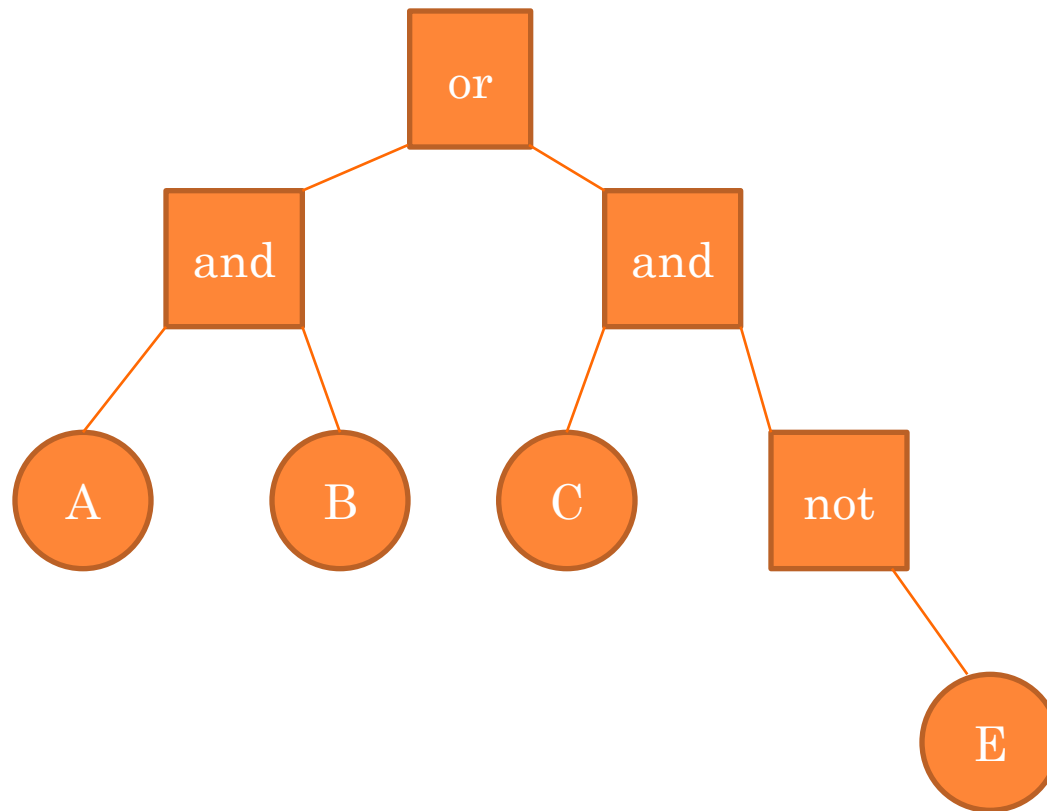


# IZRAZNA DREVESA (*EXPRESSION TREES*)

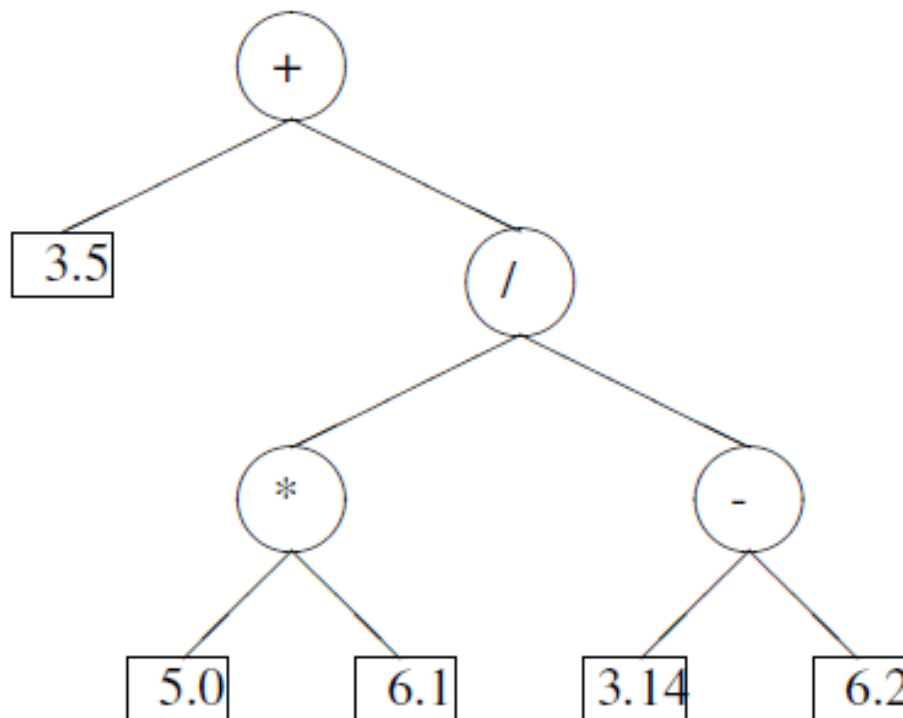
Logični izraz:

