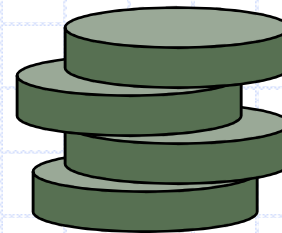
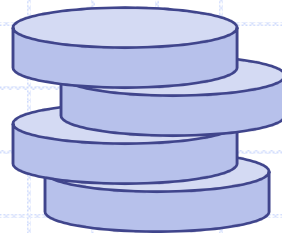
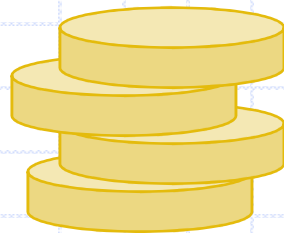


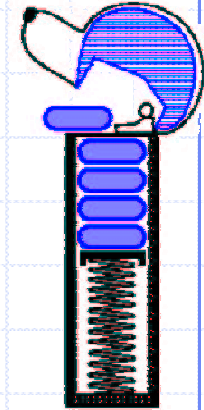
# Στοιίβες



# Αφηρημένοι Τύποι Δεδομένων(ΑΤΔ)

- Ένας αφηρημένος τύπος δεδομένων (ΑΤΔ) είναι αφαίρεση μιας δομής δεδομένων
- Ένας ΑΤΔ ορίζει:
  - Τα δεδομένα που θα αποθηκευθούν
  - Τις πράξεις στα δεδομένα
  - Συνθήκες λάθους σχετικές με τις πράξεις
- Παράδειγμα: ΑΤΔ μοντελοποίησης ενός απλού συστήματος αγοροπωλησίας μετοχών
  - Τα δεδομένα είναι εντολές πωλήσεων/αγορών
  - Οι υποστηριζόμενες πράξεις είναι
    - ◆ παραγγελία **buy**(stock, shares, price)
    - ◆ παραγγελία **sell**(stock, shares, price)
    - ◆ void **cancel**(order)
  - Συνθήκες λάθους:
    - ◆ Buy/sell μια μετοχή που δεν υπάρχει
    - ◆ Ακύρωση μη υπάρχουσας εντολής

# Ο ΑΤΔ Στοίβα



- Ο ΑΤΔ **Στοίβα** αποθηκεύει οποιαδήποτε αντικείμενα
- Οι εισαγωγές και οι διαγραφές ακολουθούν το σχήμα εξάγεται το πιο πρόσφατα εισαχθέν (LIFO last-in first-out)
- Σαν το μηχάνημα που χρησιμοποιούν τα ταξί για τα κέρματα
- Βασικές πράξεις στοίβας:
  - **push**(object): εισάγει ένα στοιχείο
  - object **pop**(): αφαιρεί και επιστρέφει το πιο πρόσφατα εισαχθέν στοιχείο
- Βοηθητικές πράξεις στοίβας:
  - object **top**(): επιστρέφει το πιο πρόσφατα εισαχθέν χωρίς να το αφαιρέσει
  - integer **size**(): επιστρέφει το πλήθος των αποθηκευμένων στοιχείων
  - boolean **isEmpty**(): δείχνει αν η στοίβα είναι κενή

Έστω ότι περνάμε από μια στοίβα τη λέξη ATNKEP (με \* η απώθηση)

	Έξοδος	Στοίβα
	A	A
	T	AT
	N	ATN
	K	ATNK
*	K	ATN
E	K	ATNE
*	KE	ATN
*	KEN	AT
*	KENT	A
P	KENT	AP
*	KENTP	A
*	KENTPA	

# Διεπαφή Στοίβας στη Java

- Η Java διεπαφή που αντιστοιχεί στον ΑΤΔ στοίβα
- Απαιτεί τον ορισμό μιας κλάσης `EmptyStackException`
- Διαφέρει από την ενσωματωμένη κλάση `java.util.Stack`

```
public interface Stack<E> {  
    public int size();  
    public boolean isEmpty();  
    public E top()  
        throws EmptyStackException;  
    public void push(E element);  
    public E pop()  
        throws EmptyStackException;  
}
```

# Εξαιρέσεις

- Η προσπάθεια εκτέλεσης μιας πράξης ΑΤΔ μπορεί μερικές φορές να προκαλέσει συνθήκη λάθους, που ονομάζεται εξαίρεση
- Λέμε ότι οι εξαιρέσεις πετάγονται ("thrown") από μια πράξη που δεν μπορεί να εκτελεσθεί
- Στις πράξεις του ΑΤΔ στοίβας οι pop και top δεν μπορούν να εκτελεσθούν αν η στοίβα είναι κενή
- Η προσπάθεια εκτέλεσης της pop ή της top σε μια κενή στοίβα πετάει μια `EmptyStackException`



