1 Concrete Syntax of Jex

```
Program     =  { Statement | Definition }.
Definition  =  Formal ";"
            |  FunDef
            |  "import" { IDENT "." } ( "*" | IDENT ) ";"
            .
FunDef      =  Type IDENT "(" [ Formals ] ")" Statement.
Type        =  "int" | "boolean" | IDENT.
Formals     =  Formal { "," Formal }.
Formal      =  Type IDENT.
```



```
Statement   =  "if" "(" Expr ")" Statement [ "else" Statement ]
            |  "while" "(" Expr ")" Statement
            |  "{" { Statement | Formal ";" } "}"
            |  Expr ";"
            |  [ Expr "." ] IDENT "=" Expr ";"
            |  "return" Expr ";"
            .
Expr        =  NUMLIT | STRINGLIT | "true" | "false"
            |  [ Expr "." ] IDENT [ "(" [ Exprs ] ")" ]
            |  Expr Operator Expr
            |  "new" Type "(" [ Exprs ] ")"
            |  "(" Expr ")"
            .
Operator    =  "+" | "−" | "*" | "/"
            |  "==" | "!=" | "<" | ">" | "≤" | "≥"
            |  "&&" | "||"
            .
Exprs       =  Expr { "," Expr }.
```


2 Lexical Syntax of Jex

- Tokens

```
ident     = letter { letter | digit }.
numlit    = digit { digit }.
stringlit = "\"" { "\" noNewline | noBackslashOrQuoteOrNewlines } "\"".
```

  All quoted terminals in the concrete syntax are valid tokens.

- Auxiliary

```
letter      = "a" | ... | "z" | "A" | ... | "Z".
digit       = "0" | ... | "9".
noNewline   = ...
```

- White space and comments

```
whitespace = " " | "\n" | "\t" | "\r".
comment    = "/" "*" { noStar | "*" { "*" } noStarOrSlash }
                       { "*" } "*" "/".
```


3 Operator expressions

We have binary operators with the following precedence:

- $*$ $/$
- $+$ $-$
- == != $<$ $>$ $\leq$ $\geq$
- &&
- ||

== and != also work on objects, the others only on primitive types.

4 Abstract Syntax of Jex

```
Tree = DEFLIST { Tree }
     | FUNDEF Tree ident { Tree } Tree
     | IMPORT { ident } boolean
     | VARDEF Tree ident
     | ASSIGN Tree Tree
     | IF Tree Tree Tree
     | WHILE Tree Tree
     | BLOCK { Tree }
     | EVAL Tree
     | RETURN Tree
```



```
     | NUMLIT int
     | STRINGLIT String
     | BOOLEANLIT boolean
     | IDENT ident
     | FUNCALL ident { Tree }
     | METHODCALL Tree ident { Tree }
     | FIELDACCESS Tree ident
     | OPERATION Tree Tree op
     | NEW Tree { Tree }
     | INTEGERTP
     | BOOLEANTP
     ;
```