A and B or C and not E



# IZRAZNA DREVESA

izraz  $3.5 + 5.0 * 6.1 / (3.14 - 6.2)$

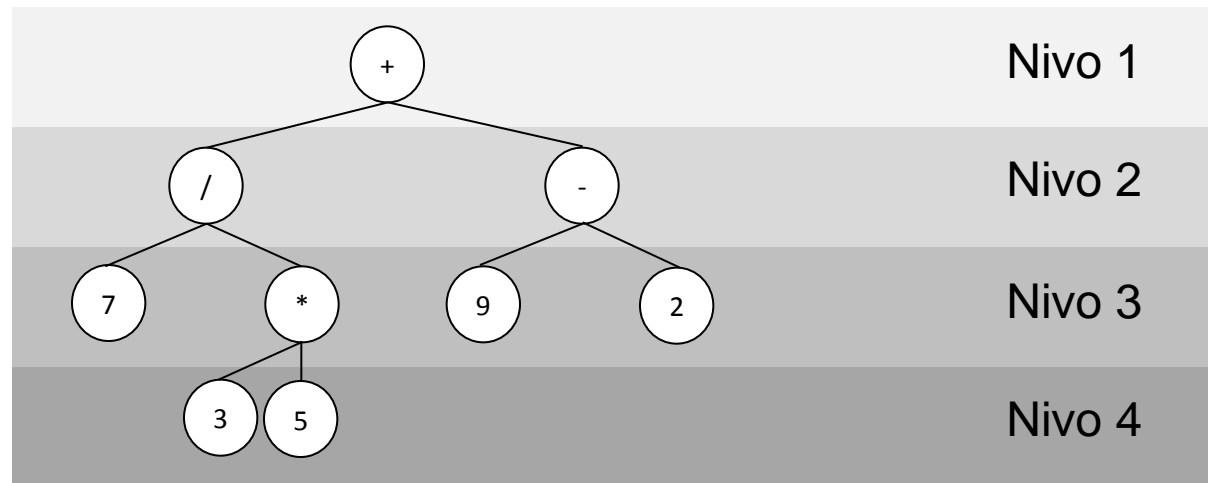


# IZRAZNA DREVEŠA: IMPLEMENTACIJA

```
public class ArithmeticExprNode {  
  
    int operator ;  
    double value ;  
    ArithmeticExprNode left, right ;  
} // class ArithmeticExprNode  
  
public class ArithmeticExprTree {  
    static final int PLUS = '+' ;  
    static final int MINUS = '-' ;  
    static final int TIMES = '*' ;  
    static final int DIVIDE = '/' ;  
    static final int L_BRACKET = '(' ;  
    static final int R_BRACKET = ')' ;  
    static final int NUM_VALUE = '\0' ;  
  
    ArithmeticExprNode rootNode ;  
  
} // class ArithmeticExprTree
```



# PRIMER: IZRAČUN IZRAZA



**Izrazno drevo:** v notranjih vozliščih operacije, v listih števila

$$7/(3*5)+(9-2)$$

Kaj je rekurzijska spremenljivka? ( $T$  = koren drevesa)

Kaj mi pomaga rešitev manjšega problema?

Iz vrednosti obeh poddreves lahko izračunam vrednost drevesa.

Kaj je robni pogoj? ( $T$  = Število,  $v(\text{Število}) = \text{Število}$ )

Kaj je splošni primer?

$$v(\text{operator}) = v(T1) \text{ operator } v(T2)$$

# IZRAZNA DREVESA

```
private double evaluate(ArithmeticExprNode node) {
    if (node == null)
        return 0.0 ; // alternatively throw exception ;
    else if (node.operator == NUM_VALUE)
        return node.value ;
    else {
        double lValue = evaluate(node.left) ;
        double rValue = evaluate(node.right) ;
        switch (node.operator) {
            case PLUS: return lValue + rValue ;
            case MINUS: return lValue - rValue ;
            case TIMES: return lValue * rValue ;
            case DIVIDE: return lValue / rValue ;
            default: return 0.0 ; // alternatively throw exception ;
        } // switch
    } // else
} // evaluate
```



# GRADNJA IZRAZNIH DREVES

---

Iz danega izraza na vhodu želimo zgraditi izrazno drevo.

- 1) Sintaktična analiza vhoda
- 2) Gradnja drevesa

Sintaktična analiza:

poznati moramo strukturo vseh možnih izrazov.

- Leva asociativnost:  $a+b+c = (a+b)+c$
- Vezava (prioriteta) operatorjev:  $a+b*c = a+(b*c)$

