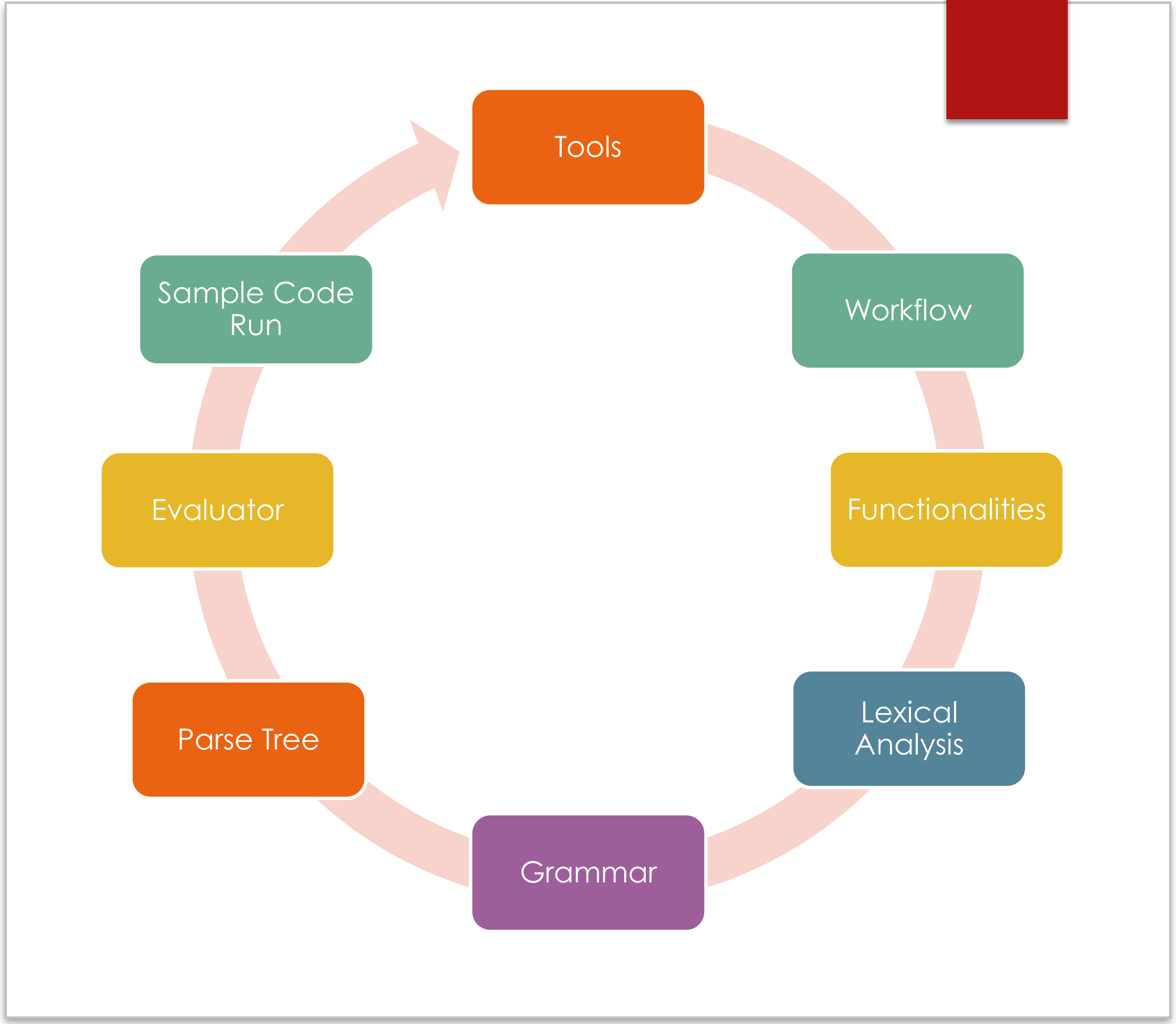
MUVY
PROGRAMMING LANGUAGE

SER-502 Team-19

Our Team

- Harsha Vardhan Mupparaju
- Kethan Yeddla
- Pavithra Moravaneni
- Vishnu Teja Vantukala
- Vishwanath Reddy Yasa

Contents


Tools
Workflow
Functionalities
Lexical Analysis
Grammar
Parse Tree
Evaluator
Sample Code Run

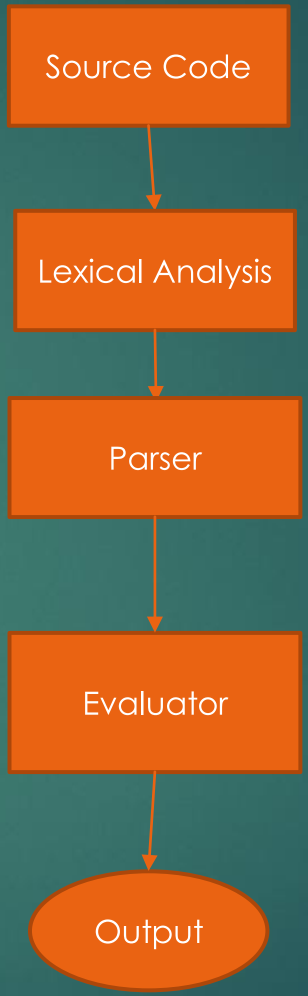
Tools

- SWISH Prolog – Run Time Environment.
- Prolog – Lexical Analysis, Parse Tree and Evaluator
- .sp – Parse Tree Code.
- .mvy – File Extension

Features

- Arithmetic Operations. (+, -, /, *)
- Boolean Operations. (and, or, not)
- Primitive data Types. (int, string, bool)
- Relational Operators. (= >, <, >=, <=, ==)
- Assignment Operators.
- Conditional Operators. (if, if-then-else, ternary)
- Iterative Statements. (for, while, for in range)
- Increment and Decrement Operators.
- Structured block.
- Low Level Language.

