Stacks

Stack

- Linear Data Structure
- Access is allowed only at one point of the structure, normally termed the *top* of the stack
  - access to the most recently added item only

Stack

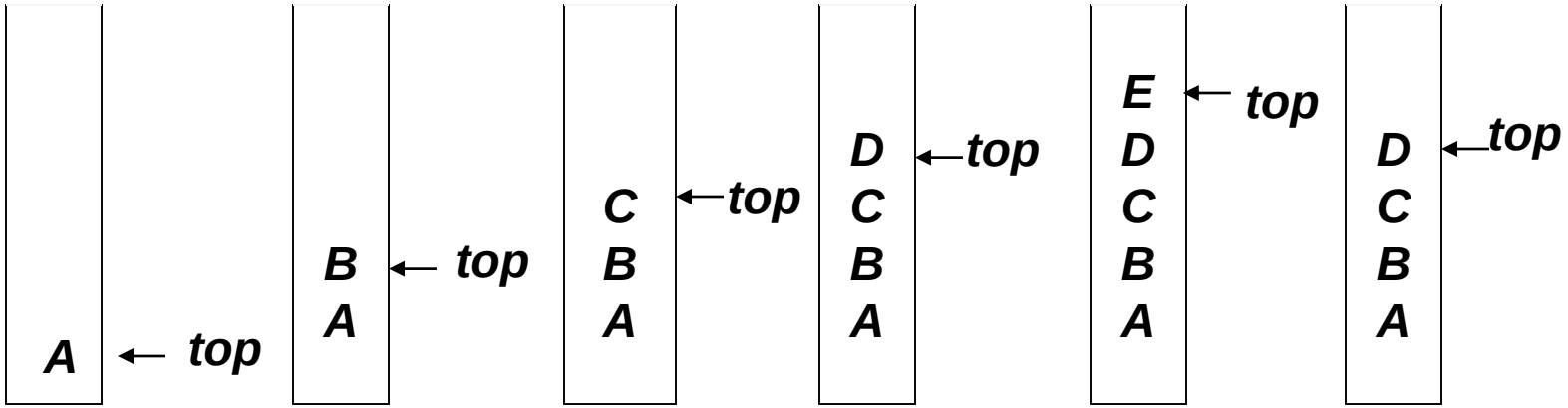
- Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack).
- Described as a

"Last In First Out" (LIFO) data structure.

OR First In Last Out data structure.

- Operations are limited:
  - push (add item to stack)
  - pop (remove top item from stack)

Last In First Out



Operation - PUSH

Procedure PUSH (S, TOP, X). This procedure inserts an element X to the top of a stack which is represented by a vector S containing N elements with a pointer TOP denoting the top element in the stack.

1. [Check for stack overflow]

If TOP >= N

then Write ("STACK OVERLOW")

Return

2. [Increment TOP]

TOP ← TOP + 1

3. [Insert element]

S [TOP] ← X

4. [Finished]

Return □

Operation - POP

Function POP (S, TOP). This function removes the top element from a stack which is represented by a vector S and returns this element. TOP is a pointer to the top element of the stack.

1. [Check for underflow on stack]

If TOP = 0

then Write ("STACK UNDERFLOW ON POP") take action in response to underflow

Exit

2. [Decrement Pointer]

