6/8/22

top=2 push(10)
top=1 push(20)
top=0 push(30)

2	30
1	20
0	10

papergrid
Date: / /

pop → removing
push → inserting.

Stacks

→ Linear DS
→ LIFO
↓
Last In First Out list

```
void push (int);
int pop ();
int topele ();
void display ();
int isEmpty ();
int isFull ();
```

if(top==-1) ←

```
#declare MAXSIZE 50
class Stack
{
    int top;
    int arr [MAXSIZE];
public:
```

initial value of top = -1;

push()
increment top by 1
place element in top position.

top hold index of topmost stack element.
↑ Always

when size of stack = 0
top = -1

```
pop()
    if (top == -1)
        stack is empty
    else
    {
        ele = arr [top];
        decrement top by 1;
    }
```

Size of stack = (top+1)
top = SS-1
top = 0-1

SS = top + 1
ss = top + 1
= maxsize - 1 + 1
SS = maxsize

```
int topele()
{
    if (top == -1)
        return -999;
    return (arr[top]);
```

```
#declare MAXSIZE 50
class Stack
{
    int top;
    int arr[MAXSIZE];
    public:
        Stack()
        {
            top = -1;
        }
```

returning nothing : void

```
        void push(int);
        int pop();
        int isEmpty();
        int isFull();
        int topele();
};
void Stack :: push(int ele)
{
    if (isFull())
        cout << "Stack is Full";
    else
    {
        top = top + 1;
        arr[top] = else;
    }
}
```

→ arr[++top] = else

```
int stack::pop()
{
    if (isEmpty())
    {
        cout << "Stack is empty";
        return -999;
    }
    else
    return (a[top--]);
}
```

```
int ele = a[top];
top = top - 1;
return (ele);
```

```
int stack::isEmpty()
{
    if (top==-1)
        return (1); — true
        return (0); — false
    }
```

```
int stack::isFull()
{
    if (top == MAXSIZE-1)
        return (1); ⟶ true
        return (0); ⟶ false
    }
```

topelement

```
int topele()
{
    if (isEmpty())
        return (-999);
        return (a[top]);
    }
```

1. push
2. pop
3. Display
4. topele
5. Exist

```
void display()
{
    int i = top;
    while (i>=0){
    cout << arr[i]; ⟶ i=i-1; }}
```

```
if (isEmpty())
    cout << ("stack is empty");
```

(start from top till bottom of stack)

```
void main ()
{
    Stack s; int ele;
    s.push(10);
    s.push(20);
    s.push(30);
    if (s.isEmpty());
        s.pop();
    if (ele == -99)
            cout << "popped element is " << ele;
~~s.topele()~~
    cout << "Top element :" << s.topele();
            s.display();
            }
```

Date: / /

Stack Application

Base conversion;

① $(20)_{10} = (10100)_2$

2	20	0
2	10	0
2	5	1
2	2	0
	1	

1
0
1
0
0

② $(100)_{10} = (C8)_{16}$

16	200	8
16	12	

12
8

10 - A
11 - B
12 - C

Class stack

```
void base-conv (int number ; int base)
{
    Stack S ; int q , r ;
    while (number >= 0)
    {
        r = number % base ;
        S.push (r);
        number = number / base ;
    }
    S.display ();
}
```

s.push (number % base);

0	× 10
1	× 100
0	× 1000
1	× 10000

10+55 → 65
A A

papergrid
Date: / /

```
Void stack::display()
{
    int i = top;
    if (s.isEmpty())
      while (i>=0)
        {
          if (arr[i] <= 9)
          cout << arr[i];
          else
          cout << (char)(arr[i] + 55);
        }
```

8/9/22

Stack DS:

top == -1 Stack empty

top == maxsize -1 stack Full

Size of stack → ss = top + 1

= -1 + 1 = 0

push arr[++top] = ele (element)

pop return (arr[top--]);

top holds index of topmost stack element.

display:

```
i = top to i>=0
{
  private arr[i];
    i = i - 1;
}
```

if s[13] → then to the top-most (s3) will have more no. of stack = 4 (instead of 3)

Date: / / /

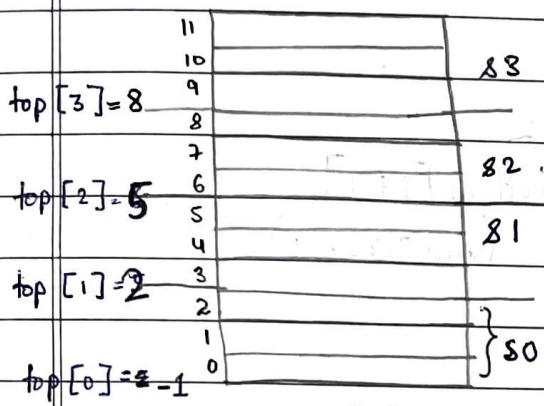
24, 25, 26, 27 ← 28, 29, 30, 31

Multiple Stack:

maxsize = 12

ns = 4 (no. of stack)

