Uses of Stack

Bracket Matching:

Balanced

$(6+5) * ((4-2) / 3)$.

{ } ✓
[ ] ✓
( ) ✓

→ A stack of characters.

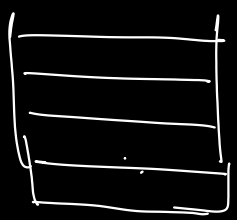
Opening bracket ✓
↳ push it to the stack.

Closing bracket ✓
↳ hold ✓
↳ pop from the stack ✓
↳ type same, Yes ⇒ balanced ✓
↳ type different ⇒ Not balanced ✓

Expression is complete
→ Is stack empty ? No ⇒ Not balanced

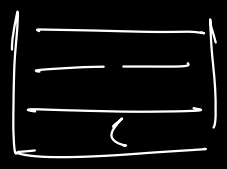
return balanced ✓

Example: → $(6+5) * ((4-2) / 3)$



1. (popped) read — balanced ✓
2. (popped) read — balanced ✓
3. (popped) read — balanced ✓
4. empty popped) read — Not balanced

((5-2)+3



top →

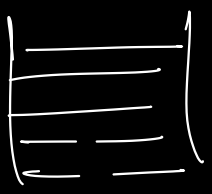
1- ( ) read balanced

Is stack empty ?

No

Not balanced.

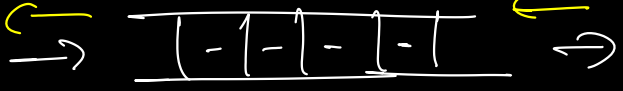
{5-2) * 4



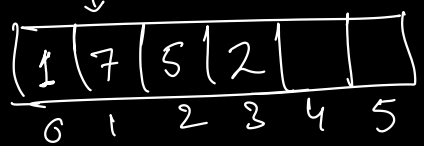
{ ⟷) read Not balanced

poped

Queues



dequeue() : 1



front = ~~0~~ 1 rear = ~~-1~~ 0 ~~1~~ ~~2~~ 3

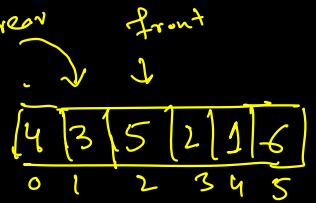
enqueue(1)
enqueue(7)
enqueue(5)
enqueue(2)

```
class Queue
{
private:
    int *arr;
    int front;
    int rear;
    int size;
    int noOfElements;
public:
    Queue(int s)
    {
        arr = new int[s];
        size = s;
        front = 0;
        rear = -1;
        noOfElements = 0;
    }
```

```
void enqueue(int val)
{
    if (isFull())
    {
        cout<<"Queue overflow"<<endl;
        return;
    }
    if (rear == (size - 1))
        rear = 0;
    else
        rear++;

    arr[rear] = val;
    noOfElements++;
}
```

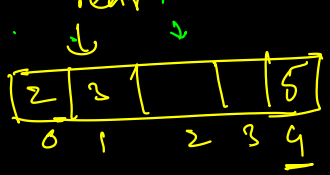
```
bool isFull()
{
    if (noOfElements == size)
        return true;
    else
        return false;
}
```



front = 2
rear = ~~5~~ 0 1
enqueue(4)
enqueue(3)

```
int dequeue()
{
 if(isEmpty())
 { cout<<"Queue Underflow"<<endl;
   return;
 } int val = arr[front];
  if(front == (size-1))
     front =0;
  else
     front++;
  noofElements--;
  return val;
}
```

```
bool isEmpty()
{
  if(noofElements ==0)
     return true;
   else
     return false;
}
```



dequeue(): 5

dequeue(): 2

dequeue(): 3

front = 4 ⊘ 1

rear = 1