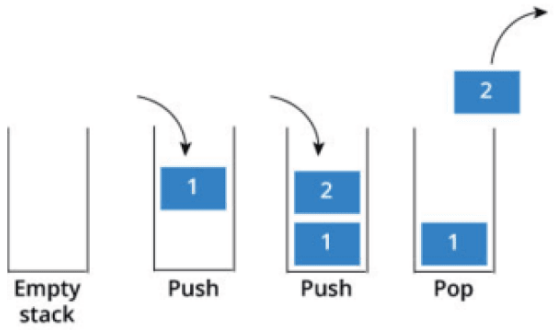
# DATA STRUCTURE & PROGRAMMING II

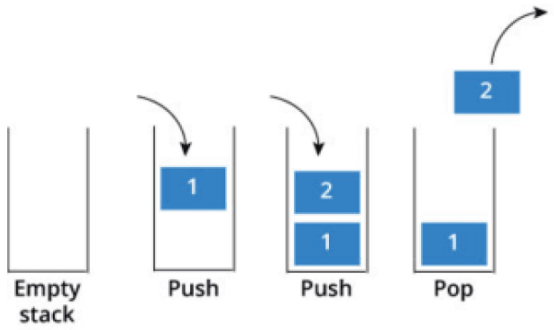
Stack data structure



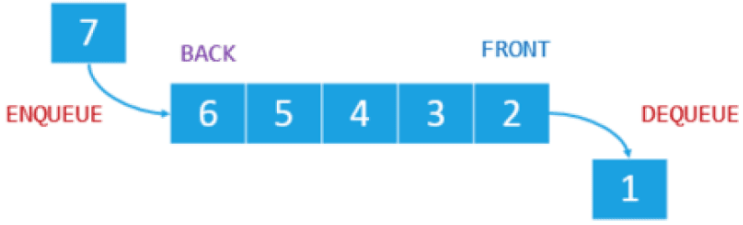

Stack


Queue

# Stack




Stack

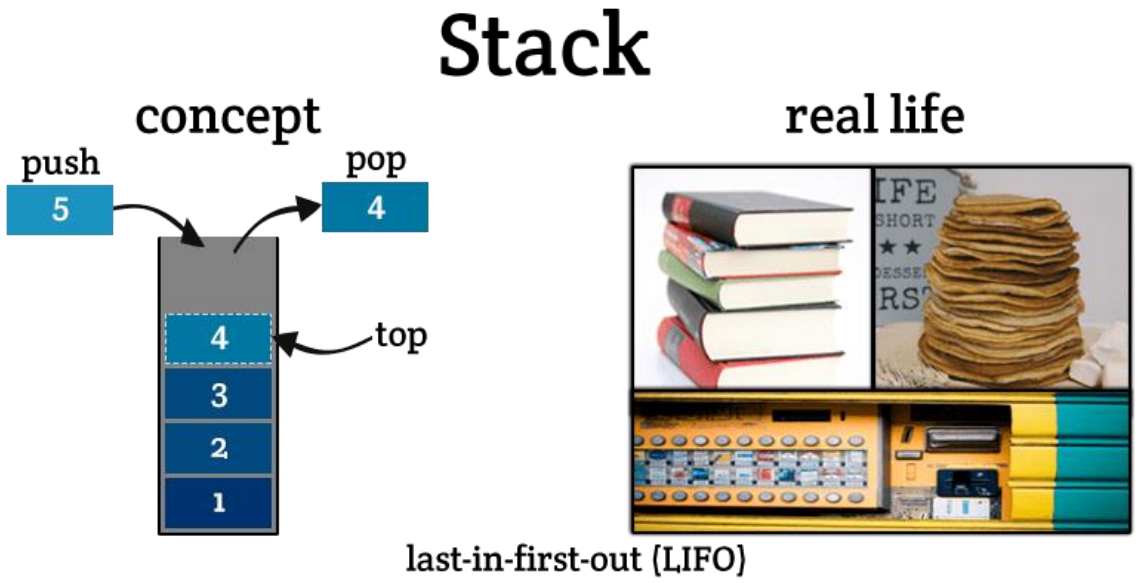



Queue

# Outline

## ❑ A Brief of Outline

- What is Stack?
- What are Stack operations?
- How to implement Stack in C++
- Examples



