SIDDAGANGA INSTITUTE OF TECHNOLOGY

www.sit.ac.in

Data Structures Laboratory

LAB MANUAL

Prabodh C P
Asst Professor
Dept of CSE, SIT

Contents

Listings

Data Structures Laboratory

Instructions

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

References

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

Chapter 1

File Management

Question

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

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

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

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

C Code - Text I/O

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

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

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

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

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

```
90          printf("\nNo Records found\n");
91      fclose(fp);
92  }
93
94  void fnAddRecord()
95  {
96      FILE *fp;
97      employee_t emp;
98
99      printf("\nEnter Employee details\n");
100     printf("\nID : ");
101     scanf("%d",&emp.emp_id);      getchar();
102     printf("\nName : ");
103     fgets(emp.emp_name,25,stdin);
104     printf("\nDept : ");
105     fgets(emp.emp_dept,25,stdin);
106     printf("\nSalary : ");
107     scanf("%d",&emp.emp_salary);
108     printf("\nAge : ");
109     scanf("%d",&emp.emp_age);
110
111     fp = fopen("emp.dat", "a");
112     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);
113     fclose(fp);
114 }
115
116 void fnSearchEmpID(int id)
117 {
118     int iCount = 0;
119     employee_t ep;
120     FILE *fp;
121
122     fp = fopen("emp.dat", "r");
123     if(fp==NULL)
124     {
125         printf("\nFile does not exist\n");
126         return;
127     }
128     while(fscanf(fp,"%d%s%s%d%d",&ep.emp_id, ep.emp_name, ep.emp_dept, &ep.
    emp_salary, &ep.emp_age)!=EOF)
129     {
130         if(ep.emp_id == id)
131         {
132             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);
133             iCount++;
134         }
135     }
136     if(0 == iCount)
137         printf("\nNo Records found\n");
138     fclose(fp);
139 }
140
141 void fnSearchEmpSal(int sal)
142 {
143     int iCount = 0;
144     employee_t ep;
145     FILE *fp;
146
147     fp = fopen("emp.dat", "r");
148     if(fp==NULL)
```

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

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:
34                     fnAddRecord();
35                     break;
36
37             case 2:
38                     printf("\n Employee Details \n");
39                     fnDisplayAllRecords();
40                     break;
41
42             case 3:
43                     printf("\nEnter the emp_id that you want to search\n");
44                     scanf("%d",&id);
45                     fnSearchEmpID(id);
46                     break;
47
48             case 4:
```

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

```
110     //fgets(wEmp.emp_dept, 25, stdin);
111     printf("\nSalary : ");
112     scanf("%d",&wEmp.emp_salary);
113     printf("\nAge : ");
114     scanf("%d",&wEmp.emp_age);
115 
116     fp = fopen("bemp.dat", "ab");
117 
118     fwrite(&wEmp, sizeof(employee_t),1,fp);
119     //write(fp,&wEmp,sizeof(employee_t));
120 
121     fclose(fp);
122 }
123 
124 void fnSearchEmpID(int id)
125 {
126     int iCount = 0;
127     employee_t sEmp;
128     FILE *fp;
129 
130     fp = fopen("bemp.dat", "r");
131     if(fp==NULL)
132     {
133         printf("\nFile does not exist\n");
134         return;
135     }
136     while(fread(&sEmp, sizeof(employee_t),1,fp))
137     {
138         if(sEmp.emp_id == id)
139         {
140             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);
141             iCount++;
142         }
143         if(feof(fp))
144             break;
145     }
146 
147     if(0 == iCount)
148         printf("\nNo Records found\n");
149     fclose(fp);
150 }
151 
152 void fnSearchEmpSal(int sal)
153 {
154     int iCount = 0;
155     employee_t sEmp;
156     FILE *fp;
157 
158     fp = fopen("bemp.dat", "r");
159     if(fp==NULL)
160     {
161         printf("\nFile does not exist\n");
162         return;
163     }
164     while(fread(&sEmp, sizeof(employee_t),1,fp))
165     {
166         if(sEmp.emp_salary == sal)
167         {
168             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);
169             iCount++;
```