# SINTAKTIČNA ANALIZA

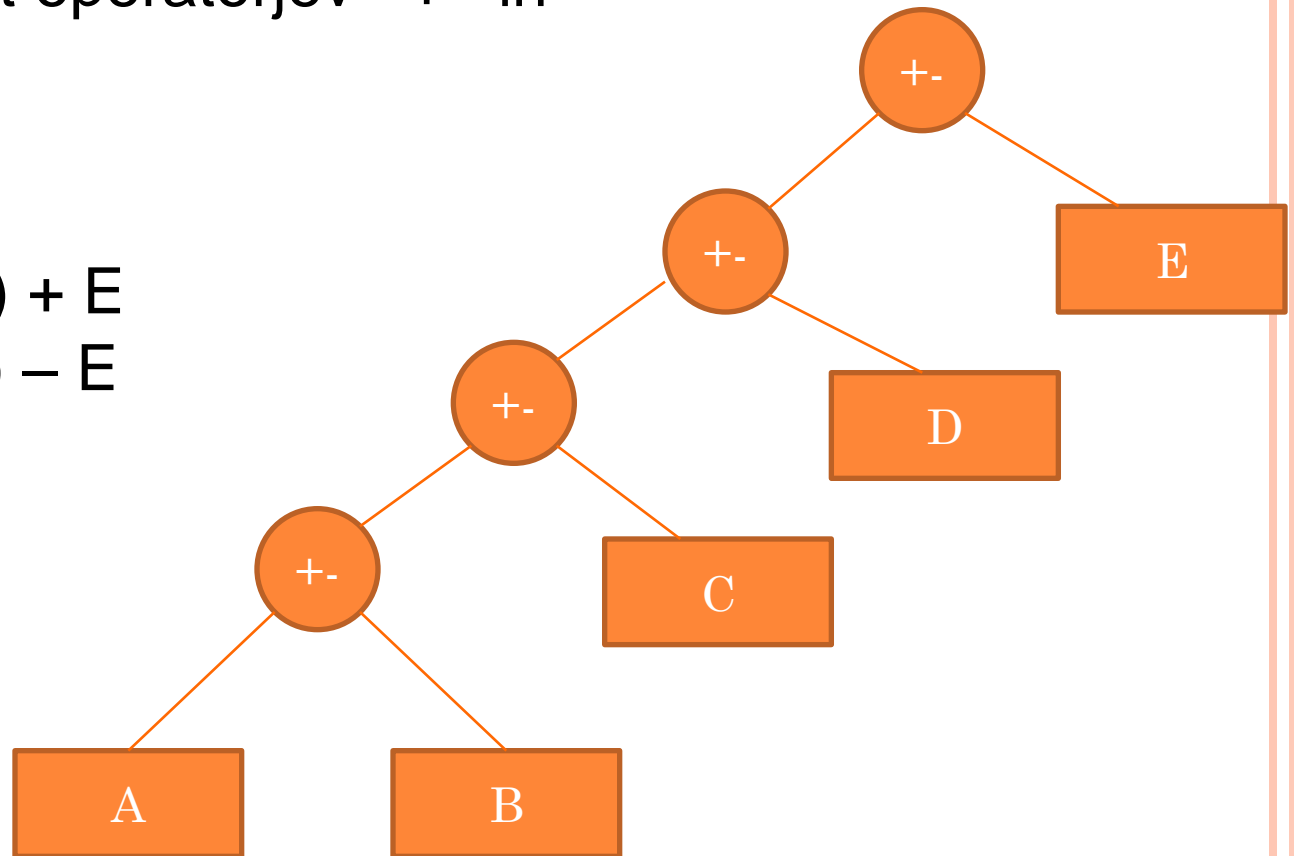
Leva asociativnost operatorjev + in -

$A + B + C + D + E$

$A - B - C - D - E$

$((A + B) + C) + D + E$

$((A - B) - C) - D - E$



# SINTAKTIČNA ANALIZA

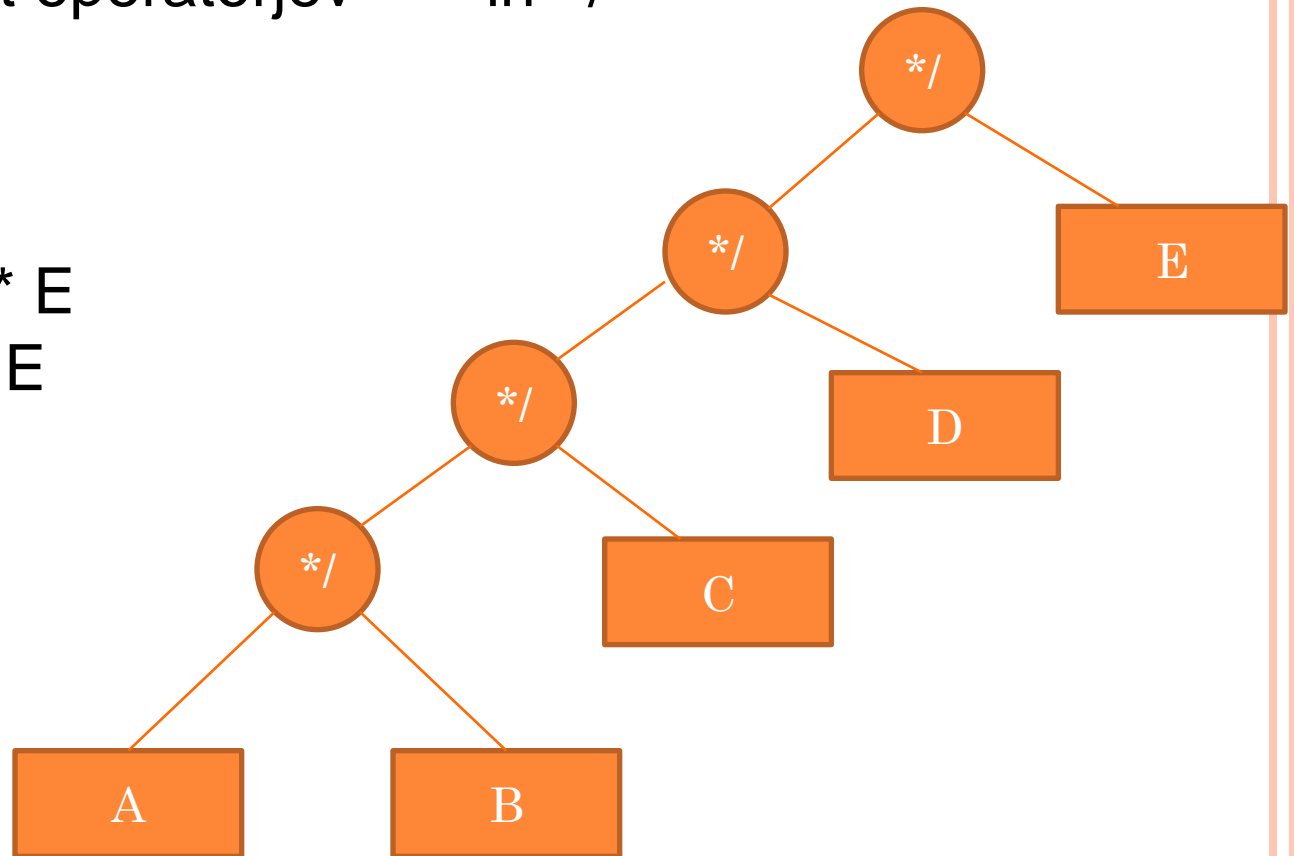
Leva asociativnost operatorjev \* in /

$A * B * C * D * E$

$A / B / C / D / E$

$((A * B) * C) * D * E$

$((A / B) / C) / D / E$





# KONTEKSTNO NEODVISNE GRAMATIKE

---

Pomagamo si s kontekstno neodvisno gramatiko, ki opisuje vse možne aritmetične izraze.

