**Anuj Koranga**

**22MCA0125**

**Linked List implementation in Java**

import java.util.Scanner;

class Node

{

int data;

Node next;

public Node()

{

data = -1;

next = null;

}

public Node(int x)

{

data = x;

next = null;

}

}

class LinkedList

{

Node head;

public LinkedList()

{

head = null;

}

void insertBeg(int x)

{

Node temp = new Node(x);

temp.next = head;

head = temp;

System.out.println("\nInserted at beginning: " + head.data);

}

void insertEnd(int x)

{

Node temp = new Node(x);

Node iter = head;

if(head == null)

head = temp;

else

{

while(iter.next != null)

iter = iter.next;

iter.next = temp;

}

System.out.println("\nInserted at end: " + temp.data);

}

void insertAtPos(int x, int pos)

{

if(pos < 1 || pos >= size())

{

System.out.println("Invalid position input!");

return;

}

Node temp = new Node(x);

int count = 2;

if(pos == 1)

{

temp.next = head;

head = temp;

}

else

{

Node iter = head;

while(iter.next != null)

{

if(pos == count)

{

temp.next = iter.next;

iter.next = temp;

break;

}

count++;

iter = iter.next;

}

if(pos == count)

iter.next = temp;

}

System.out.println("\nInserted at position " + pos + ": " + temp.data);

}

int size()

{

int count = 0;

Node iter = head;

while(iter != null)

{

count++;

iter = iter.next;

}

return count;

}

void deleteBeg()

{

if(head == null)

System.out.println("\nUnderflow");

else

{

System.out.println("\nDeleted from beginning: " + head.data);

head = head.next;

}

}

void deleteEnd()

{

if(head == null)

System.out.println("\nUnderflow");

else if(head.next == null)

{

head = null;

}

else

{

Node iter = head;

while(iter.next.next != null)

iter = iter.next;

System.out.println("\nDeleted from end: " + iter.next.data);

iter.next = null;

}

}

void deleteAtPos(int pos)

{

if(pos < 1 || pos > size())

{

System.out.println("Invalid position input!");

return;

}

int count = 2;

if(pos == 1)

{

deleteBeg();

return;

}

else

{

Node iter = head;

while(iter.next != null)

{

if(pos == count)

{

System.out.println("\nDeleted from position " + pos + ": " + iter.next.data);

iter.next = iter.next.next;

break;

}

count++;

iter = iter.next;

}

}

}

void display()

{

Node iter = head;

System.out.print("\nThe list is: ");

while(iter != null)

{

System.out.print(iter.data + " ");

iter = iter.next;

}

}

void reverse()

{

if(head == null)

System.out.println("\nList is empty!");

else if(head.next == null)

System.out.println("\nOnly one element in list!");

else

{

Node iter1, iter2;

iter1 = head.next;

iter2 = iter1.next;

iter1.next = head;

head.next = null;

head = iter1;

while(iter2 != null)

{

iter1 = iter2;

iter2 = iter2.next;

iter1.next = head;

head = iter1;

}

System.out.println("\nList is reversed!");

}

}

}

class Main

{

public static void main(String[] args)

{

Scanner s = new Scanner(System.in);

LinkedList l = new LinkedList();

int num, ch, pos;

while(true)

{

System.out.println("\n\nMenu:\n\n1) Insert at Beginning\n2) Insert at End\n3) Insert at Position\n4) Delete from Beginning\n5) Delete from End\n6) Delete from Position\n7) Reverse List\n8) Display List\n9) Size of List\n0) Exit\n\n");

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

ch = s.nextInt();

switch(ch)

{

case 1: System.out.print("\nEnter data: ");

num = s.nextInt();

l.insertBeg(num);

break;

case 2: System.out.print("\nEnter data: ");

num = s.nextInt();

l.insertEnd(num);

break;

case 3: System.out.print("\nEnter data: ");

num = s.nextInt();

System.out.print("\nEnter position: ");

pos = s.nextInt();

l.insertAtPos(num, pos);

break;

case 4: l.deleteBeg();

break;

case 5: l.deleteEnd();

break;

case 6: System.out.print("\nEnter position: ");

pos = s.nextInt();

l.deleteAtPos(pos);

break;

case 7: l.reverse();

break;

case 8: l.display();

break;

case 9: System.out.print("\nSize of list is: " + l.size());

break;

case 0: System.exit(0);

}

}

}

}

**Stack using Array**

import java.util.Scanner;

class Stack

{

int arr[], size;

int tos;

public Stack()

{

size = 100;

arr = new int[size];

tos = -1;

}

public Stack(int n)

{

size = n;

arr = new int[size];

tos = -1;

}

void push(int x)

{

if(isFull())

System.out.println("\nStack Overflow!");

else

arr[++tos] = x;

}

int pop()

{

return arr[tos--];

}

int peek()

{

return arr[tos];

}

void display()

{

System.out.println("\nThe stack is: ");

for(int i = tos; i >= 0; i--)

System.out.print(arr[i] + " ");

}

boolean isEmpty()

{

return tos == -1;

}

boolean isFull()

{

return tos == size-1;

}

}

class Main

{

public static void main(String[] args)

{

Scanner s = new Scanner(System.in);

int num, ch, n;