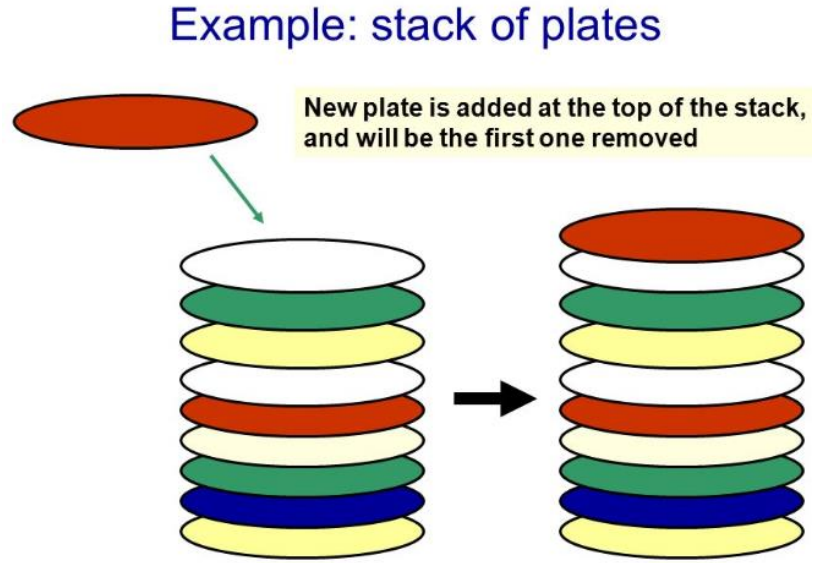
# What is Stack?

## ❑ Definition

- **Stack** is a data structure that stores data in such a way that the element stored last will be retrieved first
- This method is also called **LIFO (Last In First Out)**

**Examples:**

- A stack of copies
  - The first copy put in the stack is the last one to be removed
  - Similarly, the last copy put in stack is the first one to be removed
- Stack of plates
- Stack of chairs

