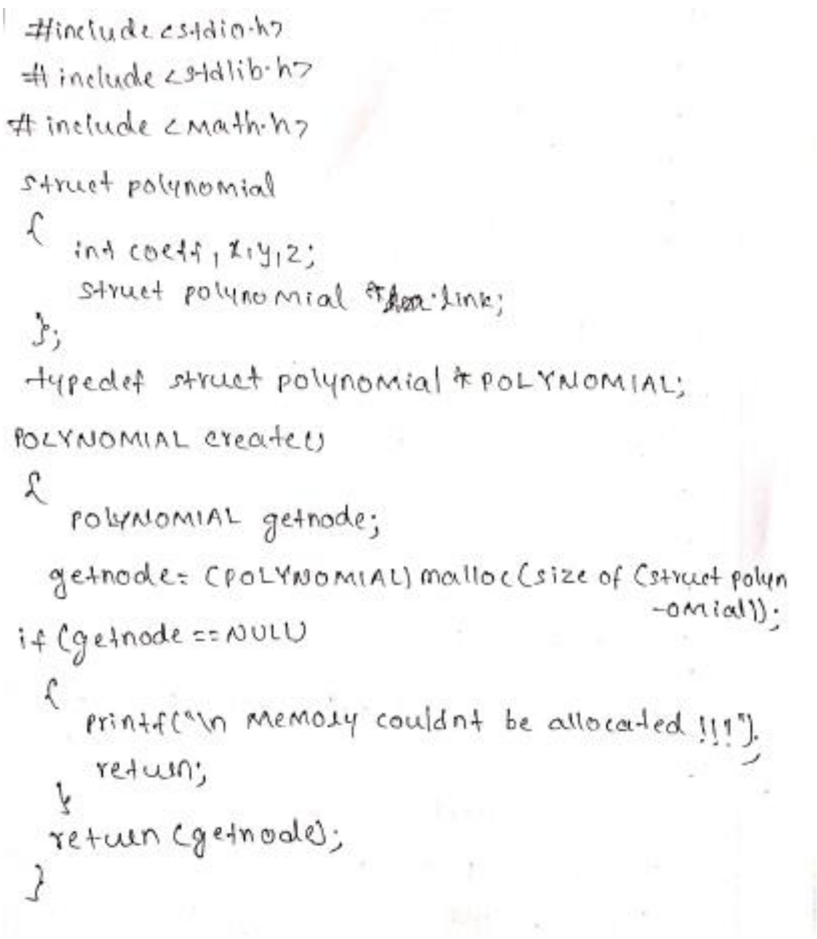
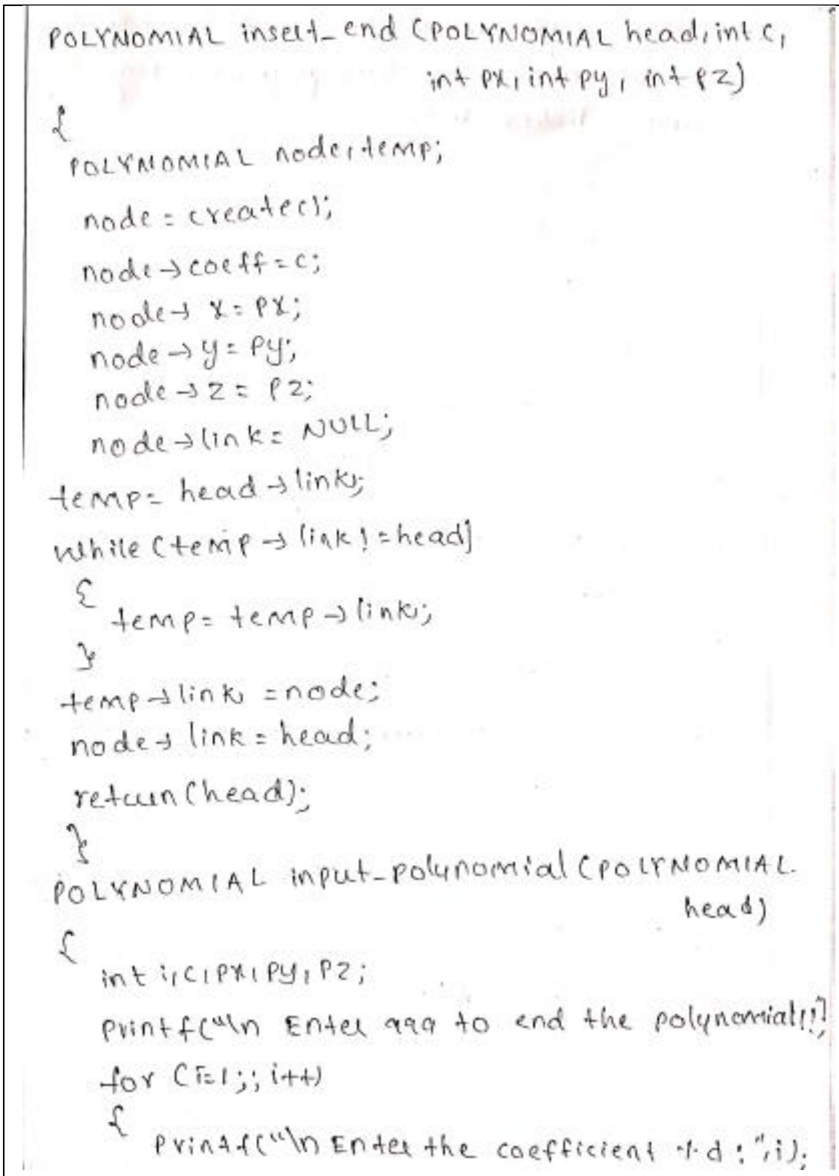
Internal Assessment Test 3 – November 2019								
Sub:	Data Structures and Applications				Sub Code:	18CS32	Branch:	CSE
Date:	19/11/2019	Duration:	90 mins	Max Marks:	50	Sem/ Sec:	3rd /A,B,C	OBE

Answer any FIVE FULL Questions	MARK S	CO	RB T
1. Write a C program to take input two polynomials and store the sum in third polynomial using Singly Circular Linked List with header nodes.	[5+5]	CO3	L3



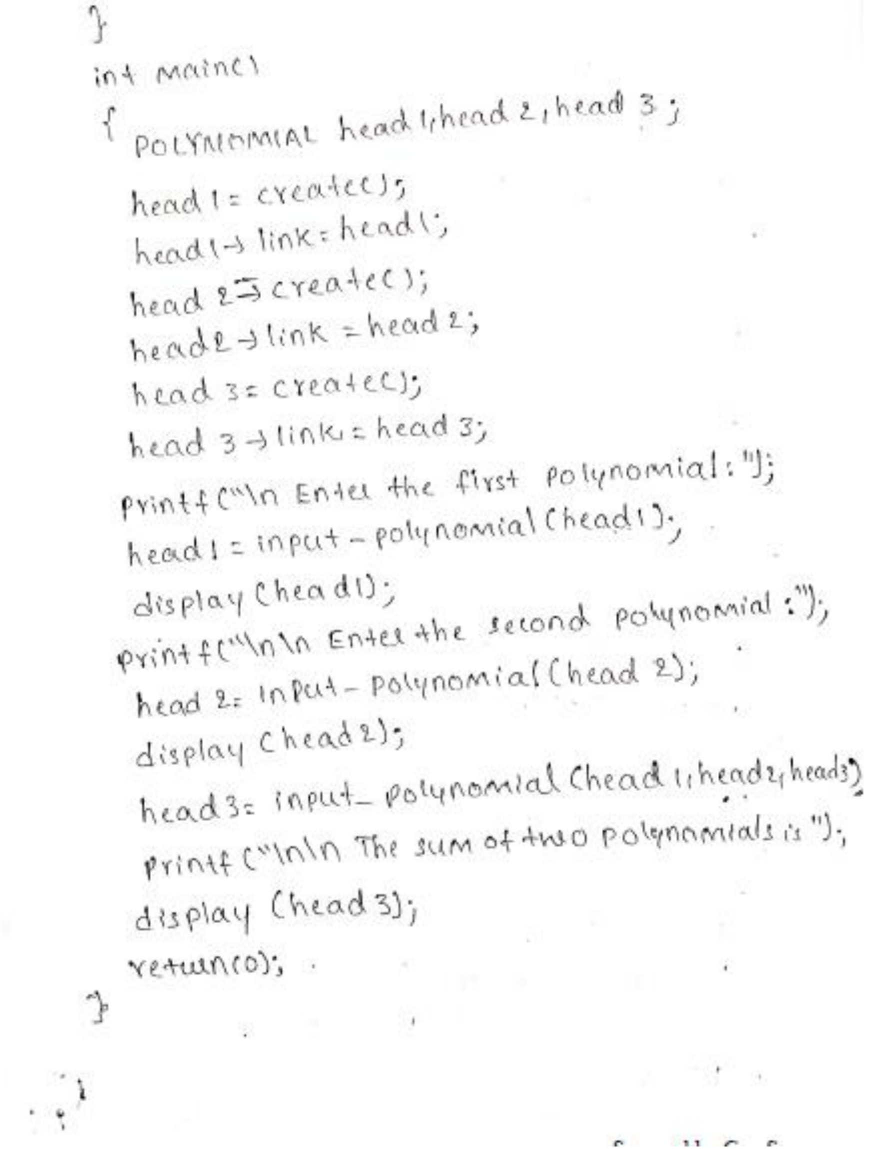
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct polynomial
{
    int coeff, x,y,z;
    struct polynomial *link;
};
typedef struct polynomial *POLYNOMIAL;
POLYNOMIAL create()
{
    POLYNOMIAL getnode;
  getnode= (POLYNOMIAL) malloc(size of (struct polyn
                                           -omial));
if (getnode ==NULL)
 {
    printf("\n Memory couldnt be allocated !!!");
    return;
  }
 return (getnode);
}
```



```
POLYNOMIAL insert_end (POLYNOMIAL head, int c,
                              int px, int py, int pz)
{
 POLYNOMIAL node, temp;
  node = create();
  node->coeff = c;
   node-> x = px;
   node-> y = py;
   node-> z = pz;
   node->link = NULL;
temp = head->link;
while (temp->link != head)
 {
   temp = temp->link;
 }
temp->link = node;
 node->link = head;
 return(head);
}
POLYNOMIAL input_polynomial (POLYNOMIAL
                                   head)
{
   int i, c, px, py, pz;
   printf("\n Enter 999 to end the polynomial!");
   for (i=1;; i++)
   {
      printf("\n Enter the coefficient %d:", i);
```

```
scanf("%d",&c);
 if(c==999)
     break;
printf("\n Entee the powee of x:");
scanf("%d", &px);
printf("\n Entee the powee of y:");
scanf ("%d", &py);
printf("\n Entee the polynomial powee of z:");
 scanf("%d", &pz);
 head=inselt_end(head,c, px,py,pz);
}
retuen(head);
}
void display(POLYNOMIAL head)
{
 POLYNOMIAL temp;
 if(head->link==head)
 {
  printf("\n polynomial doesnt exist!");
