SIDDAGANGA INSTITUTE OF TECHNOLOGY

www.sit.ac.in

Data Structures Laboratory

LAB MANUAL

Prabodh C P
Asst Professor
Dept of CSE, SIT

Contents

Listings

Data Structures Laboratory

Instructions

- All the C programs need to be executed using GCC Compiler.
- Algorithms and Flowcharts are compulsory for all the programs.
- All experiments must be included in practical examinations.

References

Part A: Behrouz A. Forouzan , Richard F. Gilberg , Computer Science: **A Structured programming Approach Using C** - Cengage Learning; 3rd edition
For writing flowcharts refer to **Appendix C** of the above book.

Chapter 1

File Management

Question

Write a C program to create a sequential file with at least five records, each record having the structure shown in the table:

Write necessary functions to perform the following operations:
i) to display all the records in the file.
ii) to search for a specific record based on EMPLOYEE_ID/SALARY/DEPARTMENT/AGE.

In case if the required record is not found, suitable message should be displayed.

EMPLOYEE_ID	NAME	DEPARTMENT	SALARY	AGE
Non-Zero +ve Integer	25 Characters	25 Characters	+ve Integer	+ve Integer

1.1 C Code - Text I/O

===================================

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct{
    unsigned emp_id;
    char emp_name[25];
    char emp_dept[25];
    unsigned emp_salary, emp_age;
}employee_t;

/* FUNCTION PROTOTYPES */
void fnAddRecord(void);
void fnSearchEmpID(int);
void fnSearchEmpSal(int);
void fnSearchEmpDept(char[]);
void fnSearchEmpAge(int);
void fnDisplayAllRecords(void);

int main()
{
    int id, sal, age, iChoice;
    char dept[10];

    for(;;)
    {
        printf("\n1.Add Record\n2.Display Records\n3.Search Employee by ID\n");
        printf("4.Search Employee by Dept\n5.Search Employee by salary\n");
        printf("6.Search Employee by Age\n7.Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1: fnAddRecord();
                    break;

            case 2: printf("\n Employee Details \n");
                    fnDisplayAllRecords();
                    break;

            case 3: printf("\nEnter the emp_id that you want to search\n");
                    scanf("%d",&id);
                    fnSearchEmpID(id);
                    break;

            case 4: printf("\nEnter the dept that you want to search\n");
                    scanf("%s",dept);
                    fnSearchEmpDept(dept);
                    break;

            case 5: printf("\nEnter the salary that you want to search\n");
                    scanf("%d",&sal);
                    fnSearchEmpSal(sal);
                    break;

            case 6: printf("\nEnter the age that you want to search\n");
                    scanf("%d",&age);
                    fnSearchEmpAge(age);
                    break;

            case 7: exit(0);
        }
    }
    return 0;
}

void fnDisplayAllRecords()
{
    int iCount = 0;
    employee_t ep;
    FILE *fp;

    fp = fopen("emp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    printf("\nID\tName\tDept\tSalary\tAge\n");
    while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
    emp_salary, &ep.emp_age)!=EOF)
    {
        printf("%d\t%s\t%s\t%d\t%d\n",ep.emp_id, ep.emp_name, ep.emp_dept, ep.
    emp_salary, ep.emp_age);
        iCount++;
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
    fclose(fp);
}
```

```c

void fnAddRecord()
{
    FILE *fp;
    employee_t emp;

    printf("\nEnter Employee details\n");
    printf("\nID : ");
    scanf("%d", &emp.emp_id);     getchar();
    printf("\nName : ");
    scanf("%s", emp.emp_name);
    printf("\nDept : ");
    scanf("%s", emp.emp_dept);
    printf("\nSalary : ");
    scanf("%d", &emp.emp_salary);
    printf("\nAge : ");
    scanf("%d", &emp.emp_age);

    fp = fopen("emp.dat", "a");
    fprintf(fp,"%d\t%s\t%s\t%d\t%d\n",emp.emp_id, emp.emp_name, emp.emp_dept, emp.
   emp_salary, emp.emp_age);
    fclose(fp);
}

void fnSearchEmpID(int id)
{
    int iCount = 0;
    employee_t ep;
    FILE *fp;

    fp = fopen("emp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    printf("\nID\tName\tDept\tSalary\tAge\n");
    while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
   emp_salary, &ep.emp_age)!=EOF)
    {
        if(ep.emp_id == id)
        {
            printf("%d\t%s\t%s\t%d\t%d\n",ep.emp_id, ep.emp_name, ep.emp_dept, ep.
   emp_salary, ep.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
    fclose(fp);
}

void fnSearchEmpSal(int sal)
{
    int iCount = 0;
    employee_t ep;
    FILE *fp;

    fp = fopen("emp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    printf("\nID\tName\tDept\tSalary\tAge\n");
    while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.emp_salary, &ep.emp_age)!=EOF)
    {
        if(ep.emp_salary == sal)
        {
            printf("%d\t%s\t%s\t%d\t%d\n",ep.emp_id, ep.emp_name, ep.emp_dept, ep.emp_salary, ep.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
    fclose(fp);
}

void fnSearchEmpDept(char dept[])
{
    int iCount = 0;
    employee_t ep;
    FILE *fp;


    fp = fopen("emp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    printf("\nID\tName\tDept\tSalary\tAge\n");
    while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.emp_salary, &ep.emp_age)!=EOF)
    {
        if(!strcmp(ep.emp_dept, dept))
        {
            printf("%d\t%s\t%s\t%d\t%d\n",ep.emp_id, ep.emp_name, ep.emp_dept, ep.emp_salary, ep.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
}

void fnSearchEmpAge(int age)
{
    int iCount = 0;
    employee_t ep;
    FILE *fp;

    fp = fopen("emp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    printf("\nID\tName\tDept\tSalary\tAge\n");
    while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.emp_salary, &ep.emp_age)!=EOF)
    {
        if(ep.emp_age == age)
        {
            printf("%d\t%s\t%s\t%d\t%d\n",ep.emp_id, ep.emp_name, ep.emp_dept, ep.
    emp_salary, ep.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
}
```

Listing 1.1: 01EmployeeDB.c

1.2 C Code - Binary I/O

=================================

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct{
    unsigned emp_id;
    char emp_name[25];
    char emp_dept[25];
    unsigned emp_salary, emp_age;
}employee_t;

void fnAddRecord(void);
void fnSearchEmpID(int);
void fnSearchEmpSal(int);
void fnSearchEmpDept(char[]);
void fnSearchEmpAge(int);
void fnDisplayAllRecords(void);

int main()
{
    int id, sal, age, iChoice;
    char dept[10];
    printf("%lu bytes\n",sizeof(employee_t));
    for(;;)
    {
        printf("\n1.Add Record\n2.Display Records\n3.Search Employee by ID\n");
        printf("4.Search Employee by Dept\n5.Search Employee by salary\n");
        printf("6.Search Employee by Age\n7.Exit");
        printf("\nEnter your choice : ");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1: fnAddRecord();
                    break;

            case 2: printf("\n Employee Details \n");
                    fnDisplayAllRecords();
                    break;

            case 3: printf("\nEnter the emp_id that you want to search\n");
                    scanf("%d",&id);
                    fnSearchEmpID(id);
                    break;

            case 4: printf("\nEnter the dept that you want to search\n");
```

```
                    scanf("%s",dept);
                    fnSearchEmpDept(dept);
                    break;

            case 5: printf("\nEnter the salary that you want to search\n");
                    scanf("%d",&sal);
                    fnSearchEmpSal(sal);
                    break;

            case 6: printf("\nEnter the age that you want to search\n");
                    scanf("%d",&age);
                    fnSearchEmpAge(age);
                    break;

            case 7: exit(0);
        }
    }
    return 0;
}

void fnDisplayAllRecords()
{
    int iCount = 0;
    employee_t rEmp;
    FILE *fp;

    fp = fopen("bemp.dat", "rb");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }

    while(fread(&rEmp, sizeof(employee_t),1,fp))
    {
        printf("%6d\t%15s\t%8s\t%8d\t%4d\n",rEmp.emp_id, rEmp.emp_name, rEmp.
   emp_dept, rEmp.emp_salary, rEmp.emp_age);
        iCount++;
        if(feof(fp))
            break;
    }

    if(0 == iCount)
        printf("\nNo Records found\n");
    fclose(fp);
}

void fnAddRecord()
{
    FILE *fp;
    employee_t wEmp;

    printf("\nEnter Employee details\n");
    printf("\nID : ");
    scanf("%d",&wEmp.emp_id);        getchar();
    printf("\nName : ");
    gets(wEmp.emp_name);
    //fgets(wEmp.emp_name, 25, stdin);
    printf("\nDept : ");
    gets(wEmp.emp_dept);
    //fgets(wEmp.emp_dept, 25, stdin);
    printf("\nSalary : ");
    scanf("%d",&wEmp.emp_salary);
    printf("\nAge : ");
    scanf("%d",&wEmp.emp_age);

    fp = fopen("bemp.dat", "ab");

    fwrite(&wEmp, sizeof(employee_t),1,fp);
    //write(fp,&wEmp,sizeof(employee_t));

    fclose(fp);
}

void fnSearchEmpID(int id)
{
    int iCount = 0;
    employee_t sEmp;
    FILE *fp;

    fp = fopen("bemp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    while(fread(&sEmp, sizeof(employee_t),1,fp))
    {
        if(sEmp.emp_id == id)
        {
            printf("%d\t%s\t%s\t%d\t%d\n",sEmp.emp_id, sEmp.emp_name, sEmp.
   emp_dept, sEmp.emp_salary, sEmp.emp_age);
            iCount++;
        }
        if(feof(fp))
            break;
    }

    if(0 == iCount)
        printf("\nNo Records found\n");
    fclose(fp);
}

void fnSearchEmpSal(int sal)
{
    int iCount = 0;
    employee_t sEmp;
    FILE *fp;

    fp = fopen("bemp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    while(fread(&sEmp, sizeof(employee_t),1,fp))
    {
        if(sEmp.emp_salary == sal)
        {
            printf("%d\t%s\t%s\t%d\t%d\n",sEmp.emp_id, sEmp.emp_name, sEmp.
   emp_dept, sEmp.emp_salary, sEmp.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
    fclose(fp);
}

void fnSearchEmpDept(char dept[])
{
    int iCount = 0;
    employee_t sEmp;
    FILE *fp;


    fp = fopen("bemp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    while(fread(&sEmp, sizeof(employee_t),1,fp))
    {
        if(!strcmp(sEmp.emp_dept, dept))
        {
            printf("%d\t%s\t%s\t%d\t%d\n",sEmp.emp_id, sEmp.emp_name, sEmp.
   emp_dept, sEmp.emp_salary, sEmp.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
}

void fnSearchEmpAge(int age)
{
    int iCount = 0;
    employee_t sEmp;
    FILE *fp;

    fp = fopen("bemp.dat", "r");
    if(fp==NULL)
    {
        printf("\nFile does not exist\n");
        return;
    }
    while(fread(&sEmp, sizeof(employee_t),1,fp))
    {
        if(sEmp.emp_age == age)
        {
            printf("%d\t%s\t%s\t%d\t%d\n",sEmp.emp_id, sEmp.emp_name, sEmp.
   emp_dept, sEmp.emp_salary, sEmp.emp_age);
            iCount++;
        }
    }
    if(0 == iCount)
        printf("\nNo Records found\n");
}
```

