



SIDDAGANGA INSTITUTE OF TECHNOLOGY
www.sit.ac.in

Data Structures Laboratory

LAB MANUAL

Prabodh C P
Asst Professor
Dept of CSE, SIT



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Contents

1	File Management	5
1.1	C Code - Text I/O	5
1.2	C Code - Binary I/O	9
2	Stack Implementation	16
2.1	C Code - Array Representation	16
2.2	C Code - Structure Representation	18
3	Infix to Postfix Conversion	24
4	Evaluation of Prefix Expression	27
5	Linear Queue	29
5.1	C Code - Array Representation	29
5.2	C Code - Structure Representation	31
6	Circular Queue	34
6.1	C Code - Array Representation	34
6.2	C Code - Structure Representation	36
7	Singly Linked List	40
8	Ordered Linked List	44
9	Singly Linked List Applications	48
9.1	Stack using SLL	48
9.2	Queue using SLL	51
9.3	Polynomial Addition	54
10	Doubly Linked Lists with Header Node	57
11	Double Ended Queue using Doubly Linked List	63
12	Binary Search Tree	67
13	Expression Tree	72

Listings

1.1	01EmployeeDB.c	5
1.2	01EmployeeDBBinary.c	9
1.3	out1.c	13
2.1	02Stack.c	16
2.2	02Stack2Struct.c	18
2.3	out2.c	20
3.1	03ConvInfix.c	24
3.2	out3.c	25
4.1	04EvalPrefix.c	27
5.1	05LinearQueue.c	29
5.2	05StructLinearQueue.c	31
6.1	06CircQueue.c	34
6.2	06StructCircularQueue.c	36
7.1	07SinglyLinkedList.c	40
8.1	08OrderedSinglyLinkedList.c	44
9.1	09aStackLL.c	48
9.2	09bQueueLL.c	51
9.3	09cPolynomial.c	54
10.1	10DoublyLinkedList.c	57
11.1	11DequeueDLL.c	63
12.1	12BinarySearchTree.c	67
13.1	13ExpressionTree.c	72

Data Structures Laboratory

Instructions

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

References

Part A: Behrouz A. Forouzan , Richard F. Gilberg , Computer Science: **A Structured programming Approach Using C** - Cengage Learning; 3rd edition

For writing flowcharts refer to **Appendix C** of the above book.

Chapter 1

File Management

Question

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

Write necessary functions to perform the following operations:

i) to display all the records in the file.

ii) to search for a specific record based on EMPLOYEE ID/SALARY/DEPARTMENT/AGE.

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

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

1.1 C Code - Text I/O

```
=====
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 /* FUNCTION PROTOTYPES */
12 void fnAddRecord(void);
13 void fnSearchEmpID(int);
14 void fnSearchEmpSal(int);
15 void fnSearchEmpDept(char[]);
16 void fnSearchEmpAge(int);
17 void fnDisplayAllRecords(void);
18
19 int main()
20 {
21     int id, sal, age, iChoice;
22     char dept[10];
23
24     for(;;)
25     {
26         printf("\n1.Add Record\n2.Display Records\n3.Search Employee by ID\n");
27         printf("4.Search Employee by Dept\n5.Search Employee by salary\n");
28         printf("6.Search Employee by Age\n7.Exit");
```

```

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

```

```

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

```

```

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

```



```

205         if(ep.emp_age == age)
206         {
207             printf("%d\t%s\t%s\t%d\t%d\n", ep.emp_id, ep.emp_name, ep.emp_dept, ep.
emp_salary, ep.emp_age);
208             iCount++;
209         }
210     }
211     if(0 == iCount)
212         printf("\nNo Records found\n");
213 }

