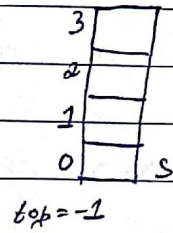
(8)

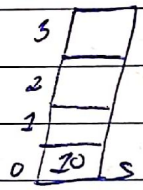
# Implementation of stack using array :-

- Elements will be kept in an array.
- We need one variable to keep track of the "top" of the stack.

Push 10 ✓

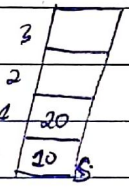



top = top + 1;
top = 0
S[top] = 10




Push 20

top = top + 1
∴ top = 1
S[top] = 20

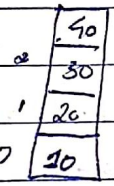



Push 30

top = top + 1
top = 2
S[top] = 30


Push 40

top = 3




Stack overflow :-

→ when the value of top reaches (max. size - 1) we cannot push any more element as there is no more space in the array.

Procedure push (S, MAXSIZE, top, element)

1. If (top == MAXSIZE-1) Then
2. write ('Stack Overflow').
3. Return
4. End If.
5. top = top + 1
6. S[top] = element
7. Return.

Lecture — 6

Date
25/09/21

Procedure pop (S, top)

1. If (top == -1) Then
2. Write ('stack underflow')
3. Return
4. End If
5. Write ('Deleted Element: ', s[top])
6. top = top - 1
7. Return

Procedure display (S, top)

1. If (top == -1) Then
2. Write ('Stack underflow')
3. Return
4. End If
5. Repeat step 6 for I = 0, 1, ---, top
6. Write ('The Element:', s[I])
7. Return.

The C Programming Language by — Kernighan & Rechie.

```
#define <stdio.h>
#define maxsize 20

int s[maxsize], top;

int main() {
    top = -1;
    return 0;
    push();
    push();
    pop();
    push();
    display();
    return 0;
}
```

Date 27/09/21

Lecture-7

```
void push() {
    int element;
    printf("Element : ");
    scanf("%d", &element);
    if (top == maxsize - 1)
    {
        printf("Stack Overflow\n");
        return;
    }
    s[top++] = element;
    return;
}
```

```
void pop() {
    if (top == -1)
    {
        printf("Stack underflow");
        return
    }
    top
    printf("Popped element is: %d\n", s[top]);
    top = top - 1;
    return;
}
```

* → or can be combined as

```
printf("Popped Element is : %d\n", s[top--]);
```

```
void display() {
    int i;
    for (i=0; i <= top ; i++)
        printf("Element is : %d \n", s[i]);
    return;
}
```

for top to bottom

```
for(i=top; i >=0; i--)
    printf - ---
```

[ Array is implicitly passed by reference ].

## Queue

A queue is a linear list where all the insertions are done at one end of the list (known as rear) and all the deletions are done at the other end of the list (known as front). A queue exhibits FIFO property.

Example:-

Insert : 10 20 30

( )

(10)

(10 20)

(10 20 30)

Delete : Deleted element is 10

(20, 30)

Delete : Deleted element is 20

(30)

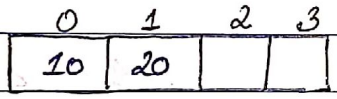
12

Delet : Deleted element is 30.
( )

Insertion Order : 10 20 30 FIFO
Deletion Order : 10 20 30

# Implementation of Queue using Array :

- We use an array to keep the elements of the queue.
- We need two more variables to track of the 'rear' end and the 'front' end.
We call these REAR and FRONT.




Rear = 0 , front = 0

① Insert 10 : Rear = Rear + 1
Q[Rear] = 10
front = front + 1

② Insert 20 : Rear = Rear + 1
Q[Rear] = 20

L-8

Date 29/09/21

③ Insert 30 : Rear = Rear + 1
Q[Rear] = 30

④ Insert 40 : Rear = Rear + 1
Q[Rear] = 40

⑤ Insert 50 : No more space in array.
Queue Overflow or ~~Queue~~ Queue full cond^

## Queue Overflow or Queue full Cond".

Q

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 10 | 20 | 30 | 40 |

Rear = 3 , front = 0

#define maxsize 4.
int Q[maxsize];

To give error on Queue Overflow [prevent and return] we use :-

If (Rear == maxsize - 1)

## Deletions :-

① Delete

② front = front + 1

Q

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 10 | 20 | 30 | 40 |

#dl :- Rear = 3 , front = 1

Queue is always from index 'front' to index 'Rear'

2° Delete : front = front + 1 → Rear = 3 , front = 2

'20' is deled.

③ Delete : front = front + 1 → Rear = 3 , front = 3

30 is deleted

Q

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| ~~10~~ | ~~20~~ | ~~30~~ | 40 |

Rear = 3 , front = 3

④ In this case,

40 is deleted

& Rear = -1 , front = -1 → [Only set when last element i.e. when front = Rear & ≠ -1]

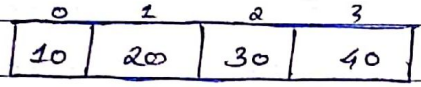
⑤ Delete :- Cannot because

Queue Underflow or Queue Empty Condition.

