ITEC2620 Introduction to Data Structures

Lecture 7a
ADTs and Stacks

Abstract Data Types

- A way to specify the functionality of an entity without worrying about its implementation
- Similar to a JAVA interface or an API

List ADT I

```
interface SortedList
{
        public void insert (Object item);
        public Object remove ();
        public boolean isInList ();
        …
}
```

List ADT II

- How can we implement a SortedList?
  - Arrays
  - Linked-list
  - Binary search tree

List ADT III

- Does the implementation affect the available functions?
  - No
- Does the implementation affect the time to execute each function?
  - Yes – e.g. insert
  - **O**(n), **O**(n), **O**(ℓogn)

Stacks I

- A stack is a “Last-In, First-Out” = “LIFO” buffer
  - e.g. dinner plates at the buffet line
    - ➢Plates are added to the top of the stack
    - ➢Plates are removed from the top of the stack

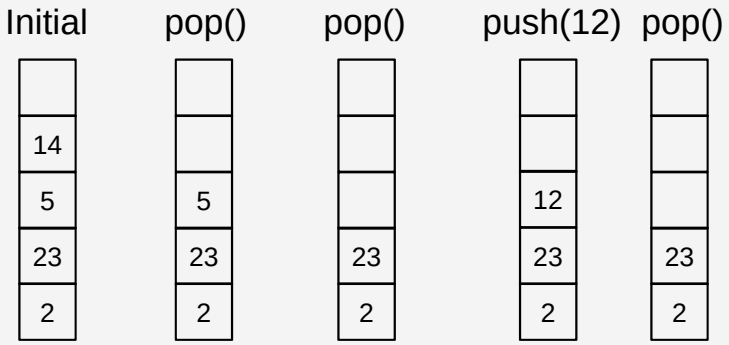
Stacks II

- Only the **top** element of a stack is accessible
  - When elements are added, they are **pushed** onto the top
  - When elements are removed, they are **popped** off of the top
- push() and pop() are the two defining functions of a stack

Example of Stacks I

- Stacks are usually drawn as “vertical arrays”

Example of Stacks II



Implementation of Stacks

- Stacks are an ADT
  - Functionality of stack has been specified
  - Implementation of stack has not
- A stack can be implemented with any data structure that can provide access to the “top” element
  - Expect efficiency (i.e. **O**(1))

Array-Based Implementation I

- Move a “top” pointer (to the first empty spot) up and down

```
public class Stack
{
    private int size;
    private int top;
    private Object[] stackArray;
```

Array-Based Implementation II

```
public boolean push (Object item)
{
    // stack overflow error – stack out of memory
    if (top = stackArray.length)
        return false;

    stackArray[top] = item;
    top++;
    return true;
}
```

Array-Based Implementation III

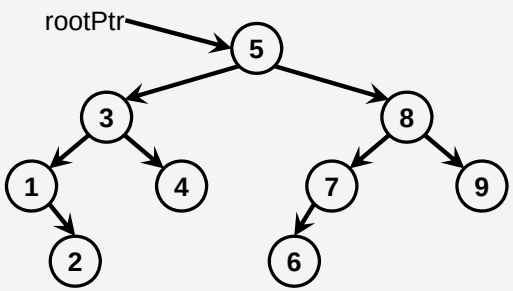
```
public Object pop ()
{
    // stack underflow error – nothing in the stack
    if (top == 0)
        return null;

    top--;
    return stackArray[top];
}
}
```

Stack-based Recursion

- Computers use a control stack to manage function calls/function returns
  - When a function is called, data is pushed onto the stack
  - When a function finishes, data is popped off of the stack
- Instead of the computer's stack, we can use our own to implement recursion

Stack-based Tree Traversal I



Stack-based Tree Traversal II

- Need a keep information about where we are in the tree…

```
public class BinaryNode
{
    public BinaryNode left;
    public BinaryNode right;
    public int key;
    public boolean expanded;
}
```

Stack-based Tree Traversal III

```
public void printTree (BinaryNode root)
{
    Stack treeStack = new Stack();
    BinaryNode current = root;

    // first time (expanded == false), do left, this, right
    // second time (expanded == true), just print
    root.expanded = false;
```

Stack-based Tree Traversal IV

```
    while (current != null)
    // when the stack is empty,
    // you've finished the tree traversal
    {
        // second time (expanded == true),
        // just print
        if (current.expanded)
            System.out.println(current.key);
```

