```
class Program{
	int var, integerId = 4;

	static void _main(){
		var = -23 - 22;
		print("N
		Sanity.");
		for(integerId in 0 to 100 steps 10){
			;
		}
	}

	// Comment.
	bool boolean(){
		real var_2 = 0.14;

		if(var < 8 && 32 - var > var_2){
			var = var | (integerId - 2);
			var = var & 0b10010010;
		}else if(Ayyye :D){
			while(true){
				var = 3;
				break;
			}
			return true;
		}
		return false;
	}

	real _func(int i, int j){
		return i % j * 3.52;
	}
}
```

Symbol Table

Lexeme	Token
0	Program
1	var
2	integerId
3	_main
4	Boolean
5	var_2
6	_func
7	i
8	j

```
macros -> /* Lambda */
classes -> /* Lambda */
symbol_decs -> /* Lambda */
return_type -> INT_TYPE
var_type -> return_type
var_list_item -> ID
var_list -> var_list_item
exp -> INTEGER
var_list_item -> ID = exp
var_list -> var_list , var_list_item
var_dec -> var_type var_list ;
symbol_dec -> var_dec
symbol_decs -> symbol_decs symbol_dec
formal_arguments -> /* Lambda */
statements_list -> /* Lambda */
lvalue -> ID
exp -> INTEGER
unary_operation -> - exp
exp -> unary_operation
exp -> INTEGER
binary_operation -> exp - exp
exp -> binary_operation
assignment -> lvalue = exp ;
statement -> assignment
statements_list -> statements_list statement
print -> PRINT ( STRING )
statement -> print
statements_list -> statements_list statement
statement -> ;
statements_list -> statements_list statement
exp -> INTEGER
exp -> INTEGER
exp -> INTEGER
statements_list -> /* Lambda */
statement -> ;
statements_list -> statements_list statement
block -> { statements_list }
for -> FOR ( ID IN exp TO exp STEPS exp ) block
statement -> for
statements_list -> statements_list statement
block -> { statements_list }
func_body -> ID ( formal_arguments ) block
func_dec -> STATIC VOID func_body
symbol_dec -> func_dec
```

symbol_decs -> symbol_decs symbol_dec
return_type -> BOOL_TYPE
var_type -> return_type
formal_arguments -> /* Lambda */
statements_list -> /* Lambda */
return_type -> REAL_TYPE
exp -> REAL
var_list_item -> ID = exp
var_list -> var_list_item
statement_var_dec -> return_type var_list ;
statement -> statement_var_dec
statements_list -> statements_list statement
lvalue -> ID
exp -> lvalue
exp -> INTEGER
comparison_operation -> exp < exp
exp -> comparison_operation
exp -> INTEGER
lvalue -> ID
exp -> lvalue
binary_operation -> exp - exp
exp -> binary_operation
lvalue -> ID
exp -> lvalue
comparison_operation -> exp > exp
exp -> comparison_operation
logical_operation -> exp && exp
exp -> logical_operation
statements_list -> /* Lambda */
lvalue -> ID
lvalue -> ID
exp -> lvalue
lvalue -> ID
exp -> lvalue
exp -> INTEGER
binary_operation -> exp - exp
exp -> binary_operation
exp -> (exp)
bitwise_operation -> exp | exp
exp -> bitwise_operation
assignment -> lvalue = exp ;
statement -> assignment
statements_list -> statements_list statement
lvalue -> ID

```
lvalue -> ID
exp -> lvalue
exp -> INTEGER
bitwise_operation -> exp & exp
exp -> bitwise_operation
assignment -> lvalue = exp ;
statement -> assignment
statements_list -> statements_list statement
block -> { statements_list }
lvalue -> ID
exp -> lvalue
statements_list -> /* Lambda */
exp -> TRUE
statements_list -> /* Lambda */
lvalue -> ID
exp -> INTEGER
assignment -> lvalue = exp ;
statement -> assignment
statements_list -> statements_list statement
break -> BREAK ;
statement -> break
statements_list -> statements_list statement
block -> { statements_list }
while -> WHILE ( exp ) block
statement -> while
statements_list -> statements_list statement
exp -> TRUE
return -> RETURN exp ;
statement -> return
statements_list -> statements_list statement
block -> { statements_list }
if -> if ( exp ) block
statement -> if
block -> statement
else_block -> ELSE block
if -> if ( exp ) block else_block
statement -> if
statements_list -> statements_list statement
exp -> FALSE
return -> RETURN exp ;
statement -> return
statements_list -> statements_list statement
block -> { statements_list }
func_body -> ID ( formal_arguments ) block
```

func_dec -> var_type func_body
symbol_dec -> func_dec
symbol_decs -> symbol_decs symbol_dec
return_type -> REAL_TYPE
var_type -> return_type
return_type -> INT_TYPE
formal_argument -> return_type ID
formal_arguments_list -> formal_argument
return_type -> INT_TYPE
formal_argument -> return_type ID
formal_arguments_list -> formal_arguments_list , formal_argument
formal_arguments -> formal_arguments_list
statements_list -> /* Lambda */
lvalue -> ID
exp -> lvalue
lvalue -> ID
exp -> lvalue
binary_operation -> exp 1.812128e-307xp
exp -> binary_operation
exp -> REAL
binary_operation -> exp * exp
exp -> binary_operation
return -> RETURN exp ;
statement -> return
statements_list -> statements_list statement
block -> { statements_list }
func_body -> ID (formal_arguments) block
func_dec -> var_type func_body
symbol_dec -> func_dec
symbol_decs -> symbol_decs symbol_dec
class -> CLASS ID { dymbol_decs }
classes -> classes class
program -> macros classes