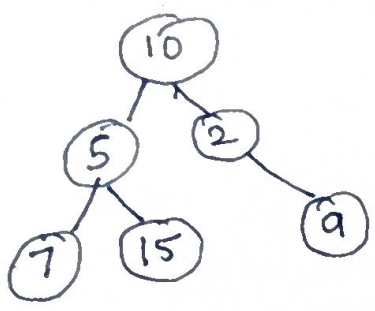
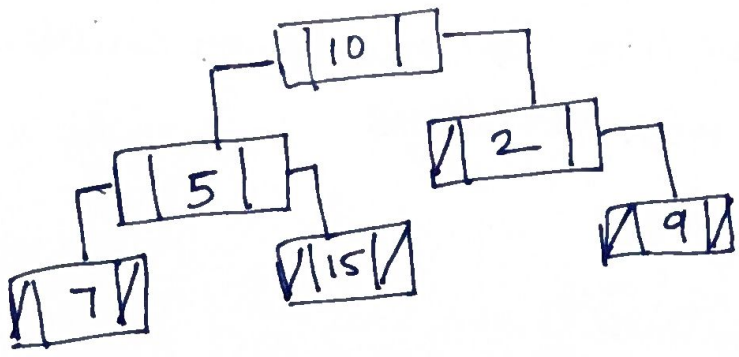
DSA — UNIT-5 B & B+ TREE

THREADED BINARY TREE:





no of nodes = n = 6
no of null address = n+1 = 7
→ no node so

Classification:

1. Left Threaded Binary Tree (predecessor node) → left address
2. Right Threaded " " (add of successor node) → right address
3. Fully Threaded Bina tree (predecessor node - left, sucessor node - right)

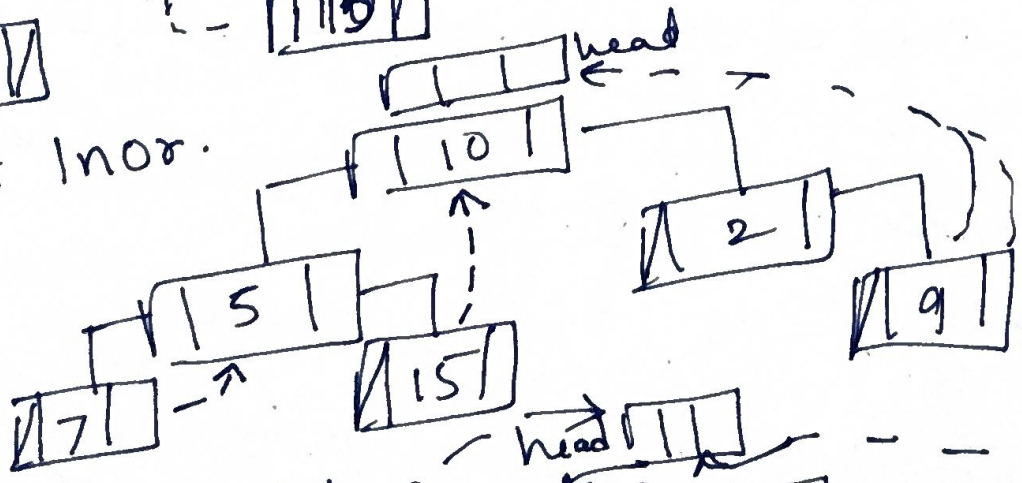
Threaded Binary: is a binary tree where in the address field will either hold addr of successor node/ predecessor node.

(1) Inorder Threaded Binary trees (2) Preorder (3) Postorder

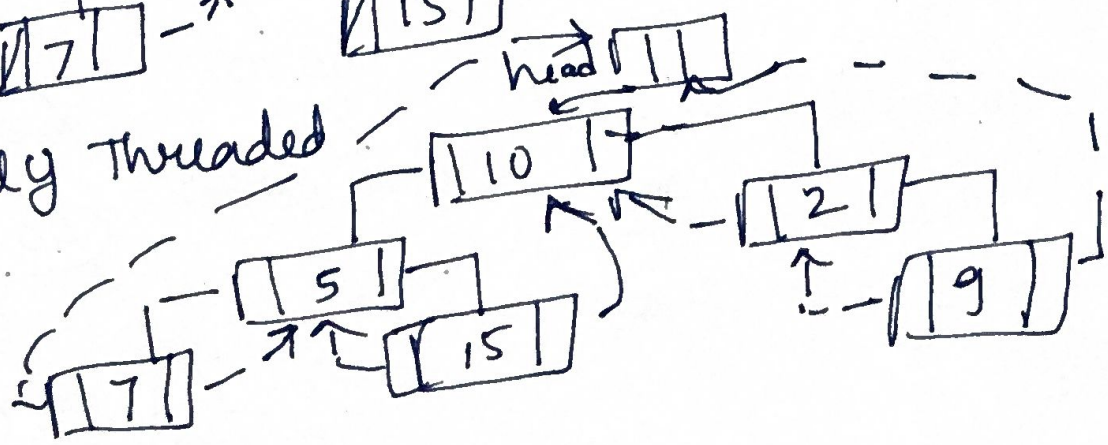
Inorder: 7 5 15 10 2 9

a) Left Inorder threaded binary tree.

b) Right Inor.



c) Fully Threaded

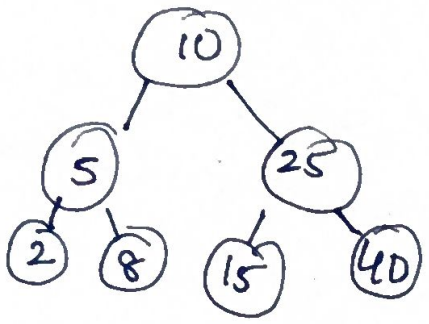


(m-way)

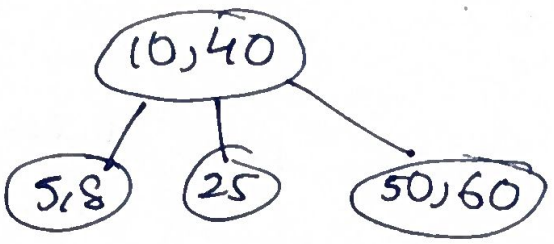
all leaf nodes will be at same level.

→ m → denotes order of tree → max of m children & each node can store / hold up to max m-1 keys

m = 2



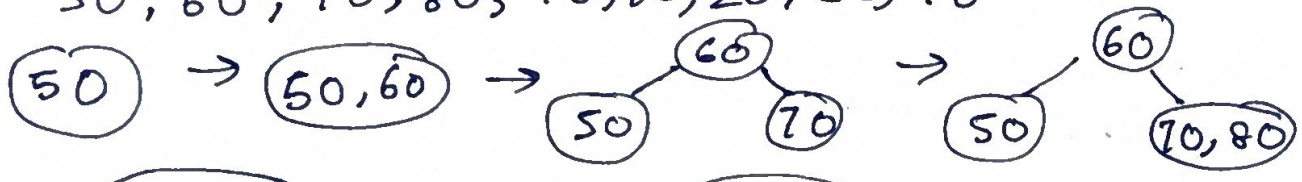
m = 3

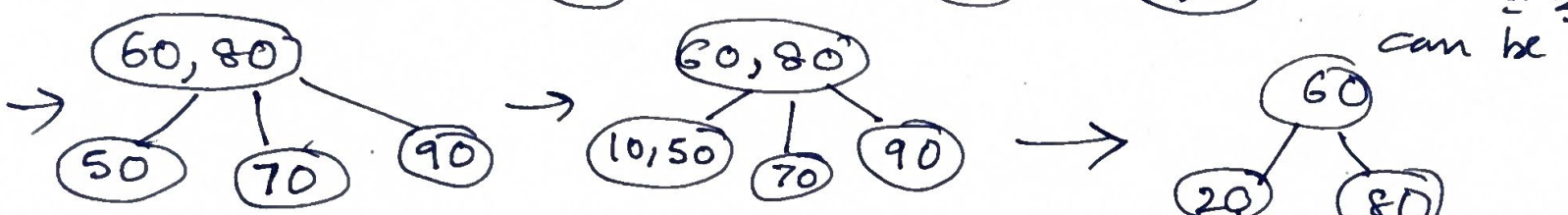


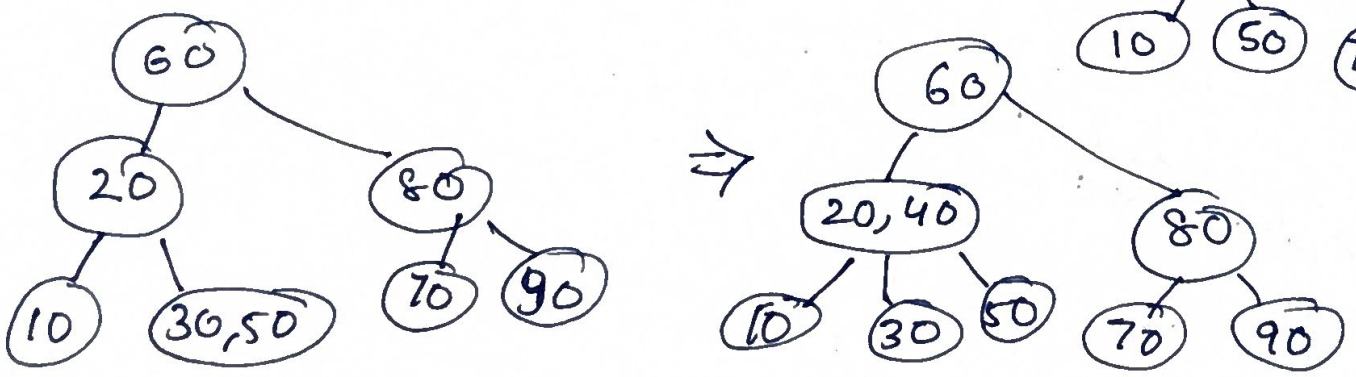
Q Construct B tree of order 3 for the keys

