DATE: __/__/____
PAGE: _____

operating system

Stack

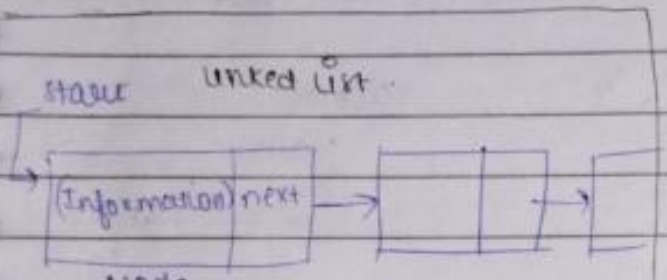
top = -1 [stack empty]
top = n-1 [stack full]

```
void main()
{
  int a[50]; //stack
  top=-1
  push(a, 5, top);
  push(a, 6, top);
```

```
void push(a[], x, top)
{
  if (top == n-1)
  {
    printf("stack overflow"); }
  Else
    { top = top+1
      a[top] = x }.
```

x = Element you want to insert / push.

```
int pop(a[], top)
{
  if (top == -1)
  { printf("stack underflow") }.
  Else
  { temp = a[top]; } temp = a[top]
    top = top-1;
    return temp;
  }
```




```
struct node
{
  int info;
  node * next;
};

node * stack, * n1, * n2, * n3
```

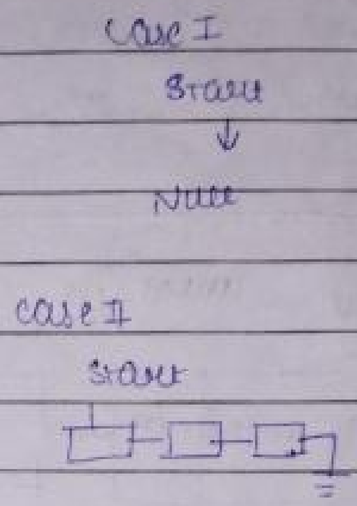
beginning insertion

```
node * insert_beg(node * stack, int x)
{
  node * newnode;
  newnode = (node*) malloc (size of (node));
  if (newnode == NULL)
  {
    printf("memory not available");
    exit(0);
  }
```

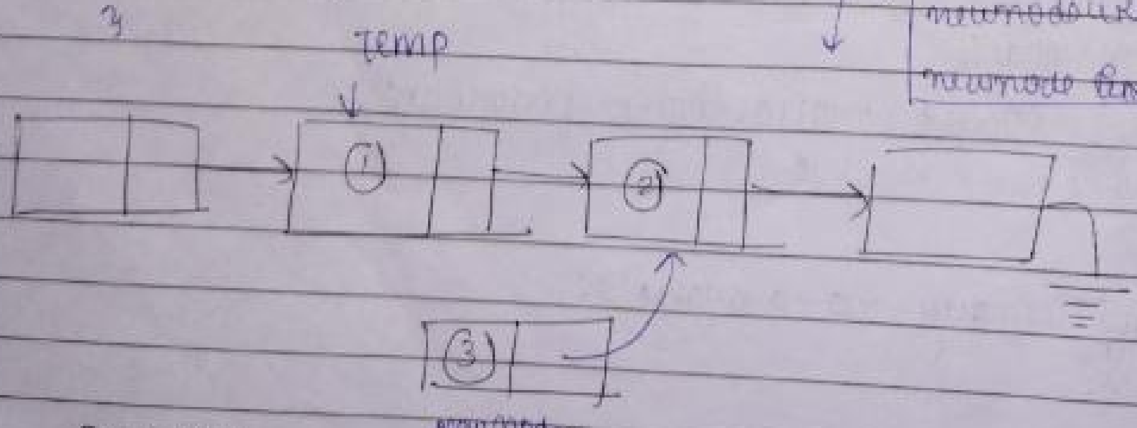
```
{
 newnode -> info = x;
 newnode -> link = start;
 start = newnode;
 return start;
 }
```