Workflow



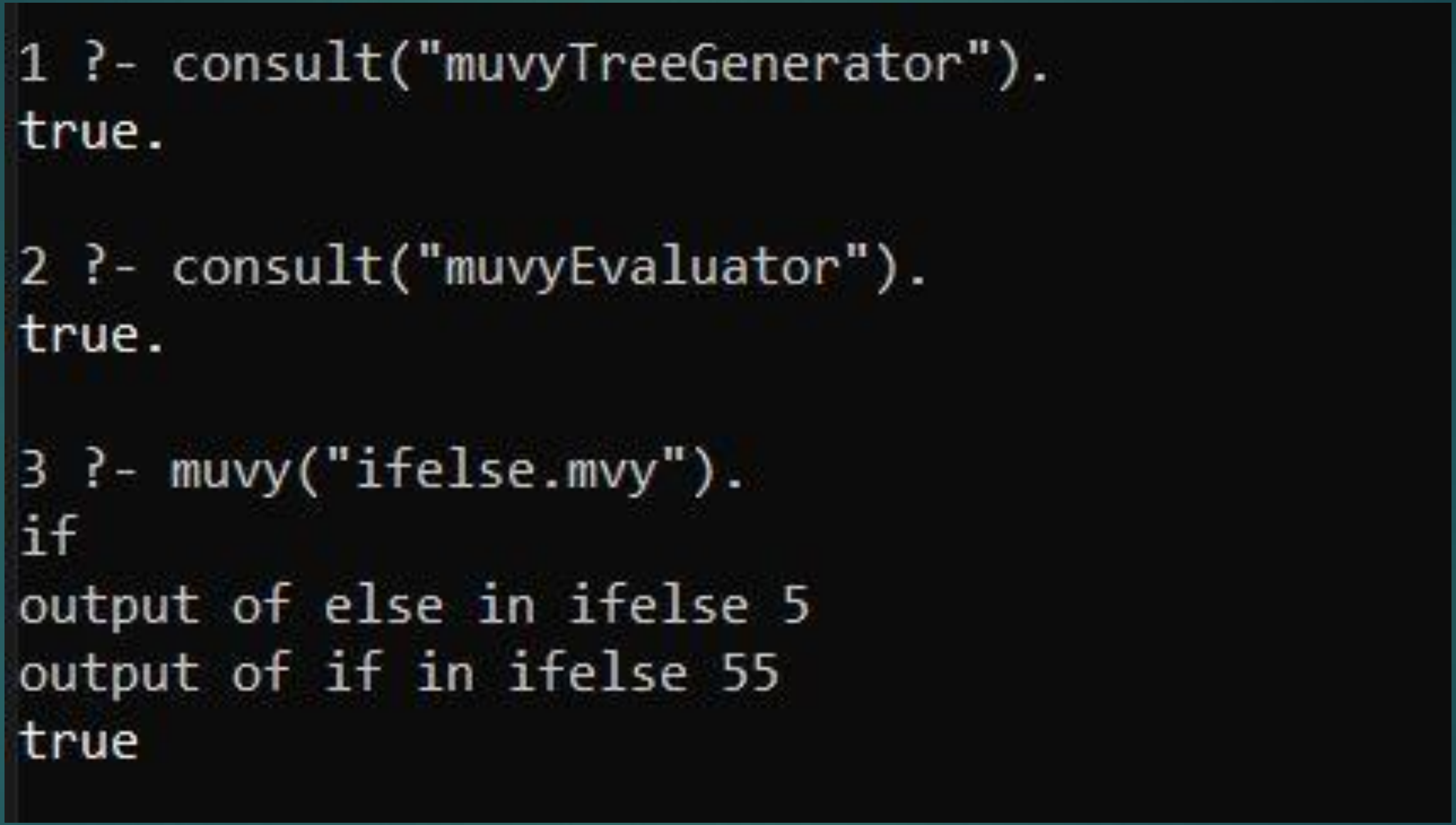
- Input file with extension .mvy containing source code.

- Here the source code converted into tokens

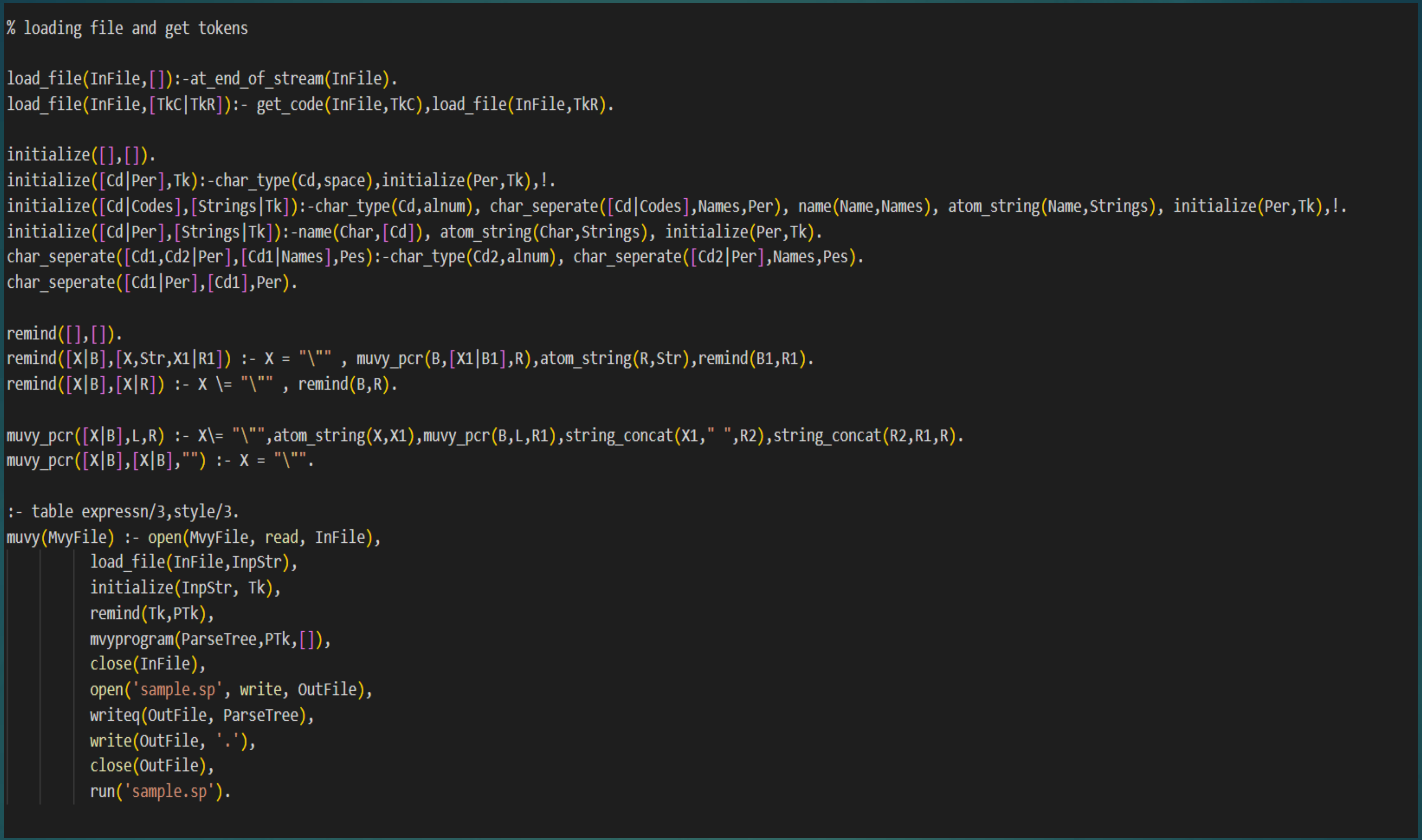
- Tokens generated are converted into Parse Tree.