50, 60, 70, 80, 90, 10, 20, 30, 40

m = 3
max 3 children
each node : m-1
= 3-1 = 2 keys
can be stored



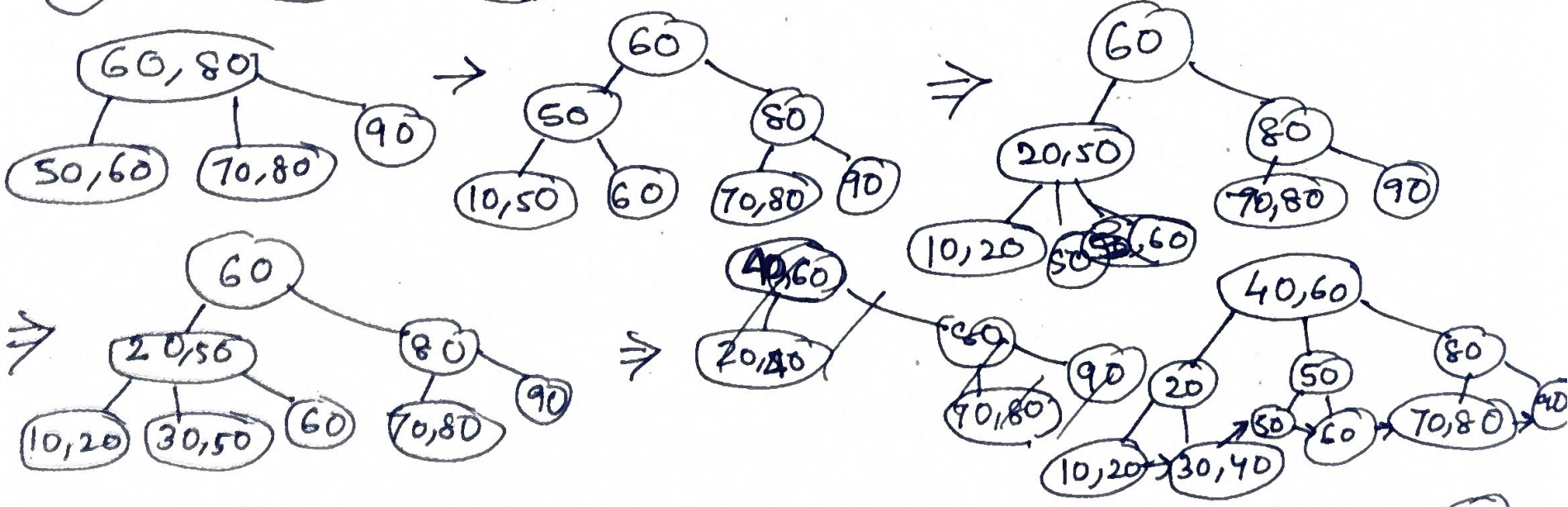




B+ TREE

Q construct B+ tree of order 3 for keys: 50, 60, 70, 80, 90, 10, 20, 30, 40.

m = 3
no. of children = 3
each node = 2



OBJECTIVE OF B & B+TREE:

(1) Reduced Tree height (2) Efficient Disk access
(3) Faster searching (4) Always balanced

B TREE	B+ Tree
(1) Data stored in both internal & leaf nodes	(1) Data stored only in leaf node
(2) Leaf nodes may/may not be linked.	(2) All leaf nodes are linked
(3) Less efficient	(3) Highly efficient
(4) Data may be found at any level	(4) Data found at leaf node only.
(5) more disk Input/output	(5) Fewer disk I/O operations

1st check for overflow ⇒ push, check for underflow ⇒ pop

Stack using Array -

```
# include <stdio.h>
# define SIZE 5
int top = -1;
int stack [SIZE];
void push(int item){
    if (top == SIZE-1)
        printf(" Overflow ");
    else { top = top+1;
           stack[top] = item;
         }
}
void pop ( ) {
    if (top = = -1)
        printf ("Underflow");
    else { printf ("Element deleted is
        %d", stack [top]);
           stack[top]
           top = top-1;
         }
}
void display ( ){
    int i;
    if (top = = -1)
        printf ("Stack empty");
    else {
        pf ("Stack contents are: ");
        for(i=top; i>=0; i--)
        pf ("%d\n", stack[i]);
    }
}
int main ( ){
    int ch, item;
    for (;;){
        printf ("1. Push");
        pf ("2. Pop");
        pf ("3. Display");
```

```
    pf("4.Exit");
pf("Enter choice:");
switch(c
 scanf("%d",&ch);
switch(ch)
 { case 1: Enter element
     printf("to push");
     scanf("%d",&item);
     push(item);
     break;
  case 2: pop();
          break;
  case 3: display();
          break;
  default : exit(0);
  }
}
return 0;
}
```

STACK USING STRUCTURE & POINTER

```
#include<stdio.h>
#define SIZE 5
struct stack{ int item;
              int data[SIZE];
}; typedef struct stack STACK;
void push( STACK *s, int item){
   if (s->top==SIZE-1)
      pf("Overflow");
   else {
        s->top=s->top+1;
        s->data[s->top]=item;
      }
}
int push(STACK *s){
   if (s->top==-1){
     pf("Stack underflow");
     return -1;
   }
   else{
     return s->data[(s->top)--];
   }
}
void display(STACK *s){
     int i;
     if (s->top==-1){
       pf("Stack empty");
     else { pf("Elements are:"
      for(i=s.top;i>=0;i--)
         pf("%d\n",s.data[i]);
      }
}
int main(){
   STACK S;
   S.top=-1;
   int ch,item,del;
   for(;;){
     pf("1.push"); pf("2.pop");
     pf("3.display"); pf("4.Exit");
    pf("Read choice");
    sf("%d",&ch");
    switch(ch){
      case 1: pf("Read elements to
          push");
        scanf("%d",&item);
       push(&S, item);
       break;
     case 2: del=pop(S);
             if(del!=-1)
                pf("Element delete
             is %d",del);
           break;
     case 3:display(S);
           break;
     default: exit(0);
    }
  } return 0;
}
```

EXPRESSION:

1. Infix: a + b
2. Postfix: ab +
3. Prefix: + ab

Infix	Postfix	Prefix
1) a+b*c	a+bc* abc*+	+a*bc
2) a+b-c	abc-+	+a-bc -+abc
3) (a+b)*c	ab+c*	+ab*c *+abc

4) ((((a+(b-c)*d)/e)^f)

Let $T_1 = b - c = bc-$ (a+bc-*d / abc-d*+)

(((a+T_1*d)/e)^f)

(((a+T_1d*)/e)^f) $T_2 = T_1 d *$

(((a+T_2)/e)^f) $T_3 = aT_2 +$

(T_3/e)^f)

(T_3e/f)^) $T_4 = T_3 e /$

(T_4f^)

T_3e/f^

aT_2+e/f^

aT_1d*+e/f^

abc-d*+e/f^ → postfix.

ALGO TO CONV INFIX TO POSTFIX

1. Scan infix exp from left to right
2. If scanned symbol is:
   i) operand → place on postfix
   ii) left parenthesis → pop stack & place on postfix until we get left parenthesis
   iii) Operator: if stack empty / top of stack is left parenthesis – push operator on stack.
   else: until top of stack having higher precedence compared to scanned symbol, pop stack & place on postfix.
   – push operator onto stack.
3. Until stack becomes empty pop stack contents & place on postfix.

