Output

Enter no name and salary 1.

abcd

20000

No 1 Name abcd Salary 20000.000000

★ Inside of ~~alwa~~ always writing struct emp, typedef keyword can be used.

★ typedef struct emp employee;

employee e;

→ used to redefine ~~keywords~~

★ typedef int num;

num eno;

★ emp e[10];

&e[i].eno, &e[i].ename, &e[i].esal

★ Total size of array of structure

= 26 × 10 = 260 bytes

Try to Do: Read & display polynomial ~~using stru~~ using structure & array

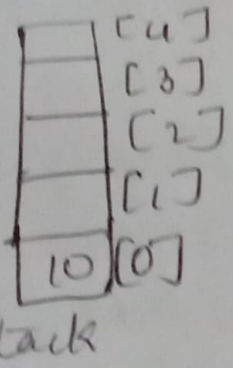
13/9/2021: data structure : Structure used for storing data

Types of data structures

① linear data structure - Eg: Array

② non-linear data structure.

- Using array stack can be implemented
- Stack is a linear data structure
  - → special variable : top
  - → operations
    - push - inserting element
    - pop - removing element
- assumption initially top = -1



- top = -1;
  push (10)
  top+1 = 5 ? No
  so top = top+1
  i.e top = 0 ; stack [top] = 10

} Push

- algo for push

```
push (int e)
{ if (top+1 == s)        → size of stack
    announce ("stack is full");
```

```
    else
     {  top=top+1;
        stack[top]=e;
     }
  }
```

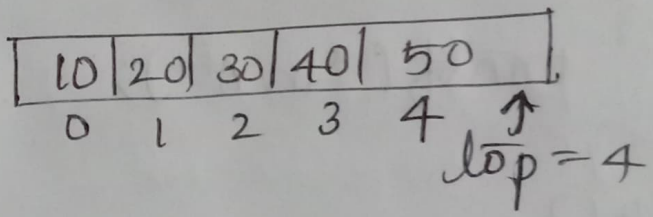
- algo for pop

```
  pop()
  { if (top==-1)
    announce("stack is empty");
    else
      { display (stack[top]);
        top=top-1;
      }
  }
```

Stack Program Using Global Variables

```
#define SIZE 5
int stack[SIZE], top=-1;
  void push (int e) {
    if(top+1==SIZE)
      { printf("Error: stack is full");
      }
    else
      {
        top=top+1;
    } } stack[top]=e;
```

```
void pop()
  if(top == -1)
    { printf("Error: stack is empty");
    }
  else
    {
      printf("%d", stack[top]);
      top = top - 1;
    }
}
int main()
{ push(10);
  push(20);
  push(30);
  push(40);
  push(50);
  push(60); // Error: stack is full
  pop(); //50
  pop(); //40
  pop(); //30
  pop(); //20
  pop(); //10
  pop(); // Error: stack is empty
  return 0;
}
```



× LIFO → Last In First Out data structure

× Stack is LIFO & as well as FILO,

Interview

First In Last Out

Application of Stack

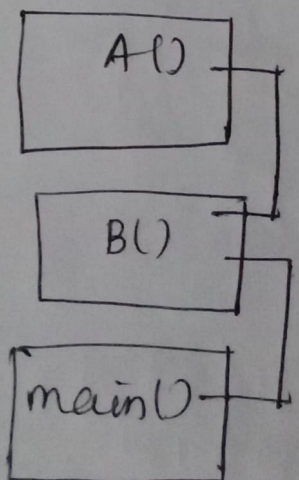
× undo, redo operations

× To achieve function calls

• control stac

```
A()
{
  printf("\nA");
}
B()
{
  A();
}
main()
 { B();
 }
```

/* A will complete /terminate first */



first main executed first, from there B call, and B, A called.
So A terminated first then control goes to B then at last control goes to main()

Q) Reverse a string using stack using local variables

Imp lab Ques

15/9/2021

* Strings

x "abc" → string constant

x string is a collection of characters terminated by null character ('\0').

x Array of strings: char a[10][10];

Q) Read n names and search for a given name.

Q) Read n names and display in alphabetical order

Comparing Strings

→ Return values:

	String 1	String 2
+1	bigger	small
-1	small	big
0	equal	equal

→ strcmpi() ⇒ Ignore case and compare.

Interview

Implement library functions without using library functions

Expressions

- Infix : operator blw operands
  Eg: 2+3
- Evaluation of expression according to BODMAS rule.