- Here the parsed code is interpreted, evaluated using the semantic rules and generates the result

How to Run ?

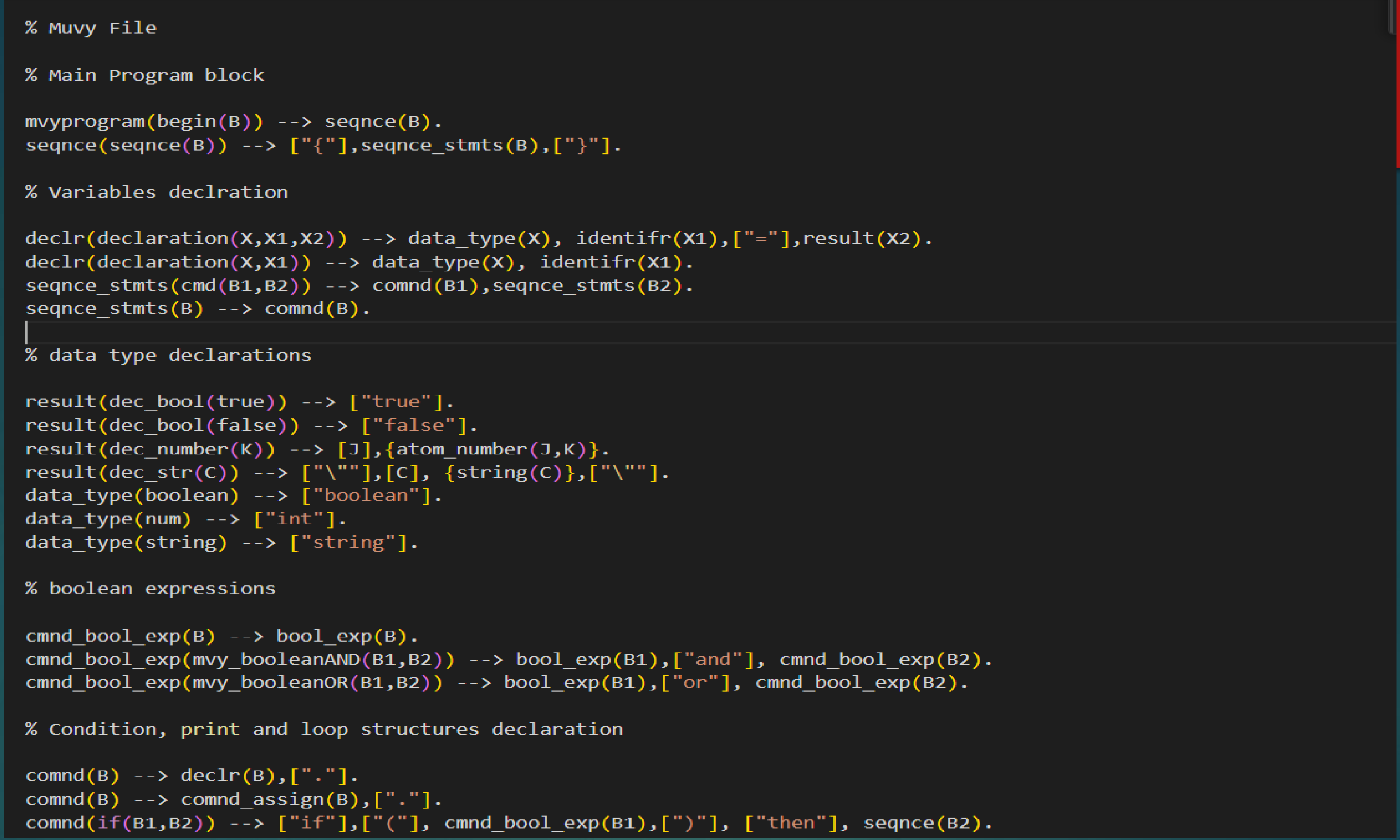


```
1 ?- consult("muvyTreeGenerator").
true.

2 ?- consult("muvyEvaluator").
true.

3 ?- muvy("ifelse.mvy").
if
output of else in ifelse 5
output of if in ifelse 55
true
```



