21BCE7371

RADHA KRISHNA GARG

DSA LAB ASSIGNMENT-2

1.Write a Program to implement single linked list and its operations.

INPUT

```
// Linked list operations in Java
class LinkedList {
  Node head;
  // Create a node
  class Node {
    int data;
    Node next;
    Node(int d) {
      data = d;
      next = null;
    }
  }
  // Insert at the beginning
  public void insertAtBeginning(int new_data) {
    // insert the data
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
  }
  // Insert after a node
  public void insertAfter(Node prev_node, int new_data) {
    if (prev_node == null) {
```

```
    System.out.println("The given previous node cannot be null");
    return;
  }
  Node new_node = new Node(new_data);
  new_node.next = prev_node.next;
  prev_node.next = new_node;
}
```

// Insert at the end

```
public void insertAtEnd(int new_data) {
  Node new_node = new Node(new_data);
  if (head == null) {
    head = new Node(new_data);
    return;
  }
  new_node.next = null;
  Node last = head;
  while (last.next != null)
    last = last.next;
  last.next = new_node;
  return;
}
```

// Delete a node

```
void deleteNode(int position) {
  if (head == null)
    return;
  Node temp = head;
  if (position == 0) {
```

```
    head = temp.next;
    return;
  }

    // Find the key to be deleted
  for (int i = 0; temp != null && i < position - 1; i++)
    temp = temp.next;
  // If the key is not present
  if (temp == null || temp.next == null)
    return;
    // Remove the node
  Node next = temp.next.next;
  temp.next = next;
}
// Search a node
boolean search(Node head, int key) {
  Node current = head;
  while (current != null) {
    if (current.data == key)
      return true;
    current = current.next;
  }
  return false;
}
// Sort the linked list
 void sortLinkedList(Node head) {
  Node current = head;
  Node index = null;
```

```
    int temp;
    if (head == null) {
      return;
    } else {
      while (current != null) {
        // index points to the node next to current
        index = current.next;
        while (index != null) {
          if (current.data > index.data) {
            temp = current.data;
            current.data = index.data;
            index.data = temp;
          }
          index = index.next;
        }
        current = current.next;
      }
    }
  }
  // Print the linked list
  public void printList() {
    Node tnode = head;
    while (tnode != null) {
      System.out.print(tnode.data + " ");
      tnode = tnode.next;
    }
  }
public static void main(String[] args) {
    LinkedList llist = new LinkedList();
    llist.insertAtEnd(8);
```

```
    llist.insertAtBeginning(2);
    llist.insertAtBeginning(3);
    llist.insertAtBeginning(1);
    llist.insertAtBeginning(0);
    llist.insertAtEnd(9);
    llist.insertAtEnd(4);
    llist.insertAtEnd(7);
    llist.insertAfter(llist.head.next, 5);
    llist.insertAfter(llist.head.next, 6);
    System.out.println("Linked list: ");
    llist.printList();
    System.out.println("\nAfter deleting an element: ");
    llist.deleteNode(3);
    llist.printList();
    System.out.println();
    int item_to_find = 3;
    if (llist.search(llist.head, item_to_find))
      System.out.println(item_to_find + " is found");
    else
      System.out.println(item_to_find + " is not found");
    llist.sortLinkedList(llist.head);
    System.out.println("\nSorted List: ");
    llist.printList();
  }
}
```

```
// Linked list operations in Java
class LinkedList {
  Node head;
  // Create a node
