```
1: //Queue using circular array implementation
2:
3: #include<stdio.h>
4:
5: #define SIZE 5
6: #define TRUE 1
7: #define FALSE 0
8:
9: typedef int BOOL;
10:
11: typedef int queue_entry;
12:
13: typedef struct queue
14: {
15:     queue_entry entry[SIZE];
16:     int front;
17:     int rear;
18:     int count;
19: }QUEUE;
20:
21: int main()
22: {
23:     QUEUE q;
24:     int size;
25:     queue_entry result;
26:     int d, choice = -1;
27:     void initialiseQueue(QUEUE *s);
28:     BOOL enqueue(int d, QUEUE *q);
29:     queue_entry dequeue(QUEUE *q);
30:     BOOL IsQueueEmpty(QUEUE q);
31:     BOOL IsQueueFull(QUEUE q);
32:  BOOL enqueue(int d, QUEUE *q);
33:     void clearQueue(QUEUE *q);
34:     queue_entry QueueFront(QUEUE q);
35:     void traverseQueue(QUEUE q);
36:
37:     initialiseQueue(&q);
38:
39:     while(choice)
40:     {
41:     printf("\nEnter the option");
42:     printf("\n1. Enqueue");
43:     printf("\n2. Dequeue");
44:     printf("\n3. Is Queue Empty");
45:     printf("\n4. Is Queue Full");
46:     printf("\n5: Queue Size");
47:     printf("\n6: ClearQueue");
48:     printf("\n7: TraverseQueue");
49:     printf("\n8: QueueFront");
50:     printf("\n0. Exit\n");
51:     scanf("%d", &choice);
52:         switch(choice)
53:         {
54:             case 1:
55:                 printf("Enter the data item you want to enqueue");
56:                 fflush(stdin);
57:                 scanf("%d", &d);
58:                 result = enqueue(d,&q);
59:                 if(result == 1)
60:                 {
```

```
61:
62:                     printf("Successfully enqueued");
63:                 }
64:                 else
65:                 {
66:                     printf("Queue is full, cannot enqueue");
67:                 }
68:                 break;
69:             case 2:
70:                 if(!IsQueueEmpty(q))
71:                 {
72:                     d = dequeue(&q);
73:                     printf("The dequeued item is %d", d);
74:                 }
75:                 else
76:                 {
77:
78:                     printf("Cannnot dequeue. Queue is empty");
79:                 }
80:                 break;
81:             case 3:
82:                 d = IsQueueEmpty(q);
83:                 if(d)
84:                 {
85:                     printf("Queue is empty");
86:                 }
87:                 else
88:                 {
89:                     printf("Queue is not empty");
90:                 }
91:                 break;
92:             case 4:
93:                 d = IsQueueFull(q);
94:                 if(d == 1)
95:                 {
96:                     printf("Queue is full");
97:                 }
98:                 else
99:                 {
100:                    printf("Queue is not full");
101:
102:                }
103:                break;
104:            case 5:
105:                size =  QueueSize(q);
106:                printf("\n %d", size);
107:                break;
108:
109:            case 6:
110:                clearQueue(&q);
111:                break;
112:            case 7:
113:                traverseQueue(q);
114:                break;
115:            case 8:
116:                d = QueueFront(q);
117:                printf("\n %d", d);
118:                break;
119:            default:
120:                printf("Program over");
        }
    }
```

```
121:     return(0);
122: }
123:
124: void initialiseQueue(QUEUE *q)
125: {
126:     q->front = 0;
127:     q->rear = -1;
128:     q->count = 0;
129: }
130:
131: BOOL enqueue(int d, QUEUE *q)
132: {
133:     if(!(IsQueueFull(q)))
134:     {
135:         q->count ++;
136:         q->rear = (q->rear + 1)%SIZE;
137:         q->entry[q->rear] = d;
138:
139:     }
140:     else
141:     {
142:         return(FALSE);
143:     }
144:     return(TRUE);
145: }
146:
147: int dequeue(QUEUE *q)
148: {
149:     queue_entry ans;
150:     if(!IsQueueEmpty(*q))
151:     {
152:         q->count --;
153:         ans = q->entry[q->front];
154:         q->front    = (q->front + 1)%SIZE;
155:
156:     }
157:     else
158:     {
159:         return(-1);
160:     }
161:     return(ans);
162: }
163:
164: BOOL IsQueueEmpty(QUEUE q)
165: {
166:     return(q.count == 0);
167: }
168:
169: int IsQueueFull(QUEUE q)
170: {
171:     return(q.count >= SIZE);
172: }
173:
174: int QueueSize(QUEUE q)
175: {
176:     return(q.count);
177: }
178:
179: void clearQueue(QUEUE *q)
180: {
```

```
181:     q->count = 0;
182:     q->front = q->rear + 1;
183: }
184:
185: void traverseQueue(QUEUE q)
186: {
187:     int i;
188:     printf("\n");
189:     for(i= q.front;i<=q.rear; i= (i+1)%SIZE)
190:     {
191:         printf("\t%d", q.entry[i]);
192:     }
193:
194: }
195:
196: queue_entry QueueFront(QUEUE q)
197: {
198:     return(q.front);
199: }
200:
201:
```