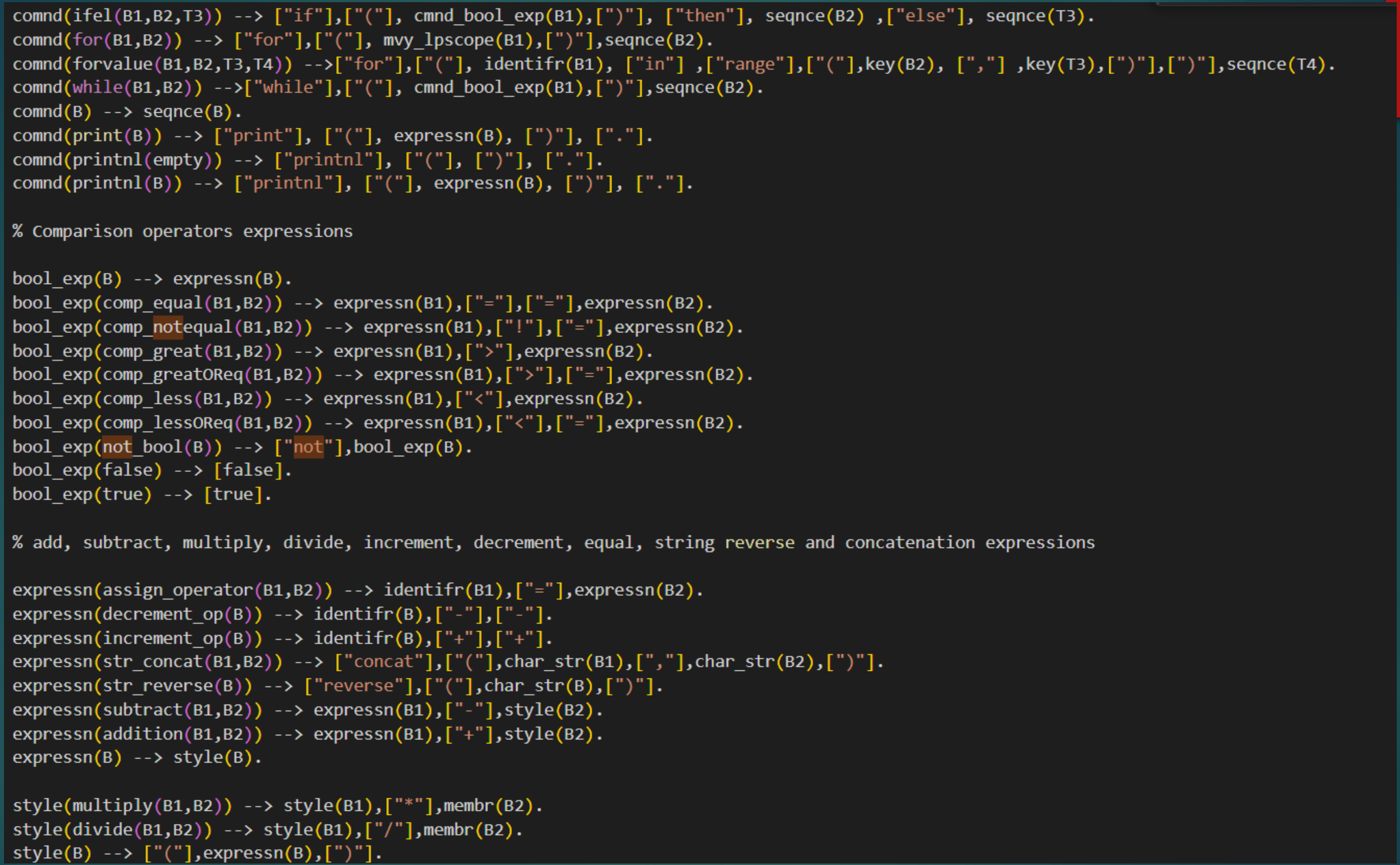
```
% loading file and get tokens

load_file(InFile,[]):-at_end_of_stream(InFile).
load_file(InFile,[TkC|TkR]):- get_code(InFile,TkC),load_file(InFile,TkR).

initialize([],[]).
initialize([Cd|Per],Tk):-char_type(Cd,space),initialize(Per,Tk),!.
initialize([Cd|Codes],[Strings|Tk]):-char_type(Cd,alnum), char_seperate([Cd|Codes],Names,Per), name(Name,Names), atom_string(Name,Strings), initialize(Per,Tk),!.
initialize([Cd|Per],[Strings|Tk]):-name(Char,[Cd]), atom_string(Char,Strings), initialize(Per,Tk).
char_seperate([Cd1,Cd2|Per],[Cd1|Names],Pes):-char_type(Cd2,alnum), char_seperate([Cd2|Per],Names,Pes).
char_seperate([Cd1|Per],[Cd1],Per).

remind([],[]).
remind([X|B],[X,Str,X1|R1]) :- X = "\"" , muvy_pcr(B,[X1|B1],R),atom_string(R,Str),remind(B1,R1).
remind([X|B],[X|R]) :- X \= "\"" , remind(B,R).

muvy_pcr([X|B],L,R) :- X\= "\"",atom_string(X,X1),muvy_pcr(B,L,R1),string_concat(X1," ",R2),string_concat(R2,R1,R).
muvy_pcr([X|B],[X|B],"") :- X = "\"".

:- table expressn/3,style/3.
muvy(MvyFile) :- open(MvyFile, read, InFile),
                load_file(InFile,InpStr),
                initialize(InpStr, Tk),
                remind(Tk,PTk),
                mvyprogram(ParseTree,PTk,[]),
                close(InFile),
                open('sample.sp', write, OutFile),
                writeq(OutFile, ParseTree),
                write(OutFile, '.'),
                close(OutFile),
                run('sample.sp').
```