  retuen;
 }
 temp=head->link;
 while(temp!=head)
 {
```

```
printf(" %dx^%dy^%dz^%d+", temp->coeff,
          temp->x, tempBy, temp->z);
 temp=temp->link;
}
printf("999");
}
POLYNOMIAL sum-polynomial(POLYNOMIAL
            head1, POLYNOMIAL head2, POLYNOMIAL
             head3)
{
 POLYNOMIAL P1, P2;
 int c, c1, c2, x1, y1, z1, z2, flag, x2, y2;
 P1=head1->link;
 while (P1!=head1)
 { c1=P1->coeff;
   x1=P1->x;
   y1=P1->y;
   z1=P1->z;
   P2=head2->link;
   flag=0;
  while (P2!=head2)
   {
     c2=p2->coeff;
     x2=p2->x;
     y2= p2->y;
     z2 = p2->z;
   if ((x1==x2)&&(y1==y2)&&(z1==z2))
   {
     head3= insert_end (head3, c1+c2, x1,y1,z1);
     p2->coeff=0;
     flag=1;
     break;
   }
   else
      p2=p2->link;
  }
  if (flag==0)
    head3= insert_end (head3, c1,x1,y1,z1);
    p1->p1->link;
  }
    p2=head2->link;
   while(p2!=head2)
   { if(p2->coeff!=0)
      head3= insert_end (head3, p2->coeff,
                        p2->x, p2->y, p2->z);
   } p2 = p2->link;
```

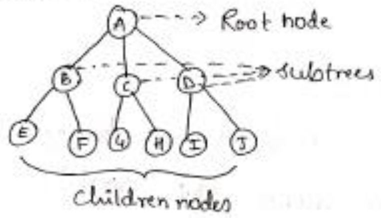


```
}
int main()
{
  POLYNOMIAL head1,head2,head3;
  head1= create();
  head1-> link=head1;
  head2= create();
  head2->link =head2;
  head3= create();
  head3->link=head3;
 printf("\n Enter the first polynomial:");
  head1 = input-polynomial(head1);
  display(head1);
 printf("\n\n Enter the second polynomial :");
  head2= input-polynomial(head2);
  display(head2);
  head3= input_polynomial(head1,head2,head3)
  printf("\n\n The sum of two polynomials is ");
  display (head3);
  return(0);
}
```

2. a) What is a tree? Write recursive C functions to traverse the tree in Pre-order, In-order and Post-order.	[1+1+2+2]	CO3	L3

A tree is a collection of nodes that shows parent -child relation such that:

- * Special node called the root node
- * Remaining nodes are partitioned into subsets $T_1, T_2, T_3, \ldots T_n$. Where $T_1, T_2, T_3, --- T_n$, which are all children of root node are themselves trees called subtrees.



