BMI 2123 DATA STRUCTURES

ASSIGNMENT 2:

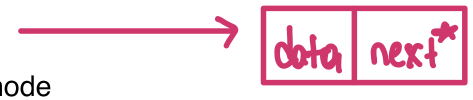
SOLUTION PAPER

1-Source code is attended to zip file

CODE'S STEPS:

Import <iostream> and <ostream> libraries

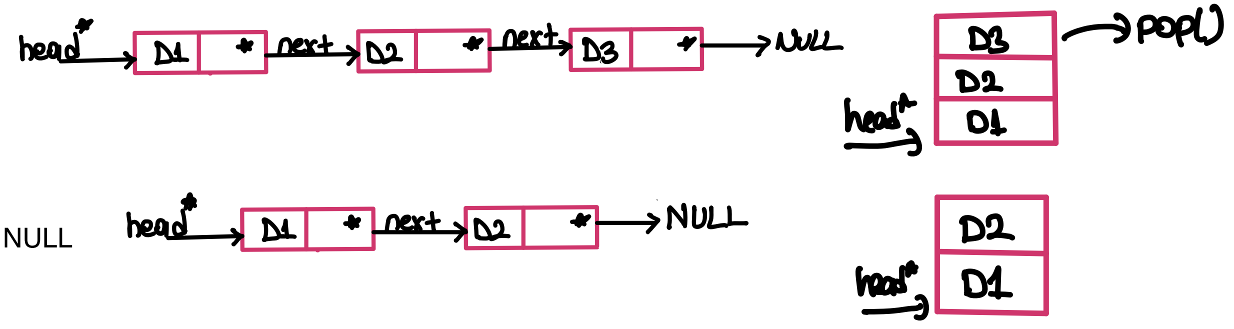
Define Node structure
- Part 1 : int data
- Part 2: pointer of next node



DATA = D

Define Stack structure
- Head pointer
- Stack constructure
- Functions
    - void pop()
    - void push(int data)
    - int top()
    - int size()
    - void isEmpty()
    - Stack stackSorter(Stack stack1, Stack stack2)
    - void printStackItems()

Explanation of pop: this function for delete the last node
- 1-enter the stack



2-if node is empty, print “ERROR”

3-if head’s next is not equal to NULL->

    Chose a node which ->next->next equal to NULL

    And now this node’s next is NULL

    So that last node deleted.

4.if it is except all these conditions

    Head equal to NULL,meaning last node deleted.

Explanation of push:this function for push a node on the stack’s top.

1-create a node

2-insert data

3-new node’s next is equal to NULL

4.if the stack is empty ,

    Head point to new node.

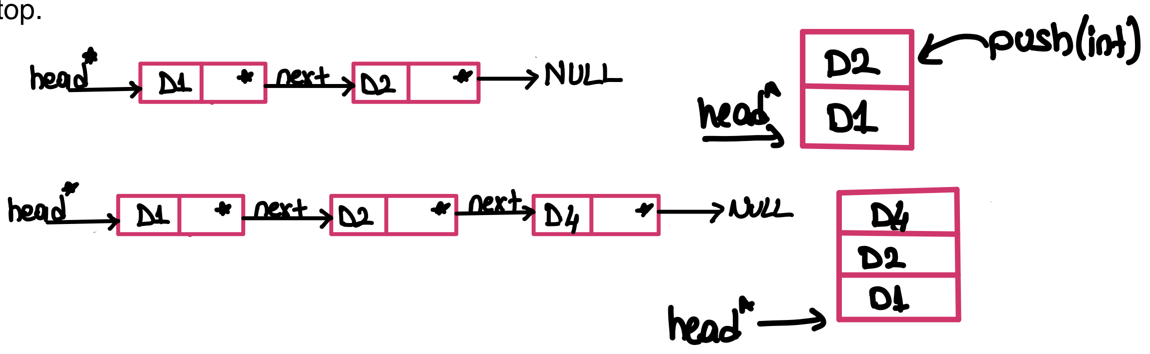
5.if the stack is not empty,

    Enter the stack

    Chose a node which ->next equal to NULL

    And now this node’s next is new node

    So that we can add a node on the top of the stack.



Explanation of top:This function is get top node’s data

1-Define an int that name is x.

