

# CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)



Government institute under Chandigarh (UT) Administration, affiliated to Punjab University,  
Chandigarh

Department of Computer Science & Engineering

**Semester: CSE 3<sup>rd</sup>**

**SUBJECT: Data Structures Practical (CS351)**

**Problem 5: Case Study of Stack and Queue**

**Submitted by:**

Bhavyam Dhand

(CO23316)

**Submitted to:**

Dr. R.B. Patel

(Professor)

**Date of Practical: 2-Sep**

**Date of Submission: 23-Sep**

## **INDEX**

S.No	Content	Page no.
I.	Objective	3
II.	Discussion & Modelling of Problem	4
III.	Implementation of Stacks	5
1.	From a Static Data Structure (Array)	
2.	From a Singly Linked Data Structure	6
3.	From a Doubly Linked Data Structure	7
	CODE	9
IV.	Implementation of Queue	17
1.	From a Static Data Structure (Array)	
2.	From a Singly Linked Data Structure	19
3.	From a Doubly Linked Data Structure	21
	CODE	24

# CODE

## 1. Array Implemented Stack:

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cmath>
5  #include <ctime>
6  #include <bitset>
7  using namespace std;
8  void LogFile(const string& event) {
9      ofstream BinFile("Machine_Code_Stack.txt",ios_base::app);
10     if (BinFile.is_open())
11     {
12         for (char c:event)
13         {
14             BinFile<<bitset<8>(c)<<" ";
15         }
16         BinFile<<endl;
17         BinFile.close();
18     }
19 }
20 const int SIZE=10;
21 bool IsFull(int size,int& Tos)
22 {
23     LogFile("Stack is Full");
24     return(Tos==size-1);
25 }
26 bool IsEmpty(int& Tos)
27 {
28     LogFile("Stack is Empty");
29     return(Tos== -1);
30 }
31 void Push(int Stack[],int size,int& Tos,int info)
32 {
33     // To check if Stack is full
34     LogFile("Pushed Value " +to_string(info)+" to Stack");
35     if (IsFull(size,Tos))
36     {
37         cout<<"\nStack Overflow!"<<endl;
38         return;
39     }
40     else
41         Stack[++Tos]=info;
42 }
43 int pop(int Stack[],int &Tos)
44 {
45     LogFile("Popped Value from Stack");
46     // To check if Stack is Empty
47     if (IsEmpty(Tos))
48     {
49         cout<<"\nStack Underflow!"<<endl;
50         return 0;
```

```

51     }
52     else
53         return Stack[Tos--];
54 }
55 void traverse(int Stack[], int BStack[], int size, int& Tos, int& BTos) {
56     if (IsEmpty(Tos)) {
57         cout << "\nStack is empty!" << endl;
58         return;
59     }
60
61     int info;
62     int tempTos = Tos; // Save the original Tos for restoring later
63
64     cout << "\nThe elements of Stack are:" << endl;
65
66     // Move elements from Stack to BStack and print them
67     while (!IsEmpty(Tos)) {
68         info = pop(Stack, Tos);
69         Push(BStack, size, BTos, info);
70         cout << info << endl;
71     }
72
73     // Restore elements from BStack back to Stack
74     while (!IsEmpty(BTos)) {
75         info = pop(BStack, BTos);
76         Push(Stack, size, Tos, info);
77     }
78
79     Tos = tempTos; // Restore original Tos
80     LogFile("traversing List");
81 }
82
83 int Peek(int Stack[],int& Tos)
84 {
85     LogFile("Peeked List");
86     if (IsEmpty(Tos))
87     {
88         cout<<"\nStack Underflow!"<<endl;
89         return 0;
90     }
91     return Stack[Tos];
92 }
93 int main()
94 {
95     LogFile("Open File");
96     int ToS=-1,BToS=-1;
97     int S[SIZE],BS[SIZE];
98     int z;
99     while (true){
100         cout<<"\n\nWelcome to Stack Manager:"<<endl;
101         cout<<"Select Your Commands:"<<endl;
102         cout<<"1. Push an element into a Stack"<<endl;
103         cout<<"2. Pop an element from the Stack"<<endl;
104         cout<<"3. Traverse a Stack"<<endl;
105         cout<<"4. Peek Your Stack"<<endl;
106         cout<<"5. Exit Program."<<endl;
107         cout<<"Enter choice:"; cin>>z;

```

