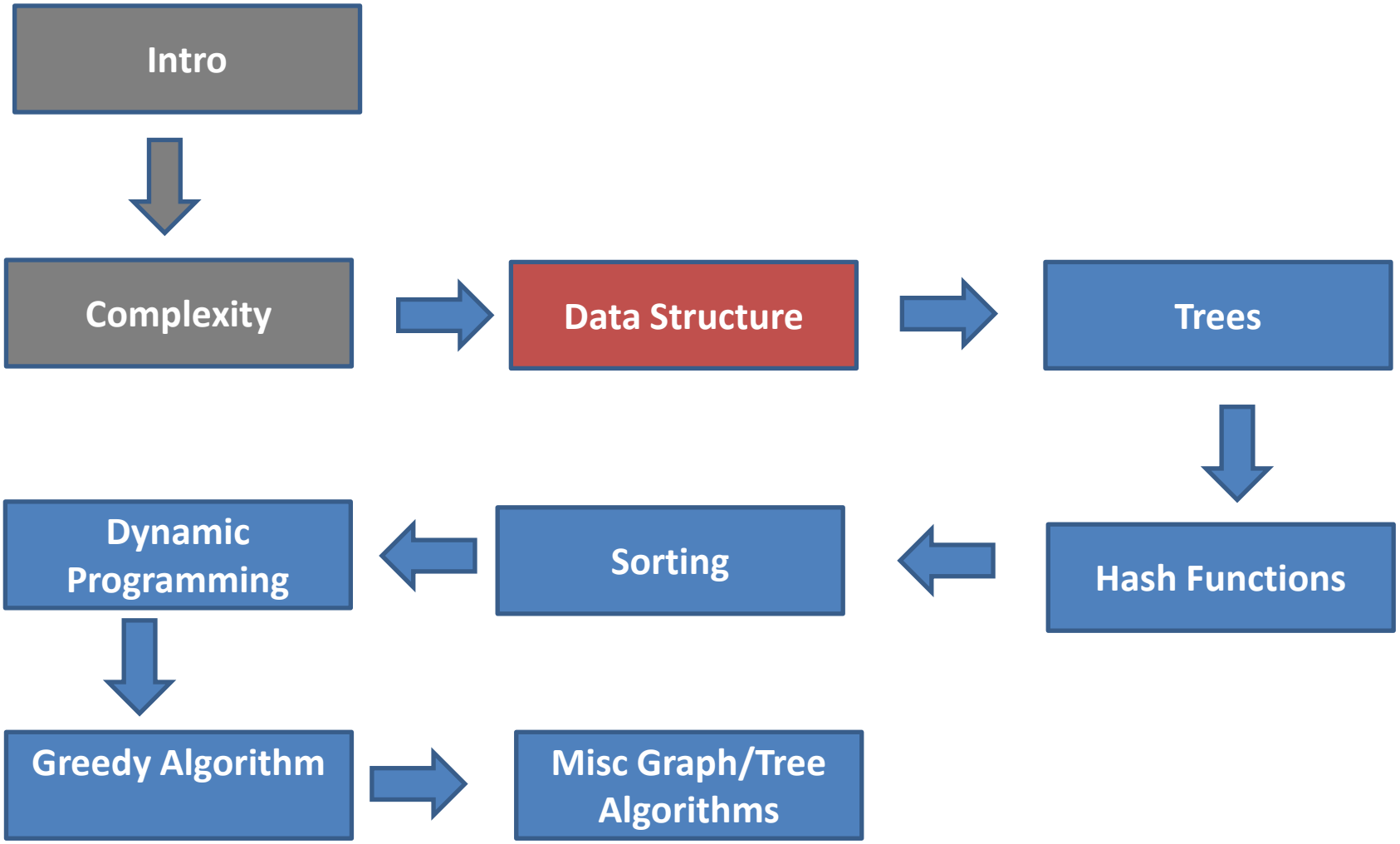
An Introduction to Algorithms

By

Hossein Rahmani

h_rahmani@iust.ac.ir

http://webpages.iust.ac.ir/h_rahmani/


Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

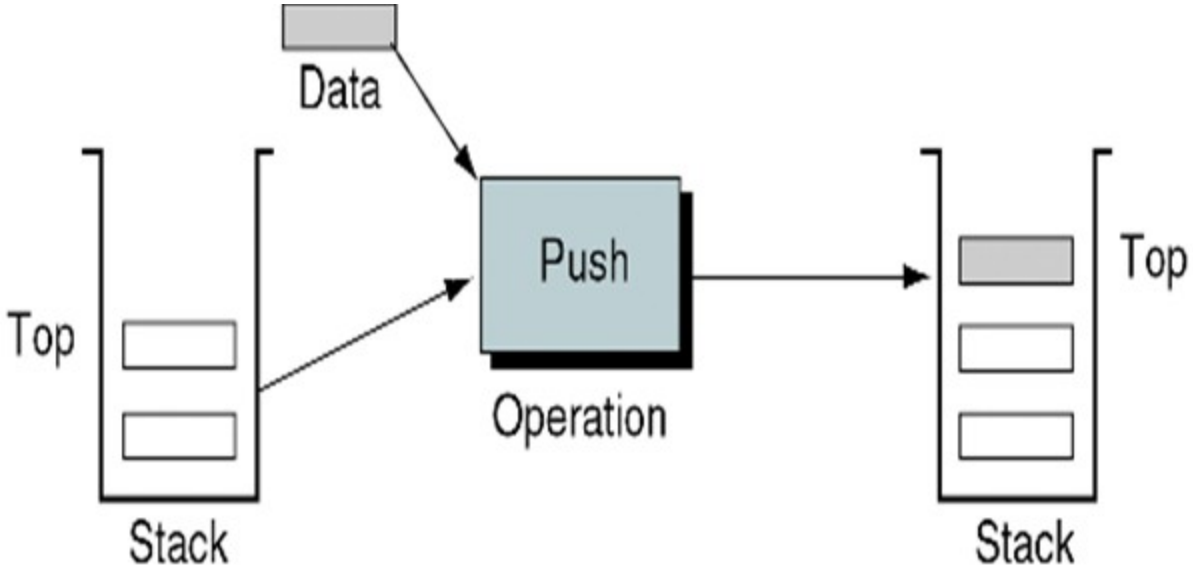
Stack of coins Stack of books Computer stack