Ex. a + b * c

Symbol	Stack	postfix
a	empty	a
+	+	a
b	+	ab
*	+, *	ab
c	+, *	abc
	Empty	abc*+

(a+b)*c

Symbol	Stack	Postfix
(	(	empty
a	(	a
+	(, +	a
b	(, +	ab
)	empty	ab+
*	*	ab+
c	*	ab+c
	empty	ab+c*

```
Void infixtopostfix (char infix[20],
 STACK *s){
 char postfix[20], symbol, temp;
int i=0, j=0;
for( i=0, infix[i] != '\0'; j++ ) {
        symbol = infix[i];
        if (isalnum(symbol))
            postfix[j++] = symbol;
        else {
            switch(symbol){
                case 'c': push(s,symbol);
                          break;
                case ')': temp=pop(s);
                  while (temp != 'c'){
                    postfix[j++]=temp;
                     temp = pop(s);
                  }
                  break;
    case '+':
    case '-':
    case '*':
    case '/':
    case '^': if(s->top==-1 ||
    s->data[s->top]=='C')
        push (s, symbol);
    else {
        while(preced (s->data[s->top])
          >= preced (symbol) &&
        s->top != -1 && s->top != 'c')
          postfix [j++] = pop (s);
        push (s, symbol);
        break; }
    }
}
while (s->top != -1)
    postfix[j++] = pop(s);
postfix[j] = '\0';
pf ("\n postfix expression is = %s",
        postfix);
}
```

```
# include<stdio.h>
# include<string.h>
# include<ctype.h>
# define SIZE 20
struct stack {
        int top;
        char data[SIZE];
}; typedef struct stack STACK;
void push (STACK *s, char item){
    s->data[s->top] = item; }
char pop (STACK *s) {
    return s->data[(s->top) - 1];
}
int preced (char symbol){
    switch (symbol) {
        case '^' : return 5;
        case '*' :
        case '/' : return 3;
        case '+' :
        case '-' : return 1;
    }
}
int main(){
    char infix[20];
    STACK S;
    S.top = -1;
    pf ("\n Read infix exp");
    sf ("%s", infix);
    infixtopostfix (infix, &S);
    return 0;
}
```

EVALUATION OF POSTFIX EXPRE

1. Scan a postfix RXP from left to right
2. if scanned symbol is
   i) operand – push it on stack
   ii) operator – pop 2 elements from stack & assign opr 2 & operator 1 respectively, perform operation op1 operator op2, push result onto stack.
3. Pop content of stack to get final result

3 + 4 * 5 → 345 * +

scanned symbol	Stack	op 1	op 2	result
3	3	E	E	E
4	3,4	E	E	E
5	3,4,5	E	E	E
*	3,20	4	5	4*5=20
+	23	3	20	3+20=23

(7 + 3) / 2 ⇒ 73 + 21

symbol	stack	op1	op2	result
7	7	E	E	E
3	7, 3	E	E	E
+	10	7	3	7+3=10
2	10,2	E	E	E
/	5	10	2	10/2=5

```
float postfix_evaluate(char postfix[20],
    STACK *S) {
int i; char symbol;
 float opr1, opr2, result;
 for(i=0; postfix[i]!='\0'; i++)
  { symbol = postfix[i];
    if (isdigit(symbol))
        push(s, symbol-98);
    else if (isalpha(symbol)){
        pf("Read value:");
        sf("%f",&value);
        push(s, value);
    else{
        opr2 = pop(s)
        opr1 = pop(s)
        res = operate(opr1, opr2,
         symbol);
        push(s, res);
  } }
 return pop(s); }
# include <stdio.h>
# include <ctype.h>
# def # define SIZE 20
struct stack { int top;
            float data[SIZE];
}; typedef struct stack STACK;
void push(STACK *s, float item){
    s → data[++(s→top)] = item;
}
float pop(STACK *s) {
   return s→data[(s→top)--];
}
float operate(float opr1, float
 opr2, char symbol){
switch(symbol){
   case '+': return opr1+opr2;
   case '-': return opr1-opr2;
   case '*': return opr1*opr2;
   case '/': return opr1/opr2;
   case '^': return opr1^opr2;
}}
int main(){
   char postfix[15]; STACK S;
   float result;
     s.top=-1;
   pf("Read postfix");
   sf("%s", postfix);
   result = postfix.evaluate(postfix
            &s);
   pf("final result is %f", result);
  return 0;
}
```

RECURSION:

Factorial of n number

~~Fact(n) = n * Fact(n-1)~~

```
int Fact (int n){
    if (n==1)
        return 1;
    return n* Fact(n-1);
}
```

GCD of 2 no.

```
int GCD(int a, int b){
    if (a==b)
        return a;
    else if (a>b)
        return GCD(a-b, b);
    return GCD(b, a);
}
```

Product of 2 no.s

```
int multiply (int a, int b) {
    if (b==1)
        return a;
    return a + multiply(a, b-1);
}
```

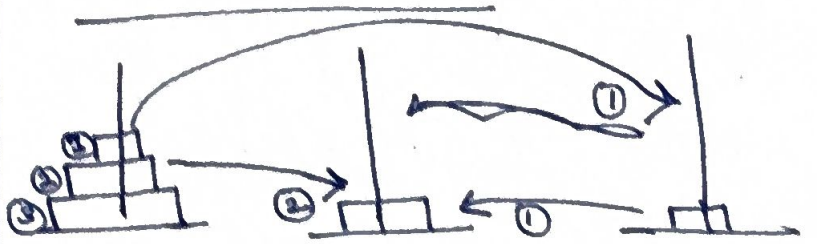
Sum of all element in array -

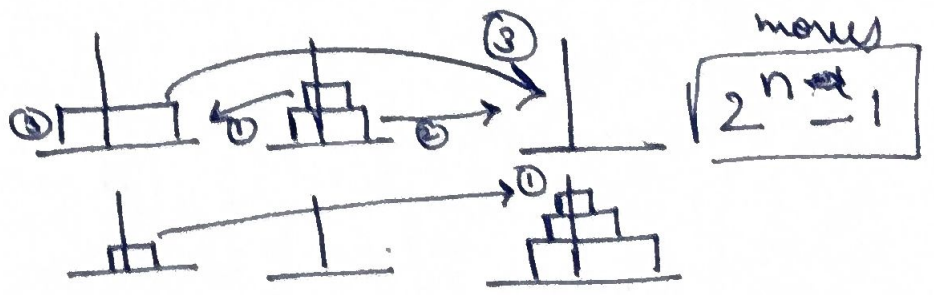
```
int sum (int A[10], int n) {
    if (n==1)
        return A[0];
    return sum(A, n-1) + A[n-1];
}
```

Binary Search

```
int binary (int arr[], int left, int right,
int key) {
  if (right >= left){
      int mid = left + (right - left)/2;
      if (arr[mid]==key)
          return mid;
      if (arr[mid]>key){
         return binary (arr, left, mid-1,
           key);
      }
    return binary (arr, mid+1, right, key
  }
  return -1;
}
```

TOWER OF HANOI






```
void TowerHanoi (int n, char s, char T,
 char D, char T) {
 if (n==1)
     Pf ("Move %d disc from %c to %c",
       n, S, D);
 else {
     TowerHanoi(n-1, S, T, D);
     Pf ("Move %d disc from %c to %c",
       n, S, D);
     TowerHanoi(n-1, T, D, S);
 } }
```

UNIT-2 – QUEUE ⇒ FIFO

USING ARRAY

```
void Enqueue (int item) {
   if (rear == SIZE-1)
         Pf("Queue is full");
   else {
        rear = rear+1;
       q[rear] = item;
       if (front == -1)
            front = front+1;
     }
}
void Dequeue( ) {
       int del;
       if ( front == -1)
          Pf ("Queue is empty");
       else{ del = q[front];
            Pf("Element deleted is %d",
              del);
            if (front == rear)
                {
                front = -1;
                rear = -1;
                }
           else{front = front+1;
           }
     }
}
void display( ) { int i;
  if (front == -1 )
      Pf ("Queue empty");
  else{
      Pf ("Queue contents are");
      for (i=f; i<=r; i++)
          Pf("%d", q[i]);
  }
}
```

```
# include <stdio.h>
# define SIZE 5
 int front = -1 , rear = -1;
int q[SIZE];
 int main() {
    int item, ch;
    for (;;) { Pf ("1. Insert");
     Pf ("2. Delete");
     Pf ("3. Display");
     Pf ("4. Exit");
     Pf ("Enter choice");
     Sf ("%d", &ch);
     switch (ch) {
        case 1: Pf ("Read elements
          to insert: ");
          Sf ("%d", &item);
          enqueue (item);
          break;
        case 2: Dequeue ();
                break;
        case 3: display ();
                break;
        default: exit (0);
     }
    } return 0;
}
```

QUEUE OF INT USING STRUCT & POINTER

```
#include <stdio.h>
# define SIZE 10
 struct queue {
          int front, rear;
          int data[SIZE];
}; typedef struct queue
   QUEUE;
```

```
void Enqueue(QUEUE *q, int item)
{ if(q->rear == SIZE-1)
      pf("Queue full");
  else {
      q->rear = q->rear+1;
      q->data[q->rear] = item;
      if(q->front == -1)
          q->front = q->front+1;
    }
}
void Dequeue(QUEUE *q){int del;
  if(q->front == -1)
      pf("Queue is empty");
  else { del = q->data[q->front];
      pf("Element deleted %d",
          del);
      if(q->front = q rear)
        { q->front = -1;
          q->rear = -1;}
      q->front = q->~~rear~~front+1;
    }
}
void display(QUEUE q){ int i;
  if(q.front == -1)
      pf("Queue is empty");
  else{pf("Queue content are");
      for(i=q.front; i<=q.rear;
          i++)
      pf("%d", q.data[i]);
  } }
int main(){ QUEUE q;
    q.front = -1; q.rear = -1;
    int ch, item;
    for(;;){
      pf("1.Insert"); 2.Dellte;
       3. Display; 4. Exit;
      pf("Enter choice");
      sf("%d", &ch);
      switch(ch){
        case 1: pf("Enter element
            to add:");
          sf("%d", &item);
          Enqueue(&q, item);
          break;
        case 2: Dequeue(&q);
              break;
        case 3: Display(q);
              break;
        default: exit(0);
        }
    } return 0;
}
```

~~CIRCULAR~~ QUEUE DRAWBACKS

If we delete any element from starting but last (rear) elem is present so it will show stack is full.