```

108     switch (z)
109     {
110     case 1:
111         LogFile("Call Push Function");
112         {
113             int info;
114             cout<<"Enter info you want to push in Stack: "; cin>>info;
115             Push(S,SIZE,ToS,info);
116         }
117         break;
118     case 2:
119         LogFile("Call Pop Function");
120         pop(S,ToS);
121         break;
122     case 3:
123         LogFile("Call Traverse Function");
124         traverse(S,BS,SIZE,ToS,BToS);
125         break;
126     case 4:
127         LogFile("Call Peek Function");
128         cout<<Peek(S,ToS)<<endl<<endl;
129         break;
130     case 5:
131         LogFile("Close File");
132         return 0;
133     default:
134         LogFile("Force to close File");
135         cout<<"Entered Invalid Option."<<endl;
136         return 0;
137         break;
138     }
139 }

```

## 2. Singly Linked Implementation of Stack

```

1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cmath>
5  #include <ctime>
6  #include <bitset>
7  using namespace std;
8
9  void LogFile(const string& event) {
10     ofstream BinFile("Machine_Code_Stack.txt",ios_base::app);
11     if (BinFile.is_open())
12     {
13         for (char c:event)
14         {
15             BinFile<<bitset<8>(c)<<" ";
16         }
17         BinFile<<endl;
18         BinFile.close();
19     }

```

```

20
21 }
22 struct Node{
23     int x;
24     Node *next;
25
26 };
27 typedef Node ST;
28 void push(ST*&start, int& val)
29 {
30     ST*newNode=new ST;
31     newNode->x=val;
32     newNode->next=NULL;
33     //check if list is empty
34     if (start==NULL)
35         start=newNode;
36     else
37     {
38         newNode->next=start;
39         start=newNode;
40     }
41     cout<<"Item Pushed!"<<endl;
42     LogFile("Pushed value " + to_string(val));
43 }
44 int pop(ST*& start)
45 {
46     // check if list is empty
47     if (start==NULL)
48     {
49         cout<<"Underflow!"<<endl;
50         return 0;
51     }
52     else
53     {
54         ST* ptr = start;
55         int value = ptr->x;
56         start = start->next;
57         delete ptr; // Free the popped node
58         cout << "Item Popped!" << endl;
59         LogFile("Popped value " + to_string(value));
60         return value;
61     }
62     LogFile("Popped value ");
63 }
64 void traverse(ST*& Start)
65 {
66     ST*ptr=Start;
67     if (Start==NULL)
68     {
69         cout<<"Underflow!"<<endl;
70         return;
71     }
72     else{
73         cout<<"Traversed List will be:"<<endl;
74         while (ptr!=NULL)
75         {
76             cout<<ptr->x<<" ";

```

```

77         ptr=ptr->next;
78     }
79     return;
80 }
81 LogFile("Traversed Stack List");
82 }
83 int peek(ST*&start)
84 {
85     if (start==NULL)
86     {
87         cout<<"Underflow!"<<endl;
88         return 0;
89     }
90     else
91     {
92         return start->x;
93     }
94     LogFile("Peeked first Node");
95 }
96
97 int main()
98 {
99     LogFile("Open File");
100    ST *ptr=NULL;
101    int z;
102    while (true){
103        cout<<"\n\nWelcome to Stack Manager:"<<endl;
104        cout<<"Select Your Commands:"<<endl;
105        cout<<"1. Push an element into a Stack"<<endl;
106        cout<<"2. Pop an element from the Stack"<<endl;
107        cout<<"3. Traverse a Stack"<<endl;
108        cout<<"4. Peek Your Stack"<<endl;
109        cout<<"5. Exit Program."<<endl;
110        cout<<"Enter choice:"; cin>>z;
111        switch (z){
112            case 1:
113                LogFile("Call Push Function");
114                {
115                    int value;
116                    cout<<"Enter Value to Push: "; cin>>value;
117                    push(ptr,value);
118                }
119                break;
120            case 2:
121                LogFile("Call Pop Function");
122                cout<<pop(ptr)<<endl;
123                break;
124            case 3:
125                LogFile("Call Traverse Function");
126                traverse(ptr);
127                break;
128            case 4:
129                LogFile("Call Peek Function");
130                cout<<peek(ptr)<<endl;

```

```

131         break;
132     case 5:
133         LogFile("Close File");
134         while (ptr != NULL) {
135             pop(ptr); // Free all remaining nodes
136         }
137         return 0;
138         break;
139     default:
140         LogFile("Force to close File");
141         return 0;
142         break;
143     }
144 }
145}

```

### 3. Doubly Linked Implementation of Stack

