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

==========

```
#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");
```

```
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  }
```

```
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");
```

```
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     {
```

```
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 }
```

Listing 1.1: 01EmployeeDB.c

1.2 C Code - Binary I/O

==========

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

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

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

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

Listing 1.2: 01EmployeeDBBinary.c

Output

==========

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

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

Enter Employee details

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

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

 Employee Details

ID  Name    Dept    Salary  Age
123 Raju    CSE 24000   26

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

Enter Employee details

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

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

Enter Employee details
```

```
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

==========

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

#define MAX 4

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

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

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

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

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

```
39
40              case 4: if(!fnStkEmpty(top))
41                      {
42                          iElem = fnPeek(stkArray, top);
43                          printf("\nElement at the  top of the stack is %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     {
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 }
```

Listing 2.1: 02Stack.c

2.2 C Code - Structure Representation

==========

```
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 is %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;
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
```

```
Enter your choice
4

Element at the  top of the stack is 3

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

Popped Element is 3

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

Element at the  top of the stack is 2

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

Enter element to be pushed onto the stack
3

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

Enter element to be pushed onto the stack
4

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

```
Enter your choice
3

Stack Contents are:
    4
    3
    2
    1

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

Stack Overflow

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

Listing 2.3: out2.c

Chapter 3

Infix to Postfix Conversion

Question

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

C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#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;
    char acExpr[50], acStack[50], acPost[50], cSymb;
    int top = -1;

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

    fnPush(acStack, &top, '#');
    for(i=0;acExpr[i]!='\0'; ++i)
    {
        cSymb = acExpr[i];
        if(isdigit(cSymb))
        {
            fnPush(acStack, &top, cSymb);
        }
        else if(cSymb == '(')
        {
            fnPush(acStack, &top, cSymb);
        }
        else if(cSymb == ')')
        {
            while(acStack[top] != '(')
            {
                acPost[j++] = fnPop(acStack, &top);
            }
            fnPop(acStack, &top);
```

```
        }
        else
        {
            while(fnPrecd(acStack[top]) >= fnPrecd(cSymb))
            {
                if(cSymb == '^' && acStack[top] == '^')
                    break;
                acPost[j++] = fnPop(acStack, &top);
            }
            fnPush(acStack, &top, cSymb);
        }

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

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

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

}

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

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

Listing 3.1: 03ConvInfix.c

Output

```
/*************************************
putta:Programs$ gcc 03ConvInfix.c
putta:Programs$ ./a.out
```

```
Enter a valid infix expression
(a^b^c)+(d/(e-f))

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

Postfix Expression is abc^^def-/+

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

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

Postfix Expression is abc-*abc*+/

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

Listing 3.2: out3.c

Chapter 4

Evaluation of Prefix Expression

Question

Write a C program to evaluate the given prefix expression.

C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#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;
    char acExpr[50], cSymb;
    int top = -1;

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

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

```
                                break;
                }
                fnPush(iaStack, &top, iRes);
        }

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

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

}

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

Listing 4.1: 04EvalPrefix.c

Output

==========

```
/*************************************
putta:Programs$ gcc 04EvalPrefix.c -Wall
putta:Programs$ ./a.out

Enter a valid prefix expression
++34*23

Value of ++34*23 expression is 13

putta:Programs$ ./a.out

Enter a valid prefix expression
*+-525*25

Value of *+-525*25 expression is 80

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

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

==========

```c
#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;
    int iElem, iChoice;

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

```
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 r)
74  {
75      if(r == QUEUE_SIZE-1)
76          return true;
77      else
78          return false;
79  }
80
81  bool fnQueueEmpty(int queue[], int f, int r)
82  {
83      if(r == f-1)
84          return true;
85      else
86          return false;
87  }
88
89  void fnInsertRear(int queue[], int *r, int iVal)
90  {
91      *r = *r + 1;
92      queue[*r] = iVal;
93  }
94
95  int fnDeleteFront(int queue[], int *f, int *r)
96  {
97      int iElem;
98      iElem = queue[*f];
99
```

```
100     if(*f == *r)
101     {
102         *f = 0;
103         *r = -1;
104     }
105     else
106     {
107         *f = *f + 1;
108     }
109     return iElem;
110 }
111 
112 void fnDisplay(int queue[], int f, int r)
113 {
114     int i;
115     for(i=f; i<=r; i++)
116     {
117         printf("%d\t", queue[i]);
118     }
119     printf("\n");
120 }
```