Stack-based Tree Traversal V

```
        else
        {
            // stack is last in, first out buffer
            // want to process left first (in order),
            // so it goes in last, right goes first
            if (current.right != null)
            {
                current.right.expanded = false;
                treeStack.push(current.right)
            }
```

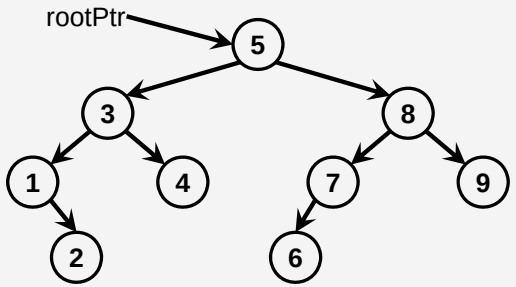
Stack-based Tree Traversal VI

```
            // we are in the process of expanding current
            // this is the first time,
            // next time will be the second time
            current.expanded = true;
            treeStack.push(current);
```

Stack-based Tree Traversal VII

```
            if (current.left != null)
            {
                current.left.expanded = false;
                treeStack.push(current.left)
            }
        }
        // finished expansion, get next item from stack
        current = (BinaryNode) treeStack.pop();
    }
}
```

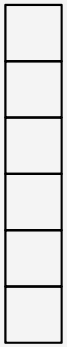
Stack Trace I



Stack Trace II

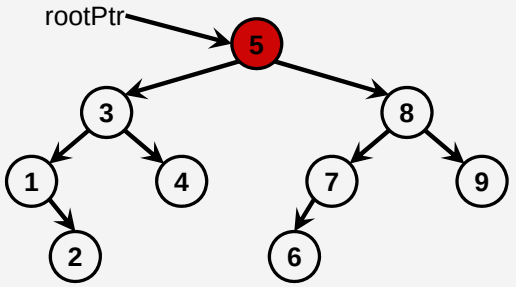
printTree(rootPtr);

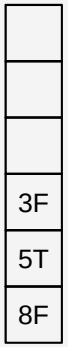
- Stack is initially empty
- root/current is not expanded
- Expand and populate stack



