# BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
## YELAHANKA – BANGALORE – 64
## Department of Computer Science and Engineering

**PREPORATORY EXAMINATION, DECEMBER–2018(CBCS)**

**Subject: Data Structure and Application** | **Subject Code: 17CS33** | **Semester : III (diploma)**
**Max. Marks : 80** | **Date: 12-12-2018** | **Time: 1.00 AM to 4.00 PM**

*Answer FIVE **full** questions, choosing one full question from each module.*

| | | |
|---|---|---|
| | **MODULE-I** | |
| 1 | A. What is Data structure? Give its classification.<br>B. List and explain the functions supported by C for Dynamic memory allocation.<br>C. Write a C-program with an appropriate structure definition and variable declaration to read and display information about 5 students using nested structures. Consider the following fields like Sname,Sid,DOJ (Date,Month,Year). | **4+6+6 m** |
| | **OR** | |
| 2 | A. Write a program to (i) Reverse a given string (ii) Concatenate two strings. Without using any library functions.<br>B. What is a polynomial? What is a degree of polynomial? Write a program function to add two polynomials. | **8+8 m** |
| | | |
| | **MODULE-II** | |
| 3 | A. Define stack. Implement push and pop for stack using arrays.<br>B. Write a C program to evaluate postfix expression. | **8+8 m** |
| | **OR** | |
| 4 | A. Describe how you could model a **maze** where 0 represents open paths and 1 represents barriers. What moves are permitted in the matrix model? Provide an example MAZE together with its allowable moves and table of moves.<br>B. Explain with a suitable example, how would you implement circular queue using dynamically allocated arrays. | **8+8 m** |
| | | |
| | **MODULE-III** | |
| 5 | A. Write a function for singly linked lists with integer data to search an element in the list that is unsorted and a list that is sorted.<br>B. Given two singly linked lists, LIST-1 and LIST-2, Write an algorithm/function to form a new list LSIT-3 using concatenation of the lists LIST-1 and LIST-2. | **8+8 m** |
| | **OR** | |
| 6 | A. Write the node structure for representing Singly Linked List (SLL). Also, Write C functions<br>I. To insert/Delete the node from the rear in the SLL<br>II. Display the nodes in the SLL.<br>B. List out the difference between the doubly linked list and singly linked list illustrate with an example the following operations on a doubly linked list<br>I. Inserting a node at the beginning.<br>II. Inserting at a the intermediate position.<br>III. Deletion on a node with a given value.<br>IV. Search a key element. | **8+8 m** |
| | | |
| | **MODULE-IV** | |
| 7 | A. With a neat illustration of BST, explain the C function to search an element in a BST.<br>B. Define a binary search tree. Draw the binary search tree (BST) for the following sequence of elements, 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9. | **8+8 m** |
| | **OR** | |
| 8 | A. Draw a binary tree for the following set of numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9. Trace the above generated tree using inorder, preorder and postorder. Also write the C functions for each.<br>B. What is the advantage of threaded binary tree over binary tree? Explain the construction of threaded binary tree for 10, 20, 30, 40, 50. | **8+8 m** |
| | | |
| | **MODULE-V** | |
| 9 | A. Write a C function for insertion sort. Sort the following list using insertion sort: 50, 30, 10, 70, 40, 20, 60.<br>B. What is collision? What are the methods to resolve collision? Explain linear probing with an example. | **8+8 m** |
| | | |
| 10 | A. What do you understand by the term file organization? Briefly summarize any three widely used file organization techniques.<br>B. State and explain WARSHALL's Algorithm with an example. | **8+8 m** |

# MODULE-1

① A. What is data structure? Give its classification

The logical or mathematical model of a particular organization of data is called a data structure.

Data Structure is classified into two types:-

→ Linear data structure

→ Non Linear data structure

Linear data structure:

A linear data structure traverses the data elements sequantially, in which only one data element can directly be reached.

Ex:- Arrays, Linked list, Stacks & Queues.

Non Linear data structure:

Every data item is attached to several other data items in an way that is specific for reflecting a relationships. The data items are not arranged in sequential structure

Ex: Trees, Graphs

B. List & Explain the functions supported by C for Dynamic memory allocation

-> The exact size of array is unknown until the compile time.
The size of array declared initially can sometimes be insufficient & some times more than required.

-> Dynamic memory allocation allows a program to obtain more memory space, while running or to release space when no space is required.

-> C provides four library functions under "stdlib.h" for dynamic memory allocation

malloc - void * malloc (size_t size)
calloc - void * calloc (size_t num, size_t size)
realloc - void * realloc (void * ptr, size_t size)
free - void * free (void * ptr)

C. write a C-program with an appropriate structure definition & variable declaration to read & display information about 5 students using nested structures. Consider the following fields like Sname, Sid, DOJ (Date, Month, Year)

```
#include <stdio.h>
#include <conio.h>
Struct stud_dob
{
    int day;
    int month;
    int year;
};
Struct student
{
    int sid;
    char sname[];
    Struct student_dob date;
};
void main()
{
    Struct student stud1;
    printf("enter the student number\n");
    scanf("%d", &stud1.sid);
    printf("enter the student name\n");
    scanf("%s", stud1.sname);
    printf("enter date of birth as day, month, year");
    scanf("%d %d %d", &stud1.date.day,
        &stud1.date.month, &stud1.date.year);
```

```
printf ("Student details are \n");
printf ("Student number is %d \n", stud1.sid);
printf ("Student name is %d \n", stud1.name);
printf ("Student dob is %d | %d | %d \n",
stud1.date.day, stud1.date.month, stud1.date.year);
getch();
}
```

(2)

A. write a program to (i) Reverse a given string (ii) concatenate two strings. without using any library functions

(i)
```
#include <stdio.h>
int main()
{
    char str[100], revstr[100];
    int i, j;
    printf ("Enter a string: ");
    scanf ("%s", str);
    j=0;
    for (i=(strlen (str)-1); i>=0; i--)
        revstr[j++] = str[i];
    revstr[j] = '\0';
```

```
    printf("\n original string is: %s", str);
    printf("\n reversed string is: %s", rev str);
}
```

(ii)

```
#include <stdio.h>
int main()
{
    char s1[100], s2[100], i, j;
    printf("Enter first string: ");
    scanf("%s", s1);
    printf("Enter second string: ");
    scanf("%s", s2);
    for (i=0; s1[i] != '\0'; ++i);
    for (j=0; s2[j] != '\0'; ++j, ++i)
    {
        s1[i] = s2[j];
    }
    s1[i] = '\0';
    printf("After concatenation: %s", s1);
}
```

B. what is a polynomial? what is a degree of polynomial? write a program function to add two polynomials.

The polynomial is nothing but the sum of the terms.

