NIMBUS

SER502 - Spring ’23

Team 14

Team

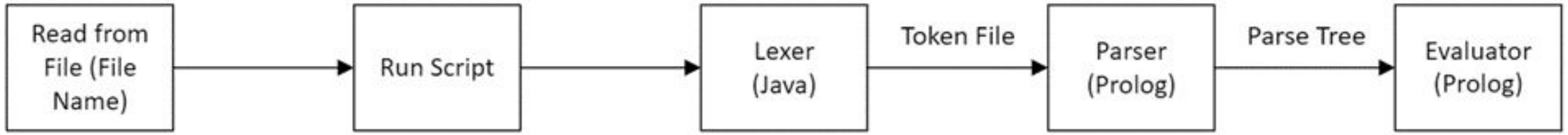
- Shanmugapriyan Ravichanthiran
- Prasant Ganesan
- Poornima Sathya Keerthi
- Joseph Thomas

Agenda

Introduction

Nimbus programming language is named after “Nimbus 2000”, the broomstick used by Harry Potter. At the time of its release in, it was the fastest broomstick in production. Thus, our programming language Nimbus, also focuses on speed.

Design


Read from File (File Name)
Run Script
Lexer (Java)
Token File
Parser (Prolog)
Parse Tree
Evaluator (Prolog)

Tool Used

ANTLR (Another Tool for Language Recognition) We have used ANTLR to write the Lexer for our language because it is an efficient tool to write well-optimized lexers and parsers. It supports our chosen language Java, and also offers a rich set of features. If we were to write the Lexer on our own, it would be a complicated process of generating tokens by splitting it and serializing those tokens.

Language Features

- Extension “**.nimb**”, example, filename.nimb
- Supports assignment operator (:=)
- Relational operators (=, <, >, <=, >=, !=)
- Logical operators (and, or, not)
- Supports control flow statements (if, if-else, for-loop, for-in-range, whi
- Ternary operator
- Supports operator precedences

Implementation

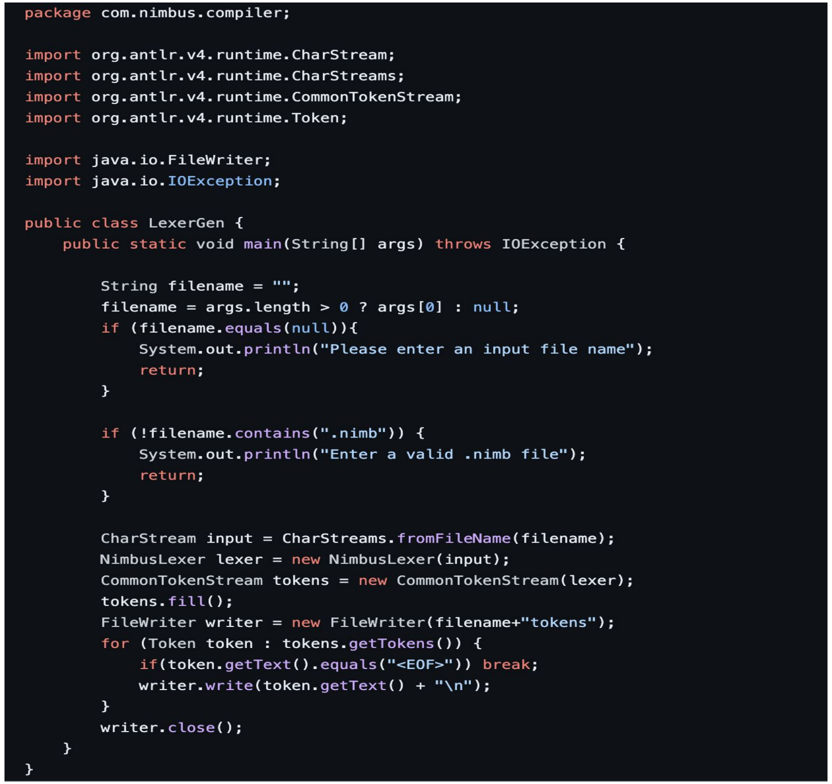
- Grammar
- Lexer
- Parser
- Evaluator

Grammar

```
program --> block.
block --> ['main'], ['('], [')'], ['{'], decl_block, command, ['}'].

decl_block --> decl.
decl_block --> decl, decl_block.


% Predicate to parse the block.
general_block --> ['{'], command, ['}'].

% Declarations (initialization).
decl -->  ['int'], identifier, [':='], integer, [';'].
decl -->  ['float'], identifier, [':='], float, [';'].
decl -->  ['string'], identifier, [':='], string, [';'].
decl -->  ['boolean'], identifier, [':='], boolean, [';'].

decl -->  ['int'], identifier, [';'].
decl -->  ['float'], identifier, [';'].

% Commands.
command --> commandblock, command.
command --> commandblock.

% Command block (loops, assignment, print commands).
if_statement --> ['if'], ['('], boolean,  [')'], general_block.
if_else_statement --> ['if'], ['('], boolean,  [')'], general_block, ['else'], general_block.
while_statement --> ['while'], ['('], boolean,  [')'], general_block.
for_loop --> ['for'], ['('], ['int'] ,identifier ,[':='] ,integer, [';'], boolean, [';'], unaryexpr, [')'], general_block.
for_in_range_loop --> ['for'], identifier, ['in'], ['range'], ['('], integer, [','], integer, [')'], general_block.
print_statement --> ['print'], ['('], identifier, [')'], [';'].
declare_in_block --> identifier, [':='], expr, [';'].
```

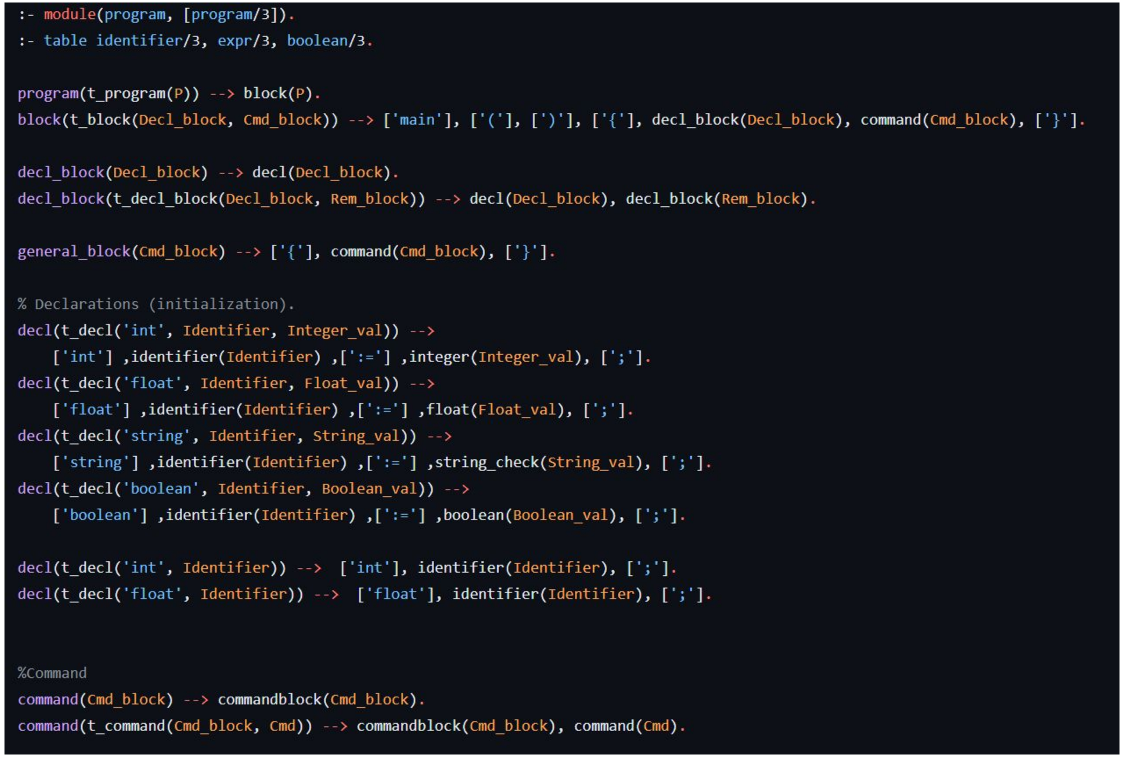
```
commandblock --> if_statement | if_else_statement | while_statement | for_loop |
                 for_in_range_loop | print_statement | declare_in_block | general_block.
commandblock --> if_statement, command | if_else_statement, command | while_statement, command |
                 for_loop, command | for_in_range_loop, command | print_statement, command |
                 declare_in_block, command | general_block, command.

% Increment decrement operators.
unaryexpr --> identifier, ['++'].
unaryexpr --> identifier, ['--'].

ternaryexpr --> ['('], boolean, [')'], ['?'], expr, [':'], expr.

% Expression for arithmetic computations.
expr --> factor.
expr --> miniexpr.
expr --> miniexpr, ['+'], expr.
expr --> factor, ['+'], miniexpr.
expr --> miniexpr, ['-'], expr.
expr --> factor, ['-'], miniexpr.
expr --> ternaryexpr.

% To incorporate precedence rules.
miniexpr --> factor.
miniexpr --> factor, ['*'], miniexpr.
miniexpr --> factor, ['/'], miniexpr.

% Lowest factor in th expressions.

factor --> ['('], expr, [')'].
factor --> identifier.
factor --> integer.
```

