UNIT-2

STACKS AND QUEUES

Stacks:- Stack is a linear data structure in which all the operations such as insertion, deletion takes place at only one end called "TOP". Thus it is called Restrictive Data Structure.

* In this stack the insertion operation is called PUSH operation & Deletion operation is called POP operation.
* This follow a principle FILO[First in Last out] (or) LIFO[Last in First out].
* Example:- The common examples of stack are
  1. Placing the books in the shelf.
  2. Bread Slices.
  3. Moving out of the bench.
  4.

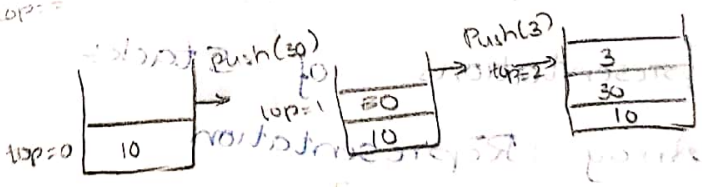
Basic operations of Stack are :-

1. PUSH operation
2. POP operation
3. Stack top operation (or) Display operation.

PUSH Operation:-

Code :-

```
if (top == max-1)
{printf("overflow");
}
else
 {
  top++;
  st[top]=x;
 }
```

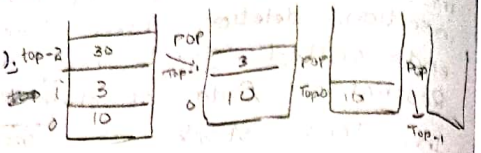


* At a time only one element can be inserted (or) deleted from the stack.

POP Operation:-

code:-

```
if (top == -1)
{
printf("underflow");
}
else
{
 temp = st(top)
 top--;
}
```



* Stack TOP / Display Operation:-

code:-

```
if (top == -1)
{
printf("stack is empty");
}
else
{
for(i=top; i>=0; i--)
printf("%d", st[top]);
}
```

```
if (top == -1)
{
printf("stack is empty");
}
else
{
for(i=top; i>0; i--)
printf("%d", st[top]);
}
```

Stack Top is used to display the data from Top to bottom.

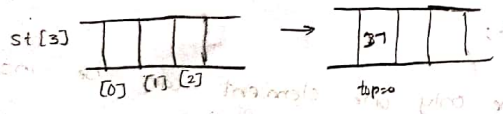
Representations of Stack:-

1. Array Representation

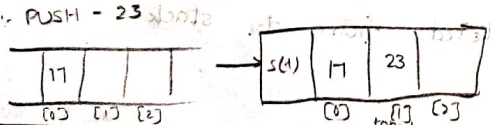
set [17, 23, 9, 5]