where each term $a x^{e}$ ; e → exponents

a → co-efficient, x → variable

The largest exponent of a polynomial is called degree of the polynomial

The addition of polynomials

$$A(x) = \sum a_i x^i$$

$$B(x) = \sum b_i x^i$$

$$A(x) + B(x) = \sum (a_i + b_i) x^i$$

Ex: $A(x) = 4x^{16} + 15x^2 + 4$

$B(x) = 19x^{10} + 8x$

| | Sa | | fa | Sa | fb | |
|---|---|---|---|---|---|---|
| Co-eff | 4 | 15 | 4 | 19 | 8 | |
| exp | 16 | 2 | 0 | 10 | 1 | |
| | 0 | 1 | 2 | 3 | 4 | 5 6 |

Sa = Start A
Sb = Start B
Fa = finish A
Fb = finish B

```
Void padd (int StartA, int finishA, int StartB,
           finishB, int * StartD, int * finishD)
{
    float coefficient;  * StartD = avail;
    while (StartA <= finishA && StartB <= finishB)
        switch (COMPARE (terms [StartA]. expon,
                         terms [StartB]. expon))
        {
            case -1: attach (terms [StartB]. coef,
                             terms [StartB]. expon);
                     StartB ++;
                     break;
            case 0: coefficient = terms [StartA]. coef +
                                  terms [StartB]. coef;
                    if (coefficient)
                        attach (coefficient, terms [StartA]
                                . expon;
                    StartA ++;
                    StartB ++;
                    break;
```

```
            case 1: attach(terms[startA].coef,
                           terms[startA].expon);
                    startA++;
                    break;
            }
        for (; startA <= finishA; startA++)
            attach(terms[startA].coef, terms[startA].expon);
        for (; startB <= finishB; startB++)
            attach(terms[startB].coef, terms[startB].expon);
            *finishD = Avail-1;
}
void attach(float coefficient, int exponent)
{
    if (avail >= MAX_TERMS)
    {
        printf("Too many terms in the polynomial\n");
        exit(EXIT_FAILURE);
    }
    terms[avail].coef = coefficient;
    terms[avail++].expon = exponent;
}
```

3. A. Define stack. Implement push and pop for stack using arrays.

## MODULE - II

A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. Stacks are sometime known as LIFO (Last in first out) lists.

As the items can be added or removed only from the top i.e., the last item to be added to a stack is the first item to be removed.

The two basic operations associated with stacks are:

- Push: Is the term used to insert an element onto a stack.
- Pop: Is the term used to delete an element onto a stack.

→
```
Void pop()
{
    if(top==0)
    {
        Printf("\n\n Stack underflow...");
        return;
    }
    else
    {
        Printf(" Poped element is: %d", Stack[top]);
    }
}
```

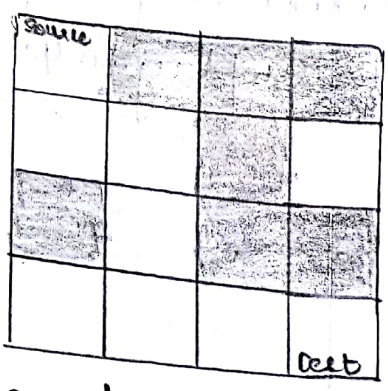
```
Void Push()
{
    int data;
    if(top == MAX)
    {
        Printf("\n\n Stack Overflow");
        return;
    }
    else
    {
        Printf("\n\n Enter data:");
        Scanf("%d", &data);
        Stack[top] = data;
        top = top+1;
        Printf("\n\n Data pushed into the stack");
    }
}
```

8. Write a C program to evaluate postfix expression.

```
#include<stdio.h>
#include<math.h>
#define MAX 20
int isoperator(char ch)
{
    if(ch=='+'||ch=='-'||ch=='*'||ch=='/'||ch=='^')
        return 1;
    else
        return 0;
}
Void main()
{
    Char postfix[MAX];
    int val;
    Char ch;
    int i=0, top=0;
    float Val_stack[MAX], Val1, Val2, res;
    clrscr();
    Printf("\n Enter a postfix expression:");
    Scanf("%s", postfix);
    while((ch=postfix[i])!='\0')
    {
        if(isoperator(ch)==1)
        {
            Val2=Val_stack[--top];
            Val1=Val_stack[--top];
            Switch(ch)
            {
                Case'+':
                    res=Val1+Val2;
                    break;
```

```
            case '-':
                    res = val1 + val2;
                    break;
            Case '*':
                    res = val1 * val2;
                    break;
            Case '/':
                    res = val1 / val2;
                    break;
            Case '^':
                    res = pow(val1, val2);
                    break;
            }
            Val_stack[top] = res;
    }
    else
        Val_stack[top] = ch - 48;
    top++;
    i++;
}
Printf("\n Value of %s is : %f", postfix, Val_stack[0]);
getch();
}
```

4A. Describe how could model a maze where 0 represents open paths and 1 represents barriers. What moves are permitted in the matrix model? Provide an example maze together with its allowable moves and table of moves.




Gray blocks are dead ends (value = 0)

following is binary matrix representation of the maze

{ 1, 0, 0, 0 }
{ 1, 1, 0, 1 }
{ 0, 1, 0, 0 }
{ 1, 1, 1, 1 }

⇒ Algorithms to generate all paths from Source to destination and one by one check if the generated path satisfies the constraints

```
While there are untried paths
{
    generate the next path.
    if this path has all blocks as 1
    {
        Print this path;
    }
}
```

```
// C program to solve a maze problem
#include <stdio.h>
#define N 4
Void solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]);
Void printSolution(int sol[N][N])
{
    for(int i=0; i<N; i++)
    {
        for(int j=0; j<n; j++)
        Printf("%d", sol[i][j]);
    }
}
/* function to check if x, y is valid index for N*N. */
Void isSafe(int maze[N][N], int x, int y)
{
    if(x>=0 && x<N && y>=0 && y<N && maze[x][y]==1)
      return true;
    return false;
}
/* this function solves the maze problem using Backtracking.
It mainly uses solveMazeUtil() to solve problem. It returns
false if no path is possible, otherwise return true and
prints the path in the form of 1s. */
Void solveMaze(int maze[N][N])
{
    int sol[N][N] = { { 0,0,0,0 },
                      { 0,0,0,0 },
                      { 0,0,0,0 },
                      { 0,0,0,0 },
                    };
```