Listing 1.2: 01EmployeeDBBinary.c

Output

```
==================================
/****************************************
putta:Programs$ gcc 01EmployeeDB.c
putta:Programs$ ./a.out

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 1

Enter Employee details

ID : 123
Name : Raju
Dept : CSE
Salary : 24000
Age : 26

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 2

 Employee Details

ID  Name    Dept    Salary  Age
123 Raju    CSE 24000   26

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 1

Enter Employee details

ID : 124
Name : Susy
Dept : ISE
Salary : 26000
Age : 25

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 1

Enter Employee details
```

```

ID : 125
Name : John
Dept : CSE
Salary : 27000
Age : 29

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 2

 Employee Details

ID  Name    Dept    Salary  Age
123 Raju    CSE     24000   26
124 Susy    ISE     26000   25
125 John    CSE     27000   29

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 3

Enter the emp_id that you want to search
127

ID  Name    Dept    Salary  Age

No Records found

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 3

Enter the emp_id that you want to search
125

ID  Name    Dept    Salary  Age
125 John    CSE     27000   29

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 4

Enter the dept that you want to search
CSE

ID  Name    Dept    Salary  Age
123 Raju    CSE     24000   26
125 John    CSE     27000   29

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 5

Enter the salary that you want to search
27000

ID  Name    Dept    Salary  Age
125 John    CSE     27000   29

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 6

Enter the age that you want to search
26

ID  Name    Dept    Salary  Age
123 Raju    CSE     24000   26

1.Add Record
2.Display Records
3.Search Employee by ID
4.Search Employee by Dept
5.Search Employee by salary
6.Search Employee by Age
7.Exit
Enter your choice : 7
****************************************/
```

Listing 1.3: out1.c

Chapter 2

Stack Implementation

Question

Write a C program to implement STACK to perform the PUSH, POP and DISPLAY operations.

2.1 C Code - Array Representation

====================================

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

#define MAX 4

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

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

    for(;;)
    {
        printf("\nSTACK OPERATIONS\n");
        printf("====================");
        printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.PEEK\n5.EXIT\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: fnPush(stkArray, &top);
                    break;

            case 2: iElem = fnPop(stkArray, &top);
                    if(iElem != -1)
                        printf("\nPopped Element is %d\n", iElem);
                    break;

            case 3: fnDisplay(stkArray, top);
                    break;

            case 4: if(!fnStkEmpty(top))
                    {
                        iElem = fnPeek(stkArray, top);
                        printf("\nElement at the  top of the stack is %d\n", iElem);
                    }
                    else
                        printf("\nEmpty Stack\n");
                    break;

            case 5: exit(1);

            default: printf("\nWrong choice\n");
        }
    }
    return 0;
}

bool fnStkFull(int t)
{
    return ((t == MAX-1) ? true : false);
}

bool fnStkEmpty(int t)
{
    return ((t == -1) ? true : false);
}

void fnPush(int stk[], int *t)
{
    int iElem;
    if(fnStkFull(*t))
    {
        printf("\nStack Overflow\n");
        return;
    }
    printf("\nEnter element to be pushed onto the stack\n");
    scanf("%d", &iElem);

    *t = *t + 1;
    stk[*t] = iElem;
}

int fnPop(int stk[], int *t)
{
    int iElem;
    if(fnStkEmpty(*t))
    {
        printf("\nStack Underflow\n");
        return -1;
    }
    iElem = stk[*t];
    *t = *t - 1;

    return iElem;
}

void fnDisplay(int stk[], int t)
{
    int i;
    if(fnStkEmpty(t))
```

```c
    {
        printf("\nStack Empty\n");
        return;
    }
    printf("\nStack Contents are: \n");
    for(i = t ; i > -1; --i)
    {
        printf("\t%d\n", stk[i]);
    }
}

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

Listing 2.1: 02Stack.c

2.2 C Code - Structure Representation

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

#define MAX 5

typedef struct{
    int stkArray[MAX];
    int top;
}STACK_TYPE;

bool fnStkFull(STACK_TYPE);
bool fnStkEmpty(STACK_TYPE);
void fnPush(STACK_TYPE*, int);
int fnPop(STACK_TYPE*);
void fnDisplay(STACK_TYPE);
int fnPeek(STACK_TYPE);

int main()
{

    STACK_TYPE myStack;
    myStack.top = -1;

    int iElem, iChoice;

    for(;;)
    {
        printf("\nSTACK OPERATIONS\n");
        printf("====================");
        printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.PEEK\n5.EXIT\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: fnPush(stkArray, &top);
                    break;

            case 2: iElem = fnPop(stkArray, &top);
                    if(iElem != -1)
```

```
                        printf("\nPopped Element is %d\n", iElem);
                    break;

                case 3: fnDisplay(stkArray, top);
                    break;

                case 4: if(!fnStkEmpty(top))
                    {
                        iElem = fnPeek(stkArray, top);
                        printf("\nElement at the  top of the stack is %d\n", iElem
    );
                    }
                    else
                        printf("\nEmpty Stack\n");
                    break;

                case 5: exit(1);

                default: printf("\nWrong choice\n");
            }
        }
        return 0;
}

bool fnStkFull(int t)
{
    return ((t == MAX-1) ? true : false);
}

bool fnStkEmpty(int t)
{
    return ((t == -1) ? true : false);
}

void fnPush(int stk[], int *t)
{
    int iElem;
    if(fnStkFull(*t))
    {
        printf("\nStack Overflow\n");
        return;
    }
    printf("\nEnter element to be pushed onto the stack\n");
    scanf("%d", &iElem);

    *t = *t + 1;
    stk[*t] = iElem;
}

int fnPop(int stk[], int *t)
{
    int iElem;
    if(fnStkEmpty(*t))
    {
        printf("\nStack Underflow\n");
        return -1;
    }
    iElem = stk[*t];
    *t = *t - 1;

    return iElem;
}
```

```c

void fnDisplay(int stk[], int t)
{
    int i;
    if(fnStkEmpty(t))
    {
        printf("\nStack Empty\n");
        return;
    }
    printf("\nStack Contents are: \n");
    for(i = t ; i > -1; --i)
    {
        printf("\t%d\n", stk[i]);
    }
}

