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 3rd

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

CODE

1. Array Implemented Stack:

```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <ctime>
#include <bitset>
using namespace std;
void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Stack.txt",ios_base::app);
    if (BinFile.is_open())
    {
        for (char c:event)
        {
            BinFile<<bitset<8>(c)<<" ";
        }
        BinFile<<endl;
        BinFile.close();
    }

}
const int SIZE=10;
bool IsFull(int size,int& Tos)
{
    LogFile("Stack is Full");
    return(Tos==size-1);
}
bool IsEmpty(int& Tos)
{
    LogFile("Stack is Empty");
    return(Tos==-1);
}
void Push(int Stack[],int size,int& Tos,int info)
{
    // To check if Stack is full
    LogFile("Pushed Value " +to_string(info)+" to Stack");
    if (IsFull(size,Tos))
    {
        cout<<"\nStack Overflow!"<<endl;
        return;
    }
    else
        Stack[++Tos]=info;
}
int pop(int Stack[],int &Tos)
{
    LogFile("Popped Value from Stack");
    // To check if Stack is Empty
    if (IsEmpty(Tos))
    {
        cout<<"\nStack Underflow!"<<endl;
```

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

```
108         cout<<"Enter choice:"; cin>>z;
109         switch (z)
110         {
111         case 1:
112             LogFile("Call Push Function");
113             {
114                 int info;
115                 cout<<"Enter info you want to push in Stack: "; cin>>info;
116                 Push(S,SIZE,ToS,info);
117             }
118             break;
119         case 2:
120             LogFile("Call Pop Function");
121             pop(S,ToS);
122             break;
123         case 3:
124             LogFile("Call Traverse Function");
125             traverse(S,BS,SIZE,ToS,BToS);
126             break;
127         case 4:
128             LogFile("Call Peek Function");
129             cout<<Peek(S,ToS)<<endl<<endl;
130             break;
131         case 5:
132             LogFile("Close File");
133             return 0;
134         default:
135             LogFile("Force to close File");
136             cout<<"Entered Invalid Option."<<endl;
137             return 0;
138             break;
139         }
140     }
```

Machine code:

```
01001111 01110000 01100101 01101110 00100000 01000110 01101001 01101100 01100101
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 00110100 00110011 00110100 00100000 01110100
01101111 00100000 01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000110 01110101 01101100 01101100
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 00111000 00111001 00100000 01110100 01101111
00100000 01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000110 01110101 01101100 01101100
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 00110000 00100000 01110100 01101111 00100000
01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000110 01110101 01101100 01101100
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 00110010 00110011 00100000 01110100 01101111
00100000 01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000110 01110101 01101100 01101100
01000011 01100001 01101100 01101100 00100000 01010000 01101111 01110000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01101111 01110000 01110000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 01100110 01110010 01101111 01101101 00100000
01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000101 01101101 01110000 01110100 01111001
01000011 01100001 01101100 01101100 00100000 01010100 01110010 01100001 01110110
01100101 01110010 01110011 01100101 00100000 01000110 01110101 01101110 01100011
01110100 01101001 01101111 01101110
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000101 01101101 01110000 01110100 01111001
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000101 01101101 01110000 01110100 01111001
01010000 01101111 01110000 01110000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 01100110 01110010 01101111 01101101 00100000
01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000101 01101101 01110000 01110100 01111001
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01010110 01100001
01101100 01110101 01100101 00100000 00110000 00100000 01110100 01101111 00100000
01010011 01110100 01100001 01100011 01101011
01010011 01110100 01100001 01100011 01101011 00100000 01101001 01110011 00100000
01000110 01110101 01101100 01101100
```

2. Singly Linked Implementation of Stack

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <ctime>
#include <bitset>
using namespace std;

void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Stack.txt",ios_base::app);
    if (BinFile.is_open())
    {
        for (char c:event)
        {
            BinFile<<bitset<8>(c)<<" ";
        }
        BinFile<<endl;
        BinFile.close();
    }

}
struct Node{
    int x;
    Node *next;