# Στοιβες- Εφαρμογές

- Άμεσες εφαρμογές
  - Ιστορικό των σελίδων που επισκέφθηκε ένας browser
  - Ακολουθία ακυρώσεων πράξεων ενός κειμενογράφου
  - Ακολουθία κλήσεων μεθόδων μιας Java εικονικής μηχανής
- Έμμεσες εφαρμογές
  - Βοηθητική δομή δεδομένων για αλγόριθμους
  - Στοιχείο άλλων δομών δεδομένων

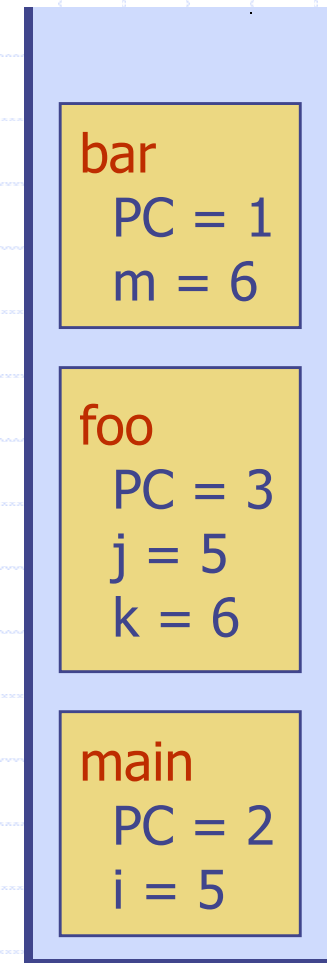
## Στοίβα Μεθόδων στην JVM (Java Virtual Machine)

- Η The Java Virtual Machine (JVM) κρατάει στη σειρά σε στοίβα τις ενεργές μεθόδους
- Όταν καλείται μια μέθοδος, η JVM τοποθετεί στη στοίβα ένα πλαίσιο που περιέχει
  - Τις τοπικές μεταβλητές και την επιστρεφόμενη τιμή
  - Μετρητή του προγράμματος, που κρατάει την εντολή που εκτελείται
- Όταν τελειώσει μια μέθοδος, βγαίνει το πλαίσιο της από τη στοίβα και ο έλεγχος περνάει στη μέθοδο στην κορυφή της στοίβας
- Υποστηρίζει **αναδρομή**

```
main() {  
    int i = 5;  
    foo(i);  
}
```

```
foo(int j) {  
    int k;  
    k = j+1;  
    bar(k);  
}
```

```
bar(int m) {  
    ...  
}
```



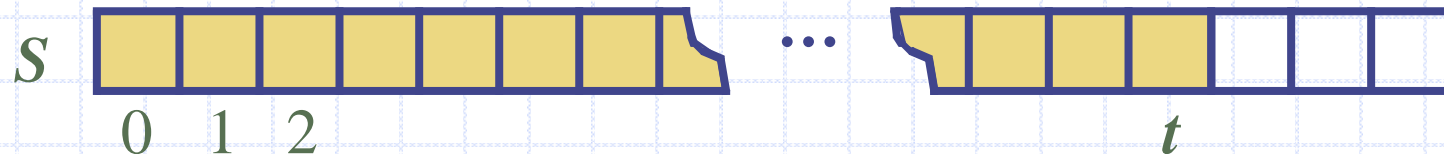


# Στοιβα που βασίζεται σε πίνακα

- Ένας απλός τρόπος υλοποίησης του ΑΤΔ της στοίβας χρησιμοποιεί ένα πίνακα
- Προσθέτουμε στοιχεία από αριστερά προς τα δεξιά
- Μια μεταβλητή δείχνει στο στοιχείο στην κορυφή

```
Algorithm size()  
return  $t + 1$ 
```

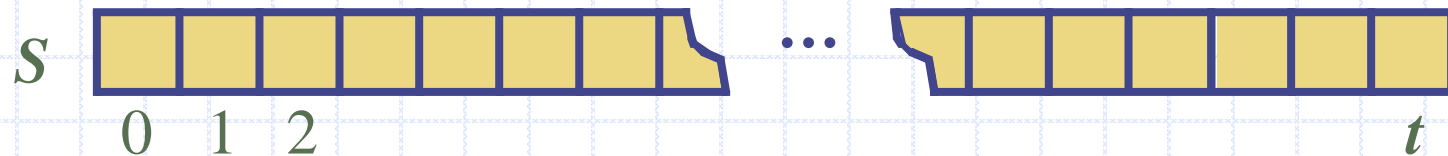
```
Algorithm pop()  
if isEmpty() then  
    throw EmptyStackException  
else  
     $t \leftarrow t - 1$   
    return  $S[t + 1]$ 
```



