Program 1. STRING OPERATIONS

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

int search(char p[],char t[])
{
      int n, m, i, j;
      n = strlen(t);
      m = strlen(p);

      for(i=0; i<=n-m; i++)
      {
            j=0;
            while(j<m && p[j] == t[i+j])
            {
                  j++;
            }

            if (j==m) return i;
      }
      return -1;
}

void replace(char p[] ,char t[] ,char r[] , int pos)
{
      int i,k;
      char d[30];
      for(k=0;k<pos;k++) d[k] = t[k];
      for(i=0;i<strlen(r);i++) d[k++] = r[i];
      pos += strlen(p);
      for(i=pos;i<=strlen(t);i++) d[k++] = t[i];
      for(i=0;i<=strlen(d);i++) t[i] = d[i];
}

void main()
{
      char t[30],p[30],r[30];
      int pos;
      printf("\nSTR: "); scanf("%[^\n]",t);
      printf("PAT: "); scanf("%s",p);
      printf("REP: "); scanf("%s",r);
      pos = search(p,t);
      if(pos == -1)
      {
            printf("Pattern String not found!!\n\n");
            return;
      }

      for(;;)
      {
            replace(p,t,r,pos);
```

```
            pos = search(p,t);
            if(pos == -1) break;
      }

      printf("FINAL : %s\n\n",t);
}
```

Program 2. STACK OPERATIONS

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

#define STACK_SIZE 10

int stack[10];

void push(int item, int stack[], int *top)
{
      if (*top == STACK_SIZE - 1)
      {
            printf("Stack overflow\n");
            return;
      }
      stack[++(*top)] = item;
}

void pop(int *top, int stack[])
{
      if (*top == -1)
      {
            printf("Stack underflow\n");
            return;
      }
      printf("Item deleted = %d\n", stack[(*top)--]);
}

void display(int top, int stack[])
{
      if (top == -1)
      {
            printf("Stack is empty\n");
            return;
      }
      printf("Stack: ");
      for (int i = 0; i <= top; i++)
            printf("%d ", stack[i]);
```

```
      printf("\n");
}

void palindrome(char str[],int top)
{
      int i;
      for(i=0;str[i]!='\0';i++)
      stack[++top] = str[i];
      for(i=0;str[i]!='\0';i++)
      {
            if(str[i] == stack[top--]) continue;
            printf("%s : Is not a Palindrome\n",str);
            return;
      }
      printf("%s : is a Palindrome\n",str);
}

int main()
{
      int choice, item, top = -1;
      char str[10];
      while (1)
      {
            printf("1.Push\n2.Pop\n3.Display\n4.Is Palindrome or
not ?\n5.Exit\n");
            printf("Enter your choice: ");
            scanf("%d", &choice);

            switch (choice)
            {
                  case 1:
                        printf("Enter item to push: ");
                        scanf("%d", &item);
                        push(item, stack, &top);
                        break;
                  case 2:
                        pop(&top, stack);
                        break;
                  case 3:
                        display(top, stack);
                        break;
                  case 4:
                        printf("Enter a string: ");
                        scanf(" %[^\n]",str);

                        palindrome(str,top);
                        break;
                  default:
                        exit(0);
            }
      }
      return 0;
}
```

Program 3. INFIX TO POSTFIX

```
#include<stdio.h>

int F(char symbol)
{
      switch(symbol)
      {
            case '#': return -1;

            case '+':
            case '-': return 2;

            case '*':
            case '/': return 4;

            case '^':
            case '$': return 5;

            case '(': return 0;

            default: return 8;
      }
}

int G(char symbol)
{
      switch(symbol)
      {
            case ')': return 0;

            case '+':
            case '-': return 1;

            case '*':
            case '/': return 3;

            case '^':
            case '$': return 6;

            case '(': return 9;

            default : return 7;
      }
}

void infix_2_postix(char infix[], char postfix[])
{
```

```
	int i, j=0, top = -1;
	char s[20];
	s[++top] = '#';
	for(i=0; infix[i] != '\0'; i++)
	{
		while(F(s[top]) > G(infix[i]))
		{
			postfix[j++] = s[top--];
		}
		if(F(s[top]) != G(infix[i]))
			s[++top] = infix[i];
		else
			top--;
	}

	while(s[top] != '#')
	{
		postfix[j++] = s[top--];
	}
	postfix[j] = '\0';
}

void main()
{
	char infix[50], postfix[50];
	printf("Enter the infix expession : ");
	scanf("%s",infix);
	infix_2_postix(infix,postfix);
	printf("Postfix Expression is : %s\n",postfix);
}
```

