1st program

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

#define NUM_DAYS_IN_WEEK 7

typedef struct {
    char *acDayName;
    int iDate;
    char *acActivity;
} DAYTYPE;

void fnFreeCal(DAYTYPE *);
void fnDispCal(DAYTYPE *);
void fnReadCal(DAYTYPE *);
DAYTYPE *fnCreateCal();

int main() {
    DAYTYPE *weeklyCalendar = fnCreateCal();
    fnReadCal(weeklyCalendar);
    fnDispCal(weeklyCalendar);
    fnFreeCal(weeklyCalendar);
    return 0;
}

DAYTYPE *fnCreateCal() {
    DAYTYPE *calendar = (DAYTYPE *) malloc(NUM_DAYS_IN_WEEK * sizeof(DAYTYPE));
    for (int i = 0; i < NUM_DAYS_IN_WEEK; i++) {
        calendar[i].acDayName = NULL;
        calendar[i].iDate = 0;
        calendar[i].acActivity = NULL;
    }
    return calendar;
}

void fnReadCal(DAYTYPE *calendar) {
    char cChoice;
    for (int i = 0; i < NUM_DAYS_IN_WEEK; i++) {
        printf("Do you want to enter details for day %d [Y/N]: ", i + 1);
        scanf(" %c", &cChoice); // space before %c to skip whitespace
        getchar();
        if (tolower(cChoice) == 'n')
            continue;
        printf("Day Name: ");
        char nameBuffer[50], activityBuffer[100];
        scanf("%s", nameBuffer);
        calendar[i].acDayName = strdup(nameBuffer);
```

```
        printf("Date: ");
        scanf("%d", &calendar[i].iDate);
        printf("Activity: ");
        scanf(" %[^\n]", activityBuffer); // read until newline character
        calendar[i].acActivity = strdup(activityBuffer);
        printf("\n");
        getchar(); // remove trailing enter character in input buffer
    }
}

void fnDispCal(DAYTYPE *calendar) {
    printf("\nWeek's Activity Details:\n");
    for (int i = 0; i < NUM_DAYS_IN_WEEK; i++) {
        printf("Day %d:\n", i + 1);
        if (calendar[i].iDate == 0) {
            printf("No Activity\n\n");
            continue;
        }
        printf(" Day Name: %s\n", calendar[i].acDayName);
        printf(" Date: %d\n", calendar[i].iDate);
        printf(" Activity: %s\n\n", calendar[i].acActivity);
    }
}

void fnFreeCal(DAYTYPE *calendar) {
    for (int i = 0; i < NUM_DAYS_IN_WEEK; i++) {
        free(calendar[i].acDayName);
        free(calendar[i].acActivity);
    }
    free(calendar);
}
```

2nd program

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

int main() {
    char acMainStr[200], acSrchStr[30], acRepStr[30], acResStr[200];
    int i, j, k, iMtchCnt, iNumOfMatch = 0;

    printf("\nEnter the main string\n");
    scanf(" %[^\n]", acMainStr);

    printf("\nEnter the Pattern string\n");
    scanf(" %[^\n]", acSrchStr);

    printf("\nEnter the Replace string\n");
```

```
    scanf(" %[^\n]", acRepStr);

    for (i = 0; i < (strlen(acMainStr) - strlen(acSrchStr) + 1); i++) {
        iMtchCnt = 0;
        for (j = 0; j < strlen(acSrchStr); j++) {
            if (acMainStr[i + j] == acSrchStr[j]) {
                iMtchCnt++;
            } else {
                break;
            }
            if (iMtchCnt == strlen(acSrchStr)) {
                iNumOfMatch++;
                strcpy(acResStr, acMainStr);
                strncpy(acResStr + i, acRepStr, strlen(acRepStr));
                strcpy(acResStr + i + strlen(acRepStr), acMainStr + i + strlen(acSrchStr));
                strcpy(acMainStr, acResStr);
            }
        }
    }

    printf("\nInput Text\n");
    printf("%s\n", acMainStr);

    if (iNumOfMatch > 0) {
        printf("\n%d matches occurred\n\nText after replacing matched patterns is shown
below\n",
                iNumOfMatch);
        printf("\n%s\n", acResStr);
    } else {
        printf("\nPattern String not found in Text\n");
    }

    return 0;
}
```