```

1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cmath>
5  #include <ctime>
6  #include <bitset>
7  using namespace std;
8
9  void LogFile(const string& event) {
10     ofstream BinFile("Machine_Code_Stack.txt", ios_base::app);
11     if (BinFile.is_open())
12     {
13         for (char c : event)
14         {
15             BinFile << bitset<8>(c) << " ";
16         }
17         BinFile << endl;
18         BinFile.close();
19     }
20 }
21 // Doubly linked list node structure
22 struct Node {
23     int x;
24     Node *next;
25     Node *prev;
26 };
27 typedef Node ST;
28
29 void push(ST*&start, int& val)
30 {
31     ST* newNode = new ST;
32     newNode->x = val;
33     newNode->next = NULL;
34     newNode->prev = NULL;
35
36     if (start == NULL)
37     {
38         start = newNode; // Stack is empty, so newNode becomes the start
39     }

```



```

40     else
41     {
42         start->prev = newNode; // Link the new node to the current start
43         newNode->next = start; // Link newNode's next to the old start
44         start = newNode;      // Update start to point to newNode
45     }
46     cout << "Item Pushed!" << endl;
47     LogFile("Pushed value " + to_string(val));
48 }
49 int pop(ST*& start)
50 {
51     if (start == NULL)
52     {
53         cout << "Underflow!" << endl;
54         return 0;
55     }
56     else
57     {
58         ST* ptr = start;
59         int value = ptr->x;
60         start = start->next; // Move start to the next node
61         if (start != NULL)
62             start->prev = NULL; // Break the backward link for the new start
63         delete ptr; // Free the popped node
64         cout << "Item Popped!" << endl;
65         LogFile("Popped value " + to_string(value));
66         return value;
67     }
68
69 void traverse(ST*& start)
70 {
71     ST* ptr = start;
72     if (start == NULL)
73     {
74         cout << "Underflow!" << endl;
75         return;
76     }
77     else {
78         cout << "Traversed List: ";
79         while (ptr != NULL)
80         {
81             cout << ptr->x << " ";
82             ptr = ptr->next;
83         }
84         cout << endl;
85     }
86     LogFile("Traversed Stack List");
87 }
88 int peek(ST*& start)
89 {
90     if (start == NULL)
91     {
92         cout << "Underflow!" << endl;
93         return 0;

```

```

94     }
95     else
96     {
97         cout << "Peeked Value: " << start->x << endl;
98         LogFile("Peeked value " + to_string(start->x));
99         return start->x;
100    }
101}
102int main()
103{
104    LogFile("Open File");
105    ST *ptr = NULL;
106    int z;
107    while (true) {
108        cout << "\n\nWelcome to Stack Manager:" << endl;
109        cout << "Select Your Commands:" << endl;
110        cout << "1. Push an element into a Stack" << endl;
111        cout << "2. Pop an element from the Stack" << endl;
112        cout << "3. Traverse a Stack" << endl;
113        cout << "4. Peek Your Stack" << endl;
114        cout << "5. Exit Program." << endl;
115        cout << "Enter choice: "; cin >> z;
116        switch (z)
117        {
118            case 1:
119                LogFile("Call Push Function");
120                {
121                    int value;
122                    cout << "Enter Value to Push: "; cin >> value;
123                    push(ptr, value);
124                }
125                break;
126            case 2:
127                LogFile("Call Pop Function");
128                cout << pop(ptr) << endl;
129                break;
130            case 3:
131                LogFile("Call Traverse Function");
132                traverse(ptr);
133                break;
134            case 4:
135                LogFile("Call Peek Function");
136                peek(ptr);
137                break;
138            case 5:
139                LogFile("Close File");
140                while (ptr != NULL) {
141                    pop(ptr); // Free all remaining nodes
142                }
143                return 0;
144            default:
145                LogFile("Force to close File");
146                return 0;
147        }
148    }
149}

```

## CODE

### 1. Array Implementation of Queue:

```
1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cmath>
5  #include <ctime>
6  #include <bitset>
7  using namespace std;
8
9  void LogFile(const string& event) {
10     ofstream BinFile("Machine_Code_Queue.txt",ios_base::app);
11     if (BinFile.is_open())
12     {
13         for (char c:event)
14         {
15             BinFile<<bitset<8>(c)<<" ";
16         }
17         BinFile<<endl;
18         BinFile.close();
19     }
20
21 }
22
23 const int maxsize=25;
24 void insertElement(int Q[],int info,int size,int &R, int &F)
25 {
26     // Overflow check
27     if (R==size-1)
28     {
29         cout<<"Overflow!"<<endl;
30         return;
31     }
32     //Empty queue check
33     if (F==-1 || R==-1)
34     {
35         F=R=0;
36         Q[R]=info;
37     }
38     else{
39         ++R;
40         Q[R]=info;
41         for (int i = R; i > F; i--)
42         {
43             if (Q[i] < Q[i - 1])
44             {
45                 swap(Q[i], Q[i - 1]);
46             }
47         }
48     }
49
50     LogFile("Inserted "+ to_string(info)+" In Queue ");
```

```

51     return;
52 }
53 void deleteElement(int Q[], int &R, int &F, int data)
54 {
55     // Empty queue check
56     if (F == -1 || F > R)
57     {
58         cout << "Underflow!" << endl;
59         return;
60     }
61     else
62     {
63         int i = F;
64         bool found = false;
65         // Search for the element
66         for (; i <= R; i++)
67         {
68             if (Q[i] == data)
69             {
70                 found = true;
71                 break;
72             }
73         }
74         if (!found)
75         {
76             cout << "Element not found!" << endl;
77             return;
78         }
79         // Shift elements left to remove the found element
80         for (int j = i; j < R; j++)
81         {
82             Q[j] = Q[j + 1];
83         }
84         R--; // Reduce the rear index
85         if (R < F)
86         {
87             F = R = -1; // Queue becomes empty
88         }
89     }
90     LogFile("Deleted Element from Queue");
91 }
92 void Traverse(int Q[], int size, int R, int F)
93 {
94     if (F == -1 || F > R)
95     {
96         cout << "Underflow!" << endl;
97         return;
98     }
99     int x = F;
100     cout << "Elements of Queue will be: " << endl;
101     while (x <= R)
102     {
103         cout << Q[x] << endl;
104         ++x;
105     }
106     LogFile("Traverse Queue");
107 }

```

```

108int main()
109{
110    LogFile("Open File");
111    int queue[maxsize],front=-1,rear=-1,z;
112    while (true)
113    {
114        cout<<"\n\nWelcome to Queue Manager\n\n"<<endl;
115        cout<<"Select Commands"<<endl;
116        cout<<"1. Insert element"<<endl;
117        cout<<"2. Delete element"<<endl;
118        cout<<"3. Traverse Queue"<<endl;
119        cout<<"4. Quit Program\n\n"<<endl;
120        cout<<"Choose Your Command: "; cin>>z;
121        switch (z)
122        {
123            case 1:
124                LogFile("Call InsertElement Function");
125                {
126                    int info;
127                    cout<<"Enter info you want to enter: ";cin>>info;
128                    insertElement(queue,info,maxsize,rear,front);
129                }
130                break;
131            case 2:
132                LogFile("Call InsertElement Function");
133                {
134                    int info;
135                    cout<<"enter element to be deleted:"; cin>>info;
136                    deleteElement(queue,rear,front,info);
137                }
138                break;
139            case 3:
140                LogFile("Call Traverse Function");
141                Traverse(queue,maxsize,rear,front);
142                break;
143            case 4:
144                return 0;
145            default:
146                break;
147        }
148    }
149}

```

## 2. Singly Linked Implementation of Queue

```

1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cmath>
5  #include <ctime>
6  #include <bitset>
7  using namespace std;
8  void LogFile(const string& event) {
9      ofstream BinFile("Machine_Code_Queue.txt", ios_base::app);

```

```

10     if (BinFile.is_open())
11     {
12         for (char c : event)
13         {
14             BinFile << bitset<8>(c) << " ";
15         }
16         BinFile << endl;
17         BinFile.close();
18     }
19 }
20 struct Node
21 {
22     int num;
23     Node* next;
24 };
25 typedef Node Q;
26 // Insert element in the queue
27 void insertElement(Q*& F, Q*& R, int data)
28 {
29     Q* NewNode = new Q;
30     NewNode->num = data;
31     NewNode->next = NULL;
32     // If the queue is empty, set front and rear
33     if (F == NULL && R == NULL){
34         F = R = NewNode;
35     }
36     else if (data < F->num) // Insert at the front
37     {
38         NewNode->next = F;
39         F = NewNode;
40     }
41     else // Insert in the sorted position{
42         Q* temp = F;
43         Q* prev = NULL;
44
45         // Traverse to find the correct position
46         while (temp != NULL && temp->num <= data)
47         {
48             prev = temp;
49             temp = temp->next;
50         }
51         // Insert in the middle or at the end
52         prev->next = NewNode;
53         NewNode->next = temp;
54         // If inserted at the end, update the rear pointer
55         if (temp == NULL)
56         {
57             R = NewNode;
58         }
59     }
60     LogFile("Inserted element: " + to_string(data));
61 }
62 // Delete element from the queue
63 void deleteElement(Q*& F, Q*& R, int data)
64 {
65     if (F == NULL)
66     {

```

