BNF for MyLang

**Programs**

<*program*> ::= **program** <*identifier*>**()** {<*compound* *declaration*>} **is** {<*variable declaration*>} **begin** <*block*> **end**

**Declarations**

<*compound declaration*> ::= <*function definition*> | <*class declaration*> | <*array declaration*>

<*function definition*> ::= **function** <*identifier*>**(**[<*parameter list*>]**)** {<*variable declaration*>} [**return** <*type*>**;**] **is** {<*variable declaration*>} **begin** <*block*> **end** **function** <*identifier*>**;**

<*parameter list*> ::= <*identifier*> | <*parameter list*> **,** <*identifier*>

<*variable declaration*> ::= **var** <*identifier*> **is** <*type>***;**

<*class declaration*> ::= **type** <*identifier*> [**extends** <*type*>] **is** **class** <*class body>* **end class;**

<*class body*> ::= {<*variable declaration*> | <*function definition*>}

<*array declaration*> ::= **type** <*identifier*> **is array of** <*expression*> <*type*>**;**

**Blocks and Commands**

<*block*> ::= {<*statement*>}

<*statement*> ::= <*if then statement*> | <*if then else statement*> | <*while statement*> | <*repeat statement*> | <*foreach statement*> | <*continue statement*> | <*break statement*> | <*return statement*> | <*print expression*> | <*expression statement*> | <*empty statement*>

<*if then statement*> ::= **if** <*expression*> **then** <*block*> **end if**

<*if then else statement*> ::= **if** <*expression*> **then** <*block*> {<*elif list*>} **else** <*block*> **end if**

<*elif list*> ::= **elif** <*expression*> **then** <*block*>

<*while statement*> ::= **while** <*expression*> **do** <*block*> **end while**

<*repeat statement*> ::= **repeat** <*block*> **until** <*expression*>**;**

<*foreach statement*> ::= **foreach** <*identifier*> **in** <*expression*> **do** <*block*> **end foreach**

<*continue statement*> ::= **continue;**

<*break statement*> ::= **break;**

<*return statement*> ::= **return** [<*expression*>]**;**

<*empty statement*> ::= **;**

<*print statement*> ::= **print** <*expression list*>**;**

<*expression statement*> ::= <*expression*>**;**

**Expressions**

<*expression list*> ::= <*expression*> | <*expression list*> **,** <*expression*>

<*expression*> ::= <*logical or expression*> | <*expression*> **:=** <*expression*>

<*logical or expression*> ::= <*logical and expression*> | <*logical or expression*> **or** <*logical and expression*>

<*logical and expression*> ::= <*inclusive or expression*> | <*logical and expression*> **and** <*inclusive or expression*>

<*inclusive or expression*> ::= <*exclusive or expression*> | <*inclusive or expression*> **|** <*exclusive or expression*>

<*exclusive or expression*> ::= <*and expression*> | <*exclusive or expression*> **^** <*and expression*>

<*and expression*> ::= <*equality expression*> | <*and expression*> **&** <*equality expression*>

<*equality expression*> ::= <*relational expression*> | <*equality expression*> **==** <*relational expression*> | <*equality expression*> **!=** <*relational expression*>

<*relational expression*> ::= <*shift expression*>

| <*relational expression*> **<** <*shift expression*>

| <*relational expression*> **>** <*shift expression*>

| <*relational expression*> **<=** <*shift expression>*

| <*relational expression*> **>=** <*shift expression*>

<*shift expression*> ::= <*additive expression*> | <*shift expression*> **<<** <*additive expression*> | <*shift expression*> **>>** <*additive expression*>

<*additive expression*> ::= <*multiplicative expression*> | <*additive expression*> **+** <*multiplicative expression*> | <*additive expression*> **-** <*multiplicative expression*>

<*multiplicative expression*> ::= <*primary expression*>

| <*multiplicative expression*> **\*** <*primary expression*>

| <*multiplicative expression*> **/** <*primary expression*>

| <*multiplicative expression*> **%** <*primary expression*>

<*primary expression*> ::= <*literal*> | **this** | <*identifier*> | **(**<*expression*>**)** | <*filed access*> | <*method invocation*> | <*array access*>

<*filed access*> ::= <*primary expression*> **.** <*identifier*>

<*method invocation*> ::= <*identifier*> **(**[<*expression list*>]**)** | <*field access*> **(**[<*expression list*>]**)**

<*array access*> ::= <*primary expression*> **[** <*expression*> **]**

**Types**

<*type*> ::= **integer** | **boolean** | <*identifier*>

**Tokens**

<*literal*> ::= <*integer literal*> | <*boolean literal*> | <*string literal*>

<*integer literal*> ::= **0** | <*non zero digit*> <*digits*>

<*digits*> ::= {<*digit*>}

<*digit*> ::= **0** | <*non zero digit*>

<*non zero digit*> ::= **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<*boolean literal*> ::= **yes** | **no**

<*string literal*> ::= **"**{<*input character*>}**"**

<*keyword*> ::= **and | array of | begin | boolean | break | class | continue | do | elif | else | end | extends | foreach | function | if | in | integer | is | no | or | print | program | repeat | return | then | this | type | until | var | while | yes**

P.S.

<*input character*> stands for an 8-bit narrow character, a wide character or unicode character such as "龍" would be considered as a sequence of 8-bit characters, therefore would be recognized as a sequence of <*input character*>s.

<*identifier*> indicates a string that must start with a letter, including underscore(**\_**) and dollar sign(**$**), followed by any number of letters and digits, and must not be one of the keywords given above.