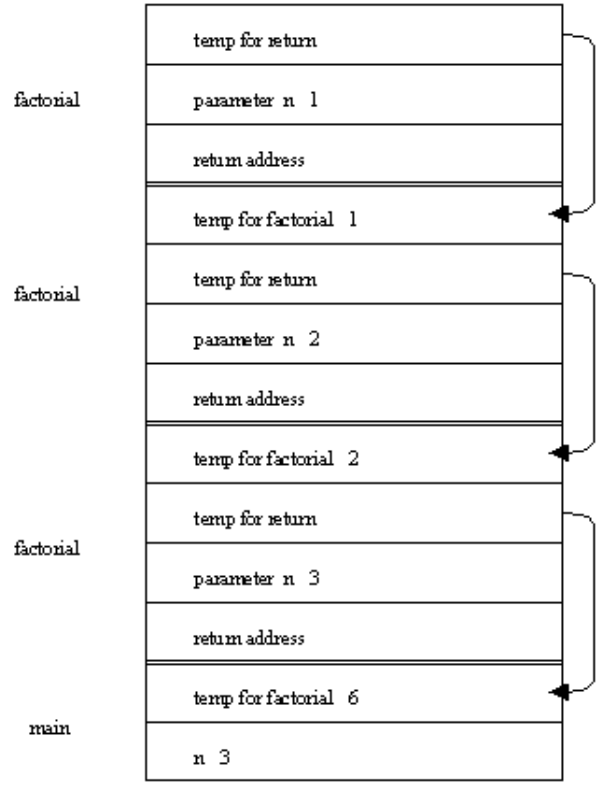
TOP ← TOP - 1

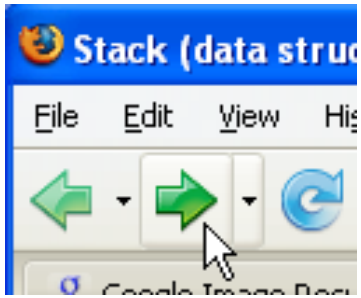
3. [Return former top element on stack]

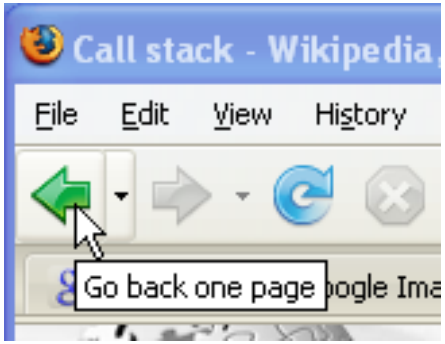
Return (S [TOP + 1]) □

Uses of Stacks

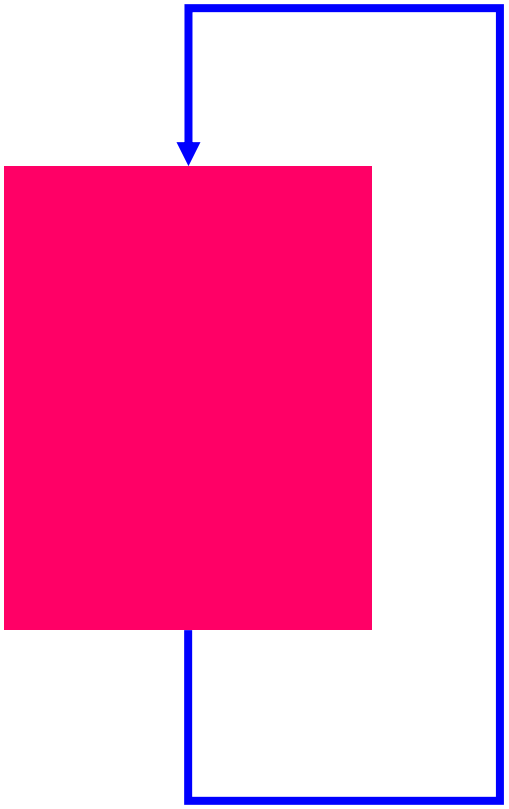
- The runtime stack used by a process (running program) to keep track of methods in progress
- Search problems
- Undo, redo, back, forward







Recursion Recursion



Recursion Recursion

- ***Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first***
- ***Recursion is a technique that solves a problem by solving a smaller problem of the same type***
- ***A procedure that is defined in terms of itself***

