UNIVERSITI TEKNOLOGI MARA

CSC508 Data Structures

Topic 5 : Stack

Compiled & edited by: Zahid Zainal

Recap

- Improving Linked List
- Doubly Linked
- Circular Linked List
- Multidimensional Linked List

UNIVERSITI TEKNOLOGI MARA

Topic Structure

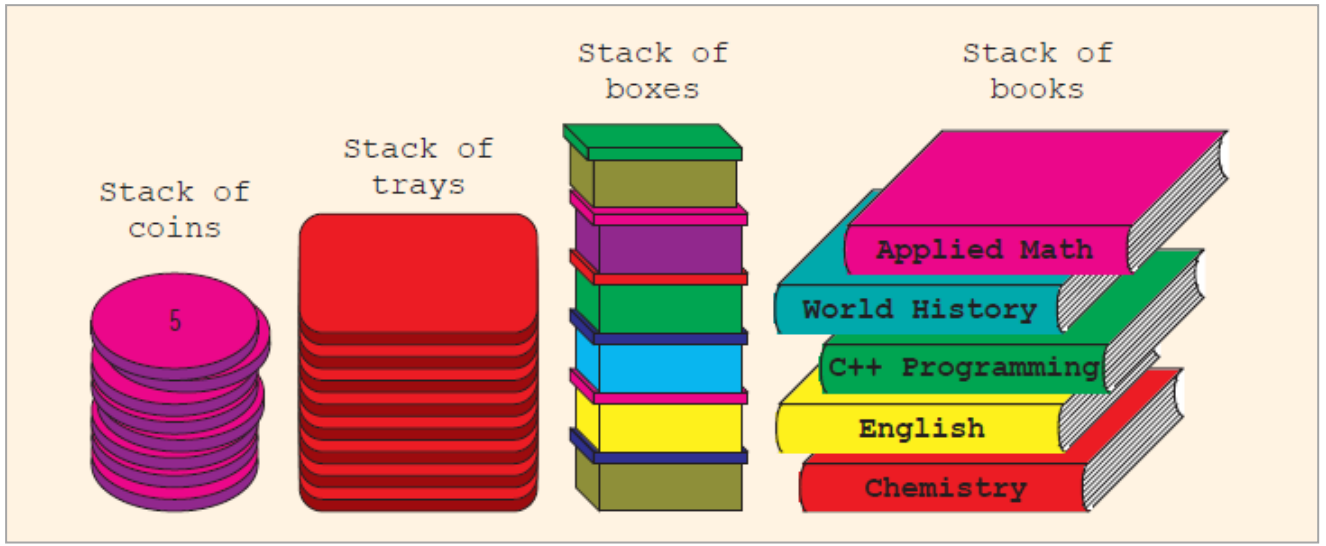
- Stack Definition
- Stack Operations
- Stack Application
- Stack Implementation

Learning Outcomes

- At the end of this lesson, students should be able to:
  - Describe stack data structure
  - Explain stack implementation
  - Implement stack operation