Preorder Traversal:

C Function to traversel preorder

```
void preorder(NODE ROOT)
{
  if (root = = NULL)
   return;
  printf("%d", root → info);
  Pre order (root → l link);
  Pre order (root → r link);
}
```

Post order traversal:-

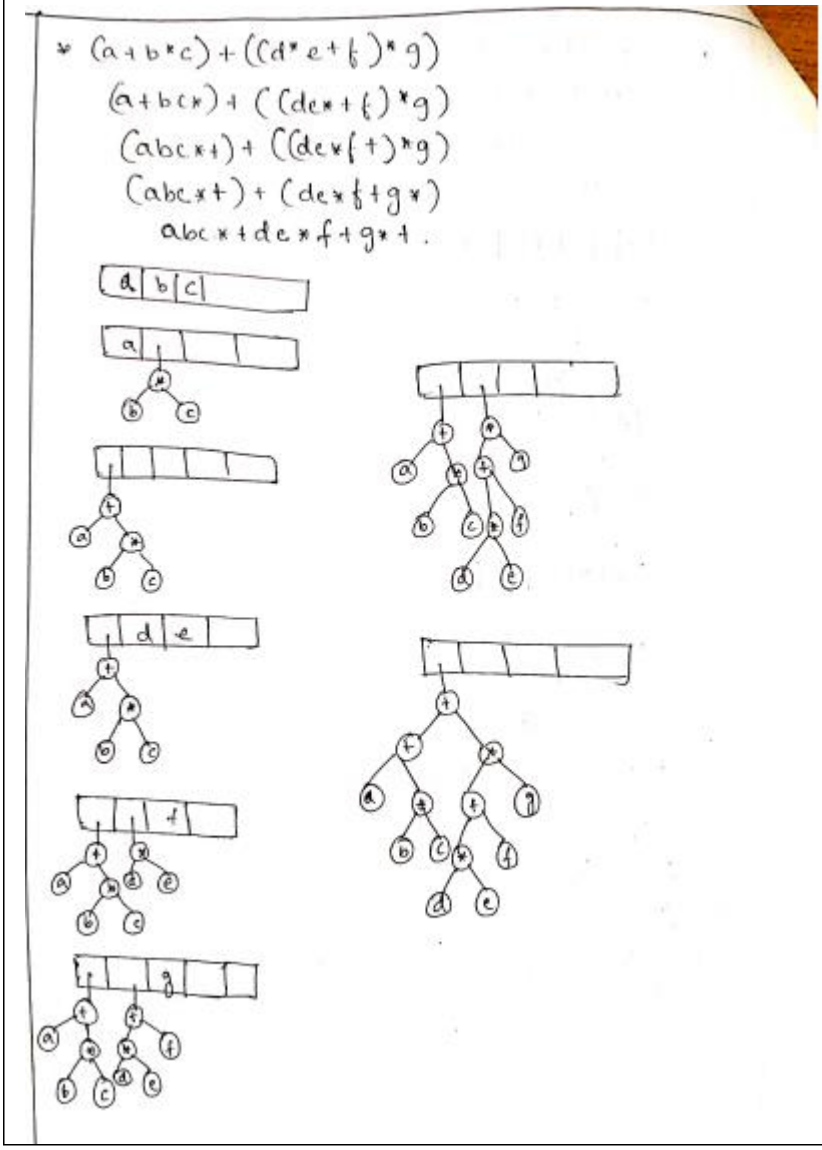
C Function for post order traversal

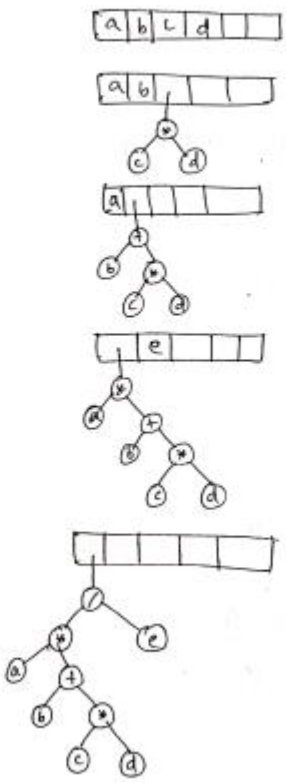
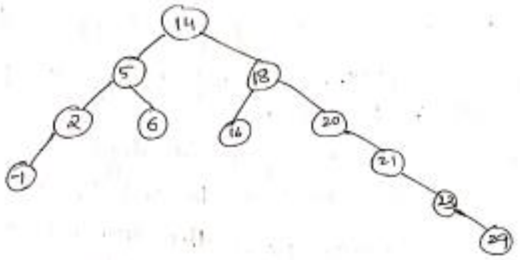
```
Void postorder (NODE root)
{
  if (root = = NULL)
   return;
  post order (root → l link);
  post order (root → r link);
  printf("%d", root → info);
}
```

Inorder traversal:-

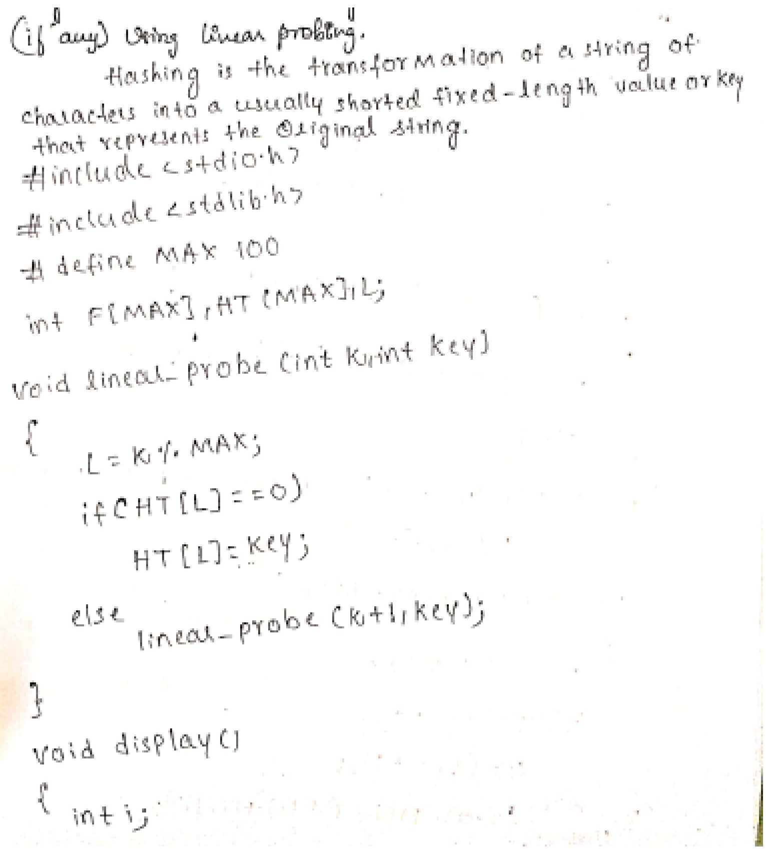
C function for Inorder traversal

```
void inorder (NODE root)
{
   if (root = = NULL)
      return;
   inorder (root → llink);
   printf("%d", root → info);
   inorder (root → l link);
}
```