  class Node {
    int data;
    Node next;
    Node(int d) {
      data = d;
      next = null;
    }
  }
  // Insert at the beginning
  public void insertAtBeginning(int new_data) {
    // insert the data
    Node new_node = new Node(new_data);
    new_node.next = head;
    head = new_node;
  }
  // Insert after a node
  public void insertAfter(Node prev_node, int new_data) {
    if (prev_node == null) {
      System.out.println("The given previous node cannot be null");
      return;
```

```
25        return;
26      }
27      Node new_node = new Node(new_data);
28      new_node.next = prev_node.next;
29      prev_node.next = new_node;
30    }
31
32
33  // Insert at the end
34    public void insertAtEnd(int new_data) {
35      Node new_node = new Node(new_data);
36      if (head == null) {
37        head = new Node(new_data);
38        return;
39      }
40      new_node.next = null;
41      Node last = head;
42      while (last.next != null)
43        last = last.next;
44      last.next = new_node;
45      return;
46    }
47
48  // Delete a node
49    void deleteNode(int position) {
50      if (head == null)
```

```
49    void deleteNode(int position) {
50      if (head == null)
51        return;
52      Node temp = head;
53      if (position == 0) {
54        head = temp.next;
55        return;
56      }
57      // Find the key to be deleted
58      for (int i = 0; temp != null && i < position - 1; i++)
59        temp = temp.next;
60      // If the key is not present
61      if (temp == null || temp.next == null)
62        return;
63      // Remove the node
64      Node next = temp.next.next;
65      temp.next = next;
66    }
67    // Search a node
68    boolean search(Node head, int key) {
69      Node current = head;
70      while (current != null) {
71        if (current.data == key)
72          return true;
73        current = current.next;
74      }
```

```
73          current = current.next;
74        }
75        return false;
76      }
77    // Sort the linked list
78      void sortLinkedList(Node head) {
79        Node current = head;
80        Node index = null;
81        int temp;
82        if (head == null) {
83          return;
84        } else {
85          while (current != null) {
86            // index points to the node next to current
87            index = current.next;
88            while (index != null) {
89              if (current.data > index.data) {
90                temp = current.data;
91                current.data = index.data;
92                index.data = temp;
93              }
94              index = index.next;
95            }
96            current = current.next;
97          }
98        }
```

```
97        }
98      }
99    }
100   // Print the linked list
101   public void printList() {
102     Node tnode = head;
103     while (tnode != null) {
104       System.out.print(tnode.data + " ");
105       tnode = tnode.next;
106     }
107   }
108 public static void main(String[] args) {
109     LinkedList llist = new LinkedList();
110     llist.insertAtEnd(8);
111     llist.insertAtBeginning(2);
112     llist.insertAtBeginning(3);
113     llist.insertAtBeginning(1);
114     llist.insertAtBeginning(0);
115     llist.insertAtEnd(9);
116     llist.insertAtEnd(4);
117     llist.insertAtEnd(7);
118     llist.insertAfter(llist.head.next, 5);
119     llist.insertAfter(llist.head.next, 6);
120     System.out.println("Linked list: ");
121     llist.printList();
122     System.out.println("\nAfter deleting an element: ");
```