Gramatično pravilo:

$\langle x \rangle ::= \text{blabla}$

pomeni, da lahko pri generiranju izraza spremenljivko  $\langle x \rangle$  zamenjamo z izrazom **blabla**.

Gramatika je torej množica pravil, ki bodo iz začetne spremenljivke  $\langle \text{izraz} \rangle$  lahko zgenerirala poljuben aritmetični izraz.



# KONTEKSTNO NEODVISNA GRAMATIKA

$\langle \text{Izraz} \rangle ::= \langle \text{produkt} \rangle$

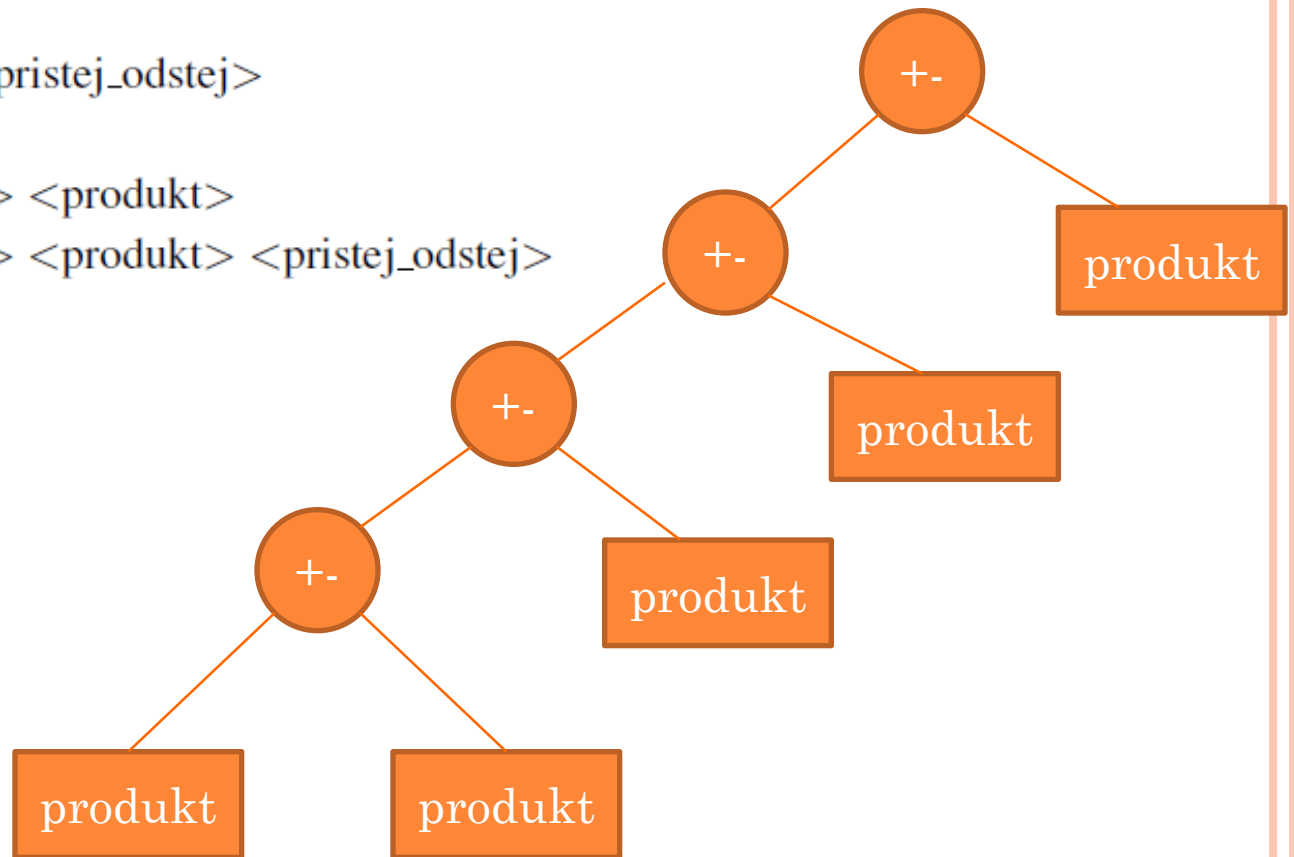
$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle + - \rangle ::= +$