Program 4

A) STACK APPPLICATIONS

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

double compute(double op1, char op, double op2)
{
	switch(op)
	{
		case '+': return op1+op2;

		case '-': return op1-op2;

		case '*': return op1*op2;
```

```
            case '/': return op1/op2;

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

double evaluate(char postfix[])
{
      int i, top =-1;
      double stack[20], op1, op2;
      for(i=0; postfix[i] != '\0'; i++)
      {
            if(postfix[i] >= '0' && postfix[i] < '9')
                  stack[++top] = postfix[i] - '0';
            else
            {
                  op2 = stack[top--];
                  op1 = stack[top--];
                  stack[++top] = compute(op1, postfix[i], op2);
            }
      }
      return stack[top--];
}

void main()
{
      char postfix[20];
      double result;
      printf("Enter the postfix expression : ");
      scanf("%s",postfix);
      result = evaluate(postfix);
      printf("Result : %lf\n",result);
}
```

B) TOWER OF HANOI

```
#include<stdio.h>

void transfer(int n, char source, char temp, char destination)
{
      if(n == 0) return;
      transfer(n-1, source, destination, temp);
      printf("Move disk %d from %c to %c\n",n,source,destination);
      transfer(n-1, temp, source, destination);
}

void main()
{
      int n;
```

```
	printf("Enter the number of disks : ");
	scanf("%d",&n);
	transfer(n,'A','B','C');
}
```

Program 5. CIRCULAR QUEUE

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

#define Q_SIZE 5

void display(int queue[], int front, int count)
{
	int i, temp;
	if(count == 0)
	{
		printf("Queue is empty\n");
		return;
	}
	printf("QUEUE : ");
	temp = front;
	for(i=0; i<count; i++)
	{
		printf("%d ",queue[temp]);
		temp = (temp+1) % Q_SIZE;
	}
	printf("\n");
}

void insert_rear(int item, int queue[], int *rear, int *count)
{
	if(*count == Q_SIZE)
	{
		printf("Queue is full\n");
		return;
	}
	*rear = (*rear + 1) % Q_SIZE;
	queue[*rear] = item;
	(*count)++;
}

void delete_front(int queue[], int *front, int *count)
{
	if(*count == 0)
	{
		printf("Queue is empty\n");
```

```
            return;
      }
      printf("Item deleted : %d\n",queue[*front]);
      (*front) = (*front + 1) % Q_SIZE;
      (*count)--;
}

void main()
{
      int choice, item, queue[10], front = 0, rear = -1, count = 0;
      for(;;)
      {
            printf("1.Insert\n2.Delete\n3.Display\n4.Exit\nEnter yur
choice : ");
            scanf("%d",&choice);

            switch(choice)
            {
                  case 1:
                        printf("Enter the item : ");
                        scanf("%d",&item);
                        insert_rear(item, queue, &rear, &count);
                        break;
                  case 2:
                        delete_front(queue, &front, &count);
                        break;
                  case 3:
                        display(queue, front, count);
                        break;
                  default:
                        exit(0);
            }
      }
}
```

Program 6. SINGLY LINKED LIST

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

struct node
{
      int sem;
      char phone[50];
      char usn[50];
      char name[50];
```

```
      char prog[50];
      struct node* link;
};

typedef struct node* NODE;

NODE createNode()
{
      NODE newNode = (NODE)malloc(sizeof(struct node));
      if(newNode == NULL)
      {
            printf("Memory allocation failed\n\n");
            exit(0);
      }
      newNode->link = NULL;
      return newNode;
}

void read_student_details(char usn[], char name[], char prog[], int *sem,
char phone[])
{
      printf("Enter the student details\n");
      printf("USN : "); scanf("%s", usn);
      printf("Name : "); scanf("%s", name);
      printf("Program : "); scanf("%s", prog);
      printf("SEM : "); scanf("%d", sem);
      printf("Phone No. : "); scanf("%s", phone);
}