Stack

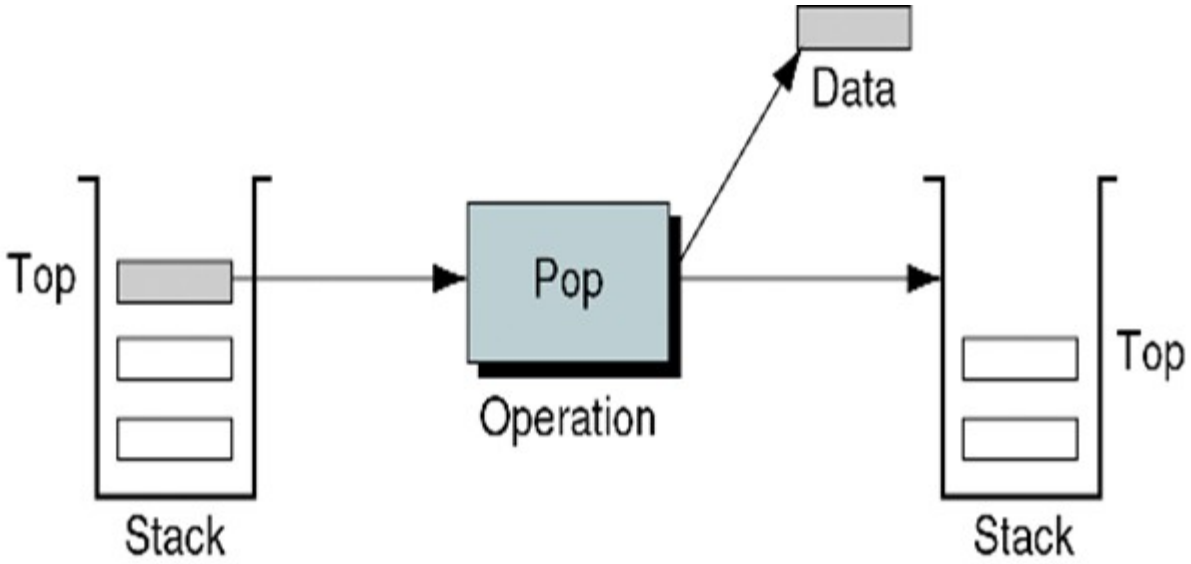
- A stack is a data structure that stores data in such a way that the last piece of data stored, is the first one retrieved
  - also called last-in, first-out
- Only access to the stack is the top element