Main.java

```
114     llist.insertAtBeginning(0);
115     llist.insertAtEnd(9);
116     llist.insertAtEnd(4);
117     llist.insertAtEnd(7);
118     llist.insertAfter(llist.head.next, 5);
119     llist.insertAfter(llist.head.next, 6);
120     System.out.println("Linked list: ");
121     llist.printList();
122     System.out.println("\nAfter deleting an element: ");
123     llist.deleteNode(3);
124     llist.printList();
125     System.out.println();
126     int item_to_find = 3;
127     if (llist.search(llist.head, item_to_find))
128       System.out.println(item_to_find + " is found");
129     else
130       System.out.println(item_to_find + " is not found");
131     llist.sortLinkedList(llist.head);
132     System.out.println("\nSorted List: ");
133     llist.printList();
134   }
135 }
```

OUTPUT

Output

```
java -cp /tmp/sUZ8iGeYxv LinkedList
Linked list: 0 1 6 5 3 2 8 9 4 7
After deleting an element:
0 1 6 3 2 8 9 4 7
3 is found

Sorted List:
0 1 2 3 4 6 7 8 9
```

Q2 Write a Program to implement stack operations using arrays.

INPUT

```
public class Stack_operation{
   private int arr[];
   private int top;
   private int capacity;

   Stack_operation(int size){
      arr = new int[size];
      capacity = size;
      top = -1;
   }
   public void push(int x){
      if(isFull()){
         System.out.println("Overflow");
         System.exit(1);
      }
      System.out.println("inserting"+x);
      arr[++top] = x;
   }
   public int pop(){
      if(isEmpty()){
         System.out.println("Stack empty");
         System.exit(1);
      }
      return arr[top--];
```

```
}
public int count(){
   return  top+1;
}
public boolean isEmpty(){
   return  top == -1;
}
public boolean isFull(){
   return top == capacity-1;
}
public void display(){
   for(int i = 0 ; i<= top ; i++){
      System.out.println(arr[i]);
   }
}
public int peek() {
   if(!this.isEmpty())
            return arr[top];
        else
        {
            System.out.println("Stack is Empty");
            return -1;
        }
}
public static void main(String [] args){
   Stack_operation stack = new Stack_operation(5);
   stack.push(1);
   stack.push(7);
   stack.push(9);
   stack.push(3);
```

```
        stack.push(16);
        System.out.println("\nStack :");
        stack.display();
        System.out.println("\nElement at the top of the stack is "+stack.peek());
        System.out.println("Total elements in the stack is "+stack.count());
        System.out.println("\nStack after popping out the last element :");
        stack.pop();
        stack.display();
        System.out.println("\nTotal elements in the stack after popping is "+stack.count());
        System.out.println("Element at the top of the stack is "+stack.peek());
    }
}
```

```java
public class Stack_operation{
    private int arr[];
    private int top;
    private int capacity;

    Stack_operation(int size){
        arr = new int[size];
        capacity = size;
        top = -1;
    }
    public void push(int x){
        if(isFull()){
            System.out.println("Overflow");
            System.exit(1);
        }
        System.out.println("inserting"+x);
        arr[++top] = x;
    }
    public int pop(){
        if(isEmpty()){
            System.out.println("Stack empty");
            System.exit(1);
        }
        return arr[top--];

```

Main.java

```
25
26      }
27      public int count(){
28          return  top+1;
29      }
30      public boolean isEmpty(){
31          return  top == -1;
32      }
33      public boolean isFull(){
34          return top == capacity-1;
35      }
36      public void display(){
37          for(int i = 0 ; i<= top ; i++){
38              System.out.println(arr[i]);
39          }
40      }
41      public int peek() {
42          if(!this.isEmpty())
43                          return arr[top];
44                  else
45                  {
46                          System.out.println("Stack is Empty");
47                          return -1;
48                  }
49      }
```

```
43                          return arr[top];
44                  else
45                  {
46                          System.out.println("Stack is Empty");
47                          return -1;
48                  }
49      }
50      public static void main(String [] args){
51          Stack_operation stack = new Stack_operation(5);
52          stack.push(1);
53          stack.push(7);
54          stack.push(9);
55          stack.push(3);
56          stack.push(16);
57          System.out.println("\nStack :");
58          stack.display();
59          System.out.println("\nElement at the top of the stack is "+stack.peek
                ());
60          System.out.println("Total elements in the stack is "+stack.count());
61          System.out.println("\nStack after popping out the last element :");
62          stack.pop();
63          stack.display();
64          System.out.println("\nTotal elements in the stack after popping is "
                +stack.count());
65          System.out.println("Element at the top of the stack is "+stack.peek());
66      }
67  }
```

OUTPUT

Output

```
java -cp /tmp/sUZ8iGeYxv Stack_operation
inserting1
inserting7
inserting9
inserting3
inserting16

Stack :
1
7
9
3
16
Element at the top of the stack is 16
Total elements in the stack is 5

Stack after popping out the last element :
1
7
9
3
Total elements in the stack after popping is 4
Element at the top of the stack is 3
```