Lexer

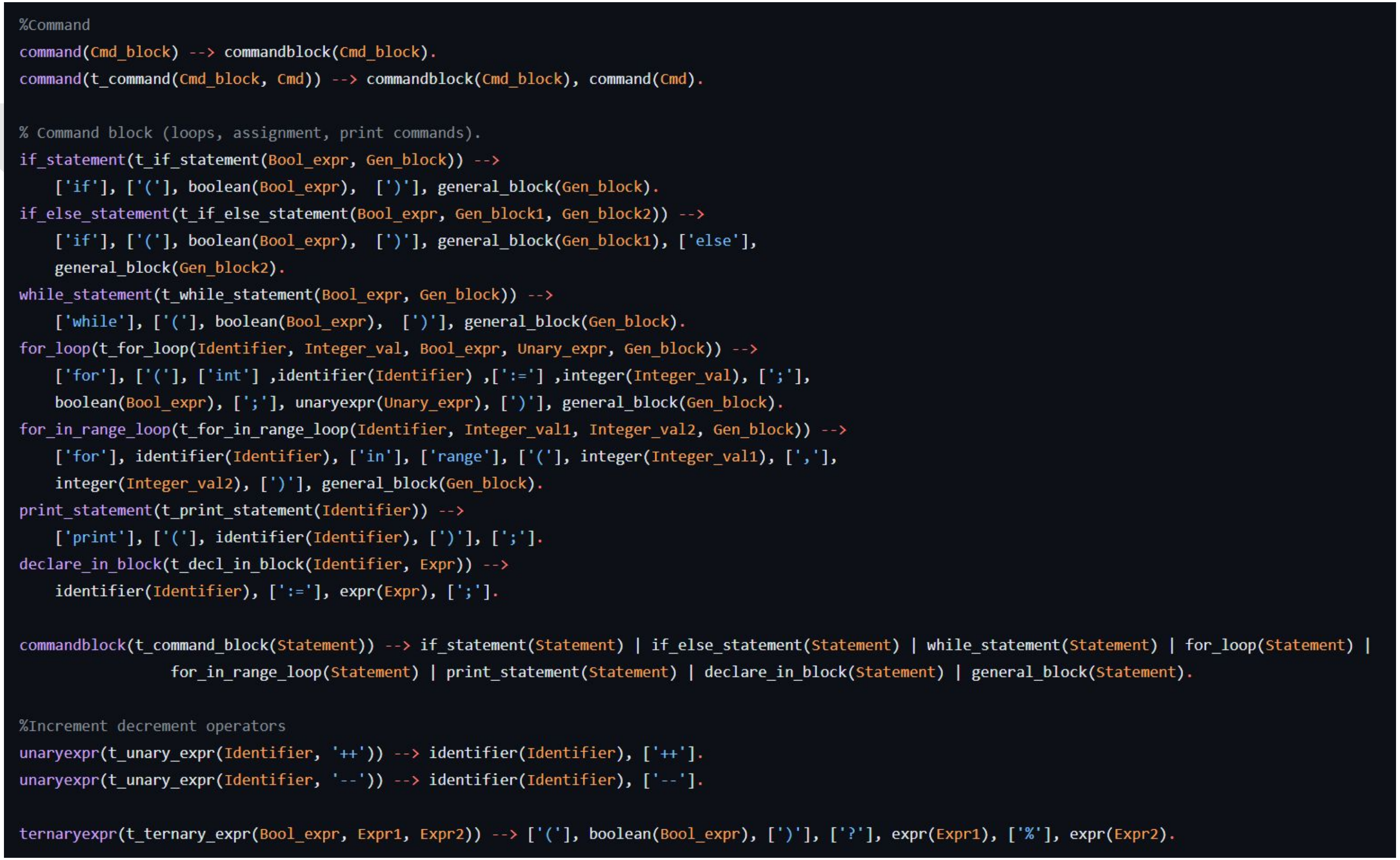


```java
package com.nimbus.compiler;

import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.Token;

import java.io.FileWriter;
import java.io.IOException;

public class LexerGen {
    public static void main(String[] args) throws IOException {

        String filename = "";
        filename = args.length > 0 ? args[0] : null;
        if (filename.equals(null)){
            System.out.println("Please enter an input file name");
            return;
        }

        if (!filename.contains(".nimb")) {
            System.out.println("Enter a valid .nimb file");
            return;
        }

        CharStream input = CharStreams.fromFileName(filename);
        NimbusLexer lexer = new NimbusLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        tokens.fill();
        FileWriter writer = new FileWriter(filename+"tokens");
        for (Token token : tokens.getTokens()) {
            if(token.getText().equals("<EOF>")) break;
            writer.write(token.getText() + "\n");
        }
        writer.close();
    }
}
```