Basic Stack Operations

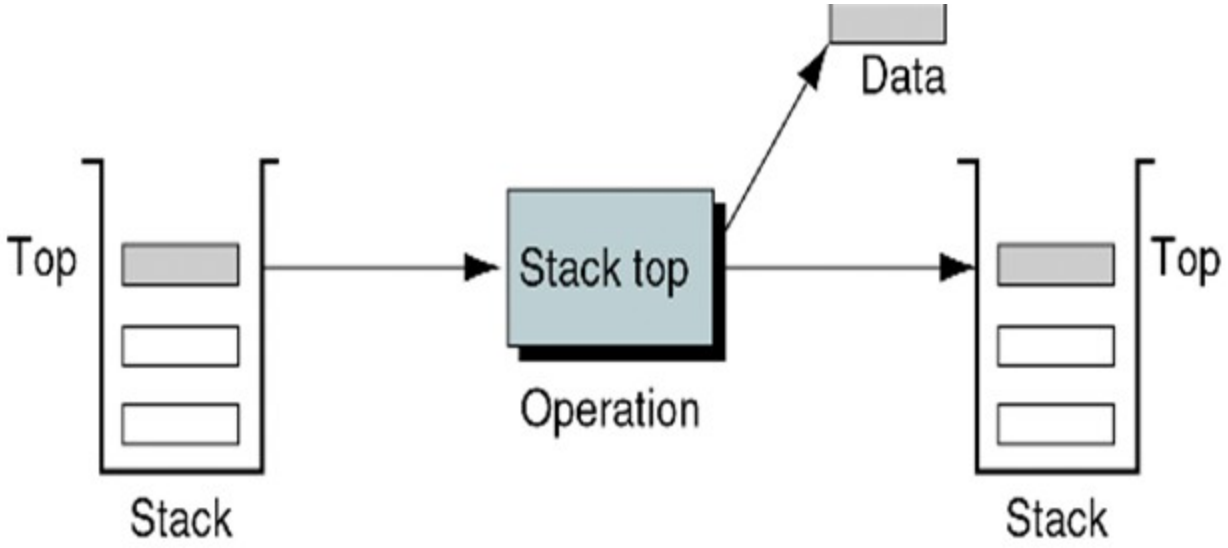
- *Push*
  - the operation to place a new item at the top of the stack
- *Pop*
  - the operation to remove the next item from the top of the stack



