SIDDAGANGA INSTITUTE OF TECHNOLOGY

www.sit.ac.in

Data Structures Laboratory (3CSL01)

III Semester CSE
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.

- All experiments must be included in practical examinations.

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");
```

```c
29          printf("\nEnter your choice : ");
30          scanf("%d",&iChoice);
31
32          switch(iChoice)
33          {
34              case 1: fnAddRecord();
35                      break;
36
37              case 2: printf("\n Employee Details \n");
38                      fnDisplayAllRecords();
39                      break;
40
41              case 3: printf("\nEnter the emp_id that you want to search\n");
42                      scanf("%d",&id);
43                      fnSearchEmpID(id);
44                      break;
45
46              case 4: printf("\nEnter the dept that you want to search\n");
47                      scanf("%s",dept);
48                      fnSearchEmpDept(dept);
49                      break;
50
51              case 5: printf("\nEnter the salary that you want to search\n");
52                      scanf("%d",&sal);
53                      fnSearchEmpSal(sal);
54                      break;
55
56              case 6: printf("\nEnter the age that you want to search\n");
57                      scanf("%d",&age);
58                      fnSearchEmpAge(age);
59                      break;
60
61              case 7: exit(0);
62          }
63      }
64      return 0;
65  }
66
67  void fnDisplayAllRecords()
68  {
69      int iCount = 0;
70      employee_t ep;
71      FILE *fp;
72
73      fp = fopen("emp.dat", "r");
74      if(fp==NULL)
75      {
76          printf("\nFile does not exist\n");
77          return;
78      }
79      printf("\nID\tName\tDept\tSalary\tAge\n");
80      while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
    emp_salary, &ep.emp_age)!=EOF)
81      {
82          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);
83          iCount++;
84      }
85      if(0 == iCount)
86          printf("\nNo Records found\n");
87      fclose(fp);
88  }
```

```c
89
90  void fnAddRecord()
91  {
92      FILE *fp;
93      employee_t emp;
94
95      printf("\nEnter Employee details\n");
96      printf("\nID : ");
97      scanf("%d", &emp.emp_id);        getchar();
98      printf("\nName : ");
99      scanf("%s", emp.emp_name);
100     printf("\nDept : ");
101     scanf("%s", emp.emp_dept);
102     printf("\nSalary : ");
103     scanf("%d", &emp.emp_salary);
104     printf("\nAge : ");
105     scanf("%d", &emp.emp_age);
106
107     fp = fopen("emp.dat", "a");
108     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);
109     fclose(fp);
110 }
111
112 void fnSearchEmpID(int id)
113 {
114     int iCount = 0;
115     employee_t ep;
116     FILE *fp;
117
118     fp = fopen("emp.dat", "r");
119     if(fp==NULL)
120     {
121         printf("\nFile does not exist\n");
122         return;
123     }
124     printf("\nID\tName\tDept\tSalary\tAge\n");
125     while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
        emp_salary, &ep.emp_age)!=EOF)
126     {
127         if(ep.emp_id == id)
128         {
129             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);
130             iCount++;
131         }
132     }
133     if(0 == iCount)
134         printf("\nNo Records found\n");
135     fclose(fp);
136 }
137
138 void fnSearchEmpSal(int sal)
139 {
140     int iCount = 0;
141     employee_t ep;
142     FILE *fp;
143
144     fp = fopen("emp.dat", "r");
145     if(fp==NULL)
146     {
147         printf("\nFile does not exist\n");
```

```c
148          return;
149      }
150      printf("\nID\tName\tDept\tSalary\tAge\n");
151      while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
         emp_salary, &ep.emp_age)!=EOF)
152      {
153          if(ep.emp_salary == sal)
154          {
155              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);
156              iCount++;
157          }
158      }
159      if(0 == iCount)
160          printf("\nNo Records found\n");
161      fclose(fp);
162  }
163
164  void fnSearchEmpDept(char dept[])
165  {
166      int iCount = 0;
167      employee_t ep;
168      FILE *fp;
169
170
171      fp = fopen("emp.dat", "r");
172      if(fp==NULL)
173      {
174          printf("\nFile does not exist\n");
175          return;
176      }
177      printf("\nID\tName\tDept\tSalary\tAge\n");
178      while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
         emp_salary, &ep.emp_age)!=EOF)
179      {
180          if(!strcmp(ep.emp_dept, dept))
181          {
182              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);
183              iCount++;
184          }
185      }
186      if(0 == iCount)
187          printf("\nNo Records found\n");
188  }
189
190  void fnSearchEmpAge(int age)
191  {
192      int iCount = 0;
193      employee_t ep;
194      FILE *fp;
195
196      fp = fopen("emp.dat", "r");
197      if(fp==NULL)
198      {
199          printf("\nFile does not exist\n");
200          return;
201      }
202      printf("\nID\tName\tDept\tSalary\tAge\n");
203      while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
         emp_salary, &ep.emp_age)!=EOF)
204      {
```

```c
205         if(ep.emp_age == age)
206         {
207             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);
208             iCount++;
209         }
210     }
211     if(0 == iCount)
212         printf("\nNo Records found\n");
213 }
```

<div align="center">Listing 1.1: 01EmployeeDB.c</div>

1.2 C Code - Binary I/O

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

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  typedef struct{
6      unsigned emp_id;
7      char emp_name[25];
8      char emp_dept[25];
9      unsigned emp_salary, emp_age;
10 }employee_t;
11
12 void fnAddRecord(void);
13 void fnSearchEmpID(int);
14 void fnSearchEmpSal(int);
15 void fnSearchEmpDept(char[]);
16 void fnSearchEmpAge(int);
17 void fnDisplayAllRecords(void);
18
19 int main()
20 {
21     int id, sal, age, iChoice;
22     char dept[10];
23     printf("%lu bytes\n",sizeof(employee_t));
24     for(;;)
25     {
26         printf("\n1.Add Record\n2.Display Records\n3.Search Employee by ID\n");
27         printf("4.Search Employee by Dept\n5.Search Employee by salary\n");
28         printf("6.Search Employee by Age\n7.Exit");
29         printf("\nEnter your choice : ");
30         scanf("%d",&iChoice);
31
32         switch(iChoice)
33         {
34             case 1: fnAddRecord();
35                     break;
36
37             case 2: printf("\n Employee Details \n");
38                     fnDisplayAllRecords();
39                     break;
40
41             case 3: printf("\nEnter the emp_id that you want to search\n");
42                     scanf("%d",&id);
43                     fnSearchEmpID(id);
44                     break;
45
```

```c
46              case 4: printf("\nEnter the dept that you want to search\n");
47                      scanf("%s",dept);
48                      fnSearchEmpDept(dept);
49                      break;
50
51              case 5: printf("\nEnter the salary that you want to search\n");
52                      scanf("%d",&sal);
53                      fnSearchEmpSal(sal);
54                      break;
55
56              case 6: printf("\nEnter the age that you want to search\n");
57                      scanf("%d",&age);
58                      fnSearchEmpAge(age);
59                      break;
60
61              case 7: exit(0);
62          }
63      }
64      return 0;
65 }
66
67 void fnDisplayAllRecords()
68 {
69      int iCount = 0;
70      employee_t rEmp;
71      FILE *fp;
72
73      fp = fopen("bemp.dat", "rb");
74      if(fp==NULL)
75      {
76          printf("\nFile does not exist\n");
77          return;
78      }
79
80      while(fread(&rEmp, sizeof(employee_t),1,fp))
81      {
82          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);
83          iCount++;
84          if(feof(fp))
85              break;
86      }
87
88      if(0 == iCount)
89          printf("\nNo Records found\n");
90      fclose(fp);
91 }
92
93 void fnAddRecord()
94 {
95      FILE *fp;
96      employee_t wEmp;
97
98      printf("\nEnter Employee details\n");
99      printf("\nID : ");
100     scanf("%d",&wEmp.emp_id);           getchar();
101     printf("\nName : ");
102     scanf("%s", wEmp.emp_name);
103     //fgets(wEmp.emp_name, 25, stdin);
104     printf("\nDept : ");
105     scanf("%s", wEmp.emp_dept);
106     //fgets(wEmp.emp_dept, 25, stdin);
```

```c
107        printf("\nSalary : ");
108        scanf("%d",&wEmp.emp_salary);
109        printf("\nAge : ");
110        scanf("%d",&wEmp.emp_age);
111
112        fp = fopen("bemp.dat", "ab");
113
114        fwrite(&wEmp, sizeof(employee_t),1,fp);
115        //write(fp,&wEmp,sizeof(employee_t));
116
117        fclose(fp);
118 }
119
120 void fnSearchEmpID(int id)
121 {
122        int iCount = 0;
123        employee_t sEmp;
124        FILE *fp;
125
126        fp = fopen("bemp.dat", "r");
127        if(fp==NULL)
128        {
129            printf("\nFile does not exist\n");
130            return;
131        }
132        while(fread(&sEmp, sizeof(employee_t),1,fp))
133        {
134            if(sEmp.emp_id == id)
135            {
136                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);
137                iCount++;
138            }
139            if(feof(fp))
140                break;
141        }
142
143        if(0 == iCount)
144            printf("\nNo Records found\n");
145        fclose(fp);
146 }
147
148 void fnSearchEmpSal(int sal)
149 {
150        int iCount = 0;
151        employee_t sEmp;
152        FILE *fp;
153
154        fp = fopen("bemp.dat", "r");
155        if(fp==NULL)
156        {
157            printf("\nFile does not exist\n");
158            return;
159        }
160        while(fread(&sEmp, sizeof(employee_t),1,fp))
161        {
162            if(sEmp.emp_salary == sal)
163            {
164                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);
165                iCount++;
166            }
```

```c
167      }
168      if(0 == iCount)
169          printf("\nNo Records found\n");
170      fclose(fp);
171  }
172
173  void fnSearchEmpDept(char dept[])
174  {
175      int iCount = 0;
176      employee_t sEmp;
177      FILE *fp;
178
179
180      fp = fopen("bemp.dat", "r");
181      if(fp==NULL)
182      {
183          printf("\nFile does not exist\n");
184          return;
185      }
186      while(fread(&sEmp, sizeof(employee_t),1,fp))
187      {
188          if(!strcmp(sEmp.emp_dept, dept))
189          {
190              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);
191              iCount++;
192          }
193      }
194      if(0 == iCount)
195          printf("\nNo Records found\n");
196  }
197
198  void fnSearchEmpAge(int age)
199  {
200      int iCount = 0;
201      employee_t sEmp;
202      FILE *fp;
203
204      fp = fopen("bemp.dat", "r");
205      if(fp==NULL)
206      {
207          printf("\nFile does not exist\n");
208          return;
209      }
210      while(fread(&sEmp, sizeof(employee_t),1,fp))
211      {
212          if(sEmp.emp_age == age)
213          {
214              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);
215              iCount++;
216          }
217      }
218      if(0 == iCount)
219          printf("\nNo Records found\n");
220  }
```

Listing 1.2: 01EmployeeDBBinary.c

Output

```
================================
1 /**************************************
2 putta:Programs$ gcc 01EmployeeDB.c
3 putta:Programs$ ./a.out
4
5 1.Add Record
6 2.Display Records
7 3.Search Employee by ID
8 4.Search Employee by Dept
9 5.Search Employee by salary
10 6.Search Employee by Age
11 7.Exit
12 Enter your choice : 1
13
14 Enter Employee details
15
16 ID : 123
17 Name : Raju
18 Dept : CSE
19 Salary : 24000
20 Age : 26
21
22 1.Add Record
23 2.Display Records
24 3.Search Employee by ID
25 4.Search Employee by Dept
26 5.Search Employee by salary
27 6.Search Employee by Age
28 7.Exit
29 Enter your choice : 2
30
31  Employee Details
32
33 ID  Name    Dept    Salary  Age
34 123 Raju    CSE 24000    26
35
36 1.Add Record
37 2.Display Records
38 3.Search Employee by ID
39 4.Search Employee by Dept
40 5.Search Employee by salary
41 6.Search Employee by Age
42 7.Exit
43 Enter your choice : 1
44
45 Enter Employee details
46
47 ID : 124
48 Name : Susy
49 Dept : ISE
50 Salary : 26000
51 Age : 25
52
53 1.Add Record
54 2.Display Records
55 3.Search Employee by ID
56 4.Search Employee by Dept
57 5.Search Employee by salary
58 6.Search Employee by Age
```

```
59  7.Exit
60  Enter your choice : 1
61
62  Enter Employee details
63
64  ID : 125
65  Name : John
66  Dept : CSE
67  Salary : 27000
68  Age : 29
69
70  1.Add Record
71  2.Display Records
72  3.Search Employee by ID
73  4.Search Employee by Dept
74  5.Search Employee by salary
75  6.Search Employee by Age
76  7.Exit
77  Enter your choice : 2
78
79   Employee Details
80
81  ID  Name     Dept     Salary  Age
82  123 Raju     CSE      24000   26
83  124 Susy     ISE      26000   25
84  125 John     CSE      27000   29
85
86  1.Add Record
87  2.Display Records
88  3.Search Employee by ID
89  4.Search Employee by Dept
90  5.Search Employee by salary
91  6.Search Employee by Age
92  7.Exit
93  Enter your choice : 3
94
95  Enter the emp_id that you want to search
96  127
97
98  ID  Name     Dept     Salary  Age
99
100 No Records found
101
102 1.Add Record
103 2.Display Records
104 3.Search Employee by ID
105 4.Search Employee by Dept
106 5.Search Employee by salary
107 6.Search Employee by Age
108 7.Exit
109 Enter your choice : 3
110
111 Enter the emp_id that you want to search
112 125
113
114 ID  Name     Dept     Salary  Age
115 125 John     CSE      27000   29
116
117 1.Add Record
118 2.Display Records
119 3.Search Employee by ID
120 4.Search Employee by Dept
```

```
121  5.Search Employee by salary
122  6.Search Employee by Age
123  7.Exit
124  Enter your choice : 4
125
126  Enter the dept that you want to search
127  CSE
128
129  ID  Name     Dept     Salary  Age
130  123 Raju     CSE      24000   26
131  125 John     CSE      27000   29
132
133  1.Add Record
134  2.Display Records
135  3.Search Employee by ID
136  4.Search Employee by Dept
137  5.Search Employee by salary
138  6.Search Employee by Age
139  7.Exit
140  Enter your choice : 5
141
142  Enter the salary that you want to search
143  27000
144
145  ID  Name     Dept     Salary  Age
146  125 John     CSE      27000   29
147
148  1.Add Record
149  2.Display Records
150  3.Search Employee by ID
151  4.Search Employee by Dept
152  5.Search Employee by salary
153  6.Search Employee by Age
154  7.Exit
155  Enter your choice : 6
156
157  Enter the age that you want to search
158  26
159
160  ID  Name     Dept     Salary  Age
161  123 Raju     CSE      24000   26
162
163  1.Add Record
164  2.Display Records
165  3.Search Employee by ID
166  4.Search Employee by Dept
167  5.Search Employee by salary
168  6.Search Employee by Age
169  7.Exit
170  Enter your choice : 7
171  ************************************/
```

