PROGRAM 1 : Calendar

a.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct CalendarDay {
char* dayName;
int date;
char* activity;
};
int main() {
struct CalendarDay calendar[7];
char *d[] = {"Sunday","Monday","Tuesday","Wednesday","Thursday",
"Friday","Saturday"};
char *a[] = {"Movie with friends", "Play-Indoor", "Outdoor", "Trip to Mysore", "Sleep hard", "Work work work", "Gaming"};
for (int i = 0; i < 7; i++) {
calendar[i].dayName = (char*)malloc(20 * sizeof(char));
calendar[i].activity = (char*)malloc(100 * sizeof(char));
calendar[i].date = i + 1;
strcpy(calendar[i].dayName, d[i]);
strcpy(calendar[i].activity, a[i]);
}
for (int i = 0; i < 7; i++) {
printf("Day %d: %s, Date: %d, Activity: %s\n", i + 1, calendar[i].dayName,
calendar[i].date, calendar[i].activity);
}
for (int i = 0; i < 7; i++) {
free(calendar[i].dayName);
free(calendar[i].activity);
}
return 0;
}
```

B.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct CalendarDay {
char* dayName;
int date;
char* activity;
} calendar[7];

void create() {
    for (int i = 0; i < 7; i++) {
    calendar[i].dayName = (char*)malloc(20 * sizeof(char));
    calendar[i].activity = (char*)malloc(100 * sizeof(char));
    }
}

void read() {
    for (int i = 0; i < 7; i++) {
        printf("Enter the day name for Day %d: ", i + 1);
        scanf("%s", calendar[i].dayName);
        printf("Enter the date for Day %d: ", i + 1);
        scanf("%d", &calendar[i].date);
        printf("Enter the activity description for Day %d: ", i + 1);
        scanf(" %99[^\n]", calendar[i].activity);
    }
  }

void display() {
    printf("\nWeekly Activity Details:\n");
    for (int i = 0; i < 7; i++)
 printf("Day %d: %s, Date: %d, Activity: %s\n", i + 1, calendar[i].dayName, calendar[i].date, calendar[i].activity);
 }
int main() {


create();
read();
display();
for (int i = 0; i < 7; i++) {
    free(calendar[i].dayName);
    free(calendar[i].activity);
}
return 0;
}
```

PROGRAM 2: String Operations

```
#include<stdio.h>
#include<string.h>
char str[100], pat[50], rep[50], ans[100]; int i, j, c, m, k, flag=0;

void stringmatch(){
   i = m = c = j = 0;
   while(str[c]!='\0')
   {
      if(str[m]==pat[i])
      {
         i++;
         m++;
         if(pat[i]=='\0')
         {
            flag= 1;
            for(k = 0; rep[k] != '\0'; k++, j++)
            ans[j] = rep[k];
            i = 0;
            c = m;
         }
      }
      else
      {
         ans[j] = str[c]; j++;
         c++;
```

```
            m = c;
            i = 0;
        }
    }
    ans[j] = '\0';
}
void main() {
    printf("\nEnter a main string \n");
    gets(str);
    printf("\nEnter a pattern string \n");
    gets(pat);
    printf("\nEnter a replace string \n");
    gets(rep);
    stringmatch();
    if(flag==1)
    printf("\nThe resultant string is\n %s" , ans);
    else
    printf("\nPattern string NOT found\n");
}
```

PROGRAM 3: Stack Operations

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 4

int stack[MAX], item,ch, top = -1, count = 0;

void push( ){

   if (top == (MAX-1))

      printf("\n\nStack is Overflow");

   else

      stack[++top] = item;

  }


int pop( ){

   int ret;

   if(top == -1)

      printf("\n\nStack is Underflow");

   else{

      ret = stack[top--];

      printf("\nPopped element is %d", ret);

   }

   return ret;

 }


void palindrome( ){

   int i, j, flag=1;

   for(i=0, j=top;i<j; i++, j--){
        if(stack[i] ==stack[j])
            flag = 1;
        else{
            flag =0;
            break;
        }
    }
    if(flag==1)
        printf("\nStack contents are Palindrome");
    else
        printf("\nStack contents are not palindrome");
 }

void display( ){
    int i;
    printf("\nThe stack contents are:");
    if(top == -1)
        printf("\nStack is Empty");
    else{
        for(i=top; i>=0; i--)
        printf("\n ------\n| %d |", stack[i]); printf("\n");
    }
}
```