Stack Definition

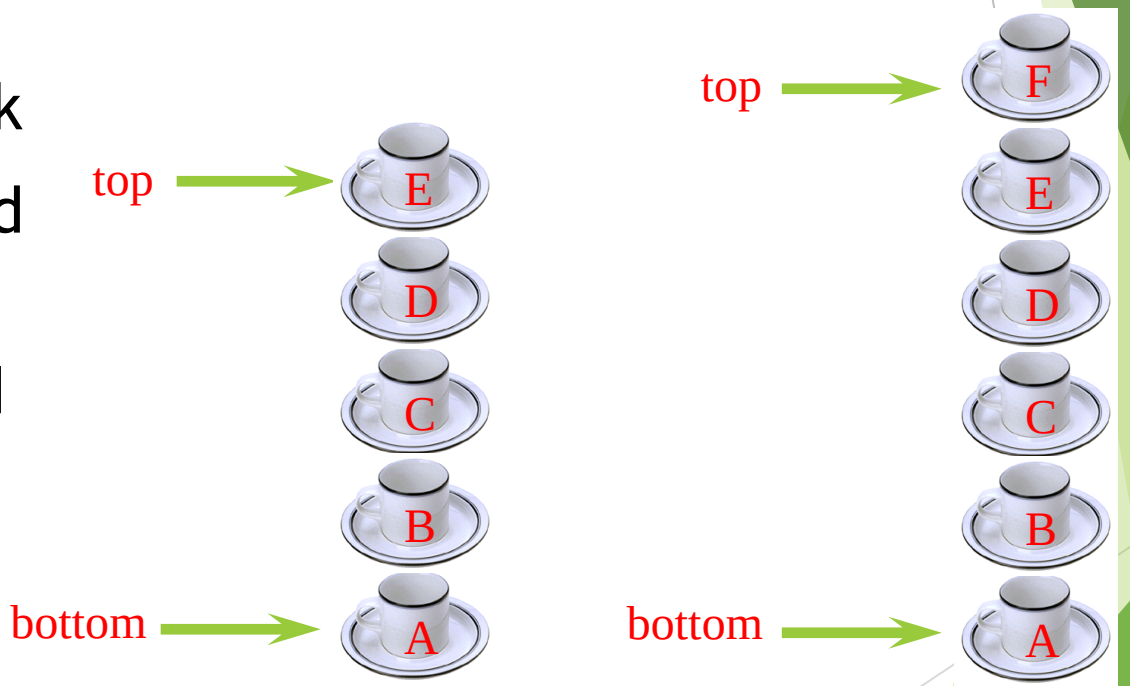
- A list of homogeneous elements; addition and deletion occur only at one end of it, called the top of the stack.



- It is based on last in first out (LIFO) algorithms

Last In First Out (LIFO)

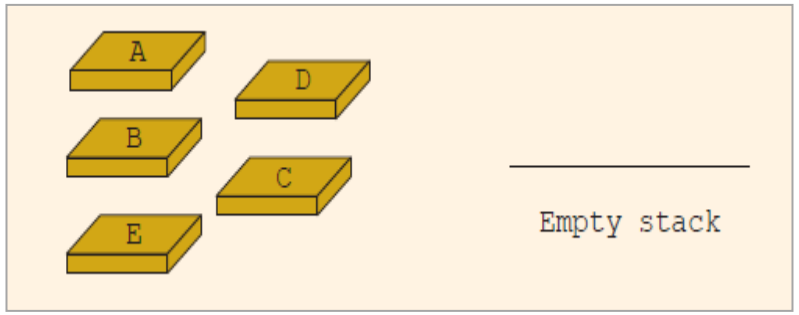
- Top element of stack is last element to be added to stack
- Elements added and removed from one end (top)
- Item added last are removed first