```

Listing 1.1: 01EmployeeDB.c

1.2 C Code - Binary I/O

```

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

```

```

46         scanf("%s", dept);
47         fnSearchEmpDept(dept);
48         break;
49
50     case 5: printf("\nEnter the salary that you want to search\n");
51             scanf("%d", &sal);
52             fnSearchEmpSal(sal);
53             break;
54
55     case 6: printf("\nEnter the age that you want to search\n");
56             scanf("%d", &age);
57             fnSearchEmpAge(age);
58             break;
59
60     case 7: exit(0);
61 }
62 }
63 return 0;
64 }
65
66 void fnDisplayAllRecords()
67 {
68     int iCount = 0;
69     employee_t rEmp;
70     FILE *fp;
71
72     fp = fopen("bemp.dat", "rb");
73     if(fp==NULL)
74     {
75         printf("\nFile does not exist\n");
76         return;
77     }
78
79     while(fread(&rEmp, sizeof(employee_t), 1, fp))
80     {
81         printf("%6d\t%15s\t%8s\t%8d\t%4d\n", rEmp.emp_id, rEmp.emp_name, rEmp.
emp_dept, rEmp.emp_salary, rEmp.emp_age);
82         iCount++;
83         if(feof(fp))
84             break;
85     }
86
87     if(0 == iCount)
88         printf("\nNo Records found\n");
89     fclose(fp);
90 }
91
92 void fnAddRecord()
93 {
94     FILE *fp;
95     employee_t wEmp;
96
97     printf("\nEnter Employee details\n");
98     printf("\nID : ");
99     scanf("%d", &wEmp.emp_id);          getchar();
100    printf("\nName : ");
101    gets(wEmp.emp_name);
102    //fgets(wEmp.emp_name, 25, stdin);
103    printf("\nDept : ");
104    gets(wEmp.emp_dept);
105    //fgets(wEmp.emp_dept, 25, stdin);
106    printf("\nSalary : ");

```

```

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

```

```

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

```

Listing 1.2: 01EmployeeDBBinary.c

Output

```
=====
```

```

1  /*****
2  putta:Programs$ gcc 01EmployeeDB.c
3  putta:Programs$ ./a.out
4
5  1.Add Record
6  2.Display Records
7  3.Search Employee by ID
8  4.Search Employee by Dept
9  5.Search Employee by salary
10 6.Search Employee by Age
11 7.Exit
12 Enter your choice : 1
13
14 Enter Employee details
15
16 ID : 123
17 Name : Raju
18 Dept : CSE
19 Salary : 24000
20 Age : 26
21
22 1.Add Record
23 2.Display Records
24 3.Search Employee by ID
25 4.Search Employee by Dept
26 5.Search Employee by salary
27 6.Search Employee by Age
28 7.Exit
29 Enter your choice : 2
30
31 Employee Details
32
33 ID Name      Dept      Salary Age
34 123 Raju     CSE 24000  26
35
36 1.Add Record
37 2.Display Records
38 3.Search Employee by ID
39 4.Search Employee by Dept
40 5.Search Employee by salary
41 6.Search Employee by Age
42 7.Exit
43 Enter your choice : 1
44
45 Enter Employee details
46
47 ID : 124
48 Name : Susy
49 Dept : ISE
50 Salary : 26000
51 Age : 25
52
53 1.Add Record
54 2.Display Records
55 3.Search Employee by ID
56 4.Search Employee by Dept
57 5.Search Employee by salary
58 6.Search Employee by Age
59 7.Exit
60 Enter your choice : 1
61
62 Enter Employee details

```

```

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

```

```

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

```

Listing 1.3: out1.c

Chapter 2

Stack Implementation

Question

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

2.1 C Code - Array Representation

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



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

```

100     {
101         printf("\nStack Empty\n");
102         return;
103     }
104     printf("\nStack Contents are: \n");
105     for(i = t ; i > -1; --i)
106     {
107         printf("\t%d\n", stk[i]);
108     }
109 }
110
111 int fnPeek(int stk[], int t)
112 {
113     return stk[t];
114 }