System.out.print("\nEnter size of array: ");

n = s.nextInt();

Stack stk = new Stack(n);

while(true)

{

System.out.println("\n\nMenu:\n\n1) Push\n2) Pop\n3) Peek\n4) isEmpty\n5) isFull\n6) Display Stack\n7) Exit\n\n");

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

ch = s.nextInt();

switch(ch)

{

case 1: System.out.print("\nEnter data: ");

num = s.nextInt();

stk.push(num);

break;

case 2: if(stk.isEmpty())

System.out.print("\nStack Underflow!");

else

System.out.print("\nPopped element: " + stk.pop());

break;

case 3: if(stk.isEmpty())

System.out.print("\nStack is empty!");

else

System.out.print("\nTop element: " + stk.peek());

break;

case 4: if(stk.isEmpty())

System.out.print("\nStack is empty!");

else

System.out.print("\nStack is NOT empty!");

break;

case 5: if(stk.isFull())

System.out.print("\nStack is full!");

else

System.out.print("\nStack is NOT full!");

break;

case 6: stk.display();

break;

case 7: System.exit(0);

}

}

}

}

import java.util.Scanner;

class Node

{

int data;

Node left;

Node right;

public Node(int x)

{

data = x;

left = null;

right = null;

}

}

class BST

{

Node root;

public BST()

{

root = null;

}

void insert(int x)

{

insertNode(x, root);

return;

}

void insertNode(int x, Node iter)

{

Node temp;

if(iter == null)

{

temp = new Node(x);

root = temp;

System.out.println("\nInserted " + x + " as root node!");

return;

}

else

{

if(x < iter.data)

{

if(iter.left == null)

{

temp = new Node(x);

iter.left = temp;

System.out.println("\nInserted " + x + " as left child of " + iter.data);

return;

}

else

{

insertNode(x, iter.left);

return;

}

}

else

{

if(iter.right == null)

{

temp = new Node(x);

iter.right = temp;

System.out.println("\nInserted " + x + " as right child of " + iter.data);

return;

}

else

{

insertNode(x, iter.right);

return;

}

}

}

}

void search(int x)

{

Node iter = searchNode(x, root);

if(iter == null)

System.out.println("\nElement not found!");

else

System.out.println("\nElement found!");

return;

}

Node searchNode(int x, Node iter)

{

if(iter == null)

return null;

else if(x == iter.data)

return iter;

else

{

if(x < iter.data)

return searchNode(x, iter.left);

else

return searchNode(x, iter.right);

}

}

void delete(int x)

{

if(root == null)

{

System.out.println("\nUnderflow!");

return;

}

else

{

Node pos = searchNode(x, root);

if(pos == null)

{

System.out.println("\nElement not found! Deletion impossible!");

return;

}

if(pos == root)

{

if(root.left == null && root.right == null)

root = null;

else if(root.left != null && root.right == null)

root = root.left;

else if(root.left == null && root.right != null)

root = root.right;

else

{

Node iter = root.right;

while(iter.left != null)

iter = iter.left;

iter.left = root.left;

root = root.right;

}

System.out.println("\nElement deleted from root!");

return;

}

else

{

deleteNode(pos);

return;

}

}

}

void deleteNode(Node pos)

{

Node parent = root;

while(parent.left != pos && parent.right != pos)

{

if(pos.data < parent.data)

parent = parent.left;

else

parent = parent.right;

}

if(parent.left == pos)

{

if(pos.left == null && pos.right == null)

parent.left = null;

else if(pos.left != null && pos.right == null)

parent.left = pos.left;

else if(pos.left == null && pos.right != null)

parent.left = pos.right;

else

{

Node iter = pos.right;

while(iter.left != null)

iter = iter.left;

iter.left = pos.left;

parent.left = pos.right;

}

System.out.println("\nElement deleted! Found as left child of " + parent.data);

return;

}

if(parent.right == pos)

{

if(pos.left == null && pos.right == null)

parent.right = null;

else if(pos.left != null && pos.right == null)

parent.right = pos.left;

else if(pos.left == null && pos.right != null)

parent.right = pos.right;

else

{

Node iter = pos.right;

while(iter.left != null)

iter = iter.left;

iter.left = pos.left;

parent.right = pos.right;

}

System.out.println("\nElement deleted! Found as right child of " + parent.data);

return;

}

}

void preorder(Node iter)

{

if(iter == null)

return;

else

{

System.out.print(iter.data + " ");

preorder(iter.left);

preorder(iter.right);

return;

}

}

void inorder(Node iter)

{

if(iter == null)

return;

else

{

inorder(iter.left);

System.out.print(iter.data + " ");

inorder(iter.right);

return;

}

}

void postorder(Node iter)

{

if(iter == null)

return;

else

{

postorder(iter.left);

postorder(iter.right);

System.out.print(iter.data + " ");

return;

}

}

}

class Main

{

public static void main(String[] args)

{

Scanner s = new Scanner(System.in);

BST b = new BST();

int n;

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

n = s.nextInt();

b.insert(n);

b.preorder(b.root);

System.out.println();

b.inorder(b.root);

System.out.println();

b.postorder(b.root);

n = s.nextInt();

b.delete(n);

System.out.println();

b.inorder(b.root);

}

}