|  |  |  |
| --- | --- | --- |
| **Academic Year:** 2024-25 | **Year:** Third Year | **Term:** II |
| **PRN No.: 1012412079** | **Name: Ratnajeet Patil** | |
| **Subject:** DS2 | | |
| **Assignment No.**: 2 |  | |
| **Date:** |  | |

**Lab Assignment: 03**

**Title: Write a modular program to implement primitive operations on Min and Max Heap.**

**Code:**

1. import java.util.\*;

2. // Write a modular program to implement primitive operations on Min/Max Heap.

3.

4. import java.util.\*;

5.

6. class HeapStructure {

7.     int[] heap;

8.     int size;

9.     int maxSize;

10.     boolean isMinHeap;

11.

12.     HeapStructure(int maxSize, boolean isMinHeap) {

13.         this.maxSize = maxSize;

14.         this.size = 0;

15.         this.isMinHeap = isMinHeap;

16.         heap = new int[this.maxSize + 1];

17.         heap[0] = isMinHeap ? Integer.MIN\_VALUE : Integer.MAX\_VALUE;

18.     }

19.

20.     int parent(int pos) {

21.         return pos / 2;

22.     }

23.

24.     int leftChild(int pos) {

25.         return (2 \* pos);

26.     }

27.

28.     int rightChild(int pos) {

29.         return (2 \* pos) + 1;

30.     }

31.

32.     boolean isLeaf(int pos) {

33.         return pos >= (size / 2) && pos <= size;

34.     }

35.

36.     void swap(int fpos, int spos) {

37.         int tmp = heap[fpos];

38.         heap[fpos] = heap[spos];

39.         heap[spos] = tmp;

40.     }

41.

42.     void heapify(int pos) {

43.         if (!isLeaf(pos)) {

44.             if (isMinHeap) {

45.                 minHeapify(pos);

46.             } else {

47.                 maxHeapify(pos);

48.             }

49.         }

50.     }

51.

52.     void minHeapify(int pos) {

53.         if (!isLeaf(pos)) {

54.             int leftPos = leftChild(pos);

55.             int rightPos = rightChild(pos);

56.             int smallestPos = pos;

57.

58.             if (leftPos <= size && heap[leftPos] < heap[smallestPos]) {

59.                 smallestPos = leftPos;

60.             }

61.             if (rightPos <= size && heap[rightPos] < heap[smallestPos]) {

62.                 smallestPos = rightPos;

63.             }

64.

65.             if (smallestPos != pos) {

66.                 swap(pos, smallestPos);

67.                 minHeapify(smallestPos);

68.             }

69.         }

70.     }

71.

72.     void maxHeapify(int pos) {

73.         if (!isLeaf(pos)) {

74.             int leftPos = leftChild(pos);

75.             int rightPos = rightChild(pos);

76.             int largestPos = pos;

77.

78.             if (leftPos <= size && heap[leftPos] > heap[largestPos]) {

79.                 largestPos = leftPos;

80.             }

81.             if (rightPos <= size && heap[rightPos] > heap[largestPos]) {

82.                 largestPos = rightPos;

83.             }

84.

85.             if (largestPos != pos) {

86.                 swap(pos, largestPos);

87.                 maxHeapify(largestPos);

88.             }

89.         }

90.     }

91.

92.     void insert(int element) {

93.         if (size >= maxSize) {

94.             System.out.println("Heap is full!");

95.             return;

96.         }

97.

98.         heap[++size] = element;

99.         int current = size;

100.

101.         if (isMinHeap) {

102.             while (heap[current] < heap[parent(current)]) {

103.                 swap(current, parent(current));

104.                 current = parent(current);

105.             }

106.         } else {

107.             while (heap[current] > heap[parent(current)]) {

108.                 swap(current, parent(current));

109.                 current = parent(current);

110.             }

111.         }

112.     }

113.

114.     void print() {

115.         if (size == 0) {

116.             System.out.println("Heap is empty!");

117.             return;

118.         }

119.

120.         for (int i = 1; i <= size / 2; i++) {

121.             System.out.print(" PARENT : " + heap[i] + " LEFT CHILD : " + heap[2 \* i]);

122.             if (2 \* i + 1 <= size) {

123.                 System.out.print(" RIGHT CHILD :" + heap[2 \* i + 1]);

124.             }

125.             System.out.println();

126.         }

127.     }

128.

129.     void buildHeap() {

130.         for (int pos = (size / 2); pos >= 1; pos--) {

131.             heapify(pos);

132.         }

133.     }

134.

