**Stack Examples**

Write a function **stackShuffle to be outside** ArrayStack class that accepts two parametrers **s1** of type ArrayStack and **value** of type E. The function is going to reshuffle s1 in a way that the shuffling will be in according to the following criteria:

If the element in the stack is greater than the value, it will be in the first half of the stack, otherwise, it will be in the second half of the stack (direction is starting from the top and go down as shown in the example after the function call).

The function returns false if stack is empty, otherwise, it returns true.

The function title is

public static boolean stackShuffle(ArrayStack<Type> s1,E value)

**Ex:**

**Before function call**

**S1**

top

23 45 13 67 22 5 14 77

value is 40

**After function call**

**S1**

top

45 67 77 23 13 22 5 14

public static<E> boolean stackShuffle(ArrayStack<E> s1,E value)

{

if (s1.isEmpty()) return false;

ArrayStack<E> st1= new ArrayStack<E>();

ArrayStack<E> st2= new ArrayStack<E>();

while(!s1.isEmpty())

{

if(((Comparable)s1.peek()).compareTo((Comparable) value)> 0)

st1.push(s1.pop());

else

st2.push(s1.pop());

}

while (! st2.isEmpty())

s1.push(st2.pop());

while(! st1.isEmpty())

s1.push(st1.pop());

return true;

}

Write a function **matchAndUpdate** to be outside the ArrayStack class that has two parameters **St1** and **St2** of type Arraystack with integer data elements and a third parameter **value** of type int. The function is to change the values of st1 if there is a full match between St1 and St2 (all elements in St1 and St2 are the same and in the same order), St1 will change in a way that value will be added to every element in St1. If no full match or any of St1 or St2 is empty or no full match, the function returns false, otherwise it returns true. Assume St1 and St2 are of the same size. Assume further that you have accessibility to ArrayStack class functions only and not allowed to use arrays. The function prototype is

public Boolean static matchAndUpdate(ArrayStack<Integer> St1, ArrayStack<Integer> St2, int value);

**Ex:**

St1:

top

2 5 7 8 9

St2:

2 5 7 8 9

Value is 2

**After function call**

**St1:**

**top**

4 7 9 10 11

**St2:**

2 5 7 8 9

public static boolean matchAndUpdate(ArrayStack<Integer> St1, ArrayStack <Integer> St2, int value)

{ ArrayStack<Integer> St11= new ArrayStack<Integer>(St1);

ArrayStack<Integer> St22= new ArrayStack<Integer>(St2);

ArrayStack<Integer> St1Reverse = new ArrayStack<Integer>();

if (St1.isEmpty() || St2.isEmpty()) return false;

while (! St11.isEmpty())

if (! St11.pop().equals(St22.pop())) return false;

while (! St1.isEmpty())

St1Reverse.push(St1.pop()+value);

while( !St1Reverse.isEmpty())

St1.push( St1Reverse.pop());

return true;

}

Write a function reverseEquality to be considered outside the arrayStack class that accepts two parameters St1 and St2 of type ArrayStack, the function returns true if both St1 and St2 are in the reverse order, otherwise, it returns false. Assume St1 and St2 are of the same size.

The function title is

public static <E> boolean reverseEquality(ArrayStack<E> St1, ArrayStack <E> St2)

public static <E> boolean reverseEquality(ArrayStack<E> St1, ArrayStack <E> St2)

{

ArrayStack<E> St11= new ArrayStack<E>(St1);

ArrayStack<E> St22= new ArrayStack<E>(St2);

ArrayStack<E> St2Reverse = new ArrayStack<E>();

if (St1.isEmpty() || St2.isEmpty()) return false;

while (!St22.isEmpty())

{

St2Reverse.push(St22.pop());

}

while(!St11.isEmpty())

{

if ( ! St11.pop().equals(St2Reverse.pop())) return false;

}

return true;

}

Write a function reverseEqualityAndFillStack to be considered outside the ArrayStack class that accepts three parameters St1 , St2 and St3 of type ArrayStack, the function checks if both St1 and St2 are in the reverse order, fill St3 with St1 and St2 in the same order. Assume St1 and St2 are of the same size. If filling is done, the function returns true, otherwise, it returns false. The function title is

public static <E> boolean reverseEqualityAndFillStack(ArrayStack<E> St1, ArrayStack <E> St2, ArrayStack<E> St3)

public static <E> boolean reverseEqualityAndFillStack(ArrayStack<E> St1, ArrayStack <E> St2, ArrayStack<E> St3)

{

ArrayStack<E> St11= new ArrayStack<E>(St1);

ArrayStack<E> St22= new ArrayStack<E>(St2);

ArrayStack<E> St2Reverse = new ArrayStack<E>();

if (St1.isEmpty() || St2.isEmpty()) return false;

while (!St22.isEmpty())

{

St2Reverse.push(St22.pop());

}

while(!St11.isEmpty())

{

if ( ! St11.pop().equals(St2Reverse.pop())) return false;

}

ArrayStack<E> St1Reverse = new ArrayStack<E>();

ArrayStack<E> St111= new ArrayStack<E>(St1);

ArrayStack<E> St22Reverse = new ArrayStack<E>();

ArrayStack<E> St222= new ArrayStack<E>(St2);

while (! St111.isEmpty())

St1Reverse.push(St111.pop());

while(!St222.isEmpty())

St22Reverse.push(St222.pop());

while (!St1Reverse.isEmpty())

St3.push(St1Reverse.pop());

while(!St22Reverse.isEmpty())

St3.push(St22Reverse.pop());

return true;

}

Write a function to be outside the ArrayStack class that accepts one parameter value as integer, the function returns true if value is palindrome, or otherwise it returns false. The function title is

public static boolean Palindrom(int value)

public static boolean Palindrom(int value)