(14)

# Circular Queue :-

Initially, Rear = -1 , Front = -1 maxsize = 4




Rear = 3 , Front = 0

✓ Delete : Q[Front] i.e. Q[0] → [Front = 1 , Rear = 3]

[Insert 50 :- will be inserted in Q[0]]

✓ Delte Q[1]

[Insert 50
Insert 60]

Insertion circular, wherever place available it fills. This solves the space wastage problem in normal queue..

Circularly Updated :-
Rear = (Rear+1) mod maxsize → modulus operator (%)

Example when Rear = 3

Rear = (3+1) % 4
= 4 % 4
= 0

So, next insertion will be at Q[0]

Again insert,

Rear = (0+1) % 4 ∵ after prev. insertion, Rear=0
= 1 % 4
= 1

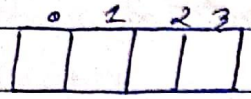
So, insertion will be at Q[1]

~~Circular update update~~ Circular updating.

## Complete e.g. on Circular Queue Insertion & Deletion :-

Insert 10, 20, 30, 40

① Q




Rear = -1, Front = -1

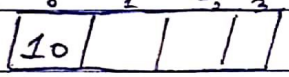
② Insert 10

Rear = (Rear + 1) % 4 , 4 = max size

= (-1 + 1) % 4

= 0 → And set Front = 0

Q

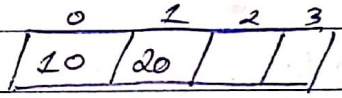



Rear = 0, Front = 0

③ Insert 20

Rear = (0 + 1) % 4

= 1

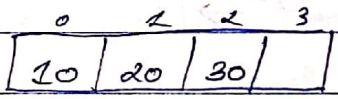



Rear = 1, Front = 0

④ Insert 30 :-

Rear = (1 + 1) % 4

= 2

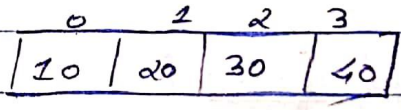



Rear = 2, Front = 0

⑤ Insert 40 :-

Rear = (2 + 1) % 4

= 3




Rear = 3, Front = 0

* Circ. Queue full cond$^{n}$ (i) → If (front == 0) & (Rear == max size - 1)

⑥ Delete :

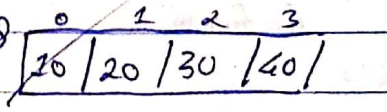
Delete Q[front] i.e. Q[0]

front = (front + 1) % max size

= (0 + 1) % 4

= 1

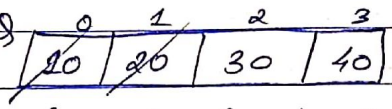
Q




Rear = 3, Front = 1

⑦ Delete:

Front = (1+1) % 4
= 2

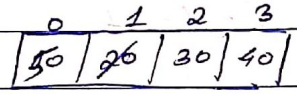



Rear = 3, Front = 2

⑧ Insert 50:

Rear = (3+1) % 4
= 4 % 4
= 0

Q[Rear] = 50
i.e. Q[0] = 50

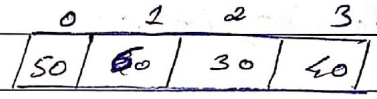



Rear = 0, Front = 2

⑨ Insert 60:

Rear = (0+1) % 4
= 1

Q[Rear] = 60
∴ Q[1] = 60




Rear = 1, Front = 2

Queue full cond$^n$ → If (Rear+1) % maxsize == Front

—x—

## Lecture – 9

Date 04/10/21

### Normal Queue:

Insert: enqueue , Delete: dequeue

```
Procedure Enqueue(Q, Rear, Front, maxsize, element)
1. If (Rear == maxsize - 1) Then
2.     Write ("Queue Overflow")
3.     Return
4. EndIf
5. Rear = Rear + 1
6. Q[Rear] = element
7. If (Front == -1) Then
8.     Front = 0
9. EndIf
10. Return
```

Procedure Dequeue (Q, Rear, Front)

1. If (Front == -1) Then
2. Write ("Queue Underflow")
3. Return
4. End If
5. Write ("Element to be deleted : ", Q[Front])
6. If (Front == Rear) Then
7. Front = Rear = -1
8. Else
9. Front = Front + 1
10. End If
11. Return