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

```
=================================
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  #define MAX 4
6
7  bool fnStkFull(int);
8  bool fnStkEmpty(int);
9  void fnPush(int [], int*);
10 int fnPop(int [], int*);
11 void fnDisplay(int[], int);
12 int fnPeek(int [], int);
13
14 int main()
15 {
16     int stkArray[MAX];
17     int top = -1;
18     int iElem, iChoice;
19
20     for(;;)
21     {
22         printf("\nSTACK OPERATIONS\n");
23         printf("====================");
24         printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.PEEK\n5.EXIT\n");
25         printf("Enter your choice\n");
26         scanf("%d",&iChoice);
27         switch(iChoice)
28         {
29             case 1: fnPush(stkArray, &top);
30                     break;
31
32             case 2: iElem = fnPop(stkArray, &top);
33                     if(iElem != -1)
34                         printf("\nPopped Element is %d\n", iElem);
35                     break;
36
37             case 3: fnDisplay(stkArray, top);
38                     break;
```

```
39
40             case 4: if(!fnStkEmpty(top))
41                     {
42                         iElem = fnPeek(stkArray, top);
43                         printf("\nElement at the  top of the stack: %d\n", iElem);
44                     }
45                     else
46                         printf("\nEmpty Stack\n");
47                     break;
48
49             case 5: exit(1);
50
51             default: printf("\nWrong choice\n");
52         }
53     }
54     return 0;
55 }
56
57 bool fnStkFull(int t)
58 {
59     return ((t == MAX-1) ? true : false);
60 }
61
62 bool fnStkEmpty(int t)
63 {
64     return ((t == -1) ? true : false);
65 }
66
67 void fnPush(int stk[], int *t)
68 {
69     int iElem;
70     if(fnStkFull(*t))
71     {
72         printf("\nStack Overflow\n");
73         return;
74     }
75     printf("\nEnter element to be pushed onto the stack\n");
76     scanf("%d", &iElem);
77
78     *t = *t + 1;
79     stk[*t] = iElem;
80 }
81
82 int fnPop(int stk[], int *t)
83 {
84     int iElem;
85     if(fnStkEmpty(*t))
86     {
87         printf("\nStack Underflow\n");
88         return -1;
89     }
90     iElem = stk[*t];
91     *t = *t - 1;
92
93     return iElem;
94 }
95
96 void fnDisplay(int stk[], int t)
97 {
98     int i;
99     if(fnStkEmpty(t))
100     {
```

```c
101          printf("\nStack Empty\n");
102          return;
103      }
104      printf("\nStack Contents are: \n");
105      for(i = t ; i > -1; --i)
106      {
107          printf("\t%d\n", stk[i]);
108      }
109  }
110
111  int fnPeek(int stk[], int t)
112  {
113      return stk[t];
114  }
```

<div align="center">Listing 2.1: 02Stack.c</div>

2.2   C Code - Structure Representation

```c
     ===============================
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  #define STK_SIZE 5
6
7  typedef struct{
8      int stkArray[STK_SIZE];
9      int top;
10 }STACK_TYPE;
11
12 bool fnStkFull(STACK_TYPE);
13 bool fnStkEmpty(STACK_TYPE);
14 void fnPush(STACK_TYPE*, int);
15 int fnPop(STACK_TYPE*);
16 void fnDisplay(STACK_TYPE);
17 int fnPeek(STACK_TYPE);
18
19 int main()
20 {
21     STACK_TYPE myStack;
22     myStack.top = -1;
23     int iElem, iChoice;
24
25     for(;;)
26     {
27         /*Code to display Menu and accept response */
28         printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.PEEK\n5.EXIT\n");
29         printf("\nEnter your Choice : ");
30         scanf("%d", &iChoice);
31         switch(iChoice)
32         {
33             case 1: if(!fnStkFull(myStack))
34                     {
35                         printf("\nEnter element to be pushed\n");
36                         scanf("%d", &iElem);
37
38                         fnPush(&myStack, iElem);
39                     }
40                     else
41                     {
```

```
42                        printf("\nStack Overflow\n");
43                    }
44                break;
45
46            case 2: if(!fnStkEmpty(myStack))
47                    {
48                        iElem = fnPop(&myStack);
49                        printf("\nPopped element is %d\n", iElem);
50                    }
51                    else
52                    {
53                        printf("\nStack Underflow\n");
54                    }
55                break;
56
57            case 3: if(!fnStkEmpty(myStack))
58                    {
59                        fnDisplay(myStack);
60                    }
61                    else
62                    {
63                        printf("\nStack is Empty\n");
64                    }
65                break;
66            case 4: if(!fnStkEmpty(myStack))
67                    {
68                        iElem = fnPeek(myStack);
69                        printf("\nElement at the top of the stack: %d\n", iElem);
70                    }
71                    else
72                    {
73                        printf("\nStack is Empty\n");
74                    }
75                break;
76
77            case 5: exit(0);
78
79        }
80    }
81    return 0;
82 }
83
84 bool fnStkFull(STACK_TYPE stk)
85 {
86    if(stk.top == STK_SIZE-1)
87        return true;
88    else
89        return false;
90 }
91
92 bool fnStkEmpty(STACK_TYPE stk)
93 {
94    if(stk.top == -1)
95        return true;
96    else
97        return false;
98 }
99
100 void fnPush(STACK_TYPE *stk, int elem)
101 {
102    stk->top = stk->top + 1;
103    stk->stkArray[stk->top] = elem;
```

```c
104
105 }
106
107 int fnPop(STACK_TYPE *stk)
108 {
109     int elem;
110     elem = stk->stkArray[stk->top];
111     stk->top = stk->top - 1;
112     return elem;
113 }
114
115 void fnDisplay(STACK_TYPE stk)
116 {
117     int i;
118     printf("\nContents of the stack :\n");
119     for(i = stk.top; i>=0; i--)
120     {
121         printf("%d\n", stk.stkArray[i]);
122     }
123     printf("\n");
124 }
125
126 int fnPeek(STACK_TYPE stk)
127 {
128     return stk.stkArray[stk.top];
129 }
```

Listing 2.2: 02StructStack.c

Output

```
==================================
1 /************************************
2 putta:Programs$ gcc 02Stack.c
3 putta:Programs$ ./a.out
4
5 STACK OPERATIONS
6 ==================
7 1.PUSH
8 2.POP
9 3.DISPLAY
10 4.PEEK
11 5.EXIT
12 Enter your choice
13 2
14
15 Stack Underflow
16
17 STACK OPERATIONS
18 ==================
19 1.PUSH
20 2.POP
21 3.DISPLAY
22 4.PEEK
23 5.EXIT
24 Enter your choice
25 3
26
27 Stack Empty
28
29 STACK OPERATIONS
```

```
30  ====================
31  1.PUSH
32  2.POP
33  3.DISPLAY
34  4.PEEK
35  5.EXIT
36  Enter your choice
37  1
38
39  Enter element to be pushed onto the stack
40  1
41
42  STACK OPERATIONS
43  ====================
44  1.PUSH
45  2.POP
46  3.DISPLAY
47  4.PEEK
48  5.EXIT
49  Enter your choice
50  1
51
52  Enter element to be pushed onto the stack
53  2
54
55  STACK OPERATIONS
56  ====================
57  1.PUSH
58  2.POP
59  3.DISPLAY
60  4.PEEK
61  5.EXIT
62  Enter your choice
63  1
64
65  Enter element to be pushed onto the stack
66  3
67
68  STACK OPERATIONS
69  ====================
70  1.PUSH
71  2.POP
72  3.DISPLAY
73  4.PEEK
74  5.EXIT
75  Enter your choice
76  3
77
78  Stack Contents are:
79      3
80      2
81      1
82
83  STACK OPERATIONS
84  ====================
85  1.PUSH
86  2.POP
87  3.DISPLAY
88  4.PEEK
89  5.EXIT
90  Enter your choice
91  4
```

```
 92
 93  Element at the  top of the stack is 3
 94
 95  STACK OPERATIONS
 96  ===================
 97  1.PUSH
 98  2.POP
 99  3.DISPLAY
100  4.PEEK
101  5.EXIT
102  Enter your choice
103  2
104
105  Popped Element is 3
106
107  STACK OPERATIONS
108  ===================
109  1.PUSH
110  2.POP
111  3.DISPLAY
112  4.PEEK
113  5.EXIT
114  Enter your choice
115  4
116
117  Element at the  top of the stack is 2
118
119  STACK OPERATIONS
120  ===================
121  1.PUSH
122  2.POP
123  3.DISPLAY
124  4.PEEK
125  5.EXIT
126  Enter your choice
127  1
128
129  Enter element to be pushed onto the stack
130  3
131
132  STACK OPERATIONS
133  ===================
134  1.PUSH
135  2.POP
136  3.DISPLAY
137  4.PEEK
138  5.EXIT
139  Enter your choice
140  1
141
142  Enter element to be pushed onto the stack
143  4
144
145  STACK OPERATIONS
146  ===================
147  1.PUSH
148  2.POP
149  3.DISPLAY
150  4.PEEK
151  5.EXIT
152  Enter your choice
153  3
```

```
154
155 Stack Contents are:
156      4
157      3
158      2
159      1
160
161 STACK OPERATIONS
162 ====================
163 1.PUSH
164 2.POP
165 3.DISPLAY
166 4.PEEK
167 5.EXIT
168 Enter your choice
169 1
170
171 Stack Overflow
172
173 STACK OPERATIONS
174 ====================
175 1.PUSH
176 2.POP
177 3.DISPLAY
178 4.PEEK
179 5.EXIT
180 Enter your choice
181 5
182 *************************************/
```

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>
#include <ctype.h>
#define STK_SIZE 10

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

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

```c
40            while(fnPrecd(acStack[top]) >= fnPrecd(cSymb))
41            {
42                if(cSymb == '^' && acStack[top] == '^')
43                    break;
44                acPost[j++] = fnPop(acStack, &top);
45            }
46            fnPush(acStack, &top, cSymb);
47        }
48    }
49    while(acStack[top] != '#')
50    {
51        acPost[j++] = fnPop(acStack, &top);
52    }
53    acPost[j] = '\0';
54
55    printf("\nInfix Expression is %s\n", acExpr);
56    printf("\nPostfix Expression is %s\n", acPost);
57    return 0;
58 }
59
60 void fnPush(char Stack[], int *t , char elem)
61 {
62    *t = *t + 1;
63    Stack[*t] = elem;
64 }
65
66 char fnPop(char Stack[], int *t)
67 {
68    char elem;
69    elem = Stack[*t];
70    *t = *t - 1;
71    return elem;
72 }
73
74 int fnPrecd(char cSymb)
75 {
76    int iPrecd;
77    switch(cSymb)
78    {
79        case '#' :  iPrecd =  -1;
80        case '(' :  iPrecd =  0;
81        case '+' :
82        case '-' :  iPrecd =  1;
83        case '*' :
84        case '/' :  iPrecd =  2;
85        case '^' :  iPrecd =  3;
86    }
87    return iPrecd;
88 }
```

Listing 3.1: 03ConvInfix.c

Output

```
1 /**************************************
2 putta:Programs$ gcc 03ConvInfix.c
3 putta:Programs$ ./a.out
4 Enter a valid infix expression
5 (a^b^c)+(d/(e-f))
6
7 Infix Expression is (a^b^c)+(d/(e-f))
8 Postfix Expression is abc^^def-/+
```

```
 9
10  putta:Programs$ ./a.out
11  Enter a valid infix expression
12  (a*(b-c)/(a+b*c))
13
14  Infix Expression is (a*(b-c)/(a+b*c))
15  Postfix Expression is abc-*abc*+/
16
17  *************************************/
```