Recursion

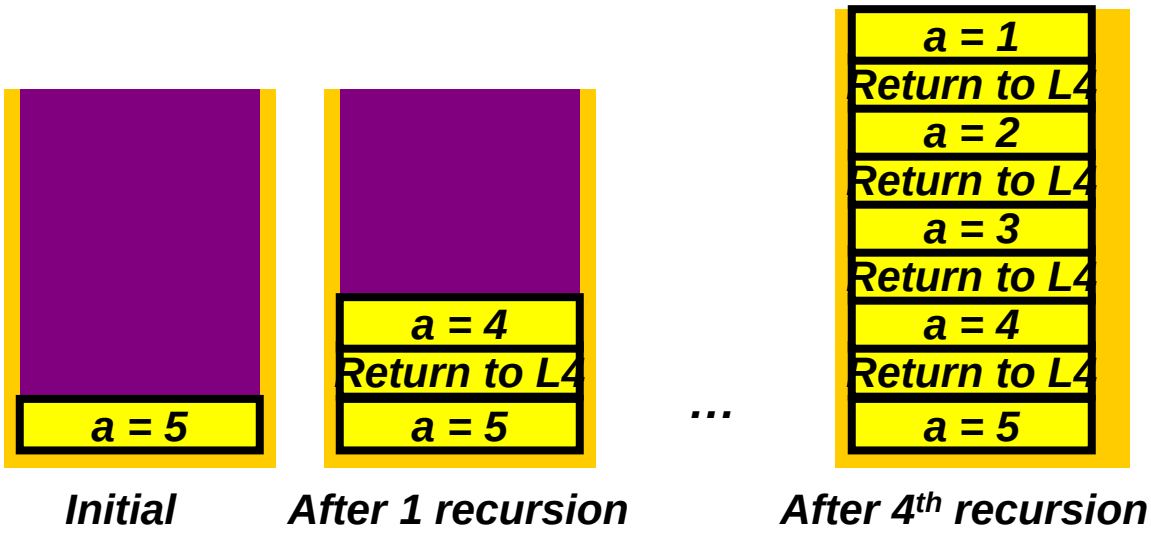
What's behind this function ?

```
public int f(int a){
        if (a==1)
          return(1);
        else
          return(a * f( a-1));
}
```

It computes f! (factorial)

Watching the Stack

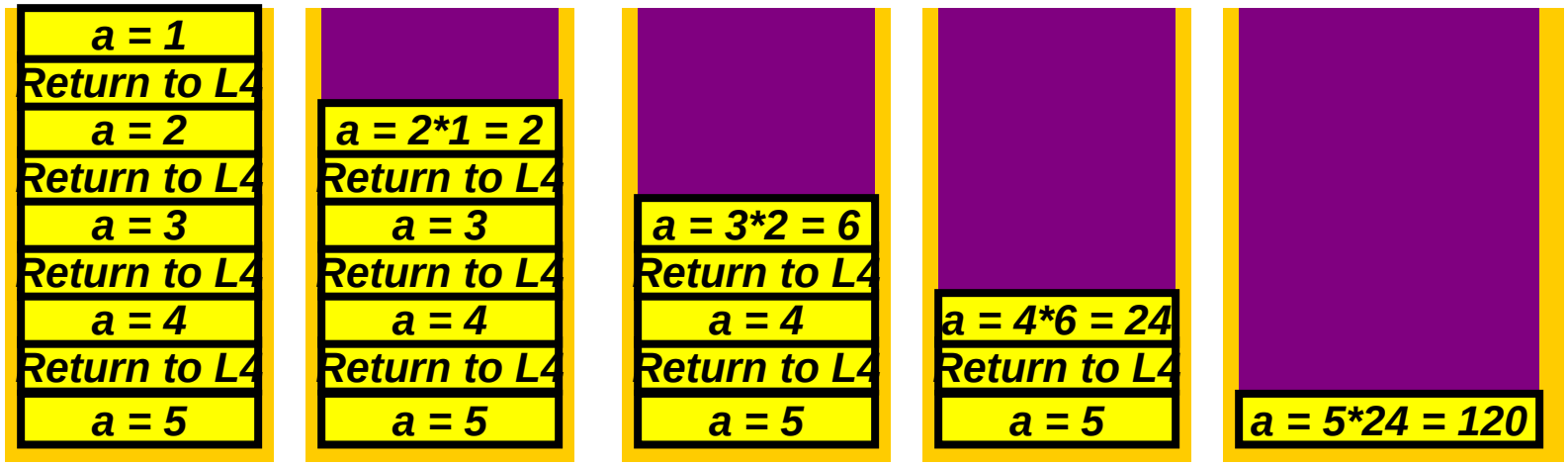
```
public int factorial(int a){
        if (a==1)
          return(1);
        else
          return(a * factorial( a-1));
}
```



Every call to the method creates a new set of local variables !

Watching the Stack

```
public int factorial(int a){
        if (a==1)
          return(1);
        else
          return(a * factorial( a-1));
}
```



Algebraic Expression

- An algebraic expression is a legal combination of operands and the operators.
- Operand is the quantity (unit of data) on which a mathematical operation is performed.
- Operand may be a variable like x, y, z or a constant like 5, 4,0,9,1 etc.
- Operator is a symbol which signifies a mathematical or logical operation between the operands. Example of familiar operators include +,-,*, /, ^
- Considering these definitions of operands and operators now we can write an example of expression as x+y*z.