NODE insert_front(char usn[], char name[], char prog[], int sem, char
phone[], NODE first)
{
      NODE newNode = createNode();
      strcpy(newNode->usn, usn);
      strcpy(newNode->name, name);
      strcpy(newNode->prog, prog);
      strcpy(newNode->phone, phone);
      newNode->sem = sem;
      newNode->link = first;
      return newNode;
}

NODE delete_front(NODE first)
{
      NODE cur = first;
      if(first == NULL)
      {
            printf("\nStudent list is empty\n\n");
            return NULL;
      }
      printf("\nStudent : %s's details deleted\n\n", first->name);
      first = first->link;
      free(cur);
      return first;
```

```
}

NODE insert_rear(NODE first, char usn[], char name[], char prog[], int
sem, char phone[])
{
      NODE newNode = createNode();
      strcpy(newNode->usn, usn);
      strcpy(newNode->name, name);
      strcpy(newNode->prog, prog);
      strcpy(newNode->phone, phone);
      newNode->sem = sem;
      if(first == NULL)
      {
            return newNode;
      }
      NODE cur = first;
      while(cur->link != NULL)
      {
            cur = cur->link;
      }
      cur->link = newNode;
      return first;
}

NODE delete_rear(NODE first)
{
      NODE cur = first, prev = NULL;
      if(first == NULL)
      {
            printf("\nList is empty\n\n");
            return NULL;
      }
      while(cur->link != NULL)
      {
            prev = cur;
            cur = cur->link;
      }
      if(prev == NULL)
      {
            free(first);
            return NULL;
      }
      prev->link = NULL;
      printf("\nStudent : %s's details deleted\n\n", cur->name);
      free(cur);
      return first;
}

int count_node(NODE first)
{
      int count = 0;
      NODE temp = first;
      while(temp != NULL)
      {
```

```
            count++;
            temp = temp->link;
      }
      return count;
}

void display_list(NODE first)
{
      NODE temp = first;
      if(first == NULL)
      {
            printf("\nList is empty\n\n");
            return;
      }
      printf("STUDENT LIST\n");
      printf("%-10s %-15s %-10s %-5s %-15s\n", "USN", "NAME", "PROGRAM",
"SEM", "PHONE NO.");
      while(temp != NULL)
      {
            printf("%-10s %-15s %-10s %-5d %-15s\n", temp->usn, temp-
>name, temp->prog, temp->sem, temp->phone);
            temp = temp->link;
      }
      printf("\n");
}

void main()
{
      char usn[50], name[50], prog[50], phone[50];
      int sem, choice, count = 0;
      NODE first = NULL;
      for(;;)
      {
            printf("1.Insert at Front\n2.Insert at Rear\n3.Delete at
Front\n4.Delete at Rear\n5.Display\n6.Count no. of Nodes\n7.Exit\n");
            printf("Enter your choice: ");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1:
                        read_student_details(usn, name, prog, &sem,
phone);
                        first = insert_front(usn, name, prog, sem, phone,
first);
                        break;
                  case 2:
                        read_student_details(usn, name, prog, &sem,
phone);
                        first = insert_rear(first, usn, name, prog, sem,
phone);
                        break;
                  case 3:
                        first = delete_front(first);
                        break;
```