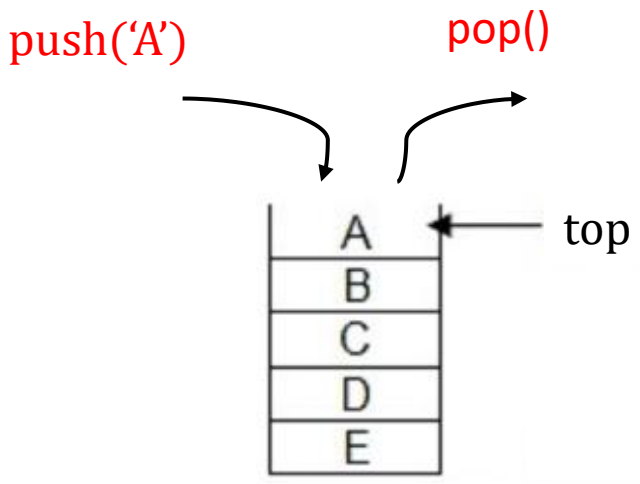


# Applications of Stack

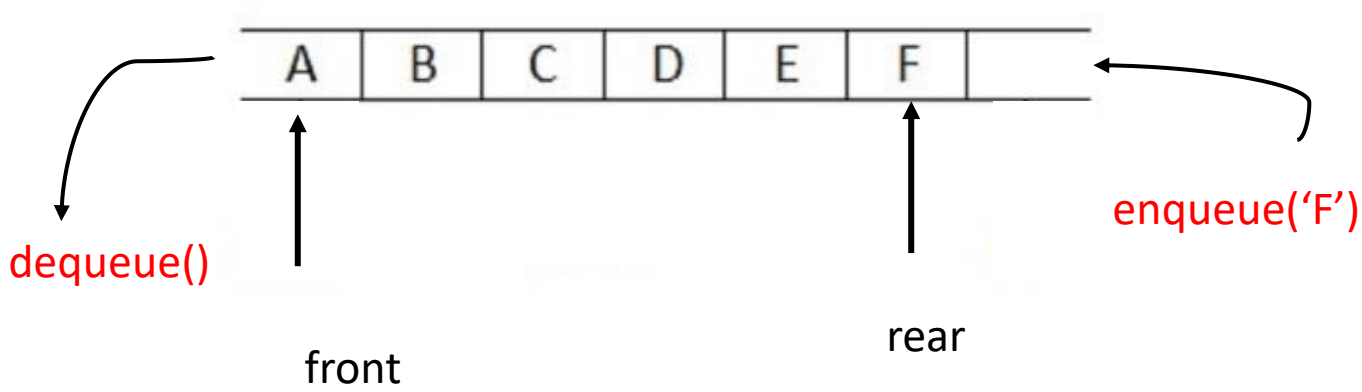
- Undo operation  (browser, Ms. Word, …)
- Remembering completed task
- Design compilers and interpreters
- etc.

# Queue Vs. Stack

- Differences




Stack

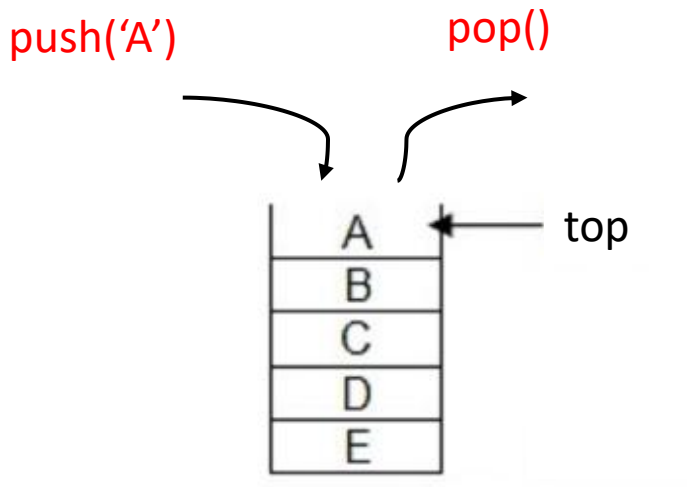



Queue

# Stack Operations

## ❑ Operation

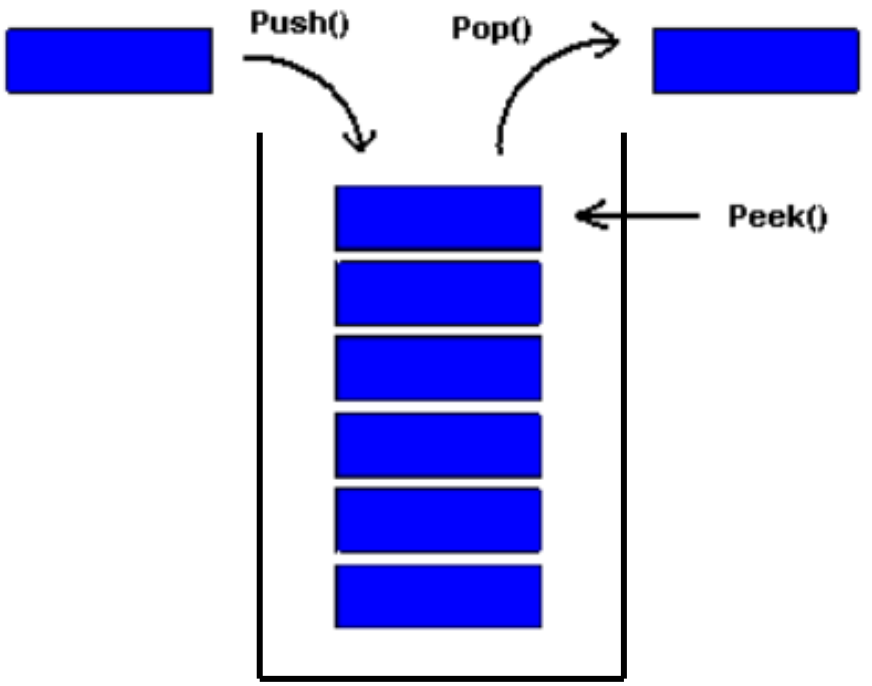
- A **stack** is controlled by two main operations which implement the **LIFO method**
  - *Insertion*
    - Add element to the stack (add to the top)
    - This method is called *push*
  - *Deletion*
    - Remove element from the stack (remove from the top)
    - This method is called *pop*