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>
#include <ctype.h>

#define STK_SIZE 10

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

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

```
40              fnPush(iaStack, &top, iRes);
41          }
42
43      }
44      iRes = fnPop(iaStack, &top);
45      printf("\nValue of %s expression is %d\n", acExpr, iRes);
46      return 0;
47  }
48
49  void fnPush(int Stack[], int *t , int elem)
50  {
51      *t = *t + 1;
52      Stack[*t] = elem;
53
54  }
55
56  int fnPop(int Stack[], int *t)
57  {
58      int elem;
59      elem = Stack[*t];
60      *t = *t - 1;
61      return elem;
62  }
```

Listing 4.1: 04EvalPrefix.c

Output

```
=================================
1  /***************************************
2  putta:Programs$ gcc 04EvalPrefix.c -Wall
3  putta:Programs$ ./a.out
4
5  Enter a valid prefix expression
6  ++34*23
7
8  Value of ++34*23 expression is 13
9
10 putta:Programs$ ./a.out
11
12 Enter a valid prefix expression
13 *+-525*25
14
15 Value of *+-525*25 expression is 80
16
17 ***************************************/
```

Listing 4.2: out4.c

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

```
=================================
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.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, 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");
                    }
```

```
38                       break;
39             case 2: if(!fnQueueEmpty(myQueue, iFront, iRear))
40                     {
41                         iElem = fnDeleteFront(myQueue, &iFront, &iRear);
42                         printf("\nDeleted element is %d\n", iElem);
43                     }
44                     else
45                     {
46                         printf("\nQueue is Empty\n");
47                     }
48
49                     break;
50             case 3: if(!fnQueueEmpty(myQueue, iFront, iRear))
51                     {
52                         printf("\nContents of the Queue is \n");
53                         fnDisplay(myQueue, iFront, iRear);
54                     }
55                     else
56                     {
57                         printf("\nQueue is Empty\n");
58                     }
59
60                     break;
61
62             case 4: exit(0);
63
64             default: printf("\nInvalid choice\n");s
65         }
66     }
67     return 0;
68 }
69
70 bool fnQueueFull(int queue[], int r)
71 {
72     if(r == QUEUE_SIZE-1)
73         return true;
74     else
75         return false;
76 }
77
78 bool fnQueueEmpty(int queue[], int f, int r)
79 {
80     if(r == f-1)
81         return true;
82     else
83         return false;
84 }
85
86 void fnInsertRear(int queue[], int *r, int iVal)
87 {
88     *r = *r + 1;
89     queue[*r] = iVal;
90 }
91
92 int fnDeleteFront(int queue[], int *f, int *r)
93 {
94     int iElem;
95     iElem = queue[*f];
96
97     if(*f == *r)
98     {
99         *f = 0;      *r = -1;
```

```c
100        }
101        else
102        {
103            *f = *f + 1;
104        }
105        return iElem;
106 }
107
108 void fnDisplay(int queue[], int f, int r)
109 {
110     int i;
111     for(i=f; i<=r; i++)
112     {
113         printf("%d\t", queue[i]);
114     }
115     printf("\n");
116 }
```

Listing 5.1: 05LinearQueue.c

5.2 C Code - Structure Representation

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

```c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 #define QUEUE_SIZE 5
6
7 typedef struct
8 {
9     int Queue[QUEUE_SIZE];
10    int iFront, iRear;
11 }QUEUE_T;
12
13 void fnInsertRear(QUEUE_T*, int);
14 int fnDeleteFront(QUEUE_T*);
15 void fnDisplay(QUEUE_T);
16 bool fnQueueFull(QUEUE_T);
17 bool fnQueueEmpty(QUEUE_T);
18
19 int main()
20 {
21     QUEUE_T myQueue;
22     int iElem, iChoice;
23     myQueue.iFront = 0;
24     myQueue.iRear = -1;
25
26     for(;;)
27     {
28         printf("\nQueue Operations\n");
29         printf("====================");
30         printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
31         printf("Enter your choice\n");
32         scanf("%d",&iChoice);
33         switch(iChoice)
34         {
35             case 1: if(!fnQueueFull(myQueue))
36                     {
37                         printf("\nEnter an element : ");
38                         scanf("%d", &iElem);
```

```
39                         fnInsertRear(&myQueue, iElem);
40                     }
41                     else
42                     {
43                         printf("\nQueue is Full\n");
44                     }
45                     break;
46
47             case 2: if(!fnQueueEmpty(myQueue))
48                     {
49                         iElem = fnDeleteFront(&myQueue);
50                         printf("\nDeleted element is %d\n", iElem);
51                     }
52                     else
53                     {
54                         printf("\nQueue is Empty\n");
55                     }
56                     break;
57
58             case 3: if(!fnQueueEmpty(myQueue))
59                     {
60                         printf("\nContents of the Queue is \n");
61                         fnDisplay(myQueue);
62                     }
63                     else
64                     {
65                         printf("\nQueue is Empty\n");
66                     }
67                     break;
68
69             case 4: exit(0);
70
71             default: printf("\nInvalid choice\n");
72                     break;
73         }
74     }
75     return 0;
76 }
77
78 bool fnQueueFull(QUEUE_T myQ)
79 {
80     if(myQ.iRear == QUEUE_SIZE-1)
81         return true;
82     else
83         return false;
84 }
85
86 bool fnQueueEmpty(QUEUE_T myQ)
87 {
88     if(myQ.iRear == myQ.iFront-1)
89         return true;
90     else
91         return false;
92 }
93
94 void fnInsertRear(QUEUE_T *myQ, int iVal)
95 {
96     (myQ->iRear)++;
97     myQ->Queue[myQ->iRear] = iVal;
98 }
99
100 int fnDeleteFront(QUEUE_T *myQ)
```

```c
101 {
102     int iElem;
103     iElem = myQ->Queue[myQ->iFront];
104
105     if(myQ->iFront == myQ->iRear)
106     {
107         myQ->iFront = 0;
108         myQ->iRear = -1;
109     }
110     else
111     {
112         myQ->iFront = myQ->iFront + 1;
113     }
114     return iElem;
115 }
116
117 void fnDisplay(QUEUE_T myQ)
118 {
119     int i;
120     for(i=myQ.iFront; i<=myQ.iRear; i++)
121     {
122         printf("%d\t", myQ.Queue[i]);
123     }
124     printf("\n");
125 }
```

Listing 5.2: 05StructLinearQueue.c

Output

```
==================================
1 /***************************
2 putta:Programs$ gcc 05LinearQueue.c -Wall
3 putta:Programs$ ./a.out
4
5 Queue Operations
6 ===================
7 1.Qinsert
8 2.Qdelete
9 3.Qdisplay
10 4.Exit
11 Enter your choice
12 2
13
14 Queue is Empty
15
16 Queue Operations
17 ===================
18 1.Qinsert
19 2.Qdelete
20 3.Qdisplay
21 4.Exit
22 Enter your choice
23 3
24
25 Queue is Empty
26
27 Queue Operations
28 ===================
29 1.Qinsert
30 2.Qdelete
```

```
31  3.Qdisplay
32  4.Exit
33  Enter your choice
34  1
35
36  Enter an element : 1
37
38  Queue Operations
39  =====================
40  1.Qinsert
41  2.Qdelete
42  3.Qdisplay
43  4.Exit
44  Enter your choice
45  1
46
47  Enter an element : 2
48
49  Queue Operations
50  =====================
51  1.Qinsert
52  2.Qdelete
53  3.Qdisplay
54  4.Exit
55  Enter your choice
56  1
57
58  Enter an element : 3
59
60  Queue Operations
61  =====================
62  1.Qinsert
63  2.Qdelete
64  3.Qdisplay
65  4.Exit
66  Enter your choice
67  3
68
69  Contents of the Queue is
70  1    2    3
71
72  Queue Operations
73  =====================
74  1.Qinsert
75  2.Qdelete
76  3.Qdisplay
77  4.Exit
78  Enter your choice
79  1
80
81  Enter an element : 4
82
83  Queue Operations
84  =====================
85  1.Qinsert
86  2.Qdelete
87  3.Qdisplay
88  4.Exit
89  Enter your choice
90  1
91
92  Enter an element : 5
```

```
 93
 94  Queue Operations
 95  ====================
 96  1.Qinsert
 97  2.Qdelete
 98  3.Qdisplay
 99  4.Exit
100  Enter your choice
101  3
102
103  Contents of the Queue is
104  1    2    3    4    5
105
106  Queue Operations
107  ====================
108  1.Qinsert
109  2.Qdelete
110  3.Qdisplay
111  4.Exit
112  Enter your choice
113  1
114
115  Queue is Full
116
117  Queue Operations
118  ====================
119  1.Qinsert
120  2.Qdelete
121  3.Qdisplay
122  4.Exit
123  Enter your choice
124  2
125
126  Deleted element is 1
127
128  Queue Operations
129  ====================
130  1.Qinsert
131  2.Qdelete
132  3.Qdisplay
133  4.Exit
134  Enter your choice
135  2
136
137  Deleted element is 2
138
139  Queue Operations
140  ====================
141  1.Qinsert
142  2.Qdelete
143  3.Qdisplay
144  4.Exit
145  Enter your choice
146  2
147
148  Deleted element is 3
149
150  Queue Operations
151  ====================
152  1.Qinsert
153  2.Qdelete
154  3.Qdisplay
```

```
155  4.Exit
156  Enter your choice
157  4
158  **********
159
160  *************************************/
```

Listing 5.3: out5.c

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>
#include <stdbool.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");
                    }
```

```c
38
39                     break;
40             case 2: if(!fnQueueEmpty(myQueue, iFront, iRear))
41                     {
42                             iElem = fnDeleteFront(myQueue, &iFront, &iRear);
43                             printf("\nDeleted element is %d\n", iElem);
44                     }
45                     else
46                     {
47                             printf("\nQueue is Empty\n");
48                     }
49
50                     break;
51             case 3: if(!fnQueueEmpty(myQueue, iFront, iRear))
52                     {
53                             printf("\nContents of the Queue is \n");
54                             fnDisplay(myQueue, iFront, iRear);
55                     }
56                     else
57                     {
58                             printf("\nQueue is Empty\n");
59                     }
60
61                     break;
62
63             case 4: exit(0);
64
65             default: printf("\nInvalid choice\n");
66
67                     break;
68         }
69     }
70     return 0;
71 }
72
73 bool fnQueueFull(int queue[], int f, int r)
74 {
75     if((r+1) % QUEUE_SIZE == f)
76         return true;
77     else
78         return false;
79 }
80
81 bool fnQueueEmpty(int queue[], int f, int r)
82 {
83     if(f == -1)
84         return true;
85     else
86         return false;
87 }
88
89 void fnInsertRear(int queue[], int *f, int *r, int iVal)
90 {
91     if(*r == -1)
92     {
93         *f = *f + 1;
94         *r = *r + 1;
95     }
96     else
97         *r = (*r + 1)%QUEUE_SIZE;
98
99     queue[*r] = iVal;
```