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

Listing 2.2: 02Stack2Struct.c

Output

=================================

```
/***************************************
putta:Programs$ gcc 02Stack.c
putta:Programs$ ./a.out

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
2

Stack Underflow

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
3

Stack Empty

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
```

```
Enter your choice
1

Enter element to be pushed onto the stack
1

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
1

Enter element to be pushed onto the stack
2

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
1

Enter element to be pushed onto the stack
3

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
3

Stack Contents are:
    3
    2
    1

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
4

Element at the  top of the stack is 3

STACK OPERATIONS
====================
1.PUSH
```

```
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
2

Popped Element is 3

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
4

Element at the  top of the stack is 2

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
1

Enter element to be pushed onto the stack
3

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
1

Enter element to be pushed onto the stack
4

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
3

Stack Contents are:
    4
    3
    2
    1
```

```

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
1

Stack Overflow

STACK OPERATIONS
====================
1.PUSH
2.POP
3.DISPLAY
4.PEEK
5.EXIT
Enter your choice
5
****************************************/
```

Listing 2.3: out2.c

Chapter 3

Infix to Postfix Conversion

Question

Write a C program to convert the given infix expression to postfix expression.

C Code

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

#define STK_SIZE 10

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

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

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

    fnPush(acStack, &top, '#');
    for(i=0;acExpr[i]!='\0'; ++i)
    {
        cSymb = acExpr[i];
        if(isdigit(cSymb))
        {
            fnPush(acStack, &top, 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("\nInfix Expression is %s\n", acExpr);
    printf("\nPostfix Expression is %s\n", acPost);
    return 0;
}

void fnPush(char Stack[], int *t , char elem)
{
    *t = *t + 1;
    Stack[*t] = elem;

}

char fnPop(char Stack[], int *t)
{
    char elem;
    elem = Stack[*t];
    *t = *t -1;
    return elem;
}

int fnPrecd(char ch)
{
    switch(ch)
    {
        case '#' :  return -1;
        case '(' :  return 0;
        case '+' :
        case '-' :  return 1;
        case '*' :
        case '/' :  return 2;
        case '^' :  return 3;
    }
}
```

Listing 3.1: 03ConvInfix.c

Output

```
/****************************************
putta:Programs$ gcc 03ConvInfix.c
putta:Programs$ ./a.out
Enter a valid infix expression
(a^b^c)+(d/(e-f))

```

```
Infix Expression is (a^b^c)+(d/(e-f))

Postfix Expression is abc^^def-/+

putta:Programs$ ./a.out
Enter a valid infix expression
(a*(b-c)/(a+b*c))

Infix Expression is (a*(b-c)/(a+b*c))

Postfix Expression is abc-*abc*+/

*************************************/
```

Listing 3.2: out3.c

Chapter 4

Evaluation of Prefix Expression

Question

Write a C program to evaluate the given prefix expression.

C Code

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

#define STK_SIZE 10

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

int main()
{
    int iaStack[50], i, iOp1, iOp2, iRes;
    char acExpr[50], cSymb;
    int top = -1;

    printf("\nEnter a valid prefix expression\n");
    scanf("%s", acExpr);

    for(i=strlen(acExpr)-1; i>=0; i--)
    {
        cSymb = acExpr[i];
        if(isdigit(cSymb))
        {
            fnPush(iaStack, &top, cSymb-'0');
        }
        else
        {
            iOp1 = fnPop(iaStack, &top);
            iOp2 = fnPop(iaStack, &top);
            switch(cSymb)
            {
                case '+' :  iRes = iOp1 + iOp2;
                            break;
                case '-' :  iRes = iOp1 - iOp2;
                            break;
                case '*' :  iRes = iOp1 * iOp2;
                            break;
                case '/' :  iRes = iOp1 / iOp2;
                            break;
            }
            fnPush(iaStack, &top, iRes);
        }

    }
    iRes = fnPop(iaStack, &top);
    printf("\nValue of %s expression is %d\n", acExpr, iRes);
    return 0;
}

void fnPush(int Stack[], int *t , int elem)
{
    *t = *t + 1;
    Stack[*t] = elem;

}

int fnPop(int Stack[], int *t)
{
    int elem;
    elem = Stack[*t];
    *t = *t - 1;
    return elem;
}
```

Listing 4.1: 04EvalPrefix.c

Output

Chapter 5

Linear Queue

Question

Write a C program to implement ordinary QUEUE to perform the insertion, deletion and display operations.

5.1 C Code - Array Representation

==================================

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

#define QUEUE_SIZE 5

void fnInsertRear(int [], int*, int);
int fnDeleteFront(int[], int*, int*);
void fnDisplay(int [], int, int);
bool fnQueueFull(int[], int);
bool fnQueueEmpty(int[], int, int);

int main()
{
    int myQueue[QUEUE_SIZE];
    int iFront = 0, iRear = -1;
    int iElem, iChoice;

    for(;;)
    {
        printf("\nQueue Operations\n");
        printf("=====================");
        printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: if(!fnQueueFull(myQueue, iRear))
                    {
                        printf("\nEnter an element : ");
                        scanf("%d", &iElem);
                        fnInsertRear(myQueue, &iRear, iElem);
                    }
                    else
                    {
                        printf("\nQueue is Full\n");
                    }

                    break;
                case 2: if(!fnQueueEmpty(myQueue, iFront, iRear))
                        {
                            iElem = fnDeleteFront(myQueue, &iFront, &iRear);
                            printf("\nDeleted element is %d\n", iElem);
                        }
                        else
                        {
                            printf("\nQueue is Empty\n");
                        }

                    break;
                case 3: if(!fnQueueEmpty(myQueue, iFront, iRear))
                        {
                            printf("\nContents of the Queue is \n");
                            fnDisplay(myQueue, iFront, iRear);
                        }
                        else
                        {
                            printf("\nQueue is Empty\n");
                        }

                    break;

                case 4: exit(0);

                default: printf("\nInvalid choice\n");

                    break;
        }
    }
    return 0;
}

bool fnQueueFull(int queue[], int r)
{
    if(r == QUEUE_SIZE-1)
        return true;
    else
        return false;
}

bool fnQueueEmpty(int queue[], int f, int r)
{
    if(r == f-1)
        return true;
    else
        return false;
}

void fnInsertRear(int queue[], int *r, int iVal)
{
    *r = *r + 1;
    queue[*r] = iVal;
}

int fnDeleteFront(int queue[], int *f, int *r)
{
    int iElem;
    iElem = queue[*f];

    if(*f == *r)
```

```c
    {
        *f = 0;
        *r = -1;
    }
    else
    {
        *f = *f + 1;
    }
    return iElem;
}

void fnDisplay(int queue[], int f, int r)
{
    int i;
    for(i=f; i<=r; i++)
    {
        printf("%d\t", queue[i]);
    }
    printf("\n");
}
```

Listing 5.1: 05LinearQueue.c

5.2 C Code - Structure Representation

```
==================================
```

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


#define QUEUE_SIZE 5

typedef struct
{
    int Queue[QUEUE_SIZE];
    int iFront, iRear;
}QUEUE_T;


void fnInsertRear(QUEUE_T*, int);
int fnDeleteFront(QUEUE_T*);
void fnDisplay(QUEUE_T);
bool fnQueueFull(QUEUE_T);
bool fnQueueEmpty(QUEUE_T);

int main()
{
    QUEUE_T myQueue;
    int iElem, iChoice;

    myQueue.iFront = 0;
    myQueue.iRear = -1;


    for(;;)
    {
        printf("\nQueue Operations\n");
        printf("=====================");
        printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: if(!fnQueueFull(myQueue))
                    {
                        printf("\nEnter an element : ");
                        scanf("%d", &iElem);
                        fnInsertRear(&myQueue, iElem);
                    }
                    else
                    {
                        printf("\nQueue is Full\n");
                    }

                break;
            case 2: if(!fnQueueEmpty(myQueue))
                    {
                        iElem = fnDeleteFront(&myQueue);
                        printf("\nDeleted element is %d\n", iElem);
                    }
                    else
                    {
                        printf("\nQueue is Empty\n");
                    }

                break;
            case 3: if(!fnQueueEmpty(myQueue))
                    {
                        printf("\nContents of the Queue is \n");
                        fnDisplay(myQueue);
                    }
                    else
                    {
                        printf("\nQueue is Empty\n");
                    }

                break;

            case 4: exit(0);

            default: printf("\nInvalid choice\n");

                break;
        }
    }
    return 0;
}

bool fnQueueFull(QUEUE_T myQ)
{
    if(myQ.iRear == QUEUE_SIZE-1)
        return true;
    else
        return false;
}

bool fnQueueEmpty(QUEUE_T myQ)
{
    if(myQ.iRear == myQ.iFront-1)
        return true;
    else
        return false;
```

```c
}

void fnInsertRear(QUEUE_T *myQ, int iVal)
{
     (myQ->iRear)++;
    myQ->Queue[myQ->iRear] = iVal;
}

int fnDeleteFront(QUEUE_T *myQ)
{
    int iElem;
    iElem = myQ->Queue[myQ->iFront];

    if(myQ->iFront == myQ->iRear)
    {
        myQ->iFront = 0;
        myQ->iRear = -1;
    }
    else
    {
        myQ->iFront = myQ->iFront + 1;
    }
    return iElem;
}

void fnDisplay(QUEUE_T myQ)
{
    int i;
    for(i=myQ.iFront; i<=myQ.iRear; i++)
    {
        printf("%d\t", myQ.Queue[i]);
    }
    printf("\n");
}
```

