Bonk Language Grammar

Keywords: flot, nubr, strg, many
Operators: @, +, -, *, /, +=, -=, *=, /=, =, ==, <, >, <=, >=, !=, blok, help, brek, hive, bowl, bonk, loop, of, and, or, not

Grammar:

```
/* Overall program structure */

- Program : HelpStatement* Declaration*
- HelpStatement : help Identifier
- Declaration : BlokDeclaration | VariableDeclaration | HiveDeclaration
- BlokDeclaration : blok Identifier ParameterListDeclaration? ReturnType? (CodeBlock | ;)
- ReturnType : : Type
- VariableDeclaration : bowl Identifier (: Type)? (= Expression)?
- ParameterListDeclaration: [(VariableDeclaration (, VariableDeclaration)*)?]
- CodeBlock: { (CodeBlock | LoopStatement | Statement;)* }
- Statement: { Expression | BonkStatement | BrekStatement | VariableDeclaration }
- BonkStatement: bonk Expression?
- BrekStatement: brek
- ArrayConstant: [Expression*]
- HiveAccess: Identifier of ExpressionPrimary
- ParameterList: [(ParameterListItem (, ParameterListItem)*)?]
- ParameterListItem: Identifier = Expression
- LoopStatement: loop ParameterListDeclaration? CodeBlock
- Type: many Type | TrivialType | Identifier
- HiveDeclaration: hive Identifier { (BlokDeclaration | VariableDeclaration;)* }

/* All kinds of expressions, operator precedence */

- ROperatorExpression<NextExpression, Operator>:
    NextExpression |
    NextExpression Operator ROperatorExpression<NextExpression, Operator>
- LOperatorExpression<NextExpression, Operator>:
    NextExpression |
    LOperatorExpression<NextExpression, Operator> Operator NextExpression

- Expression: ExpressionAssignment
- ExpressionAssignment: ROperatorExpression<ExpressionOr,         (= | += | -= | *= | /=)>
- ExpressionOr:         LOperatorExpression<ExpressionAnd,        or>
- ExpressionAnd:        LOperatorExpression<ExpressionEquality,   and>
- ExpressionEquality:   LOperatorExpression<ExpressionRelational, (== | !=)>
- ExpressionRelational: LOperatorExpression<ExpressionAdd,        (< | > | <= | >=)>
- ExpressionAdd:        LOperatorExpression<ExpressionMul,        (+ | -)>
- ExpressionMul:        LOperatorExpression<ExpressionUnary,      (* | /)>
- ExpressionUnary:
    ExpressionPrimary |
    ExpressionCall |
    UnaryOperator ExpressionUnary
- ExpressionPrimary: HiveAccess | Identifier | NumberConstant | StringConstant | ArrayConstant | (Expression) | CodeBlock
- ExpressionCall: @ Expression ArgumentList?
```

Example program

```
hive Persn {
	bowl hapiness: flot;

	blok hapy_birday[bowl how_hapy: nubr] {

		hapiness of me += how_hapy;

		bowl messag: strg;

		dogo: There are no if-s, instead one should
		dogo: use lazy expression calculation, like so:

		how_hapy > 0
			and messag = "I am so happy it's my birday!"
			or messag = "I'm getting closer and closer to death :c";


		bowl messags: many strg = ["It's my birday!"];

		loop[bowl counter = 0] {
			counter += 1 < how_hapy of me or brek;
			messags += text;
		}

		loop[bowl counter = 0] {
			counter += 1 < length of messags or brek;

			@print[text = @value of messags[index = counter]];
		}
	}
}

blok main {
	bowl you = @Persn[hapiness = 5];
	bowl he = @Persn[hapiness = 6];

	@hapy_birday of you[how_hapy = 10];

	hapiness of he = hapiness of you;

	bonk hapiness of you;
}
```