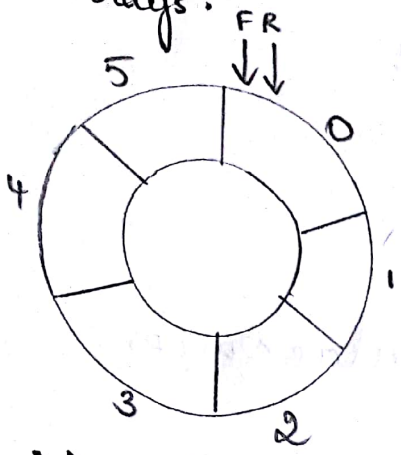
```
    if(SolveMazeUtil(maze, 0, 0, sol) == false)
    {
        Printf("Solution does not exit");
        return false;
    }
    Print solution(sol);
    return true;
}
void SolveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])
{
    if (x == N-1 && y == N-1)
    {
        sol[x][y] = 1;
        return true;
    }
    if (isSafe(maze, x, y) == true)
    {
        sol[x][y] = 1;
        if(SolveMazeUtil(maze, x+1, y, sol) == true)
            return true;
        if(SolveMazeUtil(maze, x, y+1, sol) == true)
            return true;
        sol[x][y] = 0;
        return false;
    }
    return false;
}
// driver program to test above function
int main()
{
    int maze[N][N] = {{1, 0, 0, 0},
                      {1, 1, 0, 1},
                      {0, 1, 0, 0},
                      {1, 1, 1, 1}};
```

```
    SolveMaze(maze);
    return 0;
}
```

Output:- The 1 values show the path.

```
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1
```

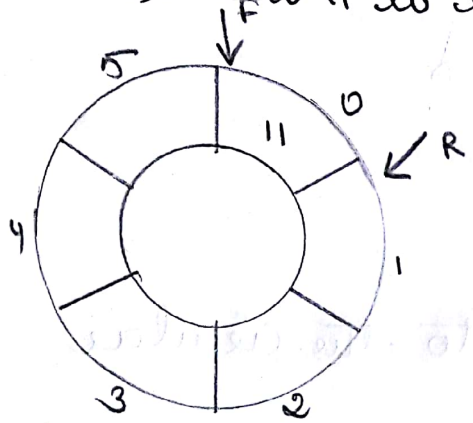
B. Explain with a suitable example, how would you implement circular queue with dynamically allocated arrays.




Let us consider a circular queue, which can hold maximum [MAX] of 6 elements. Initially the queue is empty.

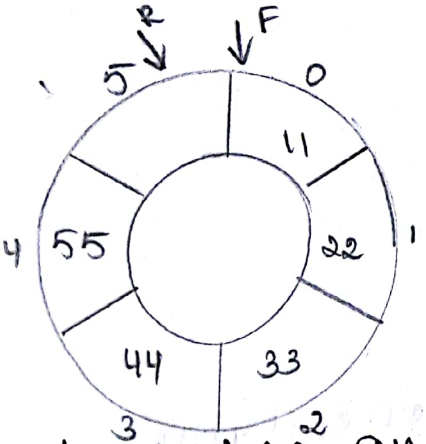
Queue empty
MAX = 6
FRONT = REAR = 0
COUNT = 0

Now, insert 11 to the circular queue. Then it will be



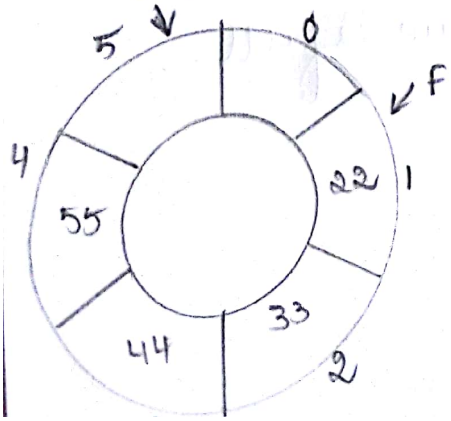

FRONT = 0
REAR = (REAR + 1) % 6 = 1
COUNT = 1

Insert new elements 22, 33, 44 and 55 into the circular queue.




FRONT = 0
REAR = (REAR + 1) % 6 = 5
COUNT = 5.

Now, delete an element. The element deleted is the element at the front of the circular queue. So, 11 is deleted.

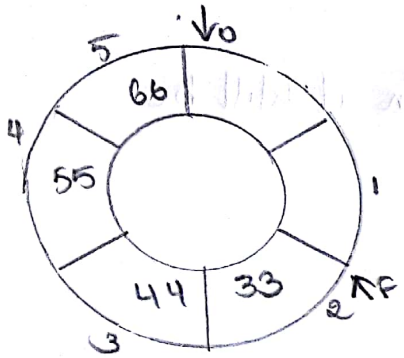



FRONT = (FRONT + 1) % 6 = 1
REAR = 5
COUNT = COUNT - 1 = 4

Again, delete an element, The element to be deleted is always Pointed to by the FRONT pointer. So, 22 is deleted.

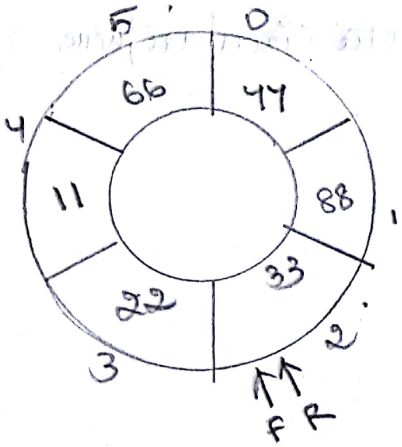

FRONT = (FRONT+1) % 6 = 2
REAR = 5
COUNT = COUNT - 1 = 3

Again, insert another element 66 to the Circular Queue




FRONT = 2
REAR = (REAR+1) % 6 = 0
COUNT = COUNT + 1 = 4

Now, insert new elements 77 and 88 into the Circular Queue. The Circular Queue is




FRONT = 2, REAR = 2
REAR = REAR % 6 = 2
COUNT = 6

Now, if we insert an element to the Circular Queue as COUNT = MAX we cannot add the element to the Circular Queue. So, the Circular Queue is full.