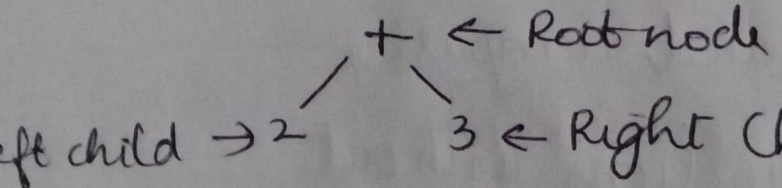
Study pgms in drive (15-09-2021)

- Why conversion of exp. needed from infix to postfix and prefix?

20/9/2021 Different ways of representing expressions

- Infix : operand1 operator operand2 (a+b)
  (Need not always be binary operator)
- Postfix : operand1 operand2 operand (ab+)
- Prefix : operator operand1 operand2 (+ab)
- An expression can be in any of these forms
- Why do we need these representations?
- Most computer use postfix for evaluation.

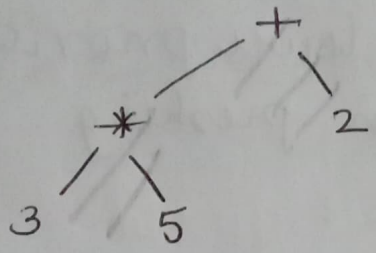
2+3 , tree structure

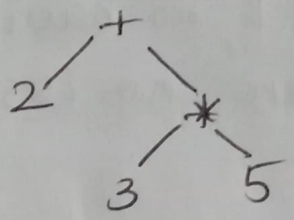


★ Left Root Right : 2+3
↳ INFIX

* Left Right Root : 2 3 +
  ↳ POSTFIX
* Root Left Right : + 2 3
  ↳ PREFIX

- 2 + 3 * 5

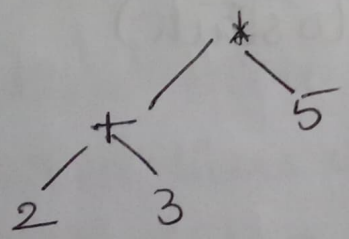




- Every node we have to apply the rule
  → Infix : 2 + 3 * 5
  → Postfix : 2 3 5 * +
  → Prefix : + 2 * 3 5

* Highest priority in bottom level

- (2 + 3) * 5



- Infix: (2 + 3) * 5
- Postfix: 2 3 + 5 *

* Read characters one by one
* Display operands

* An operator can be pushed to stack:
  - If the stack is having an operator which is greater or equal precedence than the operator to be pushed then the operator in the stack should be popped, after that push ~~op~~ operator in expression to stack
  - If the stack is having a lower priority operator then no issue in pushing.

Q) 2 + 3 * 5

↑ 2 - print

2 + 3 * 5

↑

2 + 3 * 5 23 - print

↑

2 + 3 * 5 ~~*~~ (* has higher priority than + so push * to stack)

↑

2 + 3 * 5

↑ 235 - print

Now exp is over, pop operators from stack.

235 * +

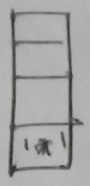
= 2 15 + = 17

Q) (2+3)*5

Q) 2*3+5

↑ print 2

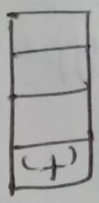
2*3+5
↑



2*3+5
↑ print 23

2*3+5 (cannot push + since * is a higher priority operand is in stack) (so pop *)
↑

so now 23*

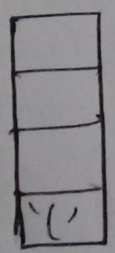


2*3+5
↑ 23*5

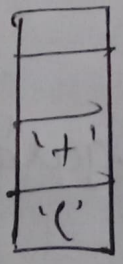
Exp over now pop from stack: 23*5+

Q) (2+3)*5

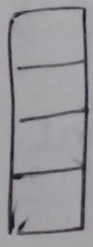
- when there is a left parenthesis push it
- when there is a right parenthesis, pop out till a left parenthesis is encountered.