```

Listing 2.1: 02Stack.c

2.2 C Code - Structure Representation

```

=====
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  #define 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         }
53         else
54             printf("\nEmpty Stack\n");
55         break;
56
57     case 5: exit(1);
58
59     default: printf("\nWrong choice\n");
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

```

=====
1  /*****
2  putta:Programs$ gcc 02Stack.c
3  putta:Programs$ ./a.out
4
5  STACK OPERATIONS
6  =====
7  1.PUSH
8  2.POP
9  3.DISPLAY
10 4.PEEK
11 5.EXIT
12 Enter your choice
13 2
14
15 Stack Underflow
16
17 STACK OPERATIONS
18 =====
19 1.PUSH
20 2.POP
21 3.DISPLAY
22 4.PEEK
23 5.EXIT
24 Enter your choice
25 3
26
27 Stack Empty
28
29 STACK OPERATIONS
30 =====
31 1.PUSH
32 2.POP
33 3.DISPLAY
34 4.PEEK
35 5.EXIT

```

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

```
98 2.POP
99 3.DISPLAY
100 4.PEEK
101 5.EXIT
102 Enter your choice
103 2
104
105 Popped Element is 3
106
107 STACK OPERATIONS
108 =====
109 1.PUSH
110 2.POP
111 3.DISPLAY
112 4.PEEK
113 5.EXIT
114 Enter your choice
115 4
116
117 Element at the top of the stack is 2
118
119 STACK OPERATIONS
120 =====
121 1.PUSH
122 2.POP
123 3.DISPLAY
124 4.PEEK
125 5.EXIT
126 Enter your choice
127 1
128
129 Enter element to be pushed onto the stack
130 3
131
132 STACK OPERATIONS
133 =====
134 1.PUSH
135 2.POP
136 3.DISPLAY
137 4.PEEK
138 5.EXIT
139 Enter your choice
140 1
141
142 Enter element to be pushed onto the stack
143 4
144
145 STACK OPERATIONS
146 =====
147 1.PUSH
148 2.POP
149 3.DISPLAY
150 4.PEEK
151 5.EXIT
152 Enter your choice
153 3
154
155 Stack Contents are:
156     4
157     3
158     2
159     1
```

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

Listing 2.3: out2.c

Chapter 3

Infix to Postfix Conversion

Question

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

C Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define STK_SIZE 10
6
7  void fnPush(char [], int*, char);
8  char fnPop(char [], int*);
9  int fnPreced(char);
10
11 int main()
12 {
13     int i, j=0;
14     char acExpr[50], acStack[50], acPost[50], cSymb;
15     int top = -1;
16
17     printf("\nEnter a valid infix expression\n");
18     scanf("%s", acExpr);
19
20     fnPush(acStack, &top, '#');
21     for(i=0; acExpr[i]!='\0'; ++i)
22     {
23         cSymb = acExpr[i];
24         if(isdigit(cSymb))
25         {
26             fnPush(acStack, &top, cSymb);
27         }
28         else if(cSymb == '(')
29         {
30             fnPush(acStack, &top, cSymb);
31         }
32         else if(cSymb == ')')
33         {
34             while(acStack[top] != '(')
35             {
36                 acPost[j++] = fnPop(acStack, &top);
37             }
38             fnPop(acStack, &top);
39         }
40     }
```



```

40     else
41     {
42         while(fnPrecd(acStack[top]) >= fnPrecd(cSymb))
43         {
44             if(cSymb == '^' && acStack[top] == '^')
45                 break;
46             acPost[j++] = fnPop(acStack, &top);
47         }
48         fnPush(acStack, &top, cSymb);
49     }
50
51 }
52 while(acStack[top] != '#')
53 {
54     acPost[j++] = fnPop(acStack, &top);
55 }
56 acPost[j] = '\0';
57
58 printf("\nInfix Expression is %s\n", acExpr);
59 printf("\nPostfix Expression is %s\n", acPost);
60 return 0;
61 }
62
63 void fnPush(char Stack[], int *t , char elem)
64 {
65     *t = *t + 1;
66     Stack[*t] = elem;
67 }
68
69
70 char fnPop(char Stack[], int *t)
71 {
72     char elem;
73     elem = Stack[*t];
74     *t = *t - 1;
75     return elem;
76 }
77
78 int fnPrecd(char ch)
79 {
80     switch(ch)
81     {
82         case '#' : return -1;
83         case '(' : return 0;
84         case '+' :
85         case '-' : return 1;
86         case '*' :
87         case '/' : return 2;
88         case '^' : return 3;
89     }
90 }