End insertion

```
node * insertat end (node * start, int x)
{
node * newnode, * temp;
 newnode = (node*) malloc (sizeOf (node));
 if (newnode = = NULL)
  { printf('memory not available'); }
   exit (0);
  }
 newnode -> info = x;
 if (start = = NULL)
    { start = newnode; }
 Else {
      temp = start;
      while (temp -> link != NULL)
      {
       temp = temp -> link }
      temp -> link = newnode;
      newnode -> link = NULL.
      return start;
      }
```



newnode
temp -> link = second ka address
newnode link equal to
newnode link = temp -> link



DATE: __/__/____
PAGE: _____

A: newnode → link = temp → link

B: temp → link = newnode.

mid insertion

```
node * insert_mid ( node * start, int x, int val )
{
  node * newnode; * temp;
  newnode = (node *) malloc (sizeof (node));
  if (newnode == NULL)
  {
    printf ("memory not available");
    exit(0);
  }
  newnode → info = x;
  temp = start;
  while (temp != NULL)
  {
    if (temp → info == val)
    {
      newnode → link = Temp → link;
      temp → link = newnode;
      break;
    }
  return start;
  }
```

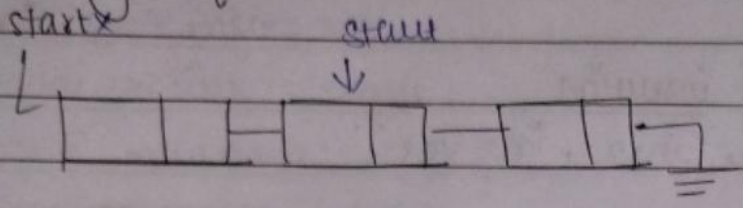
Que → create menu driven program -

choices :- insert at beginning, ~~dfdfd~~ end and middle.

Good Write

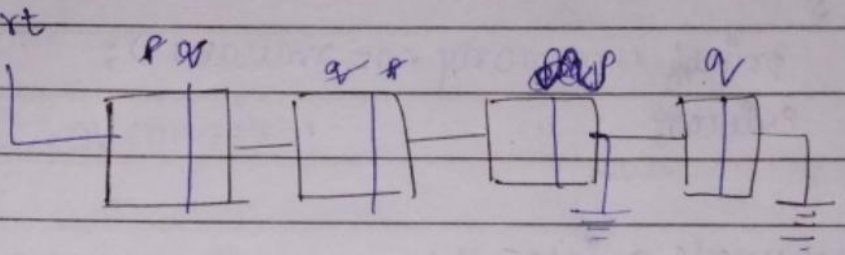
DATE: __/__/____
PAGE: _____

Deleting from linked list



delete from beg

```
temp = start
start = start → link
free(temp)
```

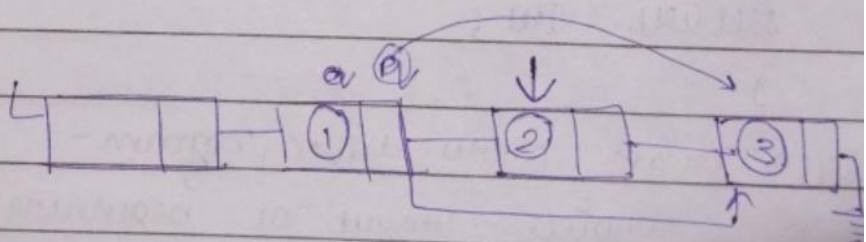


delete from end

```
node *p, *q;
q = start;
p = start;
while (q → link != NULL)
{
 p = q;
 q = q → link;
}
p → link = NULL
free(q);
```

delete from middle

```
q → link = p → link
free(p)
```



⇒ Delete from beginning

```
node * delete beg (node * start)
{
 node * p;
 if (start == NULL)
  { printf ("list is empty"); }
 else
 {
   p = start
   start = start → link;
   free (p)
  }
 return start;
}
```

⇒ Delete from end * start

```
node * delete end (node *start)
{
 node *p, *q;
 q = start
 if (start == NULL)
 { printf ('list is empty'); }
 Else { q = start;
  while if (q → link == NULL)   /* single node */
        { start = NULL;
          free (q);
        }
      else {
          while (q → link != NULL)
          {
            p = q;
            q = q → link; }
        }
          p → link = NULL;
          free (q);
```