Listing 5.2: 05StructLinearQueue.c

Output

Chapter 6

Circular Queue

Question

Write a C program to implement CIRCULAR QUEUE to perform the insertion, deletion and display operations.

6.1 C Code - Array Representation

==================================

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

#define QUEUE_SIZE 5

void fnInsertRear(int [], int*, int*, int);
int fnDeleteFront(int[], int*, int*);
void fnDisplay(int [], int, int);
bool fnQueueFull(int[], int, int);
bool fnQueueEmpty(int[], int, int);

int main()
{
    int myQueue[QUEUE_SIZE];
    int iFront = -1, iRear = -1;
    int iElem, iChoice;

    for(;;)
    {
        printf("\nQueue Operations\n");
        printf("=====================");
        printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: if(!fnQueueFull(myQueue, iFront, iRear))
                    {
                        printf("\nEnter an element : ");
                        scanf("%d", &iElem);
                        fnInsertRear(myQueue, &iFront, &iRear, iElem);
                    }
                    else
                    {
                        printf("\nQueue is Full\n");
                    }

                break;
            case 2: if(!fnQueueEmpty(myQueue, iFront, iRear))
                    {
                        iElem = fnDeleteFront(myQueue, &iFront, &iRear);
                        printf("\nDeleted element is %d\n", iElem);
                    }
                    else
                    {
                        printf("\nQueue is Empty\n");
                    }

                break;
            case 3: if(!fnQueueEmpty(myQueue, iFront, iRear))
                    {
                        printf("\nContents of the Queue is \n");
                        fnDisplay(myQueue, iFront, iRear);
                    }
                    else
                    {
                        printf("\nQueue is Empty\n");
                    }

                break;

            case 4: exit(0);

            default: printf("\nInvalid choice\n");

                break;
        }
    }
    return 0;
}

bool fnQueueFull(int queue[], int f, int r)
{
    if((r+1) % QUEUE_SIZE == f)
        return true;
    else
        return false;
}

bool fnQueueEmpty(int queue[], int f, int r)
{
    if(f == -1)
        return true;
    else
        return false;
}

void fnInsertRear(int queue[], int *f, int *r, int iVal)
{
    if(*r == -1)
    {
        *f = *f + 1;
        *r = *r + 1;
    }
    else
        *r = (*r + 1)%QUEUE_SIZE;

    queue[*r] = iVal;
}
```

```c

int fnDeleteFront(int queue[], int *f, int *r)
{
    int iElem;
    iElem = queue[*f];

    if(*f == *r)
    {
        *f = -1;
        *r = -1;
    }
    else
    {
        *f = (*f + 1)%QUEUE_SIZE;
    }
    return iElem;
}

void fnDisplay(int queue[], int f, int r)
{
    int i;
    if(f<=r)
    {
        for(i=f; i<=r; i++)
        {
            printf("%d\t", queue[i]);
        }
        printf("\n");
    }
    else
    {
        for(i=f; i<=QUEUE_SIZE-1; i++)
        {
            printf("%d\t", queue[i]);
        }
        for(i=0; i<=r; i++)
        {
            printf("%d\t", queue[i]);
        }
        printf("\n");
    }
}
```

Listing 6.1: 06CircQueue.c

6.2 C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

typedef struct
{
    int Queue[QUEUE_SIZE];
    int iFront, iRear;
}QUEUE_T;
```

```c


void fnInsertRear(QUEUE_T*, int);
int fnDeleteFront(QUEUE_T*);
void fnDisplay(QUEUE_T);
bool fnQueueFull(QUEUE_T);
bool fnQueueEmpty(QUEUE_T);

int main()
{
    QUEUE_T myQueue;
    int iElem, iChoice;

    myQueue.iFront = -1;
    myQueue.iRear = -1;


    for(;;)
    {
        printf("\nQueue Operations\n");
        printf("=====================");
        printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: if(!fnQueueFull(myQueue))
                    {
                        printf("\nEnter an element : ");
                        scanf("%d", &iElem);
                        fnInsertRear(&myQueue, iElem);
                    }
                    else
                    {
                        printf("\nQueue is Full\n");
                    }

                break;
            case 2: if(!fnQueueEmpty(myQueue))
                    {
                        iElem = fnDeleteFront(&myQueue);
                        printf("\nDeleted element is %d\n", iElem);
                    }
                    else
                    {
                        printf("\nQueue is Empty\n");
                    }

                break;
            case 3: if(!fnQueueEmpty(myQueue))
                    {
                        printf("\nContents of the Queue is \n");
                        fnDisplay(myQueue);
                    }
                    else
                    {
                        printf("\nQueue is Empty\n");
                    }

                break;

            case 4: exit(0);

            default: printf("\nInvalid choice\n");

                break;
        }
    }
    return 0;
}

bool fnQueueFull(QUEUE_T myQ)
{
    if((myQ.iRear+1) % QUEUE_SIZE == myQ.iFront)
        return true;
    else
        return false;
}

bool fnQueueEmpty(QUEUE_T myQ)
{
    if(myQ.iFront == -1)
        return true;
    else
        return false;
}

void fnInsertRear(QUEUE_T *myQ, int iVal)
{
    if(myQ->iRear == -1)
    {
        (myQ->iRear)++;
        (myQ->iFront)++;
    }
    else
        myQ->iRear = (myQ->iRear + 1) % QUEUE_SIZE;

    myQ->Queue[myQ->iRear] = iVal;
}

int fnDeleteFront(QUEUE_T *myQ)
{
    int iElem;
    iElem = myQ->Queue[myQ->iFront];

    if(myQ->iFront == myQ->iRear)
    {
        myQ->iFront = myQ->iRear = -1;
    }
    else
    {
        myQ->iFront = (myQ->iFront + 1)%QUEUE_SIZE;
    }
    return iElem;
}

void fnDisplay(QUEUE_T myQ)
{
    int i;
    if(myQ.iFront<=myQ.iRear)
    {
        for(i=myQ.iFront; i<=myQ.iRear; i++)
        {
            printf("%d\t", myQ.Queue[i]);
        }
        printf("\n");
    }
    else
    {
        for(i=myQ.iFront; i<QUEUE_SIZE; i++)
        {
            printf("%d\t", myQ.Queue[i]);
        }
        for(i=0; i<=myQ.iRear; i++)
        {
            printf("%d\t", myQ.Queue[i]);
        }
        printf("\n");

    }
}
```

Listing 6.2: 06StructCircularQueue.c

Output

Chapter 7

Singly Linked List

Question

Write a C program to perform the following operations using singly linked list:

a to insert a node at the end of the list.

b to insert a node at the end of the list.

c to insert a node at the specified position in the list.

d to display the contents of the list.

e to reverse a given list.