};
typedef Node ST;
void push(ST*&start, int& val)
{
    ST*newNode=new ST;
    newNode->x=val;
    newNode->next=NULL;
    //check if list is empty
    if (start==NULL)
        start=newNode;
    else
    {
        newNode->next=start;
        start=newNode;
    }
    cout<<"Item Pushed!"<<endl;
    LogFile("Pushed value " + to_string(val));
}
int pop(ST*& start)
{
    // check if list is empty
    if (start==NULL)
    {
        cout<<"Underflow!"<<endl;
        return 0;
    }
```

```
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 }
```

Machine code:

```
01001111 01110000 01100101 01101110 00100000 01000110 01101001 01101100 01100101

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110011 00110010

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110101 00110110

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110010 00110011

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110110 00110101

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110010 00110011 00110101

01000011 01100001 01101100 01101100 00100000 01010000 01101111 01110000 00100000 01000110
01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01101111 01110000 01110000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110011 00110010

01000011 01100001 01101100 01101100 00100000 01010100 01110010 01100001 01110110 01100101
01110010 01110011 01100101 00100000 01000110 01110101 01101110 01100011 01110100 01101001
01101111 01101110

01000011 01100001 01101100 01101100 00100000 01010100 01110010 01100001 01110110 01100101
01110010 01110011 01100101 00100000 01000110 01110101 01101110 01100011 01110100 01101001
01101111 01101110

01000011 01100001 01101100 01101100 00100000 01010000 01100101 01100101 01101011 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01001111 01110000 01100101 01101110 00100000 01000110 01101001 01101100 01100101

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110011 00110010

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110

01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110110 00110111

01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
```

3. Doubly Linked Implementation of Stack

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <ctime>
#include <bitset>
using namespace std;

void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Stack.txt", ios_base::app);
    if (BinFile.is_open())
    {
        for (char c : event)
        {
            BinFile << bitset<8>(c) << " ";
        }
        BinFile << endl;
        BinFile.close();
    }
}
// Doubly linked list node structure
struct Node {
    int x;
    Node *next;
    Node *prev;
};
typedef Node ST;

void push(ST*&start, int& val)
{
    ST* newNode = new ST;
    newNode->x = val;
    newNode->next = NULL;
    newNode->prev = NULL;

    if (start == NULL)
    {
        start = newNode;  // Stack is empty, so newNode becomes the start
    }
    else
    {
        start->prev = newNode;  // Link the new node to the current start
        newNode->next = start;  // Link newNode's next to the old start
        start = newNode;        // Update start to point to newNode
    }
    cout << "Item Pushed!" << endl;
    LogFile("Pushed value " + to_string(val));
}

int pop(ST*& start)
{
    if (start == NULL)
```

```
53      {
54          cout << "Underflow!" << endl;
55          return 0;
56      }
57      else
58      {
59          ST* ptr = start;
60          int value = ptr->x;
61          start = start->next;  // Move start to the next node
62          if (start != NULL)
63              start->prev = NULL;  // Break the backward link for the new start
64          delete ptr;  // Free the popped node
65          cout << "Item Popped!" << endl;
66          LogFile("Popped value " + to_string(value));
67          return value;
68      }
69  }
70
71  void traverse(ST*& start)
72  {
73      ST* ptr = start;
74      if (start == NULL)
75      {
76          cout << "Underflow!" << endl;
77          return;
78      }
79      else {
80          cout << "Traversed List: ";
81          while (ptr != NULL)
82          {
83              cout << ptr->x << " ";
84              ptr = ptr->next;
85          }
86          cout << endl;
87      }
88      LogFile("Traversed Stack List");
89  }
90
91  int peek(ST*& start)
92  {
93      if (start == NULL)
94      {
95          cout << "Underflow!" << endl;
96          return 0;
97      }
98      else
99      {
100         cout << "Peeked Value: " << start->x << endl;
101         LogFile("Peeked value " + to_string(start->x));
102         return start->x;
103     }
104 }
105
106 int main()
107 {
108     LogFile("Open File");
109     ST *ptr = NULL;
```

```
110     int z;
111     while (true) {
112         cout << "\n\nWelcome to Stack Manager:" << endl;
113         cout << "Select Your Commands:" << endl;
114         cout << "1. Push an element into a Stack" << endl;
115         cout << "2. Pop an element from the Stack" << endl;
116         cout << "3. Traverse a Stack" << endl;
117         cout << "4. Peek Your Stack" << endl;
118         cout << "5. Exit Program." << endl;
119         cout << "Enter choice: "; cin >> z;
120         switch (z)
121         {
122         case 1:
123             LogFile("Call Push Function");
124             {
125                 int value;
126                 cout << "Enter Value to Push: "; cin >> value;
127                 push(ptr, value);
128             }
129             break;
130         case 2:
131             LogFile("Call Pop Function");
132             cout << pop(ptr) << endl;
133             break;
134         case 3:
135             LogFile("Call Traverse Function");
136             traverse(ptr);
137             break;
138         case 4:
139             LogFile("Call Peek Function");
140             peek(ptr);
141             break;
142         case 5:
143             LogFile("Close File");
144             while (ptr != NULL) {
145                 pop(ptr);  // Free all remaining nodes
146             }
147             return 0;
148         default:
149             LogFile("Force to close File");
150             return 0;
151         }
152     }
153}
```

