CM1602 : Data Structures and Algorithms for AI

4. Stack

Lecture 4 | R. Sivaraman

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

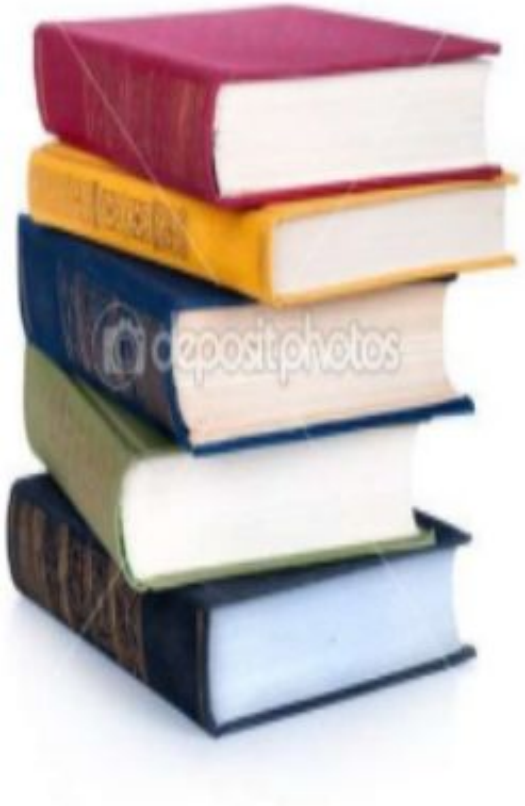
MODULE CONTENT

Lecture	Topic
Lecture 01	Introduction to Fundamentals of Algorithms
Lecture 02	Analysis of Algorithms
Lecture 03	Array and Linked Lists
Lecture 04	Stack
Lecture 05	Queue
Lecture 06	Searching algorithms and Sorting algorithms
Lecture 07	Trees
Lecture 08	Maps, Sets, and Lists
Lecture 09	Graph algorithms

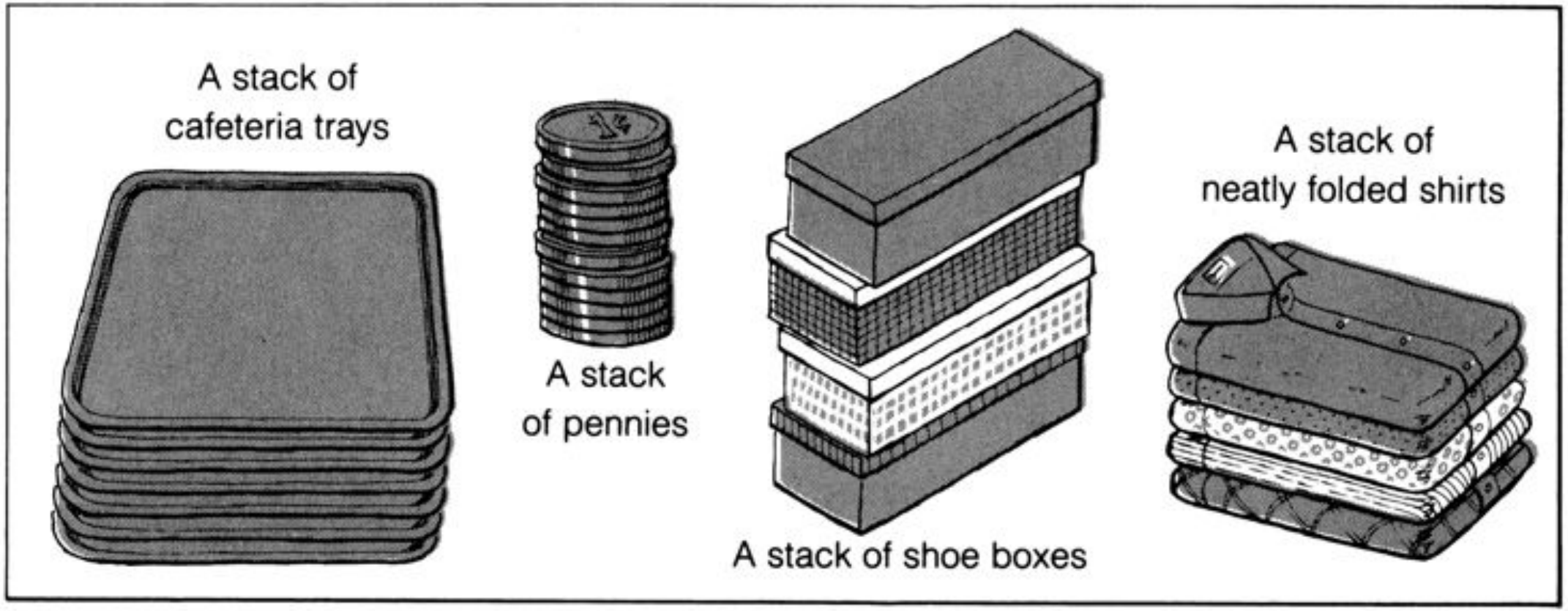
Learning Outcomes

- LO1 : Describe the fundamental concepts of algorithms and data structures.
- LO3 : Apply appropriate data structures given a real-world problem to meet requirements of programming language APIs.
- On completion of this lecture, students are expected to be able to:
  - Describe Stack and when stack is used
  - Implement Stack using an Array or Linked List

