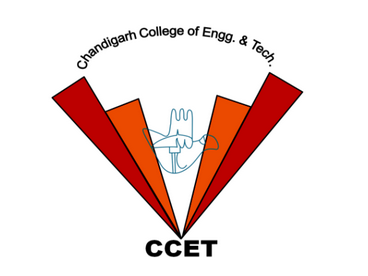
**CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)**

****

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: Submitted to:**

Bhavyam Dhand Dr. R.B. Patel

(CO23316) (Professor)

**Date of Practical:2-Sep Date of Submission:**23-Sep

**INDEX**

|  |  |  |
| --- | --- | --- |
| S.No | Content | Page no. |
|  | Objective | 3 |
|  | Discussion & Modelling of Problem | 4 |
|  | Implementation of Stacks | 5 |
|  | From a Static Data Structure (Array) |  |
|  | From a Singly Linked Data Structure | 6 |
|  | From a Doubly Linked Data Structure | 7 |
|  | CODE & Machine Code | 9 |
|  | Implementation of Queue | 21 |
|  | From a Static Data Structure (Array) |  |
|  | From a Singly Linked Data Structure | 23 |
|  | From a Doubly Linked Data Structure | 25 |
|  | CODE & Machine Code | 28 |

**CODE**

1. **Array Implemented Stack:**
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. #include <cmath>
6. #include <ctime>
7. #include <bitset>
8. using namespace std;
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. }
21. }
22. const int SIZE=10;
23. bool IsFull(int size,int& Tos)
24. {
25. LogFile("Stack is Full");
26. return(Tos==size-1);
27. }
28. bool IsEmpty(int& Tos)
29. {
30. LogFile("Stack is Empty");
31. return(Tos==-1);
32. }
33. void Push(int Stack[],int size,int& Tos,int info)
34. {
35. // To check if Stack is full
36. LogFile("Pushed Value " +to\_string(info)+" to Stack");
37. if (IsFull(size,Tos))
38. {
39. cout<<"\nStack Overflow!"<<endl;
40. return;
41. }
42. else
43. Stack[++Tos]=info;
44. }
45. int pop(int Stack[],int &Tos)
46. {
47. LogFile("Popped Value from Stack");
48. // To check if Stack is Empty
49. if (IsEmpty(Tos))
50. {
51. cout<<"\nStack Underflow!"<<endl;
52. return 0;
53. }
54. else
55. return Stack[Tos--];
56. }
57. void traverse(int Stack[], int BStack[], int size, int& Tos, int& BTos) {
58. if (IsEmpty(Tos)) {
59. cout << "\nStack is empty!" << endl;
60. return;
61. }
62. int info;
63. int tempTos = Tos; // Save the original Tos for restoring later
64. cout << "\nThe elements of Stack are:" << endl;
65. // Move elements from Stack to BStack and print them
66. while (!IsEmpty(Tos)) {
67. info = pop(Stack, Tos);
68. Push(BStack, size, BTos, info);
69. cout << info << endl;
70. }
71. // Restore elements from BStack back to Stack
72. while (!IsEmpty(BTos)) {
73. info = pop(BStack, BTos);
74. Push(Stack, size, Tos, info);
75. }
76. Tos = tempTos; // Restore original Tos
77. LogFile("traversing List");
78. }
79. int Peek(int Stack[],int& Tos)
80. {
81. LogFile("Peeked List");
82. if (IsEmpty(Tos))
83. {
84. cout<<"\nStack Underflow!"<<endl;
85. return 0;
86. }
87. return Stack[Tos];
88. }
89. int main()
90. {
91. LogFile("Open File");
92. int ToS=-1,BToS=-1;
93. int S[SIZE],BS[SIZE];
94. int z;
95. while (true){
96. cout<<"\n\nWelcome to Stack Manager:"<<endl;
97. cout<<"Select Your Commands:"<<endl;
98. cout<<"1. Push an element into a Stack"<<endl;
99. cout<<"2. Pop an element from the Stack"<<endl;
100. cout<<"3. Traverse a Stack"<<endl;
101. cout<<"4. Peek Your Stack"<<endl;
102. cout<<"5. Exit Program."<<endl;
103. cout<<"Enter choice:"; cin>>z;
104. switch (z)
105. {
106. case 1:
107. LogFile("Call Push Function");
108. {
109. int info;
110. cout<<"Enter info you want to push in Stack: "; cin>>info;
111. Push(S,SIZE,ToS,info);
112. }
113. break;
114. case 2:
115. LogFile("Call Pop Function");
116. pop(S,ToS);
117. break;
118. case 3:
119. LogFile("Call Traverse Function");
120. traverse(S,BS,SIZE,ToS,BToS);
121. break;
122. case 4:
123. LogFile("Call Peek Function");
124. cout<<Peek(S,ToS)<<endl<<endl;
125. break;
126. case 5:
127. LogFile("Close File");
128. return 0;
129. default:
130. LogFile("Force to close File");
131. cout<<"Entered Invalid Option."<<endl;
132. return 0;
133. break;
134. }
135. }

