1 - 02/7/16

STACKS. UNIT - II

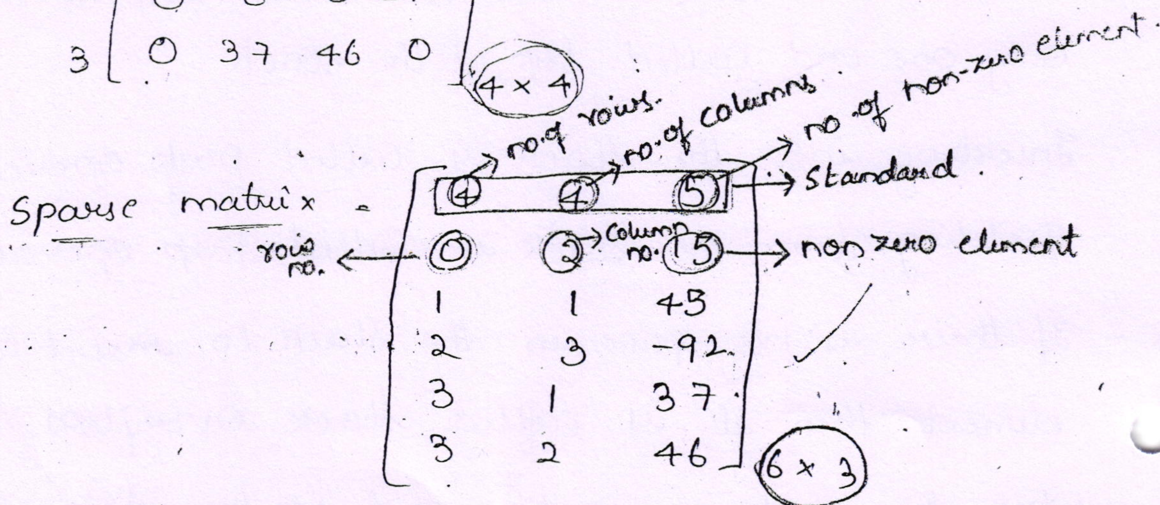
- Stack is a linear data structure which is based on LIFO (Last in First out).
- With stacks insertions and deletions happens through only one end called top of the stack.
- Inserting into the stack is called push operation.
- Deleting from the stack is called pop operation.
- If their is no space in the stack to insert a new element then it is called stack overflow.
- when the stack is empty, and we try doing any operations, then it is stack underflow.
- wan can implment the stack concept in 2 ways.
  (i) arrays
  (ii) linked list.

Applications of Stack:

- Converting an infix expression to its postfix notation.
- Evaluating the postfix expression.
- Nested function calls.
- Recursion.