```

67         cout << "Queue underflow" << endl;
68         LogFile("Queue underflow - Delete failed");
69         return;
70     }
71     Q* temp = F;
72     Q* prev = NULL;
73     // If the element to be deleted is the first element
74     if (F->num == data)
75     {
76         F = F->next;
77         // If it was the only element in the queue
78         if (F == NULL)
79         {
80             R = NULL; // The queue is now empty
81         }
82         cout << "Deleted element: " << temp->num << endl;
83         LogFile("Deleted element: " + to_string(temp->num));
84         delete temp;
85         return;
86     }
87     // Traverse to find the element to delete
88     while (temp != NULL && temp->num != data)
89     {
90         prev = temp;
91         temp = temp->next;
92     }
93     if (temp == NULL) // Element not found
94     {
95         cout << "Element not found in the queue" << endl;
96         LogFile("Element not found in the queue - Delete failed");
97         return;
98     }
99     // Unlink the node from the list
100    prev->next = temp->next;
101    // If the node to be deleted is the last node, update the rear pointer
102    if (temp == R)
103        R = prev;
104    cout << "Deleted element: " << temp->num << endl;
105    LogFile("Deleted element: " + to_string(temp->num));
106    delete temp;
107}
108// Traverse the queue and print elements
109void Traverse(Q* F){
110    if (F == NULL){
111        cout << "Queue is empty" << endl;
112        LogFile("Queue is empty - Traverse");
113        return;
114    }
115    cout << "Queue elements: ";
116    Q* temp = F;
117    while (temp != NULL){
118        cout << temp->num << " ";
119        temp = temp->next;
120    }

```

```

121     cout << endl;
122     LogFile("Traversed queue");
123 }
124 int main()
125 {
126     int z;
127     int data;
128     Q* front = NULL;
129     Q* rear = NULL;
130     LogFile("Opened File");
131     while (true){
132         cout << "\n\nWelcome to Queue Manager\n\n" << endl;
133         cout << "Select Commands" << endl;
134         cout << "1. Insert element" << endl;
135         cout << "2. Delete element" << endl;
136         cout << "3. Traverse Queue" << endl;
137         cout << "4. Quit Program\n\n" << endl;
138         cout << "Choose Your Command: ";
139         cin >> z;
140         switch (z){
141             case 1:
142                 cout << "Enter element to insert: ";
143                 cin >> data;
144                 insertElement(front, rear, data);
145                 break;
146             case 2:
147                 cout << "Enter element to delete: ";
148                 cin >> data;
149                 deleteElement(front, rear, data);
150                 break;
151             case 3:
152                 Traverse(front);
153                 break;
154             case 4:
155                 LogFile("Exiting Program");
156                 exit(0);
157             default:
158                 cout << "Invalid command" << endl;
159         }
160     }
161     return 0;
162 }

```

### 3. Doubly Linked Implementation of Queue

```

1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <cmath>
5  #include <ctime>
6  #include <bitset>
7  using namespace std;
8  void LogFile(const string& event) {
9      ofstream BinFile("Machine_Code_Queue.txt", ios_base::app);
10     if (BinFile.is_open())
11     {
12         for (char c : event)

```



```

13     {
14         BinFile << bitset<8>(c) << " ";
15     }
16     BinFile << endl;
17     BinFile.close();
18 }
19 }
20 // Doubly Linked List Node Structure
21 struct Node
22 {
23     int num;
24     Node* next;
25     Node* prev;
26 };
27 typedef Node Q;
28 // Insert element in a sorted position
29 void insertElement(Q*& F, Q*& R, int data)
30 {
31     Q* NewNode = new Q;
32     NewNode->num = data;
33     NewNode->next = NULL;
34     NewNode->prev = NULL;
35
36     // Case 1: If the list is empty
37     if (F == NULL && R == NULL)
38     {
39         F = R = NewNode;
40     }
41     else
42     {
43         Q* temp = F;
44
45         // Case 2: Insertion at the front (smallest element)
46         if (data < F->num)
47         {
48             NewNode->next = F;
49             F->prev = NewNode;
50             F = NewNode;
51         }
52         else
53         {
54             // Traverse the list to find the correct position
55             while (temp != NULL && temp->num <= data)
56             {
57                 temp = temp->next;
58             }
59             // Case 3: Insertion at the end (largest element)
60             if (temp == NULL)
61             {
62                 NewNode->prev = R;
63                 R->next = NewNode;
64                 R = NewNode;
65             }
66             else

```

