

STACKS

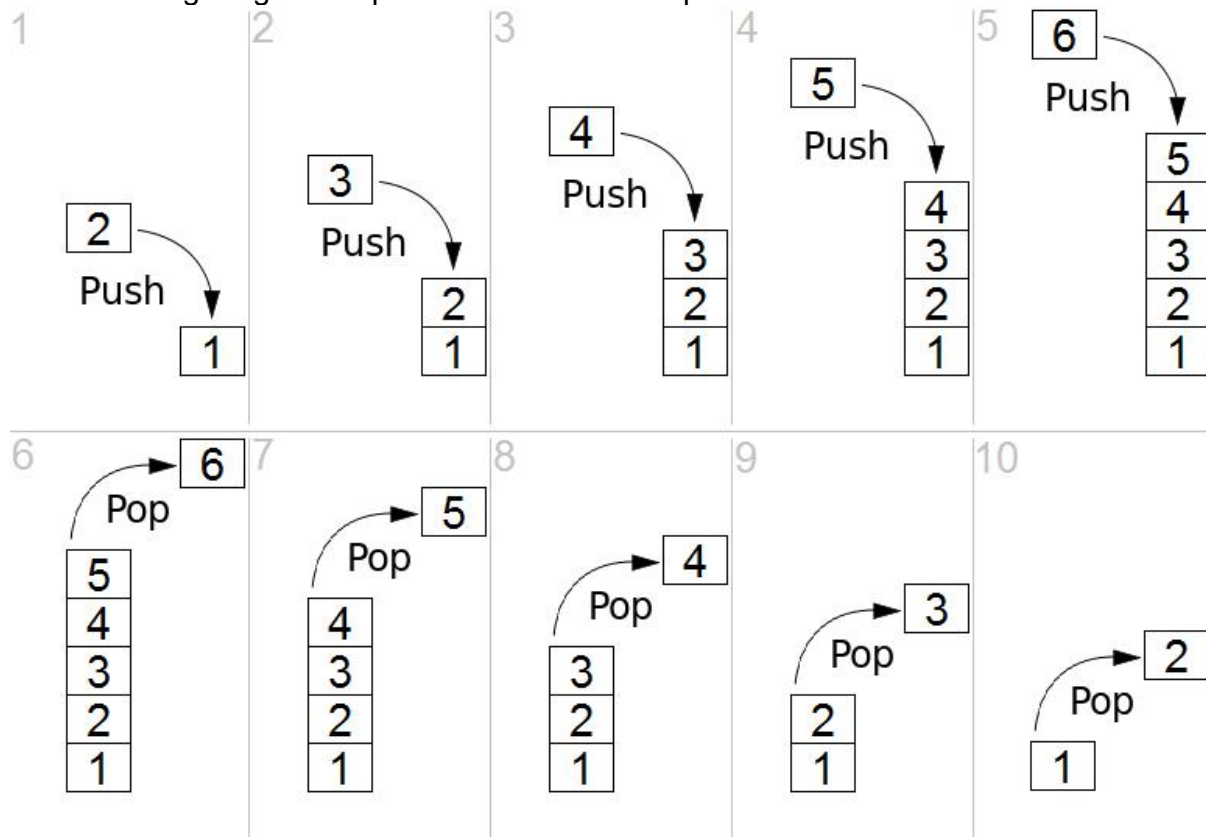
A stack is an **Abstract Data Type (ADT)**, commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

Stack Representation

The following diagram depicts a stack and its operations –



A stack can be implemented by means of Array or Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Using Inbuilt Stack class

```
import java.util.Stack;
class StackUse{
    public static void main(String[] args){
        Stack<Integer> stack1 = new Stack<>();
```

```
//Stack<Character> stack2 = new Stack<>();  
//Stack<YourOwnClass> stack3 = new Stack<>();
```

```
stack1.add(10);// Appends 10 at end of stack
```

```
stack1.push(20);// Appends 20 after 10 (does same work as add())
```

```
stack1.peek();// Looks at the Integer at the top of this stack without removing it from the  
stack.
```

```
stack1.pop();// Removes the Integer at the top of this stack and returns that object as the  
value of this function.
```

```
stack1.remove(1);// Removes the element at the specified index in this Stack. Shifts any  
subsequent elements to the left (subtracts one from their indices). Returns the element that  
was removed from the Stack. As it violates property of stack it should not be used.
```

```
st.size();// Returns the number of components in this vector.
```

```
}  
}
```

Note:

By default initial capacity of Stack is 10. Basically an object array is created at the back. When the size reaches maximum capacity the capacity is doubled.