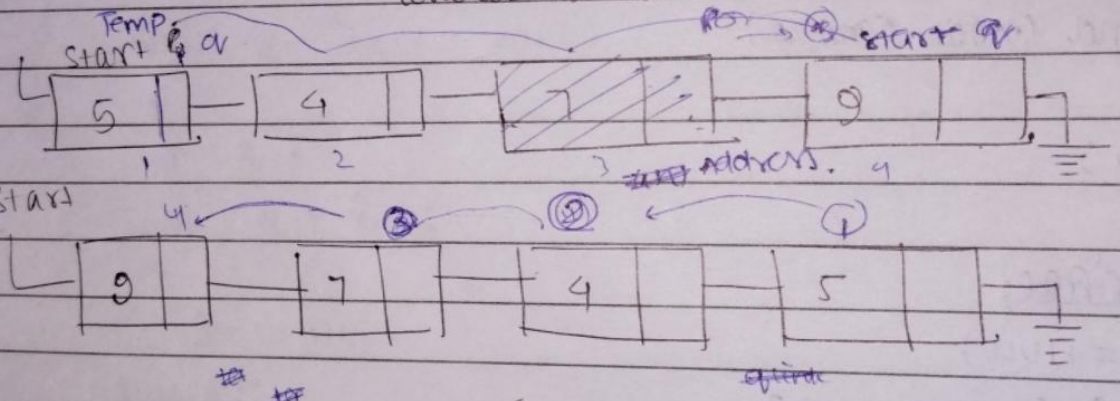
return start;
}.

⇒ delete from middle

```
node * deletemid (node *start)
{
 node *p, *q;
 if (start == NULL)
 { printf("list is empty"); }
else
 { q = start
   if (q != info) }
```

↳ information to be deleted.

Reverse a linked list



Stack [recursive function is an application of stack].

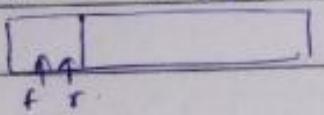
push → insertion at beg pop → delete at beg.

```
void printstack (node * top)
{
  node * temp;
  temp = top;
  while (temp != NULL)
  { printf ("%d", temp.info); }
    temp = temp->link }
```

Queue

Initial f = -1, r = -1 empty queue.

1st insertion f = 0, r = 0



Queue de

r++ (for insertion)

for deletion

f++

[if f == r then f = -1, r = -1]

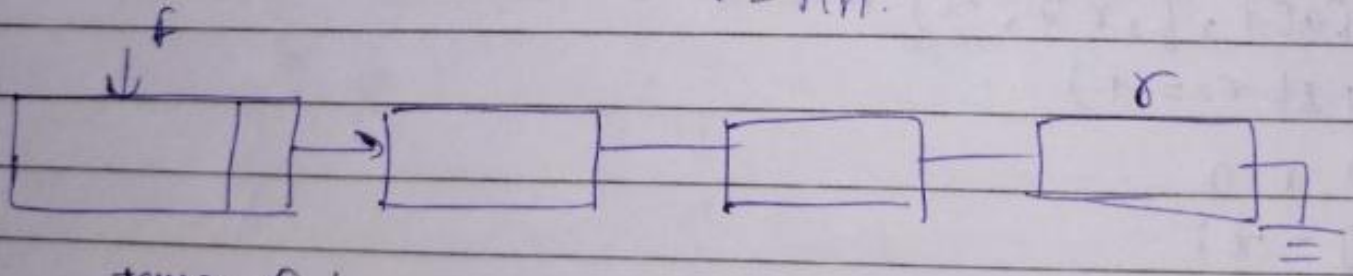
```
void insert [a[], f, r], x)
{ if (f == -1 && r == -1)
    f = 0, r = 0
    a[f] = x;
  }
else if (r <= n-1)
  { a[++r] = x; }
Else
  { printf ("queue overflow") }
```

```
void delete (a[0], f, r)
{ if (f == -1 && r == -1);
else if (f == r)
  { f = -1;
    r = -1;
  }
else { x = a[f]
       f++;
       return x;
     }
}
```

