Program 1

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

struct date {
    int day;
    int month;
    int year;
};

struct calendar {
    char dayname[10];
    struct date d;
    char activity[25];
};

typedef struct calendar* CALENDAR;
CALENDAR mycal;

void create(), read(), display();

void main() {
    create();
    read();
    display();
}

void create() {
    mycal = (CALENDAR)malloc(sizeof(struct calendar) * 7);
}

void read() {
    for (int i = 0; i < 7; i++) {
        printf("Enter the week activity details for day %d:\n", i+1);
        __fpurge(stdin);
        printf("Enter day name: ");
        scanf("%s", mycal[i].dayname);
        printf("Enter date (DD MM YYYY): ");
        scanf("%d%d%d", &mycal[i].d.day, &mycal[i].d.month, &mycal[i].d.year);
        printf("Enter day activity: ");
        scanf("%s", mycal[i].activity);
    }
}

void display() {
    printf("\nWeekly Activity Report:\n");
    for (int i = 0; i < 7; i++) {
        printf("Day Name: %s\n", mycal[i].dayname);
```

```
        printf("Date: %02d/%02d/%04d\n", mycal[i].d.day, mycal[i].d.month,
mycal[i].d.year);
        printf("Activity: %s\n", mycal[i].activity);
    }
}
```

Program 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // For strcspn

void read(char str[], char pat[], char rep_pat[]);
void replace(char str[], char pat[], char rep_pat[], char new_str[]);

int main() {
    char str[100], pat[20], rep_pat[20], new_str[100];
    read(str, pat, rep_pat);
    replace(str, pat, rep_pat, new_str);
    return 0;
}

void read(char str[], char pat[], char rep_pat[]) {
    printf("Enter the string: ");
    fgets(str, 100, stdin);
    str[strcspn(str, "\n")] = '\0';  // Remove trailing newline

    printf("Enter the pattern to be replaced: ");
    fgets(pat, 20, stdin);
    pat[strcspn(pat, "\n")] = '\0';  // Remove trailing newline

    printf("Enter the replacement string: ");
    fgets(rep_pat, 20, stdin);
    rep_pat[strcspn(rep_pat, "\n")] = '\0';  // Remove trailing newline
}

void replace(char str[], char pat[], char rep_pat[], char new_str[]) {
    int i = 0, j = 0, k, rep_ind, flag = 0, mflag = 0, n = 0;

    while (str[i] != '\0') {
        j = 0, k = i, rep_ind = 0;

        // Check if the pattern matches at the current position
        while (str[k] == pat[j] && pat[j] != '\0') {
            k++;
            j++;
        }

        if (pat[j] == '\0') { // Pattern matched
            flag = 1;
```

```
            mflag = 1;
            // Copy the replacement string to the new string
            while (rep_pat[rep_ind] != '\0') {
                new_str[n++] = rep_pat[rep_ind++];
            }
        } else {
            flag = 0;
        }

        if (flag == 1) { // Skip the matched pattern
            i = k;
        } else { // Copy the current character from original string
            new_str[n++] = str[i++];
        }
    }

    if (mflag != 1) {
        printf("Pattern not found in the string!\n");
    } else {
        new_str[n] = '\0'; // Null-terminate the new string
        printf("New string: %s\n", new_str);
    }
}
```

Program 3

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5

int top = -1;
int stack[MAX];

void push(int element) {
    if (top == MAX - 1) {
        printf("Cannot push element to stack - Stack Overflow\n\n");
        return;
    }
    stack[++top] = element;
    printf("%d pushed to stack successfully!\n\n", element);
}

int pop() {
    if (top == -1) {
        printf("Cannot pop element from stack - Stack Underflow\n\n");
        return -1; // Return -1 to indicate underflow
    }
    return stack[top--];
}
```