```c
100  }
101
102  int fnDeleteFront(int queue[], int *f, int *r)
103  {
104      int iElem;
105      iElem = queue[*f];
106
107      if(*f == *r)
108      {
109          *f = -1;
110          *r = -1;
111      }
112      else
113      {
114          *f = (*f + 1)%QUEUE_SIZE;
115      }
116      return iElem;
117  }
118
119  void fnDisplay(int queue[], int f, int r)
120  {
121      int i;
122      if(f<=r)
123      {
124          for(i=f; i<=r; i++)
125          {
126              printf("%d\t", queue[i]);
127          }
128          printf("\n");
129      }
130      else
131      {
132          for(i=f; i<=QUEUE_SIZE-1; i++)
133          {
134              printf("%d\t", queue[i]);
135          }
136          for(i=0; i<=r; i++)
137          {
138              printf("%d\t", queue[i]);
139          }
140          printf("\n");
141      }
142  }
```

Listing 6.1: 06CircQueue.c

6.2 C Code - Structure Representation

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

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5
6  #define QUEUE_SIZE 5
7  #define NAME_LENGTH 30
8
9  typedef struct
10  {
11      int Queue[QUEUE_SIZE];
12      int iFront, iRear;
```

```c
13  }QUEUE_T;
14
15
16  void fnInsertRear(QUEUE_T*, int);
17  int fnDeleteFront(QUEUE_T*);
18  void fnDisplay(QUEUE_T);
19  bool fnQueueFull(QUEUE_T);
20  bool fnQueueEmpty(QUEUE_T);
21
22  int main()
23  {
24      QUEUE_T myQueue;
25      int iElem, iChoice;
26
27      myQueue.iFront = -1;
28      myQueue.iRear = -1;
29
30
31      for(;;)
32      {
33          printf("\nQueue Operations\n");
34          printf("====================");
35          printf("\n1.Qinsert\n2.Qdelete\n3.Qdisplay\n4.Exit\n");
36          printf("Enter your choice\n");
37          scanf("%d",&iChoice);
38          switch(iChoice)
39          {
40              case 1: if(!fnQueueFull(myQueue))
41                      {
42                          printf("\nEnter an element : ");
43                          scanf("%d", &iElem);
44                          fnInsertRear(&myQueue, iElem);
45                      }
46                      else
47                      {
48                          printf("\nQueue is Full\n");
49                      }
50
51                  break;
52              case 2: if(!fnQueueEmpty(myQueue))
53                      {
54                          iElem = fnDeleteFront(&myQueue);
55                          printf("\nDeleted element is %d\n", iElem);
56                      }
57                      else
58                      {
59                          printf("\nQueue is Empty\n");
60                      }
61
62                  break;
63              case 3: if(!fnQueueEmpty(myQueue))
64                      {
65                          printf("\nContents of the Queue is \n");
66                          fnDisplay(myQueue);
67                      }
68                      else
69                      {
70                          printf("\nQueue is Empty\n");
71                      }
72
73                  break;
74
```

```
75          case 4: exit(0);
76
77          default: printf("\nInvalid choice\n");
78
79                  break;
80          }
81      }
82      return 0;
83  }
84
85  bool fnQueueFull(QUEUE_T myQ)
86  {
87      if((myQ.iRear+1) % QUEUE_SIZE == myQ.iFront)
88          return true;
89      else
90          return false;
91  }
92
93  bool fnQueueEmpty(QUEUE_T myQ)
94  {
95      if(myQ.iFront == -1)
96          return true;
97      else
98          return false;
99  }
100
101 void fnInsertRear(QUEUE_T *myQ, int iVal)
102 {
103     if(myQ->iRear == -1)
104     {
105         (myQ->iRear)++;
106         (myQ->iFront)++;
107     }
108     else
109         myQ->iRear = (myQ->iRear + 1) % QUEUE_SIZE;
110
111     myQ->Queue[myQ->iRear] = iVal;
112 }
113
114 int fnDeleteFront(QUEUE_T *myQ)
115 {
116     int iElem;
117     iElem = myQ->Queue[myQ->iFront];
118
119     if(myQ->iFront == myQ->iRear)
120     {
121         myQ->iFront = myQ->iRear = -1;
122     }
123     else
124     {
125         myQ->iFront = (myQ->iFront + 1)%QUEUE_SIZE;
126     }
127     return iElem;
128 }
129
130 void fnDisplay(QUEUE_T myQ)
131 {
132     int i;
133     if(myQ.iFront<=myQ.iRear)
134     {
135         for(i=myQ.iFront; i<=myQ.iRear; i++)
136         {
```

```
137              printf("%d\t", myQ.Queue[i]);
138          }
139          printf("\n");
140      }
141      else
142      {
143          for(i=myQ.iFront; i<QUEUE_SIZE; i++)
144          {
145              printf("%d\t", myQ.Queue[i]);
146          }
147          for(i=0; i<=myQ.iRear; i++)
148          {
149              printf("%d\t", myQ.Queue[i]);
150          }
151          printf("\n");
152
153      }
154 }
```

Listing 6.2: 06StructCircularQueue.c

Output

```
==================================
1 /************************************
2 putta:Programs$ gcc 06CircQueue.c -Wall
3 putta:Programs$ ./a.out
4
5 Queue Operations
6 ===================
7 1.Qinsert
8 2.Qdelete
9 3.Qdisplay
10 4.Exit
11 Enter your choice
12 2
13
14 Queue is Empty
15
16 Queue Operations
17 ===================
18 1.Qinsert
19 2.Qdelete
20 3.Qdisplay
21 4.Exit
22 Enter your choice
23 1
24
25 Enter an element : 1
26
27 Queue Operations
28 ===================
29 1.Qinsert
30 2.Qdelete
31 3.Qdisplay
32 4.Exit
33 Enter your choice
34 1
35
36 Enter an element : 2
37
```

```
38  Queue Operations
39  ====================
40  1.Qinsert
41  2.Qdelete
42  3.Qdisplay
43  4.Exit
44  Enter your choice
45  1
46
47  Enter an element : 3
48
49  Queue Operations
50  ====================
51  1.Qinsert
52  2.Qdelete
53  3.Qdisplay
54  4.Exit
55  Enter your choice
56  3
57
58  Contents of the Queue is
59  1    2    3
60
61  Queue Operations
62  ====================
63  1.Qinsert
64  2.Qdelete
65  3.Qdisplay
66  4.Exit
67  Enter your choice
68  1
69
70  Enter an element : 2
71
72  Queue Operations
73  ====================
74  1.Qinsert
75  2.Qdelete
76  3.Qdisplay
77  4.Exit
78  Enter your choice
79  4
80  putta:Programs$ clear
81
82  putta:Programs$ gcc 06CircQueue.c -Wall
83  putta:Programs$ ./a.out
84
85  Queue Operations
86  ====================
87  1.Qinsert
88  2.Qdelete
89  3.Qdisplay
90  4.Exit
91  Enter your choice
92  2
93
94  Queue is Empty
95
96  Queue Operations
97  ====================
98  1.Qinsert
99  2.Qdelete
```

```
100  3.Qdisplay
101  4.Exit
102  Enter your choice
103  1
104
105  Enter an element : 1
106
107  Queue Operations
108  ====================
109  1.Qinsert
110  2.Qdelete
111  3.Qdisplay
112  4.Exit
113  Enter your choice
114  1
115
116  Enter an element : 2
117
118  Queue Operations
119  ====================
120  1.Qinsert
121  2.Qdelete
122  3.Qdisplay
123  4.Exit
124  Enter your choice
125  1
126
127  Enter an element : 3
128
129  Queue Operations
130  ====================
131  1.Qinsert
132  2.Qdelete
133  3.Qdisplay
134  4.Exit
135  Enter your choice
136  3
137
138  Contents of the Queue is
139  1    2    3
140
141  Queue Operations
142  ====================
143  1.Qinsert
144  2.Qdelete
145  3.Qdisplay
146  4.Exit
147  Enter your choice
148  1
149
150  Enter an element : 4
151
152  Queue Operations
153  ====================
154  1.Qinsert
155  2.Qdelete
156  3.Qdisplay
157  4.Exit
158  Enter your choice
159  1
160
161  Enter an element : 5
```

```
162
163 Queue Operations
164 ====================
165 1.Qinsert
166 2.Qdelete
167 3.Qdisplay
168 4.Exit
169 Enter your choice
170 1
171
172 Queue is Full
173
174 Queue Operations
175 ====================
176 1.Qinsert
177 2.Qdelete
178 3.Qdisplay
179 4.Exit
180 Enter your choice
181 3
182
183 Contents of the Queue is
184 1    2    3    4    5
185
186 Queue Operations
187 ====================
188 1.Qinsert
189 2.Qdelete
190 3.Qdisplay
191 4.Exit
192 Enter your choice
193 2
194
195 Deleted element is 1
196
197 Queue Operations
198 ====================
199 1.Qinsert
200 2.Qdelete
201 3.Qdisplay
202 4.Exit
203 Enter your choice
204 3
205
206 Contents of the Queue is
207 2    3    4    5
208
209 Queue Operations
210 ====================
211 1.Qinsert
212 2.Qdelete
213 3.Qdisplay
214 4.Exit
215 Enter your choice
216 2
217
218 Deleted element is 2
219
220 Queue Operations
221 ====================
222 1.Qinsert
223 2.Qdelete
```

```
224  3.Qdisplay
225  4.Exit
226  Enter your choice
227  3
228
229  Contents of the Queue is
230  3    4    5
231
232  Queue Operations
233  ====================
234  1.Qinsert
235  2.Qdelete
236  3.Qdisplay
237  4.Exit
238  Enter your choice
239  1
240
241  Enter an element : 6
242
243  Queue Operations
244  ====================
245  1.Qinsert
246  2.Qdelete
247  3.Qdisplay
248  4.Exit
249  Enter your choice
250  3
251
252  Contents of the Queue is
253  3    4    5    6
254
255  Queue Operations
256  ====================
257  1.Qinsert
258  2.Qdelete
259  3.Qdisplay
260  4.Exit
261  Enter your choice
262  4
263
264  *************************************/
```

Listing 6.3: out6.c

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 fnDeletePosition(int, 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)
        {
```

```c
32            case 1: printf("\nEnter a element\n");
33                    scanf("%d", &iElem);
34                    first = fnInsertFront(iElem, first);
35                    break;
36
37            case 2: first = fnDeleteFront(first);
38                    break;
39
40            case 3: printf("\nEnter a element\n");
41                    scanf("%d", &iElem);
42                    printf("\nEnter the position\n");
43                    scanf("%d", &iPos);
44                    first = fnInsertPosition(iElem, iPos, first);
45                    break;
46
47            case 4: fnDisplay(first);
48                    break;
49
50            case 5: first = fnReverse(first);
51                    break;
52            case 6: exit(0);
53        }
54    }
55    return 0;
56 }
57
58 NODEPTR fnGetNode(void)
59 {
60    NODEPTR newNode;
61    newNode = ( NODEPTR ) malloc (sizeof(struct node));
62    if(newNode == NULL)
63    {
64        printf("\nOut of Memory");
65        exit(0);
66    }
67    return newNode;
68 }
69
70 void fnFreeNode(NODEPTR x)
71 {
72    free(x);
73 }
74
75 NODEPTR fnInsertFront(int elem,NODEPTR first)
76 {
77    NODEPTR temp;
78    temp = fnGetNode();
79    temp->info = elem;
80    temp->link = first;
81    first = temp;
82    return first;
83 }
84
85 NODEPTR fnDeleteFront(NODEPTR first)
86 {
87    NODEPTR temp;
88    if(first == NULL)
89    {
90        printf("\nList is Empty cannot delete\n");
91        return first;
92    }
93
```

```
94      temp = first;
95      printf("\nElement deleted is %d\n", temp->info);
96      fnFreeNode(temp);
97      first = first->link;
98      return first;
99  }
100
101 NODEPTR fnInsertPosition(int elem,int pos,NODEPTR first)
102 {
103     NODEPTR temp,prev,cur;
104     int count;
105
106     temp = fnGetNode();
107     temp->info = elem;
108     temp->link = NULL;
109
110     if(first == NULL && pos == 1)
111         return temp;
112
113     if(first == NULL)
114     {
115         printf("\nInvalid Position");
116         return first;
117     }
118
119     if(pos == 1)
120     {
121         temp->link = first;
122         return temp;
123     }
124
125     count = 1;
126     prev = NULL;
127     cur = first;
128
129     while(cur != NULL && count != pos)
130     {
131         prev = cur;
132         cur = cur->link;
133         count++;
134     }
135
136     if(count == pos)
137     {
138         prev->link = temp;
139         temp->link = cur;
140         return first;
141     }
142
143     printf("\nInvalid Position");
144     return first;
145 }
146
147 void fnDisplay(NODEPTR first)
148 {
149     NODEPTR temp;
150
151     if(first == NULL)
152     {
153         printf("\nList is Empty\n");
154         return;
155     }
```

```
156
157      printf("\nList Contents\n");
158      printf("=================================\n");
159      for(temp = first; temp != NULL; temp = temp->link)
160          printf("%4d",temp->info);
161      printf("\n=================================\n");
162      printf("\n\n");
163
164  }
165
166  NODEPTR fnReverse(NODEPTR first)
167  {
168      NODEPTR cur, prev, next;
169      if(first == NULL)
170      {
171          printf("\nList is Empty\n");
172          return first;
173      }
174      if(first->link == NULL)
175      {
176          return first;
177      }
178
179      prev = first;
180      cur = first->link;
181      next = cur->link;
182      prev->link = NULL;
183      while(cur->link!=NULL)
184      {
185          cur->link = prev;
186          prev = cur;
187          cur = next;
188          next = next->link;
189      }
190      cur->link = prev;
191      return cur;
192  }
```

Listing 7.1: 07SinglyLinkedList.c

Output

```
=================================
1  /***************************************
2  putta:Programs$ gcc 07SinglyLinkedList.c -Wall
3  putta:Programs$ ./a.out
4
5  1.Insert Front
6  2.Delete Front
7  3.Insert At Position
8  4.Display
9  5.Reverse
10 6.Exit
11 Enter your choice
12 1
13
14 Enter a element
15 4
16
17 1.Insert Front
18 2.Delete Front
19 3.Insert At Position
```

```
20  4.Display
21  5.Reverse
22  6.Exit
23  Enter your choice
24  1
25
26  Enter a element
27  3
28
29  1.Insert Front
30  2.Delete Front
31  3.Insert At Position
32  4.Display
33  5.Reverse
34  6.Exit
35  Enter your choice
36  1
37
38  Enter a element
39  2
40
41  1.Insert Front
42  2.Delete Front
43  3.Insert At Position
44  4.Display
45  5.Reverse
46  6.Exit
47  Enter your choice
48  4
49
50  List Contents
51  ================================
52      2    3    4
53  ================================
54
55  1.Insert Front
56  2.Delete Front
57  3.Insert At Position
58  4.Display
59  5.Reverse
60  6.Exit
61  Enter your choice
62  3
63  Enter a element
64  1
65
66  Enter the position
67  1
68
69  1.Insert Front
70  2.Delete Front
71  3.Insert At Position
72  4.Display
73  5.Reverse
74  6.Exit
75  Enter your choice
76  4
77
78  List Contents
79  ================================
80      1    2    3    4
81  ================================
```