C Code

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *link;
};

typedef struct node* NODEPTR;

NODEPTR fnGetNode(void);
void fnFreeNode(NODEPTR x);
NODEPTR fnInsertFront(int ,NODEPTR);
NODEPTR fnDeleteFront(NODEPTR);
NODEPTR fnInsertPosition(int ,int ,NODEPTR);
void fnDisplay(NODEPTR first);
NODEPTR fnReverse(NODEPTR);

int main()
{
    NODEPTR first = NULL;
    int iElem, iChoice, iPos;
    for(;;)
    {
        printf("\n1.Insert Front\n2.Delete Front\n3.Insert At Position");
        printf("\n4.Display\n5.Reverse\n6.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: printf("\nEnter a element\n");
```

```
                    scanf("%d", &iElem);
                    first = fnInsertFront(iElem, first);
                    break;

            case 2: first = fnDeleteFront(first);
                    break;

            case 3: printf("\nEnter a element\n");
                    scanf("%d", &iElem);
                    printf("\nEnter the position\n");
                    scanf("%d", &iPos);
                    first = fnInsertPosition(iElem, iPos, first);
                    break;

            case 4: fnDisplay(first);
                    break;

            case 5: first = fnReverse(first);
                    break;
            case 6: exit(0);
        }
    }
    return 0;
}

NODEPTR fnGetNode(void)
{
    NODEPTR newNode;
    newNode = ( NODEPTR ) malloc (sizeof(struct node));
    if(newNode == NULL)
    {
        printf("\nOut of Memory");
        exit(0);
    }
    return newNode;
}

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

NODEPTR fnInsertFront(int elem,NODEPTR first)
{
    NODEPTR temp;
    temp = fnGetNode();
    temp->info = elem;
    temp->link = first;
    first = temp;
    return first;

}

NODEPTR fnDeleteFront(NODEPTR first)
{
    NODEPTR temp;
    if(first == NULL)
    {
        printf("\nList is Empty cannot delete\n");
        return first;
    }

    temp = first;
    printf("\nElement deleted is %d\n", temp->info);
    fnFreeNode(temp);
    first = first->link;
    return first;
}

NODEPTR fnInsertPosition(int elem,int pos,NODEPTR first)
{
    NODEPTR temp,prev,cur;
    int count;

    temp = fnGetNode();
    temp->info = elem;
    temp->link = NULL;

    if(first == NULL && pos == 1)
        return temp;

    if(first == NULL)
    {
        printf("\nInvalid Position");
        return first;
    }

    if(pos == 1)
    {
        temp->link = first;
        return temp;
    }


    count = 1;
    prev = NULL;
    cur = first;

    while(cur != NULL && count != pos)
    {
        prev = cur;
        cur = cur->link;
        count++;
    }

    if(count == pos)
    {
        prev->link = temp;
        temp->link = cur;
        return first;
    }

    printf("\nInvalid Position");
    return first;
}

void fnDisplay(NODEPTR first)
{
    NODEPTR temp;

    if(first == NULL)
    {
        printf("\nList is Empty\n");
        return;
    }

    printf("\nList Contents\n");
    printf("==================================\n");
    for(temp = first; temp != NULL; temp = temp->link)
        printf("%4d",temp->info);
    printf("\n==================================\n");
    printf("\n\n");

}

NODEPTR fnReverse(NODEPTR first)
{
    NODEPTR cur, prev, next;
    prev = first;
    cur = first->link;
    next = cur->link;
    prev->link = NULL;
    while(cur->link!=NULL)
    {
        cur->link = prev;
        prev = cur;
        cur = next;
        next = next->link;
    }
    cur->link = prev;
    return cur;
}
```

Listing 7.1: 07SinglyLinkedList.c

Output

Chapter 8

Ordered Linked List

Question

Write a C program to construct two ordered singly linked lists with the following operations:

a insert into list1.

b insert into list2.

c to perform UNION(list1,list2)

d to perform INTERSECTION(list1,list2)

e display the contents of all three lists.

C Code

```c
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int info;
    struct node *link;
};

typedef struct node* NODEPTR;

NODEPTR fnGetNode(void);
NODEPTR fnInsertOrder(int ,NODEPTR);
NODEPTR fnInsertRear(int ,NODEPTR);
NODEPTR fnUnion(NODEPTR ,NODEPTR);
NODEPTR fnIntersection(NODEPTR ,NODEPTR);
void fnDisplay(NODEPTR first);

int main()
{
    NODEPTR list1 , list2, list3, list4;
    list1 = list2 = list3 = list4 = NULL;
    int iElem, iChoice;
    for(;;)
    {
        printf("\n1.Insert into List 1\n2.Insert into List 2\n3.Display");
        printf("\n4.Union\n5.Intersection\n6.Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&iChoice);
        switch(iChoice)
        {
            case 1: printf("\nEnter a element\n");
```

```
                        scanf("%d", &iElem);
                        list1 = fnInsertOrder(iElem, list1);
                        break;

                case 2: printf("\nEnter a element\n");
                        scanf("%d", &iElem);
                        list2 = fnInsertOrder(iElem, list2);
                        break;

                case 3: printf("\nList 1 Contents\n");
                        fnDisplay(list1);
                        printf("\nList 2 Contents\n");
                        fnDisplay(list2);
                        break;

                case 4: printf("\nList 1 Contents\n");
                        fnDisplay(list1);
                        printf("\nList 2 Contents\n");
                        fnDisplay(list2);
                        list3 = fnUnion(list1, list2);
                        printf("\nUnion\n");
                        fnDisplay(list3);
                        break;

                case 5: printf("\nList 1 Contents\n");
                        fnDisplay(list1);
                        printf("\nList 2 Contents\n");
                        fnDisplay(list2);
                        list4 = fnIntersection(list1, list2);
                        printf("\nIntersection\n");
                        fnDisplay(list4);
                        break;
                case 6: exit(0);
            }
        }
    return 0;
}

NODEPTR fnGetNode(void)
{
    NODEPTR newNode;
    newNode = ( NODEPTR ) malloc (sizeof(struct node));
    if(newNode == NULL)
    {
        printf("\nOut of Memory");
        exit(0);
    }
    return newNode;
}

NODEPTR fnIntersection(NODEPTR l1,NODEPTR l2)
{
    NODEPTR t1, t2, t3;
    t1 = l1;
    while(t1 != NULL)
    {
        t2 = l2;
        while(t2 != NULL)
        {
            if(t1->info == t2->info)
                t3 = fnInsertRear(t1->info, t3);
            t2 = t2->link;
        }
        t1 = t1->link;
    }
    return t3;
}


NODEPTR fnUnion(NODEPTR l1,NODEPTR l2)
{
    NODEPTR t1, t2, t3;
    t1 = l1;
    t2 = l2;
    while(t1 != NULL && t2 != NULL)
    {
        if(t1->info < t2->info)
        {
            t3 = fnInsertRear(t1->info, t3);
            t1 = t1->link;
        }
        else if(t1->info > t2->info)
        {
            t3 = fnInsertRear(t2->info, t3);
            t2 = t2->link;
        }
        else
        {
            t2 = t2->link;
        }

    }
    while(t1 != NULL)
    {
        t3 = fnInsertRear(t1->info, t3);
        t1 = t1->link;
    }
    while(t2 != NULL)
    {
        t3 = fnInsertRear(t2->info, t3);
        t2 = t2->link;
    }
    return t3;
}


NODEPTR fnInsertOrder(int elem, NODEPTR first)
{
    NODEPTR temp,prev,cur;

    temp = fnGetNode();
    temp->info = elem;
    temp->link = NULL;

    if(first == NULL)
        return temp;

    if(elem <= first->info)
    {
        temp->link = first;
        return temp;
    }

    prev = NULL;
```

```c
    cur = first;

    while(cur != NULL && elem > cur->info)
    {
        prev = cur;
        cur = cur->link;
    }
    prev->link = temp;
    temp->link = cur;
    return first;
}

void fnDisplay(NODEPTR first)
{
    NODEPTR temp;

    if(first == NULL)
    {
        printf("\nList is Empty\n");
        return;
    }

    printf("==================================\n");
    for(temp = first; temp != NULL; temp = temp->link)
        printf("%4d",temp->info);
    printf("\n==================================\n");
}

NODEPTR fnInsertRear(int iElem, NODEPTR first)
{
    NODEPTR temp,cur;
    temp = fnGetNode();
    temp->info = iElem;
    temp->link = NULL;

    if(first == NULL)
        return temp;

    cur = first;
    while(cur->link != NULL)
    {
        cur = cur->link;
    }
    cur->link = temp;
    return first;
}
```

Listing 8.1: 08OrderedSinglyLinkedList.c

Output

Chapter 9

Singly Linked List Applications

9.1 Stack using SLL

==================================

Write a C program to implement a STACK using singly linked list.

C Code

```
#include<stdio.h>                              /*CPP*/
#include<stdlib.h>

struct node
{
    int Info;
    struct node *link;
};

typedef struct node* NODEPTR;

NODEPTR fnGetNode(void);
void fnFreeNode(NODEPTR x);
NODEPTR fnPush(int ,NODEPTR);
NODEPTR fnPop(NODEPTR);
void fnDisplay(NODEPTR first);

int main(void)
{
    NODEPTR first = NULL;
    int iChoice,iElem;


    for(;;)
    {
        printf("\nSTACK OPERATIONS");
        printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
        printf("\nEnter your iChoice\n");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1: printf("\nEnter Element to be pushed onto Stack\n");
                scanf("%d",&iElem);
                first = fnPush(iElem,first);
                break;

            case 2: first = fnPop(first);
```

```
                break;

            case 3: fnDisplay(first);
                break;

            case 4: return;
        }
    }
    return 0;
}

NODEPTR fnGetNode()
{
    NODEPTR newborn;
    newborn = (NODEPTR)malloc(sizeof(struct node));

    if(newborn == NULL)
    {
        printf("\nMemory Overflow");
        exit(0);
    }
    return newborn;
}