Machine code:

```
01001111 01110000 01100101 01101110 00100000 01000110 01101001 01101100 01100101
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110010 00110011
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110111 00110110 00110100
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00111001
01000011 01100001 01101100 01101100 00100000 01010000 01110101 01110011 01101000 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01110101 01110011 01101000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110011
01000011 01100001 01101100 01101100 00100000 01010000 01101111 01110000 00100000 01000110
01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01101111 01110000 01110000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110011
01000011 01100001 01101100 01101100 00100000 01010100 01110010 01100001 01110110 01100101
01110010 01110011 01100101 00100000 01000110 01110101 01101110 01100011 01110100 01101001
01101111 01101110
01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 01100100 00100000
01010011 01110100 01100001 01100011 01101011 00100000 01001100 01101001 01110011 01110100
01000011 01100001 01101100 01101100 00100000 01010000 01100101 01100101 01101011 00100000
01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01010000 01100101 01100101 01101011 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00111001
01000011 01101100 01101111 01110011 01100101 00100000 01000110 01101001 01101100 01100101
01010000 01101111 01110000 01110000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00111001
01010000 01101111 01110000 01110000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110111 00110110 00110100
01010000 01101111 01110000 01110000 01100101 01100100 00100000 01110110 01100001 01101100
01110101 01100101 00100000 00110010 00110011
```

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 }
```

```
int main()
{
    LogFile("Open File");
    int queue[maxsize],front=-1,rear=-1,z;
    while (true)
    {
        cout<<"\n\nWelcome to Queue Manager\n\n"<<endl;
        cout<<"Select Commands"<<endl;
        cout<<"1. Insert element"<<endl;
        cout<<"2. Delete element"<<endl;
        cout<<"3. Traverse Queue"<<endl;
        cout<<"4. Quit Program\n\n"<<endl;
        cout<<"Choose Your Command: "; cin>>z;
        switch (z)
        {
        case 1:
            LogFile("Call InsertElement Function");
            {
                int info;
                cout<<"Enter info you want to enter: ";cin>>info;
                insertElement(queue,info,maxsize,rear,front);
            }
            break;
        case 2:
            LogFile("Call InsertElement Function");
            {
                int info;
                cout<<"enter element to be deleted:"; cin>>info;
                deleteElement(queue,rear,front,info);
            }
            break;
        case 3:
            LogFile("Call Traverse Function");
            Traverse(queue,maxsize,rear,front);
            break;
        case 4:
            return 0;
        default:
            break;
        }
    }
}
```

Machine Code:

```
01001111 01110000 01100101 01101110 00100000 01000110 01101001 01101100 01100101
01000011 01100001 01101100 01101100 00100000 01001001 01101110 01110011 01100101
01110010 01110100 01000101 01101100 01100101 01101101 01100101 01101110 01110100
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
00110011 00110010 00100000 01001001 01101110 00100000 01010001 01110101 01100101
01110101 01100101 00100000
01000011 01100001 01101100 01101100 00100000 01001001 01101110 01110011 01100101
01110010 01110100 01000101 01101100 01100101 01101101 01100101 01101110 01110100
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
00110011 00110100 00110011 00100000 01001001 01101110 00100000 01010001 01110101
01100101 01110101 01100101 00100000
01000011 01100001 01101100 01101100 00100000 01001001 01101110 01110011 01100101
01110010 01110100 01000101 01101100 01100101 01101101 01100101 01101110 01110100
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
00110010 00110011 00100000 01001001 01101110 00100000 01010001 01110101 01100101
01110101 01100101 00100000
01000011 01100001 01101100 01101100 00100000 01001001 01101110 01110011 01100101
01110010 01110100 01000101 01101100 01100101 01101101 01100101 01101110 01110100
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
00110101 00110011 00100000 01001001 01101110 00100000 01010001 01110101 01100101
01110101 01100101 00100000
01000011 01100001 01101100 01101100 00100000 01001001 01101110 01110011 01100101
01110010 01110100 01000101 01101100 01100101 01101101 01100101 01101110 01110100
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01000100 01100101 01101100 01100101 01110100 01100101 01100100 00100000 01000101
01101100 01100101 01101101 01100101 01101110 01110100 00100000 01100110 01110010
01101111 01101101 00100000 01010001 01110101 01100101 01110101 01100101
01000011 01100001 01101100 01101100 00100000 01010100 01110010 01100001 01110110
01100101 01110010 01110011 01100101 00100000 01000110 01110101 01101110 01100011
01110100 01101001 01101111 01101110
01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 00100000
01010001 01110101 01100101 01110101 01100101
01001111 01110000 01100101 01101110 00100000 01000110 01101001 01101100 01100101
01000011 01100001 01101100 01101100 00100000 01001001 01101110 01110011 01100101
01110010 01110100 01000101 01101100 01100101 01101101 01100101 01101110 01110100
00100000 01000110 01110101 01101110 01100011 01110100 01101001 01101111 01101110
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
00110011 00110010 00100000 01001001 01101110 00100000 01010001 01110101 01100101
01110101 01100101 00100000
```

2. Singly Linked Implementation of Queue

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <ctime>
#include <bitset>
using namespace std;
void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Queue.txt", ios_base::app);
    if (BinFile.is_open())
    {
        for (char c : event)
        {
            BinFile << bitset<8>(c) << " ";
        }
        BinFile << endl;
        BinFile.close();
    }
}
struct Node
{
    int num;
    Node* next;
};
typedef Node Q;
// Insert element in the queue
void insertElement(Q*& F, Q*& R, int data)
{
    Q* NewNode = new Q;
    NewNode->num = data;
    NewNode->next = NULL;
    // If the queue is empty, set front and rear
    if (F == NULL && R == NULL){
        F = R = NewNode;
    }
    else if (data < F->num)  // Insert at the front
    {
        NewNode->next = F;
        F = NewNode;
    }
    else // Insert in the sorted position{
        Q* temp = F;
        Q* prev = NULL;

        // Traverse to find the correct position
        while (temp != NULL && temp->num <= data)
        {
            prev = temp;
            temp = temp->next;
        }
        // Insert in the middle or at the end
        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
109 void 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}
124int 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}
```

Machine Code:

```
01001111 01110000 01100101 01101110 01100101 01100100 00100000 01000110 01101001
01101100 01100101
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110010 00110011
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110100 00110010
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110110 00110101
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110011 00110100 00110101
01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110011 00110100 00110010 00110100
01000100 01100101 01101100 01100101 01110100 01100101 01100100 00100000 01100101
01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000 00110010
00110011
01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 01100100
00100000 01110001 01110101 01100101 01110101 01100101
```