```
82
83  1.Insert Front
84  2.Delete Front
85  3.Insert At Position
86  4.Display
87  5.Reverse
88  6.Exit
89  Enter your choice
90  5
91
92  1.Insert Front
93  2.Delete Front
94  3.Insert At Position
95  4.Display
96  5.Reverse
97  6.Exit
98  Enter your choice
99  4
100
101 List Contents
102 ==================================
103     4   3   2   1
104 ==================================
105
106 1.Insert Front
107 2.Delete Front
108 3.Insert At Position
109 4.Display
110 5.Reverse
111 6.Exit
112 Enter your choice
113 2
114
115 Element deleted is 4
116
117 1.Insert Front
118 2.Delete Front
119 3.Insert At Position
120 4.Display
121 5.Reverse
122 6.Exit
123 Enter your choice
124 4
125
126 List Contents
127 ==================================
128     3   2   1
129 ==================================
130
131 1.Insert Front
132 2.Delete Front
133 3.Insert At Position
134 4.Display
135 5.Reverse
136 6.Exit
137 Enter your choice
138 6
139
140 **************************************/
```

Listing 7.2: out7.c

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");
```

```
32                        scanf("%d", &iElem);
33                        list1 = fnInsertOrder(iElem, list1);
34                        break;
35
36                case 2: printf("\nEnter a element\n");
37                        scanf("%d", &iElem);
38                        list2 = fnInsertOrder(iElem, list2);
39                        break;
40
41                case 3: printf("\nList 1 Contents\n");
42                        fnDisplay(list1);
43                        printf("\nList 2 Contents\n");
44                        fnDisplay(list2);
45                        break;
46
47                case 4: printf("\nList 1 Contents\n");
48                        fnDisplay(list1);
49                        printf("\nList 2 Contents\n");
50                        fnDisplay(list2);
51                        list3 = fnUnion(list1, list2);
52                        printf("\nUnion\n");
53                        fnDisplay(list3);
54                        break;
55
56                case 5: printf("\nList 1 Contents\n");
57                        fnDisplay(list1);
58                        printf("\nList 2 Contents\n");
59                        fnDisplay(list2);
60                        list4 = fnIntersection(list1, list2);
61                        printf("\nIntersection\n");
62                        fnDisplay(list4);
63                        break;
64            case 6: exit(0);
65        }
66    }
67    return 0;
68 }
69
70 NODEPTR fnGetNode(void)
71 {
72    NODEPTR newNode;
73    newNode = ( NODEPTR ) malloc (sizeof(struct node));
74    if(newNode == NULL)
75    {
76        printf("\nOut of Memory");
77        exit(0);
78    }
79    return newNode;
80 }
81
82 NODEPTR fnIntersection(NODEPTR l1,NODEPTR l2)
83 {
84    NODEPTR t1, t2, t3;
85    t1 = l1;
86    while(t1 != NULL)
87    {
88        t2 = l2;
89        while(t2 != NULL)
90        {
91            if(t1->info == t2->info)
92                t3 = fnInsertRear(t1->info, t3);
93            t2 = t2->link;
```

```
 94          }
 95          t1 = t1->link;
 96      }
 97      return t3;
 98 }
 99
100
101 NODEPTR fnUnion(NODEPTR l1,NODEPTR l2)
102 {
103      NODEPTR t1, t2, t3=NULL;
104      t1 = l1;
105      t2 = l2;
106      while(t1 != NULL && t2 != NULL)
107      {
108          if(t1->info < t2->info)
109          {
110              t3 = fnInsertRear(t1->info, t3);
111              t1 = t1->link;
112          }
113          else if(t1->info > t2->info)
114          {
115              t3 = fnInsertRear(t2->info, t3);
116              t2 = t2->link;
117          }
118          else
119          {
120              t2 = t2->link;
121          }
122
123      }
124      while(t1 != NULL)
125      {
126          t3 = fnInsertRear(t1->info, t3);
127          t1 = t1->link;
128      }
129      while(t2 != NULL)
130      {
131          t3 = fnInsertRear(t2->info, t3);
132          t2 = t2->link;
133      }
134      return t3;
135 }
136
137
138 NODEPTR fnInsertOrder(int elem, NODEPTR first)
139 {
140      NODEPTR temp,prev,cur;
141
142      temp = fnGetNode();
143      temp->info = elem;
144      temp->link = NULL;
145
146      if(first == NULL)
147          return temp;
148
149      if(elem <= first->info)
150      {
151          temp->link = first;
152          return temp;
153      }
154
155      prev = NULL;
```

```
156        cur = first;
157
158        while(cur != NULL && elem > cur->info)
159        {
160            prev = cur;
161            cur = cur->link;
162        }
163        prev->link = temp;
164        temp->link = cur;
165        return first;
166    }
167
168    void fnDisplay(NODEPTR first)
169    {
170        NODEPTR temp;
171
172        if(first == NULL)
173        {
174            printf("\nList is Empty\n");
175            return;
176        }
177
178        printf("=================================\n");
179        for(temp = first; temp != NULL; temp = temp->link)
180            printf("%4d",temp->info);
181        printf("\n=================================\n");
182    }
183
184    NODEPTR fnInsertRear(int iElem, NODEPTR first)
185    {
186        NODEPTR temp,cur;
187        temp = fnGetNode();
188        temp->info = iElem;
189        temp->link = NULL;
190
191        if(first == NULL)
192            return temp;
193
194        cur = first;
195        while(cur->link != NULL)
196        {
197            cur = cur->link;
198        }
199        cur->link = temp;
200        return first;
201    }
```

Listing 8.1: 08OrderedSinglyLinkedList.c

Output

```
=================================
1   /***************************************
2   putta:Programs$ gcc 08OrderedSinglyLinkedList.c -Wall
3   putta:Programs$ ./a.out
4
5   1.Insert into List 1
6   2.Insert into List 2
7   3.Display
8   4.Union
9   5.Intersection
10  6.Exit
```

```
11  Enter your choice
12  1
13
14  Enter a element
15  6
16
17  1.Insert into List 1
18  2.Insert into List 2
19  3.Display
20  4.Union
21  5.Intersection
22  6.Exit
23  Enter your choice
24  1
25
26  Enter a element
27  4
28
29  1.Insert into List 1
30  2.Insert into List 2
31  3.Display
32  4.Union
33  5.Intersection
34  6.Exit
35  Enter your choice
36  1
37
38  Enter a element
39  2
40
41  1.Insert into List 1
42  2.Insert into List 2
43  3.Display
44  4.Union
45  5.Intersection
46  6.Exit
47  Enter your choice
48  2
49
50  Enter a element
51  7
52
53  1.Insert into List 1
54  2.Insert into List 2
55  3.Display
56  4.Union
57  5.Intersection
58  6.Exit
59  Enter your choice
60  2
61
62  Enter a element
63  5
64
65  1.Insert into List 1
66  2.Insert into List 2
67  3.Display
68  4.Union
69  5.Intersection
70  6.Exit
71  Enter your choice
72  2
```

```
73
74  Enter a element
75  3
76
77  1.Insert into List 1
78  2.Insert into List 2
79  3.Display
80  4.Union
81  5.Intersection
82  6.Exit
83  Enter your choice
84  3
85
86  List 1 Contents
87  ================================
88      2    4    6
89  ================================
90
91  List 2 Contents
92  ================================
93      3    5    7
94  ================================
95
96
97  1.Insert into List 1
98  2.Insert into List 2
99  3.Display
100 4.Union
101 5.Intersection
102 6.Exit
103 Enter your choice
104 5
105
106 List 1 Contents
107 ================================
108     2    4    6
109 ================================
110
111 List 2 Contents
112 ================================
113     3    5    7
114 ================================
115
116 Intersection
117
118 List is Empty
119
120 1.Insert into List 1
121 2.Insert into List 2
122 3.Display
123 4.Union
124 5.Intersection
125 6.Exit
126 Enter your choice
127 2
128
129 Enter a element
130 2
131
132 1.Insert into List 1
133 2.Insert into List 2
134 3.Display
```

```
135  4.Union
136  5.Intersection
137  6.Exit
138  Enter your choice
139  2
140
141  Enter a element
142  4
143
144  1.Insert into List 1
145  2.Insert into List 2
146  3.Display
147  4.Union
148  5.Intersection
149  6.Exit
150  Enter your choice
151  3
152
153  List 1 Contents
154  ================================
155     2    4    6
156  ================================
157
158  List 2 Contents
159  ================================
160     2    3    4    5    7
161  ================================
162
163  1.Insert into List 1
164  2.Insert into List 2
165  3.Display
166  4.Union
167  5.Intersection
168  6.Exit
169  Enter your choice
170  5
171
172  List 1 Contents
173  ================================
174     2    4    6
175  ================================
176
177  List 2 Contents
178  ================================
179     2    3    4    5    7
180  ================================
181
182  Intersection
183  ================================
184     2    4
185  ================================
186
187  1.Insert into List 1
188  2.Insert into List 2
189  3.Display
190  4.Union
191  5.Intersection
192  6.Exit
193  Enter your choice
194  4
195
196  List 1 Contents
```

```
197  ====================================
198      2    4    6
199  ====================================
200
201  List 2 Contents
202  ====================================
203      2    3    4    5    7
204  ====================================
205
206  Union
207  ====================================
208      2    3    4    5    6    7
209  ====================================
210
211  1.Insert into List 1
212  2.Insert into List 2
213  3.Display
214  4.Union
215  5.Intersection
216  6.Exit
217  Enter your choice
218  6
219
220  *************************************/
```

Listing 8.2: out8.c

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

```c
#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);
```

```c
39                break;
40
41            case 3: fnDisplay(first);
42                    break;
43
44            case 4: exit(0);
45        }
46    }
47    return 0;
48 }
49
50 NODEPTR fnGetNode()
51 {
52    NODEPTR newborn;
53    newborn = (NODEPTR)malloc(sizeof(struct node));
54
55    if(newborn == NULL)
56    {
57        printf("\nMemory Overflow");
58        exit(0);
59    }
60    return newborn;
61 }
62
63 void fnFreeNode(NODEPTR x)
64 {
65    free(x);
66 }
67
68
69 NODEPTR fnPush(int iElem,NODEPTR first)  /*Insert front*/
70 {
71        NODEPTR temp;
72
73        temp = fnGetNode();
74
75        temp->Info = iElem;
76
77        temp->link = first;
78
79        return temp;
80 }
81
82 NODEPTR fnPop(NODEPTR first)                /*Delete front*/
83 {
84    NODEPTR temp;
85    if(first == NULL)
86    {
87        printf("\nStack is empty cannot delete\n");
88        return first;
89    }
90    temp = first;
91
92    first = first->link;
93
94    printf("\nElement deleted is %d \n",temp->Info);
95    fnFreeNode(temp);
96
97    return first;
98
99 }
100
```

```c
101
102
103 void fnDisplay(NODEPTR first)
104 {
105     NODEPTR curr;
106     if(first == NULL)
107     {
108         printf("\nStack is empty\n");
109         return;
110     }
111
112     printf("\nThe contents of Stack are :\n");
113     curr = first;
114     while(curr != NULL)
115     {
116         printf("\n%d",curr->Info);
117         curr = curr->link;
118     }
119     printf("\n");
120 }
```

Listing 9.1: 09aStackLL.c

Output

```
==================================
1  /**************************************
2  putta:Programs$ gcc 09aStackLL.c -Wall
3  putta:Programs$ ./a.out
4
5  STACK OPERATIONS
6  1.PUSH
7  2.POP
8  3.DISPLAY
9  4.EXIT
10
11 Enter your iChoice
12 2
13
14 Stack is empty cannot delete
15
16 STACK OPERATIONS
17 1.PUSH
18 2.POP
19 3.DISPLAY
20 4.EXIT
21
22 Enter your iChoice
23 1
24
25 Enter Element to be pushed onto Stack
26 1
27
28 STACK OPERATIONS
29 1.PUSH
30 2.POP
31 3.DISPLAY
32 4.EXIT
33
34 Enter your iChoice
35 1
36
```

```
37 Enter Element to be pushed onto Stack
38 2
39
40 STACK OPERATIONS
41 1.PUSH
42 2.POP
43 3.DISPLAY
44 4.EXIT
45
46 Enter your iChoice
47 1
48
49 Enter Element to be pushed onto Stack
50 3
51
52 STACK OPERATIONS
53 1.PUSH
54 2.POP
55 3.DISPLAY
56 4.EXIT
57
58 Enter your iChoice
59 1
60
61 Enter Element to be pushed onto Stack
62 4
63
64 STACK OPERATIONS
65 1.PUSH
66 2.POP
67 3.DISPLAY
68 4.EXIT
69
70 Enter your iChoice
71 3
72
73 The contents of Stack are :
74
75 4
76 3
77 2
78 1
79
80 STACK OPERATIONS
81 1.PUSH
82 2.POP
83 3.DISPLAY
84 4.EXIT
85
86 Enter your iChoice
87 2
88
89 Element deleted is 4
90
91 STACK OPERATIONS
92 1.PUSH
93 2.POP
94 3.DISPLAY
95 4.EXIT
96
97 Enter your iChoice
98 3
```

```
 99
100  The contents of Stack are :
101
102  3
103  2
104  1
105
106  STACK OPERATIONS
107  1.PUSH
108  2.POP
109  3.DISPLAY
110  4.EXIT
111
112  Enter your iChoice
113  2
114
115  Element deleted is 3
116
117  STACK OPERATIONS
118  1.PUSH
119  2.POP
120  3.DISPLAY
121  4.EXIT
122
123  Enter your iChoice
124  3
125
126  The contents of Stack are :
127
128  2
129  1
130
131  STACK OPERATIONS
132  1.PUSH
133  2.POP
134  3.DISPLAY
135  4.EXIT
136
137  Enter your iChoice
138  4
139
140  **************************************/
```