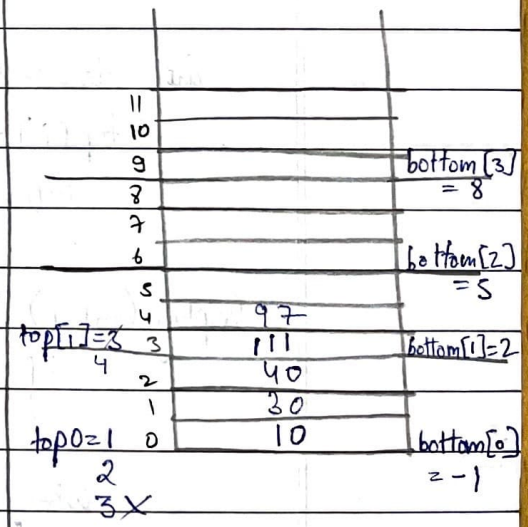
size of each stack = maxsize/ns = 12/4 = 3



top [0] = -1
top [1] = 2
top [2] = 2
top [2] = 5
top [3] = 8

top [i] = (Size of each stack * i) - 1

top [0] = (3 * 0) - 1 = -1
top [1] = (3 * 1) - 1 = 2
top [2] = (3 * 2) - 1 = 5
top [3] = (3 * 3) - 1 = 9



if I want to push (10) in s0, → increase top[0] = -1 by +1

⇓

top [0] = 1 → then place 10

push (0, 10)
push (1, 111)
push (1, 97)
push (0, 30)
push (0, 40)
push (0, 50) — X top of [i] has reached the boundary.

Date: / /

Stack empty :

top[i] == bottom[i];

push :- s[++top[i]] = ele;

maxstack full : (top[i] = bottom[i+1]
top[i] == (maxsize - 1) } for topmost stack.

pop (int i) {

return (s[top[i]--]);

write

int ele = s[top[i]]; index of s:
top[i]--;
return (ele);

Display (int i) {
if (top[i] == bottom[i])
cout << "stack" << i << "is empty";
int index = top[i];
while (index > bottom[i])
{
cout << s[index] << " ";
index--;
}
}

instead use Any

(i) is not inclusive.

```
int index;
for (index = top[i] ; index > bottom [i] ; index --)
        cout << s [index ];
```

PUSH OPERATION:

```
class    mstack
{

    int top[20], bottom [20];
    int ns , maxsize ;

public:
    mstack ()
    {
        ns = 2;
        maxsize = 10; }
    mstack (int n , int ms)
        {
            ns = n;
            maxsize = ms ; }
    void push (int , int);
     int pop (int );
    void display (int);
}
void mstack :: push (int i, int ele)
    {
        if (top[i] == bottom [i+1] :: top[i] == maxsize-1)
        {
            cout <<i<<" i th stack is full ";
            return;
        }
```

Date: / /

```
        top[i]++;
        s[top[i]] = ele;
    }
```

s[top[i]++] = ele;
s[++top[i]] = ele;

POP OPERATION:

```
int mstack :: pop (int i)
{
    int ele;
    if (top[i] == bottom[i])
    {
        cout << i << "th Stack is empty";
        return (-999);
    }
    else {
        ele = s[top[i]];
        top[i]--;
        return (ele);
    } }
```

return (s[top[i]--]);

```
void mstack :: display (int i)
{
    int index = top[i];
    if (top[i] == bottom[i])
        cout << i << "th stack is empty";
    else
    {
        while (index > bottom[i])
        {
            cout << s[index];
            index = index-1;
        }
    } // else
} // display.
```

no. of elements in ith stack

top[i] - bottom[i]

Date: / /

```
mstack()
{
    maxsize = 10;
    ns =
```

```
mstack(int m, int n)
{
    maxsize = m
    ns = n;
    for(i=0; i<ns; i++)
        top[i] = bottom[i] =
                 ((maxsize/ns)*i)-1
}
```

mstack empty :- top[i] == bottom[i]

mstack Full :- top[i] == bottom[i+1]

push :- s[++top[i]] = ele;

pop :- return (s[top[i]--]);

display :- index = top[i] to index > bottom[i] {

display s[index]

index = index - 1;

}

Date: / /

Check Palindrome using stack:

step 1: read string s (char array) from user

step 2: for each char in string s
push (char)

step 3: for each char in stack
(→ 0 to length of s)
if (s[i] == s.pop())
flag = 1
else
{
flag = 0
break;
}

step 4: if flag == 1 and stack is empty and i = length of s
"String" is palindrome"
else
Not palindrome.

eg: a.bcdcda

```
class cstack
{
    char arr[20];
    int top;
public:
    cstack()
    { top = -1; }
    void push(char);
    char pop();
```

```
int main()
{
    char s[20]; stack s1;
    int i = 0, flag = 1; //assume string is palindrome.
    cout << "Enter the String:";
    get(s); ← <stdio.h>
    while (s[i] != '\0')
    {
        s1.push(s[i]);
        i++;
    }
```