CIRCULAR QUEUE:

```
Void Enqueue(int item){
  if(f==(r+1)%SIZE){
      pf("queue is full");
  else{
      r = (r+1)%SIZE;
      q[r] = item;
      if(f == -1)
          f = f+1;
    }
}
```

```
void Dequeue() {
    if (f == -1)
        pf("Queue is empty");
    else {
        pf("Element deleted is %d",
            a[f]);
        if (f == r) {
            f = -1;
            r = -1; }
        else
            f = (f+1) % SIZE; }
}
void display { int i;
 if (f == -1)
    pf("Queue empty");
 else { pf("contents are:");
    for(i=f; i<=r; i++)
        pf("%d", q[i]);
    pf("%d", q[i]);
  }
}
#include <stdio.h>
#define SIZE 5
int f = -1, r = -1;
int q[SIZE];
int main(){
  int ch, item;
  for(;;) {
   pf("1. Insert"); 2. Delete;
  3. Display ; 4. Exit;
   pf("Enter choice:");
   sf("%d", &ch);
  switch(ch){
     case 1: pf("Enter item
                to add:");
     sf("%d", &item");
     Enqueue(item);
     break;
  case 2: Dequeue();
          break;
  case 3: Display();
          break;
  default: exit(0);
  }
 } return 0;
}
```

CIRCULAR QUEUE USING POINTER

```
#include <stdio.h>
#define SIZE 10
struct queue { int front, rear;
               int data[SIZE];
}; typedef struct queue QUEUE;
void Enqueue (QUEUE *q, int item)
 { if (q->front == (q->r+1) % SIZE
        pf("Queue full");
  else { q->rear = (q->rear+1) % SIZE
       q->data[q->rear] = item;
      if (q->front == q->rear -1)
          q->front = q->front+1;
    }
}
void Dequeue (QUEUE *q) {
    if (q->front == -1)
        pf("Queue empty");
    else {
        pf("Element deleted
          is %d", q->data[q->front]
        if (q->front == q->rear) {
            q->front = -1;
            q->rear = -1;
        }
    else
        q->front = (q->front+1) %
                   SIZE;
} }
```

```
void display (CQUEUE q){
  int i;
  if (q.front == -1)
      pf ("Queue Empty");
  else {pf ("Queue elements: ");
    for (i=q.f ; i!=q.r ; i=(i+1)%
      SIZE)
        pf ("%d ", q.data[i]);
    pf ("%d", q.data[i]);
}
int main() { int ch, item;
  CQUEUE q;
  q.front = -1, q.rear = -1;
  for (;;)
      .
      .
      .
    case 1: pf ("Element to insert");
      sf ("%d", &item);
      Enqueue (&q, item);
      break;
    case 2: Dequeue (&q);
      break;
    case 3 : Display (q);
      break;
  :
}
```

ASCENDING PRIORITY QUEUE

```
void Enqueue (QUEUE *q, int item)
{ int pos;
  if (q->rear == SIZE-1)
      pf ("Queue full");
  else { pos = q->rear;
    q->rear = q->rear+1;
  while(pos>=0 && q->data[pos]>=
    item) {
    q->data[pos+1]=q->data[pos];
    pos = pos-1;
  }
  q->data[pos+1] = item;
  if (q->front == -1)
      q->front = q->front+1;
  }
```

INFIX TO POSTFIX

1. Scan infix exp → right to left
2. scanned symbol
   i) Operand → place on prefix exp
   ii) ')' → push on stack
   iii) '(' → pop stack content & place on prefix
   iv) until stack empty pop contents & place on prefix.
4) Reverse prefix exp to get final result

Q. a+b*c = +a*bc

scanned symbol	stack	Prefix
c	empty	c
*	*	c
b	*	cb
+	+	cb*
a	+	cb* a
		cb* a +

o/p :→ + a * cb ↵

(a+b) * c

scanned symbol	stack	Prefix
c	empty	c
*	*	c
)	*)	c
b	*)	cb
+	*) +	cb
a	*) +	cba
(	*	cba +
		cba +*

*+abc

PARENTHESIS CHECKER

```
int balparencheck(STACK *s, char expr[20])
{ int i;
  char temp, symbol;
  flag = 1;
  for(i=0; expr[i]!='\0'; i++){
    symbol = expr[i];
    if(symbol=='('||
    symbol=='['||symbol=='{')
      push(s, symbol);
    else{
      if(s->top==-1)
        flag=0;
      else{ temp=pop(s);
        if(symbol==')'&&
        (temp=='['||temp=='{')
          flag=0;
        if(symbol==']'&&(temp
        =='('||temp=='{')
          flag=0;
        if(symbol=='}'&&
        (temp=='('||temp=='[')
          flag=0;
      }
    }
  }
  if(s->top!=-1)
    flag=0;
  return flag;
}
```

DYNAMIC MEMORY ALLOCATION

MALLOC : sum of n elements dynamically

```
#include<stdio.h><stdlib.h>
int main(){
  int sum=0, i, n;
  int *ptr
  pf("Read value for n:");
  sf("%d", &n);
  ptr = (int*)malloc(n*sizeof(int));
  pf("Read elements");
  for(i=0; i<n; i++){
    scanf("%d", ptr+i);
    sum = sum + *(ptr+i);
  }
  pf("sum=%d", sum);
  free(ptr);
  return 0;
}
```

LINKED LIST

Pg 39/40...

i) delete node at beginning of list

```
NODE deletebegin(NODE start){ NODE temp;
  if(start==NULL)                =start
    {pf("List empty");
     return NULL;
    }
  start = start->addr;
  pf("Node deleted is %d",
      temp->data);
  free(temp);
  return start;
}
```

ii) del node at end of last

```
NODE deleteend (NODE start {
    NODE cur, prev;
    if (start == NULL) {
        pf ("List is empty");
        return NULL;
    }
    prev = NULL;
    cur = start;
    while (cur -> addr != NULL) {
        prev = cur;
        cur = cur -> addr;
    }
    prev -> addr = NULL;
    pf ("Node del is %d", cur -> data);
    free (cur);
    return start;
}
```

pg 41

i) Insert at beginning

```
NODE insertbegin (NODE start, int item) {
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    temp -> data = item;
    temp -> addr = start
    return temp;
}
```

ii) At end:

```
NODE insertend (NODE start, int item) {
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    temp -> data = item;
    temp -> addr = NULL;
    if (start == NULL)
        return temp;
    cur = start;
    while (cur -> addr != NULL)
        cur = cur -> addr;
    cur -> addr = temp;
    return start;
}
```

refer pg 42 & 43

STACK OF INT USING SINGLY L.L.

```
#include <stdio.h> <stdlib.h>
# define SIZE 5
int count = 0;
struct node {
    int data;
    struct node * addr;
}; typedef struct node *NODE

NODE push (NODE start, int item) {
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    if (count >= SIZE) {
        pf ("\n Stack overflow");
        return start;
    }
    temp -> data = item;
    temp -> addr = start;
    count = count + 1;
    return temp;
}

NODE pop (NODE start) {
    NODE temp;
    if (start == NULL) {
        pf ("stack underflow");
        return NULL;
    }
    temp = start;
    start = start -> addr;
    pf ("Element popped is %d", temp -> data);
```

```
        free(temp);}
        count=count-1;
        return start;
}
void display(NODE start){
    NODE temp;
    if(start==NULL)
        pf("stack empty");
    else {
        pf("stack contents are:");
        temp=start;
        while(temp!=NULL){
            pf("%d",temp->data);
            temp=temp->addr;}
        }
}
int main(){ NODE start=NULL;
int ch, item; for(;;){
 pf("1.Push"); ("2.Pop"); ("3.Display")
("4.Exit"); ("Read choice:");
sf("%d",&ch); switch(ch){
  case 1: pf("Element to insert");
          sf("%d",&item);
          start=push(start,item);
          break;
  case 2: start=pop(start);
          break;
  case 3: display(start);
          break;
  default: exit(0);
    }
  } return 0;
}
```