```
                case 4:
                      first = delete_rear(first);
                      break;
                case 5:
                      display_list(first);
                      break;
                case 6:
                      count = count_node(first);
                      printf("The number of students in the list
is: %d\n\n", count);
                      break;
                case 7:
                      exit(0);
                default:
                      printf("Invalid choice! Please try again.\n\n");
            }
      }
}
```

Program 7. DOUBLY LINKED LIST

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

typedef struct
{
      char ssn[20];
      char name[50];
      char department[20];
      char designation[20];
      char phone[20];
      float salary;
}EMPLOYEE;

struct node
{
      char ssn[20];
      char name[50];
      char department[20];
      char designation[20];
      char phone[20];
      float salary;
      struct node* llink;
      struct node* rlink;
};
```

```
typedef struct node* NODE;

NODE getNode()
{
      NODE temp = (NODE)malloc(sizeof(struct node));
      if(!temp)
      {
            printf("Memory allocation failed\n");
            return NULL;
      }
      temp->llink = temp;
      temp->rlink = temp;
      return temp;
}

void dl_display(NODE head)
{
      if(head->rlink == head)
      {
            printf("List is empty\n");
            return;
      }
      printf("EMPLOYEE LIST : \n");
      printf("%-10s %-15s %-15s %-15s %-15s %-10s\n", "SSN", "NAME",
"DEPARTMENT", "DESIGNATION", "PHONE NO.", "SALARY");
      NODE cur = head->rlink;
      while(cur != head)
      {
            printf("%-10s %-15s %-15s %-15s %-15s %-10f\n",cur->ssn, cur-
>name, cur->department, cur->designation, cur->phone, cur->salary);
            cur = cur->rlink;
      }
      printf("NULL\n");
}

int count_node(NODE head)
{
      int count = 0;
      NODE temp = head->rlink;
      if(head->rlink == head) return 0;
      while(temp != head)
      {
            count++;
            temp = temp->rlink;
      }
      return count;
}

NODE dl_insert_rear(EMPLOYEE emp, NODE head)
{
      NODE temp = getNode();
      NODE last;

      strcpy(temp->ssn, emp.ssn);
```

```
	strcpy(temp->name, emp.name);
	strcpy(temp->department, emp.department);
	strcpy(temp->designation, emp.designation);
	strcpy(temp->phone, emp.phone);
	temp->salary = emp.salary;
	last = head->llink;
	temp->llink = last;
	last->rlink = temp;
	temp->rlink = head;
	head->llink = temp;

	return head;
}

NODE dl_insert_front(EMPLOYEE emp, NODE head)
{
	NODE temp = getNode();
	NODE first = NULL;
	strcpy(temp->ssn, emp.ssn);
	strcpy(temp->name, emp.name);
	strcpy(temp->department, emp.department);
	strcpy(temp->designation, emp.designation);
	strcpy(temp->phone, emp.phone);
	temp->salary = emp.salary;
	first = head->rlink;
	first->llink = temp;
	temp->rlink = first;
	head->rlink = temp;
	temp->llink = head;
	return head;
}

NODE dl_delete_rear(NODE head)
{
	NODE last, prev;
	if(head->rlink == head)
	{
		printf("List is empty\n");
		return head;
	}
	last = head->llink;
	prev = last->llink;
	prev->rlink = head;
	head->llink = prev;
	printf("Details of Employee having SSN(%s)\n", last->ssn);
	free(last);
	return head;
}

NODE dl_delete_front(NODE head)
{
	NODE first, second;
	if(head->rlink == head)
	{
```

```
            printf("List is empty\n");
            return head;
      }
      first = head->rlink;
      second = first->rlink;
      head->rlink = second;
      second->llink = head;
      printf("Details of Employee having SSN(%s)\n", first->ssn);
      free(first);
      return head;
}

void read_employee_details(EMPLOYEE *emp)
{
      printf("Enter the student details\n");
      printf("SSN : ");
      scanf("%s", emp->ssn);
      printf("Name : ");
      scanf(" %[^\n]", emp->name);
      printf("Department : ");
      scanf(" %[^\n]", emp->department);
      printf("Designation : ");
      scanf(" %[^\n]", emp->designation);
      printf("Phone No. : ");
      scanf("%s", emp->phone);
      printf("Salary : ");
      scanf("%f",&emp->salary);
}

void main()
{
      int choice,count;
      NODE head = getNode();
      EMPLOYEE emp;
      for(;;)
      {
            printf("1. Insert at rear\n2. Insert at front\n3. Delete at
rear\n4. Delete at front\n5. Display\n6. Count\n7. Exit\nEnter your
choice: ");
            scanf("%d", &choice);
            switch(choice)
            {
                  case 1:
                        read_employee_details(&emp);
                        head = dl_insert_rear(emp, head);
                        break;
                  case 2:
                        read_employee_details(&emp);
                        head = dl_insert_front(emp, head);
                        break;
                  case 3:
                        head = dl_delete_rear(head);
                        break;
                  case 4:
```

```
                        head = dl_delete_front(head);
                        break;
                  case 5:
                        dl_display(head);
                        break;
                  case 6:
                        count = count_node(head);
                        printf("Number of Employees : %d\n",count);
                        break;
                  case 7:
                        exit(0);
                  default:
                        printf("Invalid choice !!!\n");
                        break;
            }
      }
}
```