maximum size is 3

step-1:- PUSH - 17

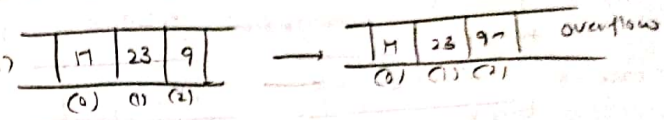


step-2:- PUSH - 23



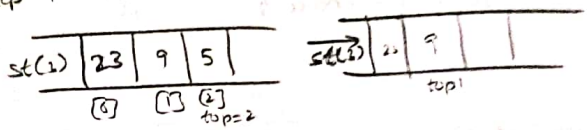
step-3 PUSH - 9

step-4 PUSH -5

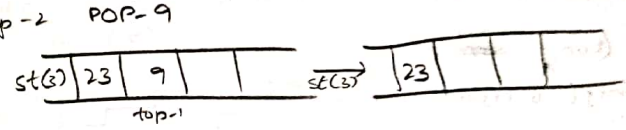


For Deletion

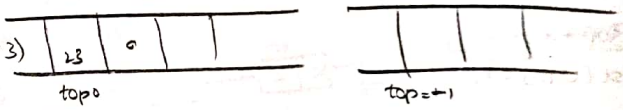
Step-1 POP-5



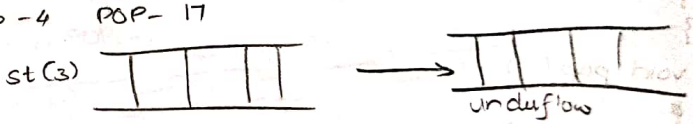
Step-2 POP-9



Step-3 POP-23



Step-4 POP-17



Stacks using Arrays:-

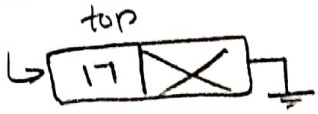
```
# define size 3
void push(int);
void pop( );
void display( );
int st[size], top=-1;
void main( );
{
 int x, ch;
printf("1. Push \n2. pop\n 3. display\n 4. Exit");
printf("enter your choice");
scanf("%d", &ch);
switch (ch);
```

```
case 1 : printf("enter element");
        scanf("%d", &x);
        push(x);
        break;
case -2 : pop();
        break;
case -3 :- display();
        break;
case 4 : exit(0);
default : printf("wrong choice");
  }
  }
  }
void push(int x)
{
 if (top == max-1);
printf("overflow");
else
{
  top++;
  st[top] = x;
}
}
void pop( )
{
 if (top = = -1)
{
printf("underflow");
}
else
{
temp = st(top)
top - - ;
 }
 void  display( )
 if (top = = -1)
{
printf("stack is empty");
}
```

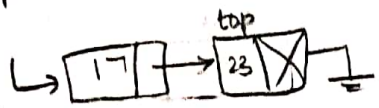
```
else
{
for(i=top; i>=0; i--)
printf("%d ", st[top]);
}
```

Linked list Representation :-

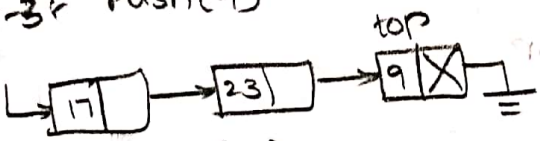
step-1 :- Push(17)



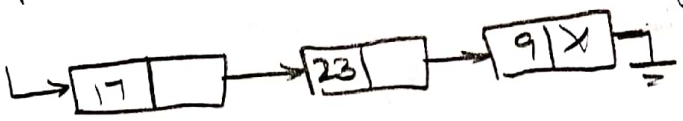
step-2 :- Push(23)



step-3 :- Push(9)

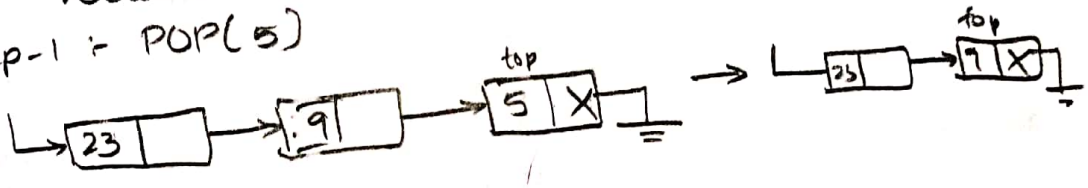


step-4 :- Push(5)

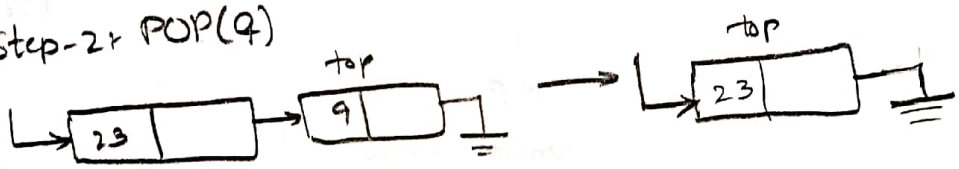


For Deletion

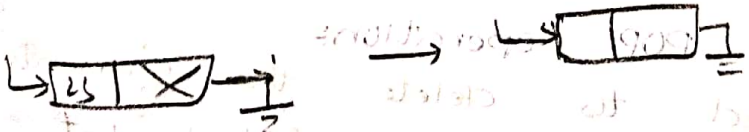
step-1 :- POP(5)



Step-2 :- POP(9)



Step -3 :- POP(23)



Step -4 POP(17)

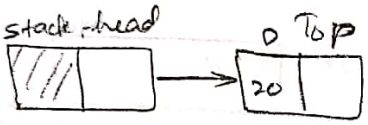
→ underflow

Algorithm for PUSH operation using linked list :-

This can be represented using single linked list where insertion and deletion process takes place at only at end of the linked list.