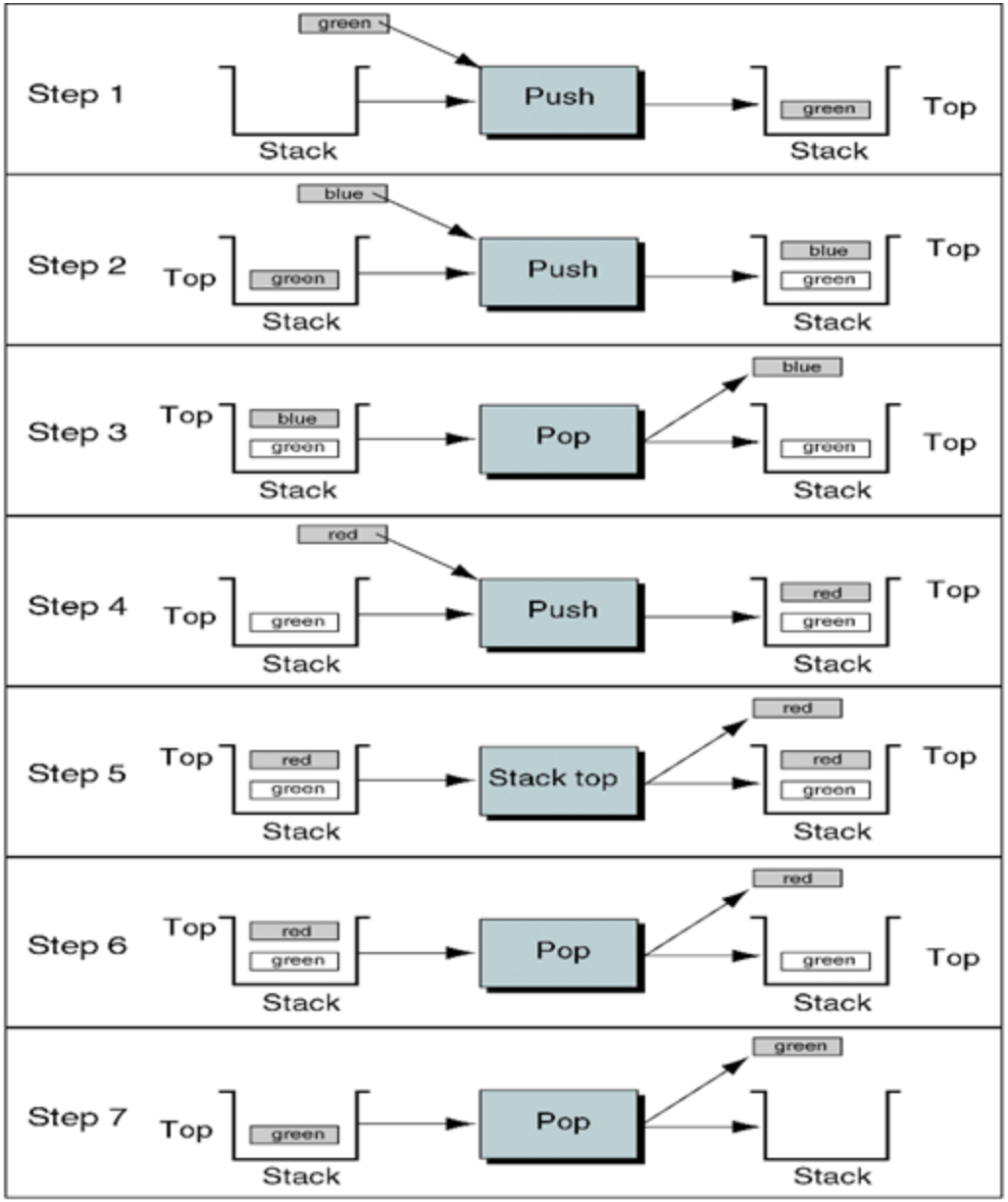
Push Stack Operation



Pop Stack Operation



Stack Top Operation



Stack Example

Implementing a Stack

- At least two different ways to implement a stack
  - array
  - linked list
- Which method to use depends on the application
  - what advantages and disadvantages does each implementation have?

Implementing Stacks: Array

- Advantages
  - best performance
- Disadvantage
  - fixed size
- Basic implementation
  - initially empty array
  - field to record where the next data gets placed into
  - if array is full, push() returns false
    - otherwise adds it into the correct spot
  - if array is empty, pop() returns null
    - otherwise removes the next item in the stack

Stack Class (array based)

```
class StackArray {
    private Object[ ] stack;
    private int nextIn;
    public StackArray(int size) {
        stack = new Object[size];
        nextIn = 0;
    }
    public boolean push(Object data);
    public Object pop();
    public void clear();
    public boolean isEmpty();
    public boolean isFull();
}
```

push() Method (array based)

```
public boolean push(Object data) {
    if(nextIn == stack.length) { return false; }   // stack is full

    // add the element and then increment nextIn
    stack[nextIn] = data;
    nextIn++;
    return true;
}
```

pop() Method (array based)

```
public Object pop() {
    if(nextIn == 0) { return null; }   // stack is empty

    // decrement nextIn and return the data
     nextIn--;
    Object data = stack[nextIn];
    return data;
}
```

Remaining Methods (array based)

```
public void clear() {
    nextIn = 0;
}

public boolean isEmpty() {
    return nextIn == 0;
}

public boolean isFull() {
    return nextIn == stack.length;
}
```

Implementing a Stack: Linked List

- Advantages:
  - always constant time to push or pop an element
  - can grow to an infinite size
- Disadvantages
  - the common case is the slowest of all the implementations
  - can grow to an infinite size
- Basic implementation
  - list is initially empty
  - *push()* method adds a new item to the head of the list
  - *pop()* method removes the head of the list

Stack Class (list based)

```
class StackList {
    private LinkedList list;
    public StackList() { list = new LinkedList(); }
    public void push(Object data) { list.addHead(data); }
    public Object pop() { return list.deleteHead(); }
    public void clear() { list.clear(); }
    public boolean isEmpty() { return list.isEmpty(); }
}
```

