PAGE NO :
DATE :

Output: 1. Create
2. Insert before
3. Delete
4. Exiting
Enter the choice : 1
Enter the number of elements 3
Enter the element : 45
Enter the element : 67
Enter the element : 80
DL created

Enter the ~~number of~~ choice : 2
enter pos 3
Insertion Success

Enter the choice : 3
enter the value to be deleted
Deleted value 32

Enter the choice : 4
Exiting

45
67
80

DS Lab (Week 8)

WAP (a) To construct a binary search tree
(b) To traverse the tree using all the methods i.e, inorder, preorder and postorder
(c) To display the elements in the tree.

```
#include<stdio.h>
#include< stdlib.h>
struct node
{ struct node *left;
  struct node* right;
  int data;
};
struct node *tree = NULL;
void create();
void pre(struct node*);
void post(struct node*);
void in( struct node*);

void main()
{ int option;
  do { printf("1. Create a binary search tree \n2
Preorder traversal \n3. Postorder traversal\n4. In
traversal\n5. Exit\nEnter an option:");
  scanf("%d", &option);
  switch(option)
  { case 1: create();
          printf("Binary search tree created\n"
          break;
    case 2: printf("\n The elements in the tree are
          pre(tree);
          break;
    case 3 : printf("\n The elements in the tree a
          post(tree); break;
```

PAGE NO :
DATE :

LeetCode (Split Linked List)

```
typedef struct ListNode lnode;
int get_len(lnode* head) {
    int n=0;
  while(head) {
      n++;
  head =head->next;
}
return n;
}
struct ListNode** splitListToParts(struct ListNode* head,
 int k, int* returnSize){
  int n=get_len(head), elems, i,j;
  *returnSize =k;
lnode **list =(lnode**)calloc(k, sizeof(lnode*)), *t=
  if(n>k) {
     for(i=0; i<k; i++) {
          elems= i<n%k ? n/K+1 : n/k;
          j=0;
        list[i] =head;
        t= head;
  while(j++ <elems) {
       t=head;
       head =head->next;
  }
  t->next =NULL;
 }
}else { for( i=0; i<n; i++) {
           list[i]=head;
         head =head->next;
       list[i]->next =NULL;
 }
}
return list;
}
```

12/24

```
parentptr->right = ptr;
}
printf("\n Enter the element :");
scanf("%d", &val);
}
}
void pre(struct node *tree)
{ if(tree != NULL)
  { printf("%d\t", tree->data);
  pre(tree->left);
    pre(tree->right);
}
void in(struct node *tree)
{ if(tree != NULL)
  { in(tree->left);
    printf("%d\t", tree->data);
     in(tree->right);
  }
}
void post(struct node *tree)
{ if(tree != NULL)
  { post(tree->left);
    post(tree->right);
    print("%d\t", tree->data);
}
}
```

PAGE NO:
DATE:

```
case 4: printf("\n The elements in the tree are"
        in(tree); break;
}
}while(option!=5);
}
void create()
{ int val;
  printf("\n Enter -1 to end");
  printf("\n Enter the element: ");
  scanf("%d", &val);
  while(val!=-1)
  { struct node *ptr, *nodeptr, *parentptr;
    ptr = (struct node*) malloc(sizeof(struct node));
    ptr->data = val;
    ptr->left = NULL;
     ptr->right = NULL;
  if(tree ==NULL)
    { tree =ptr;
      tree->left = NULL;
      tree->right = NULL;
  }
  else
  { parentptr = NULL;
    nodeptr = tree;
     while(nodeptr!=NULL)
    { parentptr = nodeptr;
      if(val < nodeptr->data)
        nodeptr = nodeptr->left;
    else
      nodeptr = nodeptr->right;
    }
  if(val < parentptr->data)
     parentptr->right = ptr;
  } else
```

Leet Code [Rotate List]

```
struct ListNode *rotateRight(struct ListNode* head, int k){
struct ListNode *ptr, *ptr1;
  int count =0, num;

if(head ==NULL || head->next==NULL) {
      return head;

}
 ptr = head;
 while(ptr ->next != NULL) {
      count ++;
      ptr = ptr->next;
 }
 num = k % (count +1);
 while(num--){
      ptr = head;
      while(ptr->next != NULL){
        ptr1 = ptr;
        ptr = ptr->next;
  }
   ptr->next= head;
   ptr1 ->next = NULL;
     head = ptr;
 }
return head;
}
```

PAGE NO:
DATE:

Output:
Create a binary search tree
Preorder traversal
Postorder traversal
Inorder traversal
Exit

Enter an option: 1
Enter -1 to end

Enter the element: 8 1 5 3 9 4 6 7
-1
Binary search tree created.

