

# Inför Lab3 (Räknare)

Joachim von Hacht

# Infix till Postfix

## Alla operander behåller sin ordning!

1. Flytta operatorer, utifrån prioritet, efter operanderna (om parenteser, respektera dessa, ta därefter bort dem).
2. Om samma prioritet, flytta utifrån associativitet (v->h eller h->v).

**Exempel**

$1 + 2 * 3$	(infix)
$1 + \boxed{2\ 3}\ *$	(1.)
$\boxed{1\ 2\ 3}\ *\ +$	(1.) (postfix)

Exemple1	$(1 + 2) * 3 \wedge 4 \wedge 5$	(infix)
	1 2 + * 3 $\wedge$ 4 $\wedge$ 5	(1.)
	1 2 + * 3 $\wedge$ 4 5 $\wedge$	(2. $\wedge$ evaluaras h->v)
	1 2 + * 3 4 5 $\wedge$ $\wedge$	(1.)
	1 2 + 3 4 5 $\wedge$ $\wedge$ *	(1.) (postfix)

# Exempel: Infix till Postfix

Skriv i postfix form

1.  $1 / 2 + 3$

2.  $6 - 2 * ( 2 - 1 )$

3.  $3 / 2 / 1 * ( 3 - 2 ) + 4$

# Evakuering av Postfix

Tag en operand eller operator i taget (v->h) från uttrycket

1. Om operand, push:a på stack.
2. Om (binär) operator, pop:a två element från stack, beräkna, push:a resultat på stack ...
3. ... tills inget kvar. Resultatet finns på stackens top. Om exakt ett värde på top så OK. Annars fel, får många/få operatorer eller operander.

Uttryck	Stack (top är index 0)
5 4 + 3 2 1 ^ ^ *	[ ]
4 + 3 2 1 ^ ^ *	[ 5 ]
+ 3 2 1 ^ ^ *	[ 4, 5 ]
3 2 1 ^ ^ *	+ [ 4, 5 ] → [ 9 ]
2 1 ^ ^ *	[ 3, 9 ]
1 ^ ^ *	[ 2, 3, 9 ]
^ ^ *	[ 1, 2, 3, 9 ]
^ *	^ [ 1, 2, 3, 9 ] → [ 2, 3, 9 ]
*	^ [ 2, 3, 9 ] → [ 9, 9 ]
	* [ 9, 9 ] → [ 81 ]

OBS! v resp h operand

# Exempel: Evaluera Postfix

Evaluera uttrycket steg för steg

1. 6 5 \* 4 + [ ] (tom stack)

2. 3 5 \* 4 2 + - [ ]

# Shunting-yard Algorithm (1)

Infix till postfix algoritmen

Infix och Postfix båda  
List<String>, Stack  
Deque<String>

Infix	Stack	Postfix
1 * 2 + 3	[]	
* 2 + 3	[]	1
2 + 3	[ * ]	1
+ 3	[ * ]	1 2
3	[ + ]	1 2 *
	[ + ]	1 2 * 3
	[]	1 2 * 3 +

// Prio. + < prio. \* , pop, push  
// Pop all, append stack

1 + 2 * 3	[]	
+ 2 * 3	[]	1
2 * 3	[ + ]	1
* 3	[ + ]	1 2
3	[ *, + ]	1 2
	[ *, + ]	1 2 3
	[]	1 2 3 * +

// Prio. \* > prio. +, push  
// Pop all, append

# Shunting-yard Algorithm (2)

Infix	Stack	Postfix	
3 - 2 + 1	[]		
- 2 + 1	[]	3	
2 + 1	[ - ]	3	
+ 1	[ - ]	3 2	
1	[ + ]	3 2 -	// Same prio. top assoc. left, pop, push
	[ + ]	3 2 - 1	
	[]	3 2 - 1 +	// Pop all, append stack
1 ^ 2 ^ 3	[]		
^ 2 ^ 3	[]	1	
2 ^ 3	[ ^ ]	1	
^ 3	[ ^ ]	1 2	
3	[ ^, ^ ]	1 2	// Same prio. top assoc. right, push
	[ ^, ^ ]	1 2 3	
	[]	1 2 3 ^ ^	// Pop all, append

# Shunting-yard Algorithm (3)

Infix	Stack	Postfix
(1 + 2) * 3 ^ 4 ^ 5	[ ]	
1 + 2) * 3 ^ 4 ^ 5	[ ( ]	// Paren. start, remember!
+ 2) * 3 ^ 4 ^ 5	[ ( ]	1
2) * 3 ^ 4 ^ 5	[ +, ( ]	1
) * 3 ^ 4 ^ 5	[ +, ( ]	1 2
* 3 ^ 4 ^ 5	[ ]	1 2 + // End. paren, pop, (skip "(")
3 ^ 4 ^ 5	[ * ]	1 2 +
^ 4 ^ 5	[ * ]	1 2 + 3
4 ^ 5	[ ^, * ]	1 2 + 3 // Prio.^ > prio. *, push
^ 5	[ ^, * ]	1 2 + 3 4
5	[ ^, ^, * ]	1 2 + 3 4 // Assoc. right, push
	[ ^, ^, * ]	1 2 + 3 4 5
	[ ]	1 2 + 3 4 5 ^ ^ * // Pop all, append



# Exempel: Shunting yard

Infix	Stack	Postfix
3 * (1 + 2 * 3) ^ 2	[]	
* (1 + 2 * 3) ^ 2	[]	3
(1 + 2 * 3) ^ 2	[*]	3
1 + 2 * 3) ^ 2	[(, *]	3
+ 2 * 3) ^ 2	[(, *]	3 1
2 * 3) ^ 2	[+, (, *]	3 1
* 3) ^ 2	[+, (, *]	3 1 2
3) ^ 2	[*, +, (, *]	3 1 2
) ^ 2	[*, +, (, *]	3 1 2 3 // Pop all until (
^ 2	[*]	3 1 2 3 * +
2	[^, *]	3 1 2 3 * +
	[^, *]	3 1 2 3 * + 2
	[]	3 1 2 3 * + 2 ^ *

# Skiss Funktionell Nedbrytning