3rd program

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

#define MAX 4

bool fnStkFull(int);
bool fnStkEmpty(int);
void fnPush(int [], int, int*);
int fnPop(int [], int*);
void fnDisplay(int[], int);
int fnPeek(int [], int);
```

```
bool fnChkPalindrome(int);

int main(void) {
    int stkArray[MAX];
    int top = -1;
    int iChoice, iElem;

    for (;;) {
        printf("\nSTACK OPERATIONS\n");
        printf("====================\n");

printf("1.Push\n2.Pop\n3.Display\n4.Peek\n5.CheckPalindrome\n6.DemonstrateOverflow\n7.
DemonstrateUnderflow\n8.EXIT\n");
        printf("Enter your choice: ");
        scanf("%d", &iChoice);

        switch (iChoice) {
            case 1:
                if (!fnStkFull(top)) {
                    printf("Enter element to be pushed onto the stack: ");
                    scanf("%d", &iElem);
                    fnPush(stkArray, iElem, &top);
                } else {
                    printf("Stack Overflow\n");
                }
                break;

            case 2:
                if (!fnStkEmpty(top)) {
                    iElem = fnPop(stkArray, &top);
                    printf("Popped Element is %d\n", iElem);
                } else {
                    printf("Stack Underflow\n");
                }
                break;

            case 3:
                if (fnStkEmpty(top)) {
                    printf("Stack Empty\n");
                } else {
                    fnDisplay(stkArray, top);
                }
                break;

            case 4:
                if (!fnStkEmpty(top)) {
                    iElem = fnPeek(stkArray, top);
                    printf("Element at the top of the stack is %d\n", iElem);
```

```
            } else {
                printf("Empty Stack\n");
            }
            break;

        case 5:
            printf("Enter number to be checked for a palindrome: ");
            scanf("%d", &iElem);
            if (fnChkPalindrome(iElem)) {
                printf("%d is a palindrome\n", iElem);
            } else {
                printf("%d is not a palindrome\n", iElem);
            }
            break;

        case 6:
            if (!fnStkFull(top)) {
                printf("There are currently %d elements in Stack\nPush %d elements for Stack
to overflow\n", top + 1, MAX - (top + 1));
                while (!fnStkFull(top)) {
                    printf("Enter an element: ");
                    scanf("%d", &iElem);
                    fnPush(stkArray, iElem, &top);
                }
                printf("Stack Overflow cannot push elements onto the stack\n");
            }
            break;

        case 7:
            if (!fnStkEmpty(top)) {
                printf("There are currently %d elements in Stack\nPop out %d elements for
Stack to Underflow\n", top + 1, MAX - (top + 1));
                while (!fnStkEmpty(top)) {
                    iElem = fnPop(stkArray, &top);
                    printf("Popped Element is %d\n", iElem);
                }
                printf("Stack Underflow cannot pop elements from the stack\n");
            }
            break;

        case 8:
            return 0;

        default:
            printf("Wrong choice\n");
      }
   }
}
```

```
bool fnStkFull(int t) {
    return (t == MAX - 1);
}

bool fnStkEmpty(int t) {
    return (t == -1);
}

void fnPush(int stk[], int iElem, int *t) {
    stk[++(*t)] = iElem;
}

int fnPop(int stk[], int *t) {
    return stk[(*t)--];
}

void fnDisplay(int stk[], int t) {
    printf("Stack Contents are: \n");
    for (int i = t; i >= 0; --i) {
        printf("%d\n", stk[i]);
    }
    printf("Stack has %d elements\n", t + 1);
}

int fnPeek(int stk[], int t) {
    return stk[t];
}

bool fnChkPalindrome(int iVal) {
    int palStk[10];
    int t = -1, iDig, iRev = 0;