```
void main( ){
    int temp;
    do{
        printf("\n\n----MAIN MENU----\n");
        printf("\n1. PUSH (Insert) in the Stack");
        printf("\n2. POP (Delete) from the Stack");
        printf("\n3. PALINDROME check using Stack");
        printf("\n4. Exit (End the Execution)");
        printf("\nEnter Your Choice: ");
        scanf("%d", &ch);
        switch(ch){
            case 1: printf("\nEnter a element to be pushed: ");
                    scanf("%d", &item);
                    push( );
                    display( );
                    break;
            case 2: temp=pop( );
                    display( );
                    break;
            case 3: palindrome( );
                    break;
            case 4: exit(0);
            default: printf("\nEND OF EXECUTION");
        }
    } while (ch != 4);
}
```

PROGRAM 4: Infix to Postfix

```
#include<stdio.h>

#include<string.h>

int F(char symbol){

    switch(symbol){

        case '+' :

        case '-': return 2;

        case '*':

        case '/': return 4;

        case '^':

        case '$': return 5;

        case '(': return 0;

        case '#': return -1;

        default: return 8;

  }

}

int G(char symbol){

    switch(symbol){

        case '+':

        case '-': return 1;

        case '*':

        case '/': return 3;

        case '^':

        case '$': return 6;

        case '(': return 9;

        case ')': return 0;
        default: return 7;
    }
}
void infix_postfix(char infix[], char postfix[]){
    int top, j, i;
    char s[30], symbol;
    top = -1;
    s[++top] = '#';
    j = 0;
    for(i=0; i < strlen(infix); i++){
        symbol = infix[i];
        while(F(s[top]) > G(symbol)){
            postfix[j] = s[top--];
            j++;
        }
        if(F(s[top])!=G(symbol))
            s[++top] = symbol;
        else
        top--;
    }
    while(s[top]!='#')
        postfix[j++] = s[top--];
    postfix[j] = '\0';
}
void main(){
    char infix[20], postfix[20];
    printf("\nEnter a valid infix expression\n");

    gets(infix);

    infix_postfix(infix,postfix);

    printf("\nThe infix expression is:\n");

    printf ("%s",infix);

    printf("\nThe postfix expression is:\n");

    printf ("%s",postfix);

}
```

PROGRAM 5: Postfix evaluation and Tower

a.

```
#include<stdio.h>

#include<math.h>

#include<string.h>

double compute(char symbol, double op1, double op2){

   switch(symbol){

      case '+': return op1 + op2;

      case '-': return op1 - op2;

      case '*': return op1 * op2;

      case '/': return op1 / op2;

      case '$':

      case '^': return pow(op1,op2);

      default: return 0;

   }

}

void main(){

   double s[20], res, op1, op2;

   int top, i;

   char postfix[20], symbol;

   printf("\nEnter the postfix expression:\n");

   gets(postfix);

   top=-1;

   for(i=0; i<strlen(postfix); i++){

      symbol = postfix[i];
```

```
        if(isdigit(symbol))
            s[++top] = symbol - '0';
        else{
            op2 = s[top--];
            op1 = s[top--];
            res = compute(symbol, op1, op2);
            s[++top] = res;
        }
    }
    res = s[top--];
    printf("\nThe result is : %f\n", res);
}
```

b.

```
#include<stdio.h>
#include<math.h>
void tower(int n, int source, int temp,int destination){
    if(n == 0) return;
    tower(n-1, source, destination, temp);
    printf("\nMove disc %d from %c to %c", n, source, destination);
    tower(n-1, temp, source, destination);
}

void main(){
    int n;
    printf("\nEnter the number of discs: \n");
    scanf("%d", &n);
    tower(n, 'A', 'B', 'C');
    printf("\n\nTotal Number of moves are: %d", (int)pow(2,n)-1);
}
```

PROGRAM 6: Circular Q

```
#include<stdio.h>

#include<stdlib.h>

#define MAX 4

int ch, front = 0, rear = -1, count=0;

char q[MAX], item;

void insert(){

   if(count == MAX)

      printf("\nQueue is Full");

   else{

      rear = (rear + 1) % MAX;

      q[rear]=item;

      count++;