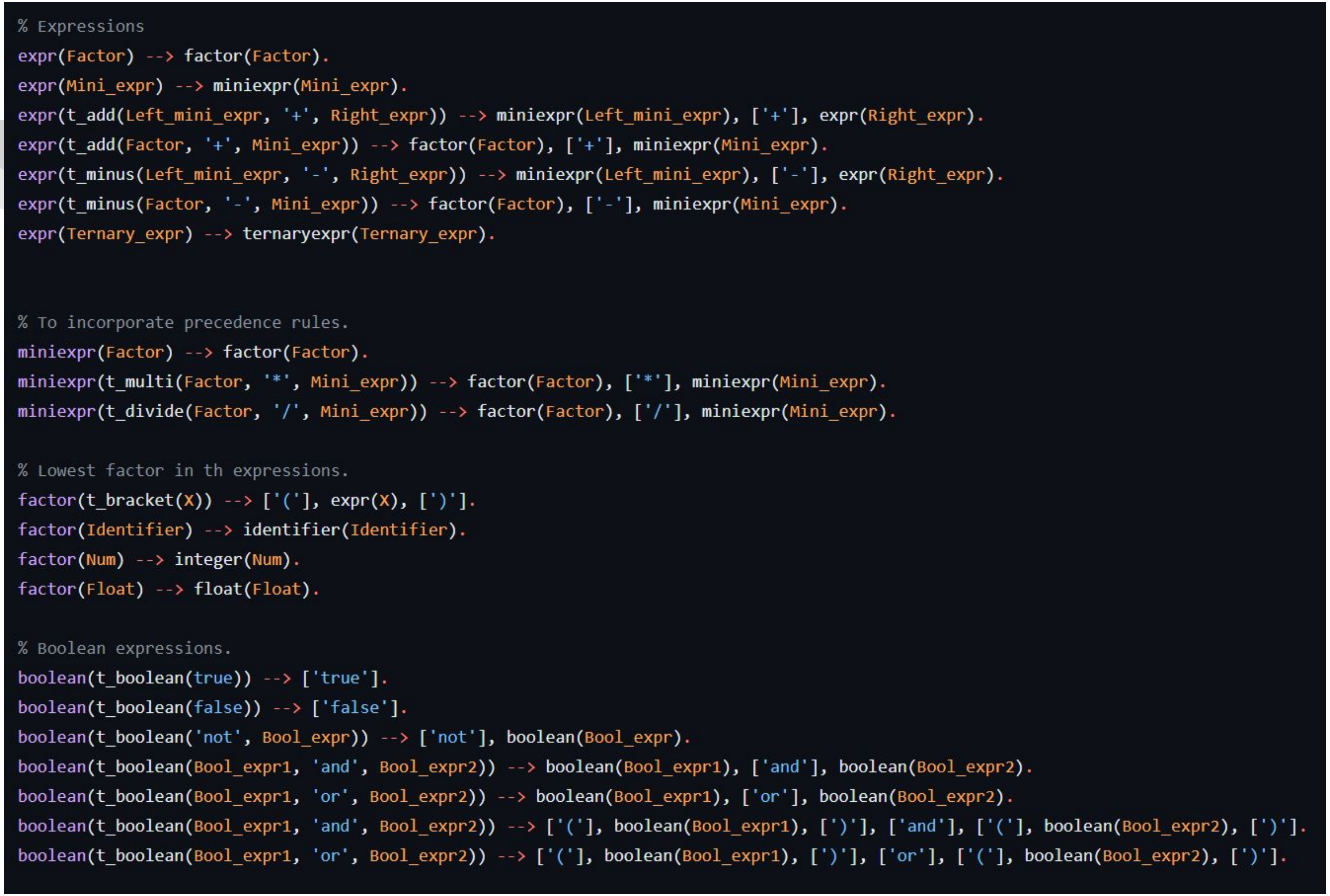
Parser



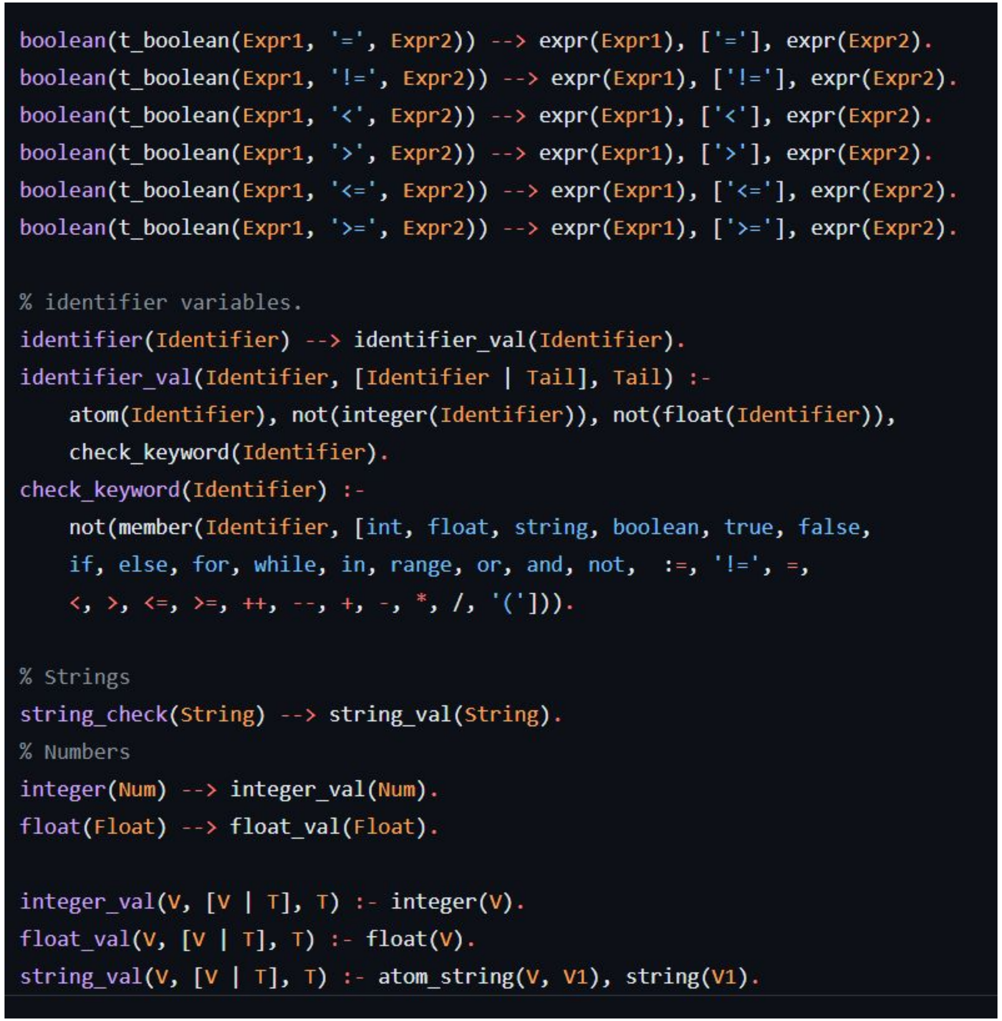
```prolog
:- module(program, [program/3]).
:- table identifier/3, expr/3, boolean/3.

program(t_program(P)) --> block(P).
block(t_block(Decl_block, Cmd_block)) --> ['main'], ['('], [')'], ['{'], decl_block(Decl_block), command(Cmd_block), ['}'].

decl_block(Decl_block) --> decl(Decl_block).
decl_block(t_decl_block(Decl_block, Rem_block)) --> decl(Decl_block), decl_block(Rem_block).

general_block(Cmd_block) --> ['{'], command(Cmd_block), ['}'].

% Declarations (initialization).
decl(t_decl('int', Identifier, Integer_val)) -->
    ['int'] ,identifier(Identifier) ,[':='] ,integer(Integer_val), [';'].
decl(t_decl('float', Identifier, Float_val)) -->
    ['float'] ,identifier(Identifier) ,[':='] ,float(Float_val), [';'].
decl(t_decl('string', Identifier, String_val)) -->
    ['string'] ,identifier(Identifier) ,[':='] ,string_check(String_val), [';'].
decl(t_decl('boolean', Identifier, Boolean_val)) -->
    ['boolean'] ,identifier(Identifier) ,[':='] ,boolean(Boolean_val), [';'].

decl(t_decl('int', Identifier)) -->  ['int'], identifier(Identifier), [';'].
decl(t_decl('float', Identifier)) -->  ['float'], identifier(Identifier), [';'].


%Command
command(Cmd_block) --> commandblock(Cmd_block).
command(t_command(Cmd_block, Cmd)) --> commandblock(Cmd_block), command(Cmd).
```



```
%Command
command(Cmd_block) --> commandblock(Cmd_block).
command(t_command(Cmd_block, Cmd)) --> commandblock(Cmd_block), command(Cmd).

% Command block (loops, assignment, print commands).
if_statement(t_if_statement(Bool_expr, Gen_block)) -->
    ['if'], ['('], boolean(Bool_expr),  [')'], general_block(Gen_block).
if_else_statement(t_if_else_statement(Bool_expr, Gen_block1, Gen_block2)) -->
    ['if'], ['('], boolean(Bool_expr),  [')'], general_block(Gen_block1), ['else'],
    general_block(Gen_block2).
while_statement(t_while_statement(Bool_expr, Gen_block)) -->
    ['while'], ['('], boolean(Bool_expr),  [')'], general_block(Gen_block).
for_loop(t_for_loop(Identifier, Integer_val, Bool_expr, Unary_expr, Gen_block)) -->
    ['for'], ['('], ['int'] ,identifier(Identifier) ,[':='] ,integer(Integer_val), [';'],
    boolean(Bool_expr), [';'], unaryexpr(Unary_expr), [')'], general_block(Gen_block).
for_in_range_loop(t_for_in_range_loop(Identifier, Integer_val1, Integer_val2, Gen_block)) -->
    ['for'], identifier(Identifier), ['in'], ['range'], ['('], integer(Integer_val1), [','],
    integer(Integer_val2), [')'], general_block(Gen_block).
print_statement(t_print_statement(Identifier)) -->
    ['print'], ['('], identifier(Identifier), [')'], [';'].
declare_in_block(t_decl_in_block(Identifier, Expr)) -->
    identifier(Identifier), [':='], expr(Expr), [';'].

commandblock(t_command_block(Statement)) --> if_statement(Statement) | if_else_statement(Statement) | while_statement(Statement) | for_loop(Statement) |
                 for_in_range_loop(Statement) | print_statement(Statement) | declare_in_block(Statement) | general_block(Statement).

%Increment decrement operators
unaryexpr(t_unary_expr(Identifier, '++')) --> identifier(Identifier), ['++'].
unaryexpr(t_unary_expr(Identifier, '--')) --> identifier(Identifier), ['--'].

ternaryexpr(t_ternary_expr(Bool_expr, Expr1, Expr2)) --> ['('], boolean(Bool_expr), [')'], ['?'], expr(Expr1), ['%'], expr(Expr2).
```

