PR Compiler Construction **Summer Term 2020**
AAU Klagenfurt / Institute of Information Technology **Taschwer**

YAPL Syntax

EBNF Syntax of YAPL

Lower-case non-terminal symbols are used for token definitions, upper-case non-terminal symbols are used within grammar productions only.

```
letter = "A" .. "Z" | "a" .. "z" | "_" .
digit = "0" .. "9" .
otherchar = "+" | "-" | "*" | "/" | "." | "," | ";" | ":" | "!"
          | "?" | "=" | "#" | "%" | "<" | ">" | "$" | "(" | ")"
          | "[" | "]" | "{" | "}" | "\" | "@" | "&" | "^" | "|" .

ident = letter { letter | digit } .
number = digit { digit } .
string = '"' { " " | letter | digit | otherchar } '"' .

RelOp = "<" | "<=" | ">=" | ">" .
EqualOp = "==" | "!=" .
AddOp = "+" | "-" .
MulOp = "*" | "/" | "%" .

Literal = "True" | "False" | number .
Selector = ( "[" Expr "]" | "." ident ) [ Selector ] .
ArrayLen = "#" ident [ Selector ] .
PrimaryExpr = Literal | "(" Expr ")" | ProcedureCall
            | ident [ Selector ] | ArrayLen .
UnaryExpr = [AddOp] PrimaryExpr .
MulExpr = UnaryExpr { MulOp UnaryExpr } .
AddExpr = MulExpr { AddOp MulExpr } .
RelExpr = AddExpr [ RelOp AddExpr ] .
EqualExpr = RelExpr [ EqualOp RelExpr ] .
CondAndExpr = EqualExpr { "And" EqualExpr } .
CreationExpr = "new" NonArrayType { "[" Expr "]" } .
Expr = CondAndExpr { "Or" CondAndExpr } | CreationExpr .

ArgumentList = Expr { "," Expr } .
ProcedureCall = ident "(" [ ArgumentList ] ")" .
Assignment = ident [ Selector ] ":=" Expr .
IfStatement = "If" Expr "Then" StatementList [ "Else" StatementList ] "EndIf" .
WhileStatement = "While" Expr "Do" StatementList "EndWhile" .
ReturnStatement = "Return" [ Expr ] .
WriteStatement = "Write" string .
Statement = IfStatement | WhileStatement | ReturnStatement
          | WriteStatement | Assignment | ProcedureCall | Block .
```

```
StatementList = { Statement ";" } .
Block = [ Decl ] "Begin" StatementList "End" .

NonArrayType = "int" | "bool" | ident .
Type = NonArrayType { "[" "]" } .
ReturnType = "void" | Type .

ConstDecl = "Const" ident "=" Literal ";" .
VarDecl = Type ident { "," ident } ";" .
TypeDecl = "Record" ident VarDecl { VarDecl } "EndRecord" ";" .
Decl = "Declare" { ConstDecl | VarDecl | TypeDecl } .

FormalParam = Type ident .
FormalParamList = FormalParam { "," FormalParam } .
Procedure = "Procedure" ReturnType ident "(" [ FormalParamList ] ")" Block ident ";" .

Program = "Program" ident { Decl | Procedure }
          "Begin" StatementList "End" ident "." .
```

Predefined Procedures

```
Procedure void writeint(int i);       /* write i to stdout */
Procedure void writebool(bool b);     /* write b ("True" or "False") to stdout */
Procedure void writeln();             /* write a newline character to stdout  */
Procedure int readint();              /* read an integer value from stdin;
                                         characters following the number up
                                         to newline are ignored. */
```