//StackDyn.java

//cs111 sample implementation of a stack implemented using dynamic memory

//import java.lang.Object;

public class StackDyn<T>

{

private static class Node<T>

{

private T data;

private Node<T> next;

private Node (T d)

{

data = d;

next= null;

}

private Node (T d, Node<T> ref)

{

data = d;

next = ref;

}

}

private Node<T> top;

private int numElements;

public StackDyn()

{

top = null;

numElements = 0;

}

public StackDyn (StackDyn<T> s) throws StackEmptyException

{ Node<T> ptr, ptr2;

if (s.top == null)

top = null;

else

{ top = new Node<T>(s.top.data);

ptr = top;

for (ptr2=s.top.next;ptr2 !=null; ptr2=ptr2.next)

{ptr.next = new Node<T>(ptr2.data);

ptr = ptr.next;

}

}

numElements = s.numElements;

}

public boolean empty()

{

return (top == null);

}

public void push (T it)

{

top = new Node<T> (it, top);

numElements++;

}

public T pop() throws StackEmptyException

{ Node<T> ptr;

if (empty())

throw new StackEmptyException ("stack empty on pop");

ptr = top;

top = top.next;

numElements--;

return ptr.data;

}

public T top() throws StackEmptyException

{

if (empty())

throw new StackEmptyException ("stack empty on top");

return top.data;

}

public int numElements()

{

return numElements;

}

public void clear()

{

top = null;

numElements =0;

}

}

//TestStackDyn.java

//CS 111 program to test the dynamic stack class

public class TestStackDyn

{

public static void main(String [] args) throws StackEmptyException

{

StackDyn<Integer> s = new StackDyn<Integer>();

System.out.println("there are " + s.numElements()+ " items on the stack");

try

{System.out.println(s.top());}

catch (StackEmptyException e)

{ System.out.println(e.getMessage());}

for (int i =0; i<5; i++)

s.push(i);

System.out.println ("the contents of the first stack are: ");

while (!s.empty())

{ System.out.println(s.pop());}

for (int i =0; i<5; i++)

s.push(i);

StackDyn<Integer>s1 = new StackDyn <Integer>(s);

System.out.println ("there are " + s1.numElements() + " items on the second stack");

System.out.println ("the contents of the first stack are: ");

while (!s.empty())

{ System.out.println(s.pop());}

//s.top.data =10;

System.out.println("the contents of the second stack is: ");

while (!s1.empty())

{System.out.println(s1.pop());}

}

}

//Balance.java

//CS 111 sample program using collections class stack

//to check for balanced parens

import java.util.Stack;

import java.util.Scanner;

public class Balance

{

public static void main(String [] args) throws StackEmptyException

{

int i;

String s;

Stack<Character> paren = new Stack<Character>();

Scanner sc = new Scanner(System.in);

System.out.println ("enter an expression: (CRTL D to end)");

while (sc.hasNext())

{

s = sc.next();

System.out.println("the expression entered: " + s);

for (i=0;i<s.length();i++)

{if (s.charAt(i)=='(')

paren.push(s.charAt(i));

else

{if (s.charAt(i)==')')

if (!paren.empty())

paren.pop();

else

{i = s.length();

System.out.println("parens out of balance too many )");

}

}

}

//anything left on the stack?

if (!paren.empty())

System.out.println("parens out of balance, too many (");

//get ready for next string

//empty out the stack

//either pop until empty

//while (!paren.empty())

// paren.pop();

//or use inherited method clear()

paren.clear();

System.out.println ("enter an expression:(Ctrl D to end)");

}

System.out.println("that's all folks");

}

}

//Postfix.java

//cs 111 sample program using the collections interface

//stack facility to evaluate postfix expressions

import java.util.Stack;

import java.io.IOException;

import java.util.Scanner;

public class Postfix

{

public static void main(String [] args) throws StackEmptyException

{

Stack<Integer> postfix = new Stack<Integer>();

char s;

int right, left, result=0;

Scanner sc = new Scanner (System.in);

String ws;

System.out.println("enter postfix expression");

ws = sc.next();

for (int i=0; i < ws.length();i++)

{s = ws.charAt(i);

if (s >= '0' && s <='9')

postfix.push((int) (s) - 48);

else

{right = postfix.pop();

left = postfix.pop();

switch (s)

{case '+':

result = left + right;

break;

case '-':

result = left - right;

break;

case '/':

result = left /right;

break;

case '\*':

result = left \* right;

break;

}

postfix.push(result);

}

}

result = postfix.pop();

System.out.println("the result of the expression is: " + result);

}

}

//QueueDyn.java

//cs111 sample implementation of a queue using dynamic memory

public class QueueDyn<T>

{

private static class Node<T>

{

private T data;

private Node<T> next;

private Node (T d)

{

data =d;

next= null;

}

private Node (T d, Node<T> ref)

{

data =d;

next = ref;

}

}

private Node<T> head,tail;

private int numElements;

public QueueDyn()

{

head = null;

tail = null;

numElements=0;

}

public QueueDyn(QueueDyn<T> q)

{ Node<T> ptr, ptr2;

if (q.head == null)

{ head = null;

tail = null;

numElements=0;

}

else

{head = new Node<T>(q.head.data);

ptr=head;

for (ptr2=q.head.next; ptr2 != null; ptr2 = ptr2.next)

{ptr.next = new Node<T>(ptr2.data);

ptr = ptr.next;