$\langle + - \rangle ::= -$



$\langle \text{izraz} \rangle \rightarrow \langle p \rangle \langle po \rangle \rightarrow \langle p \rangle \langle + - \rangle \langle p \rangle \langle po \rangle \rightarrow \langle p \rangle \langle + - \rangle \langle p \rangle \langle + - \rangle \langle p \rangle \langle po \rangle \rightarrow$   
 $\langle p \rangle \langle + - \rangle \langle p \rangle \langle + - \rangle \langle p \rangle \langle + - \rangle \langle p \rangle \langle po \rangle \rightarrow \langle p \rangle \langle + - \rangle \langle p \rangle \langle + - \rangle \langle p \rangle \langle + - \rangle \langle p \rangle \langle + - \rangle \langle p \rangle$

# KONTEKSTNO NEODVISNA GRAMATIKA

$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle$

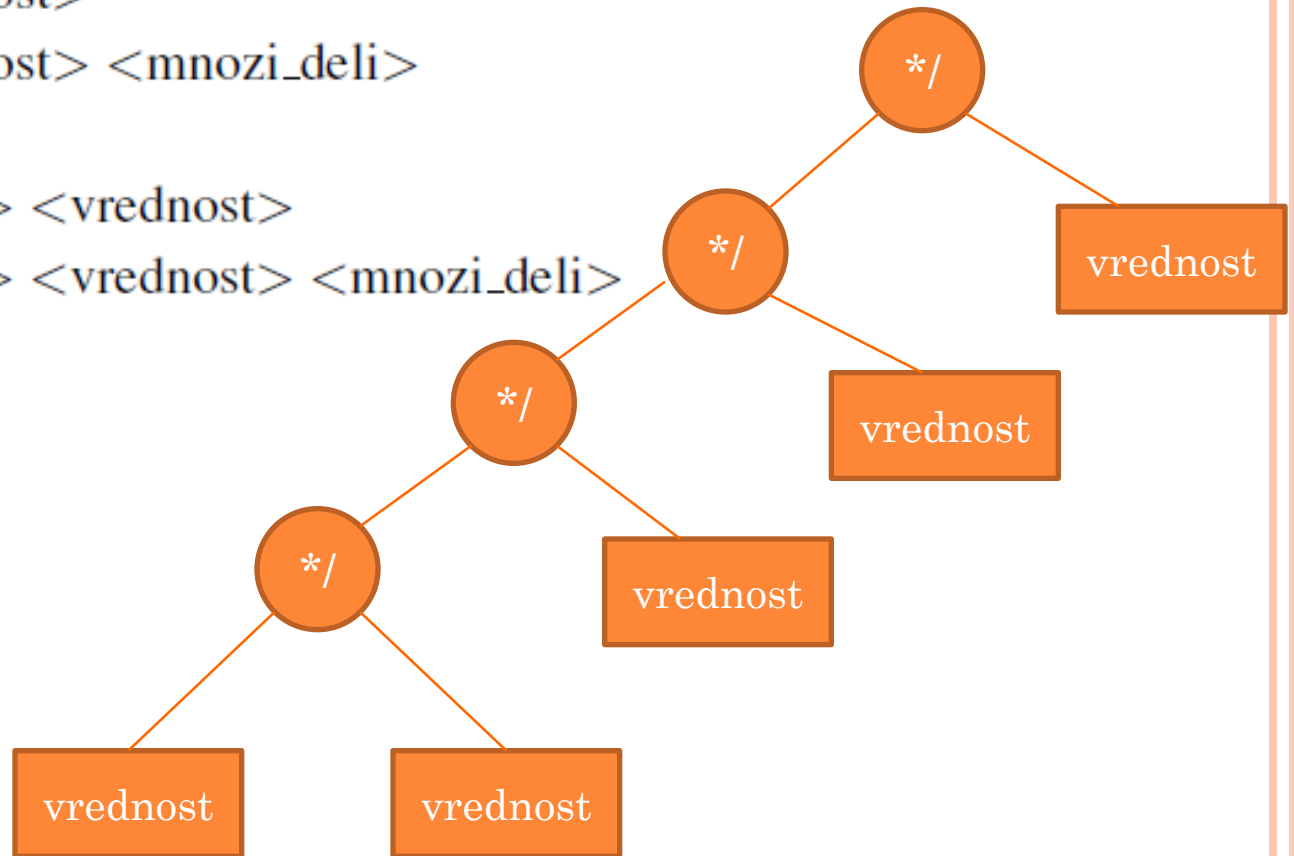
$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle \langle \text{mnozi\_deli} \rangle$

$\langle \text{mnozi\_deli} \rangle ::= \langle \text{*/} \rangle \langle \text{vrednost} \rangle$