```
void display() {
    if (top == -1) {
        printf("Stack is empty - nothing to display\n\n");
        return;
    }
    printf("Stack elements are: ");
    for (int i = top; i >= 0; i--) {
        printf("%d\t", stack[i]);
    }
    printf("\n\n");
}

void check_overflow_underflow() {
    if (top == -1) {
        printf("Stack is empty - Underflow condition\n\n");
    } else if (top == MAX - 1) {
        printf("Stack is full - Overflow condition\n\n");
    } else {
        printf("Stack is neither full nor empty. Current size: %d\n\n", top + 1);
    }
}

void palindrome() {
    if (top == -1) {
        printf("Stack is empty\n\n");
        return;
    }
    int isPalindrome = 1;
    for (int i = 0; i <= top / 2; i++) {
        if (stack[i] != stack[top - i]) {
            isPalindrome = 0;
            break;
        }
    }
    if (isPalindrome) {
        printf("Stack is a palindrome\n\n");
    } else {
        printf("Stack is not a palindrome\n\n");
    }
}

void main() {
    int choice, element, popped;
    while (1) {
        printf("Available stack operations:\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Check for overflow/underflow\n5.
Check for palindrome\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to push: ");
                scanf("%d", &element);
```

```
                push(element);
                break;
            case 2:
                popped = pop();
                if (popped != -1) {
                    printf("%d popped from stack successfully\n\n", popped);
                }
                break;
            case 3:
                display();
                break;
            case 4:
                check_overflow_underflow();
                break;
            case 5:
                palindrome();
                break;
            case 6:
                printf("Exiting....\n");
                exit(0);
            default:
                printf("Invalid operation. Please try again.\n\n");
        }
    }
}
```

Program 4

```
#include<stdio.h>

// Enumeration data type for precedence of operators
typedef enum {
    lparen,
    rparen,
    plus,
    minus,
    prod,
    div,
    mod,
    pow,
    eos,
    operand
} precedence;

// In-stack and In-coming Priorities
int isp[] = {0, 19, 12, 12, 13, 13, 13, 14, 0};
int icp[] = {20, 19, 12, 12, 13, 13, 13, 15, 0};


char expression[50], symbol;
```

```
precedence stack[50];
int top = 0, n = 0;

precedence readtoken() {
    symbol = expression[n++];
    switch (symbol) {
        case '(': return lparen;
        case ')': return rparen;
        case '+': return plus;
        case '-': return minus;
        case '*': return prod;
        case '/': return div;
        case '%': return mod;
        case '^': return pow;
        case '\0': return eos;
        default: return operand;
    }
}

// Stack operations

void push(precedence item) {
    stack[++top] = item;
}

precedence pop() {
    return stack[top--];
}

void displaytoken(precedence token) {
    switch (token) {
        case plus: printf("+"); break;
        case minus: printf("-"); break;
        case prod: printf("*"); break;
        case div: printf("/"); break;
        case mod: printf("%%"); break;
        case pow: printf("^"); break;
    }
}

void infixtopost() {
    precedence token, token1;
    stack[0] = eos;
    token = readtoken();
    while (token != eos) {
        if (token == operand) {
            printf("%c", symbol);
        } else if (token == rparen) {
            while (stack[top] != lparen) {
                token1 = pop();
                displaytoken(token1);
            }
            pop();
        } else {
```

```
            while (isp[stack[top]] >= icp[token]) {
                token1 = pop();
                displaytoken(token1);
            }
            push(token);
        }
        token = readtoken();
    }
    while ((token = pop()) != eos) {
        displaytoken(token);
    }
    printf("\n");
}

void main() {
    printf("Program to convert infix expression to postfix expression\n\n");
    printf("Enter infix expression: ");
    gets(expression);
    printf("Postfix expression: ");
    infixtopost();
}
```

Program 5a

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <ctype.h>
# include <math.h>
# define STACK_SIZE 50

int stack[STACK_SIZE];
int top = -1;

void push(int element) {
    stack[++top] = element;
    return;
}

int pop() {
    return stack[top--];
}

int eval(char s[]) {
    int temp, op1, op2, result;
    char symbol[1];
    for (int i = 0; i < strlen(s); i++) {
        if (isalnum(s[i])) {
            if (isdigit(s[i])) {
                symbol[0] = s[i];
```

```
                temp = atoi(symbol);
                push(temp);
            } else {
                printf("Enter a value of %c: ", s[i]);
                scanf("%d", &temp);
                push(temp);
            }
        } else {
            op2 = pop();
            op1 = pop();
            switch (s[i]) {
                case '^': result = pow(op1, op2); break;
                case '*': result = op1 * op2; break;
                case '/': if (op2 == 0) { printf("Cannot divide by zero!\n");
break; }
                      result = op1 / op2; break;
                case '%': if (op2 == 0) { printf("Cannot divide by zero!\n");
break; }
                      result = op1 % op2; break;
                case '+': result = op1 + op2; break;
                case '-': result = op1 - op2; break;
                default: printf("Invalid expression!\n"); exit(0);
            }
            push(result);
        }
    }
    result = pop();
    if (top != -1) {
        printf("Invalid expression!\n"); exit(0);
    }
    return result;
}

void main() {
    char s[50];
    printf("Evaluation of Postfix Expression\n");
    printf("Enter the postfix expression: ");
    gets(s);
    printf("\nThe result of the evaluation is: %d\n", eval(s));
    return;
}
```