2-Enter the stack

3-if the stack is empty

    This part is optionel: print ERROR

4-if the stack is not empty,

    Chose a node which ->next equal to NULL

Now X equal to chosen node’s data.
This part is optionel:print X

Explanation of size: This function is get stack’s length

```
1-Define an int that name is count and equal to 0.
2-Enter the stack.
3-if stack is empty,
    This part is optionel:print count so 0.
    Return count so 0.
4-if stack is not empty,
    Go to the last node with while loop
    Each time increase count by one
    Return count+1 so this number is stack’s length.
```

Explanation of isEmpty:

```
1-If head equal to NULL, this stack is empty  and print True
2-If head is not equal to NULL, this stack is not empty and print False.
```

Actually we can use boolean type in this function,but i chosed this option for now.

Explanation of printStackItems: this function for display all stack’s data on the screen.

```
1-Enter the stack
2-Display each data until the node->next is equal to NULL
    With this loop our last node is not written.
3-Therefore we print chosen node’s data on the screen.
```

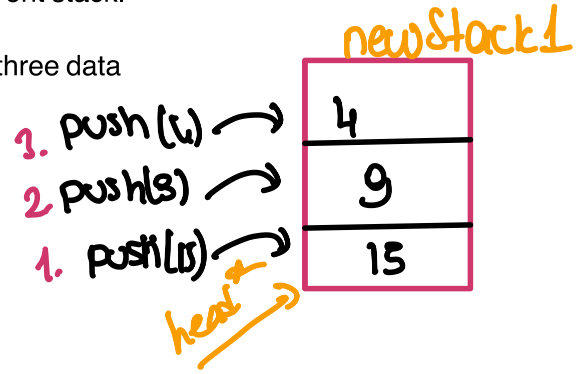
Explanation of stackSorter: This function takes two sorted stack. Firstly merge ,min to max. Then reverse. Therefore return min on the top stack .

1- we define a function that takes two stack parameters.
2- We define an int variable that name is newS
3- We define a new stack for merging.
4- we enter a while loop until the number of elements of both stacks is equal to 0.
-Summarize,we get top elements on the each stacks with top function , and smaller one add to merge stack.
We get smaller one
For saving this value, we use newS= smallerone of the top()
we remove the small value using pop() function from the its stack.
And add newS(smaller value) to merge stack using push() function.
We have 4 options for all conditions.
5-display the merge stack and this stack sorted numbers,from min to max , so max on the top
6-we define a new stack that name is sorted and this stack for the main answer.
7-we define an int variable that name is mobil(i couldn't find a name:)).
8. we enter a while loop until the number of element of stacks is equal to 0.
For saving this value, we use mobile=data of merge stack's top()
We remove the top using pop function
And add mobile to sorted stack using push() function.
9- display the sorted stack and this stack sorted numbers,from max to min , so min on the top
10- The last two lines were written to show that other functions also work.

Main function

1-I create two different stack.

2-one of them has three data
1. Push(15)
2. Push(9)
3. Push(4)