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

1. **Singly Linked Implementation of Stack**
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. #include <cmath>
6. #include <ctime>
7. #include <bitset>
8. using namespace std;
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. }
21. }
22. struct Node{
23. int x;
24. Node \*next;
25. };
26. typedef Node ST;
27. void push(ST\*&start, int& val)
28. {
29. ST\*newNode=new ST;
30. newNode->x=val;
31. newNode->next=NULL;
32. //check if list is empty
33. if (start==NULL)
34. start=newNode;
35. else
36. {
37. newNode->next=start;
38. start=newNode;
39. }
40. cout<<"Item Pushed!"<<endl;
41. LogFile("Pushed value " + to\_string(val));
42. }
43. int pop(ST\*& start)
44. {
45. // check if list is empty
46. if (start==NULL)
47. {
48. cout<<"Underflow!"<<endl;
49. return 0;
50. }
51. else
52. {
53. ST\* ptr = start;
54. int value = ptr->x;
55. start = start->next;
56. delete ptr;  // Free the popped node
57. cout << "Item Popped!" << endl;
58. LogFile("Popped value " + to\_string(value));
59. return value;
60. }
61. LogFile("Popped value ");
62. }
63. void traverse(ST\*& Start)
64. {
65. ST\*ptr=Start;
66. if (Start==NULL)
67. {
68. cout<<"Underflow!"<<endl;
69. return;
70. }
71. else{
72. cout<<"Traversed List will be:"<<endl;
73. while (ptr!=NULL)
74. {
75. cout<<ptr->x<<" ";
76. ptr=ptr->next;
77. }
78. return;
79. }
80. LogFile("Traversed Stack List");
81. }
82. int peek(ST\*&start)
83. {
84. if (start==NULL)
85. {
86. cout<<"Underflow!"<<endl;
87. return 0;
88. }
89. else
90. {
91. return start->x;
92. }
93. LogFile("Peeked first Node");
94. }
95. int main()
96. {
97. LogFile("Open File");
98. ST \*ptr=NULL;
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. case 1:
111. LogFile("Call Push Function");
112. {
113. int value;
114. cout<<"Enter Value to Push: "; cin>>value;
115. push(ptr,value);
116. }
117. break;
118. case 2:
119. LogFile("Call Pop Function");
120. cout<<pop(ptr)<<endl;
121. break;
122. case 3:
123. LogFile("Call Traverse Function");
124. traverse(ptr);
125. break;
126. case 4:
127. LogFile("Call Peek Function");
128. cout<<peek(ptr)<<endl;
129. break;
130. case 5:
131. LogFile("Close File");
132. while (ptr != NULL) {
133. pop(ptr);  // Free all remaining nodes
134. }
135. return 0;
136. break;
137. default:
138. LogFile("Force to close File");
139. return 0;
140. break;
141. }
142. }
143. }

