Singly Linked List (Program 7)

```
#include <stdio.h>

#include<stdlib.h>

#include<string.h>

struct node

{

    char usn[15];

    char name[20];

    char pgm[10];

    int sem;

    long int phno;

    struct node *next;

};

typedef struct node NODE;

NODE *first=NULL,*cur=NULL,*last=NULL;

struct node * create()

{    struct node *temp;

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

    printf("Enter the usn\t");

    scanf("%s",temp->usn);

    printf("Enter the name\t");

    scanf("%s",temp->name);

    printf("Enter the pgm\t");

    scanf("%s",temp->pgm);

    printf("Enter the sem\t");

    scanf("%d",&temp->sem);

    printf("Enter the phno\t");

    scanf("%ld",&temp->phno);

    temp->next=NULL;

    return temp;

}

void display()
```

```
{    struct node *temp;
     int count=0;
     if(first==NULL)
 {
     printf("Empty list\n");
     return;
 }
     temp=first;
     while(temp!=NULL)
     {
         count++;
         printf("Student %d",count);
         printf("usn is %s\n",temp->usn);
         printf("name is %s\n",temp->name);
         printf("pgm is %s\n",temp->pgm);
         printf("sem is %d\n",temp->sem);
         printf("phno is %ld\n",temp->phno);
         temp=temp->next;
     }
     printf("There are %d students",count);
}
void insert_end()
{   struct node *temp;
    if(first==NULL)
    {


        first=create();
    }
    else
    {
        temp=create();
        cur=first;
        while(cur->next!=NULL)
          cur=cur->next;
        cur->next=temp;
    }
}
void delete_end()
{
    if(first==NULL)
    {
        printf("Empty list\n");
        return;
    }
    if(first->next==NULL)
    {
        free(first);
        first=NULL;
        return;
    }
    else
    {
        last=first;
        cur=first;
        while(cur->next!=NULL)
        {
            last=cur;
            cur=cur->next;
        }
        last->next=NULL;
        free(cur);
    }
}
void insert_front()
```

```
{    struct node *temp;
    if(first==NULL)
    {

       first=create();
    }
    else
    {
        temp=create();
        temp->next=first;
        first=temp;
    }
}
void delete_front()
{   struct node *temp;
    if(first==NULL)
    {
        printf("Empty list\n");
        return;
    }
    else
    {
        temp=first;
        first=first->next;
        free(temp);
    }
}
void main()
{
    int choice,n,i;
    while(1)
    {
        printf("\n\nList Operations\n\n");
        printf("\n1.Create list of n students\n");
        printf("\n2.Display status and count\n");
        printf("\n3.Insertion at front\n");
        printf("\n4.Delete at front\n");
        printf("\n5.Insert at end\n");
        printf("\n6.Delete at end\n");
        printf("\n7.Exit\n");
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:printf("Enter the value of n\n");
                scanf("%d",&n);
                for(i=1;i<=n;i++)
                {
                  insert_front();
                }
                break;
          case 2:display();
                break;
          case 3:insert_front();
                break;
          case 4:delete_front();
                break;
          case 5:insert_end();
                break;
          case 6:delete_end();
                break;
          case 7:exit(1);
        }
    }
}
```

Doubly Linked List (Program 8)

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>


struct node

{

 char ssn[10];

 char name[15];

 char dept[10];

 char desgn[15];

 long int sal,phno;

 struct node *prev,*next;

};

struct node *f=NULL,*l=NULL,*r=NULL,*rlink,*llink;

struct node *create();

void insertend();

void insertfront();

void deleteend();

void deletefront();

void display();

void main()

{

 int choice,i,n;

 while(1)

 {

  printf("\n 1-create \n 2-display \n 3-insertend \n 4-deleteend \n 5-insertfront \n 6-deletefront \n 7-
dequeue \n 8-exit \n");

  scanf("%d",&choice);

  switch(choice)

  {
   case 1:printf("enter the no of employee\n");
          scanf("%d",&n);
          for(i=1;i<=n;i++)
            insertfront();
           break;
   case 2:display();
           break;
   case 3:insertend();
           break;
   case 4:deleteend();
           break;
   case 5:insertfront();
           break;
   case 6:deletefront();
           break;
   case 7:printf("since insertion and deletion can be done from both end it works as a double ended
queue\n");
   case 8:exit(0);
  }
 }
}

struct node *create()
{
 struct node *temp;
 temp=(struct node*)malloc(sizeof(struct node));
 temp->next=NULL;
 temp->prev=NULL;
 printf("enter ssn \n");
 scanf("%s",temp->ssn);
 printf("enter name \n");
 scanf("%s",temp->name);
 printf("enter dept \n");
 scanf("%s",temp->dept);
 printf("enter desgn\n");
 scanf("%s",temp->desgn);
 printf("enter salary\n");
 scanf("%ld",&temp->sal);
 printf("enter phno \n");
 scanf("%ld",&temp->phno);
 return temp;
}