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


NODEPTR fnPush(int iElem,NODEPTR first)  /*Insert front*/
{
        NODEPTR temp;

        temp = fnGetNode();

        temp->Info = iElem;

        temp->link = first;

        return temp;
}

NODEPTR fnPop(NODEPTR first)            /*Delete front*/
{
    NODEPTR temp;
    if(first == NULL)
    {
        printf("\nStack is empty cannot delete\n");
        return first;
    }
    temp = first;

    first = first->link;

    printf("\nElement deleted is %d \n",temp->Info);
    fnFreeNode(temp);

    return first;

}
```

```c


void fnDisplay(NODEPTR first)
{
    NODEPTR curr;
    if(first == NULL)
    {
        printf("\nStack is empty\n");
        return;
    }

    printf("\nThe contents of Stack are :\n");
    curr = first;
    while(curr != NULL)
    {
        printf("\n%d",curr->Info);
        curr = curr->link;
    }
    printf("\n");
}
```

Listing 9.1: 09aStackLL.c

Output

9.2 Queue using SLL

==================================

Write a C program to implement a QUEUE using singly linked list.

C Code

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

struct node
{
    int Info;
    struct node *link;
};

typedef struct node* NODEPTR;


NODEPTR fnGetNode()
{
    NODEPTR newborn;
    newborn = (NODEPTR)malloc(sizeof(struct node));

    if(newborn == NULL)
    {
        printf("\nMemory Overflow");
        exit(0);
    }
    return newborn;
}

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


NODEPTR fnIns_Rear(int iElem,NODEPTR first)
{
    NODEPTR temp,cur;

    temp = fnGetNode();

    temp->Info = iElem;

    temp->link = NULL;

    if(first == NULL)
        return temp;

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

    cur->link = temp;

    return first;
```

```c
}

NODEPTR fnDelFront(NODEPTR first)
{
    NODEPTR temp;
    if(first == NULL)
    {
        printf("\nQueue is empty cannot delete\n");
        return first;
    }
    temp = first;

    first = first->link;

    printf("\nElement deleted is %d \n",temp->Info);
    fnFreeNode(temp);

    return first;

}



void fnDisplay(NODEPTR first)
{
    NODEPTR curr;
    if(first == NULL)
    {
        printf("\nQueue is empty\n");
        return;
    }

    printf("\nThe contents of Queue are :\n");
    curr = first;
    while(curr != NULL)
    {
        printf("\n%d",curr->Info);
        curr = curr->link;
    }
    printf("\n");
}


main()
{
    NODEPTR first = NULL;
    int iChoice,iElem;

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

        switch(ch)
        {
            case 1: printf("\nEnter Element to be inserted\n");
                scanf("%d",&iElem);
                first = fnIns_Rear(iElem,first);
                break;

            case 2: first = fnDelFront(first);
                break;

            case 3: fnDisplay(first);
                break;

            case 4: return;
        }
    }

}


/*CPP*/
```

Listing 9.2: 09bQueueLL.c

Output

9.3 Polynomial Addition

==================================

Write a C program to implement addition of two polynomials using singly linked list..

C Code

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

struct polynomial{
    int coeff;
    int exponent;
    struct polynomial *link;
};
typedef struct polynomial *NODEPTR;

NODEPTR fnGetNode(void);
NODEPTR fnInsertRear(int, int, NODEPTR);
void fnDisplay(NODEPTR first);
NODEPTR fnAddPoly(NODEPTR, NODEPTR);
int evalPoly(NODEPTR, int);

int main()
{
    NODEPTR poly1, poly2, poly3;
    int i, iX, iRes, iDegree, iaCoeff[10];
    poly1 = poly2 = poly3 = NULL;

    printf("\nEnter the degree of polynomial 1\n");
    scanf("%d", &iDegree);
    printf("\nEnter the coefficients\n");
    for(i=iDegree;i>=0;i--)
    {
        scanf("%d", &iaCoeff[i]);
        poly1 = fnInsertRear(iaCoeff[i], i, poly1);
    }
    printf("\nEnter the degree of polynomial 2\n");
    scanf("%d", &iDegree);
    printf("\nEnter the coefficients\n");
    for(i=iDegree;i>=0;i--)
    {
        scanf("%d", &iaCoeff[i]);
        poly2 = fnInsertRear(iaCoeff[i], i, poly2);
    }
    poly3 = fnAddPoly(poly1, poly2);

    printf("Polynomial 1   :\t");
    fnDisplay(poly1);
    printf("Polynomial 2   :\t");
    fnDisplay(poly2);
    printf("Polynomial Sum :\t");
    fnDisplay(poly3);
    printf("\nEnter the value of x\n");
    scanf("%d", &iX);
    iRes = evalPoly(poly3, iX);
    printf("\nValue of the polynomial sum for x = %d is %d\n", iX, iRes);
    return 0;
}
```

```

NODEPTR fnInsertRear(int iCoeff, int iExp, NODEPTR first)
{
    NODEPTR temp,cur;
    temp = fnGetNode();
    temp->coeff = iCoeff;
    temp->exponent = iExp;
    temp->link = NULL;

    if(first == NULL)
        return temp;
    cur = first;
    while(cur->link != NULL)
    {
        cur = cur->link;
    }
    cur->link = temp;
    return first;
}

NODEPTR fnGetNode(void)
{
    NODEPTR newNode;
    newNode = ( NODEPTR ) malloc (sizeof(struct polynomial));
    if(newNode == NULL)
    {
        printf("\nOut of Memory");
        exit(0);
    }
    return newNode;
}

void fnDisplay(NODEPTR first)
{
    NODEPTR cur;
    for(cur = first; cur->link != NULL; cur = cur->link)
    {
//      printf(" (%d)x^(%d) +",cur->coeff,cur->exponent);
        printf(" (%d)x^%d +",cur->coeff,cur->exponent);
    }
    printf(" %d\n", cur->coeff);
}

NODEPTR fnAddPoly(NODEPTR poly1, NODEPTR poly2)
{
    NODEPTR tracker1, tracker2,poly3 = NULL;
    tracker1 = poly1;
    tracker2 = poly2;
    while(tracker1 != NULL && tracker2 != NULL)
    {
        if(tracker1->exponent > tracker2->exponent)
        {
            poly3 = fnInsertRear(tracker1->coeff, tracker1->exponent, poly3);
            tracker1 = tracker1->link;
        }
        else if(tracker1->exponent == tracker2->exponent)
        {
            poly3 = fnInsertRear(tracker1->coeff + tracker2->coeff, tracker1->
    exponent, poly3);
            tracker1 = tracker1->link;
            tracker2 = tracker2->link;
        }
```

```c
        else
        {
            poly3 = fnInsertRear(tracker2->coeff, tracker2->exponent, poly3);
            tracker2 = tracker2->link;
        }
    }
    return poly3;
}

int evalPoly(NODEPTR list, int x)
{
    int iSum = 0;
    NODEPTR cur = list;
    while(cur!=NULL)
    {
        iSum += (cur->coeff * (int)pow(x, cur->exponent));
        cur = cur->link;
    }
    return iSum;
}
```

Listing 9.3: 09cPolynomial.c

Output

```
putta:lab$ gcc 09_c_Polynomial.c -lm
putta:lab$ ./a.out

Enter the degree of polynomial 1
4

Enter the coefficients
3 4 5 6 7

Enter the degree of polynomial 2
3

Enter the coefficients
2 3 4 5
Polynomial 1   :         (3)x^4 + (4)x^3 + (5)x^2 + (6)x^1 + 7
Polynomial 2   :         (2)x^3 + (3)x^2 + (4)x^1 + 5
Polynomial Sum :         (3)x^4 + (6)x^3 + (8)x^2 + (10)x^1 + 12