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

1. **Doubly Linked Implementation of Stack**
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. #include <cmath>
6. #include <ctime>
7. #include <bitset>
8. using namespace std;
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. void push(ST\*&start, int& val)
29. {
30. ST\* newNode = new ST;
31. newNode->x = val;
32. newNode->next = NULL;
33. newNode->prev = NULL;
34. if (start == NULL)
35. {
36. start = newNode;  // Stack is empty, so newNode becomes the start
37. }
38. else
39. {
40. start->prev = newNode;  // Link the new node to the current start
41. newNode->next = start;  // Link newNode's next to the old start
42. start = newNode;        // Update start to point to newNode
43. }
44. cout << "Item Pushed!" << endl;
45. LogFile("Pushed value " + to\_string(val));
46. }
47. int pop(ST\*& start)
48. {
49. if (start == NULL)
50. {
51. cout << "Underflow!" << endl;
52. return 0;
53. }
54. else
55. {
56. ST\* ptr = start;
57. int value = ptr->x;
58. start = start->next;  // Move start to the next node
59. if (start != NULL)
60. start->prev = NULL;  // Break the backward link for the new start
61. delete ptr;  // Free the popped node
62. cout << "Item Popped!" << endl;
63. LogFile("Popped value " + to\_string(value));
64. return value;
65. }
66. }
67. void traverse(ST\*& start)
68. {
69. ST\* ptr = start;
70. if (start == NULL)
71. {
72. cout << "Underflow!" << endl;
73. return;
74. }
75. else {
76. cout << "Traversed List: ";
77. while (ptr != NULL)
78. {
79. cout << ptr->x << " ";
80. ptr = ptr->next;
81. }
82. cout << endl;
83. }
84. LogFile("Traversed Stack List");
85. }
86. int peek(ST\*& start)
87. {
88. if (start == NULL)
89. {
90. cout << "Underflow!" << endl;
91. return 0;
92. }
93. else
94. {
95. cout << "Peeked Value: " << start->x << endl;
96. LogFile("Peeked value " + to\_string(start->x));
97. return start->x;
98. }
99. }
100. int main()
101. {
102. LogFile("Open File");
103. ST \*ptr = NULL;
104. int z;
105. while (true) {
106. cout << "\n\nWelcome to Stack Manager:" << endl;
107. cout << "Select Your Commands:" << endl;
108. cout << "1. Push an element into a Stack" << endl;
109. cout << "2. Pop an element from the Stack" << endl;
110. cout << "3. Traverse a Stack" << endl;
111. cout << "4. Peek Your Stack" << endl;
112. cout << "5. Exit Program." << endl;
113. cout << "Enter choice: "; cin >> z;
114. switch (z)
115. {
116. case 1:
117. LogFile("Call Push Function");
118. {
119. int value;
120. cout << "Enter Value to Push: "; cin >> value;
121. push(ptr, value);
122. }
123. break;
124. case 2:
125. LogFile("Call Pop Function");
126. cout << pop(ptr) << endl;
127. break;
128. case 3:
129. LogFile("Call Traverse Function");
130. traverse(ptr);
131. break;
132. case 4:
133. LogFile("Call Peek Function");
134. peek(ptr);
135. break;
136. case 5:
137. LogFile("Close File");
138. while (ptr != NULL) {
139. pop(ptr);  // Free all remaining nodes
140. }
141. return 0;
142. default:
143. LogFile("Force to close File");
144. return 0;
145. }
146. }
147. }

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:**
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. #include <cmath>
6. #include <ctime>
7. #include <bitset>
8. using namespace std;
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. }
21. }
22. const int maxsize=25;
23. void insertElement(int Q[],int info,int size,int &R, int &F)
24. {
25. // Overflow check
26. if (R==size-1)
27. {
28. cout<<"Overflow!"<<endl;
29. return;
30. }
31. //Empty queue check
32. if (F==-1||R==-1)
33. {
34. F=R=0;
35. Q[R]=info;
36. }
37. else{
38. ++R;
39. Q[R]=info;
40. for (int i = R; i > F; i--)
41. {
42. if (Q[i] < Q[i - 1])
43. {
44. swap(Q[i], Q[i - 1]);
45. }
46. }
47. }
49. LogFile("Inserted "+ to\_string(info)+" In Queue ");
50. return;
51. }
52. void deleteElement(int Q[], int &R, int &F, int data)
53. {
54. // Empty queue check
55. if (F == -1 || F > R)
56. {
57. cout << "Underflow!" << endl;
58. return;
59. }
60. else
61. {
62. int i = F;
63. bool found = false;
64. // Search for the element
65. for (; i <= R; i++)
66. {
67. if (Q[i] == data)
68. {
69. found = true;
70. break;
71. }
72. }
73. if (!found)
74. {
75. cout << "Element not found!" << endl;
76. return;
77. }
78. // Shift elements left to remove the found element
79. for (int j = i; j < R; j++)
80. {
81. Q[j] = Q[j + 1];
82. }
83. R--; // Reduce the rear index
84. if (R < F)
85. {
86. F = R = -1; // Queue becomes empty
87. }
88. }
89. LogFile("Deleted Element from Queue");
90. }
91. void Traverse(int Q[], int size, int R, int F)
92. {
93. if (F==-1||F>R)
94. {
95. cout<<"Underflow!"<<endl;
96. return;
97. }
98. int x=F;
99. cout<<"Elements of Queue will be: "<<endl;
100. while (x <= R)
101. {
102. cout<<Q[x]<<endl;
103. ++x;
104. }
105. LogFile("Traverse Queue");
106. }
107. int main()
108. {
109. LogFile("Open File");
110. int queue[maxsize],front=-1,rear=-1,z;
111. while (true)
112. {
113. cout<<"\n\nWelcome to Queue Manager\n\n"<<endl;
114. cout<<"Select Commands"<<endl;
115. cout<<"1. Insert element"<<endl;
116. cout<<"2. Delete element"<<endl;
117. cout<<"3. Traverse Queue"<<endl;
118. cout<<"4. Quit Program\n\n"<<endl;
119. cout<<"Choose Your Command: "; cin>>z;
120. switch (z)
121. {
122. case 1:
123. LogFile("Call InsertElement Function");
124. {
125. int info;
126. cout<<"Enter info you want to enter: ";cin>>info;
127. insertElement(queue,info,maxsize,rear,front);
128. }
129. break;
130. case 2:
131. LogFile("Call InsertElement Function");
132. {
133. int info;
134. cout<<"enter element to be deleted:"; cin>>info;
135. deleteElement(queue,rear,front,info);
136. }
137. break;
138. case 3:
139. LogFile("Call Traverse Function");
140. Traverse(queue,maxsize,rear,front);
141. break;
142. case 4:
143. return 0;
144. default:
145. break;
146. }
147. }
148. }

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

1. **Singly Linked Implementation of Queue**
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. #include <cmath>
6. #include <ctime>
7. #include <bitset>
8. using namespace std;
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. struct Node
22. {
23. int num;
24. Node\* next;
25. };
26. typedef Node Q;
27. // Insert element in the queue
28. void insertElement(Q\*& F, Q\*& R, int data)
29. {
30. Q\* NewNode = new Q;
31. NewNode->num = data;
32. NewNode->next = NULL;
33. // If the queue is empty, set front and rear
34. if (F == NULL && R == NULL){
35. F = R = NewNode;
36. }
37. else if (data < F->num)  // Insert at the front
38. {
39. NewNode->next = F;
40. F = NewNode;
41. }
42. else // Insert in the sorted position{
43. Q\* temp = F;
44. Q\* prev = NULL;
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
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. }
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. }

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

1. **Doubly Linked Implementation of Queue**
2. #include <iostream>
3. #include <string>
4. #include <fstream>
5. #include <cmath>
6. #include <ctime>
7. #include <bitset>
8. using namespace std;
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. // Doubly Linked List Node Structure
22. struct Node
23. {
24. int num;
25. Node\* next;
26. Node\* prev;
27. };
28. typedef Node Q;
29. // Insert element in a sorted position
30. void insertElement(Q\*& F, Q\*& R, int data)
31. {
32. Q\* NewNode = new Q;
33. NewNode->num = data;
34. NewNode->next = NULL;
35. NewNode->prev = NULL;
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. // Case 2: Insertion at the front (smallest element)
45. if (data < F->num)
46. {
47. NewNode->next = F;
48. F->prev = NewNode;
49. F = NewNode;
50. }
51. else
52. {
53. // Traverse the list to find the correct position
54. while (temp != NULL && temp->num <= data)
55. {
56. temp = temp->next;
57. }
58. // Case 3: Insertion at the end (largest element)
59. if (temp == NULL)
60. {
61. NewNode->prev = R;
62. R->next = NewNode;
63. R = NewNode;
64. }
65. else
66. {
67. // Case 4: Insertion in the middle
68. NewNode->next = temp;
69. NewNode->prev = temp->prev;
70. temp->prev->next = NewNode;
71. temp->prev = NewNode;
72. }
73. }
74. }
75. LogFile("Inserted element in sorted order: " + to\_string(data));
76. }
77. // Delete element with a specific value
78. void deleteByValue(Q\*& F, Q\*& R, int data)
79. {
80. if (F == NULL)
81. {
82. cout << "Queue is empty, cannot delete." << endl;
83. LogFile("Queue underflow - Delete by value failed");
84. return;
85. }
86. Q\* temp = F;
87. // Traverse the list to find the node with the given value
88. while (temp != NULL && temp->num != data)
89. {
90. temp = temp->next;
91. }
92. if (temp == NULL)     // Case 1: Element not found
93. {
94. cout << "Element " << data << " not found in the queue." << endl;
95. LogFile("Element not found in the queue - Delete failed");
96. return;
97. }
98. // Case 2: Deleting the first node
99. if (temp == F)
100. {
101. F = F->next;
102. if (F != NULL)
103. {
104. F->prev = NULL;
105. }
106. else
107. {
108. R = NULL; // The list becomes empty
109. }
110. }
112. else if (temp == R) // Case 3: Deleting the last node
113. {
114. R = R->prev;
115. R->next = NULL;
116. }
117. else// Case 4: Deleting from the middle
118. {
119. temp->prev->next = temp->next;
120. temp->next->prev = temp->prev;
121. }
122. cout << "Deleted element: " << temp->num << endl;
123. LogFile("Deleted element: " + to\_string(temp->num));
124. delete temp;
125. }
126. void Traverse(Q\*& F) // Traverse the queue from front to rear
127. {
128. if (F == NULL)
129. {
130. cout << "Queue is empty" << endl;
131. LogFile("Queue is empty - Traverse");
132. return;
133. }
134. Q\* temp = F;
135. while (temp != NULL)
136. {
137. cout << temp->num << " ";
138. temp = temp->next;
139. }
140. cout << endl;
141. LogFile("Traversed queue");
142. }
143. void TraverseReverse(Q\*& R) // Traverse the queue in reverse from rear to front
144. {
145. if (R == NULL)
146. {
147. cout << "Queue is empty" << endl;
148. LogFile("Queue is empty - Reverse Traverse");
149. return;
150. }
151. Q\* temp = R;
152. while (temp != NULL)
153. {
154. cout << temp->num << " ";
155. temp = temp->prev;
156. }
157. cout << endl;
158. LogFile("Traversed queue in reverse");
159. }
160. int main()
161. {
162. int z;
163. int data;
164. Q\* front = NULL;
165. Q\* rear = NULL;
166. LogFile("Opened File");
167. while (true)
168. {
169. cout << "\n\nWelcome to Queue Manager\n\n" << endl;
170. cout << "Select Commands" << endl;
171. cout << "1. Insert element (sorted)" << endl;
172. cout << "2. Delete element by value" << endl;
173. cout << "3. Traverse and Reverse Traverse Queue" << endl;
174. cout << "4. Quit Program\n\n" << endl;
175. cout << "Choose Your Command: ";
176. cin >> z;
177. switch (z)
178. {
179. case 1:
180. cout << "Enter element to insert: ";
181. cin >> data;
182. insertElement(front, rear, data);
183. break;
184. case 2:
185. cout << "Enter element to delete: ";
186. cin >> data;
187. deleteByValue(front, rear, data);
188. break;
189. case 3:
190. cout << "Queue from front to rear: ";
191. Traverse(front);
192. cout << "Queue from rear to front: ";
193. TraverseReverse(rear);
194. break;
195. case 4:
196. LogFile("Exiting Program");
197. exit(0);
198. default:
199. cout << "Invalid command" << endl;
200. }
201. }
202. return 0;
203. }

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