Basic Operations

A stack is used for the following two primary operations –

- 1) push() – Pushing (storing) an element on the stack.
- 2) pop() – Removing (accessing) an element from the stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks –

- 3) peek() – get the top data element of the stack, without removing it.
- 4) isEmpty() – check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named TOP. The TOP provides top value of the stack without actually removing it.

Push Operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps –

- Step 1 – Checks if the stack is full.
- Step 2 – If the stack is full, produces an error and exit.
- Step 3 – If the stack is not full, increments top to point next empty space.
- Step 4 – Adds data element to the stack location, where top is pointing.
- Step 5 – Returns success.

If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.

Pop Operation

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element.

A Pop operation may involve the following steps –

Step 1 – Checks if the stack is empty.

Step 2 – If the stack is empty, produces an error and exit.

Step 3 – If the stack is not empty, accesses the data element at which top is pointing.

Step 4 – Decreases the value of top by 1.

Step 5 – Returns the element.

```
public class stacks {

    private int[] stackArray;
    private int top;
    private int Size;

    stacks(int Size){
        top = -1;
        this.Size = Size;
        stackArray = new int[Size]; // initially empty
    }

    public int pop() throws StackEmptyException{
        if(top == -1){
            throw new StackEmptyException();
        }
        int topElement = stackArray[top];
        top--;
        return topElement;
    }

    public void push(int Element) throws StackOverFlowException{
        if(top == Size-1){
            throws new StackOverFlowException();
        }
        top++;
        stackArray[top] = Element;
    }

    public int getSize(){
        return Size;
    }

    public int peek(){
        return stackArray[top];
    }
}
```

```

}

public boolean isEmpty(){
    return (top == -1)? true:false;
}
}

```

Stack Applications

Postfix Evaluation Algorithm

In normal algebra we use the infix notation like $a+b*c$. The corresponding postfix notation is $abc*+$. The algorithm for the conversion is as follows :

Step 1 - Scan the Postfix string from left to right.

Step 2- Initialise an empty stack.

Step 3 - If the scanned character is an operand, add it to the stack.

Step 4 - If the scanned character is an Operator, then we pop and store the top most element of the stack[top] in a variable OP2. Again Pop the stack and store the element in OP1. Now evaluate OP1(Operator)OP2. Let the result of this operation be retVal. Push retVal into the stack.

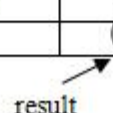
Step 5 - Repeat 3/4 step till all the characters are scanned.

Step 6 - After all characters are scanned, we will have only one element in the stack. Return stack[top].

The time complexity is $O(n)$ because each operand is scanned once, and each operation is performed once.

Example: $4325*-+=4+3-2*5=-3$

Symbol	opnd1	opnd2	value	opndstack
4				4
3				4, 3
2				4, 3, 2
5				4, 3, 2, 5
*	2	5	10	4, 3
				4, 3, 10
-	3	10	-7	4
				4, -7
+	4	-7	-3	
				(-3)



result

Infix transformation to Postfix

This process uses a stack as well. We have to hold information that's expressed inside parentheses while scanning to find the closing ')'. We also have to hold information on operations that are of lower precedence on the stack. The algorithm is:

Step 1 - Fix a priority level for each operator. For example, from high to low:

3) ' - ' (unary negation)

2) ' * ', ' / '

1) ' + ', ' - ' (subtraction)

Thus, high priority corresponds to high number in the table.

Step 2 - If the token is an operand, do not stack it. Pass it to the output.