void insertend()
{
 struct node *temp;
 temp=create();
 if(f==NULL)
 {
  f=temp;
  l=temp;
  return;
 }
 else
 {
  l->next=temp;
  temp->prev=l;
  l=temp;
 }
}

void insertfront()
{
 struct node *temp;
 temp=create();
 if(f==NULL)
 {
  f=temp;
  l=temp;
  return;
 }
 else
 {
  temp->next=f;
  f->prev=temp;
  f=temp;
 }
}

void deleteend()
{
 struct node *temp;
 if(f==NULL)
 {
  printf("empty list\n");
  return;
 }
 temp=l;
 if(f==l)
 {
  f=NULL;
  l=NULL;
  free(temp);
 }
 else
 {
  l=l->prev;
  l->next=NULL;
  free(temp);
 }
}

void deletefront()
{
 struct node *temp;
 if(f==NULL)
 {
  printf("empty list\n");
  return;
 }
 temp=f;
 if(f==l)
 {
  f=NULL;
  l=NULL;
  free(temp);
 }
 else
 {
  f=f->next;
  f->prev=NULL;
  free(temp);
 }
}

void display()
{
 struct node *temp;
 if(f==NULL)
```

```
{
 printf("the list is empty\n");
 return;
}
 printf("elements in the forward direction\n");
 for(temp=f;temp;temp=temp->next)
 printf("\n %10s %10s %10s %10s %ld %ld",temp->ssn,temp->name,temp->dept,temp->desgn,temp-
>sal,temp->phno);
 printf("\n");
 printf("elements in the backward direction\n");
 for(temp=l;temp;temp=temp->prev)
 printf("\n %10s %10s %10s %10s %ld %ld",temp->ssn,temp->name,temp->dept,temp->desgn,temp-
>sal,temp->phno);
}
```

Circular Linked List-Polynomial (Program 9)

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


#define TRUE 1

#define FALSE 0

#define COMPARE(x, y) ((x) > (y) ? 1 : ((x) == (y) ? 0 : -1))


struct polynode {

   int coeff;

   int expo;

   struct polynode *link;

};


typedef struct polynode *polyptr;

polyptr heada, headb, headc;


void display(polyptr x) {

   polyptr temp;

   temp = x->link;

   while (temp->link != x) {

      printf("%d x^%d + ", temp->coeff, temp->expo);

      temp = temp->link;

   }

   printf("%d x^%d", temp->coeff, temp->expo);

}


void attach(int c, int e, polyptr *ptr) {

   polyptr temp;

   temp = (polyptr)malloc(sizeof(struct polynode));

   temp->coeff = c;
```

```
    temp->expo = e;
    (*ptr)->link = temp;
    *ptr = temp;
}

void cpadd(polyptr a, polyptr b) {
    polyptr starta, lastc;
    int sum, done = FALSE;
    starta = a;
    a = a->link;
    b = b->link;
    headc = (polyptr)malloc(sizeof(struct polynode));
    headc->expo = -1;
    lastc = headc;

    do {
        switch (COMPARE(a->expo, b->expo)) {
            case 1:
                attach(a->coeff, a->expo, &lastc);
                a = a->link;
                break;
            case -1:
                attach(b->coeff, b->expo, &lastc);
                b = b->link;
                break;
            case 0:
                if (a == starta)
                    done = TRUE;
                else {
                    sum = a->coeff + b->coeff;
                    if (sum)
                        attach(sum, a->expo, &lastc);
                a = a->link;
                b = b->link;
            }
            break;
        }
    } while (!done);
    lastc->link = headc;
}

void main() {
    polyptr lasta, lastb, temp;
    int c, e, i, n, x, choice, sum = 0;

    heada = (polyptr)malloc(sizeof(struct polynode));
    headb = (polyptr)malloc(sizeof(struct polynode));
    heada->expo = -1;
    headb->expo = -1;
    lasta = heada;
    lastb = headb;

    printf("\nEnter the number of terms of polynomial 1\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("Enter coefficient and exponent: ");
        scanf("%d %d", &c, &e);
        attach(c, e, &lasta);
    }
    lasta->link = heada;

    printf("\nEnter the number of terms of polynomial 2\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("Enter coefficient and exponent: ");
        scanf("%d %d", &c, &e);
        attach(c, e, &lastb);
    }
    lastb->link = headb;

    printf("1 - Add\n2 - Evaluate\n");
    scanf("%d", &choice);

    if (choice == 1) {
        cpadd(heada, headb);
        printf("\nPolynomial 1 is\n");
        display(heada);
        printf("\n\nPolynomial 2 is\n");
        display(headb);
        printf("\n\nResult is\n");
        display(headc);
    } else if (choice == 2) {
        printf("Enter x value: ");
        scanf("%d", &x);
        for (temp = heada->link; temp != heada; temp = temp->link)
            sum += temp->coeff * pow(x, temp->expo);
        printf("\nPolynomial A after evaluation is %d\n", sum);
    }
}
```