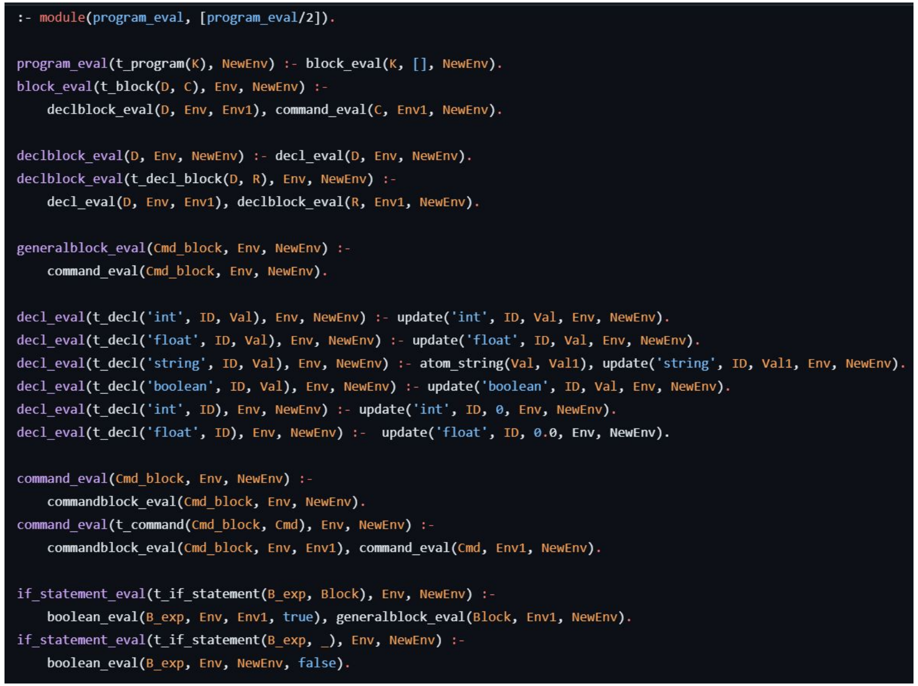


```
% Expressions
expr(Factor) --> factor(Factor).
expr(Mini_expr) --> miniexpr(Mini_expr).
expr(t_add(Left_mini_expr, '+', Right_expr)) --> miniexpr(Left_mini_expr), ['+'], expr(Right_expr).
expr(t_add(Factor, '+', Mini_expr)) --> factor(Factor), ['+'], miniexpr(Mini_expr).
expr(t_minus(Left_mini_expr, '-', Right_expr)) --> miniexpr(Left_mini_expr), ['-'], expr(Right_expr).
expr(t_minus(Factor, '-', Mini_expr)) --> factor(Factor), ['-'], miniexpr(Mini_expr).
expr(Ternary_expr) --> ternaryexpr(Ternary_expr).


% To incorporate precedence rules.
miniexpr(Factor) --> factor(Factor).
miniexpr(t_multi(Factor, '*', Mini_expr)) --> factor(Factor), ['*'], miniexpr(Mini_expr).
miniexpr(t_divide(Factor, '/', Mini_expr)) --> factor(Factor), ['/'], miniexpr(Mini_expr).

% Lowest factor in th expressions.
factor(t_bracket(X)) --> ['('], expr(X), [')'].
factor(Identifier) --> identifier(Identifier).
factor(Num) --> integer(Num).
factor(Float) --> float(Float).

% Boolean expressions.
boolean(t_boolean(true)) --> ['true'].
boolean(t_boolean(false)) --> ['false'].
boolean(t_boolean('not', Bool_expr)) --> ['not'], boolean(Bool_expr).
boolean(t_boolean(Bool_expr1, 'and', Bool_expr2)) --> boolean(Bool_expr1), ['and'], boolean(Bool_expr2).
boolean(t_boolean(Bool_expr1, 'or', Bool_expr2)) --> boolean(Bool_expr1), ['or'], boolean(Bool_expr2).
boolean(t_boolean(Bool_expr1, 'and', Bool_expr2)) --> ['('], boolean(Bool_expr1), [')'], ['and'], ['('], boolean(Bool_expr2), [')'].
boolean(t_boolean(Bool_expr1, 'or', Bool_expr2)) --> ['('], boolean(Bool_expr1), [')'], ['or'], ['('], boolean(Bool_expr2), [')'].
```

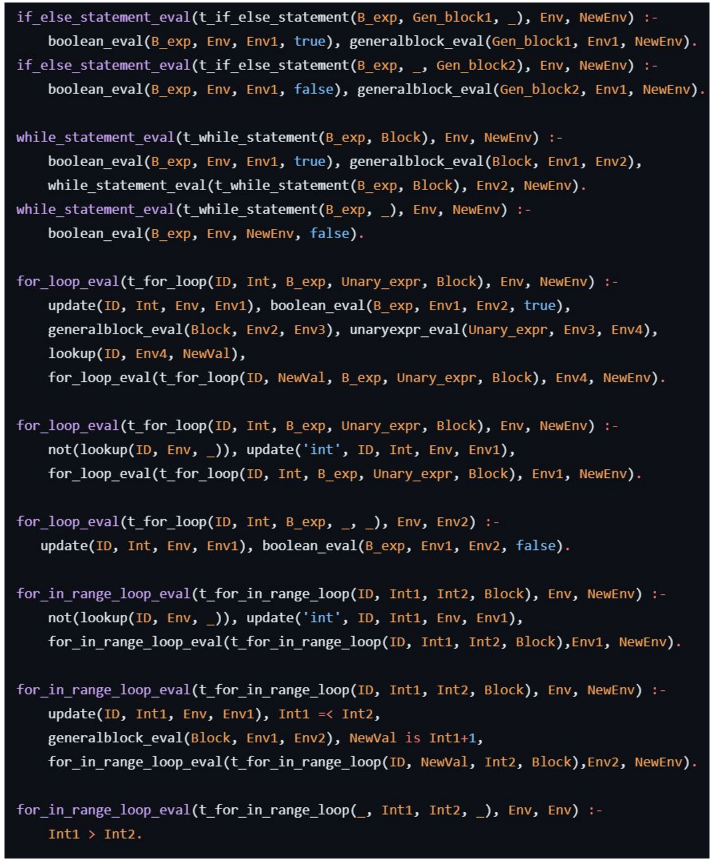


```
boolean(t_boolean(Expr1, '=', Expr2)) --> expr(Expr1), ['='], expr(Expr2).
boolean(t_boolean(Expr1, '!=', Expr2)) --> expr(Expr1), ['!='], expr(Expr2).
boolean(t_boolean(Expr1, '<', Expr2)) --> expr(Expr1), ['<'], expr(Expr2).
boolean(t_boolean(Expr1, '>', Expr2)) --> expr(Expr1), ['>'], expr(Expr2).
boolean(t_boolean(Expr1, '<=', Expr2)) --> expr(Expr1), ['<='], expr(Expr2).
boolean(t_boolean(Expr1, '>=', Expr2)) --> expr(Expr1), ['>='], expr(Expr2).

% identifier variables.
identifier(Identifier) --> identifier_val(Identifier).
identifier_val(Identifier, [Identifier | Tail], Tail) :-
    atom(Identifier), not(integer(Identifier)), not(float(Identifier)),
    check_keyword(Identifier).
check_keyword(Identifier) :-
    not(member(Identifier, [int, float, string, boolean, true, false,
    if, else, for, while, in, range, or, and, not,  :=, '!=', =,
    <, >, <=, >=, ++, --, +, -, *, /, '('])).

% Strings
string_check(String) --> string_val(String).
% Numbers
integer(Num) --> integer_val(Num).
float(Float) --> float_val(Float).

integer_val(V, [V | T], T) :- integer(V).
float_val(V, [V | T], T) :- float(V).
string_val(V, [V | T], T) :- atom_string(V, V1), string(V1).
```