$\langle \text{mnozi\_deli} \rangle ::= \langle \text{*/} \rangle \langle \text{vrednost} \rangle \langle \text{mnozi\_deli} \rangle$

$\langle \text{*/} \rangle ::= *$

$\langle \text{*/} \rangle ::= /$



# KONTEKSTNO NEODVISNA GRAMATIKA



---

Treba je definirati samo še <vrednost>

$\langle \text{vrednost} \rangle ::= \langle \text{stevilo} \rangle$

$\langle \text{vrednost} \rangle ::= ( \langle \text{izraz} \rangle )$



# KONTEKSTNO NEODVISNA GRAMATIKA

$\langle \text{Izraz} \rangle ::= \langle \text{produkt} \rangle$

$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle + - \rangle ::= +$

$\langle + - \rangle ::= -$

$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle$

$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle \langle \text{mnozi\_deli} \rangle$

$\langle \text{mnozi\_deli} \rangle ::= \langle * / \rangle \langle \text{vrednost} \rangle$

$\langle \text{mnozi\_deli} \rangle ::= \langle * / \rangle \langle \text{vrednost} \rangle \langle \text{mnozi\_deli} \rangle$

$\langle * / \rangle ::= *$

$\langle * / \rangle ::= /$

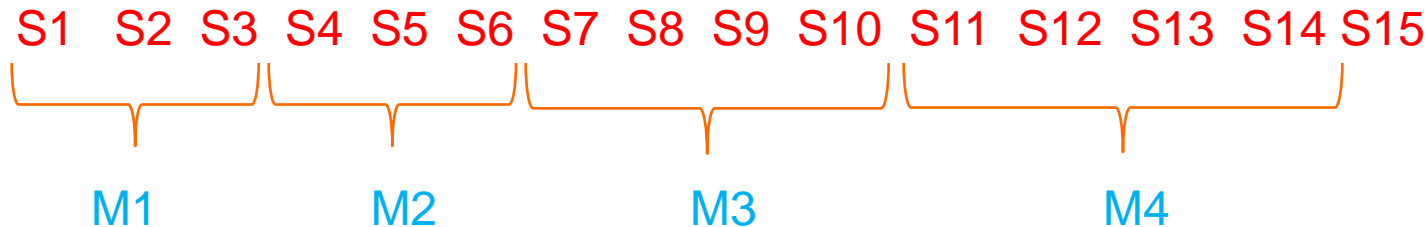
$\langle \text{vrednost} \rangle ::= \langle \text{stevilo} \rangle$

$\langle \text{vrednost} \rangle ::= ( \langle \text{izraz} \rangle )$



# SINTAKTIČNA ANALIZA

- izraz beremo na vhodu od leve proti desni
- beremo po en simbol naenkrat, ki je lahko (, ), +, -, \*, / ali **število**
- za vsako spremenljivko uvedemo eno metodo (razen za trivialne)
- vsaka metoda na začetku dobi prvi prebrani simbol in na koncu vrne en (prebrani) simbol, ki je prvi simbol za naslednjo metodo



