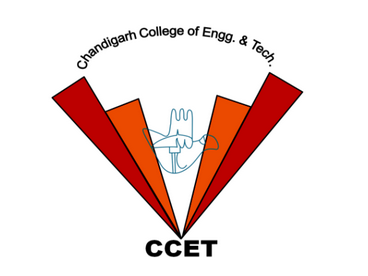
**CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)**

****

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 | 9 |
|  | Implementation of Queue | 17 |
|  | From a Static Data Structure (Array) |  |
|  | From a Singly Linked Data Structure | 19 |
|  | From a Doubly Linked Data Structure | 21 |
|  | CODE | 24 |

**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. }
20. }
21. const int SIZE=10;
22. bool IsFull(int size,int& Tos)
23. {
24. LogFile("Stack is Full");
25. return(Tos==size-1);
26. }
27. bool IsEmpty(int& Tos)
28. {
29. LogFile("Stack is Empty");
30. return(Tos==-1);
31. }
32. void Push(int Stack[],int size,int& Tos,int info)
33. {
34. // To check if Stack is full
35. LogFile("Pushed Value " +to\_string(info)+" to Stack");
36. if (IsFull(size,Tos))
37. {
38. cout<<"\nStack Overflow!"<<endl;
39. return;
40. }
41. else
42. Stack[++Tos]=info;
43. }
44. int pop(int Stack[],int &Tos)
45. {
46. LogFile("Popped Value from Stack");
47. // To check if Stack is Empty
48. if (IsEmpty(Tos))
49. {
50. 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. int info;
62. int tempTos = Tos; // Save the original Tos for restoring later
63. cout << "\nThe elements of Stack are:" << endl;
64. // Move elements from Stack to BStack and print them
65. while (!IsEmpty(Tos)) {
66. info = pop(Stack, Tos);
67. Push(BStack, size, BTos, info);
68. cout << info << endl;
69. }
70. // Restore elements from BStack back to Stack
71. while (!IsEmpty(BTos)) {
72. info = pop(BStack, BTos);
73. Push(Stack, size, Tos, info);
74. }
75. Tos = tempTos; // Restore original Tos
76. LogFile("traversing List");
77. }
78. int Peek(int Stack[],int& Tos)
79. {
80. LogFile("Peeked List");
81. if (IsEmpty(Tos))
82. {
83. cout<<"\nStack Underflow!"<<endl;
84. return 0;
85. }
86. return Stack[Tos];
87. }
88. int main()
89. {
90. LogFile("Open File");
91. int ToS=-1,BToS=-1;
92. int S[SIZE],BS[SIZE];
93. int z;
94. while (true){
95. cout<<"\n\nWelcome to Stack Manager:"<<endl;
96. cout<<"Select Your Commands:"<<endl;
97. cout<<"1. Push an element into a Stack"<<endl;
98. cout<<"2. Pop an element from the Stack"<<endl;
99. cout<<"3. Traverse a Stack"<<endl;
100. cout<<"4. Peek Your Stack"<<endl;
101. cout<<"5. Exit Program."<<endl;
102. cout<<"Enter choice:"; cin>>z;
103. switch (z)
104. {
105. case 1:
106. LogFile("Call Push Function");
107. {
108. int info;
109. cout<<"Enter info you want to push in Stack: "; cin>>info;
110. Push(S,SIZE,ToS,info);
111. }
112. break;
113. case 2:
114. LogFile("Call Pop Function");
115. pop(S,ToS);
116. break;
117. case 3:
118. LogFile("Call Traverse Function");
119. traverse(S,BS,SIZE,ToS,BToS);
120. break;
121. case 4:
122. LogFile("Call Peek Function");
123. cout<<Peek(S,ToS)<<endl<<endl;
124. break;
125. case 5:
126. LogFile("Close File");
127. return 0;
128. default:
129. LogFile("Force to close File");
130. cout<<"Entered Invalid Option."<<endl;
131. return 0;
132. break;
133. }
134. }
135. **Singly Linked Implementation of Stack**
136. #include <iostream>
137. #include <string>
138. #include <fstream>
139. #include <cmath>
140. #include <ctime>
141. #include <bitset>
142. using namespace std;
143. void LogFile(const string& event) {
144. ofstream BinFile("Machine\_Code\_Stack.txt",ios\_base::app);
145. if (BinFile.is\_open())
146. {
147. for (char c:event)
148. {
149. BinFile<<bitset<8>(c)<<" ";
150. }
151. BinFile<<endl;
152. BinFile.close();
153. }
155. }
156. struct Node{
157. int x;
158. Node \*next;
159. };
160. typedef Node ST;
161. void push(ST\*&start, int& val)
162. {
163. ST\*newNode=new ST;
164. newNode->x=val;
165. newNode->next=NULL;
166. //check if list is empty