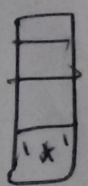
1) 

2) 2

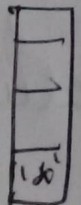
3) 

4) 23

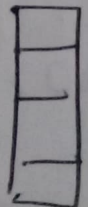
5) 
23+

6) 

23+

7) 

23+5

8) 
23+5*

Q) $\frac{-b+\sqrt{b^2-4ac}}{2a}$ (square root have one operand)

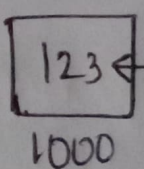
$\overset{\uparrow}{-}2+3$
unary minus

Q) Program to convert infix expression to postfix expression

24/9/2021 integer : datatype ; int - keyword
Array - derived data type

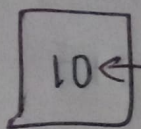
Pointers

x They are used to access memory locations in hardware
x datatype : specify size of memory location
x variable : interface b/w pgm and memory location.
x int : keyword used for representing integer
x a ⇒ variable name.
x int a;



← garbage value

int a = 10;



← value

x a variable which can point to any memory location - pointer variable

Eg: int *p;

* → indicates it is a pointer variable.

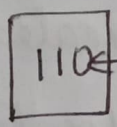
Two ways of using pointer variable

I Pointers pointing to already allocated memory location

p = 1500
&p = 2000
*p (value at which it is pointing to)

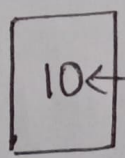
- Eg: int a; ①
  int *p; ②
  a = 10; ③
  p = &a; ④
  printf("%d", *p); ⑤
  *p = 20; ⑥
  printf("%d", a); ⑦

&a ⇒ address of a

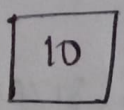
- ① 

a → 1500

② 
p → 2000

③ 

a → 1500

④ 

a → 1500

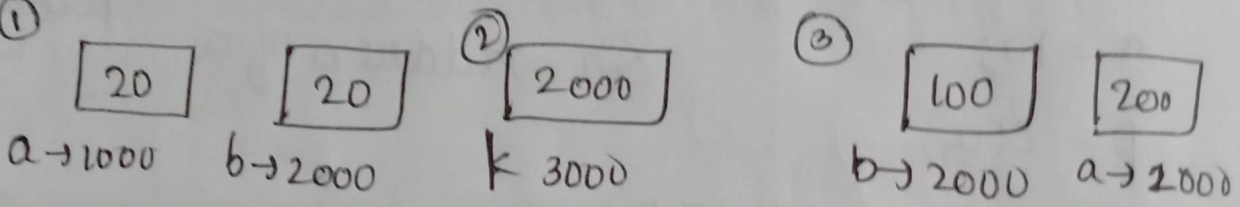
p → 2000

⑤ 10 ⑥ 20

* pointer 'p' is pointing to a memory location which is already allocated to 'a'.

- Eg:
```
int a, b;
int *k;
a = b = 20;
k = &b;
*k = 100;
a = 200;
printf("%d %d %d", a, b, *k);
```

① ② ③



o/p

200 100 100

II pointer which points to a newly created memory location (or memory allocated dynamically)

- malloc()

```
datatype *pointervariable;
pointervariable = (datatype *) malloc (sizeof (datatype));
```

Eg: int *p;

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

malloc(); // dynamic memory allocation function

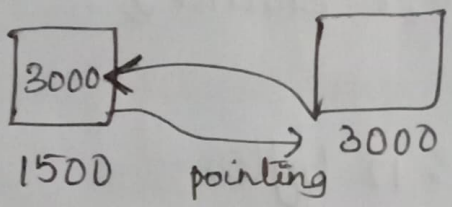
Eg: int *p; ①

p = (int *) malloc (sizeof (int)); ②

*p = 10; ③

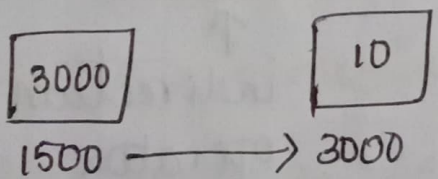
printf ("%d", *p); ④

free (p); // memory allocated by pgmmer should be freed by pgmmer. ⑤

① 

② allocate 2 bytes of memory and return that memory as a pointer to an integer variable

*p ⇒ value at which p is pointing to.

③ 

④ 10

&p = 1500

⑤ free(p); // free (3000)

```
struct emp
{
    int eno;
    char ename[10];
};
typedef struct emp emp;
void main()
{
  emp e;
  e.eno = 10;
  strcpy(e.ename, "abc");
  printf("%d %s", e.eno, e.ename);
}
```

e.eno
↳ member ~~reference~~ operator
⇒ used to access members inside the structure.