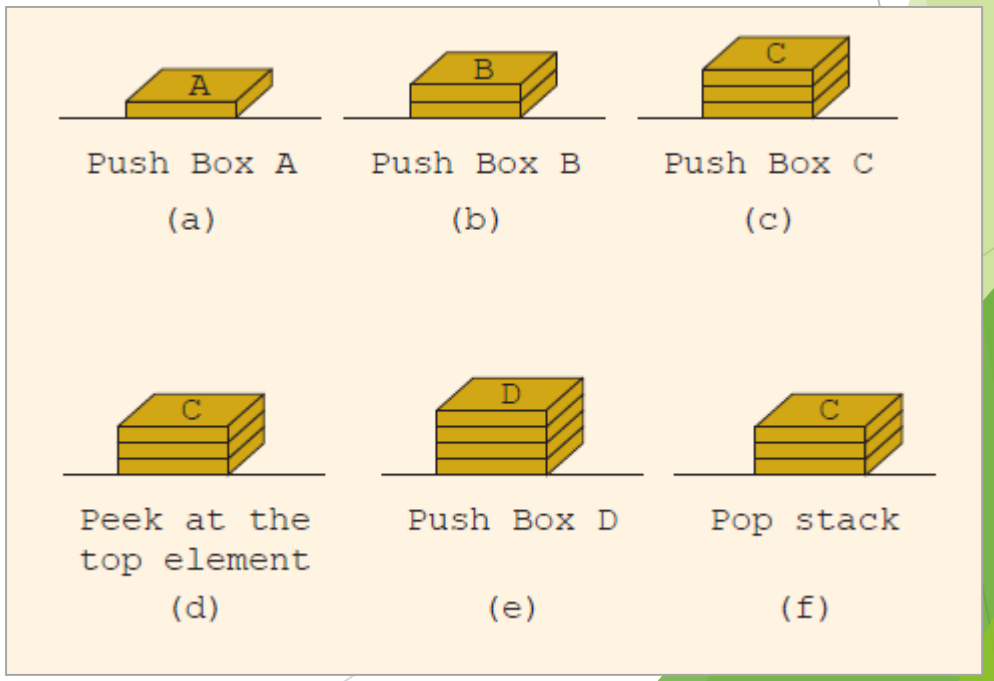


Adding a cup to the stack
A cup can be removed only from the top

Stack Operations

- Push: adds an element onto the stack
- Pop: removes an element from the stack
- Top: peeks at top stack element (without removing it)





Application: Parentheses Matching

- 0 1 2 6 13 15 19 21 25 27 31 32 34 38
  (((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)
  - Check if parentheses match, and output pairs (u,v) such that the left parenthesis at position u is matched with the right parenthesis at position v
    - (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)
- (a+b))*((c+d)
  - (0,4)
  - Error: right parenthesis at 5 has no matching left parenthesis
  - (8,12)
  - Error: left parenthesis at 7 has no matching right parenthesis

Algorithm for Parentheses Matching

- Using stack data structure, the following algorithm gives the desired solution:

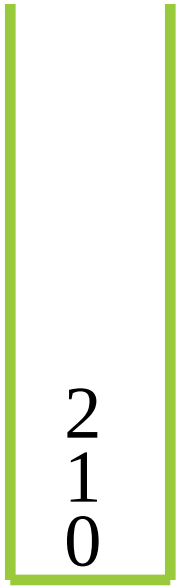
1. scan expression from left to right
2. when a left parenthesis is encountered, add its position to the stack
3. when a right parenthesis is encountered, remove matching position from stack and print the positions pair
4. If right is encountered and stack is empty, **error**
5. If expression ends and stack not empty, **error**

Simulation

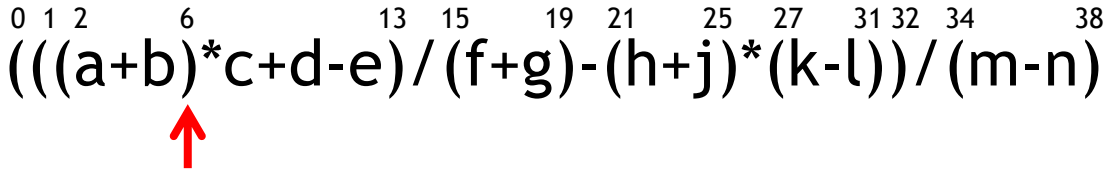
0 1 2 6 13 15 19 21 25 27 31 32 34 38

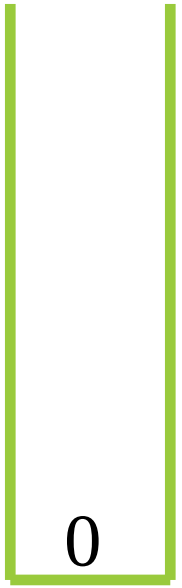
(((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

↑



Simulation



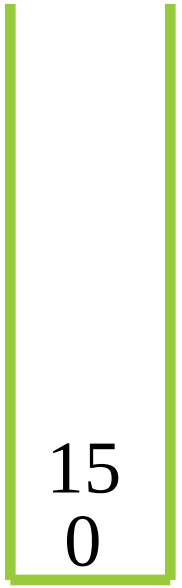


(2,6) (1,13)