Sparse Matrix Representation :-

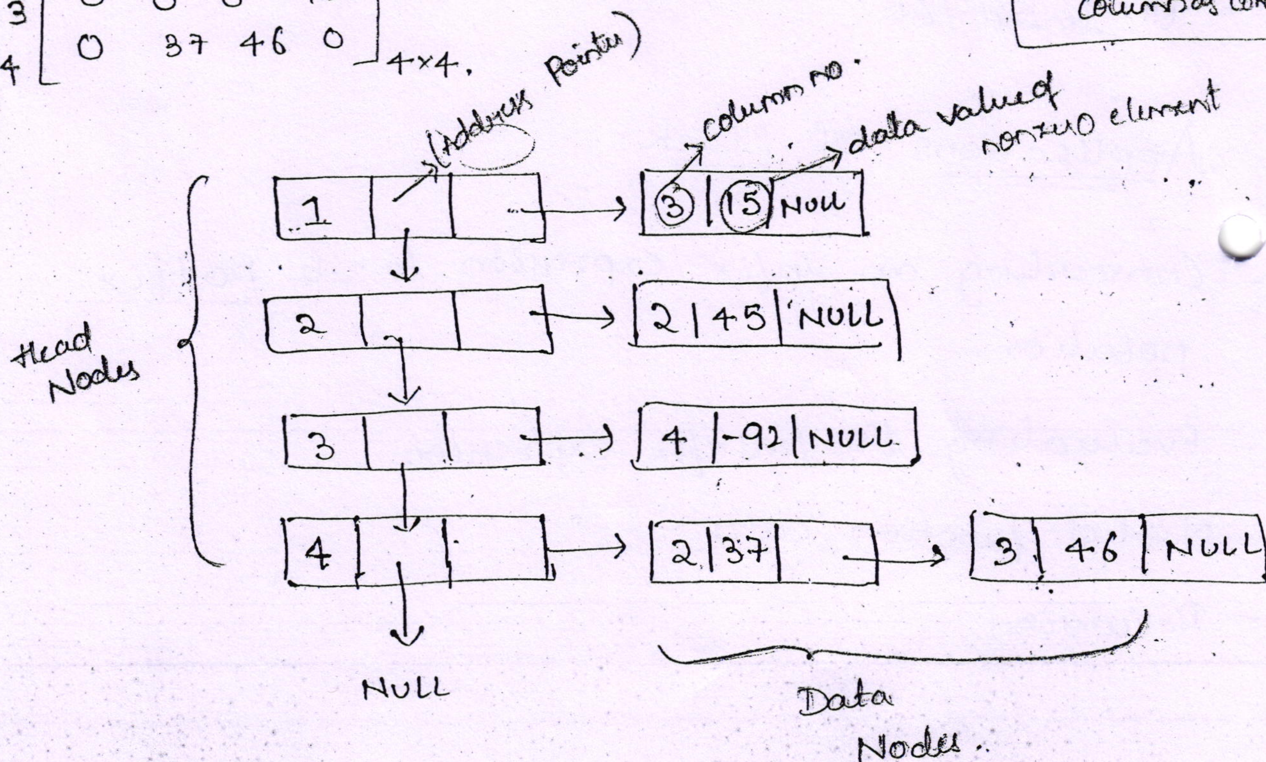
Array Representation .



Linked Representation :-

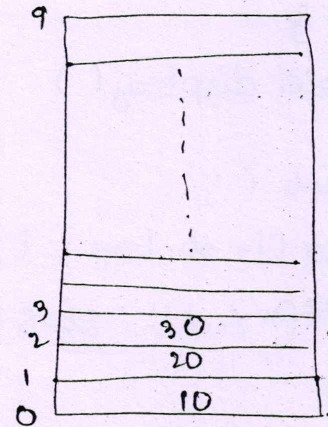
[row is taken as common]

⟹ Same linked representation can also be done by taki columns as comm



UNIT – II

```
# define SIZE 10
Struct stack.
{
int S[SIZE];
int top;
} st;
int isstoverflow()
{ int x;
if (st.top >= SIZE-1)
    x=1;
else
    x=0;
return x;
}
int isstunderflow()
{
int x;
if (st.top == -1)
  x=1;
else
  x=0;
return x;
}.
Void push (int x)
{
if (isstoverflow()==1)
Pf ("Stack overflow = %d", x);
else {
  St.top++;
  St.S[st.top]=x;
  Pf ("inserted");
  }
}.
```



```
void Pop ()
{
  int x;
 if (isstunderflow () = =1)
 Pf("Stack underflow")
else
  {
  x = st. S[st.top];
  St. top-- ;
  Pf("Element to be delete = %d", x);
  }
 }
void display ()
{
 int i;
for (i= st.top; i>=0; i--)
Pf("%d", st.S [i])
}

void main ()
{
St.top = -1;
 push(10);
 Push (20);
 Push (30);
 display();
  Pop();
  display();
 }.

void main(){
  st.top = -1;
  int x;
  Char ch;
  do {
   Pf (" enter the elements");
   Sf (" %d ", & x);
   Push (x);
   Pf("do you wish to continue (y/n)");
getchar();
ch = getchar();
} while (ch = = 'y');
display();
 POP ();
 display ();
 }
```