Enter the value of x
2

Value of the polynomial sum for x = 2 is 160
```

Chapter 10

Doubly Linked Lists with Header Node

Question

Write a C program to perform the following operations using doubly linked list with header node. Header node should maintain the count of number of nodes in the list after each operation:

- ***a** **to insert a node next to a node whose information field specified.***
- ***b** **to delete first node if pointer to the last node is given.***
- ***c** **to delete a node at the specified position in the list.***
- ***d** **to display the contents of the list.***

C Code

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

struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};

typedef struct node* NODEPTR;

NODEPTR fnSwapNodes(NODEPTR head, int m , int n);
void fnDisplay(NODEPTR head);
NODEPTR fnDelElemPos(NODEPTR head, int iPos);
NODEPTR fnInsertNext(NODEPTR head, int iItem);
NODEPTR fnDeleteFirst(NODEPTR last);
NODEPTR fnInsertFront(NODEPTR head, int iItem);
void fnFreeNode(NODEPTR x);
NODEPTR fnGetNode(void);



int main()
{
    NODEPTR head,last;
    int iChoice, iItem, iKey, iPos, iM, iN;

    head = fnGetNode();
    head->rlink = head;
    head->llink = head;
    head->info = 0;

    for(;;)

    {
        printf("\n1.Insert Front\n2.Insert to the next of a given Node");
        printf("\n3.Delete First Node");
        printf("\n4.Delete a Node whose position is specified");
        printf("\n5.Display\n6.Swap Nodes\n7.Exit\n");

        printf("\nEnter your Choice\n");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1: printf("\nEnter the iItem to be inserted\n");
                    scanf("%d",&iItem);
                    head = fnInsertFront(head, iItem);
                    break;

            case 2: printf("\nEnter the key value of the node\n");
                    scanf("%d", &iKey);
                    head = fnInsertNext(head, iKey);
                    break;

            case 3: last = head->llink;
                    head = fnDeleteFirst(last);
                    break;

            case 4: printf("\nEnter the position of the element to be deleted\n");
                    scanf("%d",&iPos);
                    head = fnDelElemPos(head, iPos);
                    break;

            case 5: fnDisplay(head);
                    break;

            case 6: printf("\nEnter the positions m and n of the nodes to be swapped such that m < n\n");
                    scanf("%d%d",&iM, &iN);
                    if(iM > iN)
                    {
                        printf("\nInvalid input\n");
                    }
                    else
                    {
                        head = fnSwapNodes(head, iM, iN);
                    }
                    break;

            case 7: exit(0);
        }
    }
    return 0;
}

NODEPTR fnGetNode(void)
{
    NODEPTR x;
    x = ( NODEPTR ) malloc (sizeof(struct node));
    if(x == NULL)
    {
        printf("\nOut of Memory");
        exit(0);
    }
    return x;
}

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

NODEPTR fnInsertFront(NODEPTR head, int iItem)
{
    NODEPTR temp,cur;
    temp = fnGetNode();
    temp = fnGetNode();
    temp->info = iItem;

    cur = head->rlink;

    head->rlink = temp;
    temp->llink = head;
    temp->rlink = cur;
    cur->llink = temp;

    head->info += 1;

    return head;

}

NODEPTR fnDeleteFirst(NODEPTR last)
{
    NODEPTR second, first, head;

    if(last->rlink == last)
    {
        printf("\nList is Empty");
        return last;
    }
    head = last->rlink;
    first = head->rlink;
    second = first->rlink;

    head->rlink = second;
    second->llink = head;
    fnFreeNode(first);
    head->info -= 1;

    return head;
}

NODEPTR fnInsertNext(NODEPTR head, int iItem)
{
    NODEPTR temp,cur,next;

    if(head->rlink == head)
    {
        printf("\nList is Empty\n");
        return head;
    }

```

```c
    cur = head->rlink;

    while(cur != head && iItem != cur->info)
    {
        cur = cur->rlink;
    }
    if(cur == head)
    {
        printf("\nSpecified Node not found\n");
        return head;
    }

    next = cur->rlink;

    printf("\nEnter the item to be inserted to the next of %d\n",iItem);

    temp = fnGetNode();
    scanf("%d",&temp->info);

    cur->rlink = temp;
    temp->llink = cur;
    next->llink = temp;
    temp->rlink = next;
    head->info += 1;

    return head;

}

NODEPTR fnDelElemPos(NODEPTR head, int iPos)
{

    NODEPTR prev,cur,next;
    int count = 1;

    if(head->rlink == head)
    {
        printf("\nList is Empty\n");
        return head;
    }

    cur = head->rlink;

    while(cur != head && count != iPos)
    {
        cur = cur->rlink;
        count++;
    }

    if(count == iPos)
    {
        prev = cur->llink;
        next = cur->rlink;

        prev->rlink = next;
        next->llink = prev;
        head->info -= 1;

        fnFreeNode(cur);
    }

    if(cur == head)
    {
        printf("\nItem not found\n");
        return head;
    }

    return head;
}

void fnDisplay(NODEPTR head)
{
    NODEPTR temp;
    if(head->rlink == head)
    {
        printf("\nList is empty\n");
        return;
    }

    printf("Contents of the List is\n");
    for(temp = head->rlink; temp != head; temp = temp->rlink)
        printf("%d\t",temp->info);

    printf("\n");
    printf("\nThere are %d nodes in the list", head->info);
    printf("\n");


}


NODEPTR fnSwapNodes(NODEPTR head, int m , int n)
{
    int temp, count = 1;
    NODEPTR cur, mpos, npos;
    cur = head->rlink;

    while(cur != head && count != m)
    {
        cur = cur->rlink;
        count++;
    }

    if(cur != head)
    {
        mpos = cur;
    }
    else
    {
        printf("\nNode #%d does not exist\n", m);
        return head;
    }

    while(cur != head && count != n)
    {
        cur = cur->rlink;
        count++;
    }
    if(cur != head)
    {
        npos = cur;
    }
    else
    {
        printf("\nNode #%d does not exist\n", n);
        return head;
    }

    temp = mpos->info;
    mpos->info = npos->info;
    npos->info = temp;

    return head;
}
```

Listing 10.1: 10DoublyLinkedList.c

Output

Chapter 11

Double Ended Queue using Doubly Linked List

Question

Write a C program to implement DEQUE using doubly linked list to perform the insertion, deletion and display operations.

C Code

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

struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};
typedef struct node* NODEPTR;

NODEPTR fnGetNode(void);
void fnFreeNode(NODEPTR x);
NODEPTR fnInsertFront(NODEPTR head, int iItem);
NODEPTR fnDeleteFront(NODEPTR head);
NODEPTR fnInsertRear(NODEPTR head, int iItem);
NODEPTR fnDeleteRear(NODEPTR head);
void fnDisplay(NODEPTR head);

int main()
{
    NODEPTR head;
    int iChoice, iItem;

    head = fnGetNode();
    head->rlink = head;
    head->llink = head;

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

        switch(iChoice)
        {
            case 1: printf("\nEnter the iItem to be inserted\n");
                    scanf("%d",&iItem);
                    head = fnInsertFront(head, iItem);
                    break;

            case 2: printf("\nEnter the iItem to be inserted\n");
                    scanf("%d",&iItem);
                    head = fnInsertRear(head, iItem);
                    break;

            case 3: head = fnDeleteFront(head);
                    break;

            case 4: head = fnDeleteRear(head);
                    break;

            case 5: fnDisplay(head);
                    break;

            case 6: exit(0);
        }
    }
    return 0;
}

NODEPTR fnGetNode(void)
{
    NODEPTR x;
    x = ( NODEPTR ) malloc (sizeof(struct node));
    if(x == NULL)
    {
        printf("\nOut of Memory");
        exit(0);
    }
    return x;
}

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

NODEPTR fnInsertFront(NODEPTR head, int iItem)
{
    NODEPTR temp,cur;
    temp = fnGetNode();
    temp = fnGetNode();
    temp->info = iItem;

    cur = head->rlink;
    head->rlink = temp;
    temp->llink = head;
    temp->rlink = cur;
    cur->llink = temp;
    return head;
}

NODEPTR fnInsertRear(NODEPTR head, int iItem)
{
    NODEPTR temp,cur;
    temp = fnGetNode();
    temp = fnGetNode();
    temp->info = iItem;

    cur = head->llink;
    head->llink = temp;
    temp->rlink = head;
    temp->llink = cur;
    cur->rlink = temp;
    return head;
}

NODEPTR fnDeleteFront(NODEPTR head)
{
    NODEPTR second, first;
    if(head->rlink == head)
    {
        printf("\nList is Empty\n");
        return head;
    }
    first = head->rlink;
    second = first->rlink;

    head->rlink = second;
    second->llink = head;
    printf("\nElement deleted is %d\n", first->info);
    fnFreeNode(first);
    return head;
}