Stack Trace III

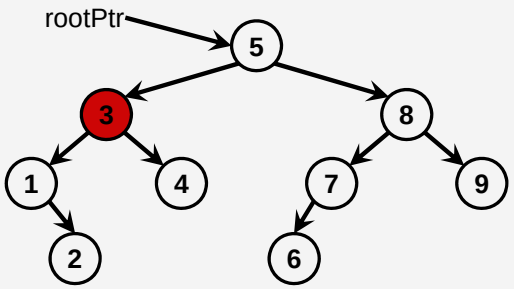
Expand root





Stack Trace IV

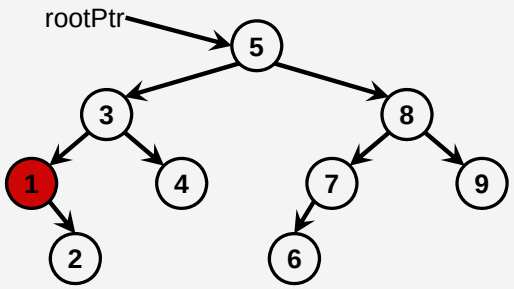
pop() → 3F



1F
3T
4F
5T
8F

Stack Trace V

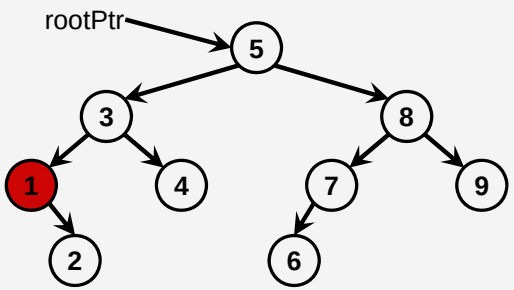
pop() → 1F



1T
2F
3T
4F
5T
8F

Stack Trace VI

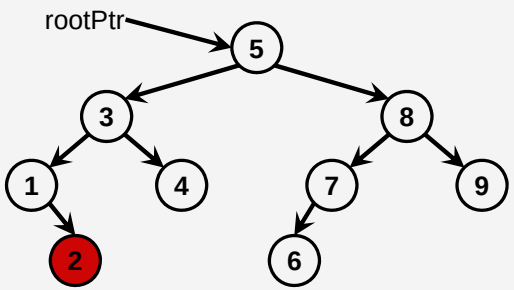
pop() → 1T Print 1



1T
2F
3T
4F
5T
8F

Stack Trace VII

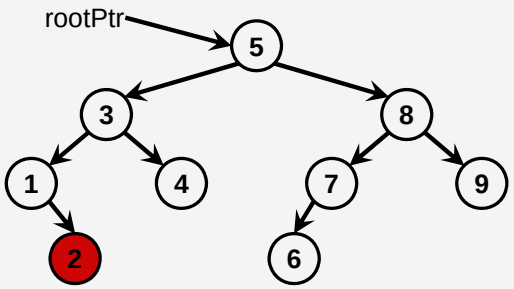
pop() → 2F



2T
3T
4F
5T
8F

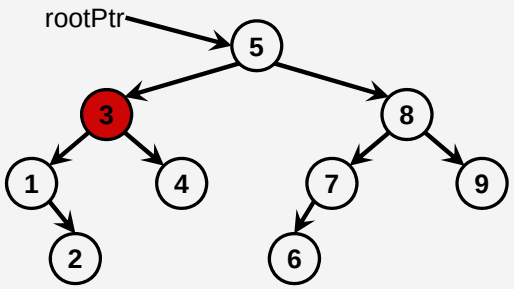
Stack Trace VIII

pop() → 2T Print 2



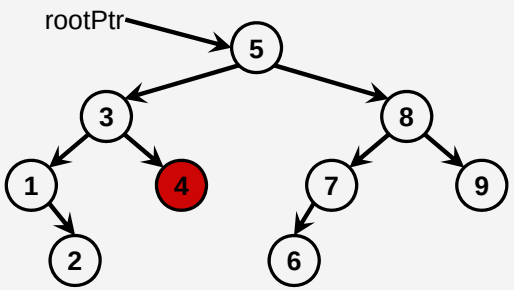
Stack Trace IX

pop() → 3T Print 3



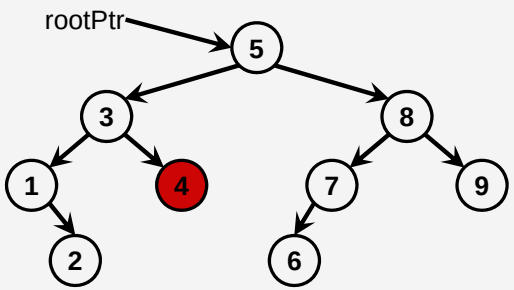
Stack Trace X

pop() → 4F



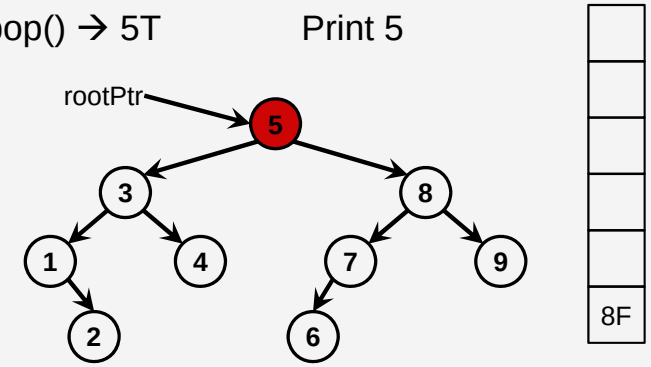
Stack Trace XI

pop() → 4T Print 4



Stack Trace XII

pop() → 5T Print 5



Stack Trace XIII

- First time a node is red
  - Expand it
- Second time a node is red
  - Print it

Stacks and Recursion

- "Any recursive algorithm can be implemented non-recursively"
  - Do you have to use a function that calls itself?
    - ➢No, use a stack to implement recursion
  - Do you have to use a recursive algorithm?
    - ➢Yes, how do you traverse a binary tree without a recursive algorithm?

Stacks in Use I

- Stacks are used to manage the control flow in a computer
  - When a method is called, the calling method is put on the stack
  - When a method returns, control is passed to the top method on the stack
- The last method to call another method is the first to get control back – LIFO

Stacks in Use II

- What do you get when your JAVA program crashes?
  - The control stack
  - main() is at the bottom
  - Internal JAVA methods are at the top
- The last method that the computer executed (that you wrote) is somewhere in between…find it!

Readings and Assignments

- Suggested Readings from Shaffer (third edition)
  - 1.2, 4.1, 4.2