    int iCopy = iVal;
    while (iCopy != 0) {
        iDig = iCopy % 10;
        fnPush(palStk, iDig, &t);
        iCopy /= 10;
    }
    int p = 0;
    while (p <= t) {
        iDig = palStk[p];
        iRev = iRev * 10 + iDig;
        p++;
    }
    return (iRev == iVal);
}
```

4th program

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

#define STK_SIZE 10

void fnPush(char [], int *, char);
char fnPop(char [], int *);
char fnPrecd(char);

int main() {
   char acStack[50], acPost[50], cSymb;
   int top = -1, j = 0;

   printf("\nEnter a valid infix expression\n");
   scanf("%s", acStack);

   fnPush(acStack, &top, '#');

   for (int i = 0; acStack[i] != '\0'; ++i) {
      cSymb = acStack[i];
      if (isalnum(cSymb)) {
         acPost[j++] = cSymb;
      } else if (cSymb == '(') {
         fnPush(acStack, &top, cSymb);
      } else if (cSymb == ')') {
         while (acStack[top] != '(') {
            acPost[j++] = fnPop(acStack, &top);
         }
         fnPop(acStack, &top);
      } else {
         while (fnPrecd(acStack[top]) >= fnPrecd(cSymb)) {
            if ((cSymb == '^') && (acStack[top] == '^')) break;
            acPost[j++] = fnPop(acStack, &top);
         }
         fnPush(acStack, &top, cSymb);
      }
   }

   while (acStack[top] != '#') {
      acPost[j++] = fnPop(acStack, &top);
   }
   acPost[j] = '\0';

   printf("\nPostfix Expression is %s\n", acPost);
   return 0;
}
```

```
void fnPush(char Stack[], int *t , char elem) {
   Stack[++(*t)] = elem;
}

char fnPop(char Stack[], int *t) {
   return Stack[(*t)--];
}

char fnPrecd(char ch) {
   char cPrecdVal = -1;
   switch(ch) {
      case '(':
         cPrecdVal = 0;
         break;
      case '+':
      case '-':
         cPrecdVal = 1;
         break;
      case '%':
      case '*':
      case '/':
         cPrecdVal = 2;
         break;
      case '^':
         cPrecdVal = 3;
         break;
   }
   return cPrecdVal;
}
```

5th a program

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

#define STACK_SIZE 50

void push(int [], int *, int);
int pop(int [], int *);

int main() {
   int stack[STACK_SIZE], top = -1, res;
   char expr[50], symb;
```

```
    printf("\nEnter a valid postfix expression\n");
    scanf("%s", expr);

    for (int i = 0; i < strlen(expr); i++) {
        symb = expr[i];
        if (isdigit(symb)) {
            push(stack, &top, symb - '0');
        } else {
            int op2 = pop(stack, &top);
            int op1 = pop(stack, &top);
            switch(symb) {
                case '+' : res = op1 + op2; break;
                case '-' : res = op1 - op2; break;
                case '*' : res = op1 * op2; break;
                case '/' : res = op1 / op2; break;
                case '%' : res = op1 % op2; break;
                case '^' : res = (int)pow(op1, op2); break;
            }
            push(stack, &top, res);
        }
    }

    res = pop(stack, &top);
    printf("\nValue of %s expression is %d\n", expr, res);

    return 0;
}

void push(int Stack[], int *t, int elem) {
    Stack[++(*t)] = elem;
}

int pop(int Stack[], int *t) {
    return Stack[(*t)--];
}
```

5th b program

```
#include <stdio.h>

void towers(int, char, char, char);

int main() {
    int num;
    printf("Enter the number of disks: ");
    scanf("%d", &num);
    printf("The sequence of moves involved in the Tower of Hanoi are:\n");
    towers(num, 'A', 'C', 'B');
```