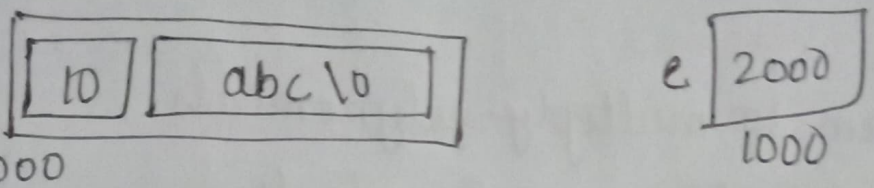
Memory Allocated to e : 12 bytes.

Above pgm in way I

```
struct emp
{
  int eno;
  char ename[10];
};
typedef struct emp emp;
void main()
{
  emp *e, a;
  a.eno = 10;
  strcpy(a.ename, "abc");
  e = &a;
```

e → eno
↑
indirection operator.

```
    printf("%d %s", e->eno, e->ename);
}
```



In Way II

```
struct emp
{
   int eno;
   char ename[10];
};
typedef struct emp emp;
void main()
{
   emp *e;
   e=(emp *)malloc(sizeof(emp));
   e->eno=10;
  strcpy(e->ename,"abc");
  printf("%d %s", e->eno, e->ename);
   free(e);
}
```

Interview

- sizeof() - operator which looks like function

Interview

Name an operator which look likes a fn

Imp

* pointer should start with *

29/9/2021

Lab
- Program to multiply polynomials
- Reverse a string using stack
- Matrix addition, subtraction, multiplication, transpose, determinant.
- Infix to postfix
- Evaluation of infix expression.
- Post increment : first use then increment. (i++)
  [increment operation is postponed]
- ++i ⇒ Pre increment: first increment then use for remaining operation.
- peek → check priority of operator.

Interview

Compiler dependent

b = a++ + ++a || 6 2

higher priority ⇒ pre-increment.

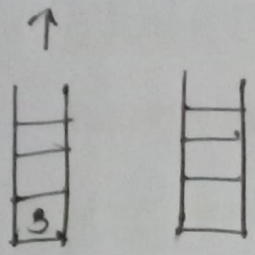
a = 2

b = a++ + ++a

* postfix: also called polish notation.

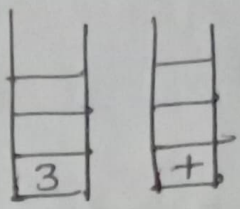
→ two separate stack for operator and operand