> Implementation of Circular Queue, using array.

```
#include<stdio.h>
#include<conio.h>
#define MAX 6
int CQ[MAX];
int front=0;
int rear=0;
int count=0;
void insertCQ()
{
    int data;
    if(count==MAX)
    {
        Printf("\n Circular Queue is full");
    }
    else
    {
        Printf("\n Enter data:");
        Scanf("%d",&data);
        CQ[rear]=data;
        rear=(rear+1)%MAX;
        count++;
        Printf("\n Data Inserted in the circularQueue");
    }
}
void deleteCQ()
{
    if(count==0)
    {
        Printf("\n\n CircularQueue is empty");
    }
    else
```

```
    {
        Printf("\n Deleted element from circular Queue is
%d", CQ[front]);
        front=(front+1) % MAX;
        count--;
    }
}
void displayCQ()
{
    int i, j;
    if(count==0)
    {
      Printf("\n\n Circular Queue is Empty");
    }
    else
    {
        Printf("\n Elements in circular Queue are:");
        j=count;
        for(i=front; j!=0; j--)
        {
            Printf("%d\t", CQ[i]);
            i=(i+1) % MAX;
        }
    }
}
```

```
int menu()
{
    int ch;
    clscr();
    Printf("\n\t Circular Queue operations using array...");
    Printf("\n 1. Insert\n 2. delete\n 3. display\n 4. Quit");
    Printf("\n Enter your choice:");
    Scanf("%d", &ch);
    return ch;
}
Void main()
{
    int ch;
    do
    {
      ch = menu();
      switch(ch)
       {
            case 1: insertCQ();
                    break;
            case 2:
                    deleteCQ();
                    break;
            case 3:
                    displayCQ();
                    break;
            case 4: return;
            default:
                    Printf("\n Enter valid choice");
       }
      getch();
    } while(1);
```

# Module-3

5) A) write a function for Singly linked lists with integer data to search an element in the list that is unsorted and a list that is sorted?

Ans:-

```
void Search_Key()
{
    int key;
    node *new1, *prev, *temp=start;
    if(start==NULL)
    {
        printf("empty\n");
        return;
    }
    printf("enter the key\n");
    Scanf("%d",&key);
    while(temp!=NULL && temp->info!=key)
    {
        prev=temp;
        temp=temp->next;
    }
    if(temp==NULL)
    {
        printf("key not found\n");
        return;
    }
    printf("key found\n");
}
```

B) 02 Given two Singly linked list, LIST-1 and LIST-2 write, an algorithm/function to form a new list LIST-3 using concatenation of the lists LIST-1 and LIST-2

Ans:/* Assuming two linked lists are created with start1 pointing to 1st list and start2 pointing to 2nd list */

// C function

```
node * concat(node *start1, node *start2)
{
    node *temp = start1;
    if (start1 == NULL)
        return(start2);
    if (start2 == NULL)
        return(start1);
    if (start1 == NULL && start2 == NULL)
    {
        printf("list is empty\n");
        return;
    }
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = Start2;
    return(Start1);
}
```

6] A write the node structure for representing Singly
A] Linked List (SLL). Also, write C function
I. To insert/delete the node from the rear in the SLL
II. Display the nodes in the SLL

03

Ans:-

```
#include<stdio.h>
#include<alloc.h>
struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;
node *start;
// C'Fuction for insert node at rear
void insert_rear()
{
    node *n1, *temp=start;
    n1=getnodenode();
    if(start==NULL)
    {
        start=n1;
        return;
    }
    while(temp->link!=NULL)
        temp=temp->link;
    temp->link=n1;
}
```

// C function for delete node at rear



```
void delete_end()
{
    node *temp = start, *prev;
    if (start == NULL)
    {
        printf("\n Empty list\n");
        return;
    }
    if (start->link == NULL)
    {
        printf("\n %d\t is deleted", start->info);
        free(start);
        start = NULL; return;
    }
    while (temp->next != NULL)
    {
        prev = temp;
        temp = temp->link;
    }
    prev->link = NULL;
    printf("the deleted node is %d\t", temp->info);
    free(temp);
}
```

11. Display the nodes in SLL

```
void display()
{
    node *temp=start;
    if (start==NULL)
    {
        printf("\n Empty list\n");
        return;
    }
    printf("\n the details are:\n");
    while (temp!=NULL)
    {
        printf("\n%d\t", temp->info);
        temp=temp->next;
    }
}
```

05

B) List out the difference b/w doubly linked list & Singly linked list(SLL) ~~Also write & functions~~ illustrate with an example the following operations on a doubly linked list.

1. Inserting a node at the beginning
2. Inserting at the intermediate position
3. Deletion on a node with given value
4. Search a key element.

## Singly L.L

* A Singly LL is a linked list where the node contains some data and pointer to the next node in the list
* It allows traversal only in one way
* It uses less memory per node (single pointer)
* complexity of insertion & deletion at a known position is O(n)
* Singly LL can mostly be used for stacks

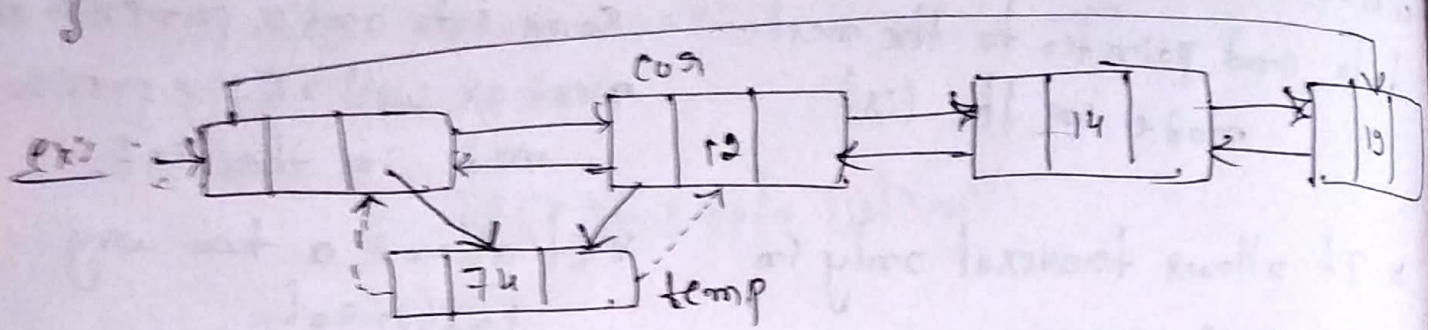
## D.LL

* DLL is a linked list where the node contains some data and a pointer to the next as well as the previous node in the list.
* It allows a two way traversal.
* It uses more memory per node (two pointers).
* complexity of insertion & deletion at a known position is O(1)
* they can be used to implement stacks, heaps & binary trees.