Simulation

0 1 2 6 13 15 19 21 25 27 31 32 34 38

(((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

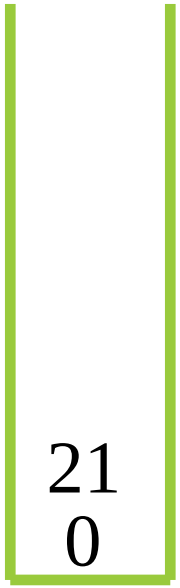


(2,6) (1,13)

Simulation

0 1 2 6 13 15 19 21 25 27 31 32 34 38

(((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

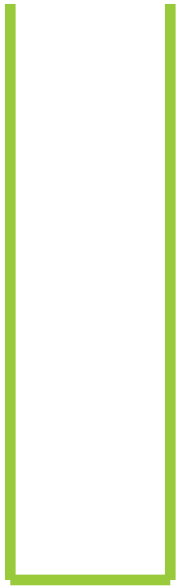


(2,6) (1,13) (15,19)

Simulation

0 1 2 6 13 15 19 21 25 27 31 32 34 38

(((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)



(2,6) (1,13) (15,19) (21,25) (27,31) (0,32)

And so on…

Application: Infix to Postfix Evaluation

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,
   >If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
   >If the precedence of the scanned operator is lower, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

   > If the scanned character is an '(', push it to the stack.

   > If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
4. Repeat steps 2 and 3 until infix expression is scanned.
5. Print the output.
6. Pop and output from the stack until it is not empty.

Example 1

Expression	Stack	Output
A+B*C-D/E		A
+B*C-D/E	+	A
B*C-D/E	+	AB
*C-D/E	+*	AB
C-D/E	+*	ABC
-D/E	-	ABC*+
D/E	-	ABC*+D
/E	-/	ABC*+D
E	-/	ABC*+DE
		ABC*+DE/-

Example 2

Expression	Stack	Output
A+B*C/(E-F)		A
+B*C/(E-F)	+	A
B*C/(E-F)	+	AB
*C/(E-F)	+*	AB
C/(E-F)	+*	ABC
/(E-F)	+/	ABC*
(E-F)	+/(	ABC*
E-F)	+/(	ABC*E
-F)	+/(-	ABC*E
F)	+/(-	ABC*EF
)	+/	ABC*EF-
		ABC*EF-/+

Stack Implementation

- Built-in `Stack` class
- User-defined
  - Array
    - Use a 1D array (can be static or dynamic)
    - Stack elements are stored in stack[0] through stack[top]
  - Linked-list
    - Elements are stored inside linked nodes
    - Elements are pushed at the beginning of the list (using head pointer)
    - Popped elements are also taken from the beginning, where the head node gets deleted

Methods in Stack Class

empty()	It returns true if nothing is on the top of the stack. Else, returns false.
peek()	Returns the element on the top of the stack, but does not remove it.
pop()	Removes and returns the top element of the stack. An 'EmptyStackException' An exception is thrown if we call pop() when the invoking stack is empty.
push(Object element)	Pushes an element on the top of the stack.
search(Object element)	It determines whether an object exists in the stack. If the element is found, It returns the position of the element from the top of the stack. Else, it returns -1.

