Mariya Eggensperger
CST 370, Spring 2017
Dr. Feiling Jia
Design/Analysis
of Algorithms

CST 370 Homework (Stacks)

1. (20 points) Consider two stacks each of size 10. When you pop an element from the first stack you multiply it by 2 and add it to the second stack. When you pop an element from the second stack you multiply it by 3 and add it to the first stack.

Push numbers 1 to 5 to the first stack in order. Push numbers 11 to 15 to the second stack in order. Pop two numbers from the first stack (remember these numbers are going to be added to the second stack). Pop three numbers from the second stack (remember these numbers are going to be added to the first stack).

Stack 1

45
30
24
3
2
1

Stack 2

14
13
12
11

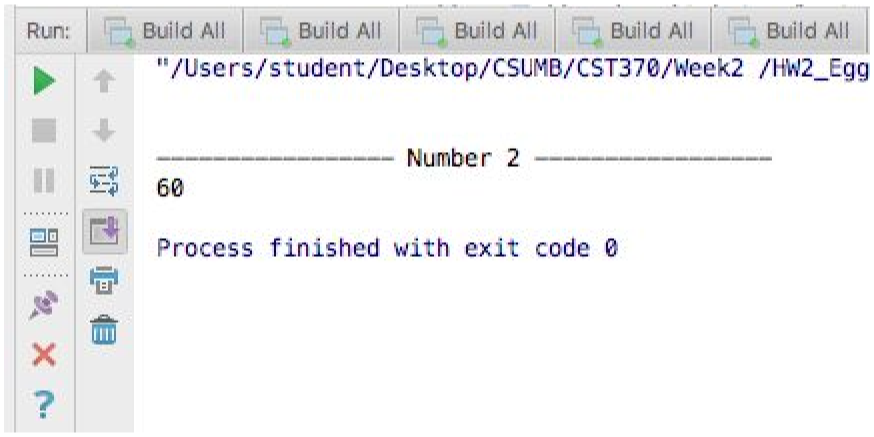
a) What is the value in the top of the first stack?
The value of myTop for Stack 1 is 45.

b) What is the value at the top of the second stack?
The value of myTop for Stack 2 is 14.

c) What is the current size of the first stack?
mySize = 6;
d) What is the current size of the second stack?
mySize = 4;

2. (20 points) Draw the stack after the following code executes

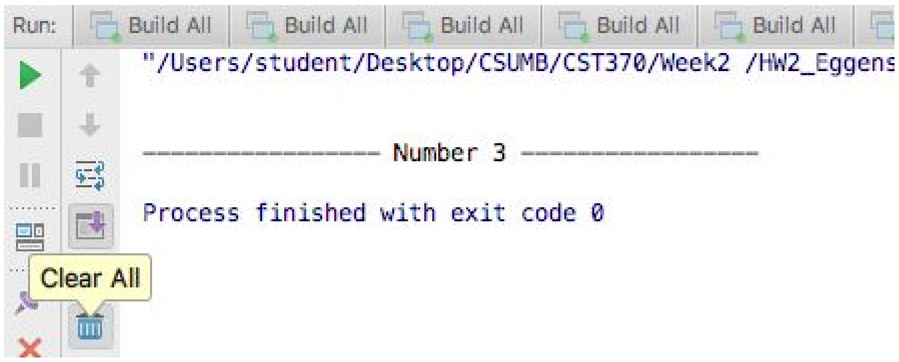
Stack s; s.push(60); s.push (25); s.push(50); s.pop( ); s.pop( );



```
"/Users/student/Desktop/CSUMB/CST370/Week2 /HW2_Egg

------------------ Number 2 ------------------
60

Process finished with exit code 0
```

3. (20 points) Draw the stack after the following code executes

Stack s;
s.push (44); s.push (33); s.push (22);
while (!s.empty())
s.pop();

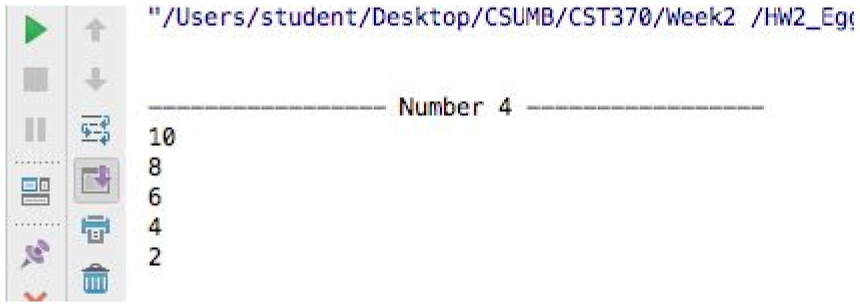


```
"/Users/student/Desktop/CSUMB/CST370/Week2 /HW2_Eggens

------------------ Number 3 ------------------

Process finished with exit code 0
```

4. (20 points) Draw the stack after the following code executes

Stack s;
for (int i = 1; i <= 5; i++)

s. push (2*i);



```
"/Users/student/Desktop/CSUMB/CST370/Week2 /HW2_Eg

------------------ Number 4 ------------------
10
8
6
4
2
```

5. (20 points) Suppose that some application requires using two stacks whose elements are of the same type. A natural storage structure of such a two-stack data type would consist of two arrays and two top pointers. Explain why this may not be a space- wise efficient implementation.

Arrays are statically allocated and thus require CAPACITY allocation of memory for the arrays at the start of the program. For this reason, the above mentioned method does not efficiently utilize the available space in an array because the method contrarily produces overflow if there is space available in the array[ ]. To illustrate, suppose that the maximum capacity for a stack is 200 elements. We assume <element>array[200]. Therefore, 200 variables of type <element> would be allotted for in memory. However, a program that does not let the stacks grow toward each other, would in effect, fail when if only 50 of the 200 elements were filled up, leaving 150 wasted slots. A single array for the storage structure would be more efficient in allowing the stacks to grow toward each other.