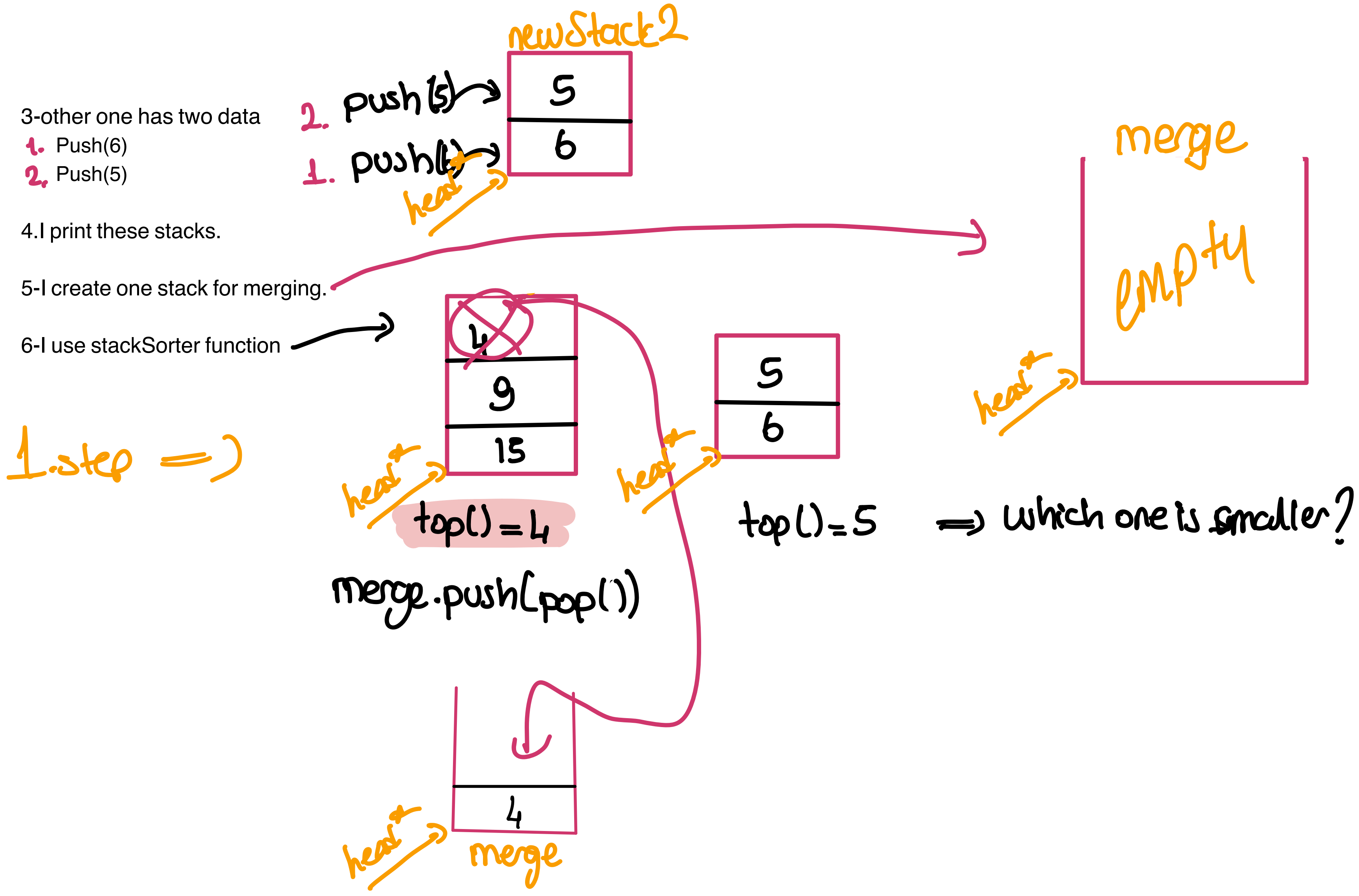
3-other one has two data

1. Push(6)
2. Push(5)

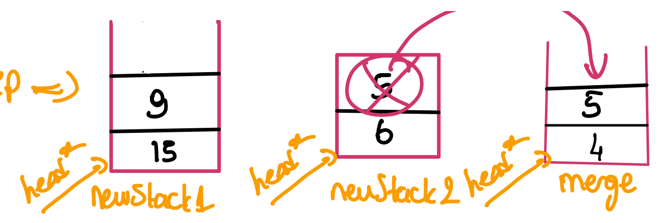
4.I print these stacks.

5-I create one stack for merging.

6-I use stackSorter function



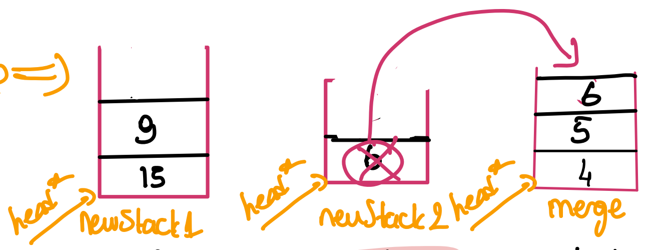
2.step =>



top()=9 top()=5 => which one is smaller?

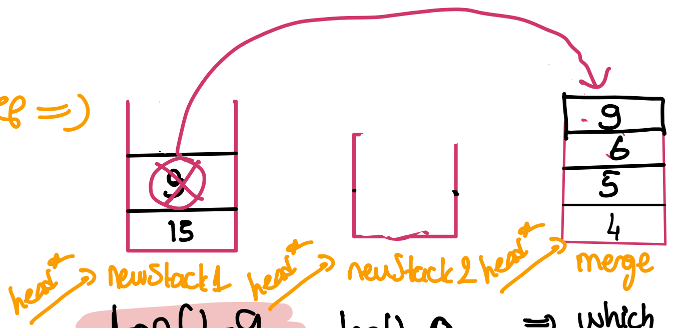
merge.push(pop())