Program 5b

```
# include <stdio.h>

int tower(int n, char s, char t, char d) {
    if (n == 0) return 0;
    tower(n-1, s, d, t);
    printf("Move disk %d from %c to %c\n", n, s, d);
```

```
    tower(n-1, t, s, d);
    return 0;
}

void main() {
    char A, B, C;
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Sequence of disk:\n");
    tower(n, 'A', 'B', 'C');
    printf("\n");
}
```

Program 6

```
#include<stdio.h>
#include<stdlib.h>
#include<stdio_ext.h>
#define MAX 5

int front = 0, rear = -1, count = 0;
char cq[MAX];

void insert(), delete(), display();

void main() {
    int choice;
    while (1) {
        printf("\nIMPLEMENTATION OF CIRCULAR QUEUE\n");
        printf("Menu -\n");
        printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        __fpurge(stdin);
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("Invalid input!\n");
        }
    }
}

void insert() {
    char item;
    if (count == MAX)
        printf("Circular queue is full\n");
    else {
```

```
        printf("Enter the element to be inserted: ");
        __fpurge(stdin);
        scanf("%c", &item);
        rear = (rear + 1) % MAX;
        cq[rear] = item;
        count++;
    }
}

void delete() {
    char item;
    if (count == 0)
        printf("Circular queue is empty\n");
    else {
        item = cq[front];
        front = (front + 1) % MAX;
        count--;
        printf("The deleted element is: %c\n", item);
    }
}

void display() {
    if (count == 0) {
        printf("Circular queue is empty\n");
    } else {
        printf("Contents of the circular queue are:\n");
        int j = front;
        for (int i = 1; i <= count; i++) {
            printf("%c\t", cq[j]);
            j = (j + 1) % MAX;
        }
        printf("\n");
    }
}
```

Program 7

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

struct node {
    char usn[12], name[20], branch[10], phno[15];
    int sem;
    struct node *link;
};
typedef struct node* NODE;

NODE start = NULL;
NODE start1 = NULL;
```

```
NODE create_node() {
    NODE ptr;
    ptr = (NODE)malloc(sizeof(struct node));
    if (ptr == NULL) {
        printf("Insufficient memory!\n\n");
        exit(0);
    }
    printf("\nEnter student data (USN, NAME, BRANCH, PHNO, SEM):\n");
    scanf("%s%s%s%s%d", ptr -> usn, ptr -> name, ptr -> branch, ptr -> phno, &ptr
-> sem);
    ptr -> link = NULL;
    return ptr;
}

NODE insert_front(NODE start) {
    NODE ptr = create_node();
    if (start == NULL)
        start = ptr;
    else {
        ptr -> link = start;
        start = ptr;
    }
    return start;
}

NODE insert_end(NODE start) {
    NODE ptr, temp;
    ptr = create_node();
    if (start == NULL) {
        start = ptr;
    } else {
        temp = start;
        while (temp -> link != NULL) {
            temp = temp -> link;
        }
        temp -> link = ptr;
    }
    return start;
}

NODE delete_front(NODE start) {
    NODE temp;
    if (start == NULL)
        printf("\nList Empty!\n\n");
    else {
        temp = start;
        start = start -> link;
        printf("\nDeleted node is: ");
        printf("| %s | %s | %s | %s | %d |\n", temp -> usn, temp -> name, temp ->
branch, temp -> phno, temp -> sem);
        free(temp);
    }
    return start;
}
```

