Stack & Arithmetic Expression

Last-In-First-Out (LIFO)

Dr. Nadia M. Mohammed

Applic^{n} 2: Arithmetic Expression (1/7)

- Terms
  - Expression: a = b + c * d
  - Operands: a, b, c, d
  - Operators: =, +, –, *, /, %
- Precedence rules: Operators have priorities over one another as indicated in a table (which can be found in most books & our first few lectures)
  - Example: * and / have higher precedence over + and –.
  - For operators at the same precedence (such as * and /), we process them from left to right

Priority of operation

Priority	Operation
1	^ , unary (-) , unary (+), Not
2	* , / , AND, DIV , MOD
3	+, - , OR
4	= , < ,> , <> , <= , >=

Applicⁿ 2: Arithmetic Expression (2/7)

Infix : operand1 operator operand2

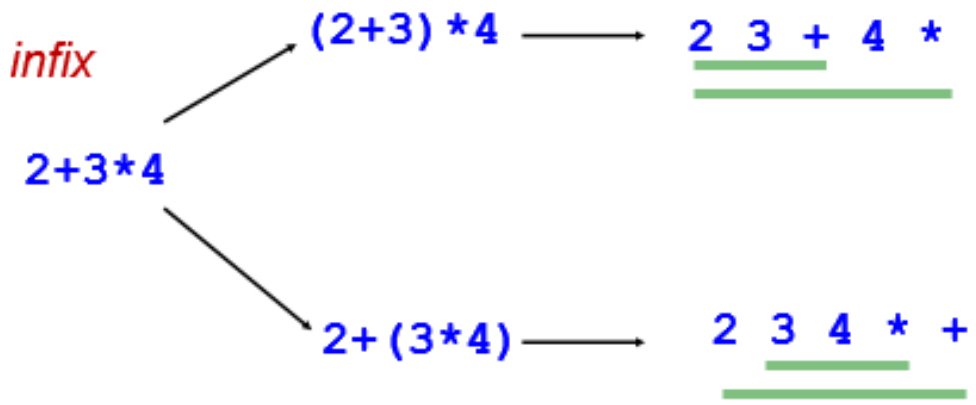
Prefix : operator operand1 operand2

Postfix : operand1 operand2 operator

Ambiguous, need () or precedence rules

Unique interpretation

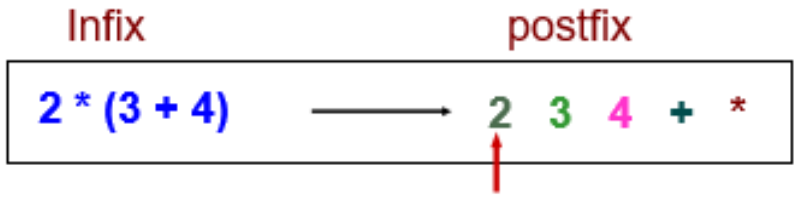
postfix



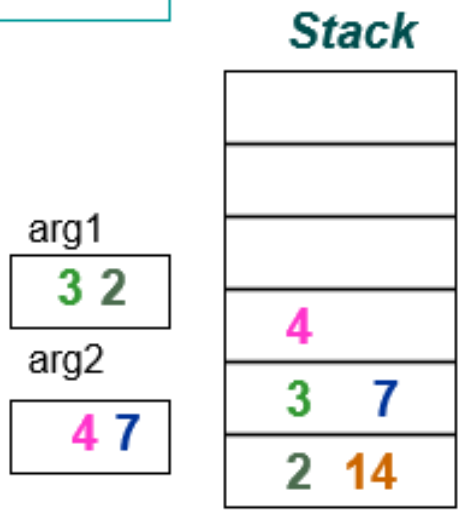
Applicⁿ 2: Arithmetic Expression (3/7)

Algorithm: Calculating Postfix expression with stack

Create an empty stack
for each item of the expression,
  if it is an operand,
    push it on the stack
  if it is an operator,
    pop arguments from stack;
    perform the operation;
    push the result onto the stack