QUEUE OF INT USING SINGLY LL

```
#include<stdio.h><stdlib.h>
#define SIZE 5
int count=0
struct node{ int data;
             struct node *addr;
}; typedef struct node *NODE;
NODE Enqueue(NODE start, int item){
 NODE cur, temp;
 if(count>=SIZE){
     pf("Queue is full");
     return start; }
temp=(NODE) malloc(sizeof(struct node));
temp->data=item;
temp->addr=NULL;
 cur=start;
 while(cur->addr!=NULL)
     cur=cur->addr;
cur->addr=temp; count=count+1;
return start;
}
NODE dequeue(NODE start){
 NODE temp;
 if(start==NULL){
     pf("Queue is empty");
     return NULL; }
temp=start;
start=start->addr;
pf("Node deletdis %d",temp->data);
free(temp); count=count-1;
 return start; }

void display(NODE start){
NODE temp; if(start==NULL)
 pf("Queue is empty");
else { pf("Queue contents are");
   temp=start; while(temp!=NULL){
   pf("%d",temp->data); temp=temp->addr;}
```

```
} } }
int main() { NODE start = NULL;
int ch, item; for(;;) {
case 1: pf("read element to insert");
        sf("%d", &item);
        start = Enqueue(start, item);
        break;
  .
  .
  .
}
```

ORDERED LINKED LIST:

```
NODE orderlist(NODE start, int item)
{ NODE temp, prev, cur;
 temp = (NODE) malloc(sizeof(struct node));
 temp->data = item;
 temp->addr = NULL;
 if (start == NULL)
        return temp;
 if (item < start->data) {
        temp->addr = start;
        return temp;
 }
 ~~while~~
 prev = NULL;
 cur = start;
while(cur != NULL && item > cur->data){
        prev = cur;
        cur = cur->addr;
}
prev->addr = temp;
temp->addr = cur;
return start;
}
```

REVERSE SINGLY LINKED LIST

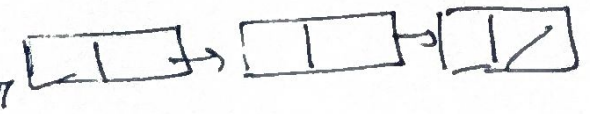
```
NODE rev(NODE start){
  NODE temp = NULL, prev = NULL;
  cur;
  if (start == NULL) {
        pf("List empty");
        return NULL; }
  cur = start;
  while (cur != NULL) {
        temp = prev;
        prev = cur;
        cur = cur->addr;
  prev->addr = temp;
}
```

DELETE USING NODE SINGLY LINKED LIST

```
Node deletekey(NODE start, int key){
   NODE prev, temp, cur;
  if (key == start->data) {
        temp = start;
        start = start->addr;
        pf("Node deleted is %d",
         temp->data);
        free(temp);
        return start;
}
```

TRAVERSE TO NODE TO DELETE

```
prev = NULL; cur = start;
while (cur != NULL && key != cur->data){
        prev = cur;
        cur = cur->addr; }
if (cur == NULL) {
        pf("Key not found");
        return start;
}
prev->addr = cur->addr;
  pf("Node deleted is %d",
    cur->info);
  free(cur);
  return start;
}
```



DELETE BASED ON POSITION

1) ~~start~~ Delete 1st node

```
NODE deletepos(NODE start, int pos)
{NODE temp, cur, prev; int count=1;
    if (pos==1){
        temp = start;
        start = start->addr;
        pf("Node del is %d",
          temp->adddta);
        free(temp);
        return start;
    }
```

2) SOME POSITION

```
prev = NULL, cur = start;
while (cur != NULL && pos != count){
    prev = cur;
    cur = cur->addr;
    count = count+1;
}
if (pos != count){
    pf("Node not found");
    return start;
}
prev->addr = cur->addr;
pf("Node dele is %d", cur->data);
free(cur);
return start;
}
```

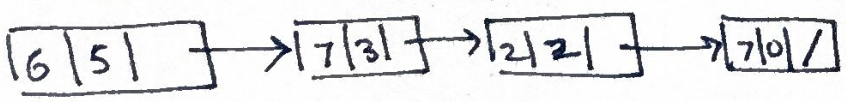
~~CIRCULAR SINGLY LIN~~ DOUBLY

POLYNOMIAL MULTI OF ~~SINGLY~~

LINKED LIST

$6x^5 + 7x^3 + 2x^2 + 7$

6 * X ^ 5 +

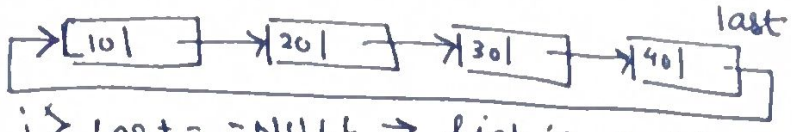



```
# include <stdio.h> <stdlib.h>
struct node { int co, po;
        struct node *addr;
}; typedef struct node * NODE;
```

```
NODE insertend(NODE start, int co,
int po) {
    NODE temp; cur;
    temp = (NODE)malloc(sizeof(struct
                        node));
    temp->co = co;
    temp->po = po;
    temp->addr = NULL;
    if (start == NULL)
        return temp;
    cur = start;
    while (cur->addr != NULL)
        cur = cur->addr;
    cur->addr = temp;
    return start;
}
void display(NODE start){
    NODE temp;
    if (start == NULL)
        pf("Poly empty");
    else{ temp = start;
        while(temp->addr != NULL){
            pf("%d*x^%d+",
              temp->co, temp->po);
            temp = temp->addr;
        }
        pf("%d*x^%d+", temp->co
          temp->po);
    }
}
NODE multiplypoly(NODE poly1, NODE
  poly2){
  NODE first, second, res = NULL
  for(first = poly1; first != NULL; first = first
                                  ->addr)
    for(second = poly2; second != NULL;
      second = second->addr)
      res = addterm(res, first->co*
      second->co, first->po+second->po);
  return res;
}
NODE addterm(NODE res, int co, int po){
    NODE temp, cur; int flag = 0;
    temp = (NODE)malloc(sizeof(struct node
    temp->co = co;
    temp->po = po;
    temp->addr = NULL
    if (res == NULL)
        return temp;
```

```
    cur = res;
    while (cur != NULL) {
        if (cur->po == po) {
            cur->co = cur->co + co;
            flag = 1;
        }
        cur = cur->addr;
    }
    if (flag == 0) {
        cur = start;
        while (cur != NULL)
            cur = cur->addr;
        cur->addr = temp;
    }
    return res;
}

int main() {
    NODE poly1 = NULL, poly2 = NULL, res;
    int i, n, co, po;
    Pf ("Read no of terms of 1st poly: ");
    Sf ("%d", &n);
    for (i = 1; i <= n; i++) {
        Pf ("Read co & po of %d term: ", i);
        Sf ("%d %d", &co, &po);
        poly1 = insertend (poly1, co, po);
    }
    pf ("First polynomial is");
    display (poly1);
    Pf ("Read no. of terms of 2nd poly: ");
    Sf ("%d", &n);
    for (i = 1; i <= n; i++) {
        pf ("Enter co & po of %d term: ", i);
        Sf ("%d %d", &co, &po);
        poly2 = insertend (poly2, co, po);
    }
    pf ("Second polynomial is");
    display (poly2);
    res = multiplypoly (poly1, poly2);
    pf ("Resultant polynomial is");
    display (res);
    return 0; }
```

CIRCULAR SINGLY LINKED LIST



i) last == NULL → list is empty
ii) last → addr == last → list is having only 1 node

```
struct node { int data;
        struct node * NODE;
}; typedef struct node * NODE
```

```
NODE insertbegin (NODE last, int item) {
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    temp -> data = item;
    if (last == NULL) {
        temp -> addr = temp;
        return temp;
    }
    temp -> addr = last -> addr
    last -> addr = temp;
    return last;
}
```

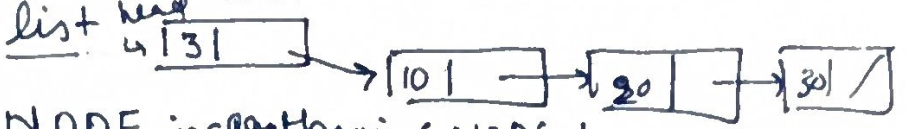
```
NODE insertend (NODE last, int ite
{ NODE temp;
    temp = (NODE) malloc (sizeof(struc
                                node));
    temp -> data = item;
    if (last == NULL) {
        temp -> addr = temp;
        return temp;
    }
    temp -> addr = last -> addr;
    last -> addr = temp;
    return temp;
}
```

DELETION :

```
NODE deletebegin (NODE last) {
    NODE temp;
    if (last == NULL) {
        pf ("List is empty");
        return NULL;
    }
    if (last -> addr == last) {
        pf ("Node deleted is %d", last->data
        free (last);
        return NULL;
}
```

```
    temp = last → addr;
    last → addr = temp → addr;
    pf ("Node deleted is %d", temp → data);
    free (temp);
    return temp;
}
NODE deleteend (NODE last) {
    NODE cur;
    if (last == NULL) {
        pf ("List empty");
        return NULL;
    }
    if (last → addr == last) {
        pf ("Node del is %d", last → data);
        free (last);
        return NULL;
    }
    cur = last → addr;
    while (cur → addr != last)
        cur = cur → data;
    cur → addr = last → addr
    pf ("Node del is %d", last → data);
    free (last);
    return cur;
}

Void display (NODE last) { NODE temp;
    if (last == NULL)
        pf ("List empty");
    else {
        pf ("List contents are:");
        temp = last → addr;
        while (temp != last) {
            pf ("%d", temp → data);
            temp = temp → addr;
        }
        pf ("%d", temp → data);
    }
}
```