4|7

St
#
#
St
i
St
}
void
int
do {
Pf ("
Sf ("
newn
new
new
if (
}.
else
Pf
ge
Ch
}
wh
}

4/7/16

Stack Using Linked List :-

```
# include <stdio.h>
# include <stdlib.h>

struct node {
int data;
struct node * link;
} * top = NULL;

void push (){
struct node * newnode
int x, char choice;
do{

pf (" Enter the numbers ");
sf (" %d ", &x);

newnode = (struct node *) malloc (sizeof (struct node));
newnode -> data = x;
newnode -> link = NULL;
if (top == NULL)
    {
       top = newnode;
    }
else
    {
    newnode -> link = top;
      top = newnode;
    }
pf (" do you wish to continue (y/n)");
get char();
choice = get char();
}
While (choice == 'y');
}
```

```
void display ( )
 {
 struct node *temp = top;
 while (temp != NULL)
  {
   pf("%d", temp->data);
   temp = temp -> link;
  }
 }
void pop ( ) {
 Struct node *temp=top;
 if (top == NULL)
 {
 pf("cannot delete")
 }
 else {
   top = top -> link;
 }
}
void main(){
Push();
display();
Pop();
display();
}
```

Level of Priority

#

^

*, /, %

+, -

(

5/7/16

Converting Infix Expression to its Postfix form.

1) Read the infix expression.

2) Repeat steps 3, 4, 5, 6, 7, 8 for the each character in the infix expression.

3) Store current character in the ~~variable 'x'~~ infix expression in the variable 'x'.

4) If 'x' is left bracket (( ) put it in the stack.

5) If 'x' is an operand write it on the postfix expression.

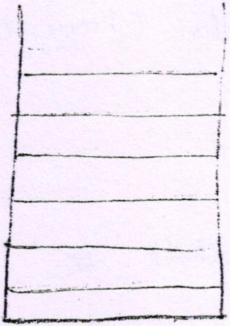
6) If 'x' is right bracket ( )) ~~for each operators~~ then remove all the operators from the stack and write them on the postfix expression until, we get the left bracket in the stack,
- Now remove left bracket the stack.

7) If 'x' is an operator
Remove all the operators from the stack whose Prescidence is greater than or equal to x:
Then, put them into postfix and then put the operator 'x' in the stack.

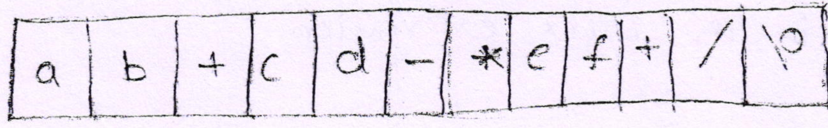
8) Retriev next character into 'x'
Until you reach NULL character.

9) Display the characters stored in postfix expression.

Infix Expression :- $((a+b)*(c-d)/(e+f))$



Stack



Postfix Expression

```
#
#

char St[20];
int top = -1;
void push (char x)
 {
   top ++ ;
   St [top] = x;
 }.

char pop ( )
{
  char x ;
  x = St [top] ;
  top -- ;
  return x ;
 }.
```

```
int
{.
if (

else

}.
int p
{
if (

else if

else if

else

}
Void
{
Char
char
int
pf('
gets (
for
{ x
if
}
```

```
int isoperator (int Char x)
{
if (x == '+' || x == '-' || x == '*' || ........
    return 1;
else
    return 0;
}
int prec (Char x)
{
if (x == '^')
    return 4;
else if (x == '*' || x == '/' || x == '%')
    return 3;
else if (x == '+' || x == '-')
    return 2;
else
    return 1;
}
Void main ( )
{
Char infix [80], Postfix [80];
char x, y;
int i, j = 0;

Pf ("enter infix Expression");
gets (infix);
for (i=0 ; infix [i] != '\0' ; i++)
{ x = infix [i];
  if (x == '(')
  { push (x);
  }
```