Step 3 - If token is an operator or parenthesis, do the following:

3.1 Pop the stack until you find a symbol of lower priority number than the current one. An incoming left parenthesis will be considered to have higher priority than any other symbol. A left parenthesis on the stack will not be removed unless an incoming right parenthesis is found. The popped stack elements will be written to output.

3.2 Stack the current symbol.

3.3 If a right parenthesis is the current symbol, pop the stack down to (and including) the first left parenthesis. Write all the symbols except the left parenthesis to the output (i.e. write the operators to the output).

3.4 After the last token is read, pop the remainder of the stack and write any symbol (except left parenthesis) to output.

Infix to Postfix using stack ...

- Example $A * (B + C * D) + E$ becomes $A B C D * + * E +$

	current symbol	operator stack	postfix string
1	A		A
2	*	*	A
3	(*(A
4	B	*(A B
5	+	*(+	A B
6	C	*(+	A B C
7	*	*(+ *	A B C
8	D	*(+ *	A B C D
9)	*	A B C D * +
10	+	+	A B C D * + *
11	E	+	A B C D * + * E
12			A B C D * + * E +

Backtracking

Backtracking is used in algorithms in which there are steps along some path (state) from some starting point to some goal.

1. Find your way through a maze.
2. Find a path from one point in a graph (roadmap) to another point.

3. Play a game in which there are moves to be made (checkers, chess).

In all of these cases, there are choices to be made among a number of options. We need some way to remember these decision points in case we want/need to come back and try the alternative

Consider the maze. At a point where a choice is made, we may discover that the choice leads to a dead-end. We want to retrace back to that decision point and then try the other (next) alternative.

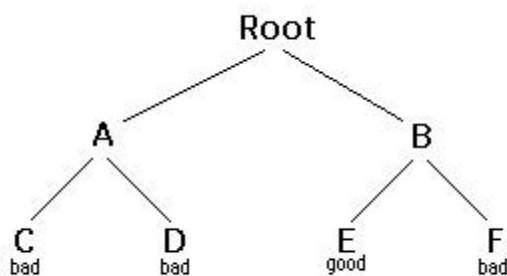
Again, stacks can be used as part of the solution. Recursion is another, typically more favored, solution, which is actually implemented by a stack.

Backtracking is a form of recursion.

The usual scenario is that you are faced with a number of options, and you must choose one of these. After you make your choice you will get a new set of options; just what set of options you get depends on what choice you made. This procedure is repeated over and over until you reach a final state. If you made a good sequence of choices, your final state is a goal state; if you didn't, it isn't.

Conceptually, you start at the root of a tree; the tree probably has some good leaves and some bad leaves, though it may be that the leaves are all good or all bad. You want to get to a good leaf. At each node, beginning with the root, you choose one of its children to move to, and you keep this up until you get to a leaf. Suppose you get to a bad leaf. You can backtrack to continue the search for a good leaf by revoking your most recent choice, and trying out the next option in that set of options. If you run out of options, revoke the choice that got you here, and try another choice at that node. If you end up at the root with no options left, there are no good leaves to be found.

Example.



1. Starting at Root, your options are A and B. You choose A.
2. At A, your options are C and D. You choose C.
3. C is bad. Go back to A.
4. At A, you have already tried C, and it failed. Try D.

5. D is bad. Go back to A.
6. At A, you have no options left to try. Go back to Root.
7. At Root, you have already tried A. Try B.
8. At B, your options are E and F. Try E.
9. E is good. Congratulations!

Note:

In this example we drew a picture of a tree. The tree is an abstract model of the possible sequences of choices we could make. There is also a data structure called a tree, but usually we don't have a data structure to tell us what choices we have. (If we do have an actual tree data structure, backtracking on it is called depth-first tree searching.)

Some more applications of stack

1. Evaluation of Prefix expression.
2. Convert Infix expression to Prefix.
3. Reverse string or list.
4. Tower of Hanoi.
5. Call Stack: call and return process of methods.
6. Navigating to previous webpages.
7. Undo feature in text editors.
8. Navigating to parent directory.