## Στοίβα που βασίζεται σε πίνακα (συν.)

- Ο πίνακας που αποθηκεύει τη στοίβα μπορεί να γεμίσει
- Τότε μια πράξη ώθησης θα πετάξει μια *FullStackException*
  - Περιορισμός της υλοποίησης με πίνακα
  - Ουσιαστικά διαφέρει από τον ΑΤΔ στοίβας

```
Algorithm push(o)  
  if  $t = S.length - 1$  then  
    throw FullStackException  
  else  
     $t \leftarrow t + 1$   
     $S[t] \leftarrow o$ 
```



# Απόδοση και Περιορισμοί

## □ Απόδοση

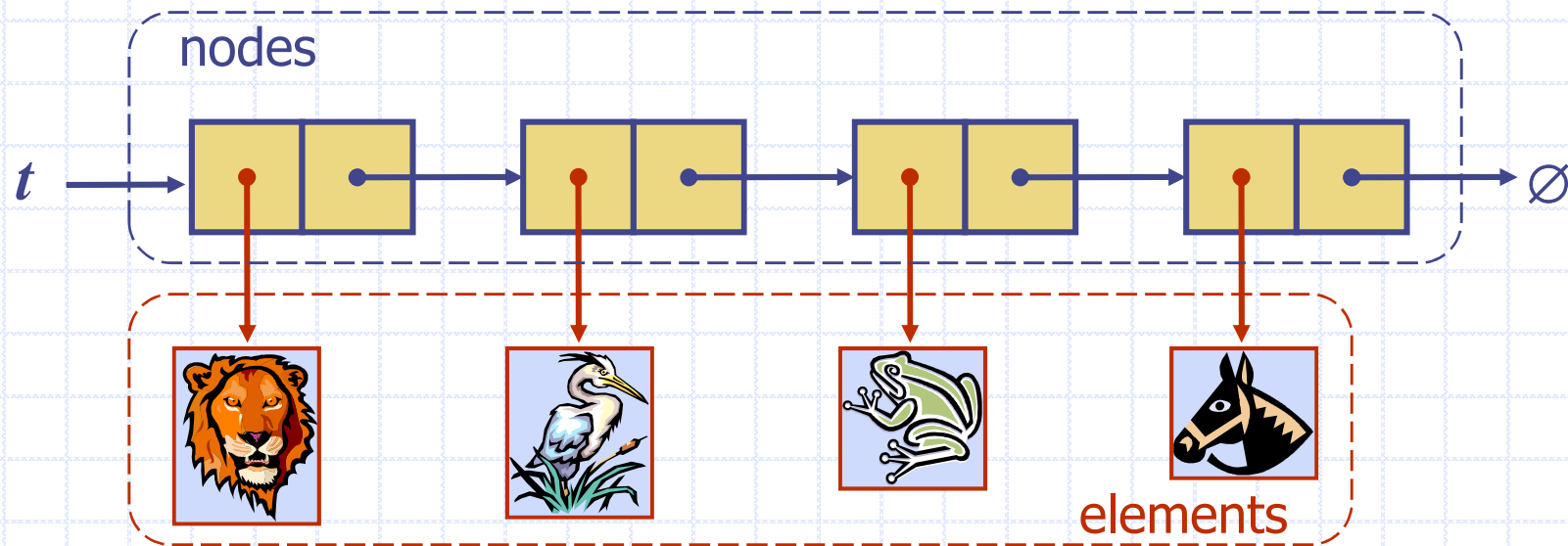
- Έστω  $n$  το πλήθος των στοιχείων της στοίβας
- Ο χώρος που χρησιμοποιείται είναι  $O(n)$
- Κάθε πράξη τρέχει σε χρόνο  $O(1)$

## □ Περιορισμοί

- Πρέπει να ορίζεται από πριν το μέγιστο μέγεθος της στοίβας και δεν μπορεί να αλλάξει
- Προσπάθεια ώθησης ενός νέου στοιχείου σε μια γεμάτη στοίβα προκαλεί μια εξαίρεση υλοποίησης