Code :-

```
new = getnode(node)
new → data = item
new → link = TOP
TOP = new
stack - Head → Link = TOP
stop
```




```
If (top == NULL)// This indicates to check whether the top value of the linked list NULL or Not if NULL then create a new node and attach it to the header.
{
top = (struct node *) malloc (size of (struct node));
top → ptr (link) = NULL;
top → info = data;
}
else
{
temp = (struct node*) malloc (size of (struct node));
temp → ptr = top;
temp → info = da
count ++;
}
```

Algorithm for POP operation :-

This is used to delete the elements from the end of the linked list

code:-

```
pop( )
{
if (top==-1)
{
printf("empty");
}
else
{
temp= top // here make the pointer of the top to be assigned
top = top→ptr;
temp →link = Null;
return node (temp);
}
```

```
temp=top
top=top→ptr;
temp→link=Null;
return node (temp);
}
```

Display Operation:-

This is used to print all the values present in the linked stack

code:-

```
display( )
{
if (top==-1);
{
printf("empty");
return();
}
else
{
temp=top;
while (temp!=Null);
{
print("temp→data");
temp=temp→link;
}
}
```

```
display( )
{
if(top==-1);
{
printf("empty");
return( );
}
else
{
temp=top;
while(temp!=NULL);
{
print("temp→data");
temp=temp→link
}
}
```

Applications of Stacks:-

1. Converting the infix expression into the postfix expression.

Steps:-

(1) Scan the infix string from left to right one character at a time.

2. * While the characters are left then switch on to the particular character.

(i) If it is a letter then place it in the postfix string.

(ii) If it is an operator check precedance of top of the string in the stack whether it is higher than the scanned character. Then pop the top of the stack and add to postfix.

(iii) If it is the Left perantrasis then push into the stack.

(iv) If it is a right peranthrasis then pop out from stack till the left peranthesis is encountered.

3. % pop out the stack contents untill the stack is empty and add to the postfix expression.

~~PRECEDANCE AND~~ ASSOCIATIVITY

PRECEDENCE AND ASSOCIATIVITY

Precedance	Operator	Associativity
1.	++ -- ( ) [ ] . (period operator) →	Left to Right
2.	! ~ * (pointers) & address SIZE of	Right to left
3.	Multiplication *, Division / quotient, Modulus % reminder	Left to Right
4.	+, -	Left to Right
5.	<<, >>	Left to Right
6.	< > <= >=	Left to Right
7.	==, !=	Left to Right
8.	& [Bitwise AND]	Left to Right
9.	^ [Bitwise Exclusive OR]	left to Right
10.	| [Bitwise Inclusive OR]	left to Right
11.	&& [Logical AND]	left to Right
12.	|| [Logical OR]	left to Right
13.	?: [Ternary operator]	right to left
14.	= += -= *= /= %= &= ^= != <<= >>=	right to left
15.	comma operator	left to right.

Example:

a+b*c-d

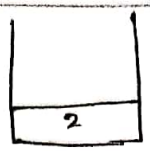
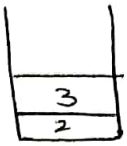
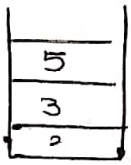
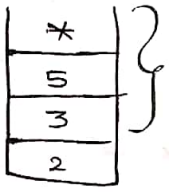
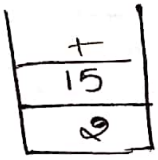
Character scanned	stack content	post fix expression	Explanation
a		a	
+	+	a	
b	+	a b	
*	+ *	a b	+ > * (F)
c	+ *	a b c	
-	+ * -	a b c *	* > - (T) so shift * post-fix
	+ -	a b c * +	// Here there are same precedance operator + & - so shift the + to post fix. + > - (both same) shift +
d	-	abc * + d	
end	empty	abc * + d -	

A+(B*C)

Scanned Character	Stack Content	Postfix expression	Explanation
A		A	
+	+	A	
(	+ (	A	
B	+ (	A B	
*	+ (*	A B	
C	+ (*	A B C	
)	+ (*)	A B C	
	+	A B C *	

end empty ABC * +

Application-2:
Evaluation of postfix expression.

Character Scanned	Stack content	Explanation
2		Push 2 in to stack.
3		Push 3 in to stack.
5		Push 5 in to the stack.
*		5, 3 is pop out and perform 3*5 we get 15. Push result 15 into stack
		
+		15 & 2 is poped out and perform 15+2 we get 17. Put 17 in to stack.
		17 is the output

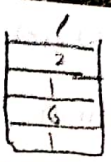
~~A+(B*C)~~

A+[B*C-(D/E-F)*G)*H

Character scanned	Stack content	~~Explanation~~ Post fix Expression
A		A
+	+	A
(	+ (	A
B	+ (	A B
*	+ (*	A B
C	+ (*	A B C
-	+ (* -	A B C

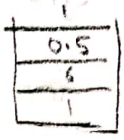
(	+ (-	ABC +
D	+ (- (	ABC *
/	+ (- (	ABC * D
E	+ (- (/	ABC * DE
-	+ (- (/ -	ABC * DE
F	+ (- (/ -	ABC * DE /
)	+ (- (-	ABC * DE / F
*	+ (- *	ABC * DE / F -
G	+ (- *	ABC * DE / F - G
)	+ - *	ABC * DE / F - G
*	+ - *	ABC * DE / F - G
	+ - * *	ABC * DE / F - G *
	+ *	ABC * DE / F - G * +
H	+ *	ABC * DE / F - G * + H
end	empty	ABC * DE / F - G * + H *