   }

}


void del(){

   if(count == 0)

      printf("\nQueue is Empty");

   else{

      if(front>rear && rear==MAX-1){

         front=0;

         rear=-1;

         count=0;

      }

      else{
            item=q[front];
            printf("\nDeleted item is: %c",item);
            front = (front + 1) % MAX;
            count--;
        }
    }
}

void display(){
    int i, f=front, r=rear;
    if(count == 0)
        printf("\nQueue is Empty");
    else{
        printf("\nContents of Queue is:\n");
        for(i=f; i!=r; i=(i+1)%MAX)
            printf("%c  ",q[i]);
        printf("%c",q[i]);
    }
}

void main(){
    do{
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit");
        printf("\nEnter the choice: ");
        scanf("%d", &ch);
        switch(ch){
            case 1: printf("\nEnter the character / item to be inserted: ");
                    scanf(" %c",&item);
                    insert();
                    break;
            case 2: del(); break;
            case 3: display(); break;
            case 4: exit(0); break;
        }
    } while(ch!=4);
}
```

PROGRAM 7: Singly Linked List

```
#include<stdio.h>

#include<stdlib.h>

int MAX=4, count;

struct student{

   char usn[10];

   char name[30];

   char branch[5];

   int sem;

   char phno[10];

   struct student *next;

};

typedef struct student NODE;

NODE *head;


int countnodes(){

   NODE *p;

   count=0;

   p=head;

   while(p!=NULL)

   {

      p=p->next;

      count++;

   }

   return count;

}
```

```
NODE* getnode(){
   NODE *newnode;
   newnode= (NODE*)malloc(sizeof(NODE));
   printf("\nEnter USN, Name, Branch, Sem, Ph.No\n");
   scanf("%s",newnode->usn);
   scanf("%s",newnode->name);
   scanf("%s",newnode->branch);
   scanf("%d",&(newnode->sem));
   scanf("%s",newnode->phno);
   newnode->next=NULL;
   return newnode;
}

NODE* display(){
   NODE *p;
   if(head == NULL)
      printf("\nNo student data\n");
   else{
      p=head;
      printf("\n----STUDENT DATA----\n");
      printf("\nUSN\tNAME\t\tBRANCH\tSEM\tPh.NO.");
      while(p!=NULL){
         printf("\n%s\t%s\t\t%s\t%d\t%s", p->usn, p->name, p->branch, p->sem,p->phno);
         p = p->next;
      }
      printf("\nThe no. of nodes in list is: %d",countnodes(head));
   }
   return head;
}

NODE* create(){
   NODE *newnode;
   if(head==NULL){
      newnode=getnode();
      head=newnode;
   }
   else{
      newnode=getnode();
      newnode->next=head;
      head=newnode;
   }
   return head;
}

void insert_front(){
   if(countnodes(head)==MAX)
      printf("\nList is Full / Overflow!!");
   else
      head=create( );
}
void delete_front(){
    NODE *p;
    if(head==NULL)
        printf("\nList is Empty/Underflow (STACK/QUEUE)");
    else{
        p=head;
        head=head->next;
        free(p);
        printf("\nFront(first)node is deleted");
    }
}
void main(){
    int ch, i, n;
    head=NULL;
    printf("\n*----------Studednt Database-----------*");
    do
    {
        printf("\n 1.Create\t 2.Display\t 3.Insert_f\t 4.Delete_f\t 5.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch){
            case 1: printf("\nHow many student data you want to create: ");
                    scanf("%d", &n);
                    for(i=0;i<n;i++)
                        create();
                    break;
            case 2: display();
```

```
                break;
            case 3: insert_front();
                break;
            case 4: delete_front();
                break;
            case 5: exit(0);
        }
    } while(ch!=7);
}
```

PROGRAM 8: Doubly Linked List

```
#include<stdio.h>
#include<stdlib.h>
int MAX=4, count;

struct emp{
int ssn,sal;
char name[20],dept[10],desig[15],phno[10];
struct emp *left;
struct emp *right;
};
typedef struct emp NODE;

int countnodes(NODE *head){
   NODE *p;
   count=0;
   p=head;
   while(p!=NULL){
      p=p->right;
      count++;
   }
   return count;
}