- The variable TOP is used to keep track of the top element in the stack

# Stack Operations

## ❑ More operations



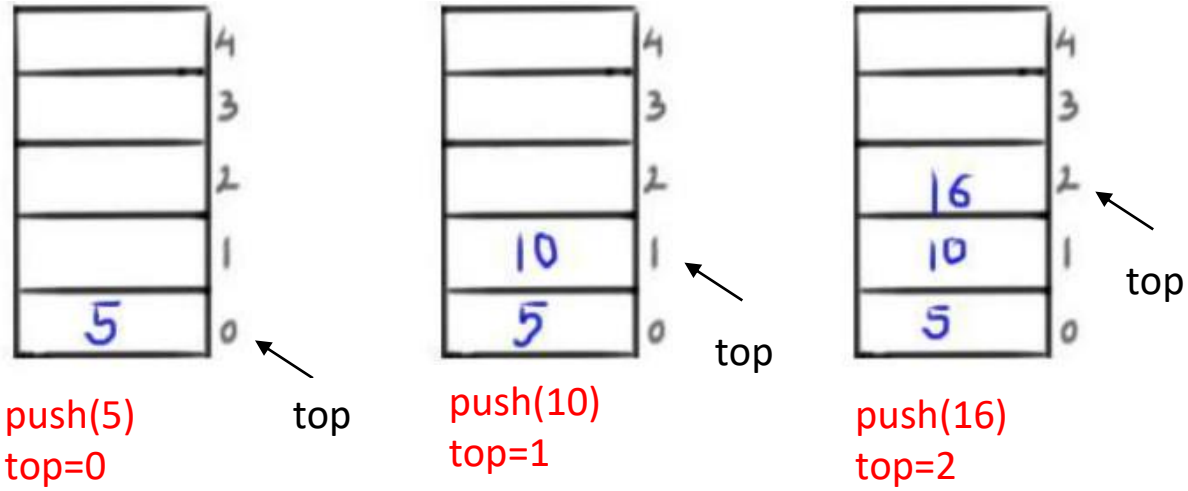

- **`push()`:** Add element to top of stack
- **`pop()`:** Remove element from top of stack
- **`isEmpty()`:** Check if stack is empty
- **`isFull()`:** Check if stack is full
- **`peek()`:** Get the value at the top of stack without removing it

# Stack Implementation

## ❑ Implementation

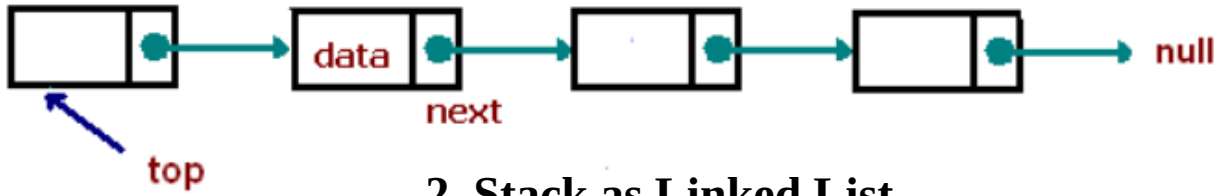
- Stack can be implemented in two way

1. As an Array
   - An array to store data
   - An integer type variable called TOP which stores the index of the top element of the stack