Character scanned	Stack content	Explanation
1	1	Push 1 into the stack
2	2 1	Push 2 into the stack
3	3 2 1	Push 3 into the stack
*	* 3 2 1	3 * 2 is poped out & perform 2 * 3 * 100 gets 6 Push 6 in to the stack
	6 1	
1	1 6 1	Push 1 to the stack
2	2 1 6 1	Push 2 into the stack

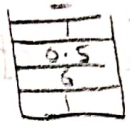


2 / 1 is poped out & perform 1/2 we get 0.5. Push 0.5 to stack.

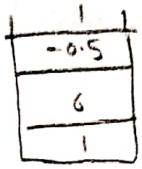
Put into the stack.



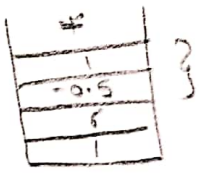
0.5 & 1 is poped out and perform, 0.5 - 1 we get -0.5 and push it in to the stack

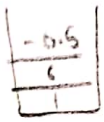


Push 1 into the stack



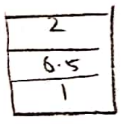
1 & -0.5 is poped out and perform 1 x -0.5 and push in to the stack.



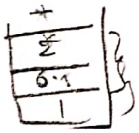


6 & -0.5 is poped out and perform 6 - (-0.5) we get 6.5

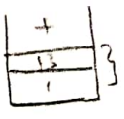
Push 2 into the stack



2 & 6.5 is poped out & perform 2 * 6.5 we get 13. Push it in to the stack.



13 & 1 is poped out & perform 13 + 1 we get 14. Push it in the stack

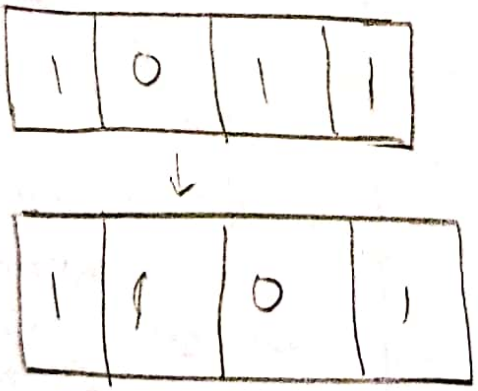


14 is the result

Other applications of stacks :-

* It is used for viewing the page history visited in any browser.
* It is used in reversing the array elements of the given order.
* It is also used in number conversion. This can be converted into binary number by the use of stack data structure and can display the obtained output.

```
while (num ≤ 0)
{
digit = num % 2;
push(digit)
num = num/2;
}
```



PART-2

QUEUES

* Queue is also a linear data structure where the ~~inter~~ insertion and deletion takes place at two different ends.

They are 1. Rear End — FIFO
2. Front End — LIFO

* Insertion Operation of Queue:-

Algorithm:-

```
if (rear == max-1)
{
printf("full");  / printf("queue is full");
}
else
{rear++;  / rear = rear+1;
qu[rear] = x;
}
```

* Deletion Operation:-

Algorithm :-

```
if(front == -1)
{
printf("queue is empty");
}
else
{
front++;  / front = front+1;
qu[front]=x;  (or)  x = qu[front];
}
```

* Display Operation:-

Algorithm:-

```
if(front == rear);
{
printf("queue is empty");
}
else
{
for(i=front ; i<=rear ; i++);
printf("%d", qu[front]);
}
```

```
if (front == rear)
{
printf("queue is empty");
}
else
{
for(i=front ; i<=rear ; i++);
printf("%d", qu[front]);
}
```

Types of queues:

Circular Queue:

* It can also called as Ring Buffer.
* Here both the insertion and deletion takes place in a circular fashion.

i.e in a clockwise direction.

Algorithm

Note: The insertion process in the queue is called "enqueue".