```
170         }
171     }
172     if(0 == iCount)
173         printf("\nNo Records found\n");
174     fclose(fp);
175 }
176 
177 void fnSearchEmpDept(char dept[])
178 {
179     int iCount = 0;
180     employee_t sEmp;
181     FILE *fp;
182 
183 
184     fp = fopen("bemp.dat", "r");
185     if(fp==NULL)
186     {
187         printf("\nFile does not exist\n");
188         return;
189     }
190     while(fread(&sEmp, sizeof(employee_t),1,fp))
191     {
192         if(!strcmp(sEmp.emp_dept, dept))
193         {
194             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);
195             iCount++;
196         }
197     }
198     if(0 == iCount)
199         printf("\nNo Records found\n");
200 }
201 
202 void fnSearchEmpAge(int age)
203 {
204     int iCount = 0;
205     employee_t sEmp;
206     FILE *fp;
207 
208     fp = fopen("bemp.dat", "r");
209     if(fp==NULL)
210     {
211         printf("\nFile does not exist\n");
212         return;
213     }
214     while(fread(&sEmp, sizeof(employee_t),1,fp))
215     {
216         if(sEmp.emp_age == age)
217         {
218             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);
219             iCount++;
220         }
221     }
222     if(0 == iCount)
223         printf("\nNo Records found\n");
224 }
```

Listing 1.2: 01EmployeeDBBinary.c

Output

Chapter 2

Stack Implementation

Question

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

C Code Array Implementation

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

#define MAX 5

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

C Code Structure Implementation

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  
5  #define MAX 5
6  
7  typedef struct{
8      int stkArray[MAX];
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 
22     STACK_TYPE myStack;
23     myStack.top = -1;
24 
25     int iElem, iChoice;
26 
27     for(;;)
28     {
29         printf("\nSTACK OPERATIONS\n");
30         printf("====================");
31         printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.PEEK\n5.EXIT\n");
32         printf("Enter your choice\n");
33         scanf("%d",&iChoice);
34         switch(iChoice)
35         {
36             case 1: fnPush(stkArray, &top);
37                     break;
38 
39             case 2: iElem = fnPop(stkArray, &top);
40                     if(iElem != -1)
41                         printf("\nPopped Element is %d\n", iElem);
42                     break;
```

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

```
104 {
105     int i;
106     if(fnStkEmpty(t))
107     {
108         printf("\nStack Empty\n");
109         return;
110     }
111     printf("\nStack Contents are: \n");
112     for(i = t ; i > -1; --i)
113     {
114         printf("\t%d\n", stk[i]);
115     }
116 }
117 
118 int fnPeek(int stk[], int t)
119 {
120     return stk[t];
121 }
```

Listing 2.2: 02Stack2Struct.c

Output

Chapter 3

Infix to Postfix Conversion

Question

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

C Code

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

#define STK_SIZE 10

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

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

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

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

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

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

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

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

}

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

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

Listing 3.1: 03ConvInfix.c

Output

Chapter 4

Evaluation of Prefix Expression

Question

Write a C program to evaluate the given prefix expression.

C Code

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

#define STK_SIZE 10

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

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

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

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

```
40              }
41              fnPush(iaStack, &top, iRes);
42          }
43
44      }
45      iRes = fnPop(iaStack, &top);
46      printf("\nValue of %s expression is %d\n", acExpr, iRes);
47      return 0;
48  }
49
50  void fnPush(int Stack[], int *t , int elem)
51  {
52      *t = *t + 1;
53      Stack[*t] = elem;
54
55  }
56
57  int fnPop(int Stack[], int *t)
58  {
59      int elem;
60      elem = Stack[*t];
61      *t = *t - 1;
62      return elem;
63  }
```

Listing 4.1: 04EvalPrefix.c

Output

Chapter 5

Linear Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

Listing 5.1: 05LinearQueue.c

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

Chapter 6

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

                break;

            case 4: exit(0);

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

                break;
        }
    }
    return 0;
}

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

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

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

    queue[*r] = iVal;
}

```

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

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

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

Listing 6.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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

Chapter 7

Circular Queue

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 - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

```
                    }
                    else
                    {
                        printf("\nQueue is Full\n");
                    }

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

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

                break;

            case 4: exit(0);