```
    printf("\n");
    return 0;
}

void towers(int num, char frompeg, char topeg, char auxpeg) {
    if (num == 1) {
        printf("\nMove disk 1 from peg %c to peg %c", frompeg, topeg);
        return;
    }
    towers(num - 1, frompeg, auxpeg, topeg);
    printf("\nMove disk %d from peg %c to peg %c", num, frompeg, topeg);
    towers(num - 1, auxpeg, topeg, frompeg);
}
```

6th program

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

#define SIZE 5

void insert(char [], int*, int*, char);
char del(char[], int*, int*);
void display(char [], int, int);
bool qfull(int, int);
bool qempty(int, int);

int main() {
    char q[SIZE];
    int f = -1, r = -1;
    int ch;
    char elem;

    for (;;) {
        printf("\nQueue Operations\n");
        printf("======================");
        printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        getchar();

        switch(ch) {
            case 1:
                if (!qfull(f, r)) {
                    printf("\nEnter an element : ");
                    scanf("%c", &elem);
                    insert(q, &f, &r, elem);
```

```
            } else {
                printf("\nQueue is Full\n");
            }
            break;
        case 2:
            if (!qempty(f, r)) {
                elem = del(q, &f, &r);
                printf("\nDeleted element is %c\n", elem);
            } else {
                printf("\nQueue is Empty\n");
            }
            break;
        case 3:
            if (!qempty(f, r)) {
                printf("\nContents of the Queue is \n");
                display(q, f, r);
            } else {
                printf("\nQueue is Empty\n");
            }
            break;
        case 4:
            exit(0);
        default:
            printf("\nInvalid choice\n");
            break;
      }
   }
   return 0;
}

bool qfull(int fr, int rr) {
   return (rr + 1) % SIZE == fr;
}

bool qempty(int fr, int rr) {
   return fr == -1;
}

void insert(char queue[], int *f, int *r, char val) {
   if (*r == -1) {
      *f = *r = 0;
   } else {
      *r = (*r + 1) % SIZE;
   }
   queue[*r] = val;
}

char del(char queue[], int *f, int *r) {
```

```
    char el = queue[*f];
    if (*f == *r) {
        *f = *r = -1;
    } else {
        *f = (*f + 1) % SIZE;
    }
    return el;
}

void display(char queue[], int fr, int rr) {
    int i;
    if (fr <= rr) {
        for (i = fr; i <= rr; i++) {
            printf("%c\t", queue[i]);
        }
    } else {
        for (i = fr; i < SIZE; i++) {
            printf("%c\t", queue[i]);
        }
        for (i = 0; i <= rr; i++) {
            printf("%c\t", queue[i]);
        }
    }
    printf("\n");
}
```

7th program

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

struct node {
    char usn[11], name[40], prog[4], ph[11];
    int sem;
    struct node *link;
};

typedef struct node* PTR;

PTR get(void);
void freeN(PTR);
PTR insrear(PTR);
PTR delfront(PTR);
PTR insfront(PTR);
PTR delrear(PTR);
void disp(PTR);
```

```
int main() {
   PTR first = NULL;
   int ch;

   for (;;) {
      printf("\nQUEUE OPERATIONS\n");
      printf("====================");
      printf("\n1.Insert Front\n2.Insert Rear\n3.Delete Front\n4.Delete
Rear\n5.Display\n6.Exit\n");
      printf("\nEnter your choice\n");
      scanf("%d", &ch);

      switch(ch) {
         case 1: first = insfront(first); break;
         case 2: first = insrear(first); break;
         case 3: first = delfront(first); break;
         case 4: first = delrear(first); break;
         case 5: disp(first); break;
         case 6: exit(0);
      }
   }
   return 0;
}