Infix, Postfix and Prefix Expressions

- **INFIX:** From our schools times we have been familiar with the expressions in which operands surround the operator, e.g. x+y, 6*3 etc this way of writing the Expressions is called infix notation.

- **POSTFIX**: Postfix notation are also Known as Reverse Polish Notation (RPN). They are different from the infix and prefix notations in the sense that in the postfix notation, operator comes after the operands, e.g. xy+, xyz+* etc.

- **PREFIX**: Prefix notation also Known as Polish notation. In the prefix notation, as the name only suggests, operator comes before the operands, e.g. +xy, *+xyz etc.

Operator Priorities

- How do you figure out the operands of an operator?
  - a + b * c
  - a * b + c / d
- This is done by assigning operator priorities.
  - priority(*) = priority(/) > priority(+) = priority(-)
- When an operand lies between two operators, the operand associates with the operator that has higher priority.

Tie Breaker

- When an operand lies between two operators that have the same priority, the operand associates with the operator on the left.
  - a + b - c
  - a * b / c / d

WHY

- Why to use these weird looking PREFIX and POSTFIX notations when we have simple INFIX notation?

- To our surprise INFIX notations are not as simple as they seem specially while evaluating them. To evaluate an infix expression we need to consider Operators' Priority and Associative property
  - For example expression 3+5*4 evaluate to 32 i.e. (3+5)*4 or to 23 i.e. 3+(5*4).

- To solve this problem Precedence or Priority of the operators were defined. Operator precedence governs evaluation order. An operator with higher precedence is applied before an operator with lower precedence.

Infix Expression Is Hard To Parse

- Need operator priorities, tie breaker, and delimiters.
- This makes computer evaluation more difficult than is necessary.
- Postfix and prefix expression forms do not rely on operator priorities, a tie breaker, or delimiters.
- So it is easier to evaluate expressions that are in these forms.

Suppose that we would like to rewrite A+B*C in postfix

- Applying the rules of precedence,we obtained

A+B*C

A+(B*C) Parentheses for emphasis

A+(BC*) Convert the multiplication,Let D=BC*

A+D Convert the addition

A(D)+

ABC*+ Postfix Form

Examples of infix to prefix and post fix

Infix	PostFix	Prefix
A+B		
(A+B) * (C + D)		
A-B/(C*D^E)		

Examples of infix to prefix and post fix

Infix	PostFix	Prefix
A+B	AB+	+AB
(A+B) * (C + D)		
A-B/(C*D^E)		

Examples of infix to prefix and post fix

Infix	PostFix	Prefix
A+B	AB+	+AB
(A+B) * (C + D)	AB+CD+*	*+AB+CD
A-B/(C*D^E)	ABCDE^*/-	-A/B*C^DE

When do we need to use them… ☺

- So, what is actually done is expression is scanned from user in infix form; it is converted into prefix or postfix form and then evaluated without considering the parenthesis and priority of the operators.

Algorithm for Infix to Postfix

1) Examine the next element in the input.
2) If it is operand, output it.
3) If it is opening parenthesis, push it on stack.
4) If it is an operator, then
   i) If stack is empty, push operator on stack.
   ii) If the top of stack is opening parenthesis, push operator on stack
   iii) If it has higher priority than the top of stack, push operator on stack.
   iv) Else pop the operator from the stack and output it, repeat step 4
5) If it is a closing parenthesis, pop operators from stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
6) If there is more input go to step 1
7) If there is no more input, pop the remaining operators to output.

Suppose we want to convert 2*3/(2-1)+5*3 into Postfix form,

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

So, the Postfix Expression is 23*21-/53*+

Example

- (5 + 6) * 9 +10

will be

- 5 6 + 9 * 10 +

Evaluation a postfix expression

- Each operator in a postfix string refers to the previous two operands in the string.
- Suppose that each time we read an operand we **push** it into a stack. When we reach an operator, its operands will then be top two elements on the stack
- We can then **pop** these two elements, perform the indicated operation on them, and **push** the result on the stack.
- So that it will be available for use as an operand of the next operator.

Postfix Examples

Infix	Postfix	Evaluation
2 - 3 * 4 + 5	2 3 4 * - 5 +	-5
(2 - 3) * (4 + 5)	2 3 - 4 5 + *	-9
2- (3 * 4 +5)	2 3 4 * 5 + -	-15

Why ? No brackets necessary !

