**Interview Questions**

**Data-structure:** A **data structure** is a method of organizing data in a computer so that it can be used effectively.

**Asymptotic Notation:** Mathematical representation of it’s complexity.

**Linear Data-structure:** Organize the data sequential order.

Ex: Array, Stack, Queue, Linked list etc.

**Non-Linear Data-structure:** Organize the data random order.

Ex: Tree, Graph, Dictionary, Heap, Tries etc.

**Circular Linked List:** A circular linked list is a sequence of elements in which every element has a link to its next element in the sequence and the last element has a link to the first element.

**Double Linked List:** Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.

**Circular Queue:** A circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.

**Double Ended Queue:** Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

**Binary Tree:** A tree in which every node can have a maximum of two children is called Binary Tree.

**Complete Binary Tree:** A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.

**Binary search tree** is a binary tree in which a node have a value greater than all values in its left subtree and smaller than all values in its right subtree.

|  |  |
| --- | --- |
| Binary Tree | Binary Search Tree |
| Unordered | Ordered |
| Searching, Inserting, Deletion is slow.  Complexity O(n). | Searching, Inserting, Deletion is Fast.  Complexity O(logn) |
| Duplicate allow | Duplicate not allow |

|  |  |  |
| --- | --- | --- |
|  | Graph | Tree |
| Path | More than one path allowed between two node. | Only one path allowed. |
| Root | Don’t have any root node | Has exactly one root node. |
| Loops | Can have loops. | Loops are not allowed |
| Complexity | More | Less |
| Traversal Techniques | BFS,DFS | Pre-order, In-order, Post-order. |
| Number of edge | Not defined | n-1 |

**STACK**

1. Implement stack using array.
2. Implement stack using queue.
3. Implement stack using single queue.
4. #include<bits/stdc++.h>
5. **using** **namespace** std;
7. queue<**int**>q;
9. **void** push(**int** x){
10. **int** sz=q.size();
11. q.push(x);
13. **for**(**int** i=0;i<sz;i++){
14. q.push(q.front());
15. q.pop();
16. }
17. }
19. **void** pop(){
20. **if**(q.empty())
21. cout<<"Underflow"<<endl;
22. **else**{
23. q.pop();
24. }
25. }
27. **void** top(){
28. **if**(q.empty()){
29. cout<<"NULL"<<endl;
30. }
31. **else**{
32. cout<<q.front()<<endl;
33. }
34. }
36. **int** main()
37. {
38. **int** n;
39. **while**(1){
40. cin>>n;
41. **if**(n==1){
42. **int** val;
43. cin>>val;
44. push(val);
45. }
46. **else** **if**(n==2){
47. pop();
48. }
49. **else**{
50. top();
51. }
52. }
53. **return** 0;
54. }
55. Implement two stack in single array.
56. #include<bits/stdc++.h>
57. **using** **namespace** std;
59. **int** a[10];
60. **int** l1=-1,l2=10;
62. **void** push1(**int** x)
63. {
64. **if**(l2-l1<=1)
65. {
66. cout<<"Overflow"<<endl;
67. **return**;
68. }
70. l1++;
71. a[l1]=x;
72. }
74. **void** push2(**int** x)
75. {
76. **if**(l2-l1<=1)
77. {
78. cout<<"Overflow"<<endl;
79. **return**;
80. }
82. l2--;
83. a[l2]=x;
84. }
86. **void** pop1()
87. {
88. **if**(l1<0)
89. {
90. cout<<"underflow"<<endl;
91. **return**;
92. }
94. a[l1]=0;
95. l1--;
96. }
98. **void** pop2()
99. {
100. **if**(l2>=10)
101. {
102. cout<<"underflow"<<endl;
103. **return**;
104. }
106. a[l2]=0;
107. l2++;
108. }
110. **void** top1()
111. {
112. **if**(l1>=0)
113. {
114. cout<<"Top-1 : "<<a[l1]<<endl;
115. **return**;
116. }
117. cout<<"NULL"<<endl;
118. }
120. **void** top2()
121. {
122. **if**(l2<10)
123. {
124. cout<<"Top-1 : "<<a[l2]<<endl;
125. **return**;
126. }
127. cout<<"NULL"<<endl;
128. }
130. **void** print()
131. {
132. **for**(**int** i=0; i<10; i++)
133. {
134. cout<<a[i]<<" ";
135. }
136. cout<<endl;
137. }
139. **int** main()
140. {
141. **int** n;
142. **while**(1)
143. {
144. cin>>n;
145. **if**(n==1)
146. {
147. **int** x;
148. cin>>x;
149. push1(x);
150. }
151. **else** **if**(n==2)
152. {
153. pop1();
154. }
155. **else** **if**(n==3)
156. {
157. top1();
158. }
159. **else** **if**(n==4)
160. {
161. **int** x;
162. cin>>x;
163. push2(x);
164. }
165. **else** **if**(n==5)
166. {
167. pop2();
168. }
169. **else** **if**(n==6)
170. {
171. top2();
172. }
173. **else**
174. {
175. print();
176. }
177. }
178. **return** 0;
179. }
180. Implement stack using linked list.
181. #include<bits/stdc++.h>
182. **using** **namespace** std;
184. **struct** node{
185. **int** data;
186. node \*next;
187. };
189. node \*root=NULL;
191. **void** push(**int** x){
192. node\* newNode=**new** node();
193. newNode->data=x;
194. newNode->next=root;
195. root=newNode;
196. }
198. **void** pop(){
199. **if**(root==NULL){
200. **return**;
201. }
202. node \*temp=root;
203. root=root->next;
204. **delete**(temp);
205. }
207. **void** top(){
208. **if**(root==NULL){
209. cout<<"NULL"<<endl;
210. **return**;
211. }
212. cout<<root->data<<endl;
213. }
215. **int** main()
216. {
217. **int** n;
218. **while**(1){
219. cin>>n;
220. **if**(n==1){
221. **int** x;
222. cin>>x;
223. push(x);
224. }
225. **else** **if**(n==2){
226. pop();
227. }
228. **else**{
229. top();
230. }
231. }
232. **return** 0;
233. }
234. Get maximum value from stack.
235. Get minimum value from stack.