NODE* getnode(NODE *head){
   NODE *newnode;
```

```
    newnode=(NODE*)malloc(sizeof(NODE));
    newnode->right=newnode->left=NULL;
    printf("\nEnter SSN, Name, Dept, Designation, Sal, Ph.No\n");
    scanf("%d",&newnode->ssn);
    scanf("%s",newnode->name);
    scanf("%s",newnode->dept);
    scanf("%s",newnode->desig);
    scanf("%d",&newnode->sal);
    scanf("%s",newnode->phno);
    head=newnode;
    return head;
}

NODE* display(NODE *head){
    NODE *p;
    if(head==NULL)
        printf("\nNo Employee data\n");
    else{
        p=head;
        printf("\n----EMPLOYEE DATA----\n");
        printf("\nSSN\tNAME\tDEPT\tDESINGATION\tSAL\t\tPh.NO.");
        while(p!=NULL){
            printf("\n%d\t%s\t%s\t%s\t\t%d\t\t%s", p->ssn, p->name, p->dept,p->desig,p->sal, p-
>phno);
            p = p->right;
        }
        printf("\nThe no. of nodes in list is: %d",countnodes(head));
    }
    return head;
}

NODE* create(NODE *head){
    NODE *p, *newnode;
    p=head;
    if(head==NULL){
        newnode=getnode(head);
        head=newnode;
    }
    else{
        newnode=getnode(head);
        while(p->right!=NULL)
            p=p->right;
        p->right=newnode;
        newnode->left=p;
    }
    return head;
}

NODE* insert_end(NODE *head){
    if(countnodes(head)==MAX)
        printf("\nList is Full!!");
    else
        head=create(head);
    return head;
}



NODE* insert_front(NODE *head){
    NODE *p, *newnode;
    if(countnodes(head)==MAX)
    printf("\nList is Full!!");
    else{
    if(head==NULL){
        newnode=getnode(head);
        head=newnode;
    }
    else{
        newnode=getnode(head);
        newnode->right=head;
        head->left=newnode;
        head=newnode;
    }
  }
    return head;
}

NODE* insert(NODE *head){
    int ch;
    do{
        printf("\n 1.Insert at Front(First) \t 2.Insert at End(Rear/Last)\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch){
        case 1: head=insert_front(head);
                break;
        case 2: head=insert_end(head);
                break;
        case 3: break;
        }
        head=display(head);
    } while(ch!=3);
    return head;

}

NODE* delete_front(NODE *head){
    NODE *p;
    if(head==NULL)
        printf("\nList is Empty (QUEUE)");
    else{
        p=head;
        head=head->right;
        head->right->left=NULL;
        free(p);
        printf("\nFront(first)node is deleted");
    }
    return head;
}

NODE* delete_end(NODE *head){
    NODE *p, *q;
    p=head;
    while(p->right!=NULL)
        p=p->right;
    q=p->left;
    q->right=NULL;
    p->left=NULL;
    free(p);
    printf("\nLast(end) entry is deleted");
    return head;
}

NODE *del(NODE *head){
    int ch;
    do {
        printf("\n1.Delete from Front(First)\t2. Delete from End(Rear/Last))\t3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch){
            case 1: head=delete_front(head);
                    break;
            case 2: head=delete_end(head);
                    break;
            case 3: break;
        }
        head=display(head);
    } while(ch!=3);
    return head;
}

NODE* queue(NODE *head){
    int ch, ch1, ch2;
    do{
        printf("\nDLL used as Double Ended Queue");
        printf("\n1.QUEUE- Insert at Rear & Delete from Front");
        printf("\n2.QUEUE- Insert at Front & Delete from Rear");
        printf("\n3.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch){
            case 1: do{
                    printf("\n1.Insert at Rear\t2.Delete from From Front\t3.Exit");
                    printf("\nEnter your choice: ");
                    scanf("%d", &ch1);
                    switch(ch1){
                        case 1: head=insert_end(head); break;
                        case 2: head=delete_front(head); break;
```