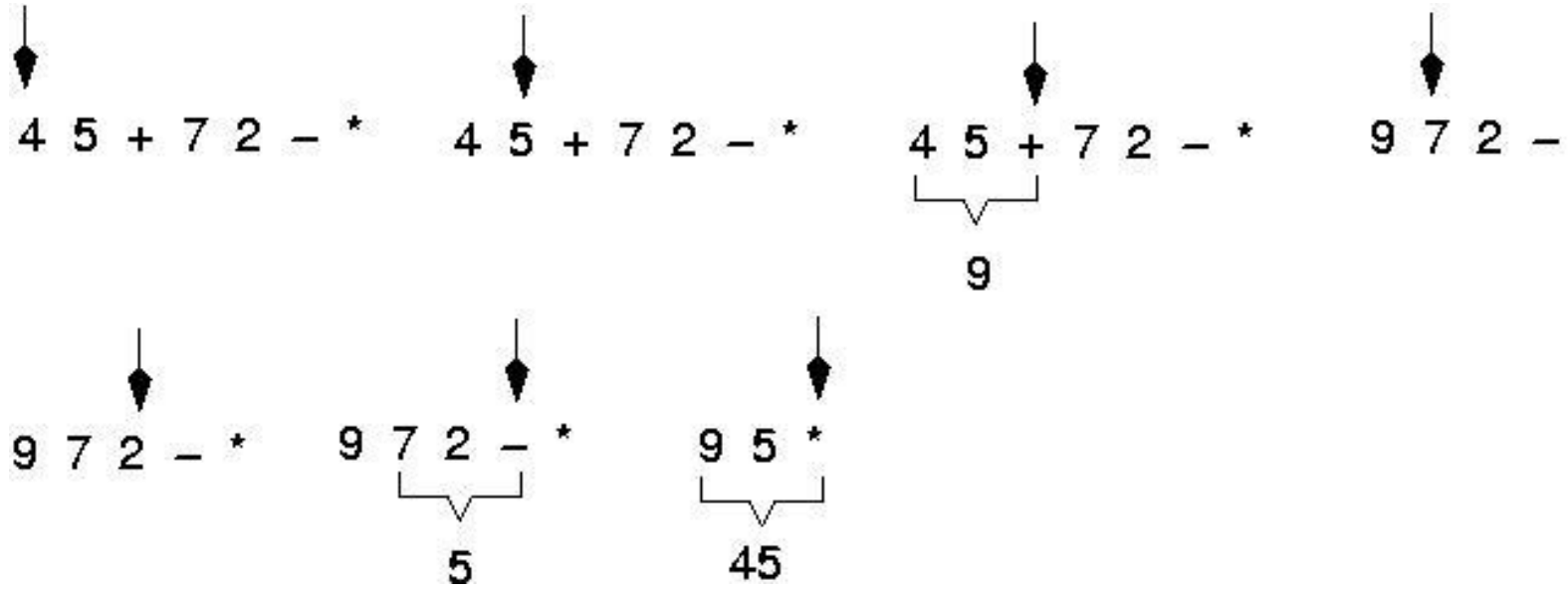
Evaluating Postfix Notation

- Use a stack to evaluate an expression in postfix notation.
- The postfix expression to be evaluated is scanned from left to right.
- Variables or constants are pushed onto the stack.
- When an operator is encountered, the indicated action is performed using the top elements of the stack, and the result replaces the operands on the stack.