Listing 5.1: 05LinearQueue.c

5.2 C Code - Structure Representation

==========

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

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

```
97          return false;
98  }
99
100 void fnInsertRear(QUEUE_T *myQ, int iVal)
101 {
102     (myQ->iRear)++;
103     myQ->Queue[myQ->iRear] = iVal;
104 }
105
106 int fnDeleteFront(QUEUE_T *myQ)
107 {
108     int iElem;
109     iElem = myQ->Queue[myQ->iFront];
110
111     if(myQ->iFront == myQ->iRear)
112     {
113         myQ->iFront = 0;
114         myQ->iRear = -1;
115     }
116     else
117     {
118         myQ->iFront = myQ->iFront + 1;
119     }
120     return iElem;
121 }
122
123 void fnDisplay(QUEUE_T myQ)
124 {
125     int i;
126     for(i=myQ.iFront; i<=myQ.iRear; i++)
127     {
128         printf("%d\t", myQ.Queue[i]);
129     }
130     printf("\n");
131 }
```

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
```

```
4.Exit
Enter your choice
3

Queue is Empty

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 1

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 2

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 3

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
1   2   3

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 4

```

```
Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 5

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
1   2   3   4   5

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Queue is Full

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
2

Deleted element is 1

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
2

Deleted element is 2

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
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

==========

```
#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");
                    }
```

```
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;
```

```
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

==========

```
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;
```

```
}QUEUE_T;


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

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

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


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

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

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

                break;

```

```
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

```
====================================
/*************************************
putta:Programs$ gcc 06CircQueue.c -Wall
putta:Programs$ ./a.out

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
2

Queue is Empty

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 1

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 2

```

```
Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 3

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
1   2   3

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 2

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
4
putta:Programs$ clear

putta:Programs$ gcc 06CircQueue.c -Wall
putta:Programs$ ./a.out

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
2

Queue is Empty

Queue Operations
====================
1.Qinsert
2.Qdelete
```

```
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 1

Queue Operations
=====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 2

Queue Operations
=====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 3

Queue Operations
=====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
1   2   3

Queue Operations
=====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 4

Queue Operations
=====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 5
```

```
Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Queue is Full

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
1   2   3   4   5

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
2

Deleted element is 1

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
2   3   4   5

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
2

Deleted element is 2

Queue Operations
====================
1.Qinsert
2.Qdelete
```

```
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
3   4   5

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
1

Enter an element : 6

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
3

Contents of the Queue is
3   4   5   6

Queue Operations
====================
1.Qinsert
2.Qdelete
3.Qdisplay
4.Exit
Enter your choice
4

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

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

```
#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)
        {
```

```
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
86  NODEPTR fnDeleteFront(NODEPTR first)
87  {
88      NODEPTR temp;
89      if(first == NULL)
90      {
91          printf("\nList is Empty cannot delete\n");
92          return first;
93      }
```

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

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

Listing 7.1: 07SinglyLinkedList.c

Output

```
==================================
/************************************
putta:Programs$ gcc 07SinglyLinkedList.c -Wall
putta:Programs$ ./a.out

1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
1

Enter a element
4

```

```
1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
1

Enter a element
3

1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
1

Enter a element
2

1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
4

List Contents
==================================
   2   3   4
==================================



1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
3

Enter a element
1

Enter the position
1

1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
```

```
4

List Contents
==================================
   1   2   3   4
==================================



1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
5

1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
4

List Contents
==================================
   4   3   2   1
==================================



1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
2

Element deleted is 4

1.Insert Front
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
4

List Contents
==================================
   3   2   1
==================================