1) 3+5*2

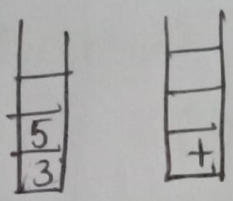


operator operand

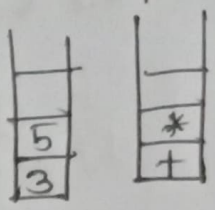
2) 3+5*2



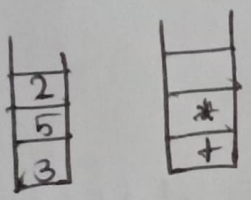
3) 3+5*2



4) 3+5*2

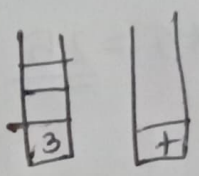


5) 3+5*2

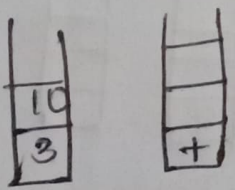


6) Pop * and 2, 5

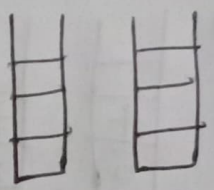
2*5=10



7) Push 10

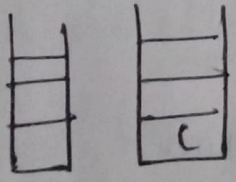


8) Pop +, 3 and 10

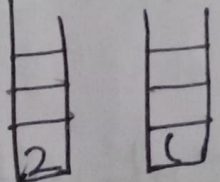


3+10 = 13

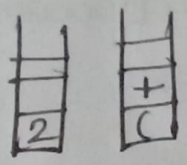
1) (2+3)*5



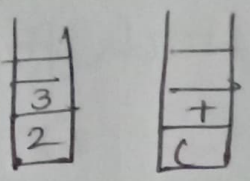
2) (2+3)*5



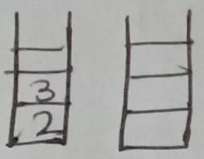
3) (2+3)*5



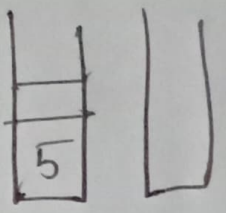
4) (2+3)*5



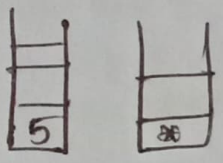
5) (2+3)*5



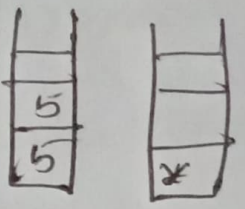
2+3=5



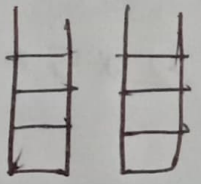
6) (2+3)*5



7) (2+3)*5



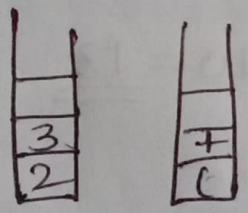
8)



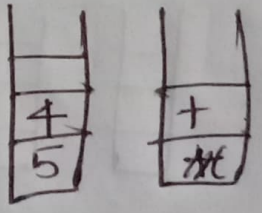
5*5=25

1) (2+3+4)*5 evaluation from left to right

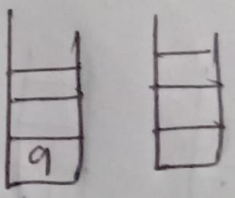
1)



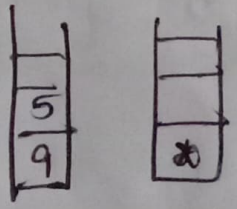
2)



3)



4)



9*5=45

Do Evaluation of infix expression
Draw separate stacks for each in exam

29/9/2024

MODULE 2

- peek operation ⇒ used to check priority
- insertion - last
- removal - 0th location] from array
- after removal shift elements to left
- Pgm to implement array like this.

```
#include<stdio.h>
#define SIZE 10
int a[SIZE];
int count = -1; // array is empty
void insert(int e)
{ if (count+1 == SIZE)
  { printf("Array is full");
  }
  else
  {
    count = count+1;
    a[count] = e;
  }
}
void delete()
{ int i;
  if (count == -1)
  { printf("Array : Empty");
  }
```

```
    else {
        printf("%d", a[0]);
        count = count - 1;
    } for(i=0; i<count; i++) {
}       a[i] = a[i+1];
    }
    count = count - 1;
  }
}

int main()
{
  insert(10); insert(5);
  insert(-3);
  delete(); delete(); delete(); delete();
  insert(200);
  delete();
  return 0;
}
```

o/p

```
10
5
-3
Array : Empty
200
```

- Drawback of this program ⇒
  time consuming : shifting elements
  time complexity involved.

- Queue : linear ds in which FIFO is performed
→ element inserted first is serviced/removed first
→ FIFO → First In First Out
→ Insertion : @ front (also called head)
→ Deletion : @ rear (also called tail)
→ Insertion called enqueue
→ Deletion called dequeue

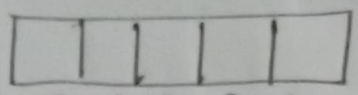
Do Implement Queue using array

Essay on Stack
→ defn (linear ds)
→ LIFO
→ spcl variable ⇒ top, push, pop, peek.
→ define operations.
→ Eg: draw an array
→ Practical Applications
→ pgm - push, pop, peep
→ explanation of few lines in pgm
→ Explain stack and evaluate exp.
→ linear ds.
→ spcl variables ⇒ top, push, pop, peek.
→ Eg:
→ Exp evaluate by drawing stack

1/9/2021

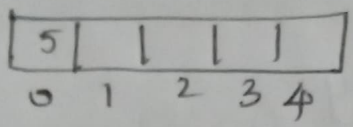
* In case of shifting elements after delete, if N is the size, and if 1 element is deleted: there should be N-1 el...