Lexical Code



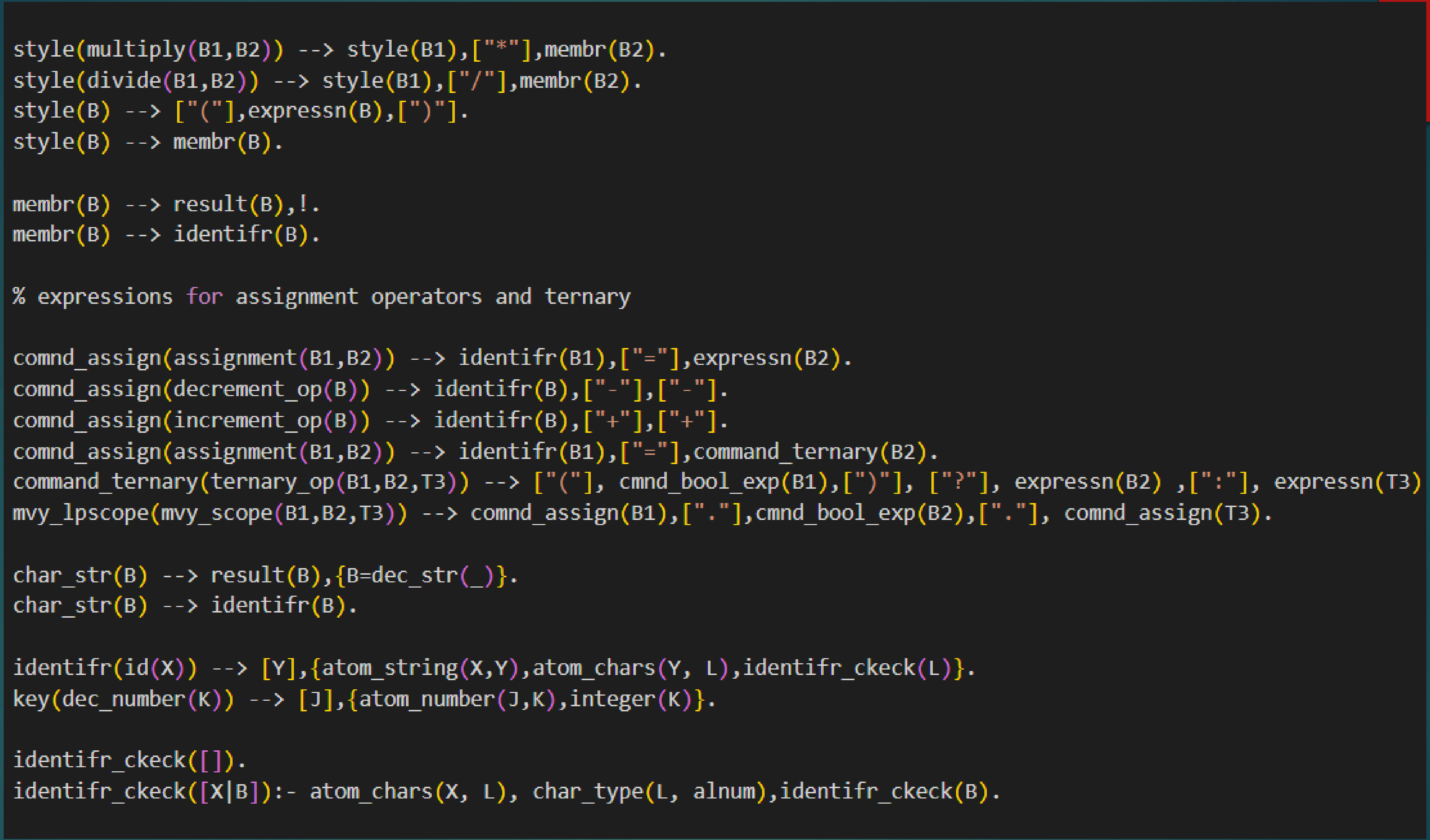
```
% Muvy File

% Main Program block

mvyprogram(begin(B)) --> seqnce(B).
seqnce(seqnce(B)) --> ["{"],seqnce_stmts(B),["}"].

% Variables declration

declr(declaration(X,X1,X2)) --> data_type(X), identifr(X1),["="],result(X2).
declr(declaration(X,X1)) --> data_type(X), identifr(X1).
seqnce_stmts(cmd(B1,B2)) --> comnd(B1),seqnce_stmts(B2).
seqnce_stmts(B) --> comnd(B).

% data type declarations

result(dec_bool(true)) --> ["true"].
result(dec_bool(false)) --> ["false"].
result(dec_number(K)) --> [J],{atom_number(J,K)}.
result(dec_str(C)) --> ["\""],[C], {string(C)},["\""].
data_type(boolean) --> ["boolean"].
data_type(num) --> ["int"].
data_type(string) --> ["string"].

% boolean expressions

cmnd_bool_exp(B) --> bool_exp(B).
cmnd_bool_exp(mvy_booleanAND(B1,B2)) --> bool_exp(B1),["and"], cmnd_bool_exp(B2).
cmnd_bool_exp(mvy_booleanOR(B1,B2)) --> bool_exp(B1),["or"], cmnd_bool_exp(B2).

% Condition, print and loop structures declaration

comnd(B) --> declr(B),["."].
comnd(B) --> comnd_assign(B),["."].
comnd(if(B1,B2)) --> ["if"],["("], cmnd_bool_exp(B1),[")"], ["then"], seqnce(B2).
```