Stream
0 1 2 3 4 5

1 pop 1
0 1 2 3 4

```
i = 0;
~~while (s[i] == st.pop() && s[i] != '\0' &&~~

while (s[i] != '\0' && !s1.isEmpty())
  {
      if (s[i] != s1.pop())
        {
          flag = 0;
          break;
        } // if
        i = i+1;
  } // while

if (flag == 0)
      cout << "String is not pali";
        else
        {
            if (s[i] == '\0' && s.isEmpty())
      cout << "String is palindrome
  } // else
} // main
```

Balanced paranthesis checking using stack

Step 1: read string s (char array) from the user

Step 2: for each c in s

```
if (c == '(' || c == '{' || c == '[')   <- for opening
  push c into stack
else if (c == ')' || c == '}' || c == ']')
                                  -> c is closing
{
  a = pop from stack
  if (c == ')' && a != '(')   flag = 0;
```

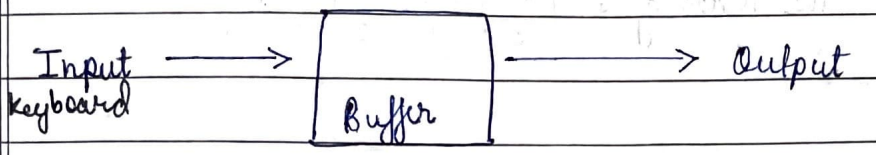
2000 11, 500
13 300

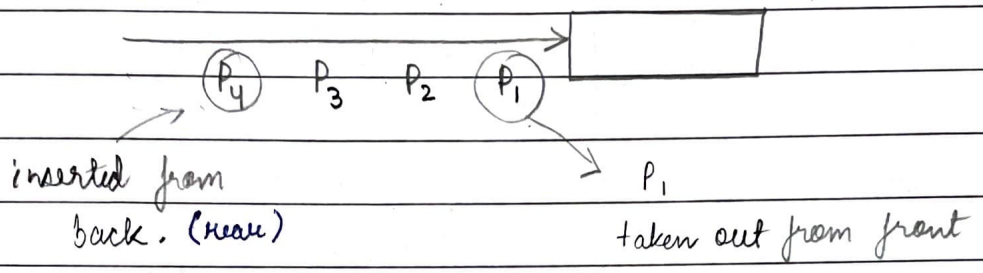
Date:

```
if (c == '}' && a != '{')  flag = 0
if (c == ']' && a != '[')  flag = 0;
    }

if (flag == 1 && s[i] != '\0' && s.isEmpty())
cout << "Balanced";
else
cout << "Imbalanced";
}
```

Linear Q:





variables : maxsize
enQ : rear
deQ : front
display()
isEmpty()
isFull()

```
void enQ (int ele)
{
        if (rear == maxsize - 1)
          cout << "Q is Full";
        else {
// Full
rear = rear + 1;
Q[rear] = ele;
}
```

Note: rear holds index of recently added element.

eg: | 10 | 20 | 30 | 40 | |

```
int deQ ()
{
// Empty
else {
front = front + 1;
return (q[front]);
} }
```

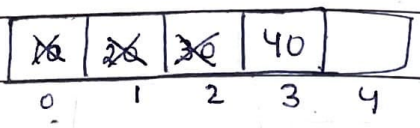
initial front = -1

for deleting 10 from stack

Date: / /

Suppose 10, 20, 30, deleted

∴ front holds the index of recently deleted element.



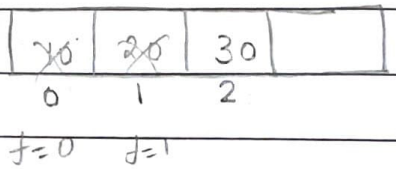
increment f by 1 ∴ f = 0

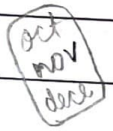
then delete the no. (10 removed)

↓

Again increment by 1

f = 1 (20 got removed)





Empty Condition:

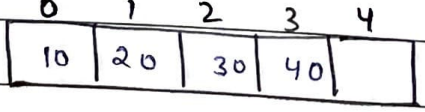
front == rear,

full condition:

rear == (maxsize - 1)

exam

Q.) Analyse display:



Ans:

f = 1

n = 3

display ():

30 40

Q is from f+1 to rear (inclusive)

empty condition :

```
void display ()
{
    if (front == rear)
        cout << "Q is empty";
    for (int i = front + 1; i <= rear; i++)
        cout << q[i];
    } // display()
```

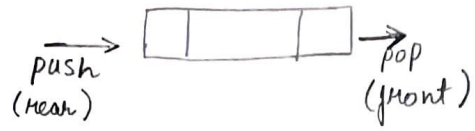
Size of Q is : length of Q = (rear - front)

if (front == rear) (rear - front) = 0

when, both f = r = -1 (it is empty condion)