1.Insert Front
```

```
2.Delete Front
3.Insert At Position
4.Display
5.Reverse
6.Exit
Enter your choice
6

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

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

```
#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

```
==================================
/*************************************
putta:Programs$ gcc 08OrderedSinglyLinkedList.c -Wall
putta:Programs$ ./a.out

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
```

```
Enter your choice
1

Enter a element
6

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
1

Enter a element
4

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
1

Enter a element
2

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
2

Enter a element
7

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
2

Enter a element
5

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
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
```

```
4.Union
5.Intersection
6.Exit
Enter your choice
2

Enter a element
4

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
3

List 1 Contents
=================================
   2   4   6
=================================

List 2 Contents
=================================
   2   3   4   5   7
=================================

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
5

List 1 Contents
=================================
   2   4   6
=================================

List 2 Contents
=================================
   2   3   4   5   7
=================================

Intersection
=================================
   2   4
=================================

1.Insert into List 1
2.Insert into List 2
3.Display
4.Union
5.Intersection
6.Exit
Enter your choice
4

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

```
#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);
```

```
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
```

```
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
```

```
Enter Element to be pushed onto Stack
2

STACK OPERATIONS
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter your iChoice
1

Enter Element to be pushed onto Stack
3

STACK OPERATIONS
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter your iChoice
1

Enter Element to be pushed onto Stack
4

STACK OPERATIONS
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter your iChoice
3

The contents of Stack are :

4
3
2
1

STACK OPERATIONS
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter your iChoice
2

Element deleted is 4

STACK OPERATIONS
1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter your iChoice
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

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

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
2

Queue is empty cannot delete

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
1

Enter Element to be inserted
1

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
1

Enter Element to be inserted
2

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
```

```
4.Exit

Enter your choice
1

Enter Element to be inserted
3

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
1

Enter Element to be inserted
4

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
3

The contents of Queue are :

1   2   3   4

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
2

Element deleted is 1

QUEUE OPERATIONS
====================
1.Insert Rear
2.Delete Front
3.Display
4.Exit

Enter your choice
3

The contents of Queue are :

2   3   4

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

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

Listing 9.5: 09cPolynomial.c

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

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

Listing 9.6: out9c.c

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

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

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

typedef struct node* NODEPTR;

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



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

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

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

```
93          printf("\nOut of Memory");
94          exit(0);
95      }
96      return x;
97  }
98
99  void fnFreeNode(NODEPTR x)
100 {
101     free(x);
102 }
103
104 NODEPTR fnInsertFront(NODEPTR head, int iItem)
105 {
106     NODEPTR temp,cur;
107     temp = fnGetNode();
108     temp = fnGetNode();
109     temp->info = iItem;
110
111     cur = head->rlink;
112
113     head->rlink = temp;
114     temp->llink = head;
115     temp->rlink = cur;
116     cur->llink = temp;
117
118     head->info += 1;
119
120     return head;
121
122 }
123
124 NODEPTR fnDeleteFirst(NODEPTR last)
125 {
126     NODEPTR second, first, head;
127
128     if(last->rlink == last)
129     {
130         printf("\nList is Empty");
131         return last;
132     }
133     head = last->rlink;
134     first = head->rlink;
135     second = first->rlink;
136
137     head->rlink = second;
138     second->llink = head;
139     fnFreeNode(first);
140     head->info -= 1;
141
142     return head;
143 }
144
145 NODEPTR fnInsertNext(NODEPTR head, int iItem)
146 {
147     NODEPTR temp,cur,next;
148
149     if(head->rlink == head)
150     {
151         printf("\nList is Empty\n");
152         return head;
153     }
154
```

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

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

```
279         printf("\nNode #%d does not exist\n", n);
280         return head;
281     }
282 
283     temp = mpos->info;
284     mpos->info = npos->info;
285     npos->info = temp;
286 
287     return head;
288 }
```

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

Enter your Choice
2

Enter the key value of the node
5

Enter the item to be inserted to the next of 5
6

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   6   8

There are 4 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
1

Enter the iItem to be inserted
9

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
9   4   5   6   8

There are 5 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
4

Enter the position of the element to be deleted
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
9   4   5   8

There are 4 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
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
```

```
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

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

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

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

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

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

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