C Implementation → Global → array, Rear, Front

- One f^n for Enqueue
- One f^n for Dequeue
- One f^n for Display

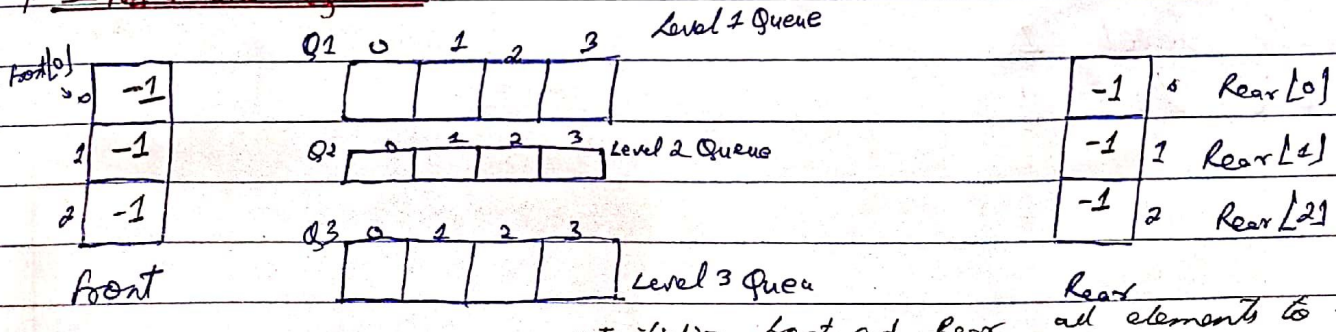
- Declare the array, rear, front locally within main
- Pass the array, rear, front as arguments

—×—

Date 06/10/21

Lecture - 10

# Multilevel Queue :-




* Initialize front and Rear all elements to -1.

(18)

Insert 10 to level 1

if (Rear[0] == maxsize - 1)

. - - - - .

Rear[0] = Rear[0] + 1

Q1[Rear[0]] = 10

Q1[0] = 10

Front[0] = front[0] + 1.

Insert 20 to level 1

Rear[0] = Rear[0] + 1

Q1[Rear[0]] = 20

Q1[1] = 20

Insert 30 to level 1

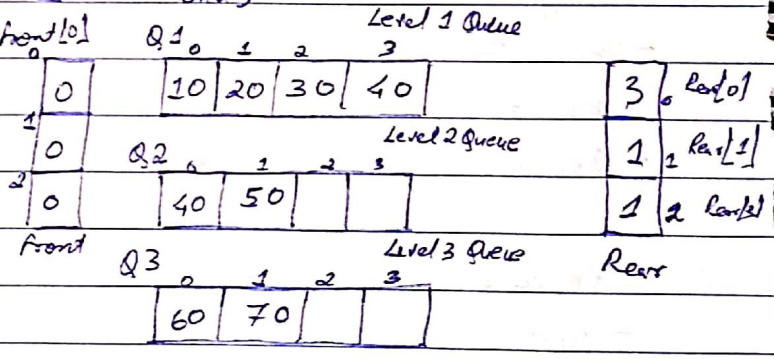
Insert 40 to level 1

Insert 40 to level 2

Insert 50 to level 2

Insert 60 to level 3

Insert 70 to level 3




Write("Deleted Element: ", Q1[Front[0]]);

Front[0] = Front[0] + 1

Delete level 1

Delete level 1

Delete level 1

- This scheme can be used to implement multiple queues
- This scheme can also be used to implement a "priority queue".

## Priority Queue

→ Each element has a priority.
→ Priorities ~~can~~ are expressed using integers.
→ ① Sometimes lower integers represent higher priority.
② Sometimes lower integers represent lower priority.

Element°

| | | |
|---|---|---|
| A | priority | 0 |
| B | priority | 1 |

In a priority queue, the element with the highest priority is deleted first.

Normal FIFO queue.

→ —

Date 16/10/22

# Lecture – 11

## Implementation of priority queue using multilevel queue :-

## # Double ended queue :-

Insertions and deletions can be done at both ends of the queue.


## # Linked Linear List (Linked List) :-

Static memory allocation

↳ memory allocated at compile time.

```
int x;
x = 10;
```

Dynamic memory allocation

↳ memory allocated at run (execution) time.

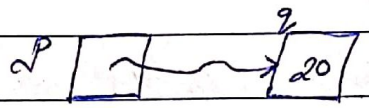
e.g. malloc f^n in C.

e.g.

```
struct node {
        int roll;
        float cgpa;
};
struct node *ptr;
```

e.g. malloc function in C

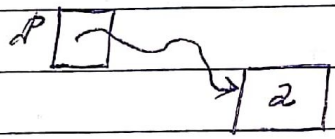
```
int *p;
int q;
p = &q;
*p = 20;
```




q refers to the contents of q by *p

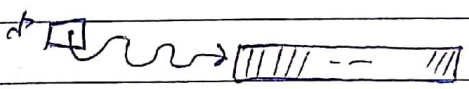
```
int *p;
p = (int *) malloc (sizeof (int));
```

```
typecasting
char *
scanf ("%d", p);
*p = *p + 1;
```




portability

```
int *p;
int n;
printf ("How many?");
scanf ("%d", &n);
p = (int *) malloc (sizeof (int) * n);
p[0] = 50;
```



# Linked List:-

It is a liked representation of a linear list.

e.g. (10 20 30)

We have node's that are dynamically allocated.

A node is, a block of memory. Each such block has an address.