T-603-THYD Fall 2017, PROJECT – part 2 (updated grammar)

Marked as orange are grammar symbols that were intentionally not left-factored as they are "harmless", that is, a recursive-descent LL(1) parser should not have any problems parsing them. Leaving the grammar like that will make it easier to build an efficient parser, for exmaple, in some cases, allow you to implement right-recursion (tail-recursion) as iteration.

program ::= **class id {** *variable_declarations method_declarations* **}**

variable_declarations ::= *type variable_list* **;** *variable_declarations* | ϵ

type ::= **int** | **real**

variable_list ::= *variable* | *variable* **,** *variable_list*

variable ::= **id**

method_declarations ::= *method_declaration* *method_declarations* | *method_declaration*

method_declaration ::= **static** *method_return_type* **id (** parameters **)**
{ *variable_declarations* *statement_list* **}**

method_return_type ::= *type* | **void**

parameters ::= *parameter_list* | ϵ

parameter_list ::= *type* *variable* | type *variable* , *parameter_list*

statement_list ::= *statement* *statement_list* | ϵ

statement ::= **if (** *expr* **)** *statement_block* *optional_else*
| **for (** *variable* **=** *expr* **;** *expr* **;** *variable op_incr_decr* **)** *statement_block*
| **return** *optional_expr* **;**
| **break ;**
| **continue ;**
| *statement_block*
| *id_start_stm*

id_start_stm ::= **id (** *expr_list* **) ;**
| **id** *op_incr_decr* **;**
| **id = ** *expr* **;**

optional_expr ::= *expr* | ϵ

statement_block ::= **{** *statement_list* **}**

optional_else ::= **else** *statement_block* | ϵ

expr_list ::= *expr* *more_expr* | ϵ

more_expr ::= **,** *expr more_expr* | ∈

expr ::= *expr_and expr'*
expr' ::= **||** *expr_and expr'* | ∈

expr_and ::= *expr_eq expr_and'*
expr_and' ::= **&&** *expr_eq expr_and'* | ∈

expr_eq ::= *expr_rel expr_eq'*
expr_eq' ::= *op_eq expr_rel expr_eq'* | ∈

expr_rel ::= *expr_add expr_rel'*
expr_rel' ::= *op_rel expr_add expr_rel'* | ∈

expr_add ::= *expr_mult expr_add'*
expr_add' ::= *op_add expr_mult expr_add'* | ∈

expr_mult ::= *expr_unary expr_mult'*
expr_mult' ::= *op_mult expr_unary expr_mult'* | ∈

expr_unary ::= *op_unary expr_unary* | *factor*

factor ::= **num** | **(** *expr* **)** | **id** | **id (** *expr_list* **)**

op_eq ::= **==** | **!=**
op_rel ::= **<** | **<=** | **>** | **>=**
op_add ::= **+** | **-**
op_mul ::= ***** | **/** | **%**
op_unary ::= **+** | **-** | **!**
op_incr_decr ::= **--** | **++**

Note, some of the **id** *terminal symbols above are referring to a variable, and should result in a VariableExprNode node to be generated. However, for ease of parsing, it is better in those situations to create the VariableExprNode explicitly without calling the variable method (this is because, to know that you were expecting a variable, you will already have had to match the* **id** *token).*