```

Listing 3.1: 03ConvInfix.c

Output

```

1  /*****
2  putta:Programs$ gcc 03ConvInfix.c
3  putta:Programs$ ./a.out
4  Enter a valid infix expression
5  (a^b^c)+(d/(e-f))
6

```

```
7 Infix Expression is (a^b^c)+(d/(e-f))
8
9 Postfix Expression is abc^^def-/
10
11 putta:Programs$ ./a.out
12 Enter a valid infix expression
13 (a*(b-c)/(a+b*c))
14
15 Infix Expression is (a*(b-c)/(a+b*c))
16
17 Postfix Expression is abc-*abc*+/
18
19 *****/
```

Listing 3.2: out3.c

Chapter 4

Evaluation of Prefix Expression

Question

Write a C program to evaluate the given prefix expression.

C Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define STK_SIZE 10
6
7  void fnPush(int [], int*, int);
8  int fnPop(int [], int*);
9
10 int main()
11 {
12     int iaStack[50], i, iOp1, iOp2, iRes;
13     char acExpr[50], cSymb;
14     int top = -1;
15
16     printf("\nEnter a valid prefix expression\n");
17     scanf("%s", acExpr);
18
19     for(i=strlen(acExpr)-1; i>=0; i--)
20     {
21         cSymb = acExpr[i];
22         if(isdigit(cSymb))
23         {
24             fnPush(iaStack, &top, cSymb-'0');
25         }
26         else
27         {
28             iOp1 = fnPop(iaStack, &top);
29             iOp2 = fnPop(iaStack, &top);
30             switch(cSymb)
31             {
32                 case '+': iRes = iOp1 + iOp2;
33                             break;
34                 case '-': iRes = iOp1 - iOp2;
35                             break;
36                 case '*': iRes = iOp1 * iOp2;
37                             break;
38                 case '/': iRes = iOp1 / iOp2;
39                             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.

5.1 C Code - Array Representation

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

```

```

38         break;
39     case 2: if(!fnQueueEmpty(myQueue, iFront, iRear))
40     {
41         iElem = fnDeleteFront(myQueue, &iFront, &iRear);
42         printf("\nDeleted element is %d\n", iElem);
43     }
44     else
45     {
46         printf("\nQueue is Empty\n");
47     }
48
49     break;
50     case 3: if(!fnQueueEmpty(myQueue, iFront, iRear))
51     {
52         printf("\nContents of the Queue is \n");
53         fnDisplay(myQueue, iFront, iRear);
54     }
55     else
56     {
57         printf("\nQueue is Empty\n");
58     }
59
60     break;
61
62     case 4: exit(0);
63
64     default: printf("\nInvalid choice\n");
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

5.2 C Code - Structure Representation

```

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

```

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



```
98 }
99
100 void fnInsertRear(QQUEUE_T *myQ, int iVal)
101 {
102     (myQ->iRear)++;
103     myQ->Queue[myQ->iRear] = iVal;
104 }
105
106 int fnDeleteFront(QQUEUE_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(QQUEUE_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.

6.1 C Code - Array Representation

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

