A sample C++ program to Solve Tower of Hanoi Problem using Stacks. This program will help you to develop the solutions for the problem given in Lab 6.

This C++ program displays the iterative solution to the Tower of Hanoi problem. Tower Of Hanoi consists of three rods and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.  
The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk may be moved at a time.  
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.  
3. No disk may be placed on top of a smaller disk.

Here is the source code of the C++ program to step by step disk transfer operations from one pole to another in a Tower Of Hanoi problem. This C++ program is successfully compiled and run on DevCpp, a C++ compiler. The program output is given below.

1. */\**
2. *\* C++ Program for Tower of Hanoi*
3. *\*/*
4. #include<stdio.h>
5. #include<conio.h>
6. #include<iostream>
7. #include<math.h>
8. using namespace std;
9. struct node1
10. {
11. int data1;
12. node1 \*next1;
13. }\*top1 = NULL, \*p1 = NULL, \*np1 = NULL;
14. struct node2
15. {
16. int data2;
17. node2 \*next2;
18. }\*top2 = NULL, \*p2 = NULL, \*np2 = NULL;
19. struct node3
20. {
21. int data3;
22. node3 \*next3;
23. }\*top3 = NULL, \*p3 = NULL, \*np3 = NULL;
24. void push1(int data)
25. {
26. np1 = new node1;
27. np1->data1 = data;
28. np1->next1 = NULL;
29. if (top1 == NULL)
30. {
31. top1 = np1;
32. }
33. else
34. {
35. np1->next1 = top1;
36. top1 = np1;
37. }
38. }
39. int pop1()
40. {
41. int b = 999;
42. if (top1 == NULL)
43. {
44. return b;
45. }
46. else
47. {
48. p1 = top1;
49. top1 = top1->next1;
50. return(p1->data1);
51. delete(p1);
52. }
53. }
54. void push2(int data)
55. {
56. np2 = new node2;
57. np2->data2 = data;
58. np2->next2 = NULL;
59. if (top2 == NULL)
60. {
61. top2 = np2;
62. }
63. else
64. {
65. np2->next2 = top2;
66. top2 = np2;
67. }
68. }
69. int pop2()
70. {
71. int b = 999;
72. if (top2 == NULL)
73. {
74. return b;
75. }
76. else
77. {
78. p2 = top2;
79. top2 = top2->next2;
80. return(p2->data2);
81. delete(p2);
82. }
83. }
84. void push3(int data)
85. {
86. np3 = new node3;
87. np3->data3 = data;
88. np3->next3 = NULL;
89. if (top3 == NULL)
90. {
91. top3 = np3;
92. }
93. else
94. {
95. np3->next3 = top3;
96. top3 = np3;
97. }
98. }
99. int pop3()
100. {
101. int b = 999;
102. if (top3 == NULL)
103. {
104. return b;
105. }
106. else
107. {
108. p3 = top3;
109. top3 = top3->next3;
110. return(p3->data3);
111. delete(p3);
112. }
113. }
114. int top\_of\_stack()
115. {
116. if (top1 != NULL && top1->data1 == 1 )
117. {
118. return 1;
119. }
120. else if (top2 != NULL && top2->data2 == 1)
121. {
122. return 2;
123. }
124. else if (top3 != NULL && top3->data3 == 1)
125. {
126. return 3;
127. }
128. }
129. void display1()
130. {
131. cout<<endl;
132. node1 \*p1;
133. p1 = top1;
134. cout<<"Tower1-> "<<"**\t**";
135. while (p1 != NULL)
136. {
137. cout<<p1->data1<<"**\t**";
138. p1 = p1->next1;
139. }
140. cout<<endl;
141. }
142. void display2()
143. {
144. node2 \*p2;
145. p2 = top2;
146. cout<<"Tower2-> "<<"**\t**";
147. while (p2 != NULL)
148. {
149. cout<<p2->data2<<"**\t**";
150. p2 = p2->next2;
151. }
152. cout<<endl;
153. }
154. void display3()
155. {
156. node3 \*p3;
157. p3 = top3;
158. cout<<"Tower3-> "<<"**\t**";
159. while (p3 != NULL)
160. {
161. cout<<p3->data3<<"**\t**";
162. p3 = p3->next3;
163. }
164. cout<<endl;
165. cout<<endl;
166. }
167. void toh(int n)
168. {
169. int i, x, a, b;
170. for (i = 0; i < (pow(2,n)); i++)
171. {
172. display1();
173. display2();
174. display3();
175. x = top\_of\_stack();
176. if (i % 2 == 0)
177. {
178. if (x == 1)
179. {
180. push3(pop1());
181. }
182. else if (x == 2)
183. {
184. push1(pop2());
185. }
186. else if (x == 3)
187. {
188. push2(pop3());
189. }
190. }
191. else
192. {
193. if (x == 1)
194. {
195. a = pop2();
196. b = pop3();
197. if (a < b && b != 999)
198. {
199. push3(b);
200. push3(a);
201. }
202. else if (a > b && a != 999)
203. {
204. push2(a);
205. push2(b);
206. }
207. else if (b == 999)
208. {
209. push3(a);
210. }
211. else if (a == 999)
212. {
213. push2(b);
214. }
215. }
216. else if (x == 2)
217. {
218. a = pop1();
219. b = pop3();
220. if (a < b && b != 999)
221. {
222. push3(b);
223. push3(a);
224. }
225. else if (a > b && a != 999)
226. {
227. push1(a);
228. push1(b);
229. }
230. else if (b == 999)
231. {
232. push3(a);
233. }
234. else if (a == 999)
235. {
236. push1(b);
237. }
238. }
239. else if (x == 3)
240. {
241. a = pop1();
242. b = pop2();
243. if (a < b && b != 999)
244. {
245. push2(b);
246. push2(a);
247. }
248. else if (a > b && a != 999)
249. {
250. push1(a);
251. push1(b);
252. }
253. else if (b == 999)
254. {
255. push2(a);
256. }
257. else if (a == 999)
258. {
259. push1(b);
260. }
261. }
262. }
263. }
264. }
265. int main()
266. {
267. int n, i;
268. cout<<"enter the number of disks**\n**";
269. cin>>n;
270. for (i = n; i >= 1; i--)
271. {
272. push1(i);
273. }
274. toh(n);
275. getch();
276. }