# Στοίβα σαν συνδεδεμένη λίστα

- Μπορούμε να υλοποιήσουμε μια στοίβα με μια απλά συνδεδεμένη λίστα
- Το πιο πάνω στοιχείο αποθηκεύεται σαν πρώτος κόμβος της λίστας
- Ο χώρος που απαιτείται είναι  $O(n)$  και κάθε πράξη του ΑΤΔ στοίβα θέλει χρόνο  $O(1)$



# Στοιβα που βασίζεται σε πίνακα στη Java

```
public class ArrayStack<E>
    implements Stack<E> {

    // περιέχει τα στοιχεία της στοίβας
    private E S[ ];

    // δείκτης στο στοιχείο στην
    κορυφή
    private int top = -1;

    // constructor
    public ArrayStack(int capacity) {
        S = (E[]) new Object[capacity];
    }
}
```

```
public E pop()
    throws EmptyStackException {
    if isEmpty()
        throw new EmptyStackException
            ("Empty stack: cannot pop");
    E temp = S[top];
    // facilitate garbage collection:
    S[top] = null;
    top = top - 1;
    return temp;
}

... (άλλες μέθοδοι διεπαφής στοίβας)
```

# Παράδειγμα Χρήσης στη Java

```
public class Tester {  
    // ... other methods  
    public intReverse(Integer a[]) {  
        Stack<Integer> s;  
        s = new ArrayStack<Integer>();  
        ... (code to reverse array a) ...  
    }  
}
```

```
public floatReverse(Float f[]) {  
    Stack<Float> s;  
    s = new ArrayStack<Float>();  
    ... (code to reverse array f) ...  
}
```



# Αντιστοιχία Παρενθέσεων

- Κάθε "(", "{", ή "[" πρέπει να κλείνει με ένα αντίστοιχο ")", "}", ή "]"
  - σωστό: ( )(( )){([ ( ))}
  - σωστό: ( )(( )){([ ( ( ) ) )}
  - λάθος: )(( )){([ ( ) )}
  - λάθος: ({ [ ] )}
  - λάθος: (

# Αλγόριθμος αντιστοίχισης Παρενθέσεων

**Algorithm** ParenMatch( $X, n$ ):

**Input:** Ένας πίνακας  $X$  με  $n$  στοιχεία, που το καθένα είναι ή σύμβολο ομαδοποίησης, μια μεταβλητή, ένας αριθμητικός τελεστής, ή ένας αριθμός

**Output:** **true** αν και μόνο αν όλα τα σύμβολα ομαδοποίησης του  $X$  ταιριάζουν  
Έστω  $S$  μια μη κενή στοίβα

**for**  $i=0$  to  $n-1$  **do**

**if**  $X[i]$  is an opening grouping symbol **then**

$S.push(X[i])$

**else if**  $X[i]$  is a closing grouping symbol **then**

**if**  $S.isEmpty()$  **then**

**return false** {δεν έχει αντίστοιχο}

**if**  $S.pop()$  does not match the type of  $X[i]$  **then**

**return false** {λάθος τύπος}

**if**  $S.isEmpty()$  **then**

**return true** {όλα τα σύμβολα ταιρίαζαν}

**else return false** {κάποια σύμβολα δεν είχαν τάιρι}

# Αντιστοίχιση HTML ετικετών

- ◆ Για μια πλήρως σωστή HTML, κάθε **<name>** σπρέπει να ζευγαρώνει με ένα αντίστοιχο **</name>**

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine. The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```

## The Little Boat

The storm tossed the little boat like a cheap sneaker in an old washing machine. The three drunken fishermen were used to such treatment, of course, but not the tree salesman, who even as a stowaway now felt that he had overpaid for the voyage.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

# Αλγόριθμος αντιστοίχισης ετικετών (σε Java)

```
import java.io.*;
import java.util.Scanner;
import net.datastructures.*;
/** Simplified test of matching tags in an HTML document. */
public class HTML {
    /** Strip the first and last characters off a <tag> string. */
    public static String stripEnds(String t) {
        if (t.length() <= 2) return null;          // this is a degenerate tag
        return t.substring(1,t.length()-1);
    }
    /** Test if a stripped tag string is empty or a true opening tag. */
    public static boolean isOpeningTag(String tag) {
        return (tag.length() == 0) || (tag.charAt(0) != '/');
    }
}
```