Complete class code for Q :

```
class Queue
{
    int q[50], maxsize, front, rear;
public :
    Queue () {
        maxsize = 10 ; front = rear = -1;
        }
Queue (int m)
    {
    maxsize = m ; front = rear = -1;
        }
    void endQ (int);
    int deQ ();
  void display ();
};
```



#
```
void Queue :: enQ (int ele)
{
    if (rear == maxsize - 1)
    {
        cout << "Q is full"; return; }
        rear = rear + 1;
        q [rear] = ele;
    }
}
```
} q [++ rear] = ele

int ~~void~~ Queue :: deQ ()
```
    if (rear == front)
    {
        cout << "Q is empty";
        return (-999);
    }
    front = front + 1;
    return (q [front]);
}
```
} return (q [++ front]);

~~void~~

#
```
void Queue :: display ()
{
    ~~int i;~~
    if (front == rear)
        cout << "Q is empty";
    for (i = front + 1; i <= rear; i++)
        cout << q [i] << "  ";
}
```
} if (front = rear)
This will not run

eg: front = 2
rear = 2
} i = 3 to 2 possible ??
↓
No

Date: / /

Stack :

- recursion
- function call

```
main ()
{
    c = sum (a, b);   ---> new context
}
```

context

Circular Q :

→ With Counter
→ Without Counter

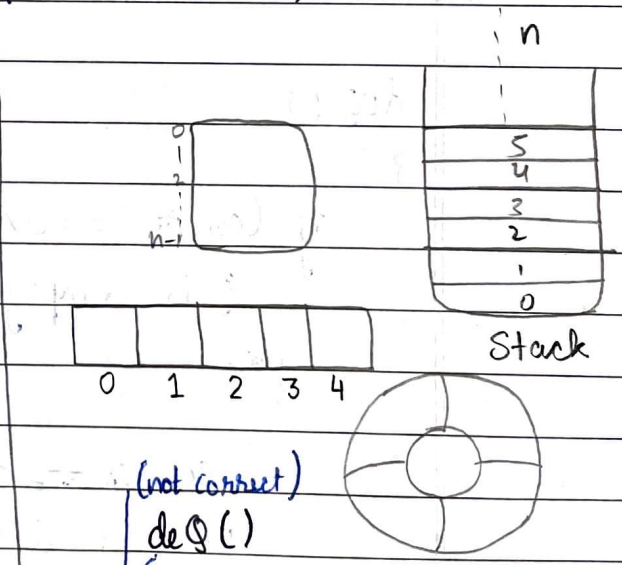
linear → Q

0	1	2	3	4
~~10~~	~~20~~	~~30~~	40	50

f = 2, r = 4

r = -1
f = -1

if (r == maxsize - 1)



With Counter : (represent size of queue)

f = r = -1

counter = 0

(counter → Size of the Q)

CQC : (Not correct)

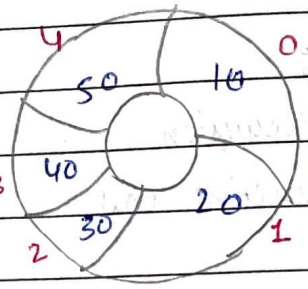
```
counter = 0
Empty :  Counter = 0 ;
Full :   Counter == maxsize ;
Enq :  if (counter == maxsize)   CQC is full
       else {
            q [++ rear] = ele ;
            counter ++ ;
       }
```

(not correct)

```
deQ ()
{
    if (counter == 0)
        Q is empty
    else
    {
        counter -- ;
        return (q [++ front]);
    }
```

rear is used for insertion.
front is used for deletion.

~~Correct~~
~~deQ (){~~
~~}~~



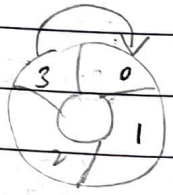
front = 1
rear = 4

CQC (Push) Correct

```
Counter = 0
Empty : Counter == 0 ;
Full : Counter == maxsize ;
if (counter == maxsize)  CQC is full
   else
      {
        rear = (rear + 1) % maxsize ;
          q[rear] = ele ;
            Counter ++ ;
        }
```

```
isFull()
{
  if ((rear+1) % N == front)
     return true ;
  else
     return false ;
```

```
deQ ()        Pop
{
   if (counter == 0)
      Q is empty
   else
   {
      counter -- ;
      front = (front + 1) % maxsize ;
   return (q[rear]) ;
}
```



N=4 (0,1,2,3)
front=3
if ((3+1)% 4 == 0
↑ front

In circular Q it start from (front + 1) to rear (inclusive)

Linear Q :

Size = ~~front - front~~ rear - front

When (front = rear) Size = 0

papergrid

Date: / /

program is for (with counter)

→ Can be used with/without Counter.

```
void display()
{
    int i = (front + 1) % maxsize;
    while (i != rear)
    {
        cout << q[i];
        i = (i+1) % maxsize;
    }
    cout << q[rear];
}
```

Without Counter:

CQWC:

Full: (((rear + 1) % maxsize) == front)

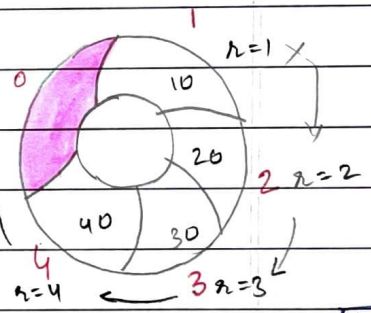
```
enq()
{
    if ((rear + 1) % maxsize == front)
        CQWC is full
    else
    {
        rear = (rear + 1) % maxsize;
        q[rear] = ele;
    }
}
```

(front) 1 position is always empty.