PTR get() {
   PTR newborn = (PTR)malloc(sizeof(struct node));
   if (newborn == NULL) {
      printf("\nMemory Overflow");
      exit(0);
   }
   printf("\nEnter USN, name, Program name, semester, and Phone no: ");
   scanf("%s %s %s %d %s", newborn->usn, newborn->name, newborn->prog,
&newborn->sem, newborn->ph);
   return newborn;
}

void freeN(PTR x) {
   free(x);
}

PTR insrear(PTR first) {
   PTR temp, cur;
   temp = get();
   temp->link = NULL;

   if (first == NULL)
      return temp;
```

```
    cur = first;
    while (cur->link != NULL)
        cur = cur->link;

    cur->link = temp;
    return first;
}

PTR delfront(PTR first) {
    PTR temp;
    if (first == NULL) {
        printf("\nSLL is empty cannot delete\n");
        return first;
    }
    temp = first;
    first = first->link;
    printf("\nNode deleted is %s\n", temp->name);
    freeN(temp);
    return first;
}

void disp(PTR first) {
    PTR curr;
    int count = 0;
    if (first == NULL) {
        printf("\nSLL is empty\n");
        return;
    }
    printf("\nThe contents of SLL are :\n");
    curr = first;
    printf("\nUSN\t\tName\tProgram\tSem\tPhone num\n");
    while (curr != NULL) {
        printf("%10s\t%s\t%s\t%d\t%s\n", curr->usn, curr->name, curr->prog, curr->sem,
curr->ph);
        curr = curr->link;
        count++;
    }
    printf("\nSLL has %d nodes\n", count);
}

PTR insfront(PTR first) {
    PTR temp = get();
    temp->link = first;
    return temp;
}

PTR delrear(PTR first) {
    PTR cur, prev;
```

```
    if (first == NULL) {
        printf("\nSLL is empty cannot delete\n");
        return first;
    }
    prev = NULL;
    cur = first;
    if (cur->link == NULL) {
        printf("\nNode deleted for %s\n", cur->name);
        freeN(cur);
        return NULL;
    }
    while (cur->link != NULL) {
        prev = cur;
        cur = cur->link;
    }
    prev->link = cur->link;
    printf("\nNode deleted for %s\n", cur->name);
    freeN(cur);
    return first;
}
```

8th program

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

struct node {
    int usn, sal;
    char name[30], dept[4], desig[30], ph[11];
    struct node *plink, *nlink;
};

typedef struct node* NODE;

NODE getn(void);
void freen(NODE);
NODE insrear(NODE);
NODE delfront(NODE);
NODE insfront(NODE);
NODE delrear(NODE);
void disp(NODE);

int main() {
    NODE first = NULL;
    int ch, num, i;

    printf("\nEnter the number of Employees N : ");
```

```
    scanf("%d", &num);

    for(i = 0; i < num; i++) {
        printf("\nEnter Data for Node %d :\n", i + 1);
        first = insrear(first);
    }

    for(;;) {
        printf("\nDLL OPERATIONS\n====================");
        printf("\n1.Insert Rear\n2.Delete Front\n3.Insert Front\n4.Delete
Rear\n5.Display\n6.Exit\n");
        printf("\nEnter your choice\n");
        scanf("%d",&ch);
        switch(ch) {
            case 1: first = insrear(first); break;
            case 2: first = delfront(first); break;
            case 3: first = insfront(first); break;
            case 4: first = delrear(first); break;
            case 5: disp(first); break;
            case 6: exit(0);
        }
    }
    return 0;
}

NODE getn() {
    NODE newborn = (NODE)malloc(sizeof(struct node));
    if (!newborn) {
        printf("\nMemory Overflow");
        exit(0);
    }
    printf("\nEnter SSN, name, Department, Designation, Salary, and Phone no: ");
    scanf("%d %s %s %s %d %s", &newborn->usn, newborn->name, newborn->dept,
newborn->desig, &newborn->sal, newborn->ph);
    return newborn;
}

void freen(NODE x) {
    free(x);
}

NODE insrear(NODE first) {
    NODE temp = getn(), cur = first;
    temp->plink = temp->nlink = NULL;
    if (!first) return temp;
    while (cur->nlink) cur = cur->nlink;
    cur->nlink = temp;
    temp->plink = cur;
```

```
    return first;
}

NODE insfront(NODE first) {
    NODE temp = getn();
    temp->plink = temp->nlink = NULL;
    temp->nlink = first;
    if (first) first->plink = temp;
    return temp;
}

NODE delrear(NODE first) {
    NODE cur = first, prev;
    if (!first) {
        printf("\nDLL is empty\n");
        return first;
    }
    if (!cur->nlink) {
        printf("\nNode deleted for %s\n", cur->name);
        freen(cur);
        return NULL;
    }
    while (cur->nlink) cur = cur->nlink;
    prev = cur->plink;
    prev->nlink = NULL;
    printf("\nNode deleted for %s\n", cur->name);
    freen(cur);
    return first;
}

NODE delfront(NODE first) {
    NODE temp;
    if (!first) {
        printf("\nDLL is empty\n");
        return first;
    }
    temp = first;
    first = first->nlink;
    if (first) first->plink = NULL;
    printf("\nNode deleted for %s\n", temp->name);
    freen(temp);
    return first;
}

void disp(NODE first) {
    NODE curr;
    int count = 0;
    if (!first) {
```

```
        printf("\nDLL is empty\n");
        return;
    }
    printf("\nThe contents of DLL are :\n");
    curr = first;
    printf("\nSSN\tName\tDept\tDesignation\tSalary\t\tPhone No");
    while (curr) {
        printf("\n%-5d\t%s\t%s\t%s\t\t%-7d\t\t%-11s", curr->usn, curr->name, curr->dept,
curr->desig, curr->sal, curr->ph);
        curr = curr->nlink;
        count++;
    }
    printf("\n\nDLL has %d nodes\n", count);
}
```