Parse tree



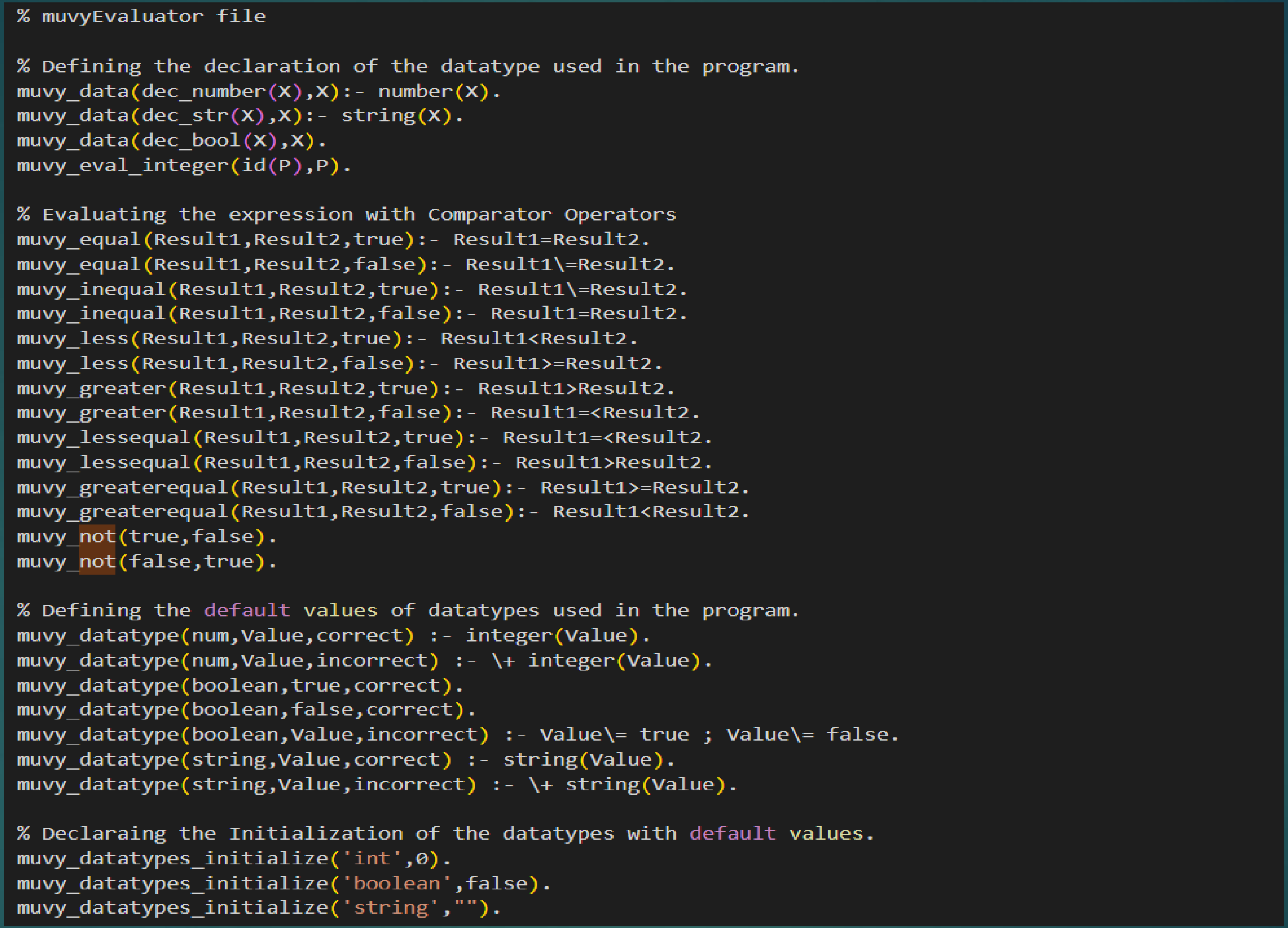
```
comnd(ifel(B1,B2,T3)) --> ["if"],["("], cmnd_bool_exp(B1),[")"], ["then"], seqnce(B2) ,["else"], seqnce(T3).
comnd(for(B1,B2)) --> ["for"],["("], mvy_lpscope(B1),[")"],seqnce(B2).
comnd(forvalue(B1,B2,T3,T4)) -->["for"],["("], identifr(B1), ["in"] ,["range"],["("],key(B2), [","] ,key(T3),[")"],[")"],seqnce(T4).
comnd(while(B1,B2)) -->["while"],["("], cmnd_bool_exp(B1),[")"],seqnce(B2).
comnd(B) --> seqnce(B).
comnd(print(B)) --> ["print"], ["("], expressn(B), [")"], ["."].
comnd(printnl(empty)) --> ["printnl"], ["("], [")"], ["."].
comnd(printnl(B)) --> ["printnl"], ["("], expressn(B), [")"], ["."].

% Comparison operators expressions

bool_exp(B) --> expressn(B).
bool_exp(comp_equal(B1,B2)) --> expressn(B1),["="],["="],expressn(B2).
bool_exp(comp_notequal(B1,B2)) --> expressn(B1),["!"],["="],expressn(B2).
bool_exp(comp_great(B1,B2)) --> expressn(B1),[">"],expressn(B2).
bool_exp(comp_greatOReq(B1,B2)) --> expressn(B1),[">"],["="],expressn(B2).
bool_exp(comp_less(B1,B2)) --> expressn(B1),["<"],expressn(B2).
bool_exp(comp_lessOReq(B1,B2)) --> expressn(B1),["<"],["="],expressn(B2).
bool_exp(not_bool(B)) --> ["not"],bool_exp(B).
bool_exp(false) --> [false].
bool_exp(true) --> [true].

% add, subtract, multiply, divide, increment, decrement, equal, string reverse and concatenation expressions

expressn(assign_operator(B1,B2)) --> identifr(B1),["="],expressn(B2).
expressn(decrement_op(B)) --> identifr(B),["-"],["-"].
expressn(increment_op(B)) --> identifr(B),["+"],["+"].
expressn(str_concat(B1,B2)) --> ["concat"],["("],char_str(B1),[","],char_str(B2),[")"].
expressn(str_reverse(B)) --> ["reverse"],["("],char_str(B),[")"].
expressn(subtract(B1,B2)) --> expressn(B1),["-"],style(B2).
expressn(addition(B1,B2)) --> expressn(B1),["+"],style(B2).
expressn(B) --> style(B).

style(multiply(B1,B2)) --> style(B1),["*"],membr(B2).
style(divide(B1,B2)) --> style(B1),["/"],membr(B2).
style(B) --> ["("],expressn(B),[")"].
```