135.     int remove() {

136.         if (size == 0) {

137.             System.out.println("Heap is empty!");

138.             return -1;

139.         }

140.

141.         int popped = heap[1];

142.         heap[1] = heap[size--];

143.         heapify(1);

144.         return popped;

145.     }

146. }

147.

148. public class HeapImplementation {

149.     public static void main(String[] arg) {

150.         Scanner scanner = new Scanner(System.in);

151.         System.out.println("Enter max size of heap: ");

152.         int maxSize = scanner.nextInt();

153.

154.         System.out.println("Select type of heap:");

155.         System.out.println("1. Min Heap");

156.         System.out.println("2. Max Heap");

157.         int heapType = scanner.nextInt();

158.

159.         boolean isMinHeap = heapType == 1;

160.         HeapStructure heapObj = new HeapStructure(maxSize, isMinHeap);

161.

162.         int choice;

163.         do {

164.             System.out.println("\nHeap Operations:");

165.             System.out.println("1. Insert element");

166.             System.out.println("2. Remove top element");

167.             System.out.println("3. Print heap");

168.             System.out.println("4. Exit");

169.             System.out.print("Enter your choice: ");

170.

171.             choice = scanner.nextInt();

172.

173.             switch (choice) {

174.                 case 1:

175.                     System.out.print("Enter element to insert: ");

176.                     int element = scanner.nextInt();

177.                     heapObj.insert(element);

178.                     System.out.println("Element inserted successfully");

179.                     break;

180.

181.                 case 2:

182.                     int removed = heapObj.remove();

183.                     if (removed != -1) {

184.                         System.out.println("Removed element: " + removed);

185.                     }

186.                     break;

187.

188.                 case 3:

189.                     System.out.println("Current heap structure:");

190.                     heapObj.print();

191.                     break;

192.

193.                 case 4:

194.                     System.out.println("Exiting program...");

195.                     break;

196.

197.                 default:

198.                     System.out.println("Invalid choice! Please try again.");

199.             }

200.

201.         } while (choice != 4);

202.

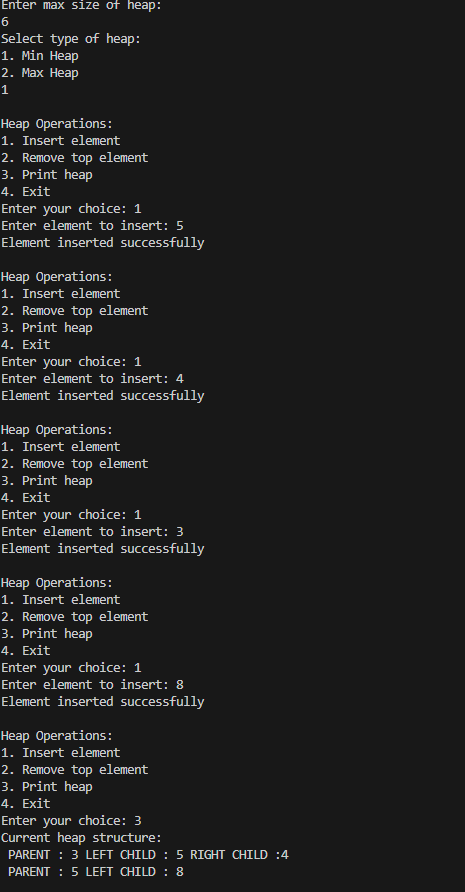
203.         scanner.close();

204.     }

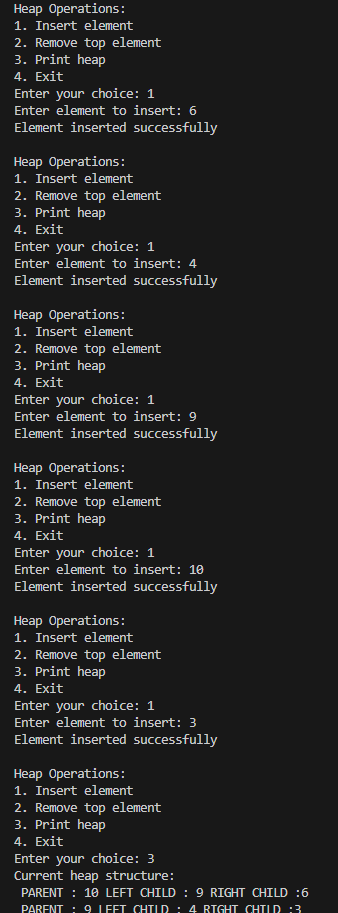
205. }

206.

**Min Heap:**

****

**Max Heap:**

****