HEADER NODE - singly linked




```
NODE insbeginn (NODE head, int item){
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    temp → data = item;
    temp → addr = NULL;
    if (head → addr == NULL) {
        head → addr = temp;
        return head;
    } temp → addr = head → addr; head → addr = temp;
    return head;
}
NODE deleteend (NODE head) {
    NODE cur, prev;
    if (head → addr == NULL) {
        pf (" List is empty");
        return head;
    }
    prev = NULL;
    cur = head → addr;
    while (cur → addr != NULL) {
        prev = cur;
        cur = cur → addr;
    }
    prev → addr = NULL;
    free (cur);
    return head;
}
void display (NODE head) {
    NODE temp;
    if (head → addr == NULL)
        pf ("List empty");
    else {
        temp = head → addr;
        while (temp != NULL) {
            pf ("%d", temp → data);
            temp = temp → addr;
        }
    }
}
```

CIRCULAR SINGLY L.L → HEADER NODE

```
NODE insertbegin (NODE head, int item) {
    NODE temp;
    temp = (NODE) malloc (sizeof(struct node));
    temp → data = item;
    if (head → data == head) {
        temp → addr = head;
        head → addr = temp;
        return head;
    }
    temp → addr = head → addr;
    head → addr = temp;
    return head;
}
NODE insertend (NODE head, int item)
{ NODE temp, cur;
  temp = (NODE) malloc (sizeof(struct node));
  temp → data = item;
  if (head → data == head){
      temp → addr = head;
      head → addr = temp;
      return head;
  }
  cur = head → addr;
  while ( cur → addr != head)
      cur = cur → addr;
  cur → addr = temp;
  temp → addr = head;
  return head;
}
void display (NODE head){
  if (head → addr == head)
      pf ("List empty");
  else { temp = head → addr;
    while(temp != head){
        pf("%d", temp → data);
        temp = temp → addr;
    }
```

ADDITION OF TWO LONG +VE INT:

```
#include<stdio.h> <string.h>
struct node { int data;
              struct node * addr;
}; typedef struct node *NODE;
int main() {
  char first[20], second[20];
  NODE head1, head2; int i;
  head1 = (NODE) malloc(sizeof(struct node));
  head2 = (NODE) malloc(sizeof(struct node));
  pf("Read 1st no");
  sf("%s", first);
  for (i=0; first[i]!='\0'; i++)
      head1 = insertend(head1, first[i]-'\0');
  pf("first no is:");
  display(first);
  pf("Read second no"); sf("%s",second);
  for(i=0; first[i]!='\0'; i++)
      head2 = insertend(head2, second[i]-'\0');
  pf("secon no is:");
  display(second);
  addzero(head1, head2);
  pf("First no. is");
  display(head1);
  pf("second no is");
  display(head2);
  add(head1, head2);
  return 0;
}
void addzero(NODE head1, NODE head2){
  int ct1=1, ct2=1, i;
  NODE t1, t2;
  t1 = head1 → addr;
  while(t1!=head){ ct1=ct1+1;
      t1=t1 → addr;
  }
```

```
t2 = head2 → addr;
while (t2 != head2) {
        ct2 = ct2+1;
        t2 = t2→addr;
}
if (ct1 > ct2){
   for(i=1;i< ct1-ct2;i++)
      head2 = insertbegin(head2,0);
   }
else { for (i=1; i<=ct2-ct1; i++)
      head1 = insertbegin(head1,0);
}}
NODE reverse (NODE head)}
  NODE cur, prev, next;
    cur = head→addr;
    prev = head;
    while(cur != head){
          next = cur→addr;
          cur→addr = prev;
          prev = cur;
          cur = next;
    }
    head→addr = prev;
    return head;
}
NODE add (NODE head1, head2)
{ NODE head, t1, t2;
  int sum=0, carry=0, x;
  head = (NODE)malloc(sizeof(struct
                        node));
    head→addr = head;
    head1 = reverse(head1);
    head2 = reverse(head2);
    t1 = head1→addr;
    t2 = head2→addr;
    while (t1 != head1 && t2 != head2)
  { x = t1→data + t2→data + carry;
     sum = x%10;
     carry = x/10;
     head = insertbegin(head, sum);
     t1 = t1→addr;
     t2 = t2→addr;
  }
  if (carry>0)
      head = insertbegin(head,
                         carry);
    pf("Final added no is");
    display(head);
}
```

DOUBLY LINKED LIST

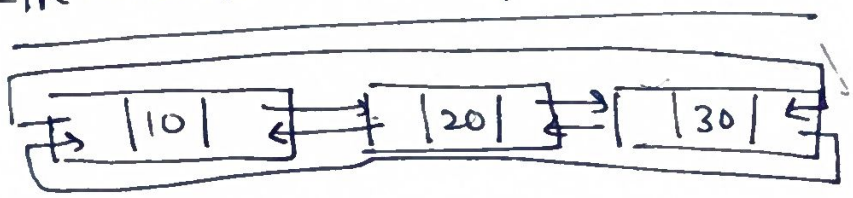
```
struct node{ int data;
        struct node * next;
        struct node * prev;
};typedef struct node * NODE;
NODE insertbegin(NODE start, int
  item){ NODE temp;
    temp=(NODE) malloc(sizeof
                  (struct node));
    temp→data = item;
    temp→next = NULL;
    temp→prev = NULL;
    if (start == NULL)
          return temp;
    temp→next = start
    start→prev = temp;
    return temp;
}
NODE insertend(NODE start, int itm)
{  NODE temp, cur
temp = (NODE) malloc(sizeof(struct
                        node));
temp→data = itm;
 temp→next = NULL;
  temp→prev = NULL;
if (start == NULL)
        return temp;
 cur = start;
```

```
while ( cur -> next!=NULL)
    cur = cur -> next;
cur -> next = temp;
temp -> prev = cur;
return start;
}
```

DELETE

```
NODE deletebegin(NODE start){
    NODE temp;
    if (start == NULL){
        pf("List empty");
        return NULL;
    }
    temp=start;
    start = start -> next;
    start -> prev = NULL;
    pf("Node deleted is %d", temp->data);
    free (temp);
    return start;
}
NODE deleteend (NODE start){
    NODE temp,cur;
    if (start == NULL){
        pf("list empty");
        return NULL;
    }
    cur = start
    while (cur -> next != NULL)
        cur = cur -> next;
    temp = cur -> prev;
    temp -> next = NULL;
    pf("Node deleted is %d",
        cur -> data);
    free(cur);
    return start;
}
void display (NODE start){
    NODE temp;
    if (start == NULL)
        pf ("list empty");
else { pf ("List contents:");
    temp = start;
    while (temp != NULL){
        pf ("%d", temp->
            data);
        temp = temp -> next;
    }
}
}
```

CIRCULAR DOUBLY LINKED LIST




```
NODE insert beg (NODE head,
        int data) {
    NODE temp,cur;
    temp = (NODE) malloc (sizeof(
                struct Node));
    if (head == NULL) {
        head = temp;
        temp -> next = temp;
        temp -> prev = temp; }
    else {
        cur = head -> prev;
        cur -> next = temp;
        temp -> prev = cur;
        temp -> next = head;
        head -> prev = temp;
    }
}
```

SPARSE MATRIX

$$\begin{bmatrix} 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$


```
# include <stdio.h>
struct node{
    int row, col, data;
    struct node * next;
    struct node * prev;
```

```
typedef struct node *NODE;
NODE insertend(NODE start,
int row, int col, int item){
NODE temp, cur;
temp = (NODE) malloc (sizeof (struct
node));
temp->row = row; temp->col = col;
temp->data = item;
temp->prev = NULL;
temp->next = NULL;
if (start == NULL)
        return temp;
cur = start;
while(cur->next != NULL)
        cur = cur->next;
cur->next = temp;
temp->prev = cur;
return start;
}
void display (NODE start){
NODE temp;
if (start == NULL)
    pf ("List is empty");
else{
    pf ("In Row \t Col \t Data");
    temp = start;
    while (temp != NULL){
        pf ("%d \t %d \t %d \n",
        temp->row, temp->col,
        temp->data);
        temp = temp->next;
    }
}
}
void displaymatrix (NODE start,
int m, int n) { int i, j;
NODE temp;
temp = start;
for (i=1; i<=m; i++){
    for (j=1; j<=n; j++)
    {
    if (temp != NULL && temp->row == i
    && temp->col == j){
    pf ("%d", temp->data);
    temp = temp->next;
    }
    else
        printf ("0 \t");
    }
    pf ("\n");
}
}
int main() { NODE start = NULL;
int m, n, i, j, item;
pf ("Read order of matrix");
sf ("%d %d", &m, &n);
pf ("Read matrix");
for (i=1; i<=m; i++){
    for (j=1; j<=n; j++){
        sf ("%d", &item);
        if (item != 0)
            start = insertbegin
            (start, i, j, item);
    }
}
display (start);
displaymatrix (start, m, n);
return 0;
}
```

TREE

refer pg 58, 59, 60.