The deletion process in the queue is called "dequeue"

Algorithm for Insertion:

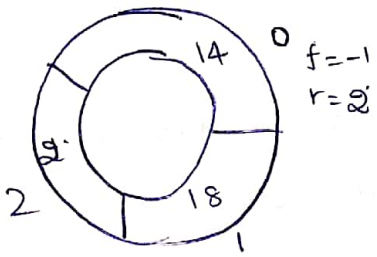
```
void enque(int x)
{
if(front==0 && rear=size-1)
printf("Queue is full");
}else
{
if (rear=size-1)
rear=0;
else
{
rear=rear+1;
cq[rear]=x;
if(front==-1);
front=0;
}
}
```

Algorithm for Deletion:

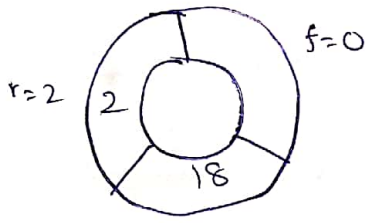
```
void dequeue(int x)
{
if (front==-1);
printf("queue is empty");
}else
```

```
printf("%d", cq[front]);
if(front==rear);
{
front=rear=-1
}
else
{
if(front==size-1);
front=0;
}
else
front=front+1;
}
}
```

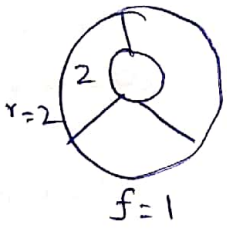
Ex:- To insert 3-element 14, 18, 2, 3,



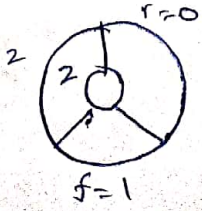
→ Now delete 14



→ Now delete 18



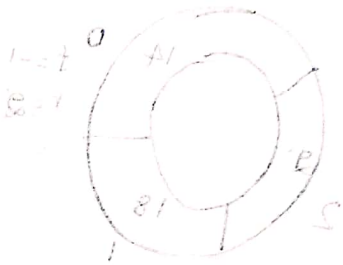
→ Now enter 3



Display :-

Algorithm for Display operation :-

```
void display( )
{
if(front==-1)
{
printf("Queue is empty");
}
if(front<=rear); if (front<=rear);
while(front<=r); while (front<=r);
{
printf("cq[front]");
front++;
}
else
{
while (front<=max-1)
{
printf("cq[front)");
front++;
}
front=0;
while(front<=rear);
{
printf("cq[front]");
front++;
}
}
}
```

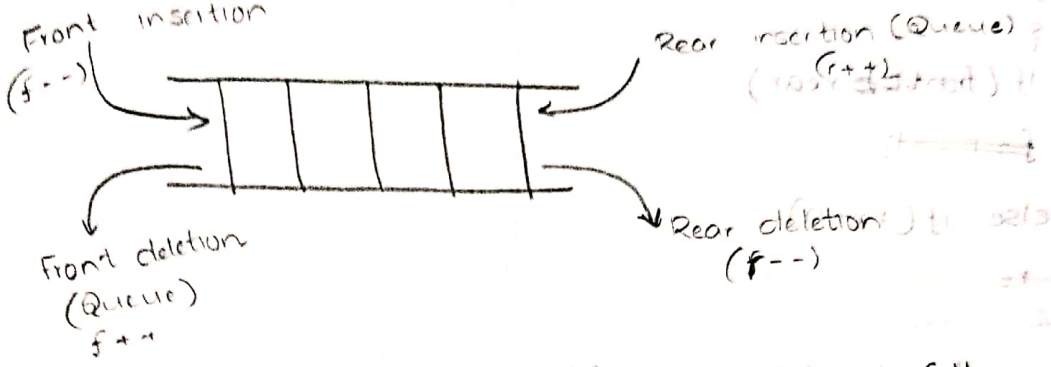


Double Ended Queue :-

Double Ended Queue is a linear data structure where insertions and deletions are done at the ends but not in the middle.

There are 4 types of operations such as

1. Front insertion (f--)
2. Front deletion (f++)
3. Rear insertion (r++)
4. Rear deletion (r--)

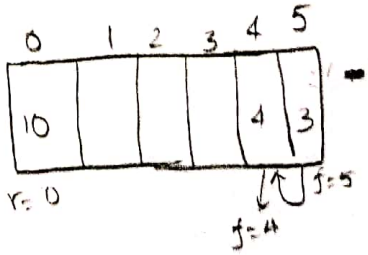


* Here rear insertion & front deletion follows are as same as in the linear queue

Front Insertion :-

```
void dq insert - front (int x)
{
if (front == -1 && rear = size -1)
{
printf (" dq is full");
}
else
{
if( front = = -1)
front = rear = 0;
else
if (front = = 0)
front = size-1;
else
front = front -1;
dq [front] = x;
}
}
```

insertion



Deletion at Rear:

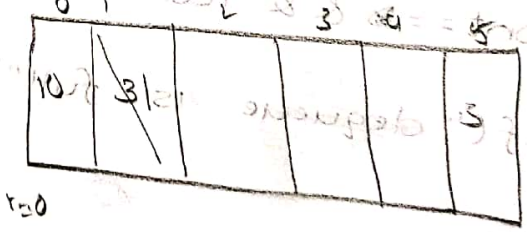
While deleting at the rear decrement the rare value by 1.

