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

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 fnInsertPosition(int ,int ,NODEPTR);
void fnDisplay(NODEPTR first);
NODEPTR fnReverse(NODEPTR);

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

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

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

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

            case 4: fnDisplay(first);
                    break;

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

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

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

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

}

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

```

```
94      temp = first;
95      printf("\nElement deleted is %d\n", temp->info);
96      fnFreeNode(temp);
97      first = first->link;
98      return first;
99  }
100
101 NODEPTR fnInsertPosition(int elem,int pos,NODEPTR first)
102 {
103     NODEPTR temp,prev,cur;
104     int count;
105
106     temp = fnGetNode();
107     temp->info = elem;
108     temp->link = NULL;
109
110     if(first == NULL && pos == 1)
111         return temp;
112
113     if(first == NULL)
114     {
115         printf("\nInvalid Position");
116         return first;
117     }
118
119     if(pos == 1)
120     {
121         temp->link = first;
122         return temp;
123     }
124
125
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 void fnDisplay(NODEPTR first)
149 {
150     NODEPTR temp;
151
152     if(first == NULL)
153     {
154         printf("\nList is Empty\n");
155         return;
```

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

Listing 7.1: 07SinglyLinkedList.c

Output

Chapter 8

Ordered Linked List

Question

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

a insert into list1.

b insert into list2.

c to perform UNION(list1,list2)

d to perform INTERSECTION(list1,list2)

e display the contents of all three lists.

C Code

```
#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;
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

Chapter 9

Singly Linked List Applications

9.1 Stack using SLL

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

C Code

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

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

typedef struct node* NODEPTR;

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

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


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

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

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

```
40
41              case 3: fnDisplay(first);
42                  break;
43
44              case 4: return;
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

9.2 Queue using SLL

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

C Code

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

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

typedef struct node* NODEPTR;


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

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

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


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

    temp = fnGetNode();

    temp->Info = iElem;

    temp->link = NULL;

    if(first == NULL)
        return temp;

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

    cur->link = temp;

    return first;
}
```

```
55
56 NODEPTR fnDelFront(NODEPTR first)
57 {
58     NODEPTR temp;
59     if(first == NULL)
60     {
61         printf("\nQueue is empty cannot delete\n");
62         return first;
63     }
64     temp = first;
65
66     first = first->link;
67
68     printf("\nElement deleted is %d \n",temp->Info);
69     fnFreeNode(temp);
70
71     return first;
72
73 }
74
75
76
77 void fnDisplay(NODEPTR first)
78 {
79     NODEPTR curr;
80     if(first == NULL)
81     {
82         printf("\nQueue is empty\n");
83         return;
84     }
85
86     printf("\nThe contents of Queue are :\n");
87     curr = first;
88     while(curr != NULL)
89     {
90         printf("\n%d",curr->Info);
91         curr = curr->link;
92     }
93     printf("\n");
94 }
95
96
97 main()
98 {
99     NODEPTR first = NULL;
100    int iChoice,iElem;
101
102    for(;;)
103    {
104        printf("\nQUEUE OPERATIONS\n");
105        printf("====================");
106        printf("\n1.Insert Rear\n2.Delete Front\n3.Display\n4.Exit\n");
107        printf("\nEnter your choice\n");
108        scanf("%d",&iChoice);
109
110        switch(ch)
111        {
112            case 1: printf("\nEnter Element to be inserted\n");
113                scanf("%d",&iElem);
114                first = fnIns_Rear(iElem,first);
115                break;
116
```

```
117             case 2: first = fnDelFront(first);
118                 break;
119
120             case 3: fnDisplay(first);
121                 break;
122
123             case 4: return;
124         }
125     }
126
127 }
128
129
130 /*CPP*/
```

Listing 9.2: 09bQueueLL.c

Output

9.3 Polynomial Addition

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

C Code

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

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

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

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

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

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

```

```
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 9.3: 09cPolynomial.c

Output

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

Enter the degree of polynomial 1
4

Enter the coefficients
3 4 5 6 7

Enter the degree of polynomial 2
3

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

Enter the value of x
2

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

Chapter 10

Doubly Linked Lists with Header Node

Question

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

a to insert a node next to a node whose information field specified.

b to delete first node if pointer to the last node is given.

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

d to display the contents of the list.

C Code - Array Representation

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

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

typedef struct node* NODEPTR;

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



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

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

```
    head->info = 0;

    for(;;)

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

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

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

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

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

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

            case 5: fnDisplay(head);
                    break;

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

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

NODEPTR fnGetNode(void)
{
    NODEPTR x;
    x = ( NODEPTR ) malloc (sizeof(struct node));
    if(x == NULL)
    {
```

```
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

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

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 - Array Representation

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

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

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

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

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

```
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->lchild = NULL;
116     temp->rchild = 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->lchild: cur->rchild;
136     }
137
138     if(iItem < prev->info)
139         prev->lchild = temp;
140     else
141         prev->rchild = 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->lchild);
153         fnPreOrder(root->rchild);
154     }
155 }
156
157 void fnInOrder(NODEPTR root)
158 {
159     if(root != NULL)
```

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

```
222         root->rchild = fnDeleteNode(root->rchild, temp->info);
223     }
224     return root;
225 }
226 
227 NODEPTR fnMinValueNode(NODEPTR node)
228 {
229     NODEPTR current = node;
230 
231     /* loop down to find the leftmost leaf */
232     while (current->lchild != NULL)
233         current = current->lchild;
234 
235     return current;
236 }
```

Listing 12.1: 12BinarySearchTree.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