2 s.push(2)
3 s.push(3)
4 s.push(4)
\+ arg2 = s.pop ()
arg1 = s.pop ()
s.push (arg1 + arg2)
* arg2 = s.pop ()
arg1 = s.pop ()
s.push (arg1 * arg2)



Problems

Find the **result of arithmetic exp** using concept of **stack**:

1. 7+6*3/2 **convert to**

2. (3+7) *2-6 **convert to**

3. ab*cde^/+ where a=5, b=6, c=8, d=2, e=2

Applicⁿ 2: Arithmetic Expression (4/7)

Brief steps for Infix to Postfix Conversion

1. Scan infix expression from left to right
2. If an operand is found, add it to the postfix expression.
3. If a "(" is found, push it onto the stack.
4. If a ")" is found
   a) repeatedly pop the stack and add the popped operator to the postfix expression until a "(" is found.
   b) remove the "(".
5. If an operator is found
   a) repeatedly pop the operator from stack which has higher or equal precedence than/to the operator found, and add the popped operator to the postfix expression.
   b) add the new operator to stack
6. If no more token in the infix expression, repeatedly pop the operator from stack and add it to the postfix expression.

Applicⁿ 2: Arithmetic Expression (6/7)

Algorithm: Converting Infix to an equivalent Postfix

ch	Stack (bottom to top)	postfixExp
a		a
–	–	a
(	– (	a
b	– (	a b
+	– (+	a b
c	– (+	a b c
*	– (+ *	a b c
d	– (+ *	a b c d
)	– (+	a b c d *
	– (	a b c d * +
	–	a b c d * +
/	– /	a b c d * +
e	– /	a b c d * + e
		a b c d * + e / –

Example: a – (b + c * d) / e

Move operators from stack to postfixExp until '('

Copy remaining operators from stack to postfixExp

Applic^{n} 2: Arithmetic Expression (7/7)

- How to code the above algorithm in **Python**
- How to do conversion of infix to prefix?

Call Function

- Begin {this is the main program}
- 100 Call A
- 102 -
- -
- 200 Call B -
- 202 -
- 300 Call
- 302 -
- -
- end

Problems

A) **Write** an **algorithm** to read string end with (*) and print it in reverse order

B) **Convert** the following **infix exp.** To **Postfix exp.** Using the concept of Stack

1- a-b*(c+d)/(e-f)^g*h

2- y*m+(a^3/b-n)-d

3- m= x/6+(a-2*(b/3)^5+f)^2

4- (a>b) and ((e-c>a)or(g<f))

5- B-a+c and n^x-(p/M or f^2)

6.1.1 The Stack Abstract Data Type

Stacks are the simplest of all data structures, yet they are also among the most important. They are used in a host of different applications, and as a tool for many more sophisticated data structures and algorithms. Formally, a stack is an abstract data type (ADT) such that an instance S supports the following two methods:

S.push(e): Add element e to the top of stack S.

S.pop( ): Remove and return the top element from the stack S; an error occurs if the stack is empty.

Additionally, let us define the following accessor methods for convenience:

S.top( ): Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.

S.is_empty( ): Return True if stack S does not contain any elements.

len(S): Return the number of elements in stack S; in Python, we implement this with the special method __len__.

By convention, we assume that a newly created stack is empty, and that there is no a priori bound on the capacity of the stack. Elements added to the stack can have arbitrary type.

Example 6.3: *The following table shows a series of stack operations and their effects on an initially empty stack S of integers.*

Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop( )	3	[5]
S.is_empty( )	False	[5]
S.pop( )	5	[ ]
S.is_empty( )	True	[ ]
S.pop( )	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top( )	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop( )	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop( )	8	[7, 9, 6]

H.W

The following table shows a series of stack operations . Find the Return Value and Stack Contents:

Operations	Return Value	Stack Contents
S.Push(1) S.Push(2) S.Push(3) len(S) S.is_empty() S.Push(4) S.Push(5) len(S) S.top() S.Pop() len(S) S.Pop() S.Pop() S.top() len(S) S.Push(6)		