9th program

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

struct polyt {
    int cf, px, py, pz;
    struct polyt* next;
};

typedef struct polyt* PTR;

PTR insert(PTR poly, int cf, int px, int py, int pz) {
    PTR nn = (PTR)malloc(sizeof(struct polyt));
    nn->cf = cf; nn->px = px; nn->py = py; nn->pz = pz;
    nn->next = poly->next;
    poly->next = nn;
    return poly;
}

void disp(PTR poly) {
    if (poly->next == poly) {
        printf("Polynomial is empty.\n");
        return;
    }
    PTR cur = poly->next;
    do {
        printf("%dx^%dy^%dz^%d ", cur->cf, cur->px, cur->py, cur->pz);
        cur = cur->next;
        if (cur != poly) printf("+ ");
    } while (cur != poly);
```

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

int evaluate(PTR poly, int x, int y, int z) {
    int result = 0;
    PTR cur = poly->next;
    while (cur != poly) {
        int termValue = cur->cf;
        termValue *= pow(x, cur->px) * pow(y, cur->py) * pow(z, cur->pz);
        result += termValue;
        cur = cur->next;
    }
    return result;
}

bool fmatch(PTR p1, PTR p2) {
    return p1->px == p2->px && p1->py == p2->py && p1->pz == p2->pz;
}

PTR add(PTR poly1, PTR poly2, PTR polySum) {
    PTR cur1 = poly1->next, cur2;
    while (cur1 != poly1) {
        polySum = insert(polySum, cur1->cf, cur1->px, cur1->py, cur1->pz);
        cur1 = cur1->next;
    }
    cur2 = poly2->next;
    while (cur2 != poly2) {
        cur1 = polySum->next;
        bool matchfound = false;
        while (cur1 != polySum) {
            if (fmatch(cur1, cur2)) {
                cur1->cf += cur2->cf;
                matchfound = true;
                break;
            }
            cur1 = cur1->next;
        }
        if (!matchfound) polySum = insert(polySum, cur2->cf, cur2->px, cur2->py, cur2->pz);
        cur2 = cur2->next;
    }
    return polySum;
}

int main() {
    PTR poly1 = (PTR)malloc(sizeof(struct polyt)); poly1->next = poly1;
    PTR poly2 = (PTR)malloc(sizeof(struct polyt)); poly2->next = poly2;
    PTR polySum = (PTR)malloc(sizeof(struct polyt)); polySum->next = polySum;
    poly1 = insert(poly1, 6, 2, 2, 1);
```

```
    poly1 = insert(poly1, 4, 0, 1, 5);
    poly1 = insert(poly1, 3, 3, 1, 1);
    poly1 = insert(poly1, 2, 1, 5, 1);
    poly1 = insert(poly1, 2, 1, 1, 3);
    printf("POLY1(x, y, z) = "); disp(poly1);
    poly2 = insert(poly2, 1, 1, 1, 1);
    poly2 = insert(poly2, 4, 3, 1, 1);
    printf("POLY2(x, y, z) = "); disp(poly2);
    polySum = add(poly1, poly2, polySum);
    printf("\nPOLYSUM(x, y, z) = "); disp(polySum);
    int x = 1, y = 2, z = 3;
    int res = evaluate(polySum, x, y, z);
    printf("\nResult of POLYSUM(%d, %d, %d): %d\n", x, y, z, res);
    return 0;
}
```