Algorithm:

```
Void dq delete - rear()
{
 if (front == -1)
{
 printf(" Queue is empty");
}
else
{
if (front == rear)
front = -1;
else if (r == 0).
r=
```

Algorithm:

```
Void dq delete - rear()
{
 if (front == -1)
{
 printf(" Queue is empty");
}
else
{
 if (front == rear)
 front = rear = -1;
 else if (rear == 0)
 rear = size - 1;
else
  rear = rear - 1;
}
}
```



Front deletion :-

```
void dq front delete(int x)
{
if (front==-1) && (rear_size+1);
{printf("double queue is empty");
}
}
else
{
 front++;
 dq[front] = x;
}
```

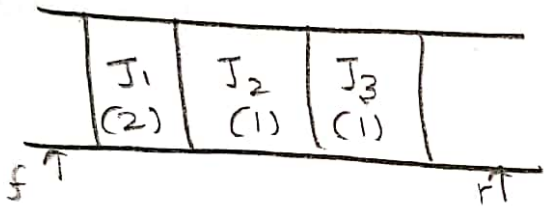
Rear insertion -

```
void dq rear insert()
{
if (rear=max-1)
{
 printf("dq is full");
}
else
{
rear++;
dq[rear] = x;
}
}
```

Priority Queue:- It is a queue where insertions and deletions are done from any the position based on the property priority.

* In a priority queue element with high priority will be deleted first. Such th we can store highest priority element at the front.

Example:- Executing the jobs.



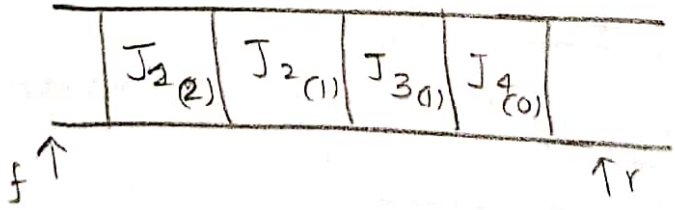
* The job J_1 is given with highest priorit

To insert J_4 (0)

Priority (2) = $J_1(2)$

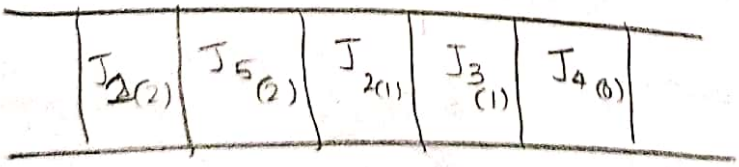
Priority (1) = $J_2(1)$ $J_3(1)$

Priority (0) = $J_4(0)$



To insert J_5 (2)

By following Linear queue pinaple FIFO / LILO



Applications of Queues:

(1) CPU Job scheduling :- FCFS CPU scheduling system.

(OR)

Time sharing System

(2) Round Robing CPU scheduling Algorithm.

(3) Priority CPU scheduling Algorithm.

(1) CPU Job Scheduling :- [Linear Queue]:

Users	Jobs
A	$J_1(4)$, $J_2(3)$
B	$J_3(2)$, $J_4(1)$, $J_5(1)$
C	$J_6(6)$

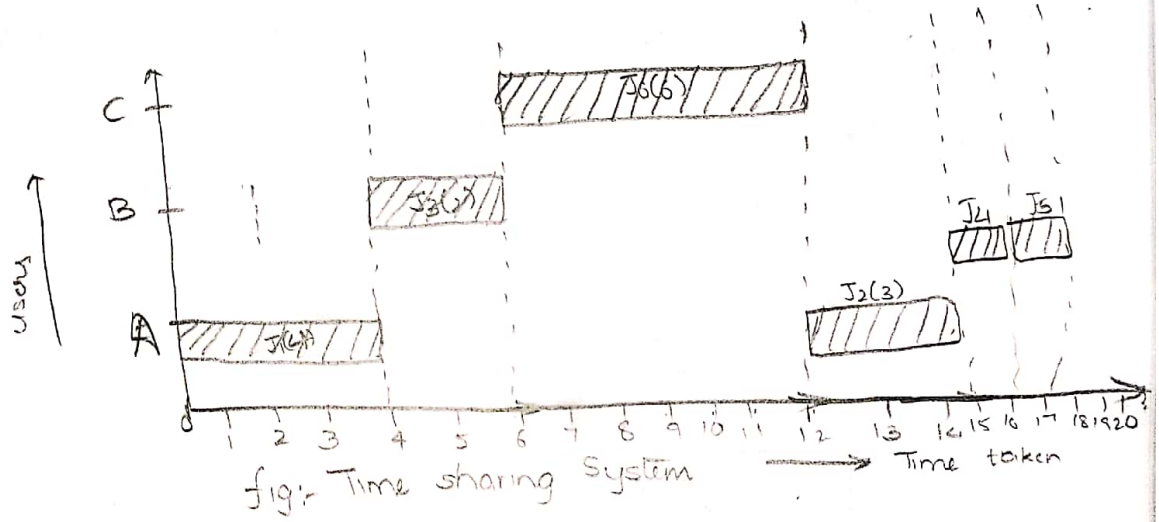


fig:- Time sharing System

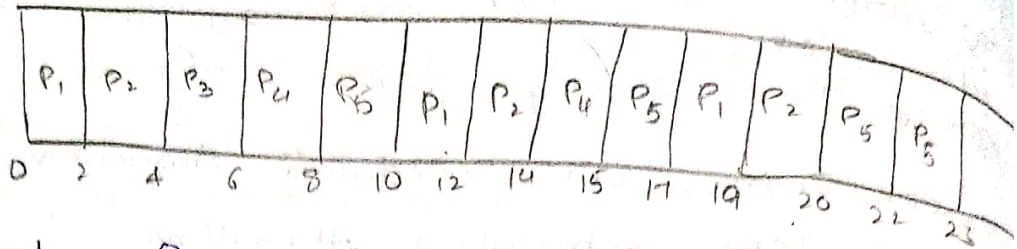
(2) Round Robing CPU scheduling algorithm :-

In this round robing algorithm the process has been scheduled based on their time quantum alloted.

Process	Time Taken → burst time
P_1	
P_2	
P_3	
P_4	
P_5	

Time quantum i.e the time which is limited

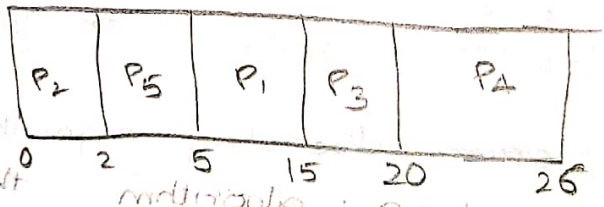
Time quantum = 2



Priority Queue :-

Priority CPU scheduling algorithm:

Process	Time taken/ burst time	Priority
P_1	10	3
P_2	2	1
P_3	5	4
P_4	6	5
P_5	3	2



Applications of Queues :-

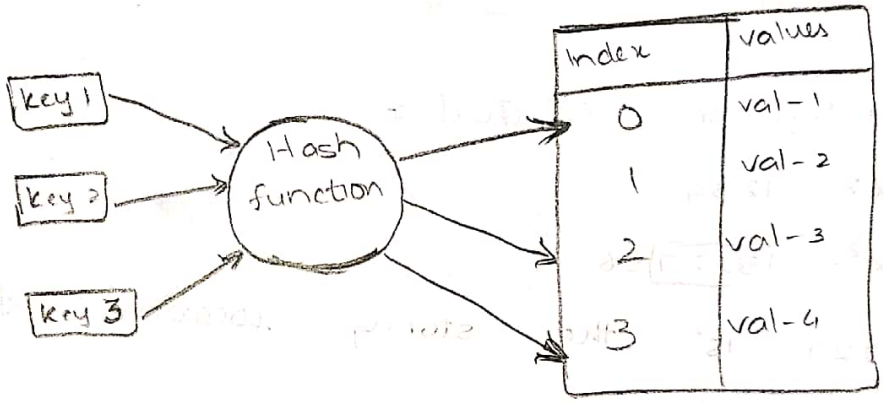
* Statements executed in ~~Queues~~. microprocessor
* OS task scheduling.
* Print lines of a document
* Printer sharing between computers
* Recognizing Palindromes
* Queues used in Simulation applications.

TABLES:-

Hash Table:-

Hash table is a kind of data structure used to store the data in the form of arrays using its own unique indexing value.

* This can be obtained by Hash function which is used for calculating the addressing values (or) hash codes



The hash index can be calculated as

(key % size) H(key) = key % size.