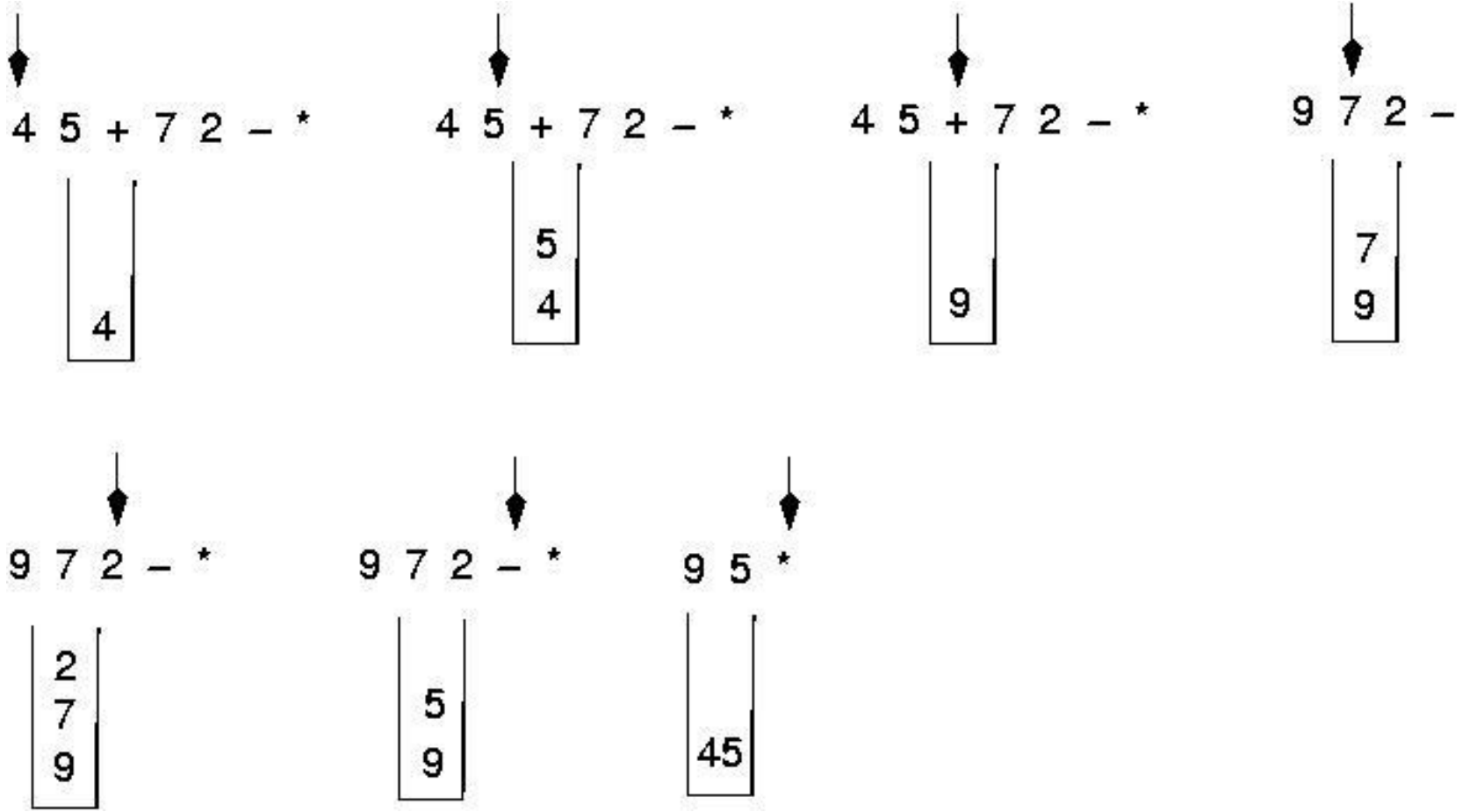
Evaluating a postfix expression

- Initialise an empty stack
- While token remain in the input stream
  - Read next token
  - If token is a number, push it into the stack
  - Else, if token is an operator, pop top two tokens off the stack,apply the operator, and push the answer back into the stack
- Pop the answer off the stack.

Example: postfix expressions (cont.)



Postfix expressions: Algorithm using stacks (cont.)



Algorithm for evaluating a postfix expression (Cond.)

```
WHILE more input items exist
{
    If symb is an operand
        then push (opndstk,symb)
    else   //symbol is an operator
    {
        Opnd2=pop(opndstk);
        Opnd1=pop(opndnstk);
        Value = opnd1 operator opnd2
        Push(opndstk,value);
    }                   //End of else
} // end while
Result = pop (opndstk);
```

Question : Evaluate the following expression in postfix : 623+-382/+*2^3+

Final answer is

- 49
- 51
- 52
- 7
- None of these

Evaluate- 623+-382/+*2^3+

Symbol	opnd1	opnd2	value	opndstk
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3

Evaluate- 623+-382/+*2^3+

Symbol	opnd1	opnd2	value	opndstk
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
^	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Stack Applications

- Recursion
- Conversion (Infix to Postfix, Infix to Prefix)
- Evaluation (Postfix, Prefix)
- Parsing
  - Matching Parenthesis
  - Match in HTML tags < >
  - Matching if else
- Browser
- Editors (undo, redo)
- Other data structures use stack for implementation