BINARY TREE TRAVERSAL:

1. Preorder
2. Inorder
3. Postorder

Preorder: Root Left Right

```
void preorder(NODE root){
    if (root != NULL){
        pf("%d", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```

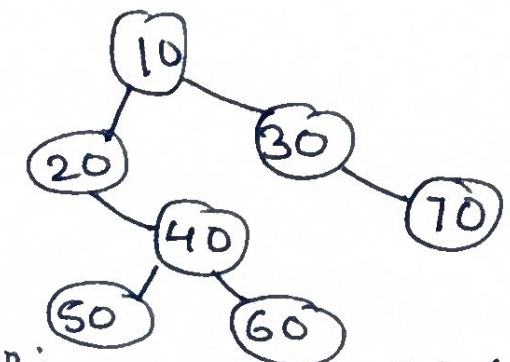
Inorder: Left Root right

```
void inorder(NODE root){
    if (root != NULL){
        pf("%d",
        inorder(root->left);
        pf("%d", root->data);
        inorder(root->right);
    }
}
```

PostOrder: Left Right Root

```
void postorder(NODE root){
    if (root != NULL){
        preorder(root->left);
        preorder(root->right);
        pf("%d", root->data);
    }
}
```

Ex.



Preorder: 10 20 40 50 60 30 70

Inorder: 20 50 40 60 10 30 70

Postorder: 50 60 40 20 70 30 10

ITERATIVE FUNC TO PREFORM PREORDER TRAVERSAL

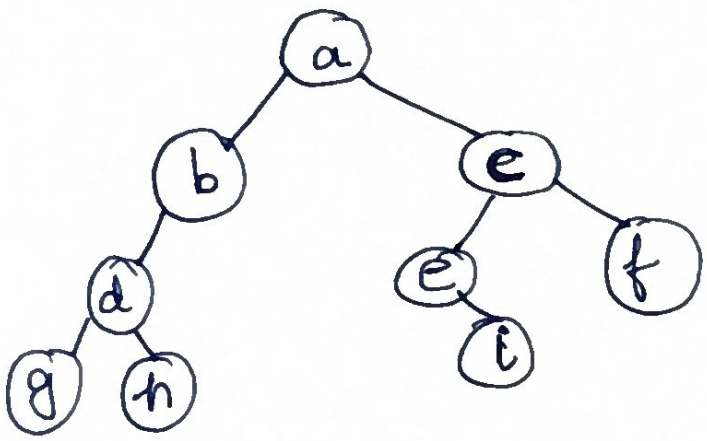
```
void preorder_iter(NODE root){
    NODE cur, s[10]; int top=-1;
    if (root == NULL){
        pf("No Traversal");
        return;
    }
    cur = root;
    while (1){
        while (cur != NULL){
            pf("%d", cur->data);
            s[++top] = cur;
            cur = cur->left;
        }
        if (top != -1){
            cur = s[top--];
            cur = cur->right;
        }
        else
            return;
    }
}
```

```
void Inorder_iter( " ){
    .
    .
    .
    while (cur != NULL){
        s[++top] = cur;
    .
    .
    if (top != -1){
        cur = s[top--];
        pf("%d \t", cur->data);
        cur = cur->right;
    }
void Postorder_iter( " )
    .
    .
    if (top != -1){
        cur = s[top--];
        cur = cur->right;
    }
```

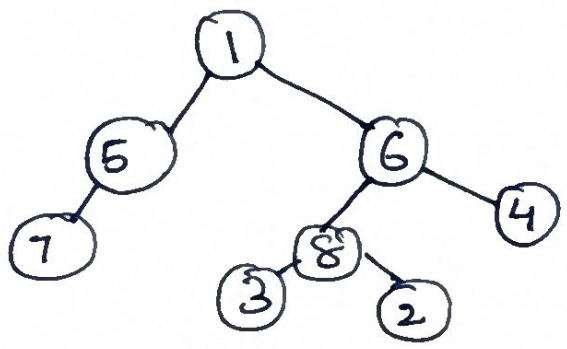
```
    }
    else
        pf("%d.dlt", cur->data);
  }
}
```

CONSTRUCTION OF BINARY TREE FROM TRAVERSALS.

Preorder : a b d g h c e i f
Inorder : g d h b a e i c f



Postorder : 7 5 3 2 8 4 6 1
Inorder : 7 5 1 3 8 2 6 4



BINARY SEARCH TREE :

```
NODE createBST(NODE root, int item){
  NODE temp, cur, prev;
  temp = (NODE)malloc(sizeof(struct node));
  temp->data = item;
  temp->left = NULL;
  temp->right = NULL;
  if (root == NULL)
      return temp;
  prev = NULL;
  cur = root;
  while (cur != NULL){
      prev = cur;
      if (item < cur->data)
          cur = cur->left;
      else
          cur = cur->right;
  }
  if (prev->data > item)
      prev->left = item;
  else prev->right = temp;
  return root;
}

NODE deletenode(NODE root,
  int key) {
  NODE temp;
  if (root == NULL)
      return root;
  if (key < root->data)
      root->left = deletenode
          (root->left);
  else if (key > root->data)
      root->right = deletenode(
          root->right, key);
  else {
      if (root->left == NULL) {
          temp = root->right;
          free(root);
          return temp;
      }
      if (root->right == NULL) {
          temp = root->left;
          free(root);
          return temp;
      }
      temp = inordersuccessor(
          root->right);
      root->data = temp->data;
      root->right = deletenode
          (root->right, temp->data);
  }
  return root;
}
NODE inordersuccessor(NODE root){
  NODE cur = root;
  while(cur->left != NULL)
      cur = cur->left;
  return cur;
}
```

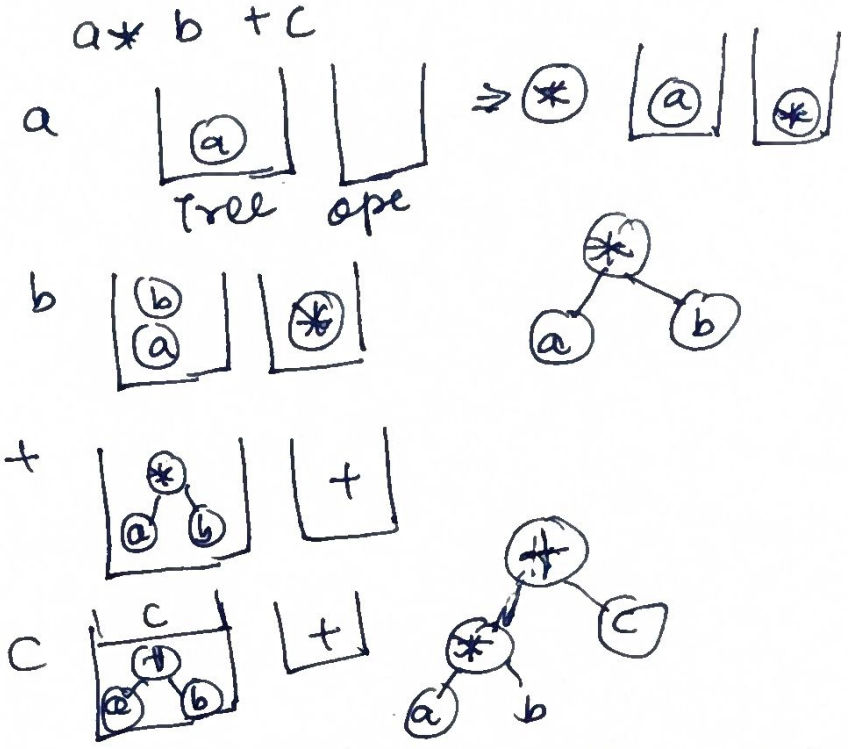
```
#include <stdio.h>
struct node{
    int data;
    struct node * right;
    struct node * left;
};typedef struct node * NODE;
int main(){
 NODE root = NULL;
 int ch, item, key;
 for(;;){
   pf("1. Insert"); 2. Delete;
   3 Preorder; 4 Inorder;
   5. Postorder; 6. Exit;
  pf("Read choice:");
  sf("%d", &ch);
  switch(ch){
    case 1: pf("Element ins:")
      sf("%d", &item);
      root = createBST(root, item)
      break;
  case 2: pf("Node to delete");
    sf("%d", &key);
    root = deletenode(root, key);
    break;
  case 3: pf("Preorder");
     preorder(root);
     break;
   :
  } return 0;
}
```

EXPRESSION TREE:

Algo to cons binary tree from infix exp:

1. Initialize 2 stacks.
2. Scan infix exp → left to right
3. Scanned symbol:
   i) operand: construct Node, push
   ii) Operator: i) if stack empty push node onto stack.
   else ii) pop node from stack.
4) until operator not empty, pop nodes.
5) Final tree will be on top of tree stack

a* b +c

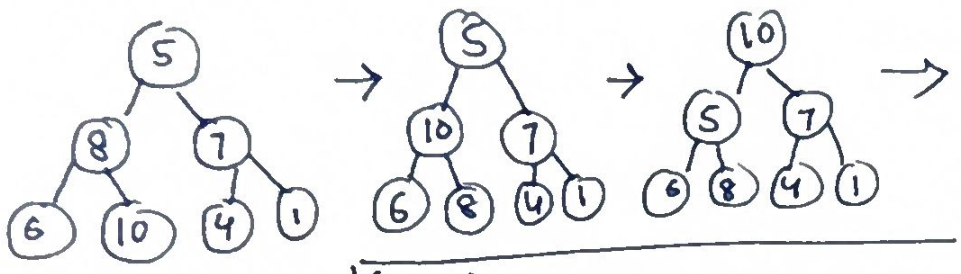


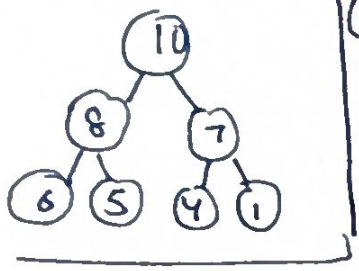