Introduction



Introduction

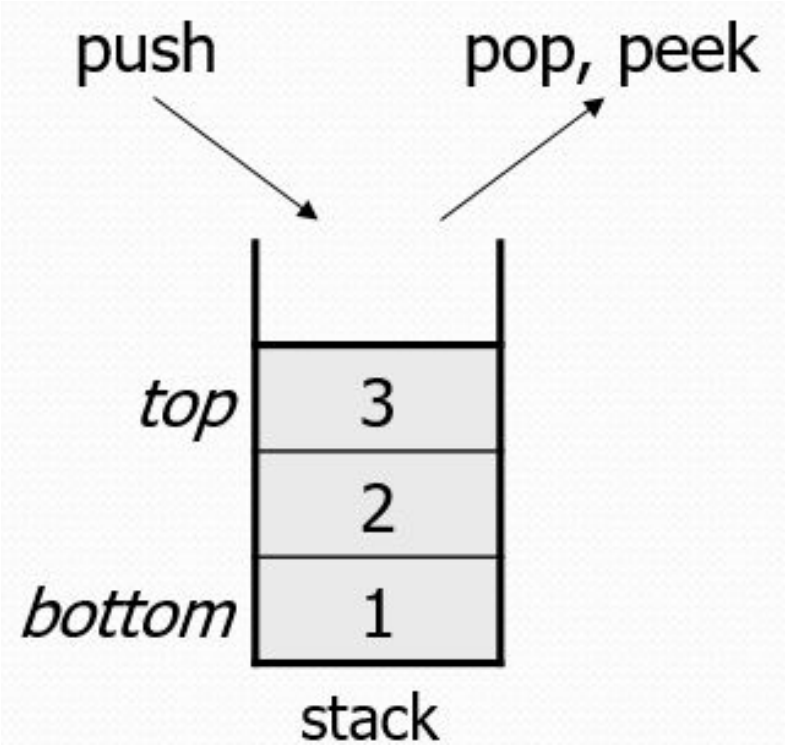


Introduction

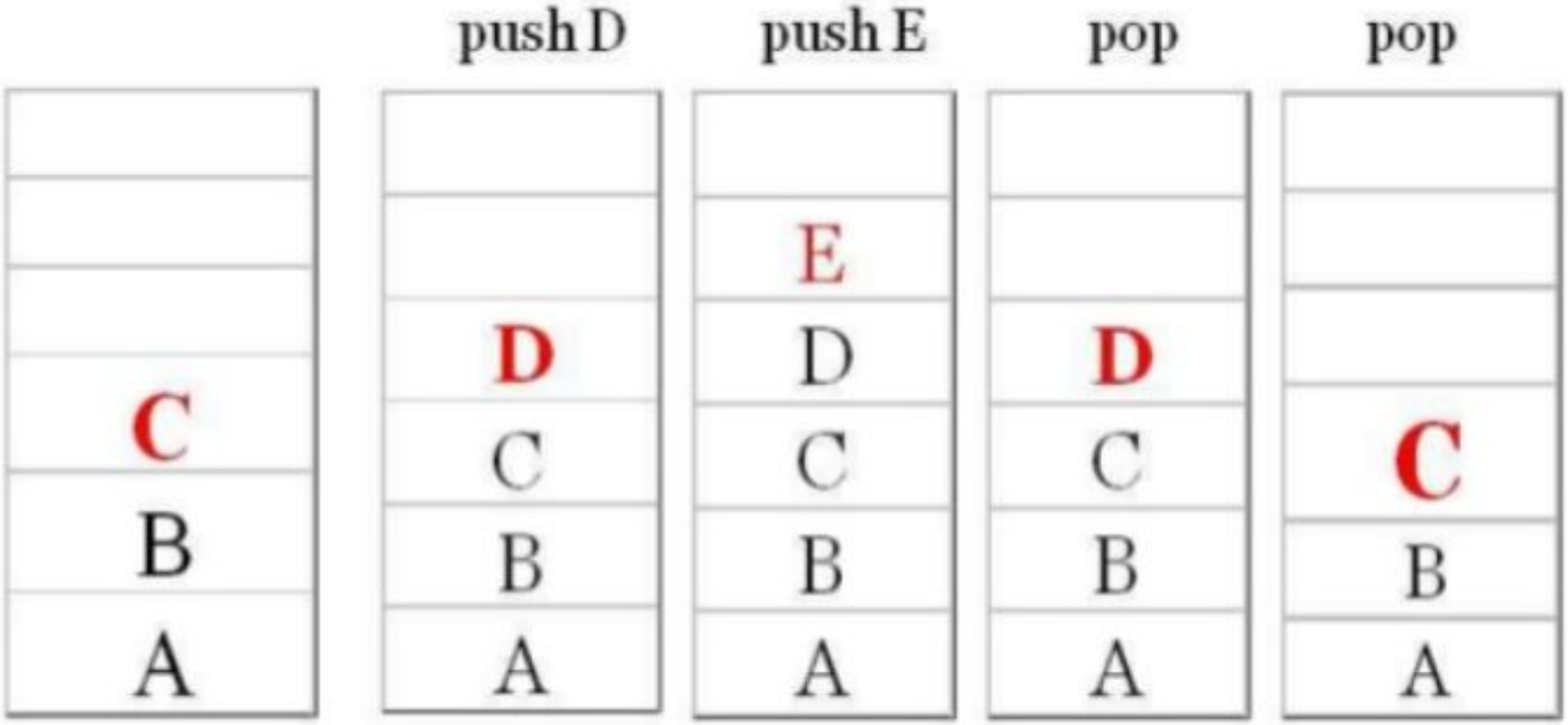
- Stack is a Linear Data Structure.
- It is an ordered group of homogeneous items of elements.
- Stack follows Last-In-First-Out (**LIFO**) principle.
- Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack).