```
                         case 3: break;
                     }
                      } while(ch1!=3);
                     break;

             case 2: do{
                         printf("\n1.Insert at Front\t2.Delete from Rear\t3.Exit");
                         printf("\nEnter your choice: ");
                         scanf("%d", &ch2);
                         switch(ch2){
                             case 1: head=insert_front(head); break;
                             case 2: head=delete_end(head); break;
                             case 3: break;
                         }
                       }while(ch2!=3);
                     break;

             case 3: break;
         }

    } while(ch!=3);

    head=display(head);
    return head;
}
```

```
void main(){
    int ch, i, n;
    NODE *head;
    head=NULL;
    printf("\n----------Employee Database-----------");
    do{
        printf("\n1.Create\t2.Display\t3.Insert\t4.Delete\t5.Queue\t6.Exit"); printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch){
            case 1: printf("\nHow many employees data you want to create: ");
                    scanf("%d", &n);
                    for(i=0;i<n;i++)
                    head=create(head);
                    break;
            case 2: head=display(head);
                    break;
            case 3: head=insert(head);
                    break;
            case 4: head=del(head);
                    break;
            case 5: head=queue(head);
                    break;
            case 6: exit(0);
        }
    } while(ch!=6); }
```

PROGRAM 9: Polynomial (SCLL)

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

struct node{
    int cf, px, py, pz,flag;
    struct node *link;
};
typedef struct node NODE;

NODE* getnode(){
    NODE *x;
    x=(NODE*)malloc(sizeof(NODE));
    if(x==NULL){
        printf("Insufficient memory\n");
        exit(0);
    }
    return x;
}

void display(NODE *head){
    NODE *temp;
    if(head->link==head){
        printf("Polynomial does not exist\n");
        return;
    }

    temp=head->link;

    printf("\n");

    while(temp!=head){

        printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp->pz);

        if(temp->link != head)

            printf(" + ");

        temp=temp->link;

        }

        printf("\n");

}


NODE* insert_rear(int cf,int x,int y,int z,NODE *head){

            NODE *temp,*cur;

            temp=getnode();

            temp->cf=cf;

            temp->px=x;

            temp->py=y;

            temp->pz=z;

            temp->flag=0;

            cur=head->link;

            while(cur->link!=head)

                cur=cur->link;

            cur->link=temp;

            temp->link=head;

            return head;
```

```
}

NODE* read_poly(NODE *head){
            int px, py, pz, cf, ch;
            do{
        printf("\nEnter coeff: ");
        scanf("%d",&cf);
        printf("\nEnter x, y, z powers(0-indiacate NO term): ");
        scanf("%d%d%d", &px, &py, &pz);
        head=insert_rear(cf,px,py,pz,head);
        printf("\nIf you wish to continue press 1 otherwCSE 0: ");
        scanf("%d", &ch);
        } while(ch!=0);
        return head;
}

NODE* add_poly(NODE *h1,NODE *h2,NODE *h3){
    NODE *p1,*p2;
    int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
    p1=h1->link;
    while(p1!=h1){
        x1=p1->px;
        y1=p1->py;
        z1=p1->pz;
        cf1=p1->cf;
        p2=h2->link;
    while(p2!=h2){
        x2=p2->px;
        y2=p2->py;
        z2=p2->pz;
        cf2=p2->cf;
        if(x1==x2 && y1==y2 && z1==z2)
        break;
        p2=p2->link;
    }
        if(p2!=h2){
            cf=cf1+cf2;
            p2->flag=1;
            if(cf!=0)
                h3=insert_rear(cf,x1,y1,z1,h3);
        }
        else
            h3=insert_rear(cf1,x1,y1,z1,h3);
            p1=p1->link;
    }
    p2=h2->link;
    while(p2!=h2){
    if(p2->flag==0)
        h3=insert_rear(p2->cf,p2->px,p2->py,p2->pz,h3);
    p2=p2->link;
    }
return h3;
```

```
}

void evaluate(NODE *h1){
          NODE *head;
          int x, y, z;
          float result=0.0;
   head=h1;
   printf("\nEnter x, y, z, terms to evaluate:\n");
   scanf("%d%d%d", &x, &y, &z);
   while(h1->link != head){
      h1=h1->link;
      result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
      h1=h1->link;
    }
   result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) * pow(z,h1->pz));
   printf("\nPolynomial result is: %f", result);
}

void main(){
   NODE*h1,*h2,*h3;
   int ch;
   h1=getnode();
   h2=getnode();
   h3=getnode();
   h1->link=h1;
   h2->link=h2;
```

```
h3->link=h3;

while(1){

    printf("\n\n1.Evaluate polynomial\n2.Add two polynomials\n3.Exit\n");

    printf("Enter your choice: ");

    scanf("%d", &ch);

    switch(ch){

        case 1: printf("\nEnter polynomial to evaluate:\n");

                h1=read_poly(h1);

                display(h1);

                evaluate(h1);

                h1->link=h1;

                break;

        case 2: printf("\nEnter the first polynomial:");

                h1=read_poly(h1);

                printf("\nEnter the second polynomial:"); h2=read_poly(h2);

                h3=add_poly(h1,h2,h3);

                printf("\nFirst polynomial is: ");

                display(h1);

                printf("\nSecond polynomial is: ");

                display(h2);

                printf("\nThe sum of 2 polynomials is: ");

                display(h3);

                h1->link=h1;

                h2->link=h2;

                h3->link=h3;

                break;
```

```
            case 3: exit(0);

            default: printf("\nInvalid entry");

                    break;

        }

    }

}
```

```
    }
```