Listing 9.2: out9a.c

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(void);
void fnFreeNode(NODEPTR);
NODEPTR fnIns_Rear(int, NODEPTR);
NODEPTR fnDelFront(NODEPTR);
void fnDisplay(NODEPTR);

int 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(iChoice)
        {
            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: exit(0);
        }
    }
    return 0;
}

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

```
54
55      if(newborn == NULL)
56      {
57          printf("\nMemory Overflow");
58          exit(0);
59      }
60      return newborn;
61  }
62
63
64  void fnFreeNode(NODEPTR x)
65  {
66      free(x);
67  }
68
69
70  NODEPTR fnIns_Rear(int iElem,NODEPTR first)
71  {
72      NODEPTR temp,cur;
73
74      temp = fnGetNode();
75      temp->Info = iElem;
76      temp->link = NULL;
77
78      if(first == NULL)
79          return temp;
80      cur = first;
81      while(cur->link != NULL)
82      {
83          cur = cur->link;
84      }
85      cur->link = temp;
86      return first;
87  }
88
89  NODEPTR fnDelFront(NODEPTR first)
90  {
91      NODEPTR temp;
92      if(first == NULL)
93      {
94          printf("\nQueue is empty cannot delete\n");
95          return first;
96      }
97      temp = first;
98      first = first->link;
99      printf("\nElement deleted is %d \n",temp->Info);
100     fnFreeNode(temp);
101     return first;
102 }
103
104 void fnDisplay(NODEPTR first)
105 {
106     NODEPTR curr;
107     if(first == NULL)
108     {
109         printf("\nQueue is empty\n");
110         return;
111     }
112
113     printf("\nThe contents of Queue are :\n");
114     curr = first;
115     printf("\n");
```

```
116     while(curr != NULL)
117     {
118         printf("%d\t",curr->Info);
119         curr = curr->link;
120     }
121     printf("\n");
122 }
123
124 /*CPP*/
```

Listing 9.3: 09bQueueLL.c

Output

```
====================================
1 /***************************************
2 putta:Programs$ gcc 09bQueueLL.c -Wall
3 putta:Programs$ ./a.out
4
5 QUEUE OPERATIONS
6 ==================
7 1.Insert Rear
8 2.Delete Front
9 3.Display
10 4.Exit
11
12 Enter your choice
13 2
14
15 Queue is empty cannot delete
16
17 QUEUE OPERATIONS
18 ==================
19 1.Insert Rear
20 2.Delete Front
21 3.Display
22 4.Exit
23
24 Enter your choice
25 1
26
27 Enter Element to be inserted
28 1
29
30 QUEUE OPERATIONS
31 ==================
32 1.Insert Rear
33 2.Delete Front
34 3.Display
35 4.Exit
36
37 Enter your choice
38 1
39
40 Enter Element to be inserted
41 2
42
43 QUEUE OPERATIONS
44 ==================
45 1.Insert Rear
46 2.Delete Front
47 3.Display
```

```
48  4.Exit
49
50  Enter your choice
51  1
52
53  Enter Element to be inserted
54  3
55
56  QUEUE OPERATIONS
57  ====================
58  1.Insert Rear
59  2.Delete Front
60  3.Display
61  4.Exit
62
63  Enter your choice
64  1
65
66  Enter Element to be inserted
67  4
68
69  QUEUE OPERATIONS
70  ====================
71  1.Insert Rear
72  2.Delete Front
73  3.Display
74  4.Exit
75
76  Enter your choice
77  3
78
79  The contents of Queue are :
80
81  1    2    3    4
82
83  QUEUE OPERATIONS
84  ====================
85  1.Insert Rear
86  2.Delete Front
87  3.Display
88  4.Exit
89
90  Enter your choice
91  2
92
93  Element deleted is 1
94
95  QUEUE OPERATIONS
96  ====================
97  1.Insert Rear
98  2.Delete Front
99  3.Display
100 4.Exit
101
102 Enter your choice
103 3
104
105 The contents of Queue are :
106
107 2    3    4
108
109 QUEUE OPERATIONS
```

```
110  ====================
111  1.Insert Rear
112  2.Delete Front
113  3.Display
114  4.Exit
115
116  Enter your choice
117  4
118
119  ***************************************/
```

Listing 9.4: out9b.c

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;
}
```

```
54
55  //poly1 = fnInsertRear(iaCoeff[i], i, poly1);
56  NODEPTR fnInsertRear(int iCoeff, int iExp, NODEPTR first)
57  {
58      NODEPTR temp,cur;
59      temp = fnGetNode();
60      temp->coeff = iCoeff;
61      temp->exponent = iExp;
62      temp->link = NULL;
63
64      if(first == NULL)
65          return temp;
66      cur = first;
67      while(cur->link != NULL)
68      {
69          cur = cur->link;
70      }
71      cur->link = temp;
72      return first;
73  }
74
75  NODEPTR fnGetNode(void)
76  {
77      NODEPTR newNode;
78      newNode = ( NODEPTR ) malloc (sizeof(struct polynomial));
79      if(newNode == NULL)
80      {
81          printf("\nOut of Memory");
82          exit(0);
83      }
84      return newNode;
85  }
86
87  void fnDisplay(NODEPTR first)
88  {
89      NODEPTR cur;
90      for(cur = first; cur->link != NULL; cur = cur->link)
91      {
92  //      printf(" (%d)x^(%d) +",cur->coeff,cur->exponent);
93          printf(" (%d)x^%d +",cur->coeff,cur->exponent);
94      }
95      printf(" %d\n", cur->coeff);
96  }
97
98  //    poly3 = fnAddPoly(poly1, poly2);
99  NODEPTR fnAddPoly(NODEPTR poly1, NODEPTR poly2)
100 {
101     NODEPTR tracker1, tracker2,poly3 = NULL;
102     tracker1 = poly1;
103     tracker2 = poly2;
104     while(tracker1 != NULL && tracker2 != NULL)
105     {
106         if(tracker1->exponent > tracker2->exponent)
107         {
108             poly3 = fnInsertRear(tracker1->coeff, tracker1->exponent, poly3);
109             tracker1 = tracker1->link;
110         }
111         else if(tracker1->exponent == tracker2->exponent)
112         {
113             poly3 = fnInsertRear(tracker1->coeff + tracker2->coeff, tracker1->
    exponent, poly3);
114             tracker1 = tracker1->link;
```

```c
115             tracker2 = tracker2->link;
116         }
117         else
118         {
119             poly3 = fnInsertRear(tracker2->coeff, tracker2->exponent, poly3);
120             tracker2 = tracker2->link;
121         }
122     }
123     return poly3;
124 }
125
126 int evalPoly(NODEPTR list, int x)
127 {
128     int iSum = 0;
129     NODEPTR cur = list;
130     while(cur!=NULL)
131     {
132         iSum += (cur->coeff * (int)pow(x, cur->exponent));
133         cur = cur->link;
134     }
135     return iSum;
136 }
```

<div align="center">Listing 9.5: 09cPolynomial.c</div>

Output

```
==================================
/*************************************
putta:Programs$ gcc 09cPolynomial.c -Wall  -lm
putta:Programs$ ./a.out

Enter the degree of polynomial 1
4

Enter the coefficients
5 4 3 2 1

Enter the degree of polynomial 2
3

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

Enter the value of x
2

Value of the polynomial sum for x = 2 is 178

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

<div align="center">Listing 9.6: out9c.c</div>

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 fnGetNode(void);
void fnFreeNode(NODEPTR);
NODEPTR fnInsertFront(NODEPTR, int);
NODEPTR fnInsertNext(NODEPTR, int);
NODEPTR fnDeleteFirst(NODEPTR);
NODEPTR fnDelElemPos(NODEPTR, int);
void fnDisplay(NODEPTR);
NODEPTR fnSwapNodes(NODEPTR, int, int);

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

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

```

```
32      for(;;)
33
34      {
35          printf("\n1.Insert Front\n2.Insert to the next of a given Node");
36          printf("\n3.Delete First Node");
37          printf("\n4.Delete a Node whose position is specified");
38          printf("\n5.Display\n6.Swap Nodes\n7.Exit\n");
39
40          printf("\nEnter your Choice\n");
41          scanf("%d",&iChoice);
42
43          switch(iChoice)
44          {
45              case 1: printf("\nEnter the iItem to be inserted\n");
46                      scanf("%d",&iItem);
47                      head = fnInsertFront(head, iItem);
48                      break;
49
50              case 2: printf("\nEnter the key value of the node\n");
51                      scanf("%d", &iKey);
52                      head = fnInsertNext(head, iKey);
53                      break;
54
55              case 3: last = head->llink;
56                      head = fnDeleteFirst(last);
57                      break;
58
59              case 4: printf("\nEnter the position of the element to be deleted\n");
60                      scanf("%d",&iPos);
61                      head = fnDelElemPos(head, iPos);
62                      break;
63
64              case 5: fnDisplay(head);
65                      break;
66
67              case 6: printf("\nEnter the positions m and n of the nodes to be
    swapped such that m < n\n");
68                      scanf("%d%d",&iM, &iN);
69                      if(iM > iN)
70                      {
71                          printf("\nInvalid input\n");
72                      }
73                      else
74                      {
75                          head = fnSwapNodes(head, iM, iN);
76                      }
77                      break;
78
79              case 7: exit(0);
80          }
81      }
82      return 0;
83  }
84
85  NODEPTR fnGetNode(void)
86  {
87      NODEPTR x;
88      x = ( NODEPTR ) malloc (sizeof(struct node));
89      if(x == NULL)
90      {
91          printf("\nOut of Memory");
92          exit(0);
```

```
 93        }
 94        return x;
 95   }
 96
 97   void fnFreeNode(NODEPTR x)
 98   {
 99        free(x);
100   }
101
102   NODEPTR fnInsertFront(NODEPTR head, int iItem)
103   {
104        NODEPTR temp,cur;
105        temp = fnGetNode();
106        temp->info = iItem;
107
108        cur = head->rlink;
109
110        head->rlink = temp;
111        temp->llink = head;
112        temp->rlink = cur;
113        cur->llink = temp;
114
115        head->info += 1;
116        return head;
117   }
118
119   NODEPTR fnDeleteFirst(NODEPTR last)
120   {
121        NODEPTR second, first, head;
122
123        if(last->rlink == last)
124        {
125            printf("\nList is Empty");
126            return last;
127        }
128        head = last->rlink;
129        first = head->rlink;
130        second = first->rlink;
131
132        head->rlink = second;
133        second->llink = head;
134        printf("\nElement deleted is %d\n", first->info);
135        fnFreeNode(first);
136        head->info -= 1;
137
138        return head;
139   }
140
141   NODEPTR fnInsertNext(NODEPTR head, int iItem)
142   {
143        NODEPTR temp,cur,next;
144
145        if(head->rlink == head)
146        {
147            printf("\nList is Empty\n");
148            return head;
149        }
150
151        cur = head->rlink;
152
153        while(cur != head && iItem != cur->info)
154        {
```

```
155            cur = cur->rlink;
156        }
157        if(cur == head)
158        {
159            printf("\nSpecified Node not found\n");
160            return head;
161        }
162
163        next = cur->rlink;
164
165        printf("\nEnter the item to be inserted to the next of %d\n",iItem);
166
167        temp = fnGetNode();
168        scanf("%d",&temp->info);
169
170        cur->rlink = temp;
171        temp->llink = cur;
172        next->llink = temp;
173        temp->rlink = next;
174        head->info += 1;
175
176        return head;
177    }
178
179    NODEPTR fnDelElemPos(NODEPTR head, int iPos)
180    {
181
182        NODEPTR prev,cur,next;
183        int count = 1;
184
185        if(head->rlink == head)
186        {
187            printf("\nList is Empty\n");
188            return head;
189        }
190
191        cur = head->rlink;
192
193        while(cur != head && count != iPos)
194        {
195            cur = cur->rlink;
196            count++;
197        }
198
199        if(count == iPos)
200        {
201            prev = cur->llink;
202            next = cur->rlink;
203
204            prev->rlink = next;
205            next->llink = prev;
206            head->info -= 1;
207            printf("\nElement deleted is %d\n", cur->info);
208            fnFreeNode(cur);
209        }
210
211        if(cur == head)
212        {
213            printf("\nItem not found\n");
214            return head;
215        }
216        return head;
```