Introduction

- basic stack operations:
  - push: Add an element to the top.
  - pop: Remove the top element.
  - peek: Examine the top element.
  - is Empty: Check if stack is empty
  - Is Full: Check if stack is full



Introduction



Stacks in computer science

- Programming languages and compilers:
  - method calls are placed onto a stack *(call=push, return=pop)*
  - compilers use stacks to evaluate expressions

- Matching up related pairs of things:
  - find out whether a string is a palindrome
  - examine a file to see if its braces { } match
  - convert "infix" expressions to pre/postfix

- Sophisticated algorithms:
  - searching through a maze with "backtracking"
  - many programs use an "undo stack" of previous operations

Stack Class

- Definitions: (provided by the user)
  - *MAX_ITEMS*: Max number of items that might be on the stack
  - Stack Array.
  - *Top – Top of the array – what is the initial value?*
- Operations
  - Peek()
  - Push ()
  - Pop ()
  - IsEmpty ()
  - IsFull ()

Stack Class

```
1  class Stack {
2      int maxSize;
3      int top;
4      int stackArray[];
5
6      Stack(int max)
7      {
8          top = -1;
9          maxSize = max;
10         stackArray = new int[maxSize];
11     }
```

Push Method

- *Function*: Adds new Item to the top of the stack.
- *Preconditions*: Stack has been initialized and is not full.
- *Post conditions*: new Item is at the top of the stack.
- Steps:
  - Check if stack is not full.
  - Increment top.
  - Add the value on top.

Push Method

```
24      void push(int x)
25      {
26          if (isFull()) {
27              System.out.println("Stack Overflow");
28          }
29          else {
30              stackArray[++top] = x;
31          }
32      }
33
```

Pop Method

- *Function*: Removes top Item from stack and returns it in item.
- *Preconditions*: Stack has been initialized and is not empty.
- *Post conditions*: Top element has been removed from stack and item is a copy of the removed element.
- Steps:
  - Check if stack is not empty.
  - Return the value on top.
  - Decrease top.

Pop Method

```
34      int pop()
35      {
36          if (isEmpty()) {
37              System.out.println("Stack Underflow");
38              return -1;
39          }
40          else {
41              //int x = stackArray[top];
42              //top--;
43              //return x;
44              return stackArray[top--];
45          }
46      }
```

Peek Method

```
48      int peek()
49      {
50          if (isEmpty()) {
51              System.out.println("Stack Underflow");
52              return -1;
53          }
54          else {
55              return stackArray[top];
56          }
57      }
```

IsEmpty() and IsFull() methods

```
13      boolean isEmpty()
14      {
15          return (top == -1);
16      }
17
18      boolean isFull()
19      {
20          return (top == (maxSize - 1);
21      }
22
```

Stack Implementation in Linked List

- Stack can be implemented by restricting Insertion and Deletion only at the end of the Linked List.
- Push() – Insert Last method of Linked List
- Pop() – Delete Last method of Linked List

Stack Node

```
class StackNode {
    int data;
    StackNode next;

    StackNode(int data)
    {
        this.data = data;
    }
}
```

StackLinkedList Class

```
public class StackAsLinkedList {

    StackNode head;
```

IsEmpty Method

```
public boolean isEmpty()
{
    if (head == null) {
        return true;
    }
    else
        return false;
}
```

IsFull Method

- This stack will never be full because it is implemented using Linked List

Push Method

```
public void push(int data)
{
    StackNode newNode = new StackNode(data);

    if (root == null) {
        root = newNode;
    }
    else {
        StackNode temp = root;
        root = newNode;
        newNode.next = temp;
    }
    System.out.println(data + " pushed to stack");
}
```

Pop Method

```
public int pop()
{
    int popped = Integer.MIN_VALUE;
    if (root == null) {
        System.out.println("Stack is Empty");
    }
    else {
        popped = root.data;
        root = root.next;
    }
    return popped;
}
```

Peek Method

```
public int peek()
{
    if (root == null) {
        System.out.println("Stack is empty");
        return Integer.MIN_VALUE;
    }
    else {
        return root.data;
    }
}
```