2b. **Construct an expression tree for the following infix expression** - **a*(b+c*d)/e** **(a+b*c)+((d*e+f)*g)** Answer:  * (a+b*c)+((d*e+f)*g) (a+bc*)+((de*+f)*g) (abc*+)+((de*f+)*g) (abc*+)+(de*f+g*) abc*+de*f+g*+.	[2+2]	CO3	L3

* a* (b+c*d)/e a* (bcd*+)/e abcd*+*/e abcd*+*e/ 			
<span>3a</span> Construct a Binary Search Tree for the following input. 14,5,6,2,18,20,16,-1,21,23,29. Also traverse the BST using in-order, pre-order and post-order traversal.  Inorder traversal -1, 2, 5, 6, 14, 16, 18, 20, 21, 23, 29 Preorder traversal 14, 5, 2, -1, 6, 18, 16, 20, 21, 23, 29 Postorder traversal -1, 2, 6, 5, 16, 29, 23, 21, 20, 18, 14.	**[4+2+2+2]**	CO3	L3

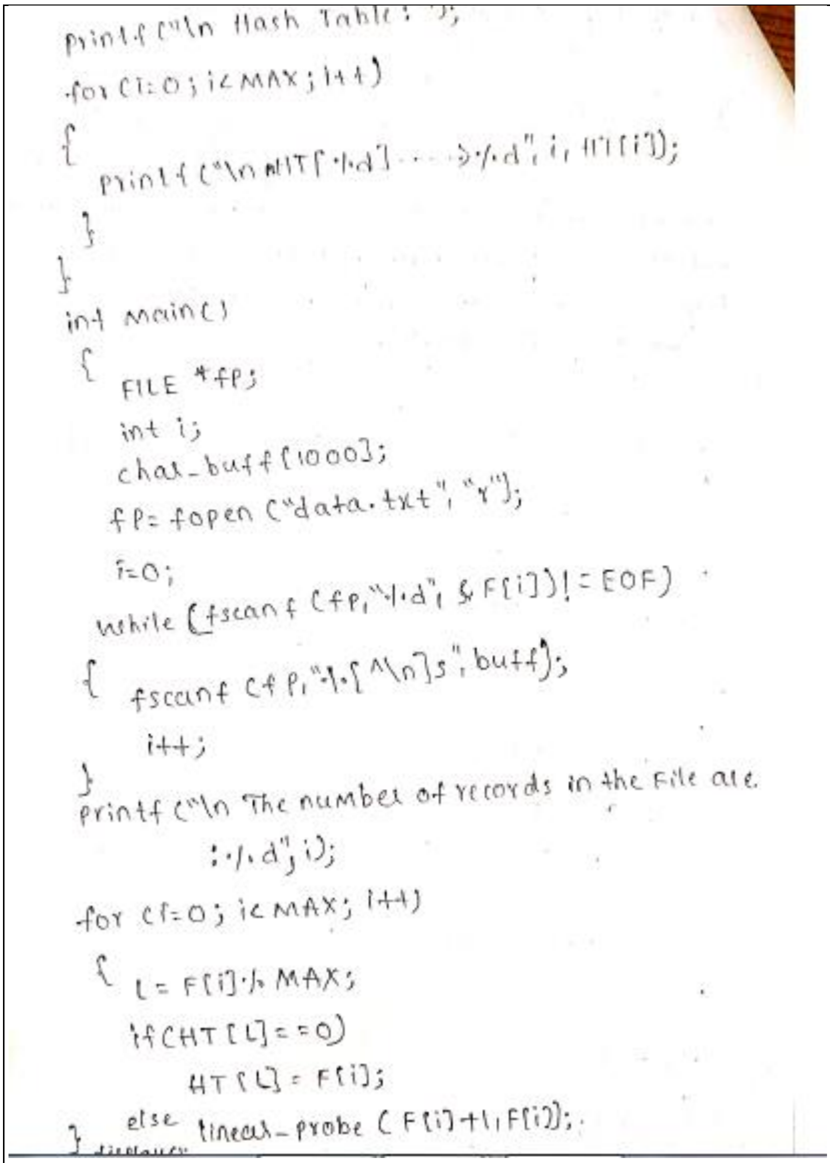
4. **What is hashing? Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let thekeys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.**	**3+3+4**	CO4	L3



(if any) using linear probing.

