1. */\**
2. *\* C++ Program to Implement Circular Doubly Linked List*
3. *\*/*
4. #include<iostream>
5. #include<cstdio>
6. #include<cstdlib>
7. using namespace std;
9. */\**
10. *\* Node Declaration*
11. *\*/*
12. struct node
13. {
14. int info;
15. struct node \*next;
16. struct node \*prev;
17. }\*start, \*last;
18. int counter = 0;
19. */\**
20. *\* Class Declaration*
21. *\*/*
22. class double\_clist
23. {
24. public:
25. node \*create\_node(int);
26. void insert\_begin();
27. void insert\_last();
28. void insert\_pos();
29. void delete\_pos();
30. void search();
31. void update();
32. void display();
33. void reverse();
34. void sort();
35. double\_clist()
36. {
37. start = NULL;
38. last = NULL;
39. }
40. };
42. */\**
43. *\* Main: Contains Menu*
44. *\*/*
45. int main()
46. {
47. int choice;
48. double\_clist cdl;
49. while (1)
50. {
51. cout<<"**\n**-------------------------------"<<endl;
52. cout<<"Operations on Doubly Circular linked list"<<endl;
53. cout<<"**\n**-------------------------------"<<endl;
54. cout<<"1.Insert at Beginning"<<endl;
55. cout<<"2.Insert at Last"<<endl;
56. cout<<"3.Insert at Position"<<endl;
57. cout<<"4.Delete at Position"<<endl;
58. cout<<"5.Update Node"<<endl;
59. cout<<"6.Search Element"<<endl;
60. cout<<"7.Sort"<<endl;
61. cout<<"8.Display List"<<endl;
62. cout<<"9.Reverse List"<<endl;
63. cout<<"10.Exit"<<endl;
64. cout<<"Enter your choice : ";
65. cin>>choice;
66. switch(choice)
67. {
68. case 1:
69. cdl.insert\_begin();
70. break;
71. case 2:
72. cdl.insert\_last();
73. break;
74. case 3:
75. cdl.insert\_pos();
76. break;
77. case 4:
78. cdl.delete\_pos();
79. break;
80. case 5:
81. cdl.update();
82. break;
83. case 6:
84. cdl.search();
85. break;
86. case 7:
87. cdl.sort();
88. break;
89. case 8:
90. cdl.display();
91. break;
92. case 9:
93. cdl.reverse();
94. break;
95. case 10:
96. exit(1);
97. default:
98. cout<<"Wrong choice"<<endl;
99. }
100. }
101. return 0;
102. }
104. */\**
105. *\*MEMORY ALLOCATED FOR NODE DYNAMICALLY*
106. *\*/*
107. node\* double\_clist::create\_node(int value)
108. {
109. counter++;
110. struct node \*temp;
111. temp = new(struct node);
112. temp->info = value;
113. temp->next = NULL;
114. temp->prev = NULL;
115. return temp;
116. }
117. */\**
118. *\*INSERTS ELEMENT AT BEGINNING*
119. *\*/*
120. void double\_clist::insert\_begin()
121. {
122. int value;
123. cout<<endl<<"Enter the element to be inserted: ";
124. cin>>value;
125. struct node \*temp;
126. temp = create\_node(value);
127. if (start == last && start == NULL)
128. {
129. cout<<"Element inserted in empty list"<<endl;
130. start = last = temp;
131. start->next = last->next = NULL;
132. start->prev = last->prev = NULL;
133. }
134. else
135. {
136. temp->next = start;
137. start->prev = temp;
138. start = temp;
139. start->prev = last;
140. last->next = start;
141. cout<<"Element inserted"<<endl;
142. }
143. }
145. */\**
146. *\*INSERTS ELEMNET AT LAST*
147. *\*/*
148. void double\_clist::insert\_last()
149. {
150. int value;
151. cout<<endl<<"Enter the element to be inserted: ";
152. cin>>value;
153. struct node \*temp;
154. temp = create\_node(value);
155. if (start == last && start == NULL)
156. {
157. cout<<"Element inserted in empty list"<<endl;
158. start = last = temp;
159. start->next = last->next = NULL;
160. start->prev = last->prev = NULL;
161. }
162. else
163. {
164. last->next = temp;
165. temp->prev = last;
166. last = temp;
167. start->prev = last;
168. last->next = start;
169. }
170. }
171. */\**
172. *\*INSERTS ELEMENT AT POSITION*
173. *\*/*
174. void double\_clist::insert\_pos()
175. {
176. int value, pos, i;
177. cout<<endl<<"Enter the element to be inserted: ";
178. cin>>value;
179. cout<<endl<<"Enter the postion of element inserted: ";
180. cin>>pos;
181. struct node \*temp, \*s, \*ptr;
182. temp = create\_node(value);
183. if (start == last && start == NULL)
184. {
185. if (pos == 1)
186. {
187. start = last = temp;
188. start->next = last->next = NULL;
189. start->prev = last->prev = NULL;
190. }
191. else
192. {
193. cout<<"Position out of range"<<endl;
194. counter--;
195. return;
196. }
197. }
198. else
199. {
200. if (counter < pos)
201. {