Source : https://www.geeksforgeeks.org/stack-class-in-java/

Sample Stack Class Implementation

```
import java.util.Stack;

public class TestStack {

    public static void main(String[] args) {
        Stack<Character> S1 = new Stack<Character>();
        S1.push('A');
        S1.push('G');
        S1.push('B');
        S1.push('H');
        System.out.println("Top 1 - " + S1.peek());
        S1.pop();
        S1.pop();
        System.out.println("Top 2 - " + S1.peek());
    }
}
```

```
Top 1 - H
Top 2 - G
```

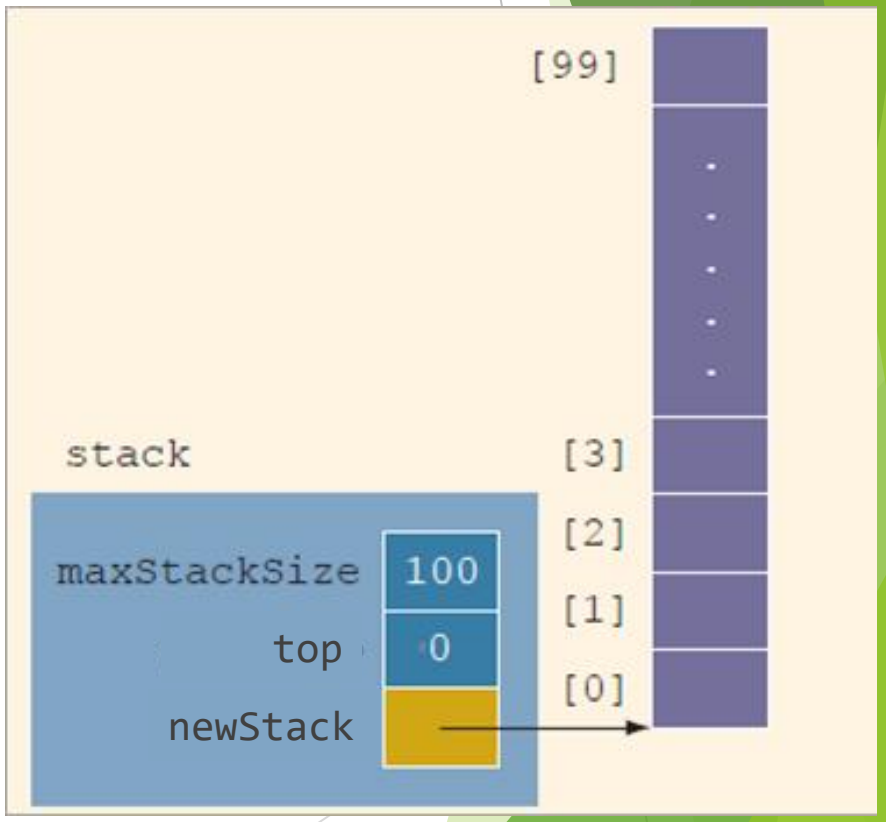
Array Implementation of Stack

- Because stack is homogeneous
  - We can use an array to implement a stack
  - Array is dynamically allocated and its size is specified upon stack creation
- First element can go in first array position, the second in the second position, etc.
- The top of the stack is the index of the last element added to the stack
- When stack is full, overflow may happen
  - Pushing elements into a full stack can either cause error, or relocate all elements into a bigger dynamic array

Stack Operations

- Declare and create stack

```
class MyStack{
    int maxStackSize = 100;
    int []newStack = new int[maxStackSize];
    int top;

    public void MyStack() {
        top = 0;
    }
}
```



Stack Operations (cont.)