167. if (start==NULL)
168. start=newNode;
169. else
170. {
171. newNode->next=start;
172. start=newNode;
173. }
174. cout<<"Item Pushed!"<<endl;
175. LogFile("Pushed value " + to\_string(val));
176. }
177. int pop(ST\*& start)
178. {
179. // check if list is empty
180. if (start==NULL)
181. {
182. cout<<"Underflow!"<<endl;
183. return 0;
184. }
185. else
186. {
187. ST\* ptr = start;
188. int value = ptr->x;
189. start = start->next;
190. delete ptr;  // Free the popped node
191. cout << "Item Popped!" << endl;
192. LogFile("Popped value " + to\_string(value));
193. return value;
194. }
195. LogFile("Popped value ");
196. }
197. void traverse(ST\*& Start)
198. {
199. ST\*ptr=Start;
200. if (Start==NULL)
201. {
202. cout<<"Underflow!"<<endl;
203. return;
204. }
205. else{
206. cout<<"Traversed List will be:"<<endl;
207. while (ptr!=NULL)
208. {
209. cout<<ptr->x<<" ";
210. ptr=ptr->next;
211. }
212. return;
213. }
214. LogFile("Traversed Stack List");
215. }
216. int peek(ST\*&start)
217. {
218. if (start==NULL)
219. {
220. cout<<"Underflow!"<<endl;
221. return 0;
222. }
223. else
224. {
225. return start->x;
226. }
227. LogFile("Peeked first Node");
228. }
229. int main()
230. {
231. LogFile("Open File");
232. ST \*ptr=NULL;
233. int z;
234. while (true){
235. cout<<"\n\nWelcome to Stack Manager:"<<endl;
236. cout<<"Select Your Commands:"<<endl;
237. cout<<"1. Push an element into a Stack"<<endl;
238. cout<<"2. Pop an element from the Stack"<<endl;
239. cout<<"3. Traverse a Stack"<<endl;
240. cout<<"4. Peek Your Stack"<<endl;
241. cout<<"5. Exit Program."<<endl;
242. cout<<"Enter choice:"; cin>>z;
243. switch (z){
244. case 1:
245. LogFile("Call Push Function");
246. {
247. int value;
248. cout<<"Enter Value to Push: "; cin>>value;
249. push(ptr,value);
250. }
251. break;
252. case 2:
253. LogFile("Call Pop Function");
254. cout<<pop(ptr)<<endl;
255. break;
256. case 3:
257. LogFile("Call Traverse Function");
258. traverse(ptr);
259. break;
260. case 4:
261. LogFile("Call Peek Function");
262. cout<<peek(ptr)<<endl;
263. break;
264. case 5:
265. LogFile("Close File");
266. while (ptr != NULL) {
267. pop(ptr);  // Free all remaining nodes
268. }
269. return 0;
270. break;
271. default:
272. LogFile("Force to close File");
273. return 0;
274. break;
275. }
276. }
277. }
278. **Doubly Linked Implementation of Stack**
279. #include <iostream>
280. #include <string>
281. #include <fstream>
282. #include <cmath>
283. #include <ctime>
284. #include <bitset>
285. using namespace std;
286. void LogFile(const string& event) {
287. ofstream BinFile("Machine\_Code\_Stack.txt", ios\_base::app);
288. if (BinFile.is\_open())
289. {
290. for (char c : event)
291. {
292. BinFile << bitset<8>(c) << " ";
293. }
294. BinFile << endl;
295. BinFile.close();
296. }
297. }
298. // Doubly linked list node structure
299. struct Node {
300. int x;
301. Node \*next;
302. Node \*prev;
303. };
304. typedef Node ST;
305. void push(ST\*&start, int& val)
306. {
307. ST\* newNode = new ST;
308. newNode->x = val;
309. newNode->next = NULL;
310. newNode->prev = NULL;
311. if (start == NULL)
312. {
313. start = newNode;  // Stack is empty, so newNode becomes the start
314. }
315. else
316. {
317. start->prev = newNode;  // Link the new node to the current start
318. newNode->next = start;  // Link newNode's next to the old start
319. start = newNode;        // Update start to point to newNode
320. }
321. cout << "Item Pushed!" << endl;
322. LogFile("Pushed value " + to\_string(val));
323. }
324. int pop(ST\*& start)
325. {
326. if (start == NULL)
327. {
328. cout << "Underflow!" << endl;
329. return 0;
330. }
331. else
332. {
333. ST\* ptr = start;
334. int value = ptr->x;