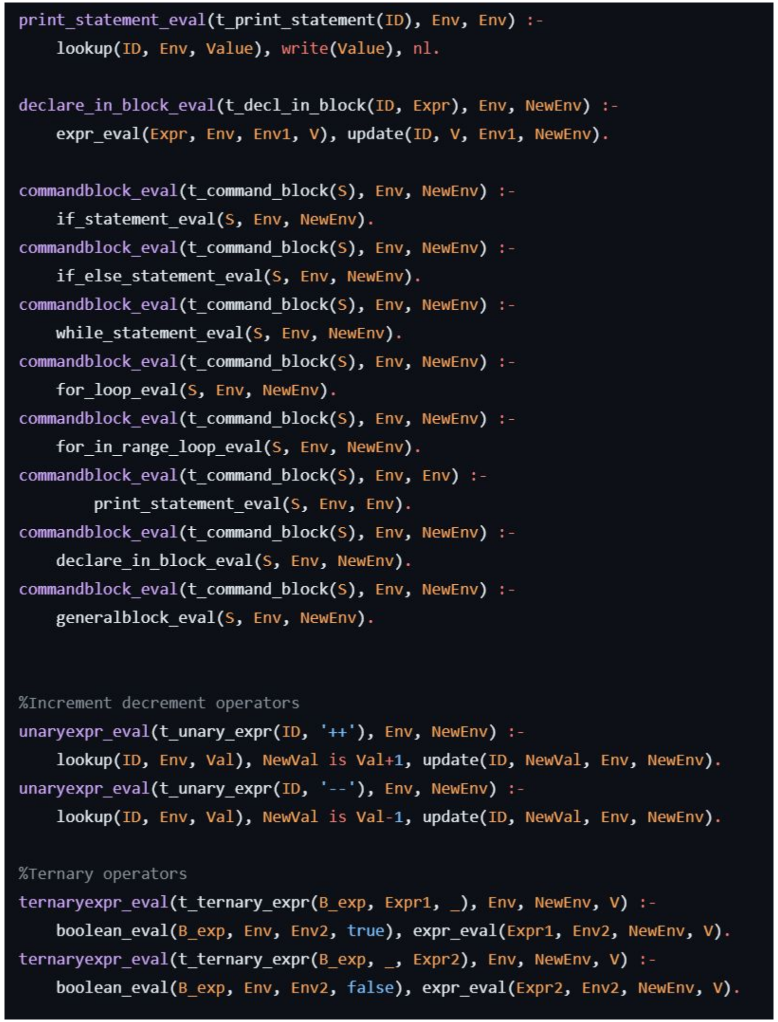
Evaluator



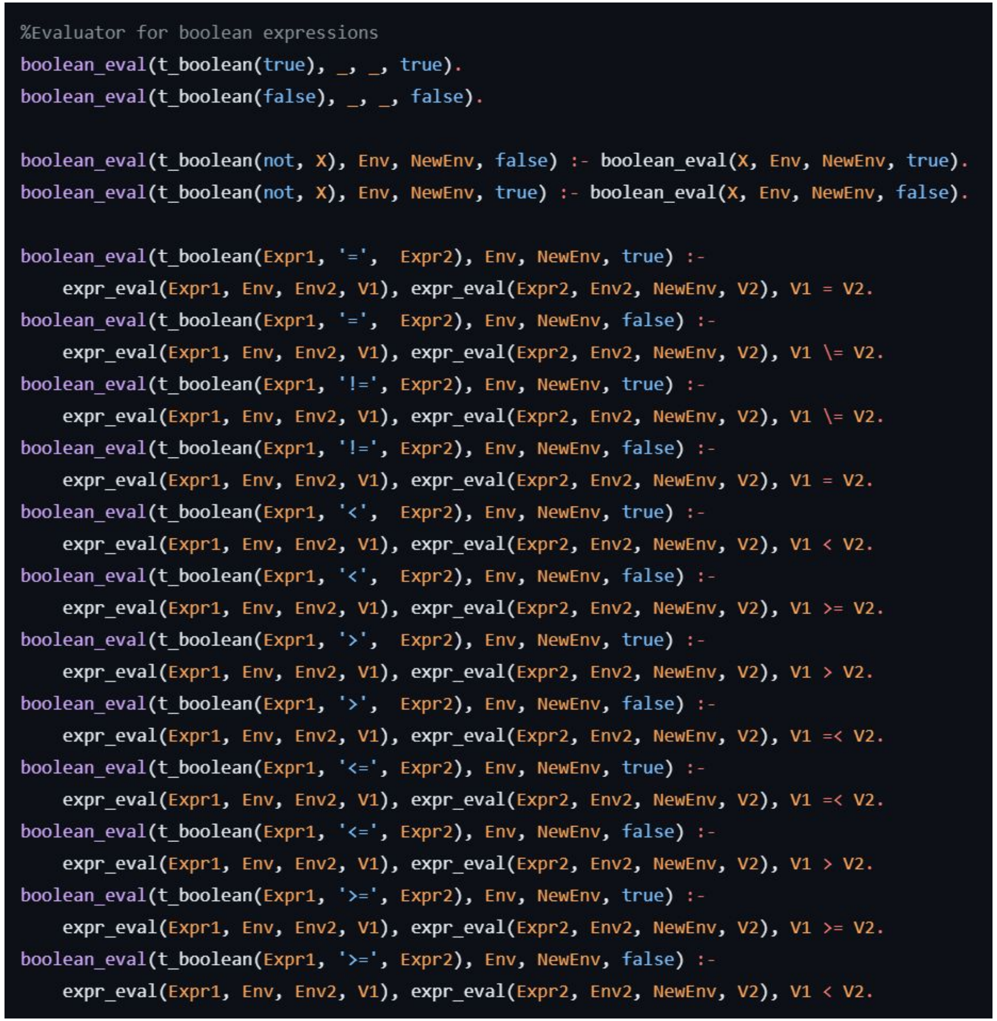
```
:- module(program_eval, [program_eval/2]).

program_eval(t_program(K), NewEnv) :- block_eval(K, [], NewEnv).
block_eval(t_block(D, C), Env, NewEnv) :-
    declblock_eval(D, Env, Env1), command_eval(C, Env1, NewEnv).

declblock_eval(D, Env, NewEnv) :- decl_eval(D, Env, NewEnv).
declblock_eval(t_decl_block(D, R), Env, NewEnv) :-
    decl_eval(D, Env, Env1), declblock_eval(R, Env1, NewEnv).

generalblock_eval(Cmd_block, Env, NewEnv) :-
    command_eval(Cmd_block, Env, NewEnv).

decl_eval(t_decl('int', ID, Val), Env, NewEnv) :- update('int', ID, Val, Env, NewEnv).
decl_eval(t_decl('float', ID, Val), Env, NewEnv) :- update('float', ID, Val, Env, NewEnv).
decl_eval(t_decl('string', ID, Val), Env, NewEnv) :- atom_string(Val, Val1), update('string', ID, Val1, Env, NewEnv).
decl_eval(t_decl('boolean', ID, Val), Env, NewEnv) :- update('boolean', ID, Val, Env, NewEnv).
decl_eval(t_decl('int', ID), Env, NewEnv) :- update('int', ID, 0, Env, NewEnv).
decl_eval(t_decl('float', ID), Env, NewEnv) :-  update('float', ID, 0.0, Env, NewEnv).

command_eval(Cmd_block, Env, NewEnv) :-
    commandblock_eval(Cmd_block, Env, NewEnv).
command_eval(t_command(Cmd_block, Cmd), Env, NewEnv) :-
    commandblock_eval(Cmd_block, Env, Env1), command_eval(Cmd, Env1, NewEnv).

if_statement_eval(t_if_statement(B_exp, Block), Env, NewEnv) :-
    boolean_eval(B_exp, Env, Env1, true), generalblock_eval(Block, Env1, NewEnv).
if_statement_eval(t_if_statement(B_exp, _), Env, NewEnv) :-
    boolean_eval(B_exp, Env, NewEnv, false).
```