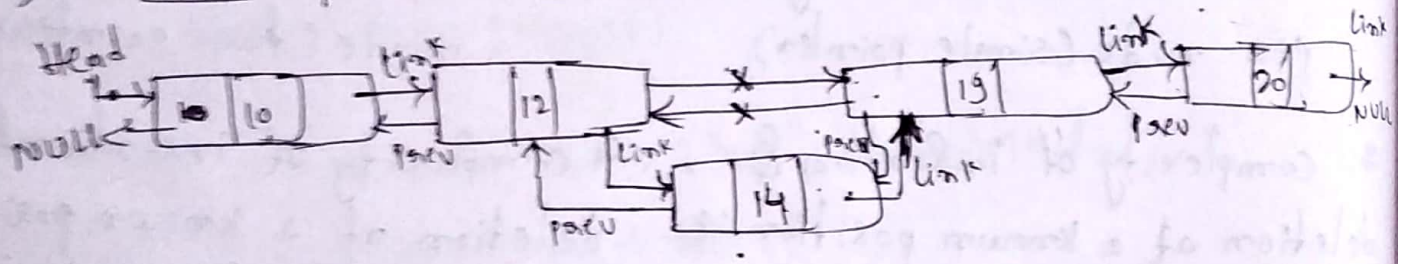
### 1. Insert node at the beginning in DLL:-

```
void insert_front()
{
    node *n;
    n = getnodeDLL();
    if(start == NULL)
    {
        start = n;
        return;
```

```
n->next = start;
start->prev = n;
start = n;
}
```




5) Inserting at intermediate position:-




```
Void insertAtMid (Node *head)
{
    Node *ptr=head;
    Node *newnode=NULL;
    Node *newnode2=head;
/*  printf("Enter node to be inserted:\n");
    scanf("%d", &num);  */
    if(head==NULL)
    {
        newnode=head;
```

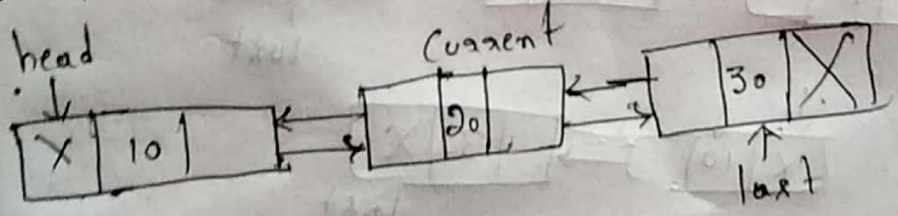
```
newnode = (Node *) malloc (sizeof (Node));
```

OR

```
newnode -> x = num;
newnode -> next = NULL;
newnode -> prev = NULL;
}
else
{
    pta = head -> prx;
    while (pta -> x != (count/2))
    {
        pta = pta -> next;
    }
    newnode -> next = pta -> next;
    newnode -> prev = pta;
    pta -> next -> prev = newnode;
    pta -> next = newnode;
}
}
```

3. delete on a node with given value :-

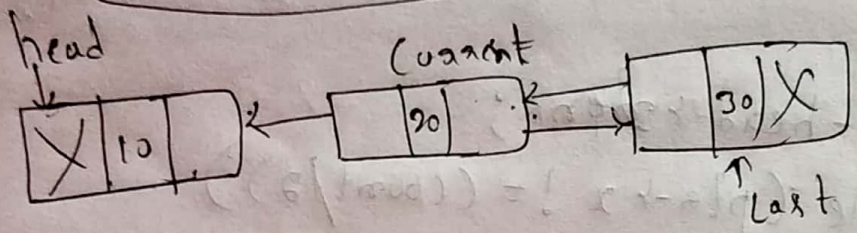
1) Traverse to $N^{th}$ node of the LL, let's say a pointer current points to $N^{th}$ node in our case 2 node.



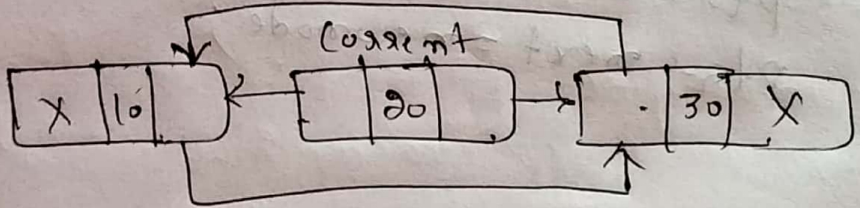

2)

2) link the node behind current node with the node ahead of current node, which means now, the N-1th node will point to N+1th node of the list. which can be implemented as

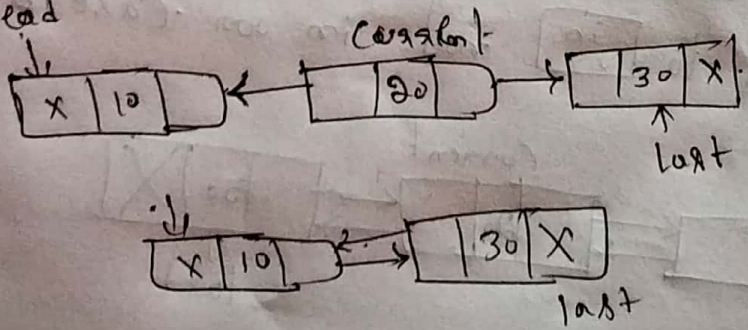
current->prev->next = current->next




3) If N+1th node is not NULL. the link the N+1th node with N-1th node. i.e now the previous address field of N+1th node will point to N-1th node. which can be implemented as current->next->prev = current->prev




4) finally delete the current node from memory and you are done.



4. search a key element

```
void search (int data)
{
    int pos=0;
    if(head == NULL)
    {
        printf("linked list not initialized\n");
        return;
    }
    current = head;
    while (current != NULL)
    {
        pos++;
        if(current->data == data)
        {
            printf("%d found \n", data, pos);
                                at %d
            return;
        }
        if(current->next != NULL)
        current=current->next;
        else
            break;
    }
    printf("%d il not exist\n", data);
}
```

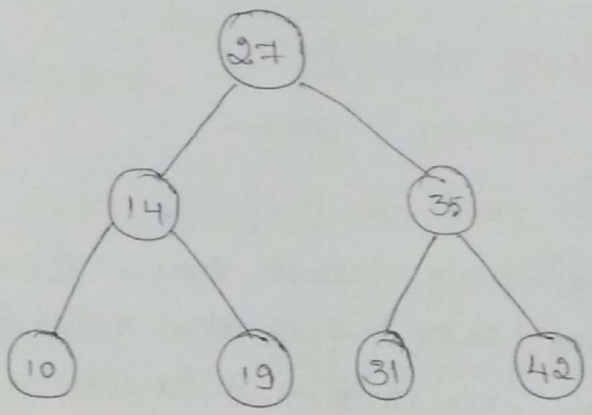
# MODULE - IV

7. A. With a neat Illustration of BST, Explain the C function to Search an Element in a BST.

Ans:- => The left subtree of a node Contains only nodes with Keys lesser than the node's Key

=> The right Subtree of a node Contains only nodes with greater than the node's Key.