```

```

38         break;
39     case 2: if(!fnQueueEmpty(myQueue, iFront, iRear))
40     {
41         iElem = fnDeleteFront(myQueue, &iFront, &iRear);
42         printf("\nDeleted element is %d\n", iElem);
43     }
44     else
45     {
46         printf("\nQueue is Empty\n");
47     }
48
49     break;
50     case 3: if(!fnQueueEmpty(myQueue, iFront, iRear))
51     {
52         printf("\nContents of the Queue is \n");
53         fnDisplay(myQueue, iFront, iRear);
54     }
55     else
56     {
57         printf("\nQueue is Empty\n");
58     }
59
60     break;
61
62     case 4: exit(0);
63
64     default: printf("\nInvalid choice\n");
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
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 6.1: 06CircQueue.c

6.2 C Code - Structure Representation

```

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

```

```
14
15
16 void fnInsertRear(QQUEUE_T*, int);
17 int fnDeleteFront(QQUEUE_T*);
18 void fnDisplay(QQUEUE_T);
19 bool fnQueueFull(QQUEUE_T);
20 bool fnQueueEmpty(QQUEUE_T);
21
22 int main()
23 {
24     QQUEUE_T myQueue;
25     int iElem, iChoice;
26
27     myQueue.iFront = -1;
28     myQueue.iRear = -1;
29
30
31     for(;;)
32     {
33         printf("\nQueue Operations\n");
34         printf("=====");
35         printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
36         printf("Enter your choice\n");
37         scanf("%d",&iChoice);
38         switch(iChoice)
39         {
40             case 1: if(!fnQueueFull(myQueue))
41                 {
42                     printf("\nEnter an element : ");
43                     scanf("%d", &iElem);
44                     fnInsertRear(&myQueue, iElem);
45                 }
46             else
47                 {
48                     printf("\nQueue is Full\n");
49                 }
50
51             break;
52             case 2: if(!fnQueueEmpty(myQueue))
53                 {
54                     iElem = fnDeleteFront(&myQueue);
55                     printf("\nDeleted element is %d\n", iElem);
56                 }
57             else
58                 {
59                     printf("\nQueue is Empty\n");
60                 }
61
62             break;
63             case 3: if(!fnQueueEmpty(myQueue))
64                 {
65                     printf("\nContents of the Queue is \n");
66                     fnDisplay(myQueue);
67                 }
68             else
69                 {
70                     printf("\nQueue is Empty\n");
71                 }
72
73             break;
74
75             case 4: exit(0);
```

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

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

Listing 6.2: 06StructCircularQueue.c

Output

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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct node
4 {
5     int info;
6     struct node *link;
7 };
8
9 typedef struct node* NODEPTR;
10
11 NODEPTR fnGetNode(void);
12 void fnFreeNode(NODEPTR x);
13 NODEPTR fnInsertFront(int ,NODEPTR);
14 NODEPTR fnDeleteFront(NODEPTR);
15 NODEPTR fnInsertPosition(int ,int ,NODEPTR);
16 void fnDisplay(NODEPTR first);
17 NODEPTR fnReverse(NODEPTR);
18
19 int main()
20 {
21     NODEPTR first = NULL;
22     int iElem, iChoice, iPos;
23     for(;;)
24     {
25         printf("\n1.Insert Front\n2.Delete Front\n3.Insert At Position");
26         printf("\n4.Display\n5.Reverse\n6.Exit\n");
27         printf("Enter your choice\n");
28         scanf("%d",&iChoice);
29         switch(iChoice)
30         {
31             case 1: printf("\nEnter a element\n");
```



```

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