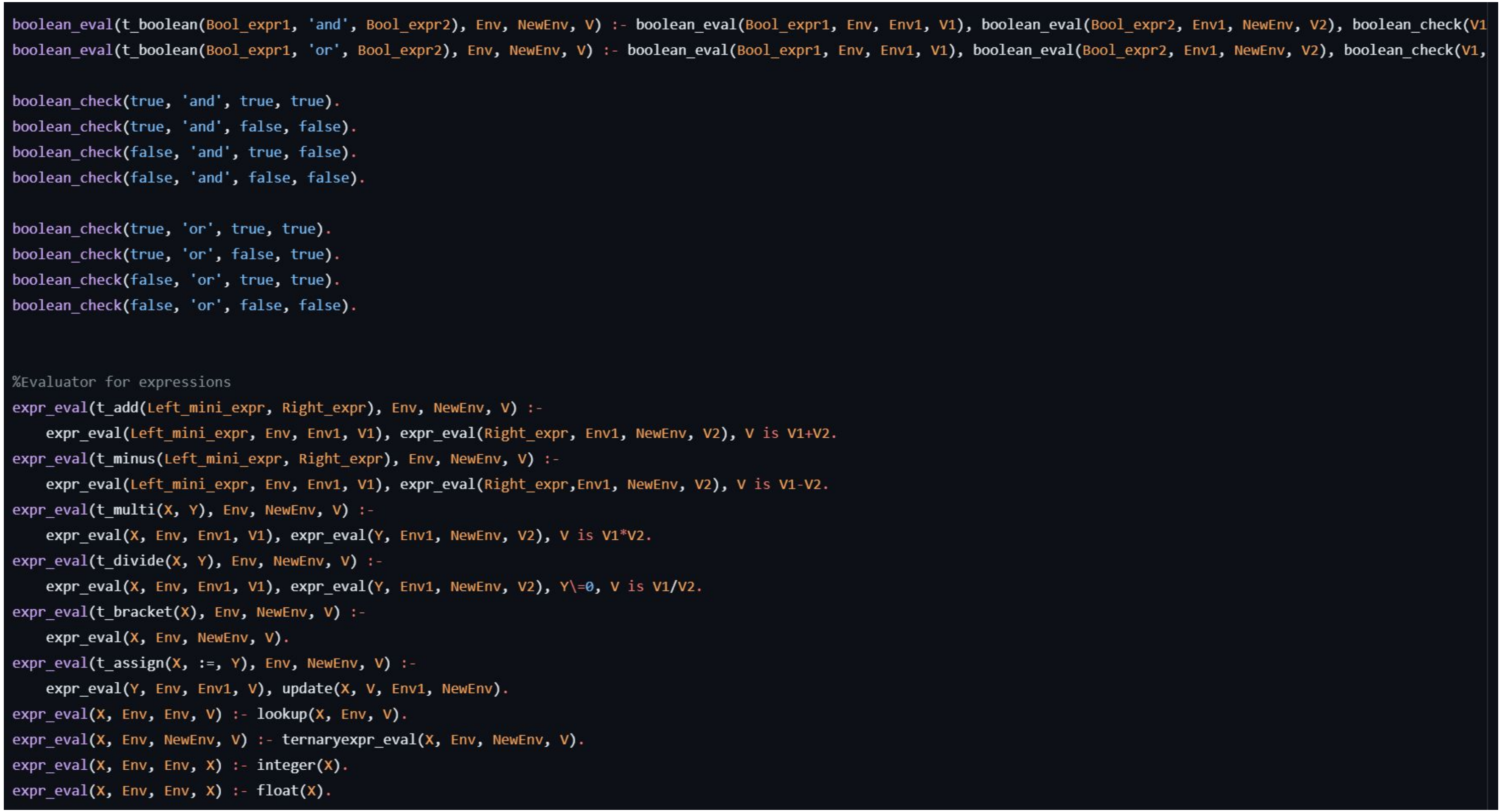
```
if_else_statement_eval(t_if_else_statement(B_exp, Gen_block1, _), Env, NewEnv) :-
    boolean_eval(B_exp, Env, Env1, true), generalblock_eval(Gen_block1, Env1, NewEnv).
if_else_statement_eval(t_if_else_statement(B_exp, _, Gen_block2), Env, NewEnv) :-
    boolean_eval(B_exp, Env, Env1, false), generalblock_eval(Gen_block2, Env1, NewEnv).

while_statement_eval(t_while_statement(B_exp, Block), Env, NewEnv) :-
    boolean_eval(B_exp, Env, Env1, true), generalblock_eval(Block, Env1, Env2),
    while_statement_eval(t_while_statement(B_exp, Block), Env2, NewEnv).
while_statement_eval(t_while_statement(B_exp, _), Env, NewEnv) :-
    boolean_eval(B_exp, Env, NewEnv, false).

for_loop_eval(t_for_loop(ID, Int, B_exp, Unary_expr, Block), Env, NewEnv) :-
    update(ID, Int, Env, Env1), boolean_eval(B_exp, Env1, Env2, true),
    generalblock_eval(Block, Env2, Env3), unaryexpr_eval(Unary_expr, Env3, Env4),
    lookup(ID, Env4, NewVal),
    for_loop_eval(t_for_loop(ID, NewVal, B_exp, Unary_expr, Block), Env4, NewEnv).

for_loop_eval(t_for_loop(ID, Int, B_exp, Unary_expr, Block), Env, NewEnv) :-
    not(lookup(ID, Env, _)), update('int', ID, Int, Env, Env1),
    for_loop_eval(t_for_loop(ID, Int, B_exp, Unary_expr, Block), Env1, NewEnv).

for_loop_eval(t_for_loop(ID, Int, B_exp, _, _), Env, Env2) :-
   update(ID, Int, Env, Env1), boolean_eval(B_exp, Env1, Env2, false).

for_in_range_loop_eval(t_for_in_range_loop(ID, Int1, Int2, Block), Env, NewEnv) :-
    not(lookup(ID, Env, _)), update('int', ID, Int1, Env, Env1),
    for_in_range_loop_eval(t_for_in_range_loop(ID, Int1, Int2, Block),Env1, NewEnv).

for_in_range_loop_eval(t_for_in_range_loop(ID, Int1, Int2, Block), Env, NewEnv) :-
    update(ID, Int1, Env, Env1), Int1 =< Int2,
    generalblock_eval(Block, Env1, Env2), NewVal is Int1+1,
    for_in_range_loop_eval(t_for_in_range_loop(ID, NewVal, Int2, Block),Env2, NewEnv).

for_in_range_loop_eval(t_for_in_range_loop(_, Int1, Int2, _), Env, Env) :-
    Int1 > Int2.
```



