Parenthesis matching using stack.

$(6+(2)) - 5$ { } [ ]

→ Number
↳ operand
→ operator. (+, −, ∗, /, ^)
→ opening / closing brackets.

Balanced? $(6+2) / (5*3)$

1. Read the expression (left to right)
2. If an opening bracket ('(', '[', '{'), push on to the stack.
3. If closing bracket (')', ']', '}'), pop from stack. Match closing & opening bracket. If match balanced, continue. Not balanced otherwise
4. After reading the entire expr, if stack is not empty ⇒ Not balanced
   ↘ opening bracket

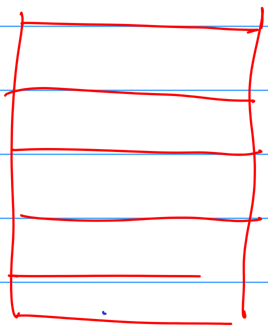
$(6+2) / (5 * 3)$

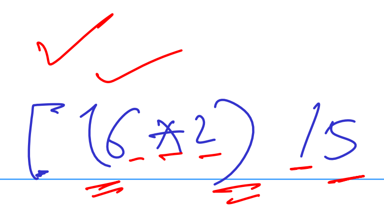
$[(6+2) / 5] / 6 \}$

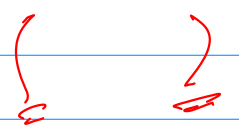
( ) Balanced ✓

[ ] Balanced ✓

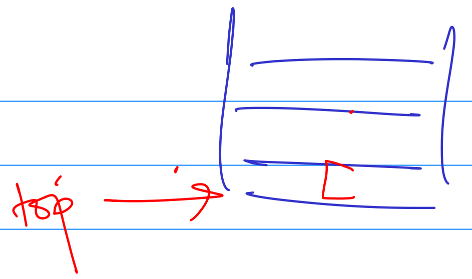
} Not balanced   top = −1







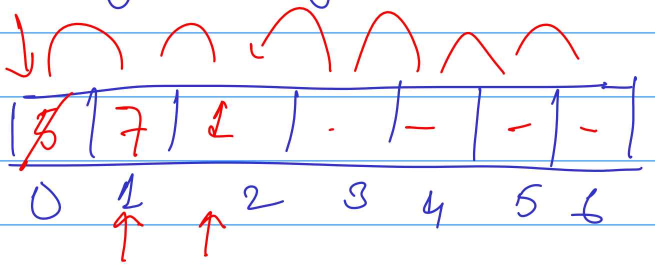
Balanced.



⇒ Stack is not empty ⇒ Not Balanced.

Queue (Using Arrays)

expensive

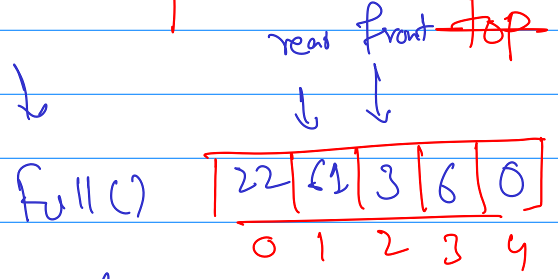


first in first out

Enqueue (5)
Enqueue (7)
Enqueue (1)

dequeue ( ) → 5

front rear



enqueue (22)

length == size ⇒ full.

```
bool isFull()
{
if ( ( front == 0 && rear == (size - 1) ) ||
     ( rear == (front - 1) ) )
{
     return true;
} else   return false;
}
```

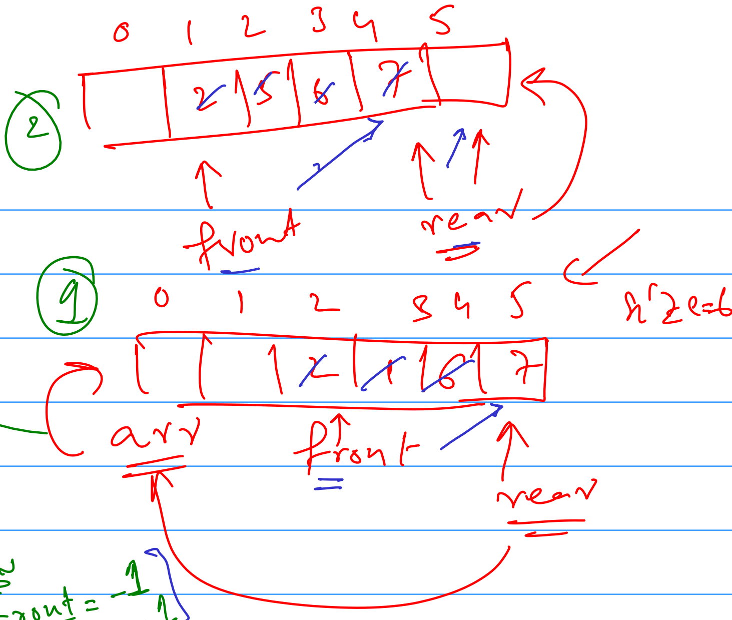
```
node* arr;
int size;
int length;
int front;
int rear;
```

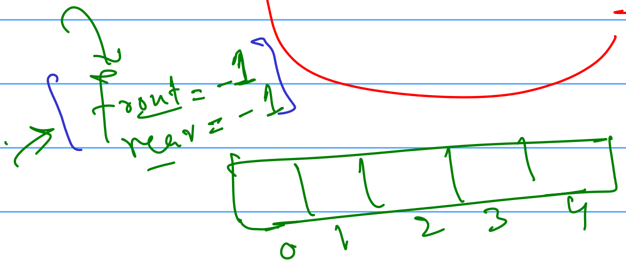
```
void Enqueue (int val)
{
  if (isFull())
  {
     ~~return;~~  }
```

} you are allowed to insert

val

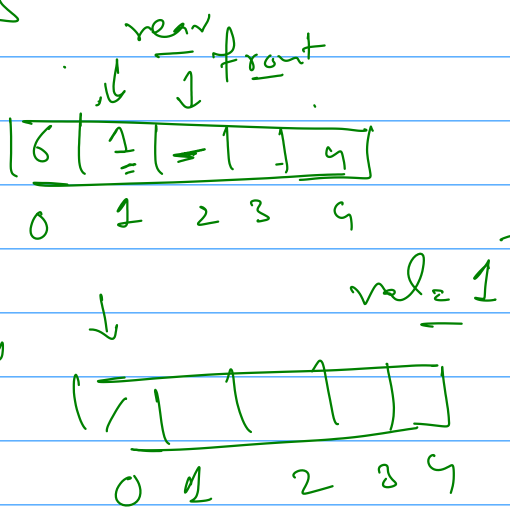
```
1 if( rear == (size-1))
  {
      rear = 0;
      arr[rear] = val;
  }
2 else
  {
      rear++;
      arr[rear] = val;
  }
      length++;
}
```





```
int dequeue()
{
  if (isEmpty()) 
  {    Underflow
       return
  }
     int val = arr[front];
     if(front == (size-1))
         front = 0;
     else
         front++;
     length--;
       return val;
}
```

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



rear = -1
front = 0