**1. Stack as Array**

2. As a Linked List
   - A linked list to store data
   - A pointer variable called TOP which points to the top element of the list



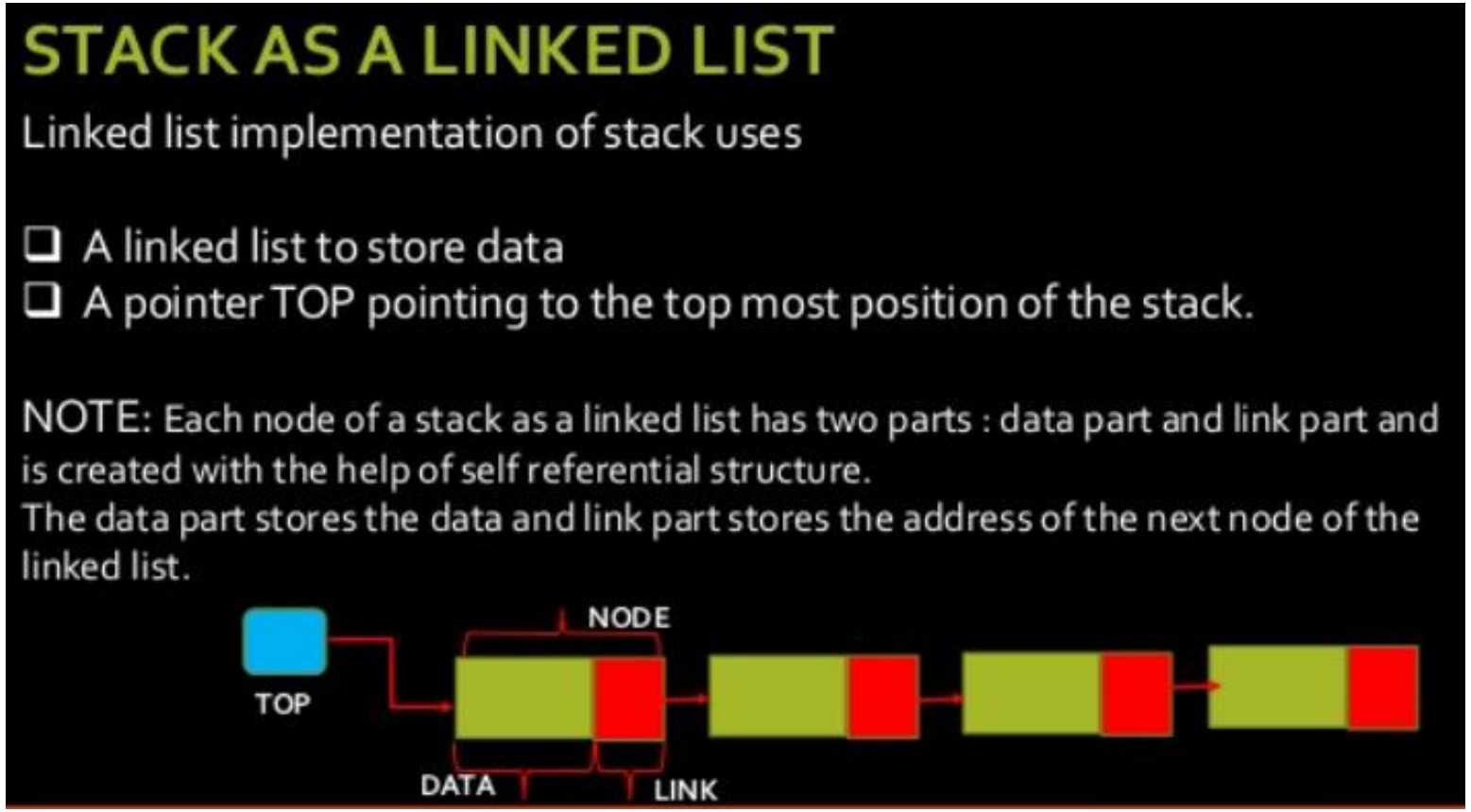

**2. Stack as Linked List**

# Stack as Linked List

- Dynamic
  - It can grow or shrink at runtime

# Stack Implementation

- Stack as a Linked List



# Stack Implementation

## ❑ Stack as a Linked List

- Implementing stack as a linked list is just like implementing a linked list with some choices

  Choice 1

  - Element is added to the first of the list **(push operation)**
  - Element can be only removed from the beginning of the list **(pop operation)**

  Choice 2

  - Element is added to the end of the list **(push operation)**
  - Element can be only removed from end of the list **(pop operation)**

**Remark**: Choice 1 is recommended.

# Stack as Array Vs. Stack as Linked list

## ❑ When to use?

### ➢ When we use array to implement stack?

- Must know exact size of the stack and it is small

### ➢ When we use Linked list to implement stack?

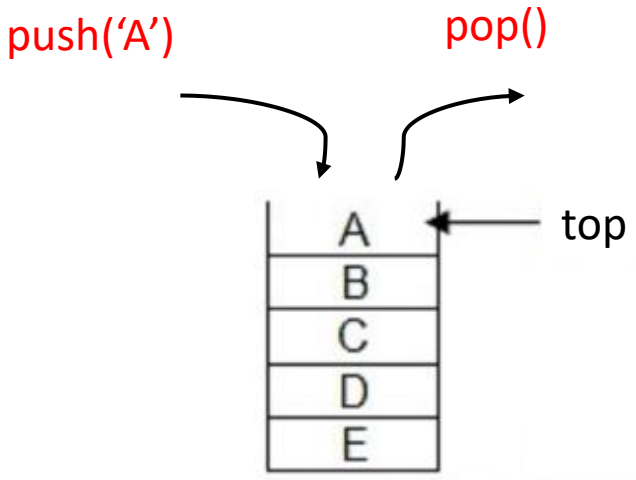
- Do not know the size of the stack.
- If the stack can be bigger and bigger, Linked list will be good to implement it.

# Stack Implementation: Examples

## ❑ Stack as a Linked List

How to implement this Stack?

Demo coding in class



# Q and A

# Homework

## ❑ Reading: **Sorting Algorithms**

1. What is Sorting algorithm?
2. Give as many sorting algorithms as you can.
3. Provide an example on how to implement one of those sorting algorithms in C++?
    1. Give an example of a program implementing a sorting algorithm

# Example 1: Class activity

## ❑ Stack as Array

- Modify code at *Part 3* to obtain the same output below

```
#include<iostream>
using namespace std;
const int SIZE=3;
int stack[SIZE];
int top=-1;

bool isEmpty();
bool isFull();
void push(int item);
void pop();
void display();

int main(){
    push(2);
    push(5);
    push(7);
    push(1);
    display();
    pop();
    pop();
    pop();
    pop();
    display();
}
```

*Part 1*

```
bool isEmpty(){
    if(top==-1){
        return true;
    }
    return false;
}
bool isFull(){
    if(top==SIZE-1){
        return true;
    }
    return false;
}
```

*Part 2*

```
        2 is added to stack
        5 is added to stack
        7 is added to stack
        Stack overflow! can't add
Display data in stack from top: 7 5 2
        7 is removed from stack
        5 is removed from stack
        2 is removed from stack
        Stack underflow! can't remove
Display data in stack from top:  Stack is empty
```