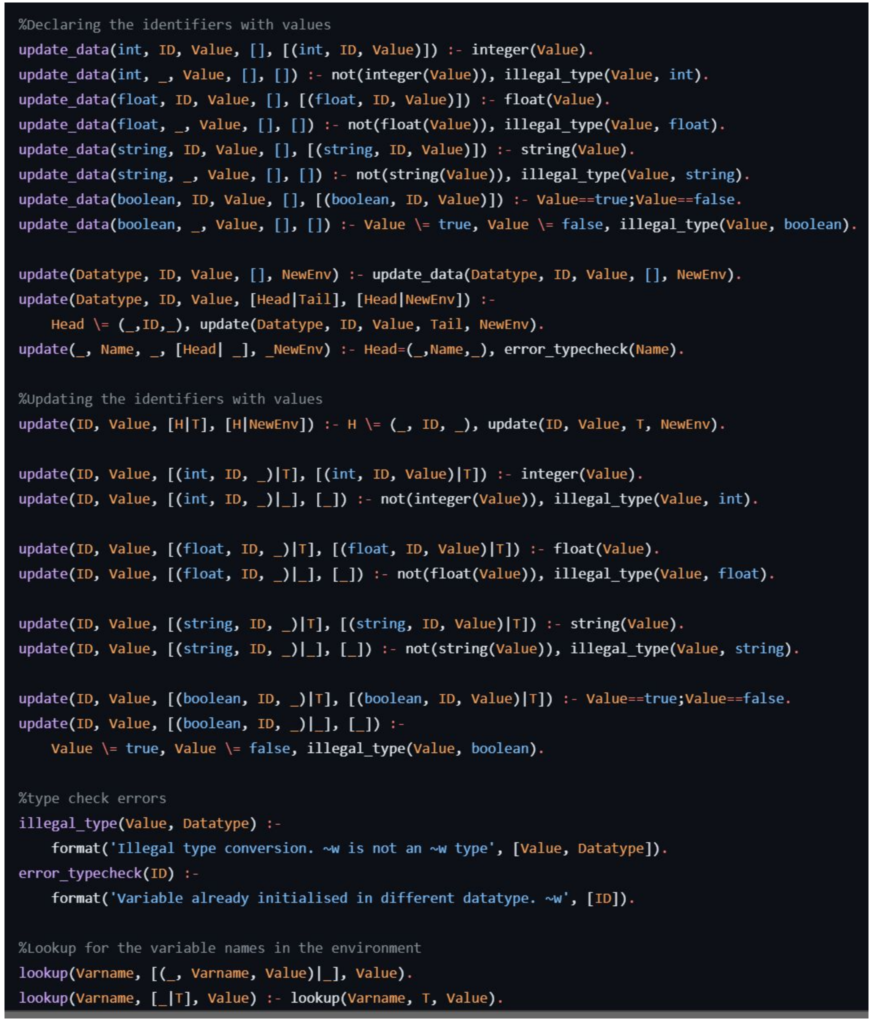
```prolog
print_statement_eval(t_print_statement(ID), Env, Env) :-
    lookup(ID, Env, Value), write(Value), nl.

declare_in_block_eval(t_decl_in_block(ID, Expr), Env, NewEnv) :-
    expr_eval(Expr, Env, Env1, V), update(ID, V, Env1, NewEnv).

commandblock_eval(t_command_block(S), Env, NewEnv) :-
    if_statement_eval(S, Env, NewEnv).
commandblock_eval(t_command_block(S), Env, NewEnv) :-
    if_else_statement_eval(S, Env, NewEnv).
commandblock_eval(t_command_block(S), Env, NewEnv) :-
    while_statement_eval(S, Env, NewEnv).
commandblock_eval(t_command_block(S), Env, NewEnv) :-
    for_loop_eval(S, Env, NewEnv).
commandblock_eval(t_command_block(S), Env, NewEnv) :-
    for_in_range_loop_eval(S, Env, NewEnv).
commandblock_eval(t_command_block(S), Env, Env) :-
        print_statement_eval(S, Env, Env).
commandblock_eval(t_command_block(S), Env, NewEnv) :-
    declare_in_block_eval(S, Env, NewEnv).
commandblock_eval(t_command_block(S), Env, NewEnv) :-
    generalblock_eval(S, Env, NewEnv).


%Increment decrement operators
unaryexpr_eval(t_unary_expr(ID, '++'), Env, NewEnv) :-
    lookup(ID, Env, Val), NewVal is Val+1, update(ID, NewVal, Env, NewEnv).
unaryexpr_eval(t_unary_expr(ID, '--'), Env, NewEnv) :-
    lookup(ID, Env, Val), NewVal is Val-1, update(ID, NewVal, Env, NewEnv).

%Ternary operators
ternaryexpr_eval(t_ternary_expr(B_exp, Expr1, _), Env, NewEnv, V) :-
    boolean_eval(B_exp, Env, Env2, true), expr_eval(Expr1, Env2, NewEnv, V).
ternaryexpr_eval(t_ternary_expr(B_exp, _, Expr2), Env, NewEnv, V) :-
    boolean_eval(B_exp, Env, Env2, false), expr_eval(Expr2, Env2, NewEnv, V).
```



```prolog
%Evaluator for boolean expressions
boolean_eval(t_boolean(true), _, _, true).
boolean_eval(t_boolean(false), _, _, false).

boolean_eval(t_boolean(not, X), Env, NewEnv, false) :- boolean_eval(X, Env, NewEnv, true).
boolean_eval(t_boolean(not, X), Env, NewEnv, true) :- boolean_eval(X, Env, NewEnv, false).

boolean_eval(t_boolean(Expr1, '=',  Expr2), Env, NewEnv, true) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 = V2.
boolean_eval(t_boolean(Expr1, '=',  Expr2), Env, NewEnv, false) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 \= V2.
boolean_eval(t_boolean(Expr1, '!=', Expr2), Env, NewEnv, true) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 \= V2.
boolean_eval(t_boolean(Expr1, '!=', Expr2), Env, NewEnv, false) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 = V2.
boolean_eval(t_boolean(Expr1, '<',  Expr2), Env, NewEnv, true) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 < V2.
boolean_eval(t_boolean(Expr1, '<',  Expr2), Env, NewEnv, false) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 >= V2.
boolean_eval(t_boolean(Expr1, '>',  Expr2), Env, NewEnv, true) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 > V2.
boolean_eval(t_boolean(Expr1, '>',  Expr2), Env, NewEnv, false) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 =< V2.
boolean_eval(t_boolean(Expr1, '<=', Expr2), Env, NewEnv, true) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 =< V2.
boolean_eval(t_boolean(Expr1, '<=', Expr2), Env, NewEnv, false) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 > V2.
boolean_eval(t_boolean(Expr1, '>=', Expr2), Env, NewEnv, true) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 >= V2.
boolean_eval(t_boolean(Expr1, '>=', Expr2), Env, NewEnv, false) :-
    expr_eval(Expr1, Env, Env2, V1), expr_eval(Expr2, Env2, NewEnv, V2), V1 < V2.
```

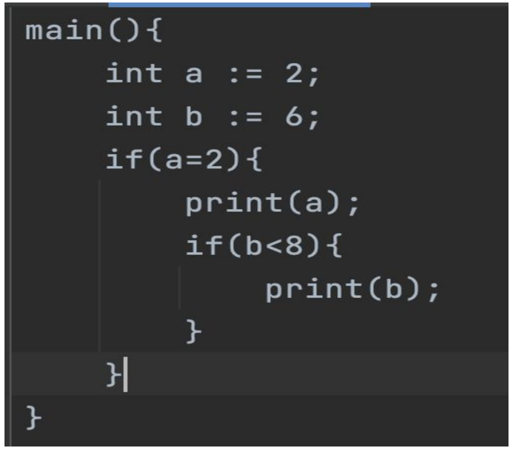


```
boolean_eval(t_boolean(Bool_expr1, 'and', Bool_expr2), Env, NewEnv, V) :- boolean_eval(Bool_expr1, Env, Env1, V1), boolean_eval(Bool_expr2, Env1, NewEnv, V2), boolean_check(V1
boolean_eval(t_boolean(Bool_expr1, 'or', Bool_expr2), Env, NewEnv, V) :- boolean_eval(Bool_expr1, Env, Env1, V1), boolean_eval(Bool_expr2, Env1, NewEnv, V2), boolean_check(V1,

boolean_check(true, 'and', true, true).
boolean_check(true, 'and', false, false).
boolean_check(false, 'and', true, false).
boolean_check(false, 'and', false, false).

boolean_check(true, 'or', true, true).
boolean_check(true, 'or', false, true).
boolean_check(false, 'or', true, true).
boolean_check(false, 'or', false, false).



%Evaluator for expressions
expr_eval(t_add(Left_mini_expr, Right_expr), Env, NewEnv, V) :-
    expr_eval(Left_mini_expr, Env, Env1, V1), expr_eval(Right_expr, Env1, NewEnv, V2), V is V1+V2.
expr_eval(t_minus(Left_mini_expr, Right_expr), Env, NewEnv, V) :-
    expr_eval(Left_mini_expr, Env, Env1, V1), expr_eval(Right_expr,Env1, NewEnv, V2), V is V1-V2.
expr_eval(t_multi(X, Y), Env, NewEnv, V) :-
    expr_eval(X, Env, Env1, V1), expr_eval(Y, Env1, NewEnv, V2), V is V1*V2.
expr_eval(t_divide(X, Y), Env, NewEnv, V) :-
    expr_eval(X, Env, Env1, V1), expr_eval(Y, Env1, NewEnv, V2), Y\=0, V is V1/V2.
expr_eval(t_bracket(X), Env, NewEnv, V) :-
    expr_eval(X, Env, NewEnv, V).
expr_eval(t_assign(X, :=, Y), Env, NewEnv, V) :-
    expr_eval(Y, Env, Env1, V), update(X, V, Env1, NewEnv).
expr_eval(X, Env, Env, V) :- lookup(X, Env, V).
expr_eval(X, Env, NewEnv, V) :- ternaryexpr_eval(X, Env, NewEnv, V).
expr_eval(X, Env, Env, X) :- integer(X).
expr_eval(X, Env, Env, X) :- float(X).
```