Program 8. SINGLY CIRCULAR LIST

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

struct node
{
      int c,px,py,pz;
      struct node* link;
};

typedef struct node* NODE;

NODE getNode()
{
      NODE temp = (NODE)malloc(sizeof(struct node));
      if(!temp)
      {
            printf("\nMemory allocation failed\n");
            return NULL;
      }
      temp->link = temp;
      return temp;
}

float evaluate(float x, float y, float z, NODE head)
{
      float sum = 0;
```

```
	NODE cur = head->link;
	while(cur != head)
	{
		sum = sum + cur->c * pow(x,cur->px) * pow(y,cur->py) *
pow(z,cur->pz);
		cur = cur->link;
	}
	return sum;
}

NODE insert_rear(NODE head, int c, int px, int py, int pz)
{
	NODE cur,temp;
	temp = getNode();
	temp->c = c;
	temp->px = px;
	temp->py = py;
	temp->pz = pz;
	cur = head->link;
	while(cur->link!=head)
		cur = cur->link;
	cur->link = temp;
	temp->link = head;
	return head;
}

NODE read_poly()
{
	NODE head = getNode();
	int c,px,py,pz;
	printf("Enter the coefficient and power of x,y,z : ");
	for(;;)
	{
		scanf("%d",&c);
		if(c==0) break;
		scanf("%d %d %d",&px,&py,&pz);
		head = insert_rear(head,c,px,py,pz);
	}
	return head;
}

NODE add_2_poly(NODE h1, NODE h2)
{
	int sum;
	NODE h3= getNode();
	NODE p,q;
	for(p=h1->link; p!=h1; p=p->link)
	{
		for(q=h2->link; q!=h2; q=q->link)
		{
			if((p->px == q->px) && (p->py == q->py) && (p->pz == q-
>pz))
			{
				sum = p->c + q->c;
```

```
                        if(sum!=0)
                        {
                              h3 = insert_rear(h3,sum,p->px,p->py,p->pz);
                        }
                        q->c = 0;
                        break;
                  }
            }
            if(q==h2)
            {
                  h3 = insert_rear(h3,p->c,p->px,p->py,p->pz);
            }
      }
      for (q = h2->link; q != h2; q = q->link)
      {
            if (q->c != 0)
            {
                  h3 = insert_rear(h3, q->c, q->px,q->py,q->pz);
            }
      }
      return h3;
}

void display(NODE head)
{
      NODE temp = head->link;
      while(temp!=head)
      {
            if(temp->c > 0)
            {
                  printf("+%dx^%d y^%d z^%d ",temp->c,temp->px,temp-
>py,temp->pz);
            }
            else
            {
                  printf("%dx^%d y^%d z^%d ",temp->c,temp->px,temp-
>py,temp->pz);
            }
            temp = temp->link;
      }
      printf("\n");
}

void main()
{
      NODE head1,head2,head3;
      int choice;
      float x,y,z,sum;
      for(;;)
      {
            printf("1.Add\n2.Evaluate\n3.Exit\nEnter your choice : ");
            scanf("%d",&choice);
            switch(choice)
            {
```