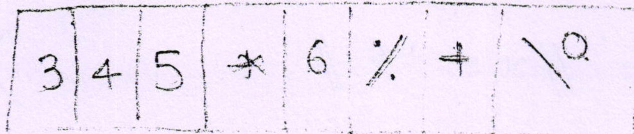
^	4
*, /, %	3
+, -	2
(,)	1

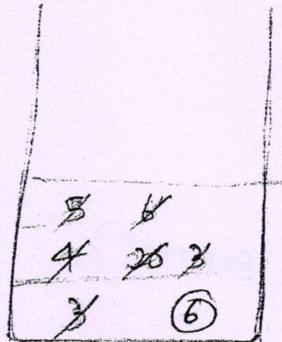
1/10

```
else if (x == ')')
{
while (St[top] != '(').
  {
   Postfix[j] = Pop();
   j++;
  }
 Pop();
}
else if (isoperator(x))
{
 while (prec(St[top]) >= prec(x)).
  {
   Postfix[j] = pop();
   j++;
  }
 Push(x);
}
else if (x >= 'a' && x <= 'z')
 {
  posfix[j] = x;
  j++;
 }
}
postfix[j] = '\0';
Puts(Postfix);
}.
```

Postfix Evaluation

Postfix:





b=5
a=4 } ①

b*a
= 20

b=20
a=6 } ②

20/6 = 3....

b=3
a=3 } ③

3+3
= 6

Ex 2 :- | 3 | 4 | + | 5 | 9 | - | * | 3 | 2 | + | / | → postfix

a=97   A=65
b=98   B= :
c :

'0' - 48
'1' - 49
'2' = 50
'3' - 51

```
#include <stdio.h>
int st[20];
int top = -1;
void push (int x)
{
  top++;
  st[top] = x;
}
int pop () {
  int x;
  x = st[top];
  top--;
  return x;
}.
```

```
int isoperator (char x )
{
if (x=='+' || x=='-' ||x=='*' || ........)

    return 1;

else
    return 0;

}.

void main( )            pf ("Enter the Expression)
                        = gets ( postfix )
{
  char postfix[80]; int i ; char x ; int a, b;
    for (i=0; postfix[i] != '\0' ; i++)
    {   x= Postfix [i] ;
        if (x>='0' && x<='9')
            Push (x-48);
        else if (isoperator(x))

        {
            b = POP();
            a = POP();
            Switch(x)
            {
              case  '+' : push (a+b); break;
              Case  '*' :  Push (a*b); break; ..
                      :
                      :

            }
          }
      }

  Pf ("%d", st[top] );

}.
```

11/7/1

- Qu
fir
- In
op
- Del
de
- Qu
to
- Qu
qu
- The
(i)
(ii)
Appl
(first c
- FCI
- usc
* Ins
* Del

11/7/16

Queue

- Queue is a linear data structure which follows first i first out (FIFO) mechanism., follows linear mechanism.
- Inserting data into the queue is called Enqueing/enqu Operation
- Deleting data from the queue is called dequeing/ deque operation.
- Queue Overflow :- When their is no space in the queue to add a new element is known as queue overflow
- Queue Underflow :- When they are no elements in the queue to remove elements is known as queue underflow
- The mechanism queue is implimented in 2 ways.
  (i) Arrays.
  (ii) Linked-List.

Applications of Queue :-

(first come first serve)
- FCFS C.P.U Scheduling algorithm.
- usage of a Printer in a shared networking environment

* Insertions into the queue we do through rear.
* Deletions through front.

