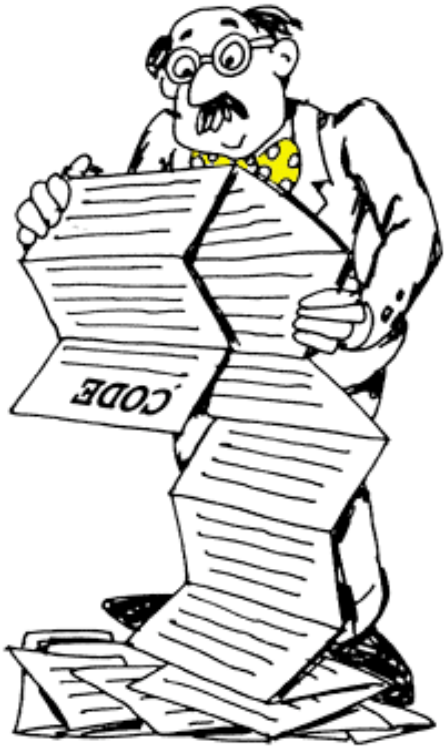
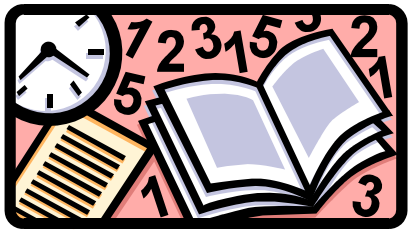
Parsing



Parsing



Mål:

Et program til indlæsning og beregning af aritmetiske udtryk

Eksempel: Beregn (3*5 + 4/2) - 1

Løs et lettere problem først:

Læs en streng og undersøg, om den udgør et lovligt aritmetisk udtryk

Grammatik for aritmetiske udtryk

Benyt en **grammatik** til at beskrive aritmetiske udtryk:

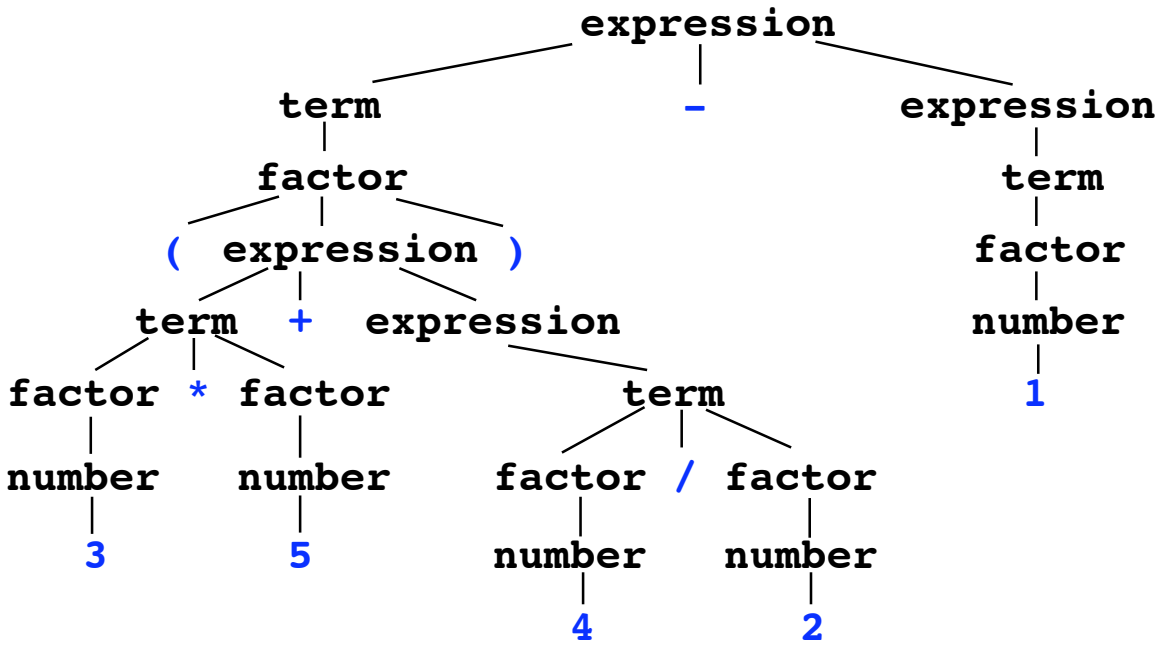
<expression> ::= <term> |
<term> + <expression> |
<term> – <expression>
<term> ::= <factor> |
<factor> * <term> |
<factor> / <term>
<factor> ::= <number> |
(<expression>)

Grammatikken er beskrevet ved **produktionsregler** og består af
(1) **nonterminale** symboler: **expression**, **term, factor** og **number**
(2) **terminale** symboler: `+`, `–`, `*`, `/`, `(`, `)` og cifre
(3) **metasymboler**: `::=`, `<`, `>`, og `|`

Syntaksanalyse

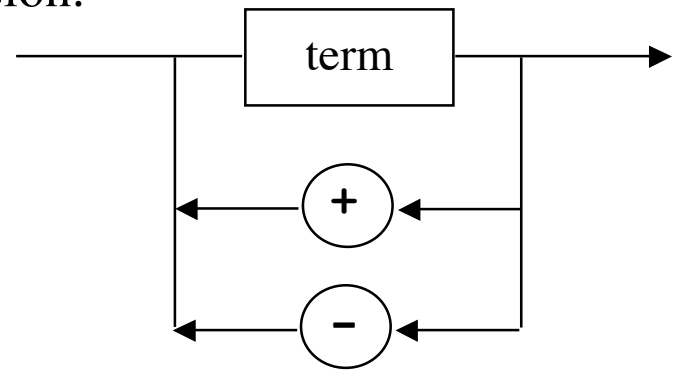
En streng er et aritmetisk udtryk, hvis det ved hjælp af produktionsreglerne er muligt at **udlede** strengen ud fra **`expression`**, d.v.s. ud fra **`expression`** i en række skridt nå frem til strengen ved i hvert skridt at erstatte et nonterminal-symbol med et af alternativerne på højresiden af en produktion for dette symbol