}

BST (Program 10)

```
#include <stdio.h>

#include <stdlib.h>


struct node {

    int data;

    struct node* left;

    struct node* right;

};


typedef struct node* treeptr;


treeptr create(int x) {

    treeptr nn = (treeptr)malloc(sizeof(struct node));

    nn->data = x;

    nn->left = NULL;

    nn->right = NULL;

    return nn;

}


treeptr insert(treeptr root, int x) {

    if (root == NULL) {

        return create(x);

    }

    if (x < root->data) {

        root->left = insert(root->left, x);

    } else if (x > root->data) {

        root->right = insert(root->right, x);

    }

    return root;

}


void inorder(treeptr root) {
```

```c
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

void preorder(treeptr root) {
    if (root) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

void postorder(treeptr root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

void search(treeptr root, int x) {
    if (root == NULL) {
        printf("Key %d not found in the tree.\n", x);
        return;
    }
    if (root->data == x) {
        printf("Key %d found in the tree.\n", x);
        return;
    }
    if (x < root->data) {
        search(root->left, x);
    } else {
        search(root->right, x);
    }
}

void main() {
    treeptr root = NULL;
    int choice, x;

    while (1) {
        printf("\nMenu:\n1. Insert\n2. Display Inorder Traversal\n3. Display Preorder Traversal\n4.
Display Postorder Traversal\n5. Search\n6. Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &x);
                root = insert(root, x);
                break;

            case 2:
                printf("Inorder traversal: ");
                inorder(root);
                printf("\n");
                break;

            case 3:
                printf("Preorder traversal: ");
                preorder(root);
                printf("\n");
                break;

            case 4:
                printf("Postorder traversal: ");
                postorder(root);
                printf("\n");
                break;

            case 5:
                printf("Enter value to search: ");
                scanf("%d", &x);
                search(root, x);
                break;

            case 6:
                exit(0);

            default:
                printf("Invalid choice. Try again.\n");
        }
    }
}
```

DFS(Program 11)

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct node {
    struct node *link;
    int vertex;
} node;
node *G[20];
int visited[20];
int n;

void insert(int vi, int vj) {
    node *p, *q;
    q = (node*)malloc(sizeof(node));
    q->vertex = vj;
    q->link = NULL;
     if (G[vi] == NULL)
        G[vi] = q;
    else {
        for (p = G[vi]; p->link != NULL; p = p->link);
        p->link = q;
    }
}

void read_graph() {
    int i, vi, vj, no_of_edges;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
     // Initialize graph
    for (i = 0; i < n; i++)
```

```
        G[i] = NULL;
    printf("Enter number of edges: ");
    scanf("%d", &no_of_edges);
    for (i = 0; i < no_of_edges; i++) {
        printf("Enter an edge (u v): ");
        scanf("%d %d", &vi, &vj);
        insert(vi, vj);
        insert(vj, vi);

    }
}

void DFS(int i) {
    node *p;
    printf("%5d", i);
    visited[i] = TRUE;
    for (p = G[i]; p; p = p->link) {
        if (!visited[p->vertex])
            DFS(p->vertex);
    }
}

int main() {
    int i;
    read_graph();
    for (i = 0; i < n; i++)
        visited[i] = FALSE;
    printf("\nNodes visited in DFS order:\n");
    DFS(0);
return 0;
}
```

Hashing-Linear Probing(Program 12)

```
#include<stdio.h>
#define max 5
#define mod(x) x%max
void linearprobe(int a[],int num,int key)
 {
   int i;
   if(a[key]==-1)
     {
       a[key]=num;
     }
   else
     {

       printf("\ncollision detected\n");
       for(i=mod(key+1); i!=key ; i= mod(++i))
        {
         if(a[i]==-1)
           break;
        }

       if(i!=key)
         {
           a[i]=num;
           printf("\nCollisssion avoided and inserted the element successfully\n");
         }
        else
          printf("hash table is full");
      }
  }

  void display(int a[])
```

```
{
  int ch,i;
   printf("\n 1.Filtered display\n2. display all\n enter choice\n");
   scanf("%d",&ch);
   printf("\nHash table is:\n");
   for(i=0;i<max;i++)
   {
     if(a[i]>0 || ch-1)
      printf("%d %d\n",i,a[i]);
   }
}

void main()
{
  int a[max],num,i;
  printf("Collision handling by linear probing\n");
  for(i=0;i<max;a[i++]=-1);
  do
  {
    printf("enter the data");
    scanf("%d",&num);
    linearprobe(a,num,mod(num));
    printf("want to continue 1/0 \n");
    scanf("%d",&i);
  }while(i);

  display(a);
}
```