Queue Using arrays

```
# define SIZE 10
int f = -1 , r = -1 ;
int q [SIZE];

int is qfull()
 {
   if (r >= SIZE-1)
     return 1;
   else
     return 0 ;
 }

int is qempty( )
 {
   if (f == -1)
     return 1;
   else
     return 0 ;
 }.

Void enqueue (int x)
{
 if (isqfull( ) == 1)
     Pf ("Queue overflow");
 else {
     r++;
     q [r] = x ;
     if (f == -1)
        f = 0 ;
   }
 }.
```

```
Voic
{
int
if(
Pf
else
for

}

Void
{
if

else

}
}
```

```
Void display( )
{
 int i ;
 if (is q empty( ) = = 1)
   pf (" No elements to display");
 else {
   for (i = f ; i <= r ; i++)
      pf ("%d ", Q[i]);
   }
 }

void dequeue( )
{
 if (is qempty ( ) = = 1)
     pf (" No elements to delete");
 else {
     x = Q[f];
     f++;
     pf (" The element deleted = %d", x);
   }
 }
```

12/7/16

Queue → Linked List

```
# include <stdio.h>
Struct node
{
 int data;
 Struct node *link;
} *f = NULL, *r = NULL;

          → Insert
Void enqueue( )
{
Struct node * newnode;
int x;
~~do~~
{
~~Pf("Enter the number");~~
~~Sf("%d", &x);~~
newnode = (Struct node *) malloc (Sizeof (Struct node));
newnode → data = x;
newnode → link = NULL;
If (f == NULL)
   {
   f = newnode;
   [r = newnode;] (not neccessary)
   }
else
   {
   r → link = newnode;
   }
 r = newnode;
}
```

You
Str
Whi

```
{
pf ("Can't insert new element");
}
else if (f==-1)
{
f=r=0;
DQ[f]=x;
}
else {
f--;
DQ[f]=x;
}
}
```

(or)
elements can be inserted by shifting element.

```
R++;
for(i=R; i>0; i--);
DQ[i]=DQ[i-1]
DQ[0]=x
```

```
Void deleter()
{
if (f==-1)
pf ("no elements in Queue to delete");
else if (f==r)
{
f=-1
r=-1;
}
else
{
r--;
}
}.
```

#
#
Str
int
Struct
}
Voi
{
Str
in
do
pf
sf
nec
ne
ne

u)

from

cition
ierted
be

tion only
d. only
e from

```
{
f = r = 0 ;
}
else {
r++ ;
}
DQ [r] = x ;
}
}

Void delet f( )
{ int x ;
if (f = = -1)
pf ("Empty can't delite");
else
{ x = DQ [f] ;
if (f = = r)
{
f = -1 ;
r = -1 ;
}
else
{
f++ ;
}
pf ("The element delited = %d", x) ;
}
}

Void insert f ( )
{
if (f = = 0)
```

(Arrays) DeQueue (Double ended Queue)

- Dequeue stands for double ended queue.
- In Dequeue insertion and deletions can be done from both the ends i.e., front end and rear end.

They are 2 types of Dequeue.

① I/P restricted Dequeue :- where their is restriction only in insertion, i.e., elements must be inserted only from rear end and deletions can be done from both the ends.

② O/P restricted Dequeue :- Where their is restriction only in deletion i.e., elements must be deleted only from front end and insertions can be done from both the ends.

```
#include <stdio.h>
# define SIZE 50.
 int DQ [SIZE];
 int f=-1, r=-1;

void insertr(int x)
{
  if (r = = SIZE-1)
     pf ("Queue overflow");
else {
     if (f = = -1)
```

```
void dequeue()
{
  int temp;
  if (f==-1)
    pf("Queue empty");
  else
  { temp=CQ[f];
    if (f==r)
    {
      f=r=-1;
    }
    else {
      f=(f+1)%SIZE;
    }
  pf("Element deleted =%d", temp);
  }
}

void display()
{
  int i;
  if (f==-1)
    pf("Queue empty");
  else
  {
   for(i=f; i!=r; i=(i+1)%5)
    {
     pf("%d", CQ[i]);
    }
   pf("%d", CQ[r]);
  }.
}
```