M1 dobi S1 in vrne S4

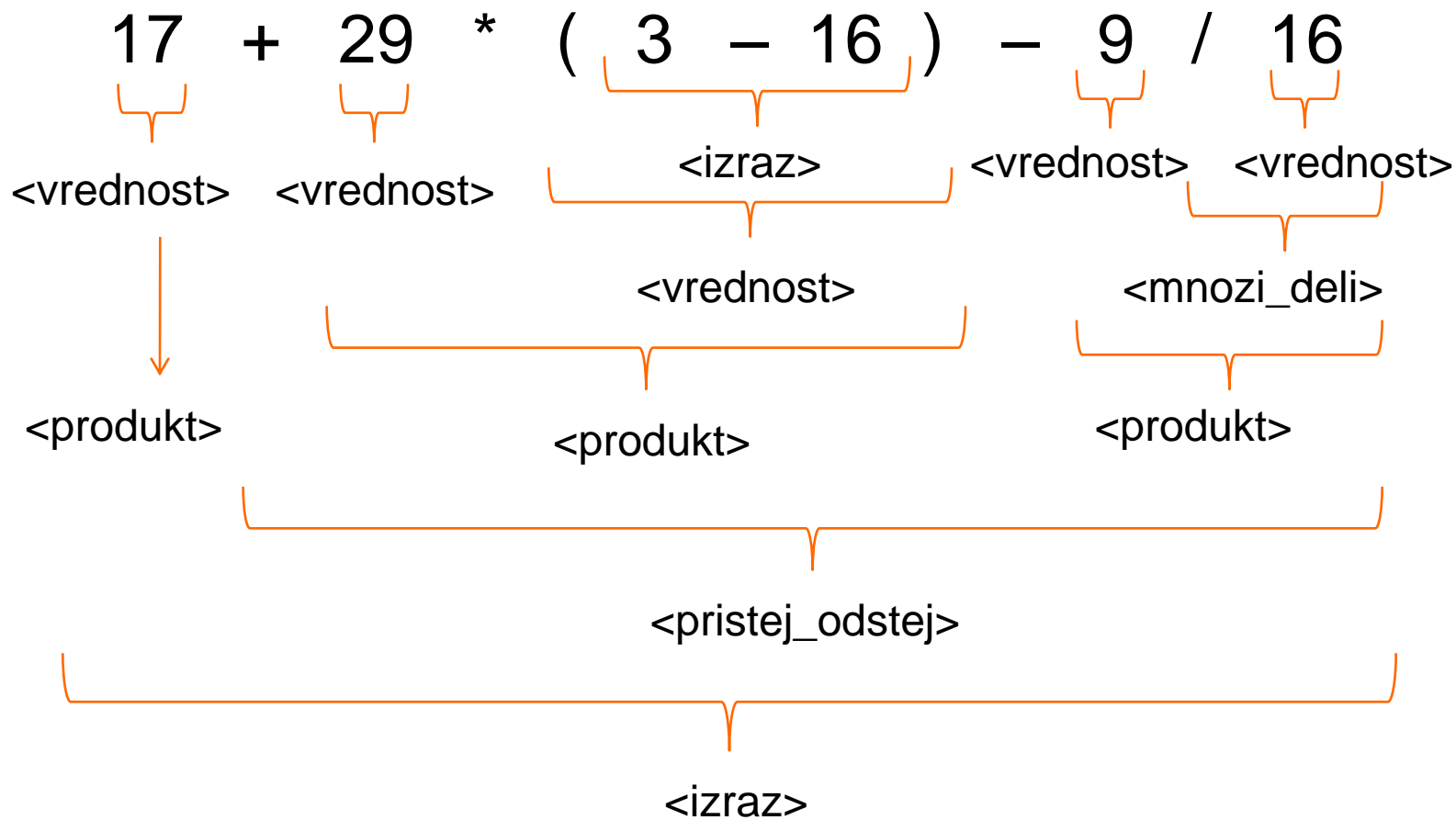
M2 dobi S4 in vrne S7

M3 dobi S7 in vrne S11

M4 dobi S11 in vrne S15



# SINTAKTIČNA ANALIZA



# SINTAKTIČNA ANALIZA

---

Napisati moramo metode:

<code>&lt;izraz&gt;</code>	<code>expression</code>
<code>&lt;produkt&gt;</code>	<code>product</code>
<code>&lt;pristej_odstej&gt;</code>	<code>addSubtract</code>
<code>&lt;vrednost&gt;</code>	<code>value</code>
<code>&lt;mnozi_deli&gt;</code>	<code>multiplyDivide</code>





# SINTAKTIČNA ANALIZA

Branje simbolov v javi: `StreamTokenizer`

```
StreamTokenizer st; // st je novi "bralec" simbolov
```

```
st.nextToken(); // prebere naslednji simbol
```

vrsta prebranega simbola `st.ttype` je lahko

-PLUS

-MINUS

-TIMES

-DIVIDE

-L\_BRACKET

-R\_BRACKET

-TT\_NUMBER

-TT\_EOF // konec vhodnega niza

Če je vrsta simbola `TT_NUMBER`,  
je vrednost prebranega števila `st.nval`



# IZRAZNA DREVEŠA: IMPLEMENTACIJA

```
public class ArithmeticExprNode {  
  
    int operator ;  
    double value ;  
    ArithmeticExprNode left, right ;  
} // class ArithmeticExprNode  
  
public class ArithmeticExprTree {  
    static final int PLUS = '+' ;  
    static final int MINUS = '-' ;  
    static final int TIMES = '*' ;  
    static final int DIVIDE = '/' ;  
    static final int L_BRACKET = '(' ;  
    static final int R_BRACKET = ')' ;  
    static final int NUM_VALUE = '\0' ;  
  
    ArithmeticExprNode rootNode ;  
  
} // class ArithmeticExprTree
```



# SINTAKTIČNA ANALIZA + DREVO

Med sintaktično analizo gradimo izrazno drevo.

Vsaka metoda dobi:

- prvi (že prebrani) simbol
- eventuelne dele za gradnjo svojega (pod)drevesa

Vsaka metoda vrne

- naslednji prebrani simbol
- ustrezno poddrevo (kazalec na koren)
- eventuelne dele poddrevesa za naslednjo metodo

Glavna metoda prebere prvi simbol in pokliče metodo `expression`, na koncu pa še preveri zaključek analize:

```
public void exprToTree(StreamTokenizer st) throws IOException {  
    st.nextToken() ;  
    rootNode = expression(st) ;  
    if (st.ttype != StreamTokenizer.TT_EOF)  
        System.out.println("Expression_error");  
} // exprToTree
```



# SINTAKTIČNA ANALIZA + DREVO

$\langle \text{Izraz} \rangle ::= \langle \text{produkt} \rangle$

$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

```
private ArithmeticExprNode expression(StreamTokenizer st)
                                throws IOException {
    ArithmeticExprNode root = product(st) ;
    if (st.ttype == PLUS || st.ttype == MINUS)
        return addSubtract(st, root) ;
    else return root ;
} // expression
```



# SINTAKTIČNA ANALIZA + DREVO

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

```
private ArithmeticExprNode addSubtract(StreamTokenizer st,  
                                         ArithmeticExprNode left) throws IOException{
```

```
    ArithmeticExprNode root = new ArithmeticExprNode(st.ttype) ;
```

```
    root.left = left ;
```

```
    st.nextToken() ;
```

```
    root.right = product(st) ;
```

```
if (st.ttype == PLUS || st.ttype == MINUS)
```

```
    return addSubtract(st, root) ;
```

```
else return root ;
```

```
} // addSubtract
```