=> The left and right Subtree Each must also be a binary Search tree




1. Start from root.
2. Compare the inserting Element with root, if less than root, then insert (or) recurse for left Else Insert for Right.
3. If Element to Search is found anywhere, return true, Else return false

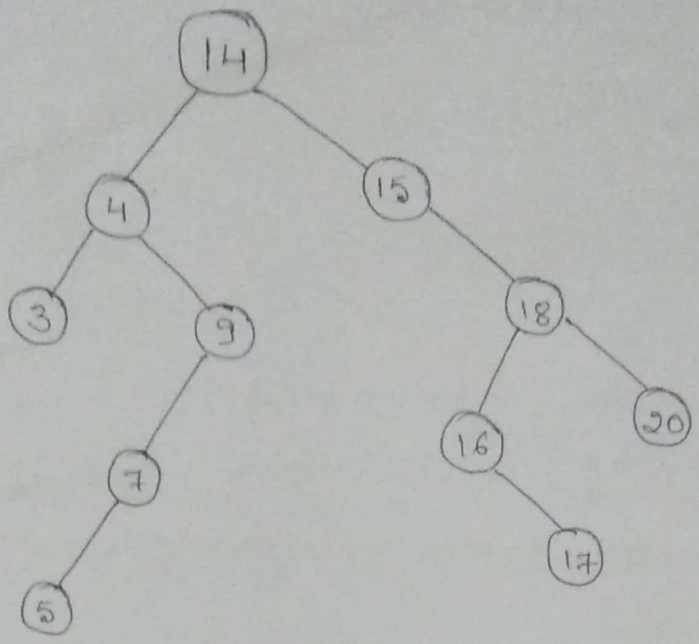
C Function to Search an Element in a BST

```
int Search (TNode root, int Key)
{
    if (root != NULL)
    {
        if (root->data == Key)
            return Key;
        if ( Key < root->data)
        return Search (root->llink, Key);
        return Search (root->rlink, Key);
    }
    return -1;
}
```

B. Define a binary Search tree. Draw the Binary Search tree (BST) for the following Sequence of Elements, 14, 15, 9, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9

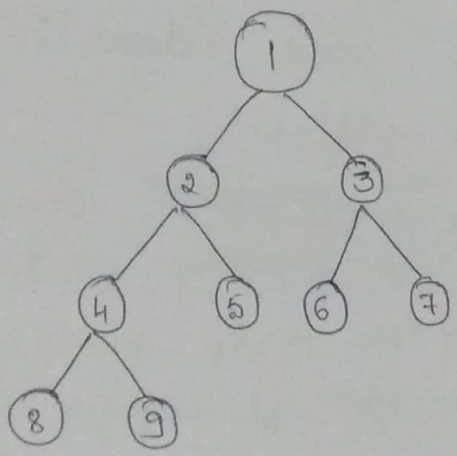
Ans: Binary Search Tree:

A binary Search tree is also Known as an ordered binary tree. Each node has no more than two child nodes. the left Subtree of a node has a Key less than or Equal to its parent node's Key. the right Sub-tree of a node has a Key greater than or Equal to its parent node's Key.




8. A. Draw a Binary tree for the following set of number: 1, 2, 3, 4, 5, 6, 7, 8, 9 Trace the above generated tree using inorder, preorder and postorder. Also Write C functions for each.

Ans:-




Inorder:- 8, 4, 9, 2, 5, 1, 6, 3, 7

preorder:- 1, 2, 4, 8, 9, 5, 3, 6, 7

postorder:- 8, 9, 4, 5, 2, 6, 7, 3, 1

Function for Inorder

```
Void inorder (TNode root)
{
    if (root != NULL)
    {
        inorder (root -> llink);
        printf ("%d\n", root -> data);
        inorder (root -> rlink);
    }
}
```

Function for preorder

```
void preorder (TNode root)
{
    if (root != NULL)
    {
        printf ("%d\n", root -> data);
        preorder (root -> llink);
        preorder (root -> rlink);
    }
}
```

Function for postorder

```
Void postorder (TNode root)
{
    if (root != NULL)
    {
        postorder (root->llink);
        postorder (root->rlink);
        printf ("%d\n", root->data);
    }
}
```

13. What is the advantage of threaded binary tree over binary tree ? Explain the Construction of threaded binary tree for 10, 20, 30, 40, 50.

Ans:

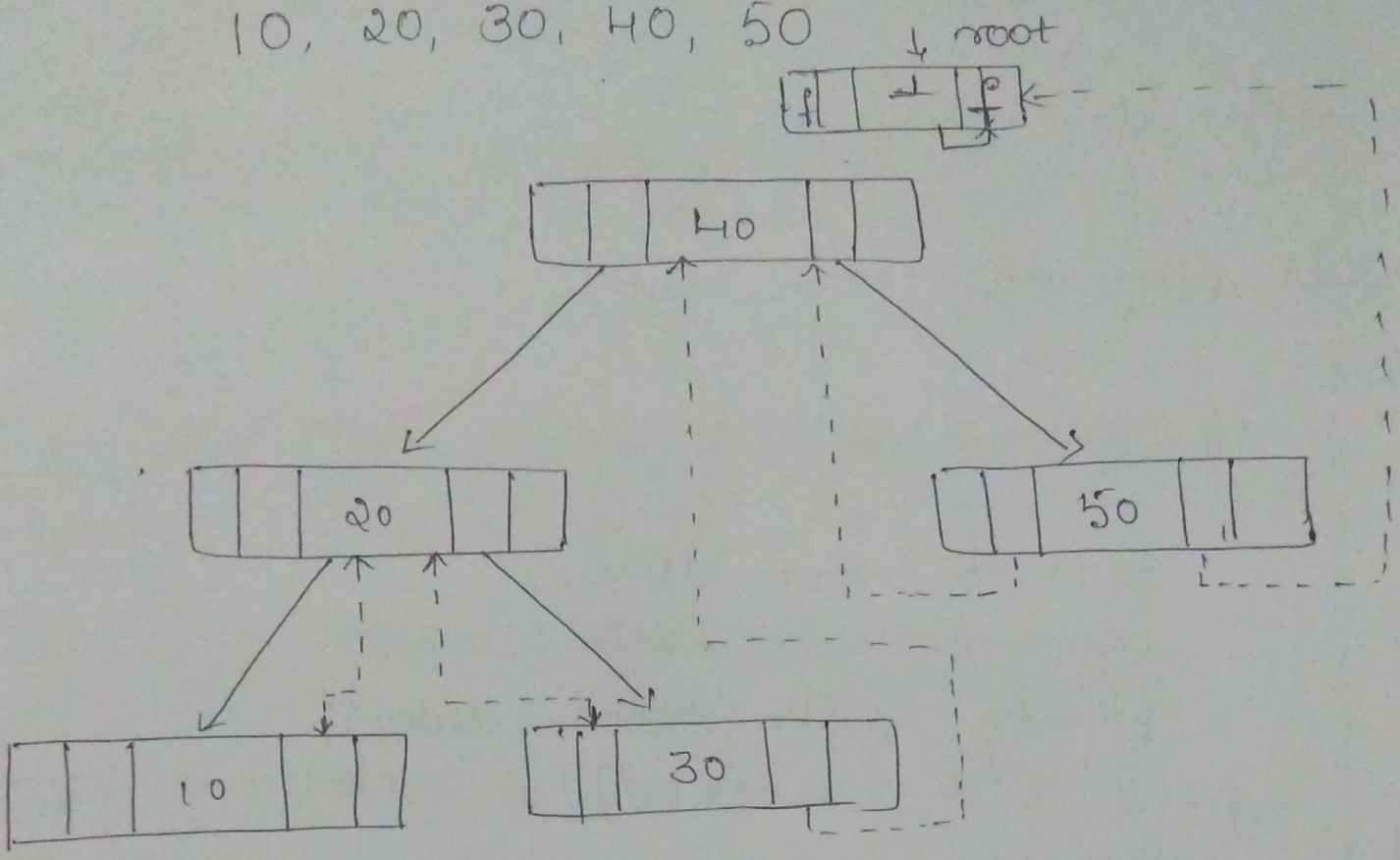
Advantages

-> It reduces the time Complexity of algorithm.

-> Operations like, traversal, insertion, deletion Can be done without Using recursive algorithm.

-> In threaded binary tree, the null pointers are used as thread.

10, 20, 30, 40, 50



K B Hemanth Raj
1BY18CS404
3rd Sem CSE

MODULE - V

9A. ~~Draw a binary tree~~ Write a C function for insertion sort. Sort the following list using insertion sort : 50, 30, 10, 70, 40, 20, 60

```
// C function for insertion Sort.
Void insertion_sort(int arr[], int n)
{
    int i, key, j;
    for(i=1; i<n; i++)
    {
        key = arr[i];
        j = i-1;
        while(j>=0 && arr[j]>key)
        {
            arr[j+1] = arr[j];
            j=j-1; //j--;
        }
        arr[j+1] = key;
    }
}
```

Assume we have n = 7. i.e, from 1 to 7 iteration & the input key sequence is 50, 30, 10, 70, 40, 20, 60. After each iteration we have.

| | [1] | [2] | [3] | [4] | [5] | [6] | [7] |
|---|---|---|---|---|---|---|---|
| - | 50 | 30 | 10 | 70 | 40 | 20 | 60. |
| 1 | 30 | 50 | 10 | 70 | 40 | 20 | 60 |
| 2 | 10 | 30 | 50 | 70 | 40 | 20 | 60. |
| 3. | 10 | 30 | 50 | 70 | 40, | 20 | 60 |
| 4 | 10 | 30 | 40 | 50 | 70 | 20 | 60 |
| 5. | 10 | 20 | 30 | 40 | 50 | 70 | 60 |
| 6 | 10 | 20 | 30 | 40 | 50 | 60 | 70. |

9B. What is collision? What are the methods to resolve collision? Explain linear probing with an example.

If for a given set of key values the hash function does not distribute them uniformly over the hash table, some entries are there which are empty and in some entries more than one key value are to be stored. Allotment of more than one key value in one locations in the hash table is called "collision".

Collision in hashing cannot be the size of the hash table. Three techniques to resolve collision are

(i) Closed hashing (LINEAR PROBING). and
(ii) Open hashing (CHAINING)

## ⇒ CLOSED HASHING (LINEAR PROBING)

The simplest method to resolve a collision is closed hashing. Here the hash table is considered as circular so that when the last location is reached the search proceeds to the first location of the table. That is why this is called closed hashing.

The search will continue until any one case:

* The key value is found
* Unoccupied location is encountered.
* It reaches to the location where the search was started.

## ⇒ EXTENDED HASHING (DYNAMIC HASHING)

Some hashing techniques allow the hash function to be modified dynamically to accommodate the growth or shrinking of the database. These are called dynamic hash functions.

* Extendable hashing is one form of dynamic hashing.
* Extendable hashing splits and coalesces buckets as database size changes
* This imposes some performance overhead, but space efficiency is maintained.
* As reorganization is on one bucket at a time, overhead is acceptably low.

10A. What do you understand by the term file organization? Briefly summarize any three widely used file organization techniques.

File Organization is a process where the files are arranged in a particular order depending on the attributes like FileName, FileSize, Type of application, File volatility, the respond time, etc....

The various file organization methods are :

* Sequential access,
* Direct or random access,
* Index sequential access.

⇒ SEQUENTIAL ACCESS :

Here the records are arranged in the ascending or descending order or chronological order of a key field which may be numeric or both. Since the records are ordered by a key field, there is no storage location identification. It is used in applications like payroll management where the file is to be processed in entirety, i.e each record is processed. Here, to have an access to a particular record, each record must be examined until we get the desired record. Sequential files are normally created and stored on magnetic tape using batch processing method.

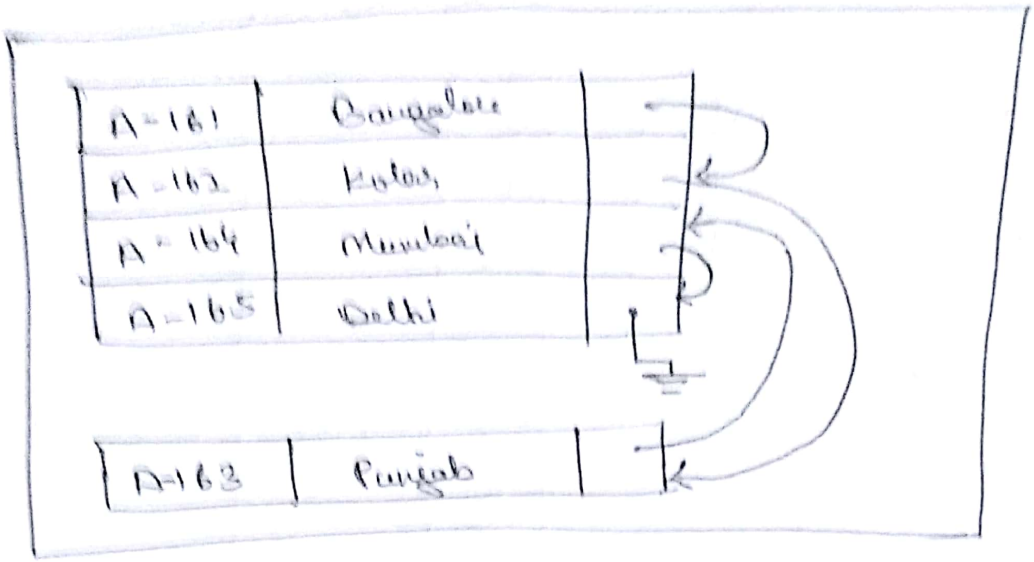



Fig: Sequential access

⇒ DIRECT ACCESS FILES ORGANIZATION: Files in this type are stored in direct access storage devices such as magnetic disk, using an identifying key. The identifying key relates to its actual storage position in the file. The Computer can directly locate the key to find the desired record without having to search through any other record first. Here the ~~data~~ records are stored randomly, hence the name random file. It uses online system where the response and updation are fast.