Output

enter the number of disks

5

Tower1-> 1 2 3 4 5

Tower2->

Tower3->

Tower1-> 2 3 4 5

Tower2->

Tower3-> 1

Tower1-> 3 4 5

Tower2-> 2

Tower3-> 1

Tower1-> 3 4 5

Tower2-> 1 2

Tower3->

Tower1-> 4 5

Tower2-> 1 2

Tower3-> 3

Tower1-> 1 4 5

Tower2-> 2

Tower3-> 3

Tower1-> 1 4 5

Tower2->

Tower3-> 2 3

Tower1-> 4 5

Tower2->

Tower3-> 1 2 3

Tower1-> 5

Tower2-> 4

Tower3-> 1 2 3

Tower1-> 5

Tower2-> 1 4

Tower3-> 2 3

Tower1-> 2 5

Tower2-> 1 4

Tower3-> 3

Tower1-> 1 2 5

Tower2-> 4

Tower3-> 3

Tower1-> 1 2 5

Tower2-> 3 4

Tower3->

Tower1-> 2 5

Tower2-> 3 4

Tower3-> 1

Tower1-> 5

Tower2-> 2 3 4

Tower3-> 1

Tower1-> 5

Tower2-> 1 2 3 4

Tower3->

Tower1->

Tower2-> 1 2 3 4

Tower3-> 5

Tower1-> 1

Tower2-> 2 3 4

Tower3-> 5

Tower1-> 1

Tower2-> 3 4

Tower3-> 2 5

Tower1->

Tower2-> 3 4

Tower3-> 1 2 5

Tower1-> 3

Tower2-> 4

Tower3-> 1 2 5

Tower1-> 3

Tower2-> 1 4

Tower3-> 2 5

Tower1-> 2 3

Tower2-> 1 4

Tower3-> 5

Tower1-> 1 2 3

Tower2-> 4

Tower3-> 5

Tower1-> 1 2 3

Tower2->

Tower3-> 4 5

Tower1-> 2 3

Tower2->

Tower3-> 1 4 5

Tower1-> 3

Tower2-> 2

Tower3-> 1 4 5

Tower1-> 3

Tower2-> 1 2

Tower3-> 4 5

Tower1->

Tower2-> 1 2

Tower3-> 3 4 5

Tower1-> 1

Tower2-> 2

Tower3-> 3 4 5

Tower1-> 1

Tower2->

Tower3-> 2 3 4 5

Tower1->

Tower2->

Tower3-> 1 2 3 4 5