Hashing is the transformation of a string of characters into a usually shorted fixed-length value or key that represents the original string.

```
#include <stdio.h>
#include <stdlib.h>
# define MAX 100
int F[MAX], HT [MAX], L;
void linear_probe (int K, int key)
{
    L = K % MAX;
    if (HT [L] == 0)
        HT [L] = key;
    else
        linear-probe (K+1, key);
}
void display ()
{ int i;
```



```
printf("\n Hash Table: ");
for(i=0;i<MAX;i++)
{
  printf("\n HT[%d]...->%d",i,HT[i]);
 }
}
int main()
 {
   FILE *fp;
   int i;
   char_buff[1000];
  fp=fopen("data.txt","r");
   i=0;
  while(fscanf(fp,"%d",&F[i])!=EOF)
 { fscanf(fp,"%[^\n]s",buff);
     i++;
 }
printf("\n The number of records in the File are.
          :%d",i);
for(i=0;i<MAX;i++)
 { L=F[i]%MAX;
   if(HT[L]==0)
       HT[L]=F[i];
 }  else linear_probe(F[i]+1,F[i]);
```

5. **Write an algorithm for radix sort. Apply radix sort and show the various passes to sort the array W where W={132,235,456,758,659,900,200,37,26,136}**	**5+5**	CO4	L3

The Algorithm can be performed Using following steps

1) Define ten queues, each representing the bucket for even digit from 0 to 9

2) Consider the least significant digit, of each number present in the least to be sorted.

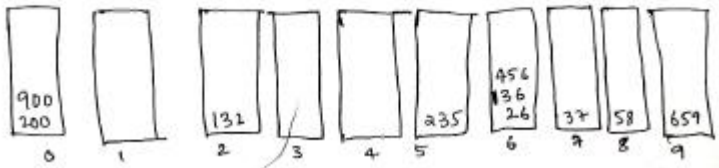
3) Insert each number prove the list into the respective queue based on the least significant digit.