Parse tree



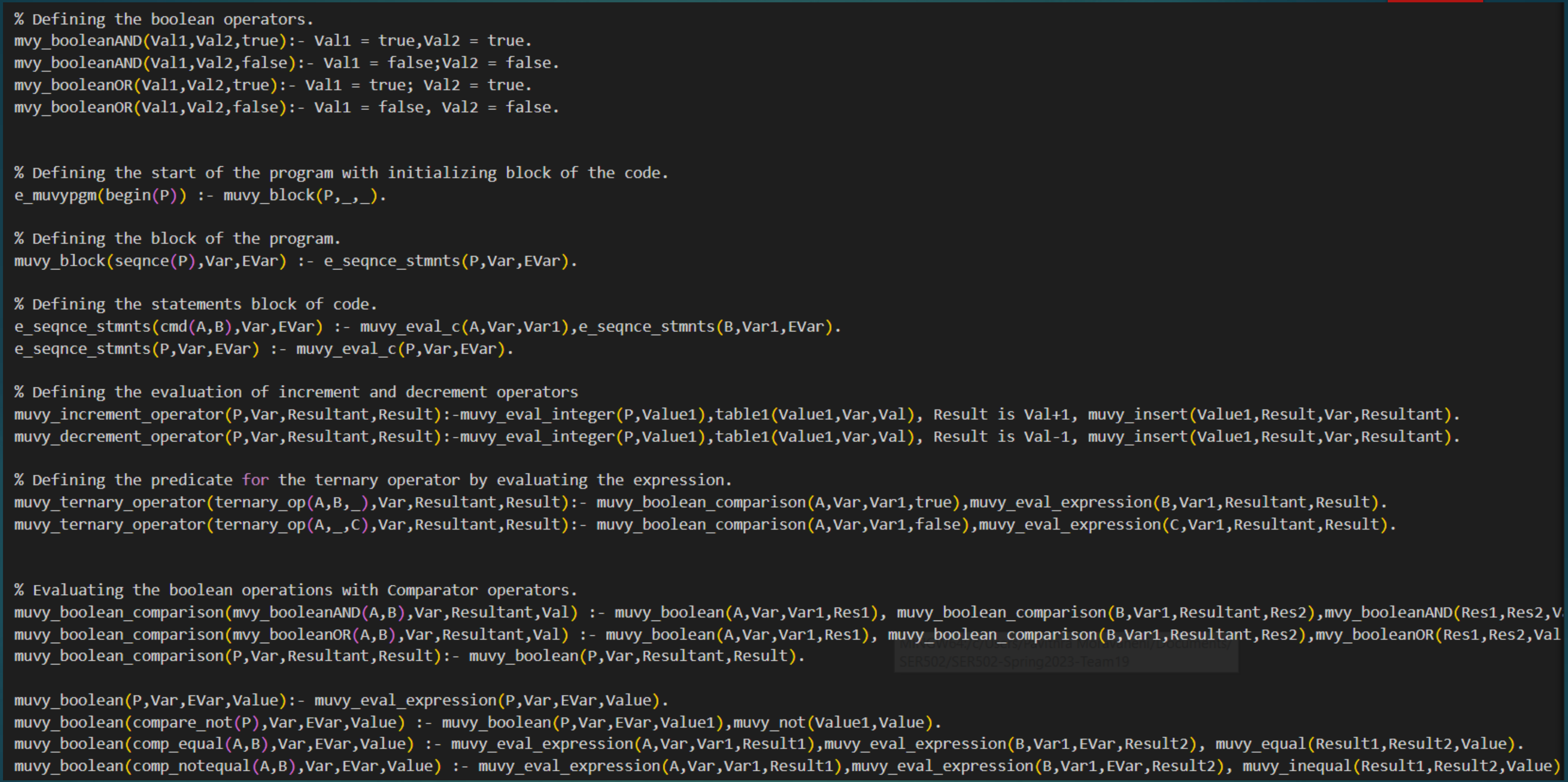
```
style(multiply(B1,B2)) --> style(B1),["*"],membr(B2).
style(divide(B1,B2)) --> style(B1),["/"],membr(B2).
style(B) --> ["("],expressn(B),[")"].
style(B) --> membr(B).

membr(B) --> result(B),!.
membr(B) --> identifr(B).

% expressions for assignment operators and ternary

comnd_assign(assignment(B1,B2)) --> identifr(B1),["="],expressn(B2).
comnd_assign(decrement_op(B)) --> identifr(B),["-"],["-"].
comnd_assign(increment_op(B)) --> identifr(B),["+"],["+"].
comnd_assign(assignment(B1,B2)) --> identifr(B1),["="],command_ternary(B2).
command_ternary(ternary_op(B1,B2,T3)) --> ["("], cmnd_bool_exp(B1),[")"], ["?"], expressn(B2) ,[":"], expressn(T3)
mvy_lpscope(mvy_scope(B1,B2,T3)) --> comnd_assign(B1),["."],cmnd_bool_exp(B2),["."], comnd_assign(T3).

char_str(B) --> result(B),{B=dec_str(_)}.
char_str(B) --> identifr(B).

identifr(id(X)) --> [Y],{atom_string(X,Y),atom_chars(Y, L),identifr_ckeck(L)}.
key(dec_number(K)) --> [J],{atom_number(J,K),integer(K)}.

identifr_ckeck([]).
identifr_ckeck([X|B]):- atom_chars(X, L), char_type(L, alnum),identifr_ckeck(B).
```

Parse tree



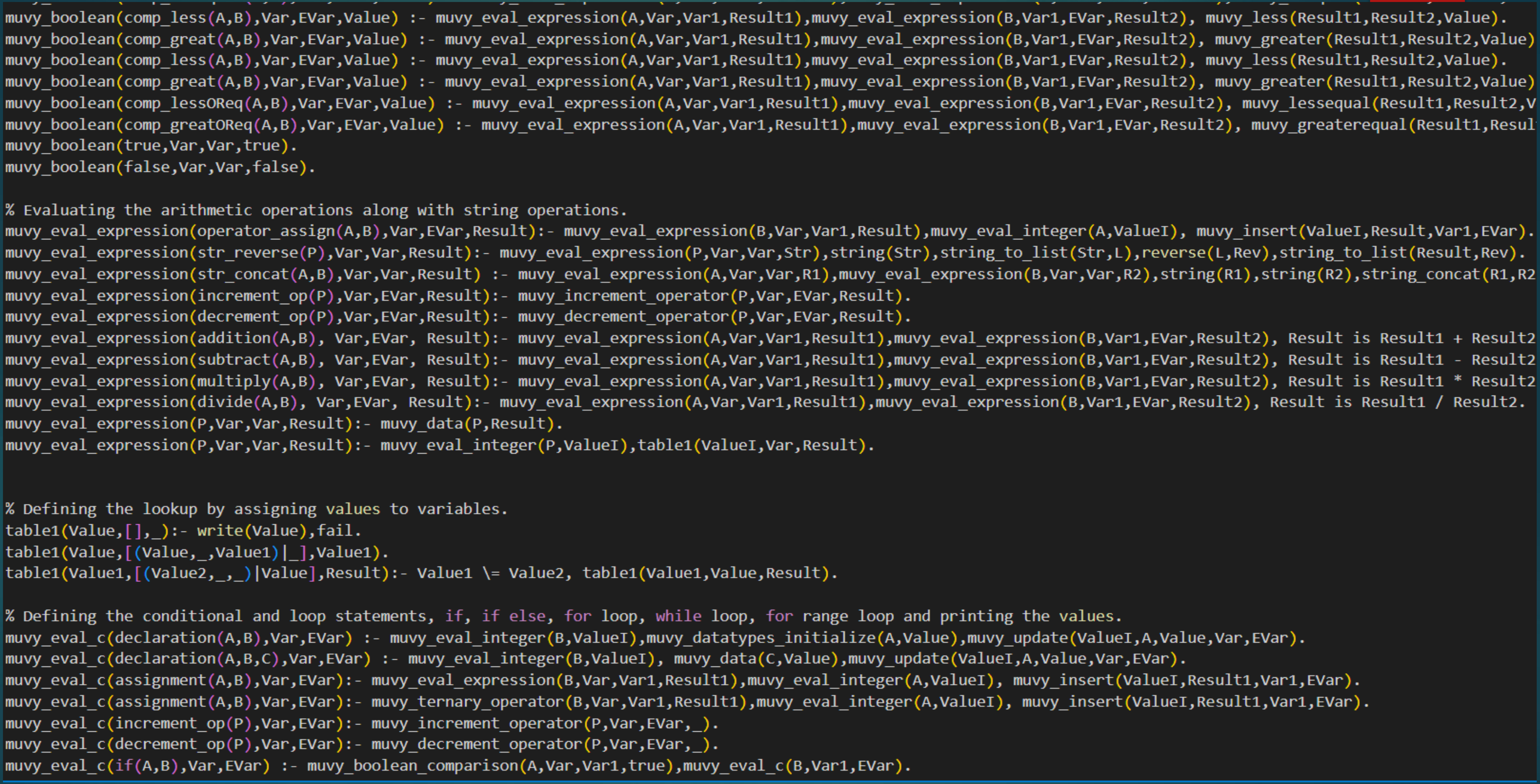
```
% muvyEvaluator file

% Defining the declaration of the datatype used in the program.
muvy_data(dec_number(X),X):- number(X).
muvy_data(dec_str(X),X):- string(X).
muvy_data(dec_bool(X),X).
muvy_eval_integer(id(P),P).

% Evaluating the expression with Comparator Operators
muvy_equal(Result1,Result2,true):- Result1=Result2.
muvy_equal(Result1,Result2,false):- Result1\=Result2.
muvy_inequal(Result1,Result2,true):- Result1\=Result2.
muvy_inequal(Result1,Result2,false):- Result1=Result2.
muvy_less(Result1,Result2,true):- Result1<Result2.
muvy_less(Result1,Result2,false):- Result1>=Result2.
muvy_greater(Result1,Result2,true):- Result1>Result2.
muvy_greater(Result1,Result2,false):- Result1=<Result2.
muvy_lessequal(Result1,Result2,true):- Result1=<Result2.
muvy_lessequal(Result1,Result2,false):- Result1>Result2.
muvy_greaterequal(Result1,Result2,true):- Result1>=Result2.
muvy_greaterequal(Result1,Result2,false):- Result1<Result2.
muvy_not(true,false).
muvy_not(false,true).

% Defining the default values of datatypes used in the program.
muvy_datatype(num,Value,correct) :- integer(Value).
muvy_datatype(num,Value,incorrect) :- \+ integer(Value).
muvy_datatype(boolean,true,correct).
muvy_datatype(boolean,false,correct).
muvy_datatype(boolean,Value,incorrect) :- Value\= true ; Value\= false.
muvy_datatype(string,Value,correct) :- string(Value).
muvy_datatype(string,Value,incorrect) :- \+ string(Value).

% Declaraing the Initialization of the datatypes with default values.
muvy_datatypes_initialize('int',0).
muvy_datatypes_initialize('boolean',false).
muvy_datatypes_initialize('string',"").
```