# Αλγόριθμος αντιστοίχισης ετικετών(συν.)

```
/** Test if stripped tag1 matches closing tag2 (first character is '/'). */
public static boolean areMatchingTags(String tag1, String tag2) {
    return tag1.equals(tag2.substring(1)); // test against name after '/'
}

/** Test if every opening tag has a matching closing tag. */
public static boolean isHTMLMatched(String[] tag) {
    Stack<String> S = new NodeStack<String>(); // Stack for matching tags
    for (int i = 0; (i < tag.length) && (tag[i] != null); i++) {
        if (isOpenTag(tag[i]))
            S.push(tag[i]); // opening tag; push it on the stack
        else {
            if (S.isEmpty())
                return false; // nothing to match
            if (!areMatchingTags(S.pop(), tag[i]))
                return false; // wrong match
        }
    }
    if (S.isEmpty()) return true; // we matched everything
    return false; // we have some tags that never were matched
}
```

# Αλγόριθμος αντιστοίχισης ετικετών(συν.)

```
public final static int CAPACITY = 1000; // Tag array size
/* Parse an HTML document into an array of html tags */
public static String[] parseHTML(Scanner s) {
    String[] tag = new String[CAPACITY]; // our tag array (initially all null)
    int count = 0;                       // tag counter
    String token;                         // token returned by the scanner s
    while (s.hasNextLine()) {
        while ((token = s.findInLine("<[^>]*>")) != null) // find the next tag
            tag[count++] = stripEnds(token); // strip the ends off this tag
        s.nextLine(); // go to the next line
    }
    return tag; // our array of (stripped) tags
}
public static void main(String[] args) throws IOException { // tester
    if (isHTMLMatched(parseHTML(new Scanner(System.in))))
        System.out.println("The input file is a matched HTML document.");
    else
        System.out.println("The input file is not a matched HTML document.");
}
```



Με χρήση στοίβας γίνεται πολύ εύκολος ο υπολογισμός μιας παράστασης σε μεταθεματική μορφή

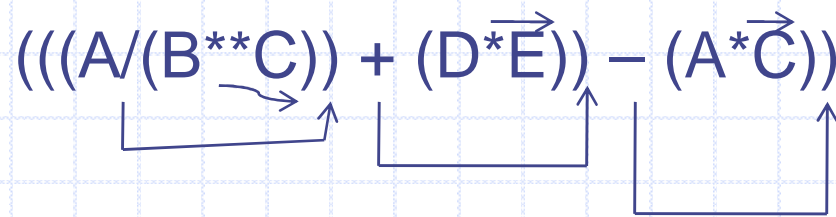
$(5 * (((9 + 8) * (4 * 6)) + 7))$

5 9 8 + 4 6 \* \* 7 + \*

5	5
9	59
8	5 9 8
+	5 17
4	5 17 4
6	5 17 4 6
*	5 17 24
*	5 408
7	5 408 7
+	5 415
*	2075

## Μετατροπή ενθεματικής σε μεταθεματική μορφή

Υπάρχει ένας απλός τρόπος όταν είναι σε πλήρη παρενθετική μορφή. Για παράδειγμα η  $A/B^{**}C + D * E - A * C$  γράφεται σε πλήρη παρενθετική μορφή  $((A/(B^{**}C)) + (D * E)) - (A * C)$


$$(((A/(B^{**}C)) + (D^{**}E)) - (A^{**}C))$$

Μεταφέρουμε κάθε τελεστή στην αντίστοιχη δεξιά παρένθεσή του και διαγράφουμε τις παρενθέσεις

Για την μετατροπή μιας ενθεματικής σε μεταθεματική χρησιμοποιούμε μια στοίβα από την οποία περνάμε τους τελεστές. Αγνοούμε τις αριστερές παρενθέσεις. Αν συναντήσουμε δεξιά παρένθεση στέλνουμε στην έξοδο τον τελεστή που βρίσκεται στην κορυφή της στοίβας

Ας δούμε το παράδειγμα της

$$(5 * (((9 + 8) * (4 * 6)) + 7))$$



(  
5      5  
\*  
(  
(  
(  
9      9  
+  
8      8  
)      +  
\*  
(  
4      4  
\*  
6      6  
)      \*  
)      \*  
+  
7      7  
)      +  
)      \*

\*  
\*  
\*  
\*  
\*  
\* +  
\* +  
\*  
\*\*  
\* \*  
\* \* \*  
\* \* \*  
\* \*  
\*  
\* +  
\* +  
\*