```

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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct node
4 {
5     int info;
6     struct node *link;
7 };
8
9 typedef struct node* NODEPTR;
10
11 NODEPTR fnGetNode(void);
12 NODEPTR fnInsertOrder(int ,NODEPTR);
13 NODEPTR fnInsertRear(int ,NODEPTR);
14 NODEPTR fnUnion(NODEPTR ,NODEPTR);
15 NODEPTR fnIntersection(NODEPTR ,NODEPTR);
16 void fnDisplay(NODEPTR first);
17
18 int main()
19 {
20     NODEPTR list1 , list2, list3, list4;
21     list1 = list2 = list3 = list4 = NULL;
22     int iElem, iChoice;
23     for(;;)
24     {
25         printf("\n1.Insert into List 1\n2.Insert into List 2\n3.Display");
26         printf("\n4.Union\n5.Intersection\n6.Exit\n");
27         printf("Enter your choice\n");
28         scanf("%d",&iChoice);
29         switch(iChoice)
30         {
31             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     while(t1 != NULL)
124     {
125         t3 = fnInsertRear(t1->info, t3);
126         t1 = t1->link;
127     }
128     while(t2 != NULL)
129     {
130         t3 = fnInsertRear(t2->info, t3);
131         t2 = t2->link;
132     }
133     return t3;
134 }
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

```
1  #include<stdio.h>                                /*CPP*/
2  #include<stdlib.h>
3
4  struct node
5  {
6      int Info;
7      struct node *link;
8  };
9
10 typedef struct node* NODEPTR;
11
12 NODEPTR fnGetNode(void) ;
13 void fnFreeNode(NODEPTR x) ;
14 NODEPTR fnPush(int ,NODEPTR) ;
15 NODEPTR fnPop(NODEPTR) ;
16 void fnDisplay(NODEPTR first);
17
18 int main(void)
19 {
20     NODEPTR first = NULL;
21     int iChoice,iElem;
22
23
24     for(;;)
25     {
26         printf("\nSTACK OPERATIONS");
27         printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
28         printf("\nEnter your iChoice\n");
29         scanf("%d",&iChoice);
30
31         switch(iChoice)
32         {
33             case 1: printf("\nEnter Element to be pushed onto Stack\n");
34                     scanf("%d",&iElem);
35                     first = fnPush(iElem,first);
36                     break;
37
38             case 2: first = fnPop(first);
```



```
39         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

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int Info;
7      struct node *link;
8  };
9
10 typedef struct node* NODEPTR;
11
12
13 NODEPTR fnGetNode()
14 {
15     NODEPTR newborn;
16     newborn = (NODEPTR)malloc(sizeof(struct node));
17
18     if(newborn == NULL)
19     {
20         printf("\nMemory Overflow");
21         exit(0);
22     }
23     return newborn;
24 }
25
26 void fnFreeNode(NODEPTR x)
27 {
28     free(x);
29 }
30
31
32 NODEPTR fnIns_Rear(int iElem, NODEPTR first)
33 {
34     NODEPTR temp, cur;
35
36     temp = fnGetNode();
37
38     temp->Info = iElem;
39
40     temp->link = NULL;
41
42     if(first == NULL)
43         return temp;
44
45     cur = first;
46     while(cur->link != NULL)
47     {
48         cur = cur->link;
49     }
50
51     cur->link = temp;
52
53     return first;

```

```

54 }
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

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  struct polynomial{
6      int coeff;
7      int exponent;
8      struct polynomial *link;
9  };
10 typedef struct polynomial *NODEPTR;
11
12 NODEPTR fnGetNode(void);
13 NODEPTR fnInsertRear(int, int, NODEPTR);
14 void fnDisplay(NODEPTR first);
15 NODEPTR fnAddPoly(NODEPTR, NODEPTR);
16 int evalPoly(NODEPTR, int);
17
18 int main()
19 {
20     NODEPTR poly1, poly2, poly3;
21     int i, iX, iRes, iDegree, iaCoeff[10];
22     poly1 = poly2 = poly3 = NULL;
23
24     printf("\nEnter the degree of polynomial 1\n");
25     scanf("%d", &iDegree);
26     printf("\nEnter the coefficients\n");
27     for(i=iDegree; i>=0; i--)
28     {
29         scanf("%d", &iaCoeff[i]);
30         poly1 = fnInsertRear(iaCoeff[i], i, poly1);
31     }
32     printf("\nEnter the degree of polynomial 2\n");
33     scanf("%d", &iDegree);
34     printf("\nEnter the coefficients\n");
35     for(i=iDegree; i>=0; i--)
36     {
37         scanf("%d", &iaCoeff[i]);
38         poly2 = fnInsertRear(iaCoeff[i], i, poly2);
39     }
40     poly3 = fnAddPoly(poly1, poly2);
41
42     printf("Polynomial 1   :\t");
43     fnDisplay(poly1);
44     printf("Polynomial 2   :\t");
45     fnDisplay(poly2);
46     printf("Polynomial Sum  :\t");
47     fnDisplay(poly3);
48     printf("\nEnter the value of x\n");
49     scanf("%d", &iX);
50     iRes = evalPoly(poly3, iX);
51     printf("\nValue of the polynomial sum for x = %d is %d\n", iX, iRes);
52     return 0;
53 }