4) Group all the numbers from Q_0 to Q_9 in the order of their insertion into the queues of consider the list for next step as input list.

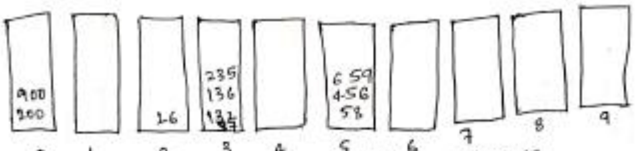
5) Repeat from step 3, based on the next least significant bit.

) Repeat Until all the. numbers are grouped based on the more significant bit.

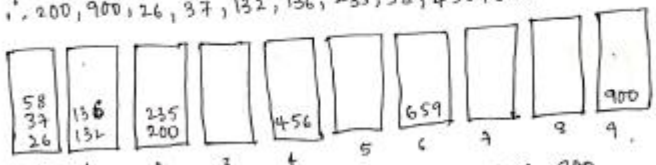
$ul = \{132, 235, 456, 758, 659, 900, 200, 37, 26, 136\}$



∴ 200, 900, 132, 235, 26, 136, 456, 37, 58, 659



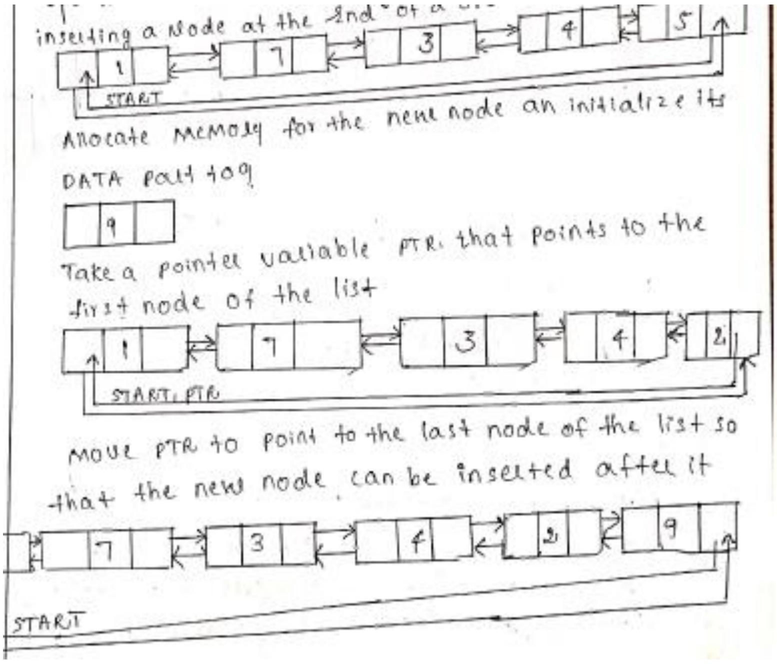
∴ 200, 900, 26, 37, 132, 136, 235, 58, 456, 659



∴ 26, 37, 58, 132, 136, 200, 235, 456, 659, 900

front() and display()

6. **Explain the insert_end(), delete_front() and display() operations on Circular Doubly Linked List.**	4+4+2	CO3	L3

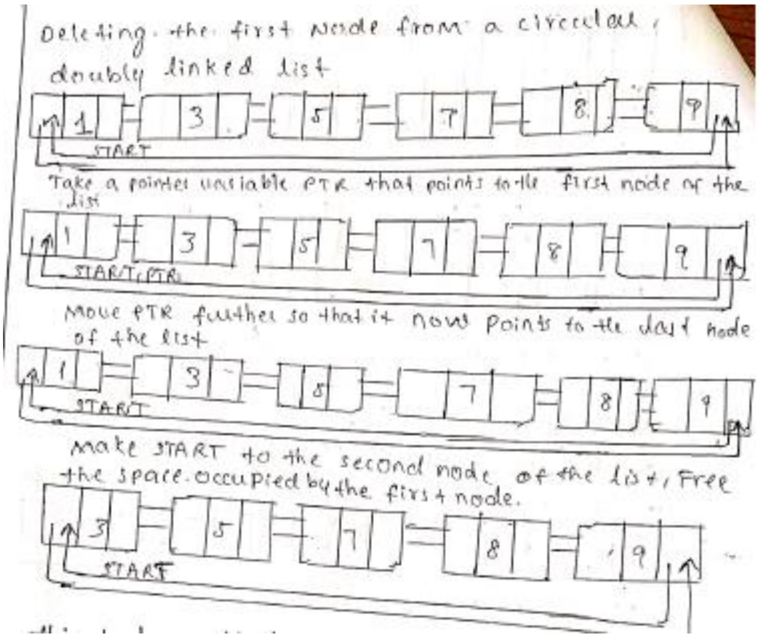


inserting a Node at the End of a

Allocate Memory for the new node an initialize its DATA part to 9

Take a pointer variable PTR that points to the first node of the list

Move PTR to point to the last node of the list so that the new node can be inserted after it

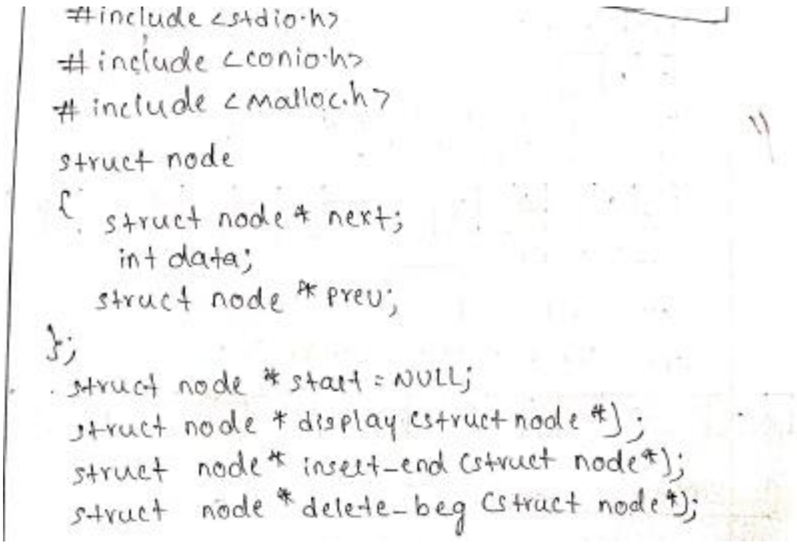


Deleting the first Node from a circular doubly linked list

Take a pointer variable PTR that points to the first node of the list

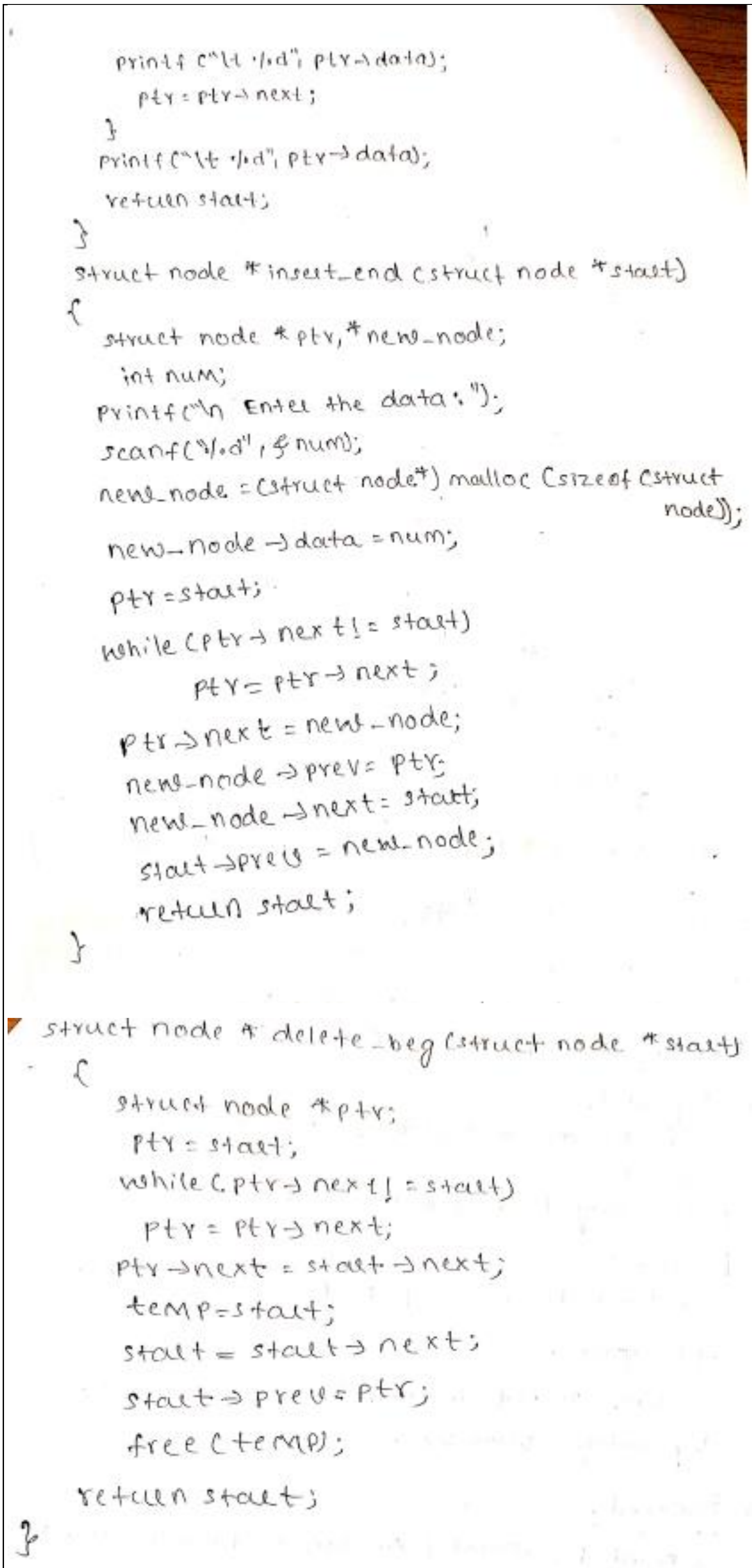
Move PTR further so that it now points to the last node of the list

Make START to the second node of the list, Free the space occupied by the first node.