3.step =>



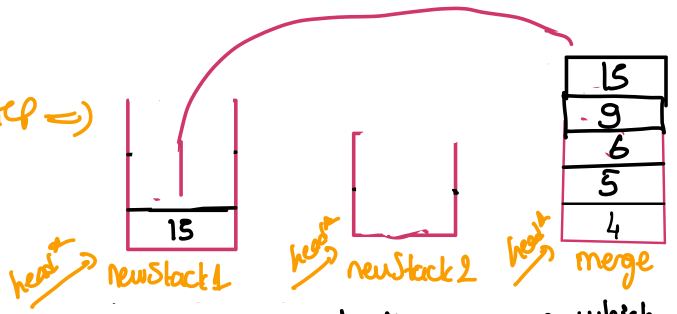
top()=9 top()=6 => which one is smaller?

4.step =>



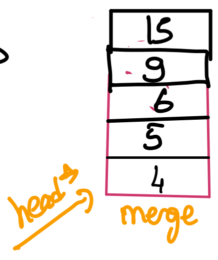
top()=9 top()=0 => which one is not 0

5.step =>

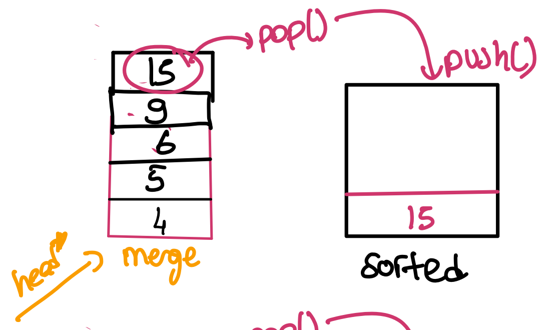


top()=15 top()=0 => which one is not 0

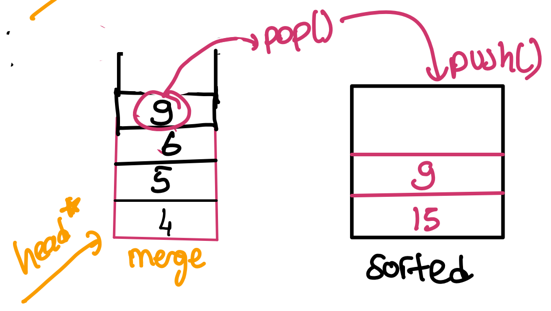
merge stack is



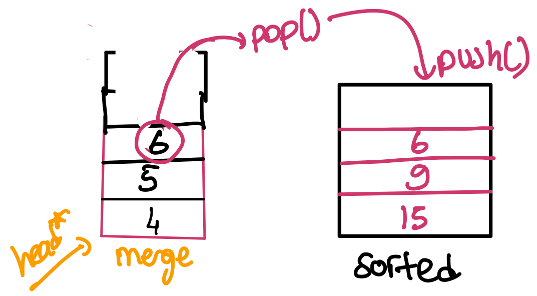
☆ reverse part



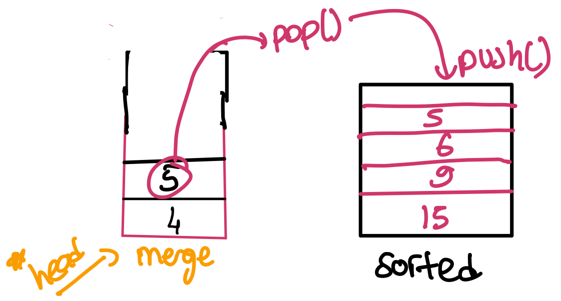
pop()
push()
15
9
6
5
4
15
head*
merge
sorted



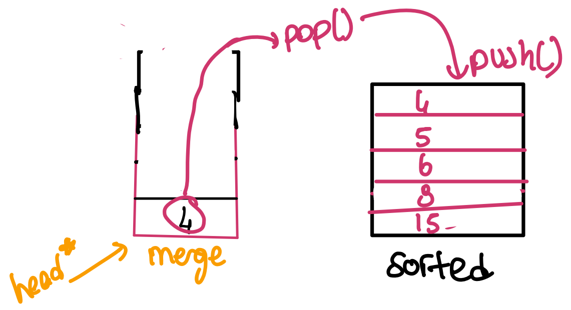
pop()
push()
9
6
5
4
9
15
head*
merge
sorted



pop()
push()
6
5
4
6
9
15
head*
merge
sorted



pop()
push()
5
4
5
6
9
15
*head
merge
sorted



pop()
push()
4
4
5
6
8
15
head*
merge
sorted