```
NODE delete_end(NODE start) {
    NODE p, temp;
    if (start == NULL) {
        printf("\nList Empty\n\n"); return;
    }
    temp = start;
    if (temp -> link == NULL)
        start = NULL;
    else {
        temp = start;
        while (temp -> link != NULL) {
            p = temp;
            temp = temp -> link;
        }
        p -> link = NULL;
    }
    printf("\nDeleted node is: ");
    printf("| %s | %s | %s | %s | %d |\n", temp -> usn, temp -> name, temp ->
branch, temp -> phno, temp -> sem);
    free(temp);
    return start;
}

void display(NODE start) {
    NODE temp;
    if (start == NULL)
        printf("\nList Empty\n\n");
    else {
        int count = 0;
        temp = start;
        printf("\nSingly Linked List:\n");
        while (temp != NULL) {
            printf("| %s | %s | %s | %s | %d |\n", temp -> usn, temp -> name, temp
-> branch, temp -> phno, temp -> sem);
            temp = temp -> link;
            count++;
        }
        printf("\nNumber of nodes in the SLL: %d\n", count);
    }
}

NODE list_stack(NODE start1) {
    int choice;
    printf("\nOperations on List as Stack:\n");
    while (1) {
        printf("1.Push\n2.Pop\n3.Display\n4.Back to Main Menu\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: start1 = insert_front(start1); break;
            case 2: start1 = delete_front(start1); break;
            case 3: display(start1); break;
            case 4: return start1;
```

```
            default: printf("Invalid option!\n");
        }
    }
}

void main() {
    int choice, n;
    while (1) {
        printf("\nSLL Operations -\n");
        printf("1. List creation using insert front\n");
        printf("2. Display and count\n");
        printf("3. Insert at end\n");
        printf("4. Delete at end\n");
        printf("5. List as Stack\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: printf("Enter number of student data to enter: ");
                scanf("%d", &n);
                for (int i = 1; i <= n; i++) {
                    printf("Enter data for student %d\n", i);
                    start = insert_front(start);
                }
                break;
            case 2: display(start); break;
            case 3: start = insert_end(start); break;
            case 4: start = delete_end(start); break;
            case 5: start1 = list_stack(start1); break;
            case 6: exit(0);
            default: printf("Invalid option!\n");
        }
    }
}
```

Program 8

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

struct node {
    struct node *llink;
    int ssn;
    char name[20], dept[20], desgn[20], phno[12];
    float salary;
    struct node *rlink;
};

typedef struct node* NODE;
```

```
NODE start = NULL;
NODE create_node() {
    NODE ptr;
    ptr = (NODE)malloc(sizeof(struct node));
    if (ptr == NULL) {
        printf("Insufficient Memory!\n"); exit(0);
    }
    printf("\nEnter the employee data (SSN, Name, Department, Designation,
PhoneNo, Salary):\n");
    scanf("%d %s %s %s %s %f", &ptr->ssn, ptr->name, ptr->dept, ptr->desgn, ptr-
>phno, &ptr->salary);
    ptr -> llink = NULL;
    ptr -> rlink = NULL;
    return ptr;
}

NODE insert_front(NODE start) {
    NODE ptr;
    ptr = create_node();
    if (start == NULL) start = ptr;
    else {
        ptr -> rlink = start;
        start -> llink = ptr;
        start = ptr;
    }
    return start;
}

NODE insert_end(NODE start) {
    NODE ptr, temp;
    ptr = create_node();
    if (start == NULL) start = ptr;
    else {
        temp = start;
        while (temp -> rlink != NULL) temp = temp -> rlink;
        temp -> rlink = ptr;
        ptr -> llink = temp;
    }
    return start;
}

NODE delete_front(NODE start) {
    NODE temp;
    if (start == NULL) printf("\nDouble linked list is empty!\n");
    else {
        temp = start;
        start = start -> rlink;
        if (start != NULL) start -> llink = NULL;
        printf("\nDeleted node is:\n");
        printf("%d, %s, %s, %s, %s, %.2f\n", temp -> ssn, temp -> name, temp ->
dept, temp -> desgn, temp -> phno, temp -> salary);
        free(temp);
    }
    return start;
```

```
}

NODE delete_end(NODE start) {
    NODE temp, p;
    if (start == NULL) printf("\nDouble linked list is empty!\n");
    else {
        temp = start;
        if (temp -> rlink == NULL) start = NULL;
        else {
            while (temp -> rlink != NULL) temp = temp -> rlink;
            p = temp -> llink;
            p -> rlink = NULL;
        }
        printf("\nDeleted node is:\n");
        printf("%d, %s, %s, %s, %s, %.2f\n", temp -> ssn, temp -> name, temp ->
dept, temp -> desgn, temp -> phno, temp -> salary);
                free(temp);
    }
    return start;
}

void display(NODE start) {
    NODE temp;
    int count = 0;
    if (start == NULL) { printf("\nDoubly linked list is empty!\n"); return; }
    temp = start;
    printf("\nThe nodes of the doubly linked list are:\n");
    while (temp != NULL) {
        printf("%d, %s, %s, %s, %s, %.2f <-> ", temp -> ssn, temp -> name, temp ->
dept, temp -> desgn, temp -> phno, temp -> salary);
        temp = temp -> rlink;
        count++;
    }
    printf("NULL\n");
}

void dll_dequeue(NODE start) {
    printf("\nDequeue operations:\n");
    printf("\nInsertion at front:\n");
    start = insert_front(start);
    display(start);
    printf("\nInsertion at end:\n");
    start = insert_end(start);
    display(start);
    printf("\nDeletion at front:\n");
    start = delete_front(start);
    display(start);
    printf("\nDeletion at end:\n");
    start = delete_end(start);
    display(start);
}

void main() {
    int choice, n;
```

```
    NODE start = NULL;
    printf("\nDouble linked list operations:\n");
    printf("\nMenu:\n1. Create DLL of N employee using Insert End\n2. Dispaly and
Count\n3. Insert End\n4. Delete end\n5. Insert Front\n6. Delete Front\n7. DLL as
Dequeue\n8. Exit\n");
    while (1) {
//      printf("\nMenu:\n1. Create DLL of N employee using Insert End\n2. Dispaly
and Count\n3. Insert End\n4. Delete end\n5. Insert Front\n6. Delete Front\n7. DLL
as Dequeue\n8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter number of employees: ");
                scanf("%d", &n);
                for (int i = 1; i <= n; i++) {printf("Employee %d:\n", i); start =
insert_end(start);}
                break;
            case 2:
                display(start); break;
            case 3:
                start = insert_end(start); break;
            case 4:
                start = delete_end(start); break;
            case 5:
                start = insert_front(start); break;
            case 6:
                start = delete_front(start); break;
            case 7:
                dll_dequeue(start); break;
            case 8:
                printf("\nExiting...\n"); exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Program 9