Syntakstræ for `(3*5+4/2)-1`

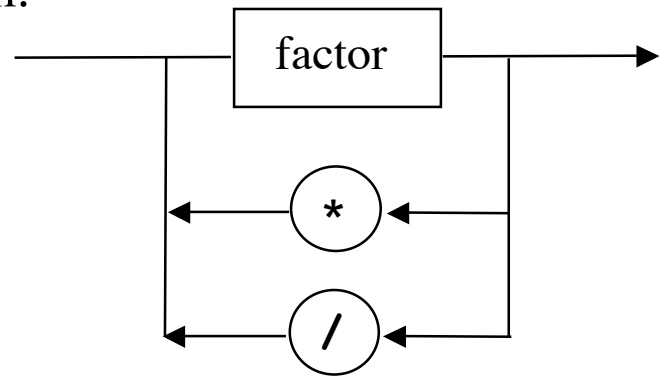


Syntaksdiagrammer

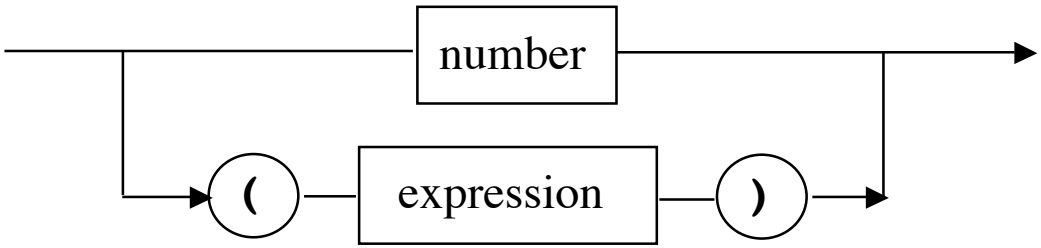
expression:



term:



factor:



Syntaksanalyse ved rekursiv nedstigning (top-down parsing)

Et rekursivt Java-program til syntaksanalyse kan konstrueres direkte ud fra syntaksdiagrammerne

```
void expression() {
    term();
    while (token == PLUS || token == MINUS)
      { getToken(); term(); }
}
```

```
static final int PLUS   = 1, MINUS = 2,
                 MULT   = 3, DIV   = 4,
                 LPAR   = 5, RPAR  = 6,
                 NUMBER = 7, EOS   = 8;
int token;
```

```
void term() {
    factor();
    while (token == MULT || token == DIV)
        { getToken(); factor(); }
}
```

```
void factor() {
    if (token == NUMBER)
        ;
    else if (token == LPAR) {
        getToken();
        expression();
        if (token != RPAR)
            error("missing right paranthesis");
    } else
        error("illegal factor: " + token);
    getToken();
}
```

```
StringTokenizer str;
```

```
void parse(String s) {
    str = new StringTokenizer(s,"+-*/() ",true);
    getToken();
    expression();
}
```

Eksempel på kald:

```
parse("(3*5+4/2)-1");
```

```
void getToken() {
    String s;
    try {
        s = str.nextToken();
    } catch(NoSuchElementException e) {
        token = EOS;
        return;
    }
    if (s.equals(" ")) getToken();
    else if (s.equals("+")) token = PLUS;
    else if (s.equals("-")) token = MINUS;
    else if (s.equals("*")) token = MULT;
    else if (s.equals("/")) token = DIV;
    else if (s.equals("(")) token = LPAR;
    else if (s.equals(")")) token = RPAR;
    else {
        try {
            Double.parseDouble(s);
            token = NUMBER;
        } catch(NumberFormatException e)
          { error("number expected"); }
    }
}
```

Beregning af aritmetiske udtryk

Beregning kan opnås ved få simple ændringer af syntaksanalyse-programmet

Analysemetoderne skal returnere med tilhørende værdi
(i stedet for **`void`**)

```
double valueOf(String s) {
    str = new StringTokenizer(s,"+-*/() ",true);
    getToken();
    return expression();
}
```

Eksempel på kald:

```
double r = valueOf("(3*5+4/2)-1");
```

```
double expression() {
    double v = term();
    while (token == PLUS || token == MINUS)
        if (token == PLUS)
          { getToken(); v += term(); }
        else
          { getToken(); v -= term(); }
    return v;
}
```

```
double term() {
    double v = factor();
    while (token == MULT || token == DIV)
        if (token == MULT)
          { getToken(); v *= factor(); }
        else
          { getToken(); v /= factor(); }
    return v;
}
```

```
double factor() {
    double v;
    if (token == NUMBER)
        v = value;
    else if (token == LPAR) {
        getToken();
        v = expression();
        if (token != RPAR)
            error("missing right paranthesis");
    }
    else
        error("illegal factor: " + token);
    getToken();
    return v;
}
```

```
void getToken() {
    String s;
    try {
        s = str.nextToken();
    } catch(NoSuchElementException e) {
        token = EOS;
        return;
    }
    if (s.equals(" ")) getToken();
    else if (s.equals("+")) token = PLUS;
    else if (s.equals("-")) token = MINUS;
    else if (s.equals("*")) token = MULT;
    else if (s.equals("/")) token = DIV;
    else if (s.equals("(")) token = LPAR;
    else if (s.equals(")")) token = RPAR;
    else {
        try{
            value = Double.parseDouble(s);
            token = NUMBER;
        } catch(NumberFormatException e)
          { error("number expected"); }
    }
}
```