∴ if n = 5

Capacity = 4

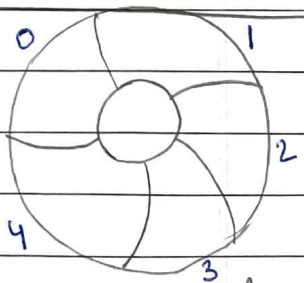
Inserting



Starting from 1. f=0

To mark the end/begining of the Q we leave '0'th place

Deleting:



for deleting the code, front position is kept unused

put x1 = 50
front becomes 2

put x2 = 10
front becomes 3

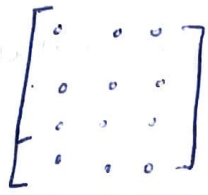
∴ the front position keeps on Changing for deleting.

Date:

CQWC :

```
deQ()
{
    if (front == rear)
        CQWC is empty
    else
    {
        front = (front + 1) % maxsize;
        return (q[front]);
    }
}
```

4 rows
3 column.



Sparse Matrix:

→ many number of 0's count > non zero counts.
→ Array of objects

m = no. of rows
n = no. of column.

eg:

	0	1
1	0	0
2	0	10
3	20	0

3×2

	r	c	v
a[0]	4 (m)	2 (n) same	2 (no. of non zero element)
a[1]	1	1	10
a[2]	3	0	20

0's (6) > 2/1 (non zero)

8 memory location

	0	1
0	0	0
1	0	10
2	0	0
3	20	0

4×2

```
class Sparse
{
    int row, col, val;
    public:
        Sparse ()          { row = col = val = 0; }      // default constructor.
        Sparse (int r, int c, int v)                     // parameterized constructor.
                           { row = r; col = c; val = v; }
        void slowTranspose (sparse *);
        void fastTranspose (sparse *);
        void display (sparce *);                         // -> array of object as 1 parameter.
};

void main ()
{
    int n, i=0, r, c, v;
    Sparse a[10];                                        // -> array of object
    cout << "Enter number of non zero elements";
```

Exam: Read 2D matrix → convert it to array of objects

```
cin >> n;
cout << "Enter no. of rows :";
cin >> r;
cout << "Enter no. of cols :";
cin >> c;
a[0] = sparse (r, c, n);
for (i = 1; i <= n; i++)
{
  cout << "Enter row" << i;
  cin >> r;
  cout << "Enter col" << i;
  cin >> c;
  cout << "Enter val:" << i;
  cin >> v;
  S[i] = sparse (r, c, v);
}
```

14/9/22

Left → Right
top → bottom

papergrid
Date: / /

Sparse Matrix:

0_{00}	2_{01}	0_{02}	0_{03}
1_{10}	0	0	0
0_{20}	0	0	4_{23}
0_{30}	3_{31}	0	0

4×4

number zeros (z) = 12
number of non-zeros (nz) = 4

Array of objects

	row	col	val
a[0]	4	4	4
a[1]	0	1	2
a[2]	1	0	1
a[3]	2	2	4 ✓
a[4]	3	1	3

Same (4) Same

	0	1	2	3
0 →	0	2	0	0
1	1	5	0	0
2	4	3	0	0
3	0	0	0	0

	row	col	val
a[0]	4	4	5
a[1]	0	1	2
a[2]	1	0	1
a[3]	1	2	5
a[4]	2	0	4
a[5]	2	1	3

Row Sorting of Sparse Matrix:

	row	col	val
a[0]	3	3	5
a[1]	0	1	2
a[2]	1	0	1
a[3]	1	2	5
a[4]	2	0	4
a[5]	2	1	3

Col = 0, pass = 0	Col = 0, pass = 1	Col = 2, pass = 2
Check for col=0	Check for col=1	Check for c=2
c_1	c_6 →	c_{11}
c_2 →	c_7	c_{12}
c_3	c_8	c_{13} →
c_4 →	c_9	c_{14}
c_5	c_{10} →	c_{15}
5	5	5

	r	c	v
b[0]	3	3	5
b[1]	0	1	1
b[2]	0	2	4
b[3]	1	0	2
b[4]	1	2	3
b[5]	2	1	5

no. of cols * no. of nz value (objects) = 15
3 x 5 = 15

Date: / /

The loop will run a[0].col x a[0].val times.

```
void SM::SlowTranspose(SM a[])
{
    SM b[10]; int ctr = 1, pass, obj;
    for(pass = 0; pass < a[0].col; pass++)
    {
        for(obj = 1; obj <= a[0].val; obj++)
        {
            if(a[obj].col == pass)
            {
                b[ctr].~~row~~ = a[obj].~~row~~;
                b[ctr].row = a[obj].col;
                b[ctr].col = a[obj].row;
                b[ctr].val = a[obj].val
                ctr = ctr + 1;
            }
        }
    }
    b[0].row = a[0].col;
    b[0].col = a[0].row;
    b[0].val = a[0].val;
}
```

counter

increment the counter otherwise the value will be stored in one place only.