```
#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 create_node() {
    NODE ptr = (NODE)malloc(sizeof(struct node));
    if (ptr == NULL) {
        printf("Insufficient memory!\n");
        exit(0);
    }
    return ptr;
}

NODE insert_end(int cf, int x, int y, int z, NODE head) {
    NODE ptr = create_node();
    ptr -> cf = cf;
    ptr -> px = x;
    ptr -> py = y;
    ptr -> pz = z;
    ptr -> flag = 0;
    if (head -> link == head) {
        head -> link = ptr;
        ptr -> link = head;
    } else {
        NODE temp = head -> link;
        while (temp -> link != head)
            temp = temp -> link;
        temp -> link = ptr;
        ptr -> link = head;
    }
    return head;
}

void display(NODE head) {
    if (head -> link == head)
        printf("\nEmpty polynomial.\n");
    else {
        NODE temp = head -> link;
        while (temp != head) {
            if (temp -> cf < 0)
                printf(" %dx^%dy^%dz^%d", temp->cf, temp->px, temp->py, temp->pz);
            else
                printf(" +%dx^%dy^%dz^%d", temp->cf, temp->px, temp->py, temp-
>pz);
            temp = temp -> link;
        }
    }
}

NODE add_polynomial(NODE h1, NODE h2, NODE h3) {
    NODE p1, p2;
    int cof;
    p1 = h1 -> link;
    while (p1 != h1) {
        p2 = h2 -> link;
        while (p2 != h2) {
            if (p1->px == p2->px && p1->py == p2->py && p1->pz == p2->pz)
```

```
                break;
            p2 = p2 -> link;
        }
        if (p2 != h2) {
            cof = p1->cf + p2->cf;
            p2 -> flag = 1;
            if (cof != 0)
                h3 = insert_end(cof, p1->px, p1->py, p1->pz, h3);
        } else
            h3 = insert_end(p1->cf, p1->px, p1->py, p1->pz, h3);
        p1 = p1 -> link;
    }
    p2 = h2 -> link;
    while (p2 != h2) {
        if (p2->flag == 0)
            h3 = insert_end(p2->cf, p2->px, p2->py, p2->pz, h3);
        p2 = p2 -> link;
    }
    return h3;
}