```

67         {
68             // Case 4: Insertion in the middle
69             NewNode->next = temp;
70             NewNode->prev = temp->prev;
71             temp->prev->next = NewNode;
72             temp->prev = NewNode;
73         }
74     }
75 }
76 LogFile("Inserted element in sorted order: " + to_string(data));
77 }
78 // Delete element with a specific value
79 void deleteByValue(Q*& F, Q*& R, int data)
80 {
81     if (F == NULL)
82     {
83         cout << "Queue is empty, cannot delete." << endl;
84         LogFile("Queue underflow - Delete by value failed");
85         return;
86     }
87
88     Q* temp = F;
89
90     // Traverse the list to find the node with the given value
91     while (temp != NULL && temp->num != data)
92     {
93         temp = temp->next;
94     }
95
96     if (temp == NULL) // Case 1: Element not found
97     {
98         cout << "Element " << data << " not found in the queue." << endl;
99         LogFile("Element not found in the queue - Delete failed");
100        return;
101    }
102    // Case 2: Deleting the first node
103    if (temp == F)
104    {
105        F = F->next;
106        if (F != NULL)
107        {
108            F->prev = NULL;
109        }
110        else
111        {
112            R = NULL; // The list becomes empty
113        }
114    }
115
116    else if (temp == R) // Case 3: Deleting the last node
117    {
118        R = R->prev;
119        R->next = NULL;
120    }
121    else // Case 4: Deleting from the middle
122    {
123        temp->prev->next = temp->next;

```

```

124     temp->next->prev = temp->prev;
125 }
126 cout << "Deleted element: " << temp->num << endl;
127 LogFile("Deleted element: " + to_string(temp->num));
128 delete temp;
129}
130void Traverse(Q*& F) // Traverse the queue from front to rear
131{
132     if (F == NULL)
133     {
134         cout << "Queue is empty" << endl;
135         LogFile("Queue is empty - Traverse");
136         return;
137     }
138     Q* temp = F;
139     while (temp != NULL)
140     {
141         cout << temp->num << " ";
142         temp = temp->next;
143     }
144     cout << endl;
145     LogFile("Traversed queue");
146}
147void TraverseReverse(Q*& R) // Traverse the queue in reverse from rear to front
148{
149     if (R == NULL)
150     {
151         cout << "Queue is empty" << endl;
152         LogFile("Queue is empty - Reverse Traverse");
153         return;
154     }
155     Q* temp = R;
156     while (temp != NULL)
157     {
158         cout << temp->num << " ";
159         temp = temp->prev;
160     }
161     cout << endl;
162     LogFile("Traversed queue in reverse");
163}
164int main()
165{
166     int z;
167     int data;
168     Q* front = NULL;
169     Q* rear = NULL;
170     LogFile("Opened File");
171     while (true)
172     {
173         cout << "\n\nWelcome to Queue Manager\n\n" << endl;
174         cout << "Select Commands" << endl;
175         cout << "1. Insert element (sorted)" << endl;
176         cout << "2. Delete element by value" << endl;
177         cout << "3. Traverse and Reverse Traverse Queue" << endl;

```

```

178     cout << "4. Quit Program\n\n" << endl;
179     cout << "Choose Your Command: ";
180     cin >> z;
181     switch (z)
182     {
183     case 1:
184         cout << "Enter element to insert: ";
185         cin >> data;
186         insertElement(front, rear, data);
187         break;
188     case 2:
189         cout << "Enter element to delete: ";
190         cin >> data;
191         deleteByValue(front, rear, data);
192         break;
193     case 3:
194         cout << "Queue from front to rear: ";
195         Traverse(front);
196         cout << "Queue from rear to front: ";
197         TraverseReverse(rear);
198         break;
199     case 4:
200         LogFile("Exiting Program");
201         exit(0);
202     default:
203         cout << "Invalid command" << endl;
204     }
205 }
206 return 0;
207}

```