- Check empty and full stack

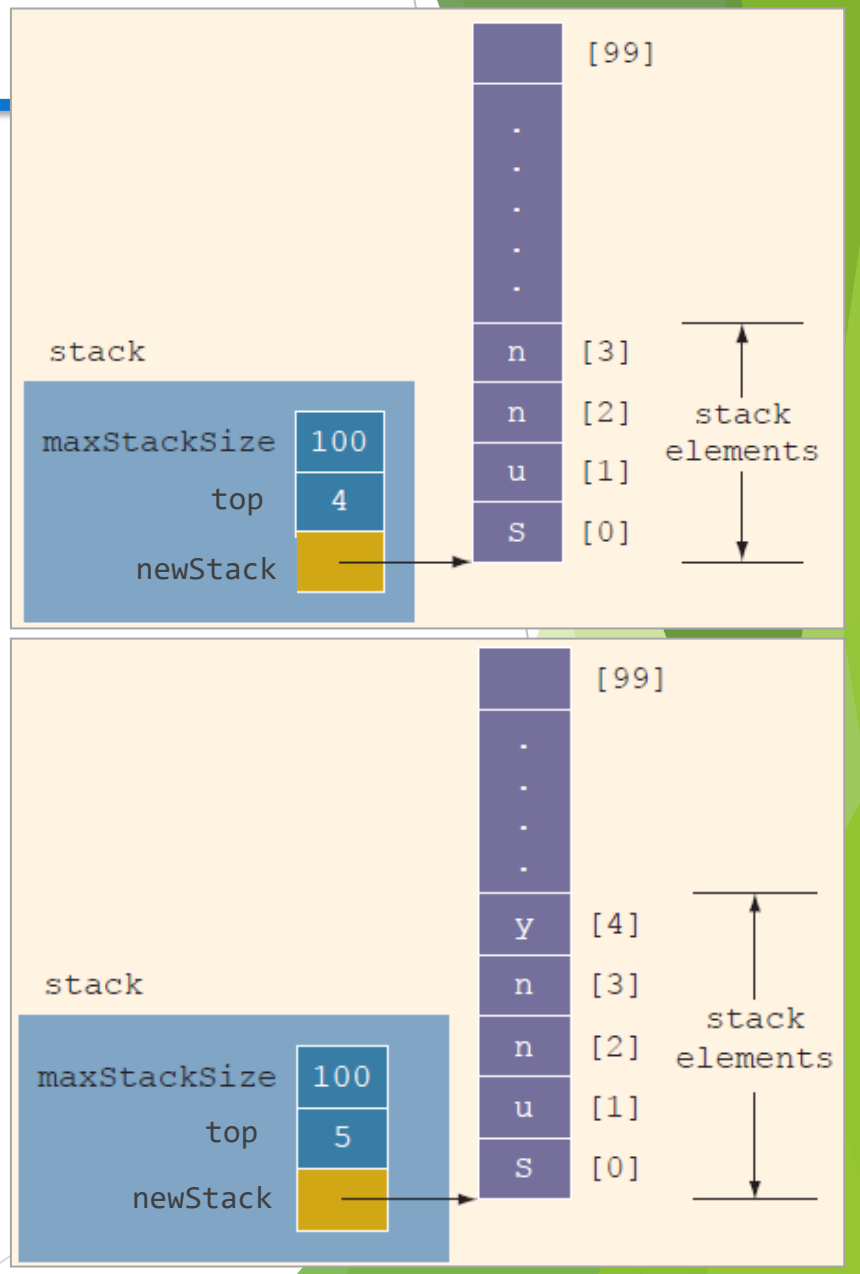
```
public boolean isEmpty() {
    return (top == 0);
}

public boolean isFull() {
    return (top == maxStackSize);
}
```

Stack Operations (cont.)

- Push

```
public void push(int elem) {
    if (!isFull())
        newStack[top++] = elem;
    else
        System.out.println("Full Stack");
}
```