NODEPTR fnDeleteRear(NODEPTR head)
{
    NODEPTR secondLast, last;
    if(head->rlink == head)
    {
        printf("\nList is Empty\n");
        return head;
    }
    last = head->llink;
    secondLast = last->llink;

    head->llink = secondLast;
    secondLast->rlink = head;
    printf("\nElement deleted is %d\n", last->info);
    fnFreeNode(last);
    return head;
}

void fnDisplay(NODEPTR head)
{
    NODEPTR temp;
    if(head->rlink == head)
    {
        printf("\nList is empty\n");
        return;
    }
    printf("Contents of the List is\n");
    for(temp = head->rlink; temp != head; temp = temp->rlink)
        printf("%d\t",temp->info);
    printf("\n");
```

```
}
```

Listing 11.1: 11DequeueDLL.c

Output

Chapter 12

Binary Search Tree

Question

Write a C program to perform the following operations:

a Construct a binary search tree of integers.

b Traverse the tree in inorder/ preorder/ postorder.

c Delete a given node from the BST.

C Code

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

struct node
{
    int info;
    struct node *lchild;
    struct node *rchild;
};
typedef struct node* NODEPTR;

/* FUNCTION PROTOTYPES */
NODEPTR fnGetNode(void);
void fnFreeNode(NODEPTR x);
NODEPTR fnInsertNode(int, NODEPTR);
void fnInOrder(NODEPTR);
void fnPreOrder(NODEPTR);
void fnPostOrder(NODEPTR);
NODEPTR fnDeleteNode(NODEPTR, int);
NODEPTR fnMinValueNode(NODEPTR);

int main()
{
    NODEPTR root = NULL;
    int iChoice,iItem;
    for(;;)
    {
        printf("\n1.Insert a node\n2.Inorder traversal\n3.Preorder traversal");
        printf("\n4.Postorder traversal\n5.Delete a node\n6.Exit\n");
        printf("\nEnter your choice");
        scanf("%d",&iChoice);

        switch(iChoice)
        {
            case 1: printf("Enter the item to be inserted \n");
```

```
                    scanf("%d",&iItem);
                    root = fnInsertNode(iItem,root);
                    break;

            case 2: if(root ==NULL)
                    {
                        printf("\nTree is Empty\n");
                    }
                    else
                    {
                        printf("\nInorder Traversal is :\n");
                        fnInOrder(root);
                        printf("\n");
                    }
                    break;

            case 3: if(root ==NULL)
                    {
                        printf("\nTree is Empty\n");
                    }
                    else
                    {
                        printf("\nPreorder Traversal is :\n");
                        fnPreOrder(root);
                        printf("\n");
                    }
                    break;

            case 4: if(root ==NULL)
                    {
                        printf("\nTree is Empty\n");
                    }
                    else
                    {
                        printf("\nPostorder Traversal is :\n");
                        fnPostOrder(root);
                        printf("\n");
                    }
                    break;

            case 5: printf("\nEnter node to be deleted :  ");
                    scanf("%d", &iItem);
                    root = fnDeleteNode(root, iItem);
                    break;

            case 6: exit(0);

            default: printf("Wrong choice\n");
                     break;

        }

    }
    return 0;
}

NODEPTR fnGetNode(void)
{
    NODEPTR x;
    x = ( NODEPTR ) malloc (sizeof(struct node));
    if(x == NULL)
    {
        printf("\nOut of Memory");
        exit(0);
    }
    return x;
}

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

NODEPTR fnInsertNode(int iItem,NODEPTR root)
{
    NODEPTR temp,prev,cur;

    temp = fnGetNode();
    temp->info = iItem;
    temp->lchild = NULL;
    temp->rchild = NULL;

    if(root == NULL)
    return temp;

    prev = NULL;
    cur = root;

    while(cur != NULL)
    {
        prev = cur;

        if(iItem == cur->info)
        {
            printf("\nDuplicate items not allowed\n");
            fnFreeNode(temp);
            return root;
        }

        cur = (iItem < cur->info)? cur->lchild: cur->rchild;
    }

    if(iItem < prev->info)
        prev->lchild = temp;
    else
        prev->rchild = temp;

    return root;

}

void fnPreOrder(NODEPTR root)
{
    if(root != NULL)
    {
        printf("%d\t",root->info);
        fnPreOrder(root->lchild);
        fnPreOrder(root->rchild);
    }
}

void fnInOrder(NODEPTR root)
{
    if(root != NULL)
    {
        fnInOrder(root->lchild);
        printf("%d\t",root->info);
        fnInOrder(root->rchild);
    }
}

void fnPostOrder(NODEPTR root)
{
    if(root != NULL)
    {
        fnPostOrder(root->lchild);
        fnPostOrder(root->rchild);
        printf("%d\t",root->info);
    }
}

NODEPTR fnDeleteNode(NODEPTR root, int iItem)
{
    NODEPTR prev, cur, leftChild, newParent;

    if(root == NULL)
    {
        printf("\nBST is empty, cannot delete");
        return root;
    }
    // If the item to be deleted is smaller than the root's item,
    // then it lies in left subtree
    if (iItem < root->info)
        root->lchild = fnDeleteNode(root->lchild, iItem);

    // If the item to be deleted is greater than the root's item,
    // then it lies in right subtree
    else if (iItem > root->info)
        root->rchild = fnDeleteNode(root->rchild, iItem);

    // if item is same as root's item, then This is the node
    // to be deleted
    else
    {
        // node with only one child or no child
        if (root->lchild == NULL)
        {
            struct node *temp = root->rchild;
            free(root);
            return temp;
        }
        else if (root->rchild == NULL)
        {
            struct node *temp = root->lchild;
            free(root);
            return temp;
        }

        // node with two children: Get the inorder successor (smallest
        // in the right subtree)
        NODEPTR temp = fnMinValueNode(root->rchild);

        // Copy the inorder successor's content to this node
        root->info = temp->info;

        // Delete the inorder successor
        root->rchild = fnDeleteNode(root->rchild, temp->info);
    }
    return root;
}

NODEPTR fnMinValueNode(NODEPTR node)
{
    NODEPTR current = node;

    /* loop down to find the leftmost leaf */
    while (current->lchild != NULL)
        current = current->lchild;

    return current;
}
```

Listing 12.1: 12BinarySearchTree.c

Output

Chapter 13

Expression Tree

Question

Write a C program to construct an expression tree for a given postfix expression and evaluate the expression tree.

C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
// An expression tree node
struct ExpTree
{
    char value;
    struct ExpTree *left, *right;
};
typedef struct ExpTree* NODEPTR;

bool isOperator(char c);
void inorder(NODEPTR t);
NODEPTR newNode(int v);
NODEPTR constructTree(char postfix[]);
void push(NODEPTR[], NODEPTR, int*);
NODEPTR pop(NODEPTR[], int*);
NODEPTR peep(NODEPTR[], int*);

int main()
{
    char postfix[30];
    printf("\nEnter a postfix expression\n");
    scanf("%s", postfix);
    NODEPTR et = constructTree(postfix);
    printf("infix expression is \n");
    inorder(et);
    printf("\n");
    return 0;
}
// A utility function to check if 'c' is an operator
bool isOperator(char c)
{
    if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
        return true;
    return false;
}
```

```

// Utility function to do inorder traversal
void inorder(NODEPTR t)
{
    if(t)
    {
        inorder(t->left);
        printf("%c ", t->value);
        inorder(t->right);
    }
}

NODEPTR newNode(int v)
{
    NODEPTR temp =(NODEPTR) malloc(sizeof(struct ExpTree));
    temp->left = temp->right = NULL;
    temp->value = v;
    return temp;
}

NODEPTR constructTree(char postfix[])
{
    NODEPTR stack[100];
    int i, top = -1;
    NODEPTR t, t1, t2;

    // Traverse through every character of input expression
    for(i=0; i<strlen(postfix); i++)
    {
        // If operand, simply push into stack
        if(!isOperator(postfix[i]))
        {
            t = newNode(postfix[i]);
            push(stack, t, &top);
        }
        else // operator
        {
            t = newNode(postfix[i]);
            // Pop two top nodes
            t1 = peep(stack, &top); // Store top
            pop(stack, &top);       // Remove top
            t2 = peep(stack, &top);
            pop(stack, &top);
            //  make them children
            t->right = t1;
            t->left = t2;
            // Add this subexpression to stack
            push(stack, t, &top);
        }
    }
    //  only element will be root of expression tree
    t = peep(stack, &top);
    pop(stack, &top);
    return t;
}

void push(NODEPTR st[], NODEPTR p, int *t)
{
    *t = *t + 1;
    st[*t] = p;
}

```

```
NODEPTR pop(NODEPTR st[], int *t)
{
    NODEPTR temp;
    temp = st[*t];
    *t = *t - 1;
    return temp;
}

NODEPTR peep(NODEPTR st[], int *t)
{
    return st[*t];
}
```

Listing 13.1: 13ExpressionTree.c

Output