• Assumption, initially f = -1, r = -1 ⇒ queue is empty

q 

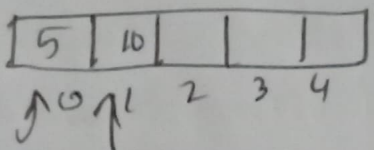
enqueue (5)

f = -1 r = -1



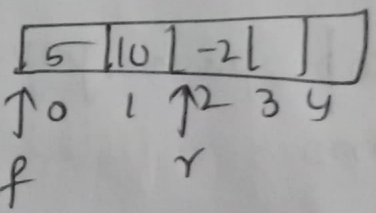
f = 0 r = 0

enqueue (10)

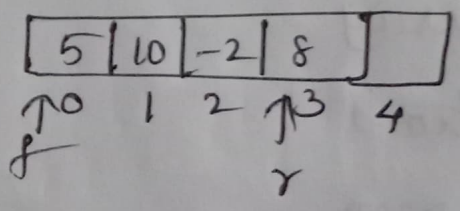


f = 0 r = 1

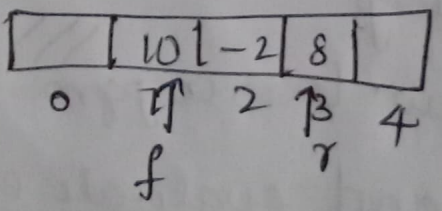
enqueue (-2)



enqueue (8)

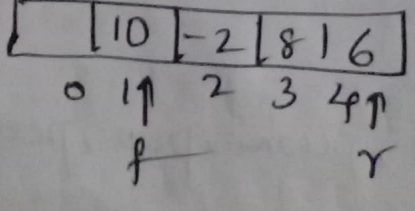


dequeue()

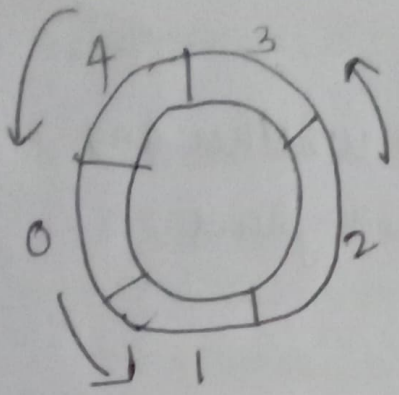


now 2 free locations: 0 and 4

enqueue (6)



queue is not full



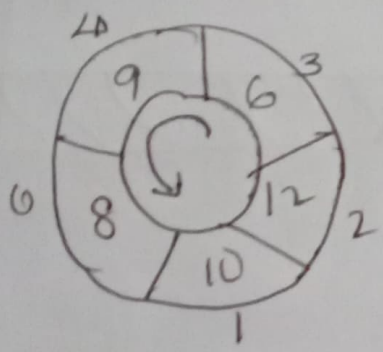
~~circular~~

Modular division

$a = 10\%5 = 0$

if((r+1) % s == f) → size

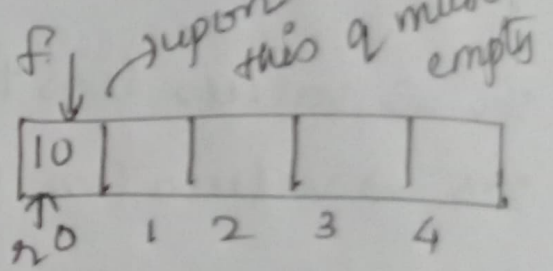
- %6 ⇒ values less than 6
- %5 ⇒ values less than 5

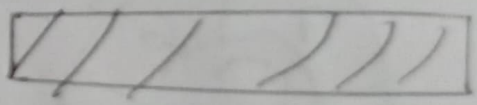


* Queue is a linear data structure in which insertion and deletion of elements occur at ~~two~~ places that is ~~front~~ rear and front

```
#include<stdio.h>
int q[5];   #define SIZE 5
int f=-1, r=-1;
```

```
void enqueue(int e) {
if ((r+1)%SIZE == f) //checking whether any
                       vacant place
 {
  printf("Q is full");
 }
else
 {
   r=(r+1)%SIZE; // moving to next location
   if(f==-1)  // enqueueing first element.
     {
       f=0;
     }
   q[r]=e;
 }
}
```





```
void dequeue()
{
 if(f==-1)
 {
  printf("Q is empty");
 }
else
 { printf("%d", q[f]);
  if(f==r)
  {
    f=r=-1;
  }
  else {
     printf("%d", q[f]);
     f=(f+1)%SIZE;
} }
```