```
#include <stdio.h>
#include <conio.h>
# include <malloc.h>
struct node
{ struct node * next;
   int data;
  struct node *prev;
};
struct node * start = NULL;
struct node * display (struct node *);
struct node * insert_end (struct node *);
struct node * delete_beg (struct node *);
```

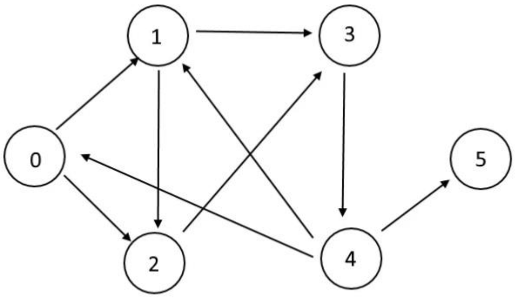
```
int main()
 {
   int option;
   clrscr();
   do
   {
    printf("\n 1: display the list");
    printf("\n 2: Add a node at the end");
     printf("\n 3: Delete a  node at the beginning");
  scanf("%d", &option);
  switch(option)
   {
          case 1 :  start = display(start);
                     break;
          case 2 : ~~start~~insert_end(start);
                     break;
          case 3 : start=delete_beg(start);
                     break;
         } ~~case~~
    } while(option != 4);
     getch();
     return 0;
   }.
struct node *display(struct node *start)
 {
    struct node *ptr;
     ptr= start;
    while (ptr->next != start)
     {
```

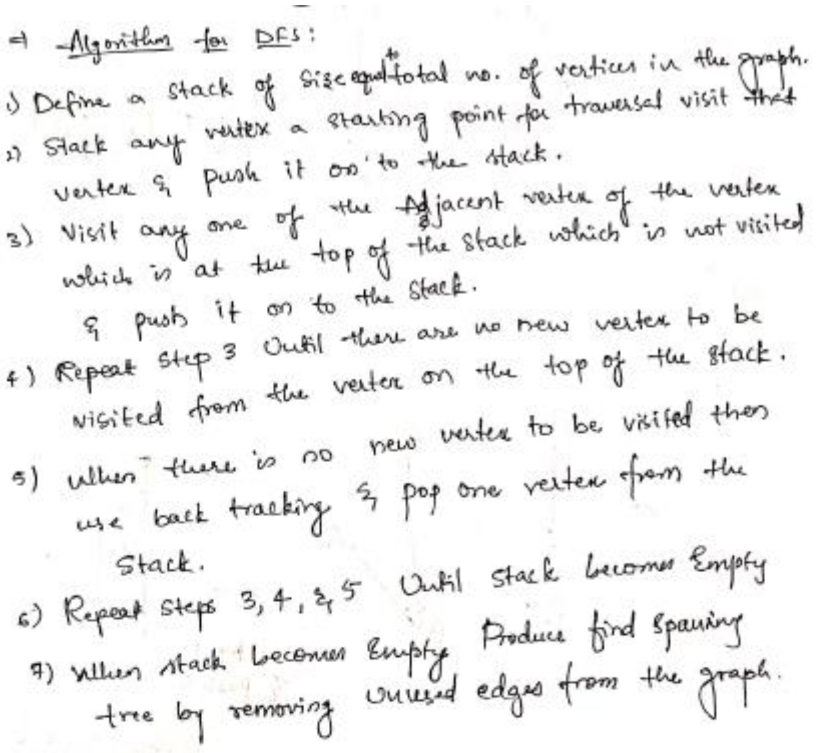


```
    printf("\t %d", ptr->data);
      ptr = ptr->next;
    }
   printf("\t %d", ptr->data);
   return start;
}
struct node *insert_end(struct node *start)
{
  struct node *ptr,*new_node;
   int num;
  printf("\n Enter the data:");
  scanf("%d", &num);
  new_node = (struct node*) malloc (sizeof (struct
                                        node));
  new_node->data = num;
  ptr = start;
  while (ptr->next != start)
         ptr = ptr->next;
   ptr->next = new_node;
   new_node->prev = ptr;
   new_node->next = start;
    start->prev = new_node;
    return start;
}

struct node * delete_beg(struct node *start)
 {
    struct node *ptr;
     ptr = start;
    while(ptr->next != start)
      ptr = ptr->next;
    ptr->next = start->next;
     temp=start;
    start = start->next;
    start->prev = ptr;
    free(temp);
  return start;
}
```

7a. **Write an algorithm for BFS/DFS. Write the adjacency matrix and also print the nodes reachable(step by step) from the starting vertex 2 using BFS and DFS for the graph given below:**	[5+2.5+2.5]	CO3	L3



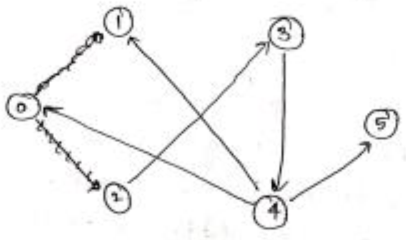

→ Algorithm for DFS:

1) Define a stack of size equal to total no. of vertices in the graph.