$(5 * (((9 + 8) * (4 * 6)) + 7))$

Αυτός ο αλγόριθμος μετατροπής θέλει δύο περάσματα καθώς χρειάζεται να φέρουμε την παράσταση πρώτα σε πλήρως παρενθετική. Όμως στις γλώσσες προγραμματισμού με κάποιες συμβάσεις που κάνουμε (προτεραιότητες τελεστών) αποφεύγουμε τις παρενθέσεις. Μπορούμε να πάρουμε την μεταθεματική σε ένα πέρασμα αν λάβουμε υπόψη τις προτεραιότητες των τελεστών στη γλώσσα για κάθε τελεστή ορίσουμε μια προτεραιότητα όταν είναι στη στοίβα και μια όταν διαβάζεται από την συμβολοσειρά. Το παρακάτω είναι ένα σύνολο προτεραιοτήτων τελεστών (που μπορεί να επεκταθεί σε όλους τους τελεστές)

<u>Σύμβολο</u>	<u>Προτεραιότητα στη Στοίβα</u>	<u>Προτεραιότητα Ανάγνωσης</u>
)	-	-
**	3	4
* /	2	2
+ -	1	1
(	0	4

## Υπολογισμός Αριθμητικών Εκφράσεων

$$14 - 3 * 2 + 7 = (14 - (3 * 2)) + 7$$

**Προτεραιότητα τελεστών**

το  $*$  έχει προτεραιότητα σε σχέση με τα  $+/-$

**Προσεταιριστική**

οι τελεστές με την ίδια προτεραιότητα υπολογίζονται από αριστερά προς τα δεξιά

Παράδειγμα:  $(x - y) + z$  αντί για  $x - (y + z)$

**Βασική Ιδέα:** ώθηση κάθε τελεστή στη στοίβα, αφαίρεση και εκτέλεση πρώτα των πράξεων με τελεστές μεγαλύτερης και *ίσης* προτεραιότητας.

# Αλγόριθμος Υπολογισμού Εκφράσεων

Δύο στοίβες:

- opStk έχει τους τελεστές
- valStk έχει τις τιμές
- Χρήση του \$ σαν ειδικού συμβόλου "end of input" με μικρότερη προτεραιότητα

Algorithm **doOp()**

```
x ← valStk.pop();
y ← valStk.pop();
op ← opStk.pop();
valStk.push( y op x )
```

Algorithm **repeatOps( refOp )**:

```
while ( valStk.size() > 1 ∧
    prec(refOp) ≤
    prec(opStk.top())
```

doOp()

Algorithm **EvalExp()**

Input: μια ροή από σύμβολα που  
παριστάνουν μια αριθμητική  
έκφραση (με αριθμούς)

Output: η τιμή της έκφρασης

**while** there's another token z

**if** isNumber(z) **then**

valStk.push(z)

**else**

repeatOps(z);