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

                break;
        }
    }
    return 0;
}

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

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

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

```
94      }
95      else
96          *r = (*r + 1)%QUEUE_SIZE;
97
98      queue[*r] = iVal;
99  }
100
101 int fnDeleteFront(int queue[], int *f, int *r)
102 {
103     int iElem;
104     iElem = queue[*f];
105
106     if(*f == *r)
107     {
108         *f = -1;
109         *r = -1;
110     }
111     else
112     {
113         *f = (*f + 1)%QUEUE_SIZE;
114     }
115     return iElem;
116 }
117
118 void fnDisplay(int queue[], int f, int r)
119 {
120     int i;
121     if(f<=r)
122     {
123         for(i=f; i<=r; i++)
124         {
125             printf("%d\t", queue[i]);
126         }
127         printf("\n");
128     }
129     else
130     {
131         for(i=f; i<=QUEUE_SIZE-1; i++)
132         {
133             printf("%d\t", queue[i]);
134         }
135         for(i=0; i<=r; i++)
136         {
137             printf("%d\t", queue[i]);
138         }
139         printf("\n");
140     }
141 }
```

Listing 7.1: 06CircQueue.c

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
```

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


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

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

    myQueue.iFront = -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");
                    }
```

```
72
73                  break;
74
75              case 4: exit(0);
76
77              default: printf("\nInvalid choice\n");
78
79                  break;
80          }
81      }
82      return 0;
83  }
84
85  bool fnQueueFull(QUEUE_T myQ)
86  {
87      if((myQ.iRear+1) % QUEUE_SIZE == myQ.iFront)
88          return true;
89      else
90          return false;
91  }
92
93  bool fnQueueEmpty(QUEUE_T myQ)
94  {
95      if(myQ.iFront == -1)
96          return true;
97      else
98          return false;
99  }
100
101 void fnInsertRear(QUEUE_T *myQ, int iVal)
102 {
103     if(myQ->iRear == -1)
104     {
105         (myQ->iRear)++;
106         (myQ->iFront)++;
107     }
108     else
109         myQ->iRear = (myQ->iRear + 1) % QUEUE_SIZE;
110
111     myQ->Queue[myQ->iRear] = iVal;
112 }
113
114 int fnDeleteFront(QUEUE_T *myQ)
115 {
116     int iElem;
117     iElem = myQ->Queue[myQ->iFront];
118
119     if(myQ->iFront == myQ->iRear)
120     {
121         myQ->iFront = myQ->iRear = -1;
122     }
123     else
124     {
125         myQ->iFront = (myQ->iFront + 1)%QUEUE_SIZE;
126     }
127     return iElem;
128 }
129
130 void fnDisplay(QUEUE_T myQ)
131 {
132     int i;
133     if(myQ.iFront<=myQ.iRear)
```

```
134     {
135         for(i=myQ.iFront; i<=myQ.iRear; i++)
136         {
137             printf("%d\t", myQ.Queue[i]);
138         }
139         printf("\n");
140     }
141     else
142     {
143         for(i=myQ.iFront; i<QUEUE_SIZE; i++)
144         {
145             printf("%d\t", myQ.Queue[i]);
146         }
147         for(i=0; i<=myQ.iRear; i++)
148         {
149             printf("%d\t", myQ.Queue[i]);
150         }
151         printf("\n");
152 
153     }
154 }
```

Listing 7.2: 06StructCircularQueue.c

Output

Chapter 8

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

                break;

            case 4: exit(0);

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

                break;
        }
    }
    return 0;
}

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

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

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

    queue[*r] = iVal;
}

```

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

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

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

Listing 8.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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 8.2: 06StructCircularQueue.c

Output

Chapter 9

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

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

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

Listing 9.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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 9.2: 06StructCircularQueue.c

Output

Chapter 10

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

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

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

Listing 10.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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 10.2: 06StructCircularQueue.c

Output

Chapter 11

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

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

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

Listing 11.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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 11.2: 06StructCircularQueue.c

Output

Chapter 12

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

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

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

Listing 12.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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 12.2: 06StructCircularQueue.c

Output

Chapter 13

Circular Queue

Question

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

C Code - Array Representation

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

#define QUEUE_SIZE 5

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

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

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

                break;
```

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

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

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

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

Listing 13.1: 06CircQueue.c

C Code - Structure Representation

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


#define QUEUE_SIZE 5
#define NAME_LENGTH 30

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


void fnInsertRear(QUEUE_T*, int);
```

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

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

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


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

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

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

                break;

            case 4: exit(0);

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

```

```
                break;
            }
        }
        return 0;
}

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

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

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

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

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

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

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

```
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 13.2: 06StructCircularQueue.c

Output