```

```

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

```

```

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

```

Listing 9.3: 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

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int info;
7      struct node *llink;
8      struct node *rlink;
9  };
10
11 typedef struct node* NODEPTR;
12
13 NODEPTR fnSwapNodes(NODEPTR head, int m , int n);
14 void fnDisplay(NODEPTR head);
15 NODEPTR fnDelElemPos(NODEPTR head, int iPos);
16 NODEPTR fnInsertNext(NODEPTR head, int iItem);
17 NODEPTR fnDeleteFirst(NODEPTR last);
18 NODEPTR fnInsertFront(NODEPTR head, int iItem);
19 void fnFreeNode(NODEPTR x);
20 NODEPTR fnGetNode(void);
21
22
23
24 int main()
25 {
26     NODEPTR head, last;
27     int iChoice, iItem, iKey, iPos, iM, iN;
28
29     head = fnGetNode();
30     head->rlink = head;
31     head->llink = head;
```

```

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

```

```

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

```

```

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

```

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

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

Listing 10.1: 10DoublyLinkedList.c

Output

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

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

```

36
37     switch(iChoice)
38     {
39         case 1: printf("\nEnter the iItem to be inserted\n");
40                 scanf("%d",&iItem);
41                 head = fnInsertFront(head, iItem);
42                 break;
43
44         case 2: printf("\nEnter the iItem to be inserted\n");
45                 scanf("%d",&iItem);
46                 head = fnInsertRear(head, iItem);
47                 break;
48
49         case 3: head = fnDeleteFront(head);
50                 break;
51
52         case 4: head = fnDeleteRear(head);
53                 break;
54
55         case 5: fnDisplay(head);
56                 break;
57
58         case 6: exit(0);
59     }
60 }
61 return 0;
62 }
63
64 NODEPTR fnGetNode(void)
65 {
66     NODEPTR x;
67     x = ( NODEPTR ) malloc (sizeof(struct node));
68     if(x == NULL)
69     {
70         printf("\nOut of Memory");
71         exit(0);
72     }
73     return x;
74 }
75
76 void fnFreeNode(NODEPTR x)
77 {
78     free(x);
79 }
80
81 NODEPTR fnInsertFront(NODEPTR head, int iItem)
82 {
83     NODEPTR temp,cur;
84     temp = fnGetNode();
85     temp = fnGetNode();
86     temp->info = iItem;
87
88     cur = head->rlink;
89     head->rlink = temp;
90     temp->llink = head;
91     temp->rlink = cur;
92     cur->llink = temp;
93     return head;
94 }
95
96 NODEPTR fnInsertRear(NODEPTR head, int iItem)
97 {

```



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