# GRAMATIKA DOLOČA PROGRAM

$\langle \text{Izraz} \rangle ::= \langle \text{produkt} \rangle$

$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle$

$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle \langle \text{mnozi\_deli} \rangle$



# GRAMATIKA DOLOČA PROGRAM

```
private ArithmeticExprNode expression(StreamTokenizer st)
                                throws IOException {
    ArithmeticExprNode root = product(st) ;
    if (st.ttype == PLUS || st.ttype == MINUS)
        return addSubtract(st, root) ;
    else return root ;
} // expression
```

```
private ArithmeticExprNode product(StreamTokenizer st)
                                throws IOException {
    ArithmeticExprNode root = value(st) ;
    if (st.ttype == TIMES || st.ttype == DIVIDE)
        return multiplyDivide(st, root) ;
    else return root ;
} // product
```



# GRAMATIKA DOLOČA PROGRAM

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle \text{mnozi\_deli} \rangle ::= \langle * / \rangle \langle \text{vrednost} \rangle$

$\langle \text{mnozi\_deli} \rangle ::= \langle * / \rangle \langle \text{vrednost} \rangle \langle \text{mnozi\_deli} \rangle$





# GRAMATIKA DOLOČA PROGRAM

```
private ArithmeticExprNode addSubtract(StreamTokenizer st,  
                                         ArithmeticExprNode left) throws IOException{
```

```
    ArithmeticExprNode root = new ArithmeticExprNode(st.ttype) ;
```

```
    root.left = left ;
```

```
    st.nextToken() ;
```

```
    root.right = product(st) ;
```

```
    if (st.ttype == PLUS || st.ttype == MINUS)
```

```
        return addSubtract(st, root) ;
```

```
    else return root ;
```

```
} // addSubtract
```

```
private ArithmeticExprNode multiplyDivide(StreamTokenizer st,  
                                           ArithmeticExprNode left) throws IOException{
```

```
    ArithmeticExprNode root = new ArithmeticExprNode(st.ttype) ;
```

```
    root.left = left ;
```

```
    st.nextToken() ;
```

```
    root.right = value(st) ;
```

```
    if (st.ttype == TIMES || st.ttype == DIVIDE) // product continues
```

```
        return multiplyDivide(st, root) ;
```

```
    else return root ;
```

```
} // multiplyDivide
```



# SINTAKTIČNA ANALIZA + DREVO

```
private ArithmeticExprNode value(StreamTokenizer st) throws IOException {
    if (st.ttype == StreamTokenizer.TT_NUMBER) {
        ArithmeticExprNode node =
            new ArithmeticExprNode(NUM_VALUE, st.nval);
        st.nextToken();
        return node;
    }
    else if (st.ttype != L_BRACKET) // should be left bracket
        return null; // error, alternatively throw exception
    else {
        st.nextToken();
        ArithmeticExprNode node = expression(st); // indirect recursion
        if (st.ttype != R_BRACKET) return null;
        else {
            st.nextToken();
            return node;
        }
    }
} // value
```

$\langle \text{vrednost} \rangle ::= \langle \text{stevilo} \rangle$

$\langle \text{vrednost} \rangle ::= ( \langle \text{izraz} \rangle )$

# REKURZIJA → ITERACIJA

```
private ArithmeticExprNode addSubtract(StreamTokenizer st,  
                                         ArithmeticExprNode left) throws IOException{
```

```
    ArithmeticExprNode root = new ArithmeticExprNode(st.ttype) ;  
    root.left = left ;  
    st.nextToken() ;  
    root.right = product(st) ;  
    if (st.ttype == PLUS || st.ttype == MINUS)  
        return addSubtract(st, root) ;  
    else return root ;  
} // addSubtract
```

```
private ArithmeticExprNode multiplyDivide(StreamTokenizer st,  
                                           ArithmeticExprNode left) throws IOException{
```

```
    ArithmeticExprNode root = new ArithmeticExprNode(st.ttype) ;  
    root.left = left ;  
    st.nextToken() ;  
    root.right = value(st) ;  
    if (st.ttype == TIMES || st.ttype == DIVIDE) // product continues  
        return multiplyDivide(st, root) ;  
    else return root ;  
} // multiplyDivide
```



# REKURZIJA → ITERACIJA

$\langle \text{Izraz} \rangle ::= \langle \text{produkt} \rangle$

$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle$

$\langle \text{pristej\_odstej} \rangle ::= \langle + - \rangle \langle \text{produkt} \rangle \langle \text{pristej\_odstej} \rangle$



$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \{ \langle + - \rangle \langle \text{produkt} \rangle \}$



# IZZIV

$\langle \text{izraz} \rangle ::= \langle \text{produkt} \rangle \{ \langle + - \rangle \langle \text{produkt} \rangle \}$

$\langle + - \rangle ::= +$

$\langle + - \rangle ::= -$

$\langle \text{produkt} \rangle ::= \langle \text{vrednost} \rangle \{ \langle * / \rangle \langle \text{vrednost} \rangle \}$

$\langle * / \rangle ::= *$

$\langle * / \rangle ::= /$

$\langle \text{vrednost} \rangle ::= \langle \text{stevilo} \rangle$

$\langle \text{vrednost} \rangle ::= ( \langle \text{izraz} \rangle )$