Evaluator



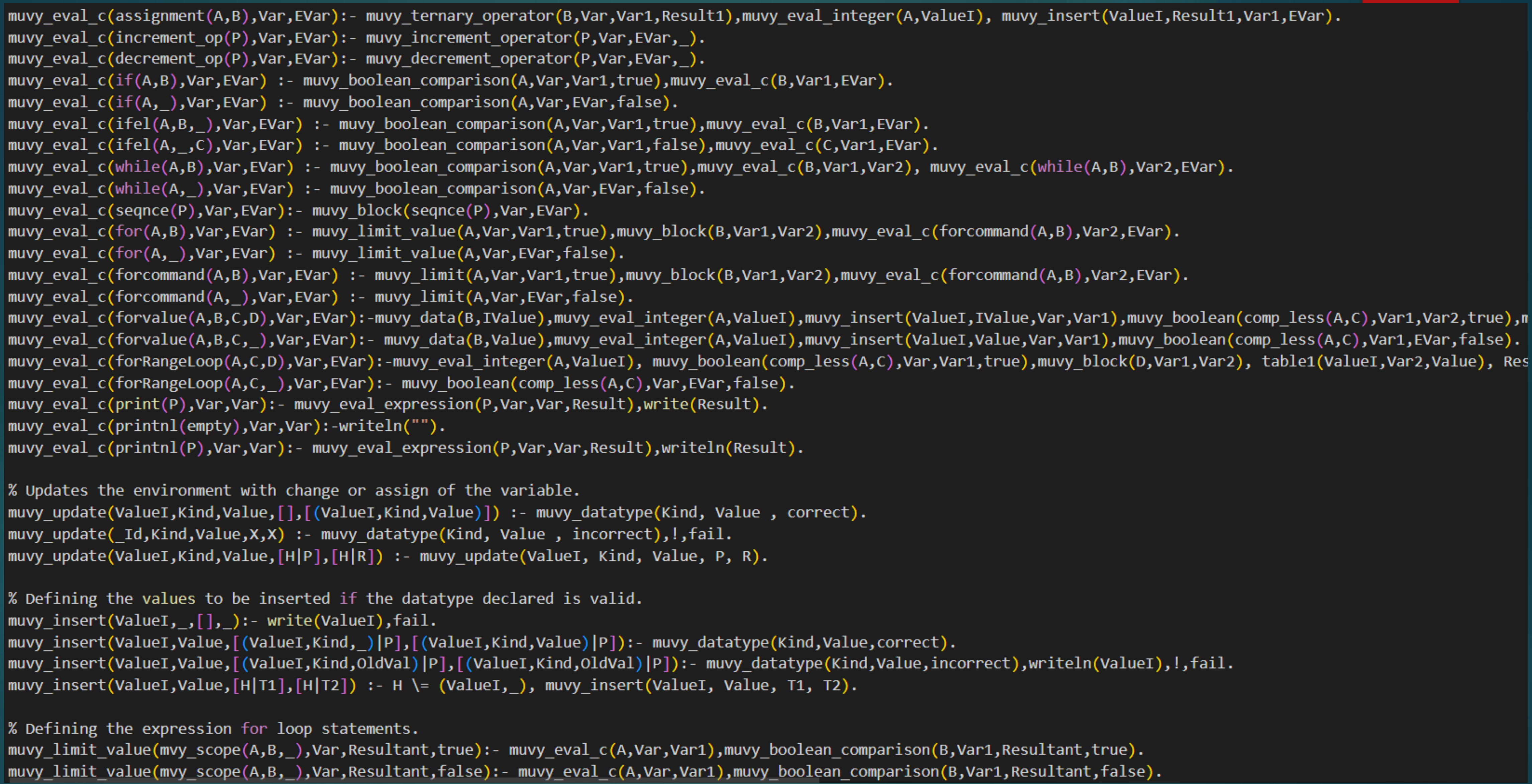
```
% Defining the boolean operators.
mvy_booleanAND(Val1,Val2,true):- Val1 = true,Val2 = true.
mvy_booleanAND(Val1,Val2,false):- Val1 = false;