⇒ INDEXED SEQUENTIAL ACCESS ORGANIZATION:

Here the records are stored sequentially on a direct access device. i.e magnetic disk and the data is accessible randomly and sequentially. It covers the positive aspects of both sequential and direct access files. The type of file organization is suitable for both batch processing and online processing. Here, the records are organized in ~~suitable~~ sequence for efficient processing of large batch jobs, but an index is also used to speed up access to the records.

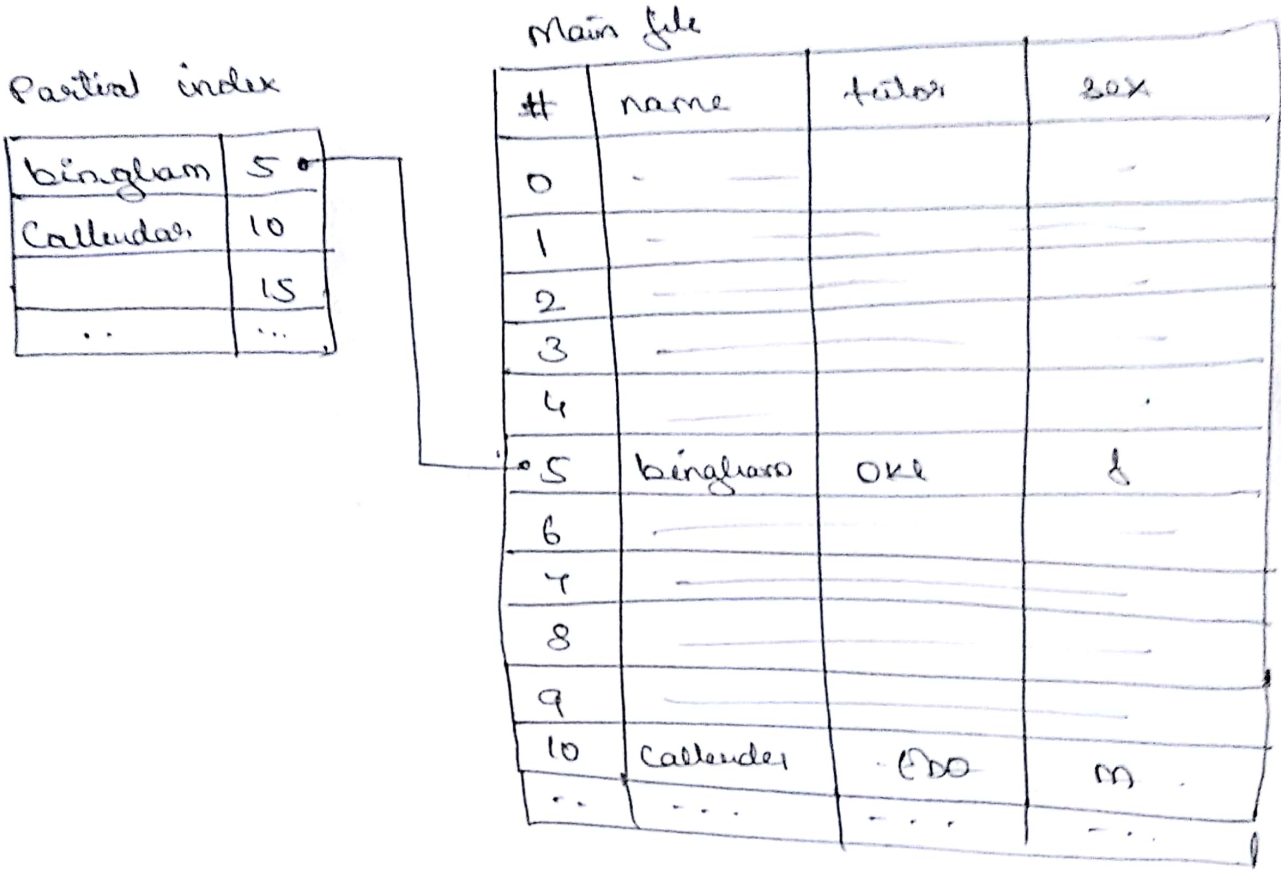



Fig: Indexed Sequential access organization.

10B. State and explain WARSHALL's algorithm with an example.

Warshall observed that $P_k[i,j]=1$ can occur only if one of the following two cases occurs.

(1) There is a simple path from $V_i$ to $V_j$ which does not use any other nodes except possible $V_1, V_2, V_3 \ldots V_{k-1}$ hence

$$P_{k-1}[i,j]=1$$

(2) There is a simple path from $V_i$ to $V_k$ and a simple path from $V_k$ to $V_j$ where each path does not use any other nodes except possibly $V_1, V_2, V_3 \ldots V_{k-1}$ hence

$$P_{k-1}[i,k]=1 \quad \text{and} \quad P_{k-1}[k,j]=1.$$


$\therefore$ $V_i \longrightarrow \cdots \longrightarrow V_s \cdots \longrightarrow V_k$

$$\therefore P_k[i,j] = P_{k-1}[i,j] \vee (P_{k-1}[i,k] \wedge P_{k-1}[k,j])$$

Algorithm: A directed graph G with M nodes is maintained in memory by its adjacency matrix A. This algorithm finds the (Boolean) path matrix P of the graph G.

(1) Repeat for I, J = 1, 2 ---- M [INITIALIZE P.]
    If A[I,J] = 0 then: SET P[I,J] = 0
    else: SET P[I,J] = 1
    [END of loop]

(2) Repeat Step 3 and 4 for K = 1, 2, ---- M

(3) Repeat ~~Step~~ 4 for I = 1, 2, ---- M.

(4) Repeat for J = 1, 2, ---- M.
    Set P[I,J] := P[I,J] $\vee$ (P[I,K] $\wedge$ P[K,J])
    [End of loop]
    [End of Step 3 loop]
[End of Step 2 loop]

(5) Exit.