val = value

```
void SM::display(SM a[])
{
    int obj = 0; cout << "\t" << row << "\t" << col << "\t" << val;
    while(obj <= a[0].val)
    {
        cout << "\t" << a[obj].row
             << "\t" << a[obj].col
             << "\t" << a[obj].val;
        obj = obj + 1;
    }
}
```

one line

if object = 5
total object = 5 + 1;

Slow transpose

```
for pass = 0      to  a[0].c  ——— Exclusive
  for obj = 1     to  a[0].v  ——— inclusive
      if (a[obj].col == pass)
      {
          b[ctr].r = a[obj].c;
          b[ctr].c = a[obj].r;
          b[ctr].val = a[obj].val;
          ctr = ctr + 1;
      }
b[0].r = a[0].c;
b[0].c = a[0].r;
b[0].v = a[0].v;
```

[no. of column in input array of object]
[no. of non zero value]

Fast Transpose:

	c_0	c_1	c_2	c_3
r_0 →	0	1	4	0
r_1 →	0	0	3	0
r_2 →	0	5	0	6

3 × 4
r × c

no. of Column

→	0	1	2	3	3 × 4
rowterm	0	2	2	1	
start pos.	1	1	3	5	

		r	c	v
	b[0]	3	4	5
1st	b[1]	1		
2nd	b[2]	1		

		r	c	v
2nd	b[3]	2		
	b[4]	2		
3rd	b[5]			

no. of comparision = no. of non zero value.

	r	c	v
a[0]	3	4	5
a[1]	0	1	1
a[2]	0	2	4
a[3]	1	2	3
a[4]	2	1	5
a[5]	2	3	6

	r	c	v
b[0]	4	3	5
b[1]	1	0	1
b[2]	1	2	5
b[3]	2	0	4
b[4]	2	1	3
b[5]	3	2	6

Example :2 :

0	10	5	0
1	0	6	0
2	4	0	8

no. of non zero term	0	1	2	3 — Column
row term	2	2	2	1
startpos.	1	3	5	7

	r	c	v
b[0]	4	3	7
μ_0 b[1]	0	1	1
b[2]	0	2	2
μ_1 b[3]	1	0	10
b[4]	1	2	4
μ_2 b[5]	2	0	5
b[6]	2	1	6
μ_3 → b[7]	3	2	8

	r	c	v	
a[0]	3	4	7	
a[1]	0	1	10	
a[2]	0	2	5	
a[3]	1	0	1	— b[1]
a[4]	1	2	6	b[6]
a[5]	2	0	2	b[2]
a[6]	2	1	4	b[4]
a[7]	2	3	8	[b[7]

Rest Code is in pdf.

Date: / /

```
void SM::fasttranspose(SM a[])
{
    int rowterm[20], startpos[20];
    int i;
    for(i=0; i<a[0].col; i++)
        rowterm[i] = 0;
    }
    for(i=1; i<=a[0].val; i++)
        rowterm[a[i].col]++;
    startpos[0]=1;
for(i=1; i<a[0].col; i++)
    startpos[i] = startpos[i-1] + rowterm[i-1];

for(i=1; i<a[0].col; i++)
    startpos[i]=st
b[0].col = a[0].row
b[0].row = a[0].col;
b[0].val = a[0].val;
```

→ without this, it will store garbage value.

→ next

1	0	0	0
2	3	0	5
0	4	0	0

3×4

	r	c	v
a[0]	3	4	5
a[1]	0	r[0]++	1
a[2]	1	r[0]++	2
a[3]	1	r[1]++	3
a[4]	1	r[3]++	5
a[5]	2	r[1]++	4

	r	c	v
a[0]	4	3	5
a[1]	0	0	1
a[2]	0	1	2
a[3]	1	1	3
a[4]	1	2	4
a[5]	3	1	5

	0	1	2	3
rowterm	2	2	0	1
startpos	1	3	5	6
	2	4		
	3			

startpos[0] = 1

startpos[i] = startpos[i-1] + rowterm[i-1];

1 to a[0].col;

```
int b_i;
for (i=1 ; i<= a[0].val ; i++)
    {
        b[i] = startpos[a[i].col];
        b[b-i].col = a[i].row;
        b[b-i].row = a[i].col;
        b[b-i].val = a[i].val;
        startpos[a[i].col]++;
    }
cout << "Transpose array of objects :");
        display(b);
    }
```

visit each object in an array.