335. start = start->next;  // Move start to the next node
336. if (start != NULL)
337. start->prev = NULL;  // Break the backward link for the new start
338. delete ptr;  // Free the popped node
339. cout << "Item Popped!" << endl;
340. LogFile("Popped value " + to\_string(value));
341. return value;
342. }
343. void traverse(ST\*& start)
344. {
345. ST\* ptr = start;
346. if (start == NULL)
347. {
348. cout << "Underflow!" << endl;
349. return;
350. }
351. else {
352. cout << "Traversed List: ";
353. while (ptr != NULL)
354. {
355. cout << ptr->x << " ";
356. ptr = ptr->next;
357. }
358. cout << endl;
359. }
360. LogFile("Traversed Stack List");
361. }
362. int peek(ST\*& start)
363. {
364. if (start == NULL)
365. {
366. cout << "Underflow!" << endl;
367. return 0;
368. }
369. else
370. {
371. cout << "Peeked Value: " << start->x << endl;
372. LogFile("Peeked value " + to\_string(start->x));
373. return start->x;
374. }
375. }
376. int main()
377. {
378. LogFile("Open File");
379. ST \*ptr = NULL;
380. int z;
381. while (true) {
382. cout << "\n\nWelcome to Stack Manager:" << endl;
383. cout << "Select Your Commands:" << endl;
384. cout << "1. Push an element into a Stack" << endl;
385. cout << "2. Pop an element from the Stack" << endl;
386. cout << "3. Traverse a Stack" << endl;
387. cout << "4. Peek Your Stack" << endl;
388. cout << "5. Exit Program." << endl;
389. cout << "Enter choice: "; cin >> z;
390. switch (z)
391. {
392. case 1:
393. LogFile("Call Push Function");
394. {
395. int value;
396. cout << "Enter Value to Push: "; cin >> value;
397. push(ptr, value);
398. }
399. break;
400. case 2:
401. LogFile("Call Pop Function");
402. cout << pop(ptr) << endl;
403. break;
404. case 3:
405. LogFile("Call Traverse Function");
406. traverse(ptr);
407. break;
408. case 4:
409. LogFile("Call Peek Function");
410. peek(ptr);
411. break;
412. case 5:
413. LogFile("Close File");
414. while (ptr != NULL) {
415. pop(ptr);  // Free all remaining nodes
416. }
417. return 0;
418. default:
419. LogFile("Force to close File");
420. return 0;
421. }
422. }
423. }

**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. }
149. **Singly Linked Implementation of Queue**
150. #include <iostream>
151. #include <string>
152. #include <fstream>
153. #include <cmath>
154. #include <ctime>
155. #include <bitset>
156. using namespace std;
157. void LogFile(const string& event) {
158. ofstream BinFile("Machine\_Code\_Queue.txt", ios\_base::app);
159. if (BinFile.is\_open())
160. {
161. for (char c : event)
162. {
163. BinFile << bitset<8>(c) << " ";
164. }
165. BinFile << endl;
166. BinFile.close();
167. }
168. }
169. struct Node
170. {
171. int num;
172. Node\* next;
173. };
174. typedef Node Q;
175. // Insert element in the queue
176. void insertElement(Q\*& F, Q\*& R, int data)
177. {
178. Q\* NewNode = new Q;
179. NewNode->num = data;
180. NewNode->next = NULL;
181. // If the queue is empty, set front and rear
182. if (F == NULL && R == NULL){
183. F = R = NewNode;
184. }
185. else if (data < F->num)  // Insert at the front
186. {
187. NewNode->next = F;
188. F = NewNode;
189. }
190. else // Insert in the sorted position{
191. Q\* temp = F;
192. Q\* prev = NULL;
193. // Traverse to find the correct position
194. while (temp != NULL && temp->num <= data)
195. {
196. prev = temp;
197. temp = temp->next;
198. }
199. // Insert in the middle or at the end
200. prev->next = NewNode;
201. NewNode->next = temp;
202. // If inserted at the end, update the rear pointer
203. if (temp == NULL)
204. {
205. R = NewNode;
206. }
207. }
208. LogFile("Inserted element: " + to\_string(data));
209. }
210. // Delete element from the queue
211. void deleteElement(Q\*& F, Q\*& R, int data)