PROGRAM 10: Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
struct BST
{
    int data;
    struct BST *left;
    struct BST *right;
};
typedef struct BST NODE;
NODE *node;

NODE* createtree(NODE *node, int data)
{
    if (node == NULL)
    {
        NODE *temp;
        temp= (NODE*)malloc(sizeof(NODE));
        temp->data = data;
        temp->left = temp->right = NULL;
        return temp;
    }
    if (data < (node->data))
    {
        node->left = createtree(node->left, data);
    }
    else if (data > node->data)

    {

        node -> right = createtree(node->right, data);

    }

    return node;

}


NODE* search(NODE *node, int data)

{

    if(node == NULL)

    printf("\nElement not found");

    else if(data < node->data)

    {

        node->left=search(node->left, data);

    }

    else if(data > node->data)

    {

        node->right=search(node->right, data);

    }

    else

        printf("\nElement found is: %d", node->data);

        return node;

    }


void inorder(NODE *node)

{
    if(node != NULL)
    {
        inorder(node->left);
        printf("%d\t", node->data);
        inorder(node->right);
    }
}

void preorder(NODE *node)
{
    if(node != NULL)
    {
        printf("%d\t", node->data);
        preorder(node->left);
        preorder(node->right);
    }
}
void postorder(NODE *node)
{
    if(node != NULL)
    {
        postorder(node->left);
        postorder(node->right);
        printf("%d\t", node->data);
    }
}
```

```
void main(){
    int data, ch, i, n;
    NODE *root=NULL;
    while (1)
    {
        printf("\n1.Insertion in Binary Search Tree");
        printf("\n2.Search Element in Binary Search Tree");
  printf("\n3.Inorder\n4.Preorder\n5.Postorder\n6.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1: printf("\nEnter N value: " );
                scanf("%d", &n);
        printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                for(i=0; i<n; i++)
                {
                    scanf("%d", &data);
                    root=createtree(root, data);
                }
                break;
            case 2: printf("\nEnter the element to search: ");
                scanf("%d", &data);
                root=search(root, data);
                break;
            case 3: printf("\nInorder Traversal: \n");
                inorder(root);
                break;
            case 4: printf("\nPreorder Traversal: \n");
                preorder(root);
                break;
            case 5: printf("\nPostorder Traversal: \n");
                postorder(root);
                break;
            case 6: exit(0);
            default:printf("\nWrong option");
                break;
        }
    }
}
```

```
}
```

PROGRAM 11: Graph Operations

```
#include <stdio.h>

#include <stdlib.h>

struct BST

{

    int data;

    struct BST *left;

    struct BST *right;

};

typedef struct BST NODE;

NODE *node;


NODE* createtree(NODE *node, int data)

{

    if (node == NULL)

    {

        NODE *temp;

        temp= (NODE*)malloc(sizeof(NODE));

        temp->data = data;

        temp->left = temp->right = NULL;

        return temp;