```
                case 1:
                        printf("Enter the terms of 1st Polynomial : \n");
                        head1 = read_poly();
                        printf("Enter the terms of 2nd Polynomial : \n");
                        head2 = read_poly();
                        printf("Polynomial 1 : ");
                        display(head1);
                        printf("\nPolynomial 2 : ");
                        display(head2);
                        head3 = add_2_poly(head1,head2);
                        printf("\nAddition of 2 polynomials is : ");
                        display(head3);
                        break;
                case 2:
                        printf("Enter a polynomial : ");
                        head1 = read_poly();
                        printf("Enter values of x,y and z : ");
                        scanf("%f %f %f",&x,&y,&z);
                        sum = evaluate(x,y,z,head1);
                        printf("Result : %f\n",sum);
                        break;
                case 3: exit(0);
                default:
                        printf("Invalid choice !!\n");
            }
        }
}
```

Program 9. CALENDER

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

#define NO_OF_DAYS 7

typedef struct
{
      char *day;
      int date;
      char *activity;
}CALENDER;

void create_calender(CALENDER a[], int i, char day[], int date, char
activity[])
{
      a[i].day = (char*)malloc(strlen(day) + 1);
```

```
      a[i].activity = (char*)malloc(strlen(activity) + 1);
      strcpy(a[i].day, day);
      a[i].date = date;
      strcpy(a[i].activity, activity);
}

void read_calender(CALENDER a[])
{
      int i, date;
      char day[10], activity[10];
      for(i=0; i<NO_OF_DAYS; i++)
      {
            scanf("%s",day);
            scanf("%d",&date);
            scanf("%s",activity);
            create_calender(a, i, day, date, activity);
      }
}

void print_weeks_activity(CALENDER a[])
{
      printf("Weeks Activity\n");
      for(int i=0; i<NO_OF_DAYS; i++)
      printf("%-10s : %s\n",a[i].day, a[i].activity);
}

void main()
{
      CALENDER a[NO_OF_DAYS];
      printf("DAY\tDATE\tACTIVITY\n");
      read_calender(a);
      print_weeks_activity(a);
}
```

Program 10. Binary Search Tree

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

struct BST
{
      int data;
      struct BST *lchild;
      struct BST *rchild;
};

typedef struct BST * NODE;
NODE create()
```

```
{
      NODE temp;
      temp = (NODE) malloc(sizeof(struct BST));
      printf("\nEnter The value: ");
      scanf("%d", &temp->data);
      temp->lchild = NULL;
      temp->rchild = NULL;
      return temp;
}
void insert(NODE root, NODE newnode);
void inorder(NODE root);
void preorder(NODE root);
void postorder(NODE root);
void search(NODE root);

void insert(NODE root, NODE newnode)
{
      if (newnode->data < root->data)
      {
            if (root->lchild == NULL)
                  root->lchild = newnode;
            else
                  insert(root->lchild, newnode);
      }
      if (newnode->data > root->data)
      {
            if (root->rchild == NULL)
                  root->rchild = newnode;
            else
                  insert(root->rchild, newnode);
      }
}

void search(NODE root)
{
      int key;
      NODE cur;
      if(root == NULL)
      {
            printf("\nBST is empty.");
            return;
      }
      printf("\nEnter Element to be searched: ");
      scanf("%d", &key);
      cur = root;
      while (cur != NULL)
      {
            if (cur->data == key)
            {
                  printf("\nKey element is present in BST");
                  return;
            }
            if (key < cur->data)
                  cur = cur->lchild;
```

```
            else
                  cur = cur->rchild;
      }
      printf("\nKey element is not found in the BST");
}

void inorder(NODE root)
{
      if(root != NULL)
      {
            inorder(root->lchild);
            printf("%d ", root->data);
            inorder(root->rchild);
      }
}

void preorder(NODE root)
{
      if (root != NULL)
      {
            printf("%d ", root->data);
            preorder(root->lchild);
            preorder(root->rchild);
      }
}

void postorder(NODE root)
{
      if (root != NULL)
      {
            postorder(root->lchild);
            postorder(root->rchild);
            printf("%d ", root->data);
      }
}

void main()
{
      int ch, key, val, i, n;
      NODE root = NULL, newnode;
      while(1)
      {
            printf("\n\n~~~~BST MENU~~~~");
            printf("\n1.Create a BST");
            printf("\n2.BST Traversals : ");
            printf("\n3.Search : ");
            printf("\n4.Exit");
            printf("\nEnter your choice : ");
            scanf("%d", &ch);
            switch(ch)
            {
                  case 1:
                        printf("\nEnter the number of elements:");
                        scanf("%d", &n);
```