212. {
213. if (F == NULL)
214. {
215. cout << "Queue underflow" << endl;
216. LogFile("Queue underflow - Delete failed");
217. return;
218. }
219. Q\* temp = F;
220. Q\* prev = NULL;
221. // If the element to be deleted is the first element
222. if (F->num == data)
223. {
224. F = F->next;
225. // If it was the only element in the queue
226. if (F == NULL)
227. {
228. R = NULL;  // The queue is now empty
229. }
230. cout << "Deleted element: " << temp->num << endl;
231. LogFile("Deleted element: " + to\_string(temp->num));
232. delete temp;
233. return;
234. }
235. // Traverse to find the element to delete
236. while (temp != NULL && temp->num != data)
237. {
238. prev = temp;
239. temp = temp->next;
240. }
241. if (temp == NULL) // Element not found
242. {
243. cout << "Element not found in the queue" << endl;
244. LogFile("Element not found in the queue - Delete failed");
245. return;
246. }
247. // Unlink the node from the list
248. prev->next = temp->next;
249. // If the node to be deleted is the last node, update the rear pointer
250. if (temp == R)
251. R = prev;
252. cout << "Deleted element: " << temp->num << endl;
253. LogFile("Deleted element: " + to\_string(temp->num));
254. delete temp;
255. }
256. // Traverse the queue and print elements
257. void Traverse(Q\* F){
258. if (F == NULL){
259. cout << "Queue is empty" << endl;
260. LogFile("Queue is empty - Traverse");
261. return;
262. }
263. cout << "Queue elements: ";
264. Q\* temp = F;
265. while (temp != NULL){
266. cout << temp->num << " ";
267. temp = temp->next;
268. }
269. cout << endl;
270. LogFile("Traversed queue");
271. }
272. int main()
273. {
274. int z;
275. int data;
276. Q\* front = NULL;
277. Q\* rear = NULL;
278. LogFile("Opened File");
279. while (true){
280. cout << "\n\nWelcome to Queue Manager\n\n" << endl;
281. cout << "Select Commands" << endl;
282. cout << "1. Insert element" << endl;
283. cout << "2. Delete element" << endl;
284. cout << "3. Traverse Queue" << endl;
285. cout << "4. Quit Program\n\n" << endl;
286. cout << "Choose Your Command: ";
287. cin >> z;
288. switch (z){
289. case 1:
290. cout << "Enter element to insert: ";
291. cin >> data;
292. insertElement(front, rear, data);
293. break;
294. case 2:
295. cout << "Enter element to delete: ";
296. cin >> data;
297. deleteElement(front, rear,data);
298. break;
299. case 3:
300. Traverse(front);
301. break;
302. case 4:
303. LogFile("Exiting Program");
304. exit(0);
305. default:
306. cout << "Invalid command" << endl;
307. }
308. }
309. return 0;
310. }
311. **Doubly Linked Implementation of Queue**
312. #include <iostream>
313. #include <string>
314. #include <fstream>
315. #include <cmath>
316. #include <ctime>
317. #include <bitset>
318. using namespace std;
319. void LogFile(const string& event) {
320. ofstream BinFile("Machine\_Code\_Queue.txt", ios\_base::app);
321. if (BinFile.is\_open())
322. {
323. for (char c : event)
324. {
325. BinFile << bitset<8>(c) << " ";
326. }
327. BinFile << endl;
328. BinFile.close();
329. }
330. }
331. // Doubly Linked List Node Structure
332. struct Node
333. {
334. int num;
335. Node\* next;
336. Node\* prev;
337. };
338. typedef Node Q;
339. // Insert element in a sorted position
340. void insertElement(Q\*& F, Q\*& R, int data)
341. {
342. Q\* NewNode = new Q;
343. NewNode->num = data;
344. NewNode->next = NULL;
345. NewNode->prev = NULL;
346. // Case 1: If the list is empty
347. if (F == NULL && R == NULL)
348. {
349. F = R = NewNode;
350. }
351. else
352. {
353. Q\* temp = F;
354. // Case 2: Insertion at the front (smallest element)
355. if (data < F->num)
356. {
357. NewNode->next = F;
358. F->prev = NewNode;
359. F = NewNode;
360. }
361. else