```
159 }
```

Listing 11.1: 11DequeDLL.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

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  struct node
5  {
6      int info;
7      struct node *lchild;
8      struct node *rchild;
9  };
10 typedef struct node* NODEPTR;
11
12 /* FUNCTION PROTOTYPES */
13 NODEPTR fnGetNode(void);
14 void fnFreeNode(NODEPTR x);
15 NODEPTR fnInsertNode(int, NODEPTR);
16 void fnInOrder(NODEPTR);
17 void fnPreOrder(NODEPTR);
18 void fnPostOrder(NODEPTR);
19 NODEPTR fnDeleteNode(NODEPTR, int);
20 NODEPTR fnMinValueNode(NODEPTR);
21
22 int main()
23 {
24     NODEPTR root = NULL;
25     int iChoice,iItem;
26     for(;;)
27     {
28         printf("\n1.Insert a node\n2.Inorder traversal\n3.Preorder traversal");
29         printf("\n4.Postorder traversal\n5.Delete a node\n6.Exit\n");
30         printf("\nEnter your choice");
31         scanf("%d",&iChoice);
32
33         switch(iChoice)
34         {
35             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

Expression Tree

Question

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

C Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <string.h>
5  // An expression tree node
6  struct ExpTree
7  {
8      char value;
9      struct ExpTree *left, *right;
10 };
11 typedef struct ExpTree* NODEPTR;
12
13 bool isOperator(char c);
14 void inorder(NODEPTR t);
15 NODEPTR newNode(int v);
16 NODEPTR constructTree(char postfix[]);
17 void push(NODEPTR[], NODEPTR, int*);
18 NODEPTR pop(NODEPTR[], int*);
19 NODEPTR peep(NODEPTR[], int*);
20
21 int main()
22 {
23     char postfix[30];
24     printf("\nEnter a postfix expression\n");
25     scanf("%s", postfix);
26     NODEPTR et = constructTree(postfix);
27     printf("infix expression is \n");
28     inorder(et);
29     printf("\n");
30     return 0;
31 }
32 // A utility function to check if 'c' is an operator
33 bool isOperator(char c)
34 {
35     if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
36         return true;
37     return false;
38 }
```



```

39
40 // Utility function to do inorder traversal
41 void inorder(NODEPTR t)
42 {
43     if(t)
44     {
45         inorder(t->left);
46         printf("%c ", t->value);
47         inorder(t->right);
48     }
49 }
50
51 NODEPTR newNode(int v)
52 {
53     NODEPTR temp = (NODEPTR) malloc(sizeof(struct ExpTree));
54     temp->left = temp->right = NULL;
55     temp->value = v;
56     return temp;
57 }
58
59 NODEPTR constructTree(char postfix[])
60 {
61     NODEPTR stack[100];
62     int i, top = -1;
63     NODEPTR t, t1, t2;
64
65     // Traverse through every character of input expression
66     for(i=0; i<strlen(postfix); i++)
67     {
68         // If operand, simply push into stack
69         if(!isOperator(postfix[i]))
70         {
71             t = newNode(postfix[i]);
72             push(stack, t, &top);
73         }
74         else // operator
75         {
76             t = newNode(postfix[i]);
77             // Pop two top nodes
78             t1 = peep(stack, &top); // Store top
79             pop(stack, &top);       // Remove top
80             t2 = peep(stack, &top);
81             pop(stack, &top);
82             // make them children
83             t->right = t1;
84             t->left = t2;
85             // Add this subexpression to stack
86             push(stack, t, &top);
87         }
88     }
89     // only element will be root of expression tree
90     t = peep(stack, &top);
91     pop(stack, &top);
92     return t;
93 }
94
95 void push(NODEPTR st[], NODEPTR p, int *t)
96 {
97     *t = *t + 1;
98     st[*t] = p;
99 }
100

```

```
101 NODEPTR pop(NODEPTR st[], int *t)
102 {
103     NODEPTR temp;
104     temp = st[*t];
105     *t = *t - 1;
106     return temp;
107 }
108
109 NODEPTR peep(NODEPTR st[], int *t)
110 {
111     return st[*t];
112 }
```

Listing 13.1: 13ExpressionTree.c

Output