**Expected Output**

```
void push(int item){
    if(isFull()){
        cout<<"\n\tStack overflow! can't add";
    }else{
        //your code
    }
}
void pop(){
    if(isEmpty()){
        cout<<"\n\tStack underflow! can't remove";
    }else{
        //your code
    }
}
void display(){
    cout<<"\n";
    cout<<"Display data in stack from top: ";
    if(!isEmpty()){
        //your code
    }else{
        cout<<" Stack is empty\n";
    }
}
```

*Part 3*

# Example 1: SOLUTION

## ❑ Stack as Array

```
#include<iostream>
using namespace std;
const int SIZE=3;
int stack[SIZE];
int top=-1;

bool isEmpty();
bool isFull();
void push(int item);
void pop();
int peek();
void display();

int main(){
    push(2);
    push(5);
    push(7);
    push(1);
    display();
    pop();
    pop();
    pop();
    pop();
    display();
}
```

Part 1

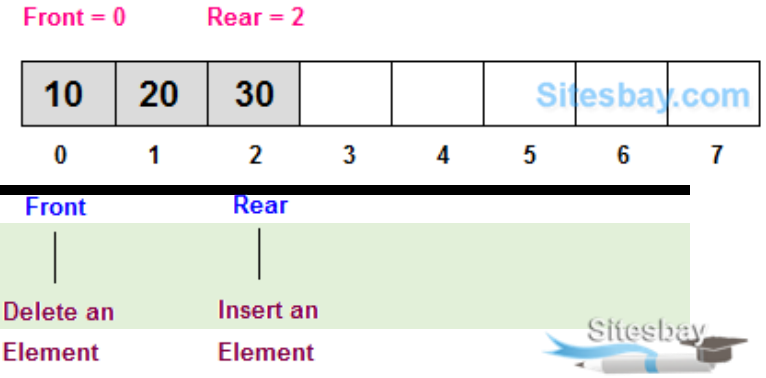
```
bool isEmpty(){
    if(top==-1){
        return true;
    }
    return false;
}
bool isFull(){
    if(top==SIZE-1){
        return true;
    }
    return false;
}
void push(int item){
    if(isFull()){
        cout<<"\n\tStack overflow!
can't add";
    }else{
        top=top+1;
        stack[top]=item;
        cout<<"\n\t"<<item<<" is
added to stack";
    }
}
```

Part 2

```
void pop(){
    if(isEmpty()){
        cout<<"\n\tStack underflow! can't remove";
    }else{
        cout<<"\n\t"<<stack[top]<<" is removed from
stack";
        stack[top]=0;
        top=top-1;
    }
}
int peek(){
    if(isEmpty()){
        cout<<"\n\tStack is empty\n";
    }else{
        return stack[top];
    }
}
void display(){
    cout<<"\n";
    cout<<"Display data in stack from top: ";
    if(!isEmpty()){
        for(int i=top; i>=0; i--){
            cout<<stack[i]<<" ";
        }
    }else{
        cout<<" Stack is empty\n";
    }
    //cout<<"\n";
}
```

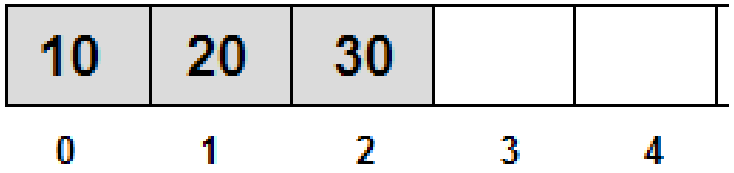
Part 3

# Queue




## ❑ Implementation

➢ ----



# Attendance record

**forms.gle/J3XzFbw3cMe1uBrf6**

- **Attendance submit:**
- **Student review & QA: 3:10 – 3:40pm**
- **Quiz about Queue: 3:45 – 4:00pm (on Moodle)**
- **Stack lecture**