May use StackOverflowException

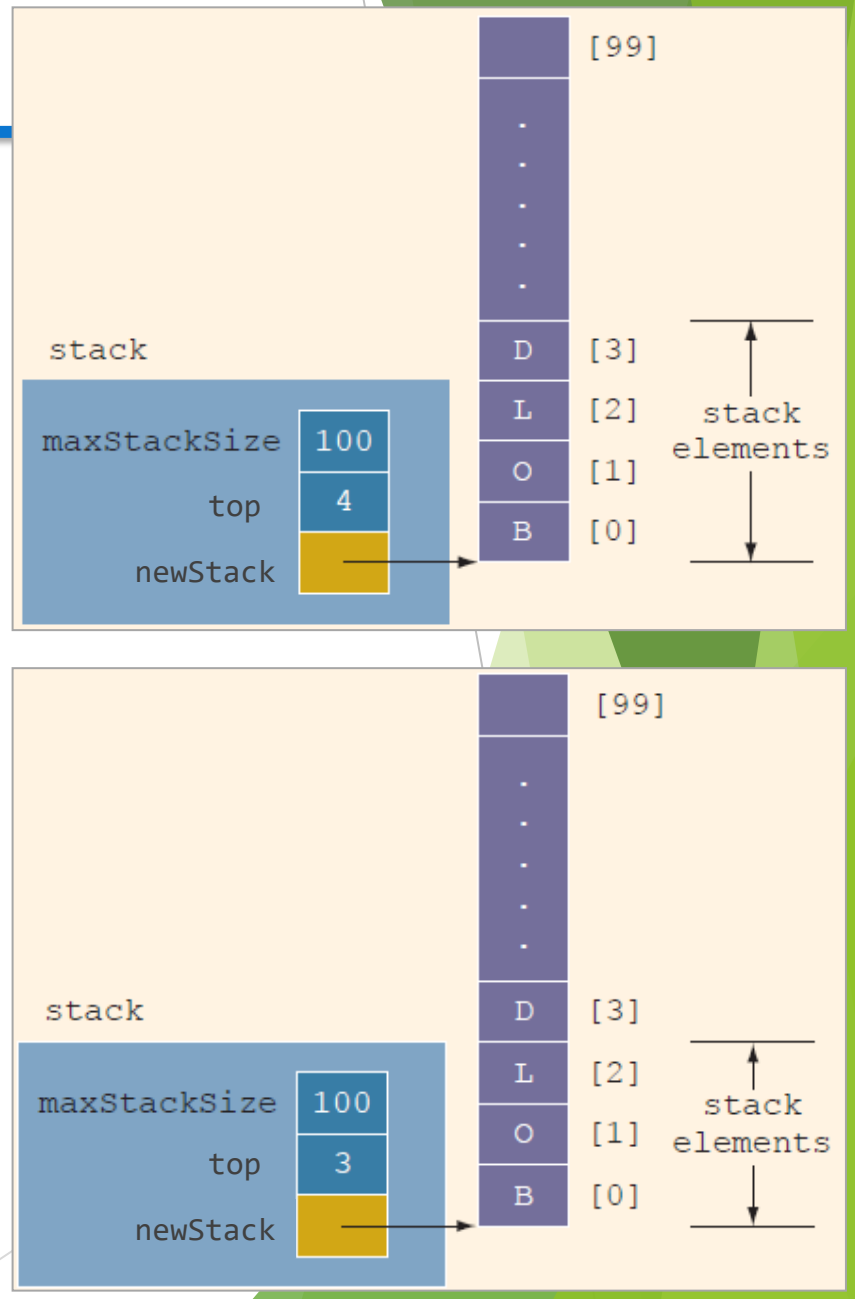


Stack Operations (cont.)

- Pop

```
public int pop() {
    if (!isEmpty())
        return(newStack[--top]);
    else {
        System.out.println("Empty Stack");
        return 0;
    }
}
```

May use StackUnderflowException



Testing

```
public static void main(String[] args) {
    MyStack S1 = new MyStack();
    S1.print();
    S1.push(6);
    S1.push(7);
    S1.push(3);
    S1.print();
    S1.pop();
    S1.print();
    S1.push(4);
    S1.push(9);
    S1.push(1);
    S1.pop();
    S1.print();
}
```

```
Stack content
Empty Stack
Stack content
6
7
3
Stack content
6
7
Stack content
6
7
4
9
```

Linked list Implementation of Stack

- Push - New element is added as the first node
- Pop – First element is removed every time
- head reference variable is renamed as top
- The stack will never be full

Summary

- A list of homogeneous elements; addition and deletion occur only at one end of it
- Based on last in first out algorithms (LIFO)
- Stack application : Parentheses matching, Infix-postfix
- Stack Implementation : `Stack` class, array, linked list

Next Topic…

- Queue
  - Concept
  - Application
  - Implementation

References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.