```
217   }
218
219   void fnDisplay(NODEPTR head)
220   {
221       NODEPTR temp;
222       if(head->rlink == head)
223       {
224           printf("\nList is empty\n");
225           return;
226       }
227
228       printf("Contents of the List is\n");
229       for(temp = head->rlink; temp != head; temp = temp->rlink)
230           printf("%d\t",temp->info);
231
232       printf("\n");
233       printf("\nThere are %d nodes in the list", head->info);
234       printf("\n");
235
236   }
237
238   NODEPTR fnSwapNodes(NODEPTR head, int m , int n)
239   {
240       int temp, count = 1;
241       NODEPTR cur, mpos, npos;
242       cur = head->rlink;
243
244       while(cur != head && count != m)
245       {
246           cur = cur->rlink;
247           count++;
248       }
249
250       if(cur != head)
251       {
252           mpos = cur;
253       }
254       else
255       {
256           printf("\nNode #%d does not exist\n", m);
257           return head;
258       }
259
260       while(cur != head && count != n)
261       {
262           cur = cur->rlink;
263           count++;
264       }
265       if(cur != head)
266       {
267           npos = cur;
268       }
269       else
270       {
271           printf("\nNode #%d does not exist\n", n);
272           return head;
273       }
274
275       temp = mpos->info;
276       mpos->info = npos->info;
277       npos->info = temp;
278
```

```
279     return head;
280 }
```

Listing 10.1: 10DoublyLinkedList.c

Output

```
==================================
/****************************************
putta:Programs$ gcc 10DoublyLinkedList.c -Wall
putta:Programs$ ./a.out

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit

Enter your Choice
1

Enter the iItem to be inserted
5

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit

Enter your Choice
1

Enter the iItem to be inserted
4

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit

Enter your Choice
5
Contents of the List is
4    5

There are 2 nodes in the list

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit
```

```
Enter your Choice
2

Enter the key value of the node
4

Enter the item to be inserted to the next of 4
8

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit

Enter your Choice
5
Contents of the List is
4    8    5

There are 3 nodes in the list

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit

Enter your Choice
6

Enter the positions m and n of the nodes to be swapped such that m < n
2
3

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit

Enter your Choice
5
Contents of the List is
4    5    8

There are 3 nodes in the list

1.Insert Front
2.Insert to the next of a given Node
3.Delete First Node
4.Delete a Node whose position is specified
5.Display
6.Swap Nodes
7.Exit
```

117
118 **Enter your Choice**
119 **2**
120
121 **Enter the key value of the node**
122 **5**
123
124 **Enter the item to be inserted to the next of 5**
125 **6**
126
127 **1.Insert Front**
128 **2.Insert to the next of a given Node**
129 **3.Delete First Node**
130 **4.Delete a Node whose position is specified**
131 **5.Display**
132 **6.Swap Nodes**
133 **7.Exit**
134
135 **Enter your Choice**
136 **5**
137 **Contents of the List is**
138 **4   5   6   8**
139
140 **There are 4 nodes in the list**
141
142 **1.Insert Front**
143 **2.Insert to the next of a given Node**
144 **3.Delete First Node**
145 **4.Delete a Node whose position is specified**
146 **5.Display**
147 **6.Swap Nodes**
148 **7.Exit**
149
150 **Enter your Choice**
151 **1**
152
153 **Enter the iItem to be inserted**
154 **9**
155
156 **1.Insert Front**
157 **2.Insert to the next of a given Node**
158 **3.Delete First Node**
159 **4.Delete a Node whose position is specified**
160 **5.Display**
161 **6.Swap Nodes**
162 **7.Exit**
163
164 **Enter your Choice**
165 **5**
166 **Contents of the List is**
167 **9   4   5   6   8**
168
169 **There are 5 nodes in the list**
170
171 **1.Insert Front**
172 **2.Insert to the next of a given Node**
173 **3.Delete First Node**
174 **4.Delete a Node whose position is specified**
175 **5.Display**
176 **6.Swap Nodes**
177 **7.Exit**
178

```
179 Enter your Choice
180 4
181
182 Enter the position of the element to be deleted
183 4
184
185 1.Insert Front
186 2.Insert to the next of a given Node
187 3.Delete First Node
188 4.Delete a Node whose position is specified
189 5.Display
190 6.Swap Nodes
191 7.Exit
192
193 Enter your Choice
194 5
195 Contents of the List is
196 9    4    5    8
197
198 There are 4 nodes in the list
199
200 1.Insert Front
201 2.Insert to the next of a given Node
202 3.Delete First Node
203 4.Delete a Node whose position is specified
204 5.Display
205 6.Swap Nodes
206 7.Exit
207
208 Enter your Choice
209 3
210
211 1.Insert Front
212 2.Insert to the next of a given Node
213 3.Delete First Node
214 4.Delete a Node whose position is specified
215 5.Display
216 6.Swap Nodes
217 7.Exit
218
219 Enter your Choice
220 5
221 Contents of the List is
222 4    5    8
223
224 There are 3 nodes in the list
225
226 1.Insert Front
227 2.Insert to the next of a given Node
228 3.Delete First Node
229 4.Delete a Node whose position is specified
230 5.Display
231 6.Swap Nodes
232 7.Exit
233
234 Enter your Choice
235 7
236
237 *************************************/
```

Listing 10.2: out10.c

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-Linear List Implementation

```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 , int);
NODEPTR fnDeleteFront(NODEPTR);
NODEPTR fnInsertRear(NODEPTR , int );
NODEPTR fnDeleteRear(NODEPTR);
void fnDisplay(NODEPTR);

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

    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");
```

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

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

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

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

                case 5: fnDisplay(first);
                        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 first, int iItem)
{
    NODEPTR temp;
    temp = fnGetNode();
    temp = fnGetNode();
    temp->info = iItem;
    temp->llink = temp->rlink = NULL;

    if(first == NULL)
    {
        return temp;
    }

    temp->rlink=first;
    first->llink = temp;
    return temp;
}

NODEPTR fnInsertRear(NODEPTR first, int iItem)
{
    NODEPTR temp,cur;
```

```
98      temp = fnGetNode();
99      temp = fnGetNode();
100     temp->info = iItem;
101     temp->llink = temp->rlink = NULL;
102
103     if(first == NULL)
104     {
105         return temp;
106     }
107
108     cur = first;
109     while(cur->rlink != NULL)
110     {
111         cur = cur->rlink;
112     }
113     cur->rlink = temp;
114     temp->llink = cur;
115     return first;
116 }
117
118 NODEPTR fnDeleteFront(NODEPTR first)
119 {
120     NODEPTR temp;
121     if(first == NULL)
122     {
123         printf("\nList is Empty\n");
124         return first;
125     }
126     if(first->rlink == NULL)
127     {
128         printf("\nElement deleted is %d\n", first->info);
129         fnFreeNode(first);
130         return NULL;
131     }
132     temp = first;
133     first = first->rlink;
134     first->llink = NULL;
135     printf("\nElement deleted is %d\n", temp->info);
136     fnFreeNode(temp);
137     return first;
138 }
139
140 NODEPTR fnDeleteRear(NODEPTR first)
141 {
142     NODEPTR secondLast, cur;
143     if(first == NULL)
144     {
145         printf("\nList is Empty\n");
146         return first;
147     }
148     if(first->rlink == NULL)
149     {
150         printf("\nElement deleted is %d\n", first->info);
151         fnFreeNode(first);
152         return NULL;
153     }
154
155     cur = first;
156     while(cur->rlink != NULL)
157     {
158         cur = cur->rlink;
159     }
```

```c
160
161        secondLast = cur->llink;
162        secondLast->rlink = NULL;
163        printf("\nElement deleted is %d\n", cur->info);
164        fnFreeNode(cur);
165        return first;
166 }
167
168 void fnDisplay(NODEPTR first)
169 {
170        NODEPTR temp;
171        if(first == NULL)
172        {
173            printf("\nList is Empty\n");
174            return;
175        }
176        printf("Contents of the List is\n");
177        for(temp = first; temp != NULL; temp = temp->rlink)
178            printf("%d\t",temp->info);
179        printf("\n");
180 }
```

Listing 11.1: 11DequeueLDLL.c

C Code-Circular List Implementation

```c
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int info;
7      struct node *llink;
8      struct node *rlink;
9  };
10 typedef struct node* NODEPTR;
11
12 NODEPTR fnGetNode(void);
13 void fnFreeNode(NODEPTR);
14 NODEPTR fnInsertFront(NODEPTR , int);
15 NODEPTR fnDeleteFront(NODEPTR);
16 NODEPTR fnInsertRear(NODEPTR , int);
17 NODEPTR fnDeleteRear(NODEPTR);
18 void fnDisplay(NODEPTR);
19
20 int main()
21 {
22     NODEPTR head;
23     int iChoice, iItem;
24
25     head = fnGetNode();
26     head->rlink = head;
27     head->llink = head;
28
29     for(;;)
30     {
31         printf("\n1.Insert Front\n2.Insert Rear");
32         printf("\n3.Delete Front\n4.Delete Rear");
33         printf("\n5.Display\n6.Exit\n");
34         printf("\nEnter your Choice\n");
35         scanf("%d",&iChoice);
36
```

```
37          switch(iChoice)
38          {
39              case 1: printf("\nEnter the iItem to be inserted\n");
40                      scanf("%d",&iItem);
41                      head = fnInsertFront(head, iItem);
42                      break;
43
44              case 2: printf("\nEnter the iItem to be inserted\n");
45                      scanf("%d",&iItem);
46                      head = fnInsertRear(head, iItem);
47                      break;
48
49              case 3: head = fnDeleteFront(head);
50                      break;
51
52              case 4: head = fnDeleteRear(head);
53                      break;
54
55              case 5: fnDisplay(head);
56                      break;
57
58              case 6: exit(0);
59          }
60      }
61      return 0;
62 }
63
64 NODEPTR fnGetNode(void)
65 {
66      NODEPTR x;
67      x = ( NODEPTR ) malloc (sizeof(struct node));
68      if(x == NULL)
69      {
70          printf("\nOut of Memory");
71          exit(0);
72      }
73      return x;
74 }
75
76 void fnFreeNode(NODEPTR x)
77 {
78      free(x);
79 }
80
81 NODEPTR fnInsertFront(NODEPTR head, int iItem)
82 {
83      NODEPTR temp,cur;
84      temp = fnGetNode();
85      temp = fnGetNode();
86      temp->info = iItem;
87
88      cur = head->rlink;
89      head->rlink = temp;
90      temp->llink = head;
91      temp->rlink = cur;
92      cur->llink = temp;
93      return head;
94 }
95
96 NODEPTR fnInsertRear(NODEPTR head, int iItem)
97 {
98      NODEPTR temp,cur;
```

```
99      temp = fnGetNode();
100     temp = fnGetNode();
101     temp->info = iItem;
102
103     cur = head->llink;
104     head->llink = temp;
105     temp->rlink = head;
106     temp->llink = cur;
107     cur->rlink = temp;
108     return head;
109 }
110
111 NODEPTR fnDeleteFront(NODEPTR head)
112 {
113     NODEPTR second, first;
114     if(head->rlink == head)
115     {
116         printf("\nList is Empty\n");
117         return head;
118     }
119     first = head->rlink;
120     second = first->rlink;
121
122     head->rlink = second;
123     second->llink = head;
124     printf("\nElement deleted is %d\n", first->info);
125     fnFreeNode(first);
126     return head;
127 }
128
129 NODEPTR fnDeleteRear(NODEPTR head)
130 {
131     NODEPTR secondLast, last;
132     if(head->rlink == head)
133     {
134         printf("\nList is Empty\n");
135         return head;
136     }
137     last = head->llink;
138     secondLast = last->llink;
139
140     head->llink = secondLast;
141     secondLast->rlink = head;
142     printf("\nElement deleted is %d\n", last->info);
143     fnFreeNode(last);
144     return head;
145 }
146
147 void fnDisplay(NODEPTR head)
148 {
149     NODEPTR temp;
150     if(head->rlink == head)
151     {
152         printf("\nList is empty\n");
153         return;
154     }
155     printf("Contents of the List is\n");
156     for(temp = head->rlink; temp != head; temp = temp->rlink)
157         printf("%d\t",temp->info);
158     printf("\n");
```

```
159 }
```