```
#include<stdio.h> <stdlib.h><ctype.h>
struct node{ char info;
    struct node * left;
    struct node * right;
}; typedef struct nodo * NODE;
struct Stack{ int top;
        NODE data[10];
}; typedef struct stack STACK;
int preced(char item){
    switch(item){
      case '^': return 5;
      case '*':
      case '/': return 3;
       case '+':
       case '-': return 1;
}   }
```

```
void preorder(NODE root){
    if (root != NULL){
        pf("%d", root->info);
        preorder(root->left);
        preorder(root->right);
    }
}

void inorder(NODE root){
    if (root != NULL){
        inorder(root->left);
        pf("%c", root->info);
        inorder(root->right);
    }
}

void postorder(NODE root){
    if (root != NULL){
        postorder(root->left);
        postorder(root->right);
        pf("%c", root->info);
    }
}

NODE createnode(char item){
    NODE temp;
    temp = (NODE) malloc(sizeof(
        struct node));
    temp->info = item;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

NODE createExpTree(char expr[20])
{   STACK tree, Op;
    Tree.top = -1;
    Op.top = -1;
    char symbol;
    int i;
    NODE temp, t, l, r;
    for(i=0; infix[i] != '\0'; i++){
        symbol = infix[i];
        temp = createnode(symbol);
        if(isalnum(symbol))
            push(&Tree, temp);
        else{ if(op.top == -1)
                push(&op, temp);
            else {
                while(op.top != -1 &&
                preced((op.data
                [op.top])->info) >=
                preced(symbol)) {
                    t = pop(&op);
                    r = pop(&Tree);
                    l = pop(&Tree);
                    t->right = r;
                    t->left = l;
                    push(&Tree, t);
                }
                return pop(&Tree); }
        }
    }
}

void push(STACK *S, NODE temp)
  { S->data[++(S->top)] = temp; }

NODE pop(STACK *S){
    return S->data[(S->top)--];
}

int main(){ NODE root = NULL;
char expr[20]; pf("\nRead exp:");
sf("%s", expr);
root = createExpTree(expr);
pf("\nInorder"); inorder(root);
pf("\nPreorder"); preorder(root);
pf("\nPostorder"); postorder(root);
    return 0; }
```

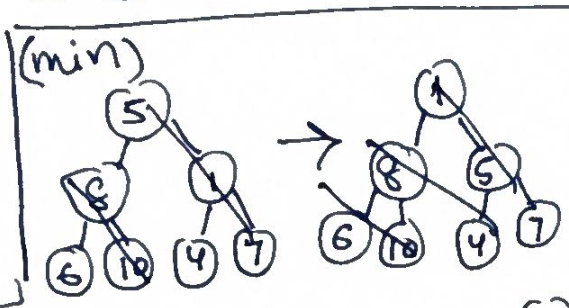
HEAP → pg 75 1> bottom-up-method (max)

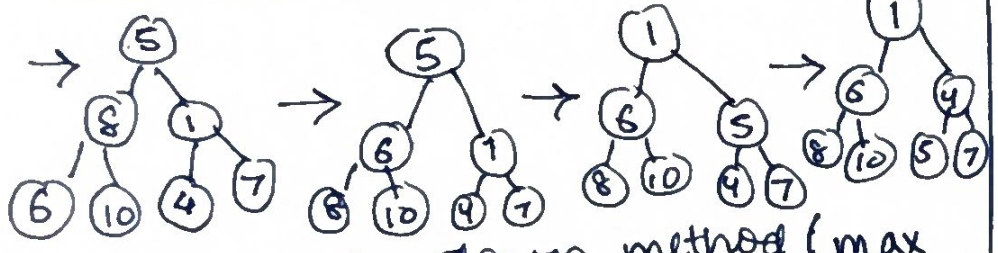
5 8 7 6 10 4 1





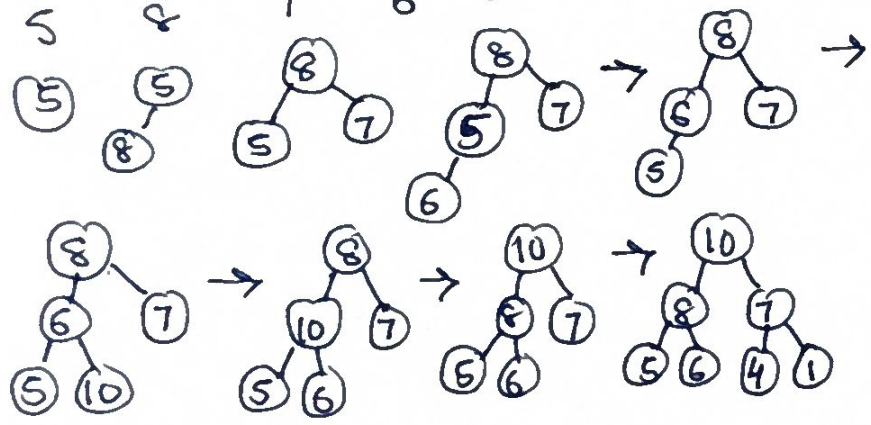
(min)



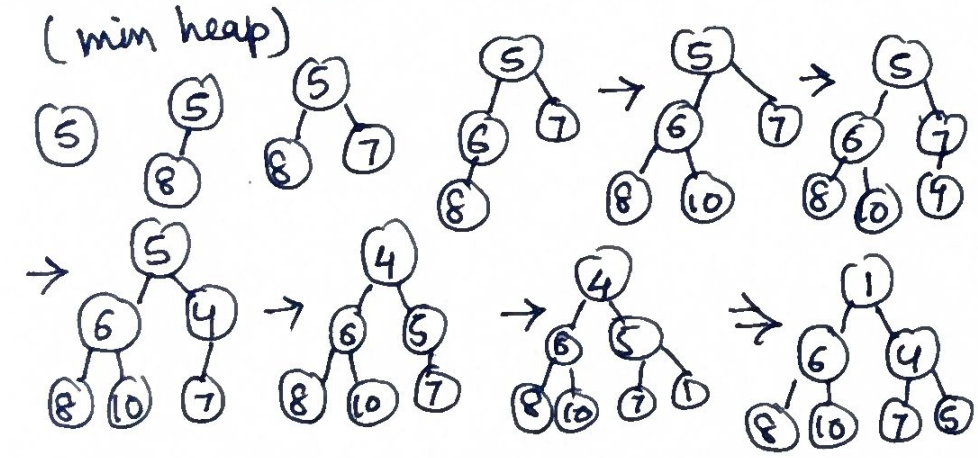


2> Top - ~~botter~~ down method (max heap)

5 8 7 6 10 4 1



(min heap)



PRIORITY QUEUE USING HEAP:

```
# include<stdio.h>
void heapify (int a[10], int n){
   int i,k,j,v,flag; for(i=n/2;i>=1;i--)
      { k=i;  v=a[k];
        flag=0;
        while (flag!=NULL && 2*k<=n){
           { j=2*k;
             if (j<n) {
               if (a[j]<a[j+1])
                    j=j+1;
             }
             if (v>=a[j])
                  flag=1;
             else{ a[k]=a[j];
                   k=j;
                 }
           } a[k]=v;
      }
}
int main(){ int i,n,ch,a[10];
for(;;){ pf("1.create heap"); 2. Delete;
  3. Exit ; pf("Read choice: ");
  sf("%d",&ch);
  switch(ch){
     case 1: pf("Read no of elements:");
             sf("%d",&n);
             pf("Read elements : ");
             for(i=1;i<=n;i++)
                  sf("%d",a[i]);
             heapify(a,n);
             pf("Elements after heapify");
             for(i=1;i<=n;i++)
                  pf("%d ",a[i]);
             break;
     case 2: if(n>=1){
             pf("Element del is %d",a[1]);
             a[1]=a[n];
             n=n-1
             heapify(a,n);
             pf("Elements after deletion");
             for(i=1;i<=n;i++)
                  pf("%d",a[i]);
     }
     else
        pf("No element for deletion");
        break;
default: exit(0);
     }
  }
return 0;
}
```