Additional Notes

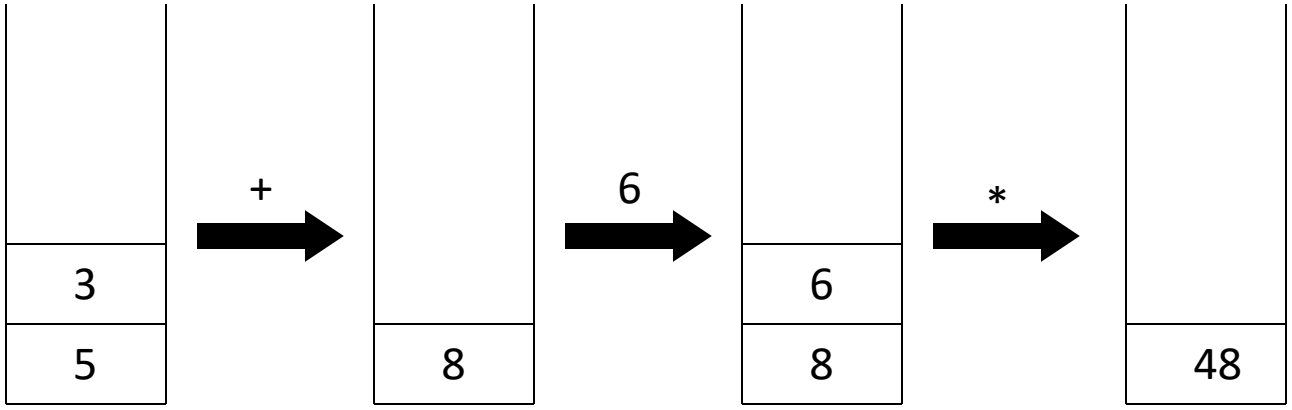
- It should appear obvious that linked lists are very well suited for stacks
  - *addHead()* and *deleteHead()* are basically the *push()* and *pop()* methods
- Our original list implementation did not have a *clear()* method
  - it's very simple to do
  - how would you do it?
- Again, no need for the *isFull()* method
  - list can grow to an infinite size

Stack Applications

- Stacks are a very common data structure
  - compilers
    - parsing data between delimiters (brackets)
  - operating systems
    - program stack
  - virtual machines
    - manipulating numbers
      - pop 2 numbers off stack, do work (such as add)
      - push result back on stack and repeat

Reverse Polish Notation

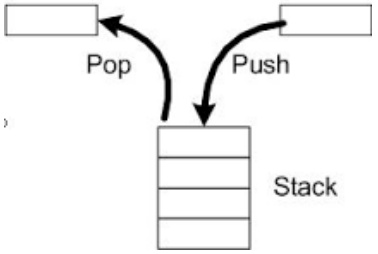
- Way of inputting numbers to a calculator
  - (5 + 3) * 6 becomes 5 3 + 6 *
  - 5 + 3 * 6 becomes 5 3 6 * +
- We can use a stack to implement this
  - consider 5 3 + 6 *



  - try doing 5 3 6 * +

```
public int rpn(String equation) {
    StackList stack = new StackList();
    StringTokenizer tok = new StringTokenizer(equation);
    while(tok.hasMoreTokens()) {
      String element = tok.nextToken();
      if(isOperator(element)) {
        char op = element.charAt(0);
        if(op == '=') {
            int result = ((Integer)stack.pop()).intValue();
            if(!stack.isEmpty() || tok.hasMoreTokens()) { return Integer.MAX_VALUE; }  // error
            else { return result; }
        }
        else {
            Integer op1 = (Integer)stack.pop()
            Integer op2 = (Integer)stack.pop();
            if((op1 == null) || (op2 == null)) { return Integer.MAX_VALUE; }
            stack.push(doOperation(op, op1, op2));
         }
      }
      else {
        Integer operand = new Integer(Integer.parseInt(element));
        stack.push(operand);
      }
    }
    return Integer.MAX_VALUE;
}
```

Quiz 1



- Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing.

```
void fun(int n)
{
    Stack S;  // Say it creates an empty stack S
    while (n > 0)
    {
      // This line pushes the value of n%2 to stack S
      push(&S, n%2);

      n = n/2;
    }

    // Run while Stack S is not empty
    while (!isEmpty(&S))
      printf("%d ", pop(&S)); // pop an element from S and print it
}
```

- What does the above function do in general?

Quiz 2

Consider the following C program:

```
#include <stdio.h>
        #define EOF -1
        void push (int); /* push the argument on the stack */
        int pop  (void); /* pop the top of the stack */
        void flagError ();
        int main ()
       {        int c, m, n, r;
                 while ((c = getchar ()) != EOF)
                { if  (isdigit (c) )
                            push (c);
                 else if ((c == '+') || (c == '*'))
                {           m = pop ();
                             n = pop ();
                             r = (c == '+') ? n + m : n*m;
                             push (r);
                  }
                  else if (c != ' ')
                            flagError ();
           }
            printf("% c", pop ());
}
```

What is the output of the program for the following input ?

5 2 * 3 3 2 + * +

Quiz 3

- Following is an incorrect pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced: Which of these unbalanced sequences does the code think is balanced?

A	((())
B	())(()
C	(()()))
D	(()))()

```
declare a character stack
while ( more input is available)
{
   read a character
   if ( the character is a '(' )
      push it on the stack
   else if ( the character is a ')' and the stack is not empty )
      pop a character off the stack
   else
      print "unbalanced" and exit
 }
 print "balanced"
```