{

int temp = value;

int rem;

ArrayStack<Integer> st = new ArrayStack<Integer>();

do

{

rem = temp %10;

st.push(rem);

temp=temp/10;

} while (temp>0);

temp=value;

while (! st.isEmpty())

{

rem = temp %10;

temp=temp/10;

if (st.pop() != rem) return false;

}

return true;

}

**Another Solution**

public static boolean Palindrom (int value)

{

int temp=value;

int rem;

ArrayStack<Integer> st = new ArrayStack<Integer>();

int count=0;

do

{

rem = temp %10;

st.push(rem);

temp=temp/10;

count++;

} while (temp>0);

ArrayStack<Integer> streverse1 = new ArrayStack<Integer>();

ArrayStack<Integer> st11= new ArrayStack<Integer>(st);

int i=0;

int size=count;

while (i < count/2)

{

streverse1.push(st11.pop());

i++;

}

if (size % 2!=0)

st11.pop();

while(! streverse1.isEmpty())

{

if(streverse1.pop() != st11.pop()) return false;

}

return true;

}

Write a function called rearrangeStack to be outside the ArrayStack class and has two parameters st1 and st2. st1 has data of type E and st2 has data of type character ‘a’ or ‘b’ values. Assume st1 and st2 are of the same size. st1 will be rearranged in a way that all elements corresponding to a’s in st2 will be on top of st1 stack and those correspond to b’s in st2 will be on the bottom of the st1. The function does not return any value.

Public void rearrangeStack(ArrayStack <E> st1, ArrayStack<Character> st2)

{

ArrayStack<Character> st2Copy = new ArrayStack<Character>();

ArrayStack<E> sta = new ArrayStack<E>();

ArrayStack<E> stb = new ArrayStack<E>();

while (!st1.isEmpty())

{

if(st2Copy.pop() == ‘a’)

sta.push(st1.pop());

else

stb.push(st1.pop());

}

while(!stb.isEmpty())

st1.push(stb.pop());

while(! sta.isEmpty())

st1.push(sta.pop());

}

Another solution:

public static void rearrangeStacks(ArrayStack<Integer> s1,

ArrayStack<Integer> s2);

Solution:

public class StackEx

{

public static void rearrangeStacks(ArrayStack<Integer>

s1,ArrayStack<Integer> s2)

{

if(s1.isEmpty() && s2.isEmpty())

System.out.println(“Both stacks are empty”);

else

{

int item;

ArrayStack<Integer> s3 = new ArrayStack<Integer>();

ArrayStack<Integer> s4 = new ArrayStack<Integer>();

while(!s1.isEmpty())

{

item = s1.pop();

if(item > 0)

s3.push(item);

else if(item < 0)

s4.push(item);

}

while(!s2.isEmpty())

{

item = s2.pop();

if(item > 0)

s3.push(item);

else if(item < 0)

s4.push(item);

}

while(!s3.isEmpty())

s1.push(s3.pop());

while(!s4.isEmpty())

s2.push(s4.pop());

}//end else

}

}

Assume that we have two stacks, namely st1 and st2. Write a method called mergeStacks in a class called StackMerging that merges the two stacks into a third one called st3. The st3 is obtained by pushing an element from st1, followed by pushing an element from st2 (if any). If no more elements are available in one of the stacks, then the remaining elements of the other stack are pushed on st3.

Method heading:

public static void mergeStacks(Arraystack<E> st1,

Arraystack<E> st2, Arraystack<E> st3);

Solution:

public class StackMerging

{

public static boolean mergeStacks(Arraystack<E> st1,

Arraystack<E> st2, Arraystack<E> st3)

{

if(st1.isEmpty() && st2.isEmpty())

System.out.println(“Both stacks are empty”);

else

{

while(!st1.isEmpty() && !st2.isEmpty())

{

st3.push(st1.pop());

st3.push(st2.pop());

}

//pushing the remaining elements of st1, if any, in st3

while(!st1.isEmpty())

st3.push(st1.pop());

//pushing the remaining elements of st2, if any, in st3

while(!st2.isEmpty())

st3.push(st2.pop());

}

}

}

Write a method called compare in a class called CompareStacks that accepts two objects st1 and st2 of type LinkedStack, as parameters. The method compares the elements of the stack st1 and st2 and performs the following actions:

If both the stacks st1 and st2 contain the same elements in the same order then the method returns 0.

If both the stacks st1 and st2 contain the same elements but in the reverse order then the method returns 1.

In all other cases, the method returns -1.

Assume that class LinkedStack is available for use.

Method heading:

public static int compare(LinkedStack<E> st1, LinkedStack<E> st2);

Solution:

public class CompareStacks

{

public static int compare(LinkedStack<E> st1,

LinkedStack<E> st2)

{

if(st1.isEmpty() && st2.isEmpty())

return 0;

LinkedStack<E> st3 = new LinkedStack<E>(st1);

LinkedStack<E> st4 = new LinkedStack<E>(st2);

while(!st3.isEmpty() && !st4.isEmpty())

{

if(st3.peek() != st4.peek())

break;

st3.pop();

st4.pop();

}

if(st3.isEmpty() && st4.isEmpty())

return 0; // Both stacks are equal

LinkedStack<E> st5(st1), st6(st2), st7;

//Reversing st6, i.e., st2 in st7

while(!st6.isEmpty())

st7.push(st6.pop());

while(!st5.isEmpty() && !st7.isEmpty())

{

if(st5.peek() != st7.peek())

break;

st5.pop();

st7.pop();

}

if(st5.isEmpty() && st7.isEmpty())

return 1; // Both stacks are equal in reverse order

else

return -1; // Both stacks are not equal

}

}