NODE read_polynomial(NODE head) {
    int i, cf, x, y, z;
    printf("Enter the coefficient and exponents (to stop, enter '-999'): ");
    for (i = 1;  ; i++) {
        printf("\nEnter the %d term:\n", i);
        printf("Coeff: "); scanf("%d", &cf);
        if (cf == -999) break;
        printf("pow(x): "); scanf("%d", &x);
        printf("pow(y): "); scanf("%d", &y);
        printf("pow(z): "); scanf("%d", &z);
        head = insert_end(cf, x, y, z, head);
    }
    return head;
}

void polysum() {
    NODE h1, h2, h3;
    h1 = create_node();
    h2 = create_node();
    h3 = create_node();
    h1 -> link = h1;
    h2 -> link = h2;
    h3 -> link = h3;
    printf("\n\nEnter the first polynomial:\n"); h1 = read_polynomial(h1);
        printf("\n\nEnter the second polynomial:\n"); h2 = read_polynomial(h2);
    h3 = add_polynomial(h1, h2, h3);
    printf("\nFirst polynomial: "); display(h1);
    printf("\nSecond polynomial: "); display(h2);
    printf("\nSum of the two polynomials: ");
    display(h3);
    printf("\n");
}
```

```
void eval() {
    NODE h, temp;
    int x, y, z, sum = 0;
    h = create_node();
    h -> link = h;
    printf("\nEnter the polynomial:\n");
    h = read_polynomial(h);
    printf("\nPolynomial: ");
    display(h);
    printf("\nEnter the values of variables x, y, z: ");
    scanf("%d%d%d", &x, &y, &z);
    temp = h -> link;
    while (temp != h) {
        sum += temp->cf * pow(x, temp->px) * pow(y, temp->py) * pow(z, temp->pz);
        temp = temp -> link;
    }
    printf("The total sum is: %d\n", sum);
}

void main() {
    int choice;
    while (1) {
        printf("\n1. Represent and Evaluate\n2. Add Two Polynomials\n3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1: eval(); break;
            case 2: polysum(); break;
            case 3: printf("Exiting...\n"); exit(0);
            default: printf("Invalid choice!\n");
        }
    }
}
```

Program 10

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

struct node {
    struct node *lchild;
    int data;
    struct node *rchild;
};
typedef struct node* NODE;

NODE root = NULL;

NODE create_node() {
    NODE ptr;
```

```
    ptr = (NODE)malloc(sizeof(struct node));
    if (ptr == NULL) {
        printf("Memory allocation failed!\n");
        exit(0);
    }
    printf("Enter data: ");
    scanf("%d", &ptr->data);
    ptr->rchild = ptr->lchild = NULL;
    return ptr;
}

NODE create_bst(NODE root) {
    NODE ptr, temp, p;
    ptr = create_node();
    if (root == NULL) {
        root = ptr;
        return root;
    }
    p = NULL;
    temp = root;
    while (temp != NULL) {
        p = temp;
        if (ptr->data == temp->data) {
            printf("Duplicate items are not allowed.\n");
            free(ptr);
            return root;
        }
        if (ptr->data < temp->data)
            temp = temp->lchild;
        else
            temp = temp->rchild;
    }
    if (ptr->data < p->data)
        p->lchild = ptr;
    else
        p->rchild = ptr;
    return root;
}

void inorder(NODE rt) {
    if (rt != NULL) {
        inorder(rt->lchild);
        printf("%d ", rt->data);
        inorder(rt->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 traverse(NODE root) {
    if (root == NULL)
        printf("BST is empty!\n");
    else {
        printf("\nPreorder traversal: "); preorder(root);
        printf("\nPostorder traversal: "); postorder(root);
        printf("\nInorder traversal: "); inorder(root);
    }
    printf("\n");
}

void search_bst(NODE root) {
    int key, flag = 0;
    printf("Enter the element to search: ");
    scanf("%d", &key);
    if (root == NULL) {
        printf("BST is empty!\n");
        return;
    } else {
        NODE temp = root;
        while (temp != NULL) {
            if (key == temp->data) {
                printf("Element is present in the BST.\n");
                flag = 1;
                break;
            } else if (key < temp->data)
                temp = temp->lchild;
            else
                temp = temp->rchild;
        }
    }
    if (flag == 0)
        printf("Element not found in the BST!\n");
}

void main() {
    int choice;
    printf("\nOperations on Binary Search Tree:\n");
    printf("1. Create BST\n2. Traverse BST\n3. Search in BST\n4. Exit\n");
    while (1) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: root = create_bst(root); break;
            case 2: traverse(root); break;
```

```
            case 3: search_bst(root); break;
            case 4: printf("\nExiting...\n"); return;
            default: printf("Invalid choice!\n");
        }
    }
}
```