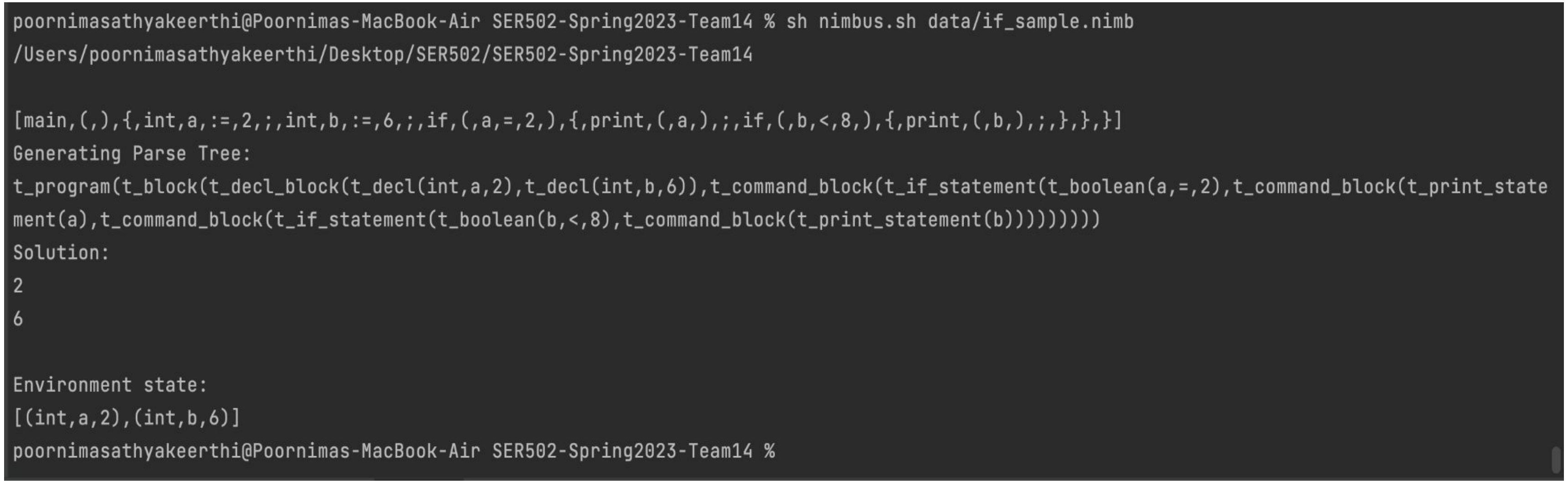


```prolog
%Declaring the identifiers with values
update_data(int, ID, Value, [], [(int, ID, Value)]) :- integer(Value).
update_data(int, _, Value, [], []) :- not(integer(Value)), illegal_type(Value, int).
update_data(float, ID, Value, [], [(float, ID, Value)]) :- float(Value).
update_data(float, _, Value, [], []) :- not(float(Value)), illegal_type(Value, float).
update_data(string, ID, Value, [], [(string, ID, Value)]) :- string(Value).
update_data(string, _, Value, [], []) :- not(string(Value)), illegal_type(Value, string).
update_data(boolean, ID, Value, [], [(boolean, ID, Value)]) :- Value==true;Value==false.
update_data(boolean, _, Value, [], []) :- Value \= true, Value \= false, illegal_type(Value, boolean).

update(Datatype, ID, Value, [], NewEnv) :- update_data(Datatype, ID, Value, [], NewEnv).
update(Datatype, ID, Value, [Head|Tail], [Head|NewEnv]) :-
    Head \= (_,ID,_), update(Datatype, ID, Value, Tail, NewEnv).
update(_, Name, _, [Head| _], _NewEnv) :- Head=(_,Name,_), error_typecheck(Name).

%Updating the identifiers with values
update(ID, Value, [H|T], [H|NewEnv]) :- H \= (_, ID, _), update(ID, Value, T, NewEnv).

update(ID, Value, [(int, ID, _)|T], [(int, ID, Value)|T]) :- integer(Value).
update(ID, Value, [(int, ID, _)|_], [_]) :- not(integer(Value)), illegal_type(Value, int).

update(ID, Value, [(float, ID, _)|T], [(float, ID, Value)|T]) :- float(Value).
update(ID, Value, [(float, ID, _)|_], [_]) :- not(float(Value)), illegal_type(Value, float).

update(ID, Value, [(string, ID, _)|T], [(string, ID, Value)|T]) :- string(Value).
update(ID, Value, [(string, ID, _)|_], [_]) :- not(string(Value)), illegal_type(Value, string).

update(ID, Value, [(boolean, ID, _)|T], [(boolean, ID, Value)|T]) :- Value==true;Value==false.
update(ID, Value, [(boolean, ID, _)|_], [_]) :-
    Value \= true, Value \= false, illegal_type(Value, boolean).

%type check errors
illegal_type(Value, Datatype) :-
    format('Illegal type conversion. ~w is not an ~w type', [Value, Datatype]).
error_typecheck(ID) :-
    format('Variable already initialised in different datatype. ~w', [ID]).

%Lookup for the variable names in the environment
lookup(Varname, [(_, Varname, Value)|_], Value).
lookup(Varname, [_|T], Value) :- lookup(Varname, T, Value).
```

Sample Run



```
main(){
    int a := 2;
    int b := 6;
    if(a=2){
        print(a);
        if(b<8){
            print(b);
        }
    }
}
```



```
poornimasathyakeerthi@Poornimas-MacBook-Air SER502-Spring2023-Team14 % sh nimbus.sh data/if_sample.nimb
/Users/poornimasathyakeerthi/Desktop/SER502/SER502-Spring2023-Team14

[main,(,),{,int,a,:=,2,;,int,b,:=,6,;,if,(,a,=,2,),{,print,(,a,),;,if,(,b,<,8,),{,print,(,b,),;,},},}]
Generating Parse Tree:
t_program(t_block(t_decl_block(t_decl(int,a,2),t_decl(int,b,6)),t_command_block(t_if_statement(t_boolean(a,=,2),t_command_block(t_print_state
ment(a),t_command_block(t_if_statement(t_boolean(b,<,8),t_command_block(t_print_statement(b)))))))))
Solution:
2
6

Environment state:
[(int,a,2),(int,b,6)]
poornimasathyakeerthi@Poornimas-MacBook-Air SER502-Spring2023-Team14 %
```

Installation Steps

Install Instructions

git clone https://github.com/pskeerth/SER502-Spring2023-Team14.git

Run Instructions

You need to have swi-prolog installed in your system(Mac) Run the below shell script command to execute:

sh nimbus.sh .nimb
Eg: sh nimbus.sh data/if_sample.nimb

Demo

Future Enhancements

- Data structures: List, Map
- Datatypes: Double, Char
- Functions

Thank You