H(key) = key %. size

Methods used in Hash tables:-

* To generate the addresses or to store the key value at any position in the hash tables we can have different types of methods

1. Division Method
2. Mid Square Method
3. Folding Method
4. Digital analysis Method

Division Method :

This method is followed by using the formula

h(key) = key % size

Ex:- 20, 25, 33, 46, 50 size 10

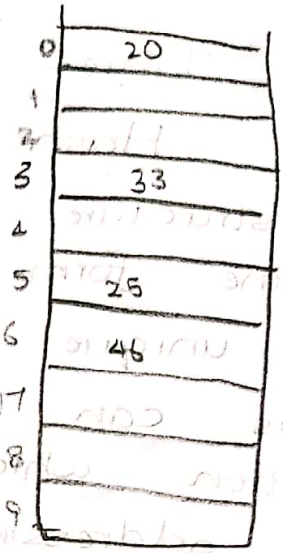
h(20) = 20 % 10 = 0

h(25) = 25 % 10 = 5

h(33) = 33 % 10 = 3

h(46) = 46 % 10 = 6

h(50) = 50 % 10 = 0



Here collision occurs, So we follows

1. Closed hashing (or)
2. Open hashing methods

to overcome this collision.

Mid Square Method :

Ex:- 1234

k^2 = 15227 56

227 is the storing location of 1234

Folding Method :

In this folding Method the given key is chopped into 'n' no. of keys and then at the sum of all the keys the final key will be stored.

* Even in this method there are 3 different types of Techniques as

1. Pure Folding Method
2. Fold shifting Method
3. Fold Boundary Method

Pure Folding Method :-

This is used to store at the location where the sum of all the partitioned keys is obtained.

Fold Shifting Method :-

In this method consider odd numbered positions of Partition keys & reverse those positions values and then add those values together.

Fold Boundary Method :-

Consider the first and last partitioned keys and reverse them then add together.

Ex :- K = 15 22 7 56

chopping = 01 52 27 56

= 136th position → Pure folding Method

Fold shifting Method :-

= 01 52 27 56

= 10 52 72 65 56

= 190th position

Fold Boundary Method :-

= 01 52 27 56

= 10 52 27 65

= 154th position

Digital Analysis Method :-

In this method the Hash address is formed by extracting even no. of digit positions and reversing them.

Ex :- 15 22 7 8 6

5 2 8

Reversing

825th position

* Among all these method Division method is used for allocating the key value in the hash tables for major applications

Types of Hashing:-

There are 2 types of Hashing Techniques

(1) Closed Hashing
(2) Open Hashing

These Techniques are used for resolving the collision problems.

Closed Hashing :-

(1) Linear Probing Technique
(2) Quadratic Probing Technique

Linear Probing Technique

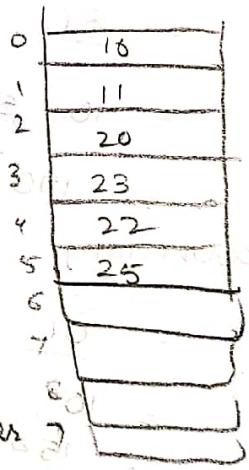
Ex:- 11, 10, 20, 23, 25, 22

size = 10

H(11) = 11 % 10 = 1

H(10) = 10 % 10 = 0

H(20) = 20 % 10 = 0



* '0' is filled & next higher address will be check
* 1 is also filled & next higher address will be check i.e 2
* ∵ 2 is empty. 20 will be stored at position 2

Quadratic Probing Technique:-

In this Technique the key elements are stored by the process of probing

1. Assign J = 0
2. Get the hash values by $(K+j^2)$ mod size

$H(K) = (K+j^2) \% \text{size}$

Ex: Insert the elements 76, 40, 48, 5, 20 with maximum size as 7.

$H(76) = (76 + 0) \% 7$

$= 6$

$H(40) = (40+0) \% 7$

$= 5$

$H(48) = (48+0) \% 7$

$= 6$

$H(48) = (48+1) \% 7$

$= 0$

$H(5) = (5+4) \% 7$

$= 02$

$= (5+3) \% 7$

$= 81$

$H(20) = 20 \% 7$

$= 6 \neq$

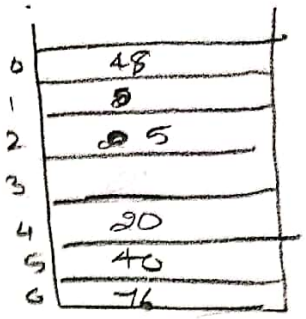
$= 21 \% 7$

$= 0$

$= (20+4) \% 7$

$= 24 \% 7$

$= 3$



Open Hashing :- [Chaining Process]

* Chaining of elements will be takes place by using linked list concept

Ex: 10, 20, 33, 43, 54

$H(K) = K \% 10$