Program 11

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

int adjacencyMatrix[50][50], numVertices, visited[50];
int queue[20], front = -1, rear = -1;
int stack[20], top = -1;

void bfs(int startVertex) {
    int i, currentVertex;
    visited[startVertex] = 1;
    queue[++rear] = startVertex;
    while (front != rear) {
        currentVertex = queue[++front];
        for (i = 1; i <= numVertices; i++) {
            if ((adjacencyMatrix[currentVertex][i] == 1) && (visited[i] == 0)) {
                queue[++rear] = i;
                visited[i] = 1;
                printf("%d ", i);
            }
        }
    }
}

void dfs(int startVertex) {
    int i;
    visited[startVertex] = 1;
    stack[++top] = startVertex;
    for (i = 1; i <= numVertices; i++) {
        if (adjacencyMatrix[startVertex][i] == 1 && visited[i] == 0) {
            printf("%d ", i);
            dfs(i);
        }
    }
}

int main() {
    int choice, startVertex, i, j;
    printf("Enter the number of vertices in graph:  ");
    scanf("%d", &numVertices);
    printf("Enter the adjacency matrix:\n");
    for (i = 1; i <= numVertices; i++) {
```

```
        for (j = 1; j <= numVertices; j++) {
            scanf("%d", &adjacencyMatrix[i][j]);
        }
    }

    printf("Enter the starting vertex: ");
    scanf("%d", &startVertex);

    printf("\n1. BFS traversal");
    printf("\n2. DFS traversal");
    printf("\n3: Exit");
    printf("\nEnter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("\nNodes reachable from starting vertex %d are: ",
startVertex);
            for (i = 1; i <= numVertices; i++) {
                visited[i] = 0;
            }
            bfs(startVertex);
            break;

        case 2:
            printf("\nNodes reachable from starting vertex %d are:\n",
startVertex);
            for (i = 1; i <= numVertices; i++) {
                visited[i] = 0;
            }
            dfs(startVertex);
            break;

        case 3:
            exit(0);

        default:
            printf("\nPlease enter a valid choice:");
    }

    return 0;
}
```

Program 12

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

struct record {
```

```
    int empno, flag;
    char name[10];
} emp[100];

int hash(int m) {
    return m % 100;
}

void main() {
    int m, k, eno, loc, n, j, i;
    char name[10];
    FILE *in;
    printf("Enter number of records to read from file: ");
    scanf("%d", &n);
    in = fopen("input.txt", "r");
    if (n <= 10) {
        for (k = 0; k < 100; k++)
            emp[k].flag = 0;
        for (i = 0; i < n; i++) {
            fscanf(in, "%d%s", &eno, name);
            loc = hash(eno);
            if (emp[loc].flag == 0) {
                printf("\nRecord: %d is mapped to address: %d\n", i, loc);
                emp[loc].empno = eno;
                emp[loc].flag = 1;
                strcpy(emp[loc].name, name);
            } else {
                printf("\nCollision occurred for record: %d. Resolved using linear
probing.\n", i);
                for (j = loc + 1; j < 100; j++) {
                    if (emp[j].flag == 0) {
                        printf("\nRecord: %d is at address: %d\n", i, j);
                        strcpy(emp[j].name, name);
                        emp[j].empno = eno;
                        emp[j].flag = 1;
                        break;
                    }
                }
                if (j >= 100) {
                    printf("Hash Table is full!\n");
                }
            }
        }
        fclose(in);
        printf("\nThe Hash Table contents are:\n");
        for (i = 0; i < 100; i++) {
            if (emp[i].flag == 1)
                printf("%d\t%d\t%s\n", i, emp[i].empno, emp[i].name);
            else
                printf("#\t#\t#\t\n");
        }
    } else {
        printf("\nFile is containing only 10 records.\n");
```

```
        }
    }
```