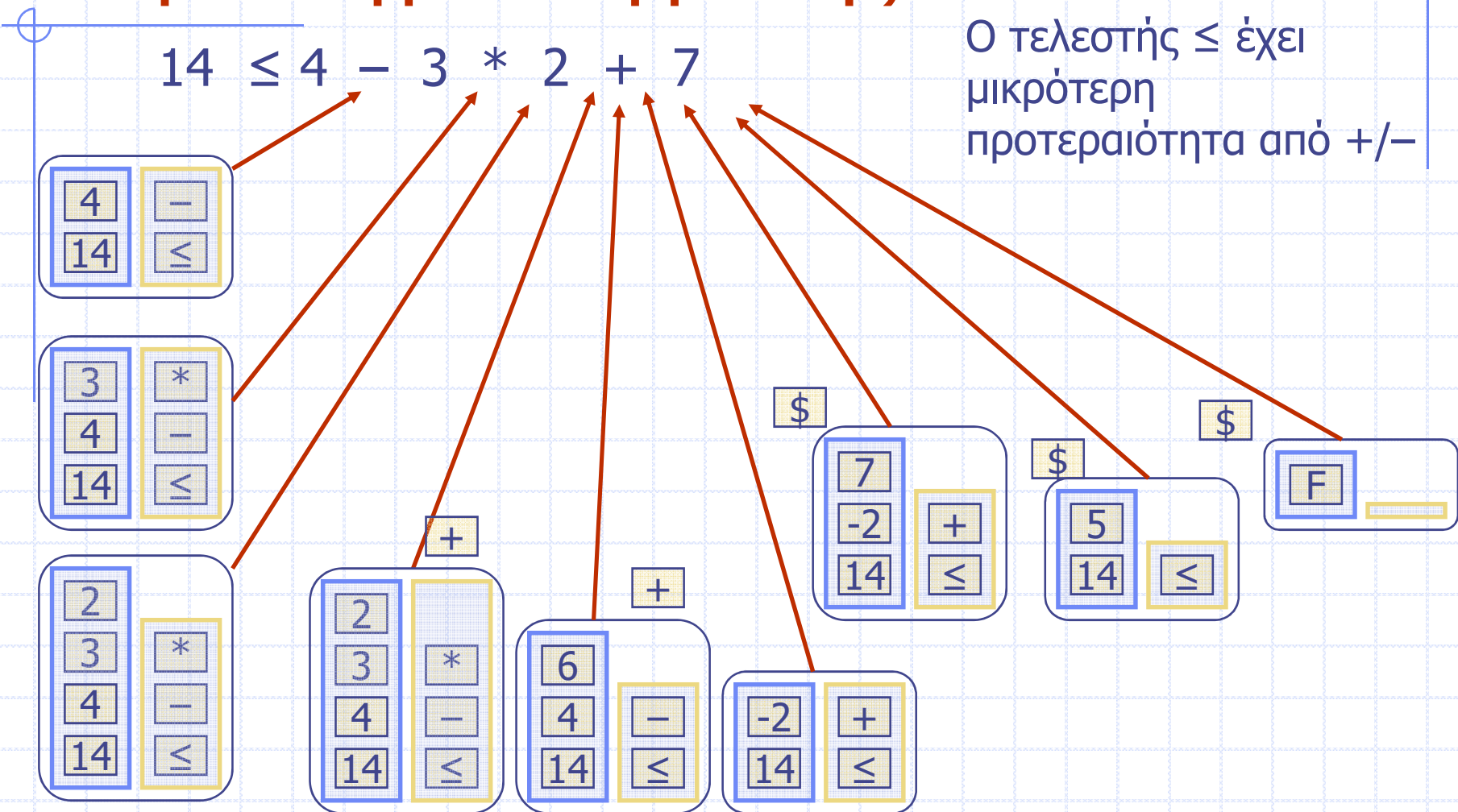
opStk.push(z)

repeatOps(\$);

**return** valStk.top()

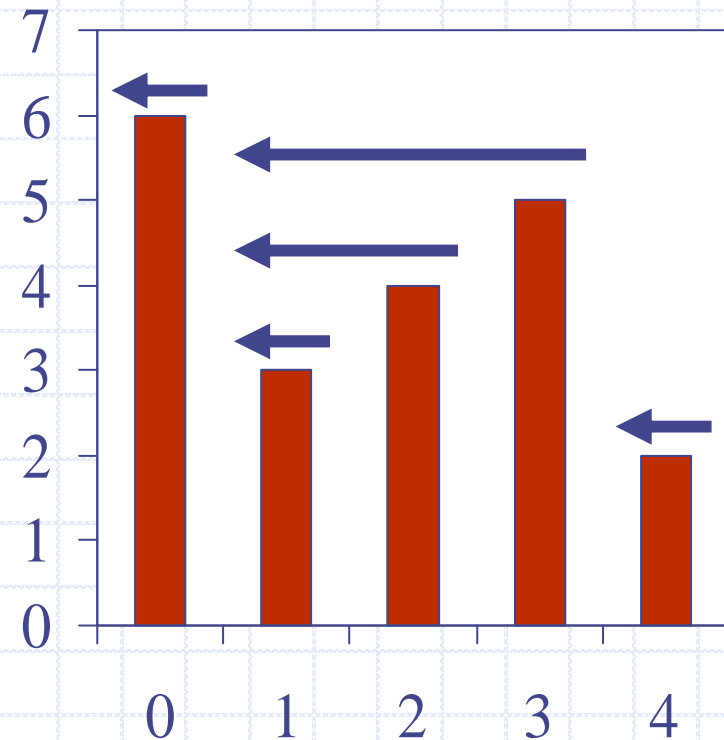


# Παράδειγμα έκφρασης



# Υπολογισμός Εκτάσεων

- Χρήση στοίβας σαν βοηθητικής δομής σε ένα αλγόριθμο
- Έστω ένας πίνακας, η **έκταση (span)**  $S[i]$  του  $X[i]$  είναι το μέγιστο πλήθος διαδοχικών στοιχείων  $X[j]$  άμεσα πριν το  $X[i]$  και τέτοιο ώστε  $X[j] \leq X[i]$
- Βρίσκουν εφαρμογές σε χρηματιστηριακές αναλύσεις
  - Π.χ., μετοχή για 52-εβδομάδες σε ψηλή τιμή