```
                        for(i=1; i<=n; i++)
                        {
                              newnode = create();
                              if (root == NULL)
                                    root = newnode;
                              else
                                    insert(root, newnode);
                        }
                        break;
                  case 2:
                        if (root == NULL)
                              printf("\nTree Is Not Created");
                        else
                        {
                              printf("\nThe Preorder display : ");
                              preorder(root);
                              printf("\n\nThe Inorder display : ");
                              inorder(root);
                              printf("\n\nThe Postorder display :");
                              postorder(root);
                        }
                        break;
                  case 3:
                        search(root);
                        break;
                  case 4:
                        exit(0);
            }
      }
}
```

Program 11. Graphs

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

int a[50][50], n, visited[50];
int q[20], front = -1, rear = -1;
int s[20], top = -1, count = 0;

void bfs(int v)
{
      int i, cur;
      visited[v] = 1;
      q[++rear] = v;
      while (front != rear)
      {
```

```
            cur = q[++front];
            for (i = 1; i <= n; i++)
            {
                  if ((a[cur][i] == 1) && (visited[i] == 0))
                  {
                        q[++rear] = i;
                        visited[i] = 1;
                        printf("%d ", i);
                  }
            }
      }
}

void dfs(int v)
{
      int i;
      visited[v] = 1;
      s[++top] = v;
      for (i = 1; i <= n; i++)
      {
            if (a[v][i] == 1 && visited[i] == 0)
            {
                  printf("%d ", i);
                  dfs(i);
            }
      }
}

int main()
{
      int ch, start, i, j;
      printf("\nEnter the number of vertices in graph:");
      scanf("%d", & n);
      printf("\nEnter the adjacency matrix:\n");
      for (i = 1; i <= n; i++)
      {
            for (j = 1; j <= n; j++)
                  scanf("%d", & a[i][j]);
      }
      for (i = 1; i <= n; i++)
            visited[i] = 0;
      printf("\nEnter the starting vertex: ");
      scanf("%d", & start);
      printf("\n==>1. BFS: Print all nodes reachable from a given
starting node");
      printf("\n==>2. DFS: Print all nodes reachable from a given
starting node");
      printf("\n==>3:Exit");
      printf("\nEnter your choice: ");
      scanf("%d", & ch);

      switch (ch)
      {
            case 1:
```

```
                  printf("\nNodes reachable from starting vertex %d are:
", start);
                  bfs(start);
                  for (i = 1; i <= n; i++)
                  {
                        if (visited[i] == 0)
                              printf("\nThe vertex that is not
                              reachable is %d", i);
                  }
                  break;
            case 2:
                  printf("\nNodes reachable from starting vertex %d
are:\n", start);
                  dfs(start);
                  break;
            case 3:
                  exit(0);
            default:
                  printf("\nPlease enter valid choice:");
      }
}
```

Program 12. Hashing

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

int key[20], n, m;
int * ht, index;
int count = 0;

void insert(int key)
{
      index = key % m;
      while (ht[index] != -1)
      {
            index = (index + 1) % m;
      }
      ht[index] = key;
      count++;
}

void display()
{
      int i;
```

```
      if (count == 0)
      {
            printf("\nHash Table is empty");
            return;
      }
      printf("\nHash Table contents are:\n ");
      for (i = 0; i < m; i++)
            printf("\n T[%d] --> %d ", i, ht[i]);
}

void main()
{
      int i;
      printf("\nEnter the number of employee records(N): ");
      scanf("%d", & n);
      printf("\nEnter the two digit memory locations(m) for hash table:
");
      scanf("%d", & m);
      ht = (int * ) malloc(m * sizeof(int));
      for (i = 0; i < m; i++)
            ht[i] = -1;
      printf("\nEnter the four digit key values (K)
      for N Employee Records:\n ");
      for (i = 0; i < n; i++)
            scanf("%d", & key[i]);
      for (i = 0; i < n; i++)
      {
            if (count == m)
            {
                  printf("\n-----Hash table is full. Cannot insert the
record %d key-----", i + 1);
                  break;
            }
            insert(key[i]);
      }
      display();
}
```