Printf (queue empt

Queue implementation using linked list

```
void insert (node *f, node *r, int x).
{
  node *nn;
  nn = (node *) malloc ( size of (node));
  if (nn == NULL)
  { printf('memory not available') }
  else
  { nn.info = x;
  if (f == NULL && r == NULL)
  { f = newnode;
    r = newnode;
    nn -> link = NULL;
  }
else {
    node * temp;
    r -> link = nn;
    nn -> link = NULL; }
    r = nn.
```




```
temp = f;
f = f -> link;
free (temp);
```

```
struct queue
{
 int info;
 queue * link;
};
queue *f = NULL
queue *r = NULL.
```

```
void
queue* Insert ( queue * f, queue * r,
                int val)
{
 queue * newnode;
 newnode = (queue*)malloc (sizeof(que
 if (newnode == Null):
 { — printf ('memory not available) ;
 else
 {.
  newnode → info = val;
  If (f == NULL · & & r == Null)
  { f = newnode;
    r = newnode;
  }
  else {
    r→link = newnode;
    r = newnode;
  }
  newnode → link = null
  }
```

PAGE:

Applications of stack

P → Postfix Expression

1) add ')' at end of P
2) Scan P from left to right until ')' encountered
{
3) If (An operand is encountered)
4) Push operand into stack
5) If (Operator is encountered)
6) { A = pop(s); (S → stack)
7) B = pop(s);
8) C = B operator A;
9) push(S,C);
}
10) value = S[top]

P = @ 5, 6, 2, +, *, 12, 4, /, -

Symbol scan	Stack	Remarks
5	5	
6	5, 6	
2	5, 6, 2	
+	5, 8	6+2 = 8
*	40	5*8 = 40
12	40, 12	
4	40, 12, 4	
/	40, 3	12/4 = 3
-	37	37

Convert Infix to post fix.

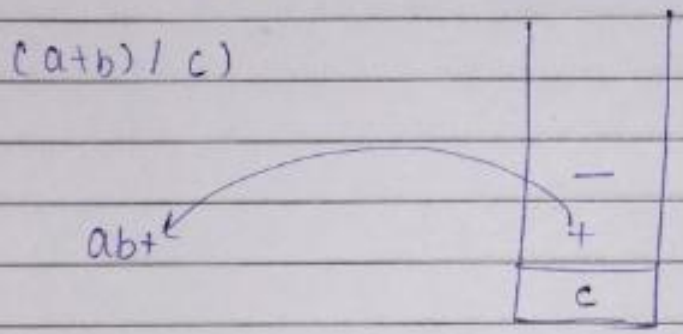
I = Infix P = Postfix

(1) Add '(' onto stack and ')' to the end of Queue.
(2) Scan I from left to right until stack is empty
(3) If (operand is encountered)
{ add it to P }

Good Write

If (left parenthesis is encountered)
push onto stack;
if (operator ⊗ is encountered)
{
(a) repeatedly pop from stack and add to P each operator which has same precedence or higher precedence than ⊗
(b) add ⊗ to stack.
}
if (right Parenthesis is encountered)
{
while (left parenthesis is encountered)
pop from stack and add to P each operator;
}
remove left parenthesis

(a+b)/c)



Q: A + (B * C - (D / E ↑ F) * G) * H

Symbol Scaned	Stack	P
1) A	~~A~~ (C	
+	~~A~~ C	A
C	C +	A
B	C + C	A
*	C + C	A,B
C	C + C *	
-	C + C *	ABC
	C + C -	ABC *

Good Write

DATE: __/__/____
PAGE: _____

C	C+C-C	ABC*
D	C+C-C	ABC*D
/	C+C-C/	
E	C+C-C/	ABC*DE
↑	C+C-C/↑	ABC*DE↑
F	C+C-C/↑	ABC*DEF
)	C+C-	ABC*DEF↑/
*	C+C-*	ABC*DEF↑/
G	C+C-*	ABC*DEF↑/G
)	C+-	ABC*DEF↑/G*-
*	C+*	ABC*DEF↑/G*-
H	C+*	ABC*DEF↑/G*-H
)		ABC*DEF↑/G*-H*+

```
void merge-sort (A, l, h)
{
  if (l<h)
  {
    m = [l+h/2]
  merge-sort(A, l, m);
  merge-sort (A, m+1, h);
  merge (A, l, m, n);
  }
}
```

Good Write