Listing 11.2: 11DequeueDLL.c

Output

```
=================================
/***************************************
putta:Programs$ gcc 11DequeueDLL.c
putta:Programs$ ./a.out

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Display
6.Exit

Enter your Choice
5

List is empty

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Display
6.Exit

Enter your Choice
3

List is Empty

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Display
6.Exit

Enter your Choice
1

Enter the iItem to be inserted
3

1.Insert Front
2.Insert Rear
3.Delete Front
4.Delete Rear
5.Display
6.Exit

Enter your Choice
1

Enter the iItem to be inserted
2

1.Insert Front
```

```
56  2.Insert Rear
57  3.Delete Front
58  4.Delete Rear
59  5.Display
60  6.Exit
61
62  Enter your Choice
63  2
64
65  Enter the iItem to be inserted
66  4
67
68  1.Insert Front
69  2.Insert Rear
70  3.Delete Front
71  4.Delete Rear
72  5.Display
73  6.Exit
74
75  Enter your Choice
76  5
77  Contents of the List is
78  2    3    4
79
80  1.Insert Front
81  2.Insert Rear
82  3.Delete Front
83  4.Delete Rear
84  5.Display
85  6.Exit
86
87  Enter your Choice
88  2
89
90  Enter the iItem to be inserted
91  5
92
93  1.Insert Front
94  2.Insert Rear
95  3.Delete Front
96  4.Delete Rear
97  5.Display
98  6.Exit
99
100 Enter your Choice
101 2
102
103 Enter the iItem to be inserted
104 6
105
106 1.Insert Front
107 2.Insert Rear
108 3.Delete Front
109 4.Delete Rear
110 5.Display
111 6.Exit
112
113 Enter your Choice
114 5
115 Contents of the List is
116 2    3    4    5    6
117
```

```
118  1.Insert Front
119  2.Insert Rear
120  3.Delete Front
121  4.Delete Rear
122  5.Display
123  6.Exit
124
125  Enter your Choice
126  3
127
128  Element deleted is 2
129
130  1.Insert Front
131  2.Insert Rear
132  3.Delete Front
133  4.Delete Rear
134  5.Display
135  6.Exit
136
137  Enter your Choice
138  5
139  Contents of the List is
140  3    4    5    6
141
142  1.Insert Front
143  2.Insert Rear
144  3.Delete Front
145  4.Delete Rear
146  5.Display
147  6.Exit
148
149  Enter your Choice
150  4
151
152  Element deleted is 6
153
154  1.Insert Front
155  2.Insert Rear
156  3.Delete Front
157  4.Delete Rear
158  5.Display
159  6.Exit
160
161  Enter your Choice
162  5
163  Contents of the List is
164  3    4    5
165
166  1.Insert Front
167  2.Insert Rear
168  3.Delete Front
169  4.Delete Rear
170  5.Display
171  6.Exit
172
173  Enter your Choice
174  6
175
176  **************************************/
```

Listing 11.3: out11.c

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 *lbranch;
    struct node *rbranch;
};
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");
```

```c
36                    scanf("%d",&iItem);
37                    root = fnInsertNode(iItem,root);
38                    break;
39
40            case 2: if(root ==NULL)
41                    {
42                        printf("\nTree is Empty\n");
43                    }
44                    else
45                    {
46                        printf("\nInorder Traversal is :\n");
47                        fnInOrder(root);
48                        printf("\n");
49                    }
50                    break;
51
52            case 3: if(root ==NULL)
53                    {
54                        printf("\nTree is Empty\n");
55                    }
56                    else
57                    {
58                        printf("\nPreorder Traversal is :\n");
59                        fnPreOrder(root);
60                        printf("\n");
61                    }
62                    break;
63
64            case 4: if(root ==NULL)
65                    {
66                        printf("\nTree is Empty\n");
67                    }
68                    else
69                    {
70                        printf("\nPostorder Traversal is :\n");
71                        fnPostOrder(root);
72                        printf("\n");
73                    }
74                    break;
75
76            case 5: printf("\nEnter node to be deleted : ");
77                    scanf("%d", &iItem);
78                    root = fnDeleteNode(root, iItem);
79                    break;
80
81            case 6: exit(0);
82
83            default: printf("Wrong choice\n");
84                     break;
85
86        }
87
88    }
89    return 0;
90 }
91
92 NODEPTR fnGetNode(void)
93 {
94     NODEPTR x;
95     x = ( NODEPTR ) malloc (sizeof(struct node));
96     if(x == NULL)
97     {
```

```c
 98         printf("\nOut of Memory");
 99         exit(0);
100     }
101     return x;
102 }
103
104 void fnFreeNode(NODEPTR x)
105 {
106     free(x);
107 }
108
109 NODEPTR fnInsertNode(int iItem,NODEPTR root)
110 {
111     NODEPTR temp,prev,cur;
112
113     temp = fnGetNode();
114     temp->info = iItem;
115     temp->lbranch = NULL;
116     temp->rbranch = NULL;
117
118     if(root == NULL)
119     return temp;
120
121     prev = NULL;
122     cur = root;
123
124     while(cur != NULL)
125     {
126         prev = cur;
127
128         if(iItem == cur->info)
129         {
130             printf("\nDuplicate items not allowed\n");
131             fnFreeNode(temp);
132             return root;
133         }
134
135         cur = (iItem < cur->info)? cur->lbranch: cur->rbranch;
136     }
137
138     if(iItem < prev->info)
139         prev->lbranch = temp;
140     else
141         prev->rbranch = temp;
142
143     return root;
144
145 }
146
147 void fnPreOrder(NODEPTR root)
148 {
149     if(root != NULL)
150     {
151         printf("%d\t",root->info);
152         fnPreOrder(root->lbranch);
153         fnPreOrder(root->rbranch);
154     }
155 }
156
157 void fnInOrder(NODEPTR root)
158 {
159     if(root != NULL)
```

```
160        {
161            fnInOrder(root->lbranch);
162            printf("%d\t",root->info);
163            fnInOrder(root->rbranch);
164        }
165  }
166
167  void fnPostOrder(NODEPTR root)
168  {
169        if(root != NULL)
170        {
171            fnPostOrder(root->lbranch);
172            fnPostOrder(root->rbranch);
173            printf("%d\t",root->info);
174        }
175  }
176
177  NODEPTR fnDeleteNode(NODEPTR root, int iItem)
178  {
179        if(root == NULL)
180        {
181            printf("\nBST is empty, cannot delete");
182            return root;
183        }
184        // If the item to be deleted is smaller than the root's item,
185        // then it lies in left subtree
186        if (iItem < root->info)
187            root->lbranch = fnDeleteNode(root->lbranch, iItem);
188
189        // If the item to be deleted is greater than the root's item,
190        // then it lies in right subtree
191        else if (iItem > root->info)
192            root->rbranch = fnDeleteNode(root->rbranch, iItem);
193
194        // if item is same as root's item, then This is the node
195        // to be deleted
196        else
197        {
198            // node with only one child or no child
199            if (root->lbranch == NULL)
200            {
201                NODEPTR temp = root->rbranch;
202                free(root);
203                return temp;
204            }
205            else if (root->rbranch == NULL)
206            {
207                NODEPTR temp = root->lbranch;
208                free(root);
209                return temp;
210            }
211
212            // node with two children: Get the inorder successor (smallest
213            // in the right subtree)
214            NODEPTR temp = fnMinValueNode(root->rbranch);
215
216            // Copy the inorder successor's content to this node
217            root->info = temp->info;
218
219            // Delete the inorder successor
220            root->rbranch = fnDeleteNode(root->rbranch, temp->info);
221        }
```

```
222     return root;
223 }
224
225 NODEPTR fnMinValueNode(NODEPTR node)
226 {
227     NODEPTR current = node;
228
229     /* loop down to find the leftmost leaf */
230     while (current->lbranch != NULL)
231         current = current->lbranch;
232
233     return current;
234 }
```

Listing 12.1: 12BinarySearchTree.c

Output

```
==================================
1 /***********************************
2 putta:Programs$ gcc 12BinarySearchTree.c
3 putta:Programs$ ./a.out
4
5 1.Insert a node
6 2.Inorder traversal
7 3.Preorder traversal
8 4.Postorder traversal
9 5.Delete a node
10 6.Exit
11
12 Enter your choice1
13 Enter the item to be inserted
14 6
15
16 1.Insert a node
17 2.Inorder traversal
18 3.Preorder traversal
19 4.Postorder traversal
20 5.Delete a node
21 6.Exit
22
23 Enter your choice1
24 Enter the item to be inserted
25 4
26
27 1.Insert a node
28 2.Inorder traversal
29 3.Preorder traversal
30 4.Postorder traversal
31 5.Delete a node
32 6.Exit
33
34 Enter your choice1
35 Enter the item to be inserted
36 8
37
38 1.Insert a node
39 2.Inorder traversal
40 3.Preorder traversal
41 4.Postorder traversal
42 5.Delete a node
43 6.Exit
```

```
Enter your choice1
Enter the item to be inserted
5

1.Insert a node
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete a node
6.Exit

Enter your choice1
Enter the item to be inserted
1

1.Insert a node
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete a node
6.Exit

Enter your choice2

Inorder Traversal is :
1    4    5    6    8

1.Insert a node
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete a node
6.Exit

Enter your choice3

Preorder Traversal is :
6    4    1    5    8

1.Insert a node
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete a node
6.Exit

Enter your choice4

Postorder Traversal is :
1    5    4    8    6

1.Insert a node
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete a node
6.Exit

Enter your choice5

Enter node to be deleted : 5
```

```
106
107  1.Insert a node
108  2.Inorder traversal
109  3.Preorder traversal
110  4.Postorder traversal
111  5.Delete a node
112  6.Exit
113
114  Enter your choice3
115
116  Preorder Traversal is :
117  6    4    1    8
118
119  1.Insert a node
120  2.Inorder traversal
121  3.Preorder traversal
122  4.Postorder traversal
123  5.Delete a node
124  6.Exit
125
126  Enter your choice2
127
128  Inorder Traversal is :
129  1    4    6    8
130
131  1.Insert a node
132  2.Inorder traversal
133  3.Preorder traversal
134  4.Postorder traversal
135  5.Delete a node
136  6.Exit
137
138  Enter your choice3
139
140  Preorder Traversal is :
141  6    4    1    8
142
143  1.Insert a node
144  2.Inorder traversal
145  3.Preorder traversal
146  4.Postorder traversal
147  5.Delete a node
148  6.Exit
149
150  Enter your choice6
151
152  **************************************/
```

Listing 12.2: out12.c

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 <math.h>
#include <string.h>
#include <ctype.h>

// An expression tree node
struct ExpTree
{
    char value;
    struct ExpTree *lbranch, *rbranch;
};
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*);
float evalPost(NODEPTR);

int main()
{
    char postfix[30];
    float fResult;
    printf("\nEnter a postfix expression\n");
    scanf("%s", postfix);
    NODEPTR et = constructTree(postfix);
    printf("infix expression is \n");
    inorder(et);
    printf("\n");
    fResult = evalPost(et);
    printf("\nValue of Postfix Expression is : %g\n\n", fResult);
    return 0;
}
// A utility function to check if 'c' is an operator
```

```
39 bool isOperator(char c)
40 {
41     if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
42         return true;
43     return false;
44 }
45 // Utility function to do inorder traversal
46 void inorder(NODEPTR t)
47 {
48     if(t)
49     {
50         inorder(t-> lbranch);
51         printf("%c ", t->value);
52         inorder(t->rbranch);
53     }
54 }
55
56 NODEPTR newNode(int v)
57 {
58     NODEPTR temp =(NODEPTR) malloc(sizeof(struct ExpTree));
59     temp-> lbranch = temp->rbranch = NULL;
60     temp->value = v;
61     return temp;
62 }
63
64 NODEPTR constructTree(char postfix[])
65 {
66     NODEPTR stack[100];
67     int i, top = -1;
68     NODEPTR t, t1, t2;
69     // Traverse through every character of input expression
70     for(i=0; i<strlen(postfix); i++)
71     {
72         // If operand, simply push into stack
73         if(!isOperator(postfix[i]))
74         {
75             t = newNode(postfix[i]);
76             push(stack, t, &top);
77         }
78         else // operator
79         {
80             t = newNode(postfix[i]);
81             // Pop two top nodes
82             t1 = pop(stack, &top);       // Remove top
83             t2 = pop(stack, &top);
84             //  make them children
85             t->rbranch = t1;
86             t-> lbranch = t2;
87             // Add this subexpression to stack
88             push(stack, t, &top);
89         }
90     }
91     //  only element will be root of expression tree
92     t = pop(stack, &top);
93     return t;
94 }
95
96 void push(NODEPTR st[], NODEPTR p, int *t)
97 {
98     *t = *t + 1;
99     st[*t] = p;
100 }
```

```
101
102  NODEPTR pop(NODEPTR st[], int *t)
103  {
104      NODEPTR temp;
105      temp = st[*t];
106      *t = *t - 1;
107      return temp;
108  }
109
110  float evalPost(NODEPTR root)
111  {
112      float fNum;
113      switch(root->value)
114      {
115          case '+' : return (evalPost(root->lbranch) + evalPost(root->rbranch));
116          case '-' : return (evalPost(root->lbranch) - evalPost(root->rbranch));
117          case '*' : return (evalPost(root->lbranch) * evalPost(root->rbranch));
118          case '/' : return (evalPost(root->lbranch) / evalPost(root->rbranch));
119          case '^' : return (pow(evalPost(root->lbranch), evalPost(root->rbranch)))
         ;
120          default:    if(isalpha(root->value))
121                      {
122                          printf("\n%c = ",root->value);
123                          scanf("%f",&fNum);
124                          return(fNum);
125                      }
126                      else
127                          return(root->value - '0');
128      }
129  }
```

Listing 13.1: 13ExpressionTree.c

Output

```
===================================
1  /*************************************
2  putta:Programs$ gcc 13ExpressionTree.c -lm
3  putta:Programs$ ./a.out
4
5  Enter a postfix expression
6  45*35/+
7  infix expression is
8  4 * 5 + 3 / 5
9  Value of Postfix Expression is : 20.6
10
11  putta:Programs$ ./a.out
12
13  Enter a postfix expression
14  abc-*de++
15  infix expression is
16  a * b - c + d + e
17  a = 2
18  b = 6
19  c = 4
20  d = 2
21  e = 2
22  Value of Postfix Expression is : 8
23
24  ************************************/
```

Listing 13.2: out13.c