Write a code which read 2D matrix and stores sparse matrix.

Date: / /

Example : 2

	0	1	2
0	0_{00}	0_{01}	0_{02}
1	1_{10}	0_{11}	5_{12}
2	2	3	0
3	0	4	0

4 x 3

	0	1	2
rowterm	2	2	1
startpos	1 2	~~3~~ 4	~~5~~ 6

	row	col	val
a[0]	4	3	5
a[1]	1	0 = 1	1
a[2]	1	2 = 5	5
a[3]	2	0 = 2	2
a[4]	2	1 = 3	3
a[5]	3	1 = 4	4

Compare with startpos

		row	col	val
	b[0]	3	4	5
r_0	b[1]	0	1	1
	b[2]	0	2	2
r_1	b[3]	1	2	3
	b[4]	1	3	4
r_2	b[5]	2	~~3~~ 1	4

Date: / /

Operator :
1. ^
2. *, /, %
3. +, −

infix	Postfix	prefix
~~a + b~~	op1 op2 opr	opr op1 op2
a + b	a b +	+ a b
a * b	a b *	* a b

op1 opr op2

$a \overset{2}{+} b \overset{1}{*} c$ → $\frac{bc*}{ab+}$ → abc*+ → *bc / +a*bc

$a \overset{3}{+} b \overset{1}{*} c \overset{2}{/} d$ → bc*d/ → abc*d/+ → a b * c / d → +a/*bcd

$a \overset{3}{+} b \overset{1}{/} d \overset{2}{*} c$ → bd/ → abd/c*+ → +a*/bdc

(a + b) * c → ab+c* → *+abc

Algorithm (using stacks):

1. infix → postfix
2. postfix → evaluation
3. postfix → infix
4. infix → prefix
5. pre evaluation
6. prefix to infix
7. prefix ↔ postfix

Read String from user and evaluate : (a+b)*c

20/9/22

papergrid

Date: / /

	Prefix	Postfix
a + b / d * c	bd /	/ bd
	bd / c *	* / bdc
	a bd / c * +	+ a * / bdc
(a+b) * c	ab +	
	ab + c *	
a + b * c	bc *	
	abc * +	

Infix to postfix :

a * (b+c) / d — ~~ab+~~ bc+ — abc+* — abc+*d/

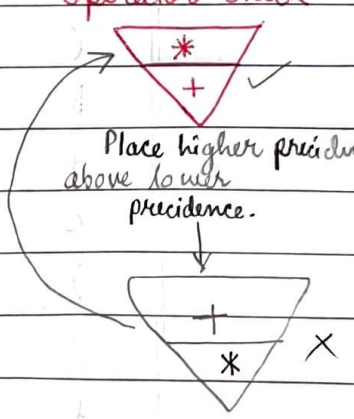
a * ((b+c) / d) — bc+ — bc+d/ — abc+d/*

a

How many push operator are performe?
⇒ no. of operator. = 3
no. of pop = no. of operator.

	token	stack (char)	Output
a+b*c →	a		a
	+	+	
	b		ab
	*	* +	
	c		abc
	'\0'		abc * +
			~~a~~
a*b+c →	a		→ a
ab*	*	*	
ab*c+	b	~~+~~	ab
	+	+ → *	→ ab*
	c		ab*c+
	'\0'		

operator stack



( ) has the highest precidence. (when outside the stack)
least precidence (when inside the stack)

papergri

Date: / /

a * b + c / d

token	Stack (operator)	Output
a		a
*	*	
b		ab
+	+	ab *
c		ab * c
/	/ +	ab * c
d		ab * cd
'\0'		→ ab * cd / +

a * b + c - d → ab * → ab * b + ab * c + d -

a		a
*	*	
b		ab
+	+	ab *
c		ab * c
-	X - + [-]	ab * c +
d		ab * c + d → ab * c + d -