3. Doubly Linked Implementation of Queue

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <ctime>
#include <bitset>
using namespace std;
void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Queue.txt", ios_base::app);
    if (BinFile.is_open())
    {
        for (char c : event)
        {
            BinFile << bitset<8>(c) << " ";
        }
        BinFile << endl;
        BinFile.close();
    }
}
// Doubly Linked List Node Structure
struct Node
{
    int num;
    Node* next;
    Node* prev;
};
typedef Node Q;
// Insert element in a sorted position
void insertElement(Q*& F, Q*& R, int data)
{
    Q* NewNode = new Q;
    NewNode->num = data;
    NewNode->next = NULL;
    NewNode->prev = NULL;

    // Case 1: If the list is empty
    if (F == NULL && R == NULL)
    {
        F = R = NewNode;
    }
    else
    {
        Q* temp = F;

        // Case 2: Insertion at the front (smallest element)
        if (data < F->num)
        {
            NewNode->next = F;
            F->prev = NewNode;
            F = NewNode;
        }
        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}
```

```cpp
int main()
{
    int z;
    int data;
    Q* front = NULL;
    Q* rear = NULL;
    LogFile("Opened File");
    while (true)
    {
        cout << "\n\nWelcome to Queue Manager\n\n" << endl;
        cout << "Select Commands" << endl;
        cout << "1. Insert element (sorted)" << endl;
        cout << "2. Delete element by value" << endl;
        cout << "3. Traverse and Reverse Traverse Queue" << endl;
        cout << "4. Quit Program\n\n" << endl;
        cout << "Choose Your Command: ";
        cin >> z;
        switch (z)
        {
            case 1:
                cout << "Enter element to insert: ";
                cin >> data;
                insertElement(front, rear, data);
                break;
            case 2:
                cout << "Enter element to delete: ";
                cin >> data;
                deleteByValue(front, rear, data);
                break;
            case 3:
                cout << "Queue from front to rear: ";
                Traverse(front);
                cout << "Queue from rear to front: ";
                TraverseReverse(rear);
                break;
            case 4:
                LogFile("Exiting Program");
                exit(0);
            default:
                cout << "Invalid command" << endl;
        }
    }
    return 0;
}
```

Machine Code:

```
01001111 01110000 01100101 01101110 01100101 01100100 00100000 01000110 01101001
01101100 01100101

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110100 00110010

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110110 00110110

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110100 00110011

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110111 00110101

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110010 00110011 00110100

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110101 00111001 00110010 00111001 00111001 00110010

01000100 01100101 01101100 01100101 01110100 01100101 01100100 00100000 01100101
01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000 00110100
00110010

01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 01100100
00100000 01110001 01110101 01100101 01110101 01100101

01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 01100100
00100000 01110001 01110101 01100101 01110101 01100101 00100000 01101001 01101110
00100000 01110010 01100101 01110110 01100101 01110010 01110011 01100101

01000101 01111000 01101001 01110100 01101001 01101110 01100111 00100000 01010000
01110010 01101111 01100111 01110010 01100001 01101101

01001111 01110000 01100101 01101110 01100101 01100100 00100000 01000110 01101001
01101100 01100101

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110100 00110010

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110110 00110110

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110100 00110011

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110111 00110101
```

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110010 00110011 00110100

01001001 01101110 01110011 01100101 01110010 01110100 01100101 01100100 00100000
01100101 01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000
00110101 00111001 00110010 00111001 00111001 00110010

01000100 01100101 01101100 01100101 01110100 01100101 01100100 00100000 01100101
01101100 01100101 01101101 01100101 01101110 01110100 00111010 00100000 00110100
00110010

01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 01100100
00100000 01110001 01110101 01100101 01110101 01100101

01010100 01110010 01100001 01110110 01100101 01110010 01110011 01100101 01100100
00100000 01110001 01110101 01100101 01110101 01100101 00100000 01101001 01101110
00100000 01110010 01100101 01110110 01100101 01110010 01110011 01100101

01000101 01111000 01101001 01110100 01101001 01101110 01100111 00100000 01010000
01110010 01101111 01100111 01110010 01100001 01101101