2) Stack any vertex a starting point for traversal visit that vertex & push it on to the stack.
3) Visit any one of the Adjacent vertex of the vertex which is at the top of the stack which is not visited & push it on to the stack.
4) Repeat step 3 Until there are no new vertex to be visited from the vertex on the top of the stack.
5) When there is no new vertex to be visited then use back tracking & pop one vertex from the stack.
6) Repeat steps 3, 4, & 5 Until stack becomes Empty
7) When stack becomes Empty Produce find Spaning tree by removing unused edges from the graph.

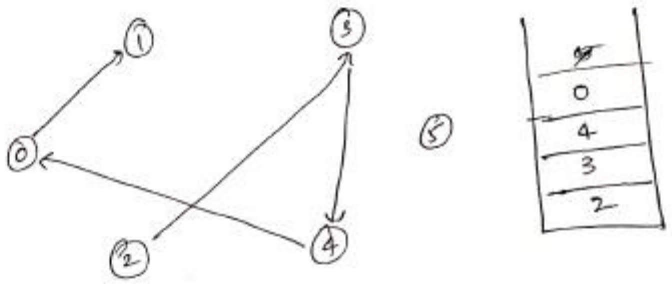
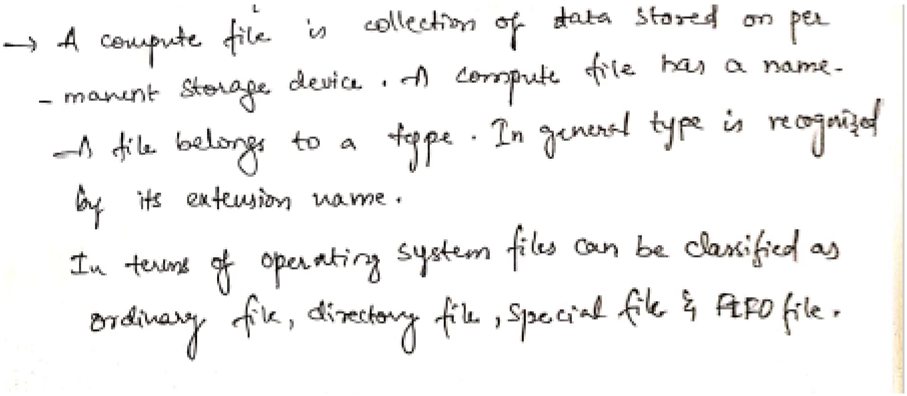
Adjacency Matrix :

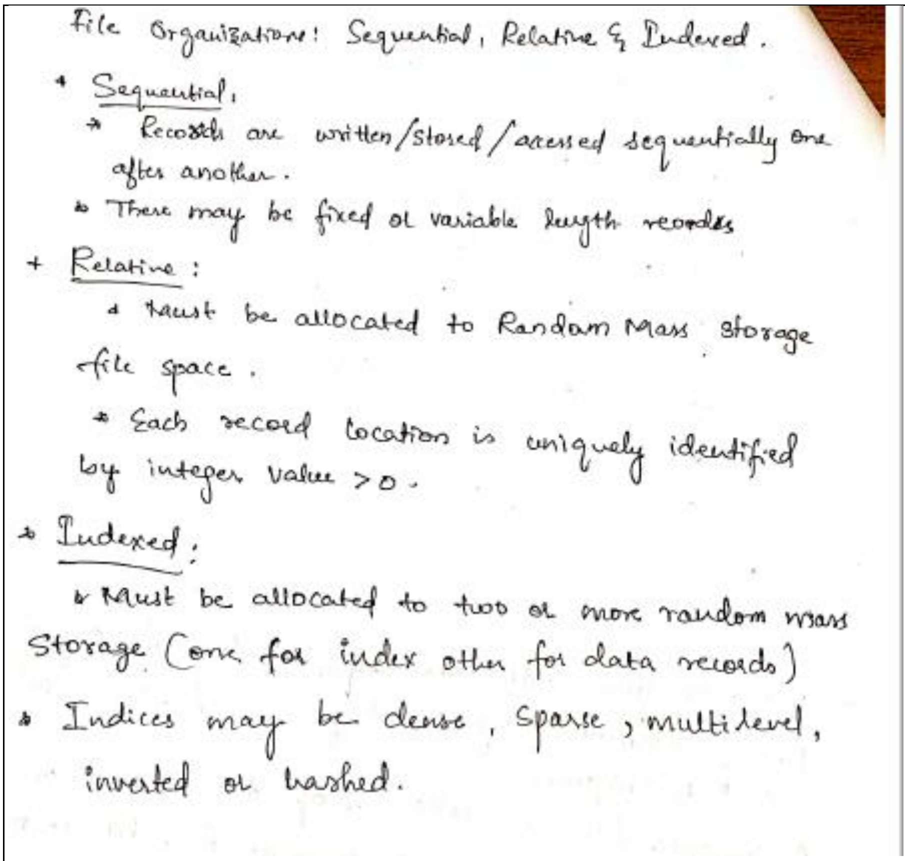
	0	1	2	3	4	5
0	0	1	1	0	0	0
1	0	0	1	1	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	1	1	0	0	0	1
5	0	0	0	0	0	0

BFS .



2	3	4	0	1	5

Pop 5 push 0  			
8. What do you understand by the term file organization? Briefly summarize any 3 widely used file organization techniques.  → A compute file is collection of data stored on permanent storage device. A compute file has a name. –A file belongs to a type. In general type is recognized by its extension name. In terms of operating system files can be classified as ordinary file, directory file, Special file & FIFO file.	2+8	CO2	L2

 File Organizations: Sequential, Relative & Indexed. * Sequential, * Records are written/stored/acessed sequentially one after another. * There may be fixed or variable length records + Relative : * Must be allocated to Random Mass storage file space. * Each record location is uniquely identified by integer value >0. * Indexed: * Must be allocated to two or more random mass Storage (one for index other for data records) * Indices may be dense, Sparse, multilevel, inverted or hashed.			