}

tail = ptr;

numElements = q.numElements;

}

}

public T front() throws QueueEmptyException

{

if (empty())

throw new QueueEmptyException ("queue empty on call to front");

return head.data;

}

public void enqueue(T it)

{ Node<T> ptr;

ptr = new Node<T>(it);

if (tail == null)

head = ptr;

else

tail.next = ptr;

tail = ptr;

numElements++;

}

public T dequeue() throws QueueEmptyException

{

Node<T> ptr;

if (empty())

throw new QueueEmptyException ("queue empty on dequeue");

if (head == tail)

tail = null;

ptr = head;

head = head.next;

numElements--;

return ptr.data;

}

public boolean empty()

{

return (head == null);

}

public int numElements()

{

return numElements;

}

public void clear()

{

head = null;

tail=null;

numElements =0;

}

}

//TestQueueDyn.java

//CS111 test rpogrma to test dynamic queue implementation

public class TestQueueDyn

{

public static void main(String [] args) throws QueueEmptyException

{

QueueDyn <Integer> q = new QueueDyn<Integer>();

System.out.println("there are " + q.numElements() + " elements in the queue q");

for (int c = 5; c > 0; c--)

q.enqueue(c);

System.out.println ("there are " + q.numElements() +" elements in the queue q");

System.out.println("the head of the queue is: " + q.front());

QueueDyn <Integer> q1 = new QueueDyn<Integer>(q);

System.out.println("there are " + q1.numElements() + " elements in the queue q1");

System.out.println ("the contents of q1 are: ");

while (!q1.empty())

{

System.out.println(q1.dequeue());

}

System.out.println ("there are " + q1.numElements()+ " elements in the queue q1");

System.out.println ("there are " + q.numElements()+ " elements in the queue q");

q.clear();

System.out.println ("there are " + q.numElements() +" elements in the queue q");

}

}

//ListDyn.java

//cs111 sample program

//dynamic implementation of an ordered list

//the template class used must provide equals, compareTo and toString

public class ListDyn<T extends Comparable>

{

private static class Node<T>

{

private T data;

private Node<T> next;

private Node (T d)

{

data =d;

next= null;

}

private Node (T d, Node<T> ref)

{

data =d;

next = ref;

}

}

private Node<T> head;

private int numElements;

public ListDyn()

{ head =null;

numElements=0;

}

public ListDyn(ListDyn<T> l)

{ Node<T> ptr, ptr2;

if (l.head==null)

{numElements=0;

head =null;

}

else

{head = new Node<T>(l.head.data);

ptr=head;

for (ptr2=l.head.next; ptr2!=null; ptr2= ptr2.next)

{ ptr.next = new Node<T>(ptr2.data);

ptr = ptr.next;

}

}

numElements=l.numElements();

}

public void display\_list()

{

Node<T> ptr;

ptr=head;

while (ptr !=null)

{ System.out.println(ptr.data);

ptr= ptr.next;

}

}

public void insert (T it)

{ Node<T> ptr, trav, prev;

prev = null;

trav= head;

while (trav != null && it.compareTo(trav.data)>0)

{ prev= trav;

trav= trav.next;

}

ptr = new Node<T>((T)it,trav);

if (prev == null)

head = ptr;

else

prev.next=ptr;

numElements++;

}

public void remove (T it) throws ListEmptyException, NotInListException

{

Node<T> ptr, prev, trav;

if (empty())

throw new ListEmptyException("list empty on remove");

prev=null;

trav = head;

while (trav !=null && it.compareTo(trav.data)>0)

{

//if (it.compareTo(trav.data)>0) throw new NotInListException("item not in list on remove");

prev=trav;

trav= trav.next;

}

if (trav == null || !(trav.data.equals(it))) throw new NotInListException ("item not in list on remove");

//if (trav==null)

// throw new NotInListException("item not in list on remove");

if(prev==null)

head = trav.next;

else

prev.next = trav.next;

numElements--;

}

public boolean empty()

{

return head ==null;

}

public int numElements()

{

return numElements;

}

public void clear()

{

numElements =0;

head=null;

}

}

//TestListDyn.java

//CS111 test rpogram for dynamic list class

public class TestListDyn

{

public static void main(String [] args)

{

ListDyn <Integer> l= new ListDyn<Integer>();

System.out.println ("there are " + l.numElements() + " items in the list");

for (int i =5; i>0; i--)

l.insert(i);

l.insert(10);

l.insert(8);

System.out.println("the contents of the list are: ");

l.display\_list();

ListDyn<Integer>l1 = new ListDyn<Integer>(l);

System.out.println ("the contents of the second list are: ");

l1.display\_list();

try

{

System.out.println ("remove 1");

l1.remove (1);

System.out.println ("the list now contains");

l1.display\_list();

System.out.print("remove 4");

l1.remove(4);

System.out.println ("the list now contains");

l1.display\_list();

System.out.print("remove 10");

l1.remove(10);

System.out.println ("the list now contains");

l1.display\_list();

System.out.println ("remove 20");

l1.remove(20);

System.out.println ("the list now contains");

l1.display\_list();

}

catch (ListEmptyException e)

{ System.out.println(e.getMessage() + " item not removed");}

catch (NotInListException e)

{ System.out.println(e.getMessage() + " item not removed");}

System.out.println("there are " + l1.numElements() + " items in the second list");

System.out.println("the original list contains: ");

l.display\_list();

}

}