<i>X</i>	6	3	4	5	2
<i>S</i>	1	1	2	3	1

# Τετραγωνικός Αλγόριθμος

**Algorithm** *spans1*( $X, n$ )

Είσοδος πίνακας  $X$  με  $n$  ακεραίους

Έξοδος πίνακας  $S$  των εκτάσεων του  $X$  #

$S \leftarrow$  new array of  $n$  integers  $n$

for  $i \leftarrow 0$  to  $n - 1$  do  $n$

$s \leftarrow 1$   $n$

        while  $s \leq i \wedge X[i - s] \leq X[i]$   $1 + 2 + \dots + (n - 1)$

$s \leftarrow s + 1$   $1 + 2 + \dots + (n - 1)$

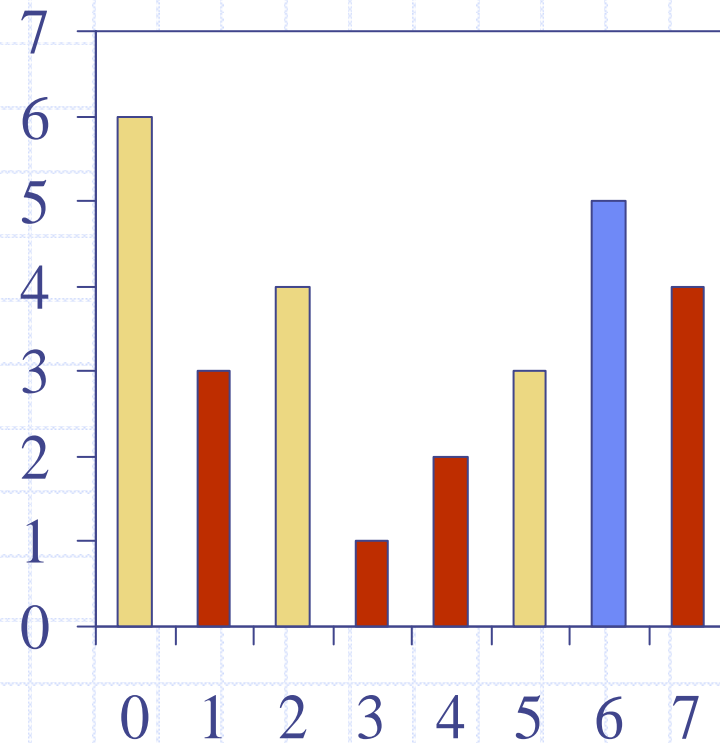
$S[i] \leftarrow s$   $n$

return  $S$  1

◆ Ο αλγόριθμος *spans1* τρέχει σε  $O(n^2)$  χρόνο

# Υπολογισμός με στοίβα

- Κρατάμε σε μια στοίβα τους δείκτες των στοιχείων που είναι ορατά όταν “κοιτάμε πίσω”
- Σαρώνουμε τον πίνακα από αριστερά προς τα δεξιά
  - Έστω  $i$  η τρέχουσα τιμή του δείκτη
  - Βγάζουμε δείκτες από τη στοίβα μέχρι να βρούμε το δείκτη  $j$  τέτοιο ώστε  $X[i] < X[j]$
  - Θέτουμε  $S[i] \leftarrow i - j$
  - Βάζουμε το  $x$  στη στοίβα



# Γραμμικός Αλγόριθμος

◆ Κάθε δείκτης του πίνακα

- Μπαίνει στη στοίβα ακριβώς μια φορά
- Βγαίνει από τη στοίβα το πολύ μια φορά

◆ Οι εντολές στην επανάληψη while εκτελούνται το πολύ  $n$  φορές

◆ Ο αλγόριθμος *spans2* τρέχει σε χρόνο  $O(n)$

<b>Algorithm</b> <i>spans2</i> ( $X, n$ )	#
$S \leftarrow$ πίνακας με $n$ ακέραιους	$n$
$A \leftarrow$ νέα κενή στοίβα	1
<b>for</b> $i \leftarrow 0$ <b>to</b> $n - 1$ <b>do</b>	$n$
<b>while</b> $(\neg A.isEmpty() \wedge$	
$X[A.top()] \leq X[i])$ <b>do</b>	$n$
$A.pop()$	$n$
<b>if</b> $A.isEmpty()$ <b>then</b>	$n$
$S[i] \leftarrow i + 1$	$n$
<b>else</b>	
$S[i] \leftarrow i - A.top()$	$n$
$A.push(i)$	$n$
<b>return</b> $S$	1

# Αρουραίός στο Λαβύρινθο