202. cout<<"Position out of range"<<endl;
203. counter--;
204. return;
205. }
206. s = start;
207. for (i = 1;i <= counter;i++)
208. {
209. ptr = s;
210. s = s->next;
211. if (i == pos - 1)
212. {
213. ptr->next = temp;
214. temp->prev = ptr;
215. temp->next = s;
216. s->prev = temp;
217. cout<<"Element inserted"<<endl;
218. break;
219. }
220. }
221. }
222. }
223. */\**
224. *\* Delete Node at Particular Position*
225. *\*/*
226. void double\_clist::delete\_pos()
227. {
228. int pos, i;
229. node \*ptr, \*s;
230. if (start == last && start == NULL)
231. {
232. cout<<"List is empty, nothing to delete"<<endl;
233. return;
234. }
235. cout<<endl<<"Enter the postion of element to be deleted: ";
236. cin>>pos;
237. if (counter < pos)
238. {
239. cout<<"Position out of range"<<endl;
240. return;
241. }
242. s = start;
243. if(pos == 1)
244. {
245. counter--;
246. last->next = s->next;
247. s->next->prev = last;
248. start = s->next;
249. free(s);
250. cout<<"Element Deleted"<<endl;
251. return;
252. }
253. for (i = 0;i < pos - 1;i++ )
254. {
255. s = s->next;
256. ptr = s->prev;
257. }
258. ptr->next = s->next;
259. s->next->prev = ptr;
260. if (pos == counter)
261. {
262. last = ptr;
263. }
264. counter--;
265. free(s);
266. cout<<"Element Deleted"<<endl;
267. }
268. */\**
269. *\* Update value of a particular node*
270. *\*/*
271. void double\_clist::update()
272. {
273. int value, i, pos;
274. if (start == last && start == NULL)
275. {
276. cout<<"The List is empty, nothing to update"<<endl;
277. return;
278. }
279. cout<<endl<<"Enter the postion of node to be updated: ";
280. cin>>pos;
281. cout<<"Enter the new value: ";
282. cin>>value;
283. struct node \*s;
284. if (counter < pos)
285. {
286. cout<<"Position out of range"<<endl;
287. return;
288. }
289. s = start;
290. if (pos == 1)
291. {
292. s->info = value;
293. cout<<"Node Updated"<<endl;
294. return;
295. }
296. for (i=0;i < pos - 1;i++)
297. {
298. s = s->next;
299. }
300. s->info = value;
301. cout<<"Node Updated"<<endl;
302. }
303. */\**
304. *\* Search Element in the list*
305. *\*/*
306. void double\_clist::search()
307. {
308. int pos = 0, value, i;
309. bool flag = false;
310. struct node \*s;
311. if (start == last && start == NULL)
312. {
313. cout<<"The List is empty, nothing to search"<<endl;
314. return;
315. }
316. cout<<endl<<"Enter the value to be searched: ";
317. cin>>value;
318. s = start;
319. for (i = 0;i < counter;i++)
320. {
321. pos++;
322. if (s->info == value)
323. {
324. cout<<"Element "<<value<<" found at position: "<<pos<<endl;
325. flag = true;
326. }
327. s = s->next;
328. }
329. if (!flag)
330. cout<<"Element not found in the list"<<endl;
331. }
332. */\**
333. *\* Sorting Doubly Circular Link List*
334. *\*/*
335. void double\_clist::sort()
336. {
337. struct node \*temp, \*s;
338. int value, i;
339. if (start == last && start == NULL)
340. {
341. cout<<"The List is empty, nothing to sort"<<endl;
342. return;
343. }
344. s = start;
345. for (i = 0;i < counter;i++)
346. {
347. temp = s->next;
348. while (temp != start)
349. {
350. if (s->info > temp->info)
351. {
352. value = s->info;
353. s->info = temp->info;
354. temp->info = value;
355. }
356. temp = temp->next;
357. }
358. s = s->next;
359. }
360. }
361. */\**
362. *\* Display Elements of the List*
363. *\*/*
364. void double\_clist::display()
365. {
366. int i;
367. struct node \*s;
368. if (start == last && start == NULL)
369. {
370. cout<<"The List is empty, nothing to display"<<endl;
371. return;
372. }
373. s = start;
374. for (i = 0;i < counter-1;i++)
375. {
376. cout<<s->info<<"<->";
377. s = s->next;
378. }
379. cout<<s->info<<endl;
380. }
381. */\**
382. *\* Reverse Doubly Circular Linked List*
383. *\*/*
384. void double\_clist::reverse()
385. {
386. if (start == last && start == NULL)
387. {
388. cout<<"The List is empty, nothing to reverse"<<endl;
389. return;
390. }
391. struct node \*p1, \*p2;
392. p1 = start;
393. p2 = p1->next;
394. p1->next = NULL;
395. p1->prev = p2;
396. while (p2 != start)
397. {
398. p2->prev = p2->next;
399. p2->next = p1;
400. p1 = p2;
401. p2 = p2->prev;
402. }
403. last = start;
404. start = p1;
405. cout<<"List Reversed"<<endl;
406. }