a * (b + c) / d : → bc + abc + * abc + * d /

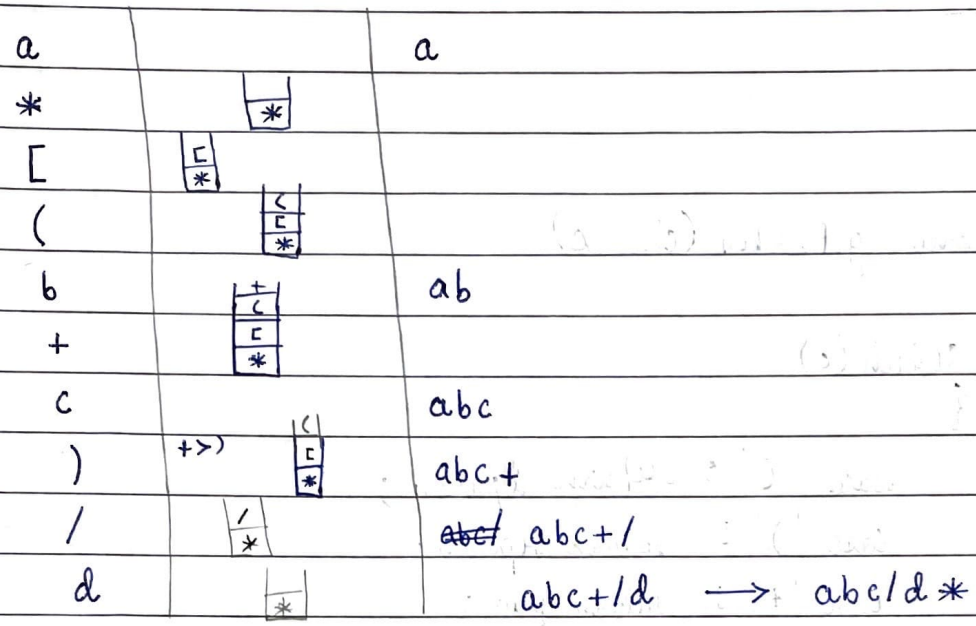
a		a
*	*	
(	(*	
b		a b
+	+ (*	
c		abc
)	*	abc +
/	/	abc + *
d		abc + * d /
'0/'		

[( )] → These have highest precedance outside stack
" " " lowest " inside stack.

papergrid

Date: / /

a * [(b+c) / d] → bc+ bc+d/ abc+d/*



# enum precedance
{
lparen , rparen , plus, minus, times , divide , mod , eos , operand ;
};

named constant (0 - 8)

enum defines the name of data values

icp = incoming precidence

int icp[ ] = { 20 , 19 , 12 , 12 , 13 , 13 , 13 , 0 } ;

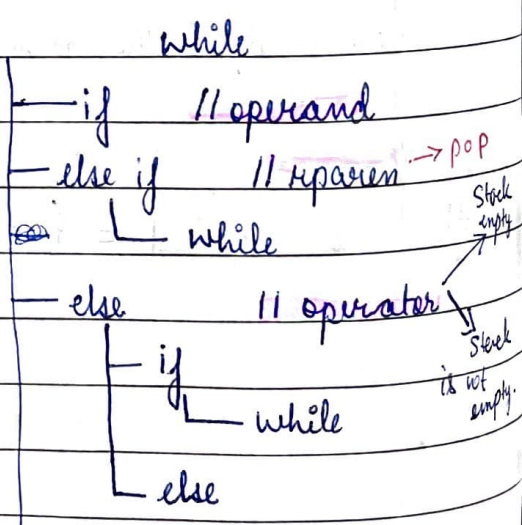
icp[lparen] → 20
icp[mod] → 13
icp[times] → 13

~~int isp[ ] =~~

instack precidence → if has lparen = 0 (lowest)

```
int isp [] = { 0, 19, 12, 12, 13, 13, 13, 0 };
```

```
char
precedence get.token (char c)
{
    switch (c)
    {
        case '(' : return lparen;
        case ')' : return rparen;
        case '+' : return plus;
        case '-' : return minus;
        case '*' : return times;
        case '/' : return divide;
        case '%' : return mod;
        ~~case~~ default : return operand;
    }
}
```



22/9/22

papergrid

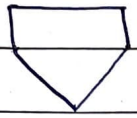
Date: / /

peak / top.

```
char stack :: topele()
{
return (arr[top]);
}
```

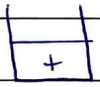
```
void postfix (char infix[])
{
    precedence temp;
    int i=0;
    Stack s;
    while (infix[i]!='\0')
    {                          // while
        temp = get_token (infix[i]);   // char -> named constant
    if (temp == operand)
        cout << infix[i];
    else if (temp == rparen)
    {
        while (s.topele()!='(')
        while (get_token (s.topele())!= lparen)
                cout << s.pop();
                char c = s.pop();          // pop & ignore lparen
        }
            pop
    else     //if operates
    {
        • if (s.isEmpty())
            s.push (infix[i]);
        • else
        {
    while (icp[temp]) <= isp[get_token (s.topele())]
                cout << s.pop();
                s.push (infix[i]);
        } }
            i++;
        }          // while
                s.display();
    }
```

Date: / /

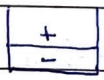
infix
↓ reverse infix '(' ↔ ')'
postfix() → 
↓ reverse
prefix

lower can should be down
& ~~lower p~~ higher should be higher.

a + b * c

token	Stack	o/p
i=4 c		c
i=3 *		
i=2 b		cb
i=1 +		cb*
i=0 a		cb*a → cb*a+
i=-1		↓ reverse
		+a*bc

a*b+c-d

d		d
-		
c		dc
+		
b		dcb
i=1 *		
i=0 a		dcba → dcba*+-
i=-1		↓ reverse
		-+*abcd

Date: / /

$(a+b)*(c-d)/e \longrightarrow)a+b(*)c-d(/e$

↓ ↓ ↓ ↓

) ( ) (