    }

    if (data < (node->data))

    {

        node->left = createtree(node->left, data);

    }
```

```
        else if (data > node->data)
        {
                node -> right = createtree(node->right, data);
        }
        return node;
}

NODE* search(NODE *node, int data)
{
        if(node == NULL)
        printf("\nElement not found");
        else if(data < node->data)
        {
                node->left=search(node->left, data);
        }
        else if(data > node->data)
        {
                node->right=search(node->right, data);
        }
        else
                printf("\nElement found is: %d", node->data);
                return node;
        }

void inorder(NODE *node)
{
        if(node != NULL)
        {
                inorder(node->left);
                printf("%d\t", node->data);
                inorder(node->right);
        }
}

void preorder(NODE *node)
{
        if(node != NULL)
        {
                printf("%d\t", node->data);
                preorder(node->left);
                preorder(node->right);
        }
}
void postorder(NODE *node)
{
        if(node != NULL)
        {
                postorder(node->left);
                postorder(node->right);
                printf("%d\t", node->data);
        }
}
```

```
void main(){
    int data, ch, i, n;
    NODE *root=NULL;
    while (1)
    {
        printf("\n1.Insertion in Binary Search Tree");
        printf("\n2.Search Element in Binary Search Tree");
  printf("\n3.Inorder\n4.Preorder\n5.Postorder\n6.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1: printf("\nEnter N value: " );
                scanf("%d", &n);
        printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
                for(i=0; i<n; i++)
                {
                    scanf("%d", &data);
                    root=createtree(root, data);
                }
                break;
            case 2: printf("\nEnter the element to search: ");
                scanf("%d", &data);
                root=search(root, data);

                break;

            case 3: printf("\nInorder Traversal: \n");

                inorder(root);

                break;

            case 4: printf("\nPreorder Traversal: \n");

                preorder(root);

                break;

            case 5: printf("\nPostorder Traversal: \n");

                postorder(root);

                break;

            case 6: exit(0);

            default:printf("\nWrong option");

                break;

        }

    }

}
```

PROGRAM 12: Hash Table

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 10


struct employee

{

	int id;

	char name[15];

};

typedef struct employee EMP;


EMP emp[MAX];

int a[MAX];


int create(int num){

	int key;

	key = num % 10;

	return key;

}


int getemp(EMP emp[],int key){

	printf("\nEnter emp id: ");

	scanf("%d",&emp[key].id);

	printf("\nEnter emp name: ");

   scanf("%s",emp[key].name);
```

```
    return key;
}

void display(){
    int i, ch;
    printf("\n1.Display ALL\n2.Filtered Display");
    printf("\nEnter the choice: ");
    scanf("%d",&ch);
    if(ch == 1){
        printf("\nThe hash table is:\n");
        printf("\nHTKey\tEmpID\tEmpName");
        for(i=0; i<MAX; i++)
           printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
    }

    else
    {
        printf("\nThe hash table is:\n");
        printf("\nHTKey\tEmpID\tEmpName");
        for(i=0; i<MAX; i++)
        if(a[i] != -1){
            printf("\n%d\t%d\t%s", i, emp[i].id, emp[i].name);
            continue;
        }
    }
}
```

```
void linear_prob(int key,int num){
    int flag,i;
    flag=0;
    for(i=key+1;i<=MAX;i++){
        if(a[i]==-1){
            a[i]=getemp(emp,i);
            flag=1;
            break;
        }
    }
    for(i=0;i<key&&flag==0;i++){
         if(a[i]==-1){
            a[i]=getemp(emp,i);
            flag=1;
            break;
        }
    }
    if(!flag)
        printf("hash tabl full:");
}

void main(){
    int num,key,i;
    int ans=1;
    printf("\nCollision handling by linear probing:");
    for(i=0;i<MAX;i++){
        a[i]=-1;
        printf("%d",a[i]);
    }
    do{
        printf("\nEnter the data:\n");
        scanf("%d",&num);
        key=create(num);
        if(a[key]==-1)
            a[key]=getemp(emp,key);
        else{
            printf("collision detected\n");
            linear_prob(key,num);
        }
        printf("\nDo you wish to continue?(1/0): ");
        scanf("%d",&ans);
    } while(ans);
    display();
}
```