Enter an option: 2

The ~~ele~~ elements in the tree are
8 1 5 3 4 6 7 8

Enter an option: 3

The elements in the tree are
4 3 7 6 5 1 9 8

Enter an option: 4

The elements in the tree are
1 3 4 5 6 7 8 9
Enter an option: 5

PAGE NO:
DATE:

DS Lab Week-9

WAP to traverse a graph using BFS method & to check whether given graph is connected or not using DFS method

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_NODES 100
#define MAX_EDGES 100

int graph[MAX_NODES][MAX_NODES];
int visited[MAX_NODES];

void DFS(int start, int n) {
    visited[start]= 1;
for(int i=0; i<n; i++) {
        if(graph[start][i] == 1 && !visited[i]) {
            DFS(i,n);
        }
    }
}

int isConnected(int n) {
    DFS(0,n);

for(int i=0; i<n; i++) {
        if(!visited[i]) {
                return 0;
        }
    }
  return 1;
}
```

Hackerrank :

functions ⇒

void inOrderTraversal(Node* root, int* result, int* index)
{ if(root == NULL) return;
inOrderTraversal(root->left, result, index);

result[(*index)++] = root->data;

inOrderTraversal(root->right, result, index);
}

void swapAtlevel(Node* root, int k, int level) {
if (root == NULL) return;
~~inOrderTraversal(root->left, result, index);~~
~~result[(*index)++] = root->data;~~
if(level % k == 0) {
Node *temp = root->left;
root->left = root->right;
root->right = temp;
}

PAGE NO :
DATE :

```
int main()
{ int n, m;
  printf("Enter the number of nodes and edges:");
  scanf("%d %d", &n, &m);

  printf("Enter the edges:\n");
  for(int i=0; i<m; i++) {
     int a, b;
     scanf("%d %d", &n, &m);

   printf("Enter the edges:\n");
  for(int i=0; i<m; i++)
   { int a, b;
      scanf("%d %d", &a, &b);
      graph[a][b]=1;
      graph[b][a]=1;
   }

if (isConnected(n)){
          printf("The graph is connected.\n");
 }
else {
   printf("The graph is not connected\n");
 }
return 0;
}
```

Output:
BFS traversal : 0 1 4 2 3
Graph is connected

PAGE NO :
DATE :

Week-10

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 the keys 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 using linear probing

```
#include<stdio.h>
#include<stdio.h>
#define TABLE_SIZE 10

struct EmployeeRecord{
        int key;
};
struct EmployeeRecord *hashTable[TABLE_SIZE];

int hashFunction(int key){
        return key % TABLE_SIZE;
}

void insert(struct EmployeeRecord *record){
    int key = record -> key;
    int index = hashFunction(key);
    int i=0;

    While(i< TABLE_SIZE){
```

PAGE NO :
DATE :

```
if(hashTable[index] == NULL) {
    hashTable[index] = record;
    printf("Inserted record with key %d at index
        %d\n", key, index);
    return;
}
i++;
index = (hashFunction(key) + i) % TABLE_SIZE
}
printf("HashTable is full. Unable to insert record
    with key %d\n", key);
}

struct EmployeeRecord *search(int key) {
    int index = hashFunction(key);
    int i = 0;
while(i<TABLE_SIZE) {
    if(hashTable[index] != NULL && hashTable[index]->k
    { printf("Record with key %d found at index
      %d\n", key, index);
  return hashTable[index];
    }
    i++;
  index = (hashFunction(key) + i) % TABLE_SIZE;
    return NULL;
}

int main() {
    for (int i=0; i<TABLE_SIZE; i++) {
        hashTable[i] = NULL;
    }
```

PAGE NO :
DATE :

```
struct EmployeeRecord record1 = {1234};
struct EmployeeRecord record2 = {5678};

insert(&record1);
insert(&record2);

search(1234);
search(5678);
search(9724);

return 0;
}
```

Output:

Inserted record with key 1234 at index 4
Inserted record with key 5678 at index 8
Record with key 1234 found at index 4
Record with key 5678 found at index 8
Record with key 9999 not found in the HashTable

29/2/24