10th program

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

struct node {
    int info;
    struct node *lbranch;
    struct node *rbranch;
};

typedef struct node* NODEPTR;

NODEPTR getNode() {
    return (NODEPTR)malloc(sizeof(struct node));
}

void freeNode(NODEPTR x) {
    free(x);
}

NODEPTR insertNode(int item, NODEPTR root) {
    NODEPTR temp, prev, cur;
    temp = getNode();
    temp->info = item;
    temp->lbranch = temp->rbranch = NULL;
    if (root == NULL) return temp;
    prev = NULL;
    cur = root;
    while (cur != NULL) {
        prev = cur;
        if (item == cur->info) {
```

```
            printf("\nDuplicate items not allowed\n");
            freeNode(temp);
            return root;
        }
        cur = (item < cur->info) ? cur->lbranch : cur->rbranch;
    }
    if (item < prev->info) prev->lbranch = temp;
    else prev->rbranch = temp;
    return root;
}

void preOrder(NODEPTR root) {
    if (root != NULL) {
        printf("%d\t", root->info);
        preOrder(root->lbranch);
        preOrder(root->rbranch);
    }
}

void inOrder(NODEPTR root) {
    if (root != NULL) {
        inOrder(root->lbranch);
        printf("%d\t", root->info);
        inOrder(root->rbranch);
    }
}

void postOrder(NODEPTR root) {
    if (root != NULL) {
        postOrder(root->lbranch);
        postOrder(root->rbranch);
        printf("%d\t", root->info);
    }
}

void searchBST(NODEPTR root, int elem) {
    if (root != NULL) {
        if (elem < root->info) searchBST(root->lbranch, elem);
        else if (elem > root->info) searchBST(root->rbranch, elem);
        else printf("\n%d is found in the BST\n", elem);
    } else {
        printf("\n%d is not found in the BST\n", elem);
    }
}

int main() {
    NODEPTR root = NULL;
    int choice, item, num, i;
```

```
    printf("Create a BST of N Integers \n");
    printf("\nEnter the number N : ");
    scanf("%d", &num);
    printf("\nEnter %d numbers\n", num);
    for (i = 0; i < num; i++) {
        scanf("%d", &item);
        root = insertNode(item, root);
    }
    for (;;) {
        printf("\n1.Inorder traversal\n2.Preorder traversal\n3.Postorder
traversal\n4.Search\n5.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                if (root == NULL) printf("\nTree is Empty\n");
                else {
                    printf("\nInorder Traversal is :\n");
                    inOrder(root);
                    printf("\n");
                }
                break;
            case 2:
                if (root == NULL) printf("\nTree is Empty\n");
                else {
                    printf("\nPreorder Traversal is :\n");
                    preOrder(root);
                    printf("\n");
                }
                break;
            case 3:
                if (root == NULL) printf("\nTree is Empty\n");
                else {
                    printf("\nPostorder Traversal is :\n");
                    postOrder(root);
                    printf("\n");
                }
                break;
            case 4:
                printf("\nEnter the element to be searched : ");
                scanf("%d", &item);
                searchBST(root, item);
                break;
            case 5:
                exit(0);
            default:
                printf("Wrong choice\n");
                break;
```

```
    }
  }
  return 0;
}
```