```
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 11.1: 11DequeueDLL.c

Output

==========

```
1 /*************************************
2 putta:Programs$ gcc 11DequeueDLL.c
3 putta:Programs$ ./a.out
4
5 1.Insert Front
6 2.Insert Rear
7 3.Delete Front
8 4.Delete Rear
9 5.Display
10 6.Exit
11
12 Enter your Choice
13 5
14
15 List is empty
16
17 1.Insert Front
18 2.Insert Rear
19 3.Delete Front
20 4.Delete Rear
21 5.Display
22 6.Exit
23
24 Enter your Choice
25 3
26
27 List is Empty
28
29 1.Insert Front
30 2.Insert Rear
31 3.Delete Front
32 4.Delete Rear
33 5.Display
34 6.Exit
35
36 Enter your Choice
37 1
38
39 Enter the iItem to be inserted
40 3
41
42 1.Insert Front
43 2.Insert Rear
44 3.Delete Front
45 4.Delete Rear
46 5.Display
47 6.Exit
48
49 Enter your Choice
50 1
51
52 Enter the iItem to be inserted
53 2
54
55 1.Insert Front
```

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

Enter your Choice
2

Enter the iItem to be inserted
4

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

Enter your Choice
5
Contents of the List is
2   3   4

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

Enter your Choice
2

Enter the iItem to be inserted
5

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

Enter your Choice
2

Enter the iItem to be inserted
6

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

Enter your Choice
5
Contents of the List is
2   3   4   5   6

```

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

Enter your Choice
3

Element deleted is 2

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

Enter your Choice
5
Contents of the List is
3   4   5   6

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

Enter your Choice
4

Element deleted is 6

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

Enter your Choice
5
Contents of the List is
3   4   5

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

Enter your Choice
6

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

Listing 11.2: 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

```
#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");
```

```
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      {
```

```
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             struct node *temp = root->rbranch;
202             free(root);
203             return temp;
204         }
205         else if (root->rbranch == NULL)
206         {
207             struct node *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
bool isOperator(char c)
{
    if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
        return true;
    return false;
}

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

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

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

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

void push(NODEPTR st[], NODEPTR p, int *t)
{
```

```
101     *t = *t + 1;
102     st[*t] = p;
103 }
104
105 NODEPTR pop(NODEPTR st[], int *t)
106 {
107     NODEPTR temp;
108     temp = st[*t];
109     *t = *t - 1;
110     return temp;
111 }
112
113 float evalPost(NODEPTR root)
114 {
115     float fNum;
116     switch(root->value)
117     {
118         case  '+' : return (evalPost(root->lbranch) + evalPost(root->rbranch));
119         case  '-' : return (evalPost(root->lbranch) - evalPost(root->rbranch));
120         case  '*' : return (evalPost(root->lbranch) * evalPost(root->rbranch));
121         case  '/' : return (evalPost(root->lbranch) / evalPost(root->rbranch));
122         case  '^' : return (pow(evalPost(root->lbranch), evalPost(root->rbranch)))
    ;
123         default:    if(isalpha(root->value))
124                     {
125                         printf("\n%c = ",root->value);
126                         scanf("%f",&fNum);
127                         return(fNum);
128                     }
129                     else
130                         return(root->value - '0');
131     }
132 }
```

Listing 13.1: 13ExpressionTree.c

Output

```
==================================
/*************************************
putta:Programs$ gcc 13ExpressionTree.c -lm
putta:Programs$ ./a.out

Enter a postfix expression
45*35/+
infix expression is
4 * 5 + 3 / 5
Value of Postfix Expression is : 20.6

putta:Programs$ ./a.out

Enter a postfix expression
ab+cd/*
infix expression is
a + b * c / d
a = 1
b = 2
c = 3
d = 4
Value of Postfix Expression is : 2.25
*************************************/
```

Listing 13.2: out13.c