362. {
363. // Traverse the list to find the correct position
364. while (temp != NULL && temp->num <= data)
365. {
366. temp = temp->next;
367. }
368. // Case 3: Insertion at the end (largest element)
369. if (temp == NULL)
370. {
371. NewNode->prev = R;
372. R->next = NewNode;
373. R = NewNode;
374. }
375. else
376. {
377. // Case 4: Insertion in the middle
378. NewNode->next = temp;
379. NewNode->prev = temp->prev;
380. temp->prev->next = NewNode;
381. temp->prev = NewNode;
382. }
383. }
384. }
385. LogFile("Inserted element in sorted order: " + to\_string(data));
386. }
387. // Delete element with a specific value
388. void deleteByValue(Q\*& F, Q\*& R, int data)
389. {
390. if (F == NULL)
391. {
392. cout << "Queue is empty, cannot delete." << endl;
393. LogFile("Queue underflow - Delete by value failed");
394. return;
395. }
396. Q\* temp = F;
397. // Traverse the list to find the node with the given value
398. while (temp != NULL && temp->num != data)
399. {
400. temp = temp->next;
401. }
402. if (temp == NULL)     // Case 1: Element not found
403. {
404. cout << "Element " << data << " not found in the queue." << endl;
405. LogFile("Element not found in the queue - Delete failed");
406. return;
407. }
408. // Case 2: Deleting the first node
409. if (temp == F)
410. {
411. F = F->next;
412. if (F != NULL)
413. {
414. F->prev = NULL;
415. }
416. else
417. {
418. R = NULL; // The list becomes empty
419. }
420. }
422. else if (temp == R) // Case 3: Deleting the last node
423. {
424. R = R->prev;
425. R->next = NULL;
426. }
427. else// Case 4: Deleting from the middle
428. {
429. temp->prev->next = temp->next;
430. temp->next->prev = temp->prev;
431. }
432. cout << "Deleted element: " << temp->num << endl;
433. LogFile("Deleted element: " + to\_string(temp->num));
434. delete temp;
435. }
436. void Traverse(Q\*& F) // Traverse the queue from front to rear
437. {
438. if (F == NULL)
439. {
440. cout << "Queue is empty" << endl;
441. LogFile("Queue is empty - Traverse");
442. return;
443. }
444. Q\* temp = F;
445. while (temp != NULL)
446. {
447. cout << temp->num << " ";
448. temp = temp->next;
449. }
450. cout << endl;
451. LogFile("Traversed queue");
452. }
453. void TraverseReverse(Q\*& R) // Traverse the queue in reverse from rear to front
454. {
455. if (R == NULL)
456. {
457. cout << "Queue is empty" << endl;
458. LogFile("Queue is empty - Reverse Traverse");
459. return;
460. }
461. Q\* temp = R;
462. while (temp != NULL)
463. {
464. cout << temp->num << " ";
465. temp = temp->prev;
466. }
467. cout << endl;
468. LogFile("Traversed queue in reverse");
469. }
470. int main()
471. {
472. int z;
473. int data;
474. Q\* front = NULL;
475. Q\* rear = NULL;
476. LogFile("Opened File");
477. while (true)
478. {
479. cout << "\n\nWelcome to Queue Manager\n\n" << endl;
480. cout << "Select Commands" << endl;
481. cout << "1. Insert element (sorted)" << endl;
482. cout << "2. Delete element by value" << endl;
483. cout << "3. Traverse and Reverse Traverse Queue" << endl;
484. cout << "4. Quit Program\n\n" << endl;
485. cout << "Choose Your Command: ";
486. cin >> z;
487. switch (z)
488. {
489. case 1:
490. cout << "Enter element to insert: ";
491. cin >> data;
492. insertElement(front, rear, data);
493. break;
494. case 2:
495. cout << "Enter element to delete: ";
496. cin >> data;
497. deleteByValue(front, rear, data);
498. break;
499. case 3:
500. cout << "Queue from front to rear: ";
501. Traverse(front);
502. cout << "Queue from rear to front: ";
503. TraverseReverse(rear);
504. break;
505. case 4:
506. LogFile("Exiting Program");
507. exit(0);
508. default:
509. cout << "Invalid command" << endl;
510. }
511. }
512. return 0;
513. }