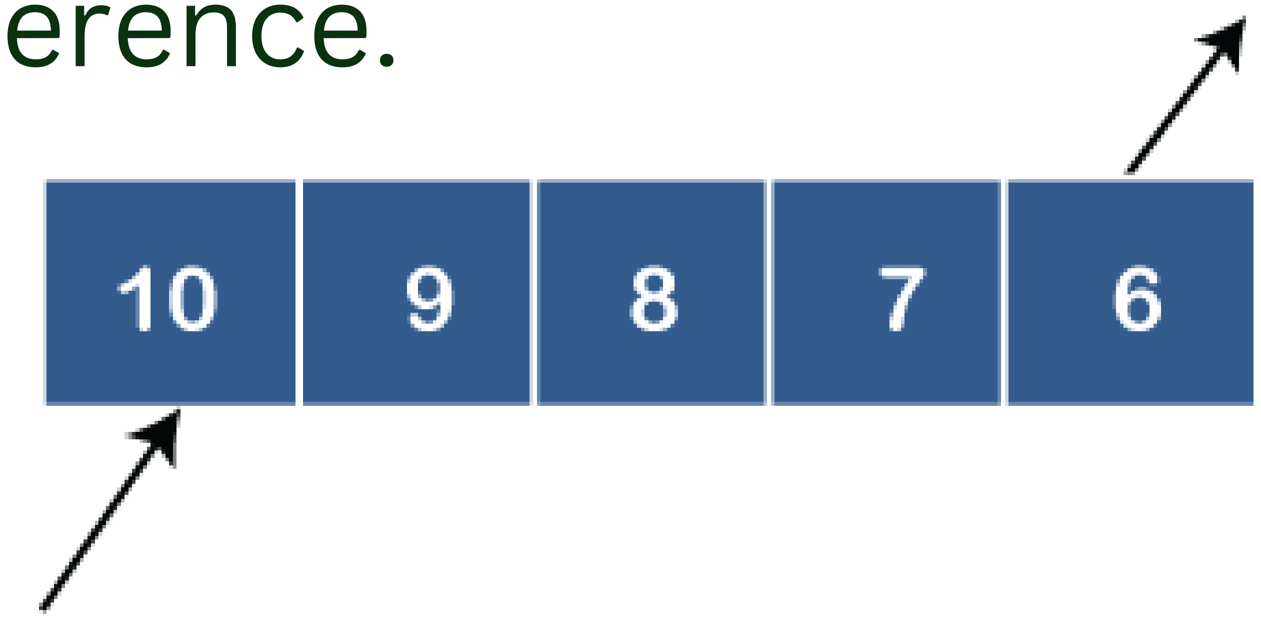
Deque

- Deque (Double Ended Queue): Allows insertion and removal of elements from both front and rear.
- Doesn't strictly follow the FIFO (First In First Out) rule.



Priority Queue

- Priority Queue: Elements are stored with a priority, and higher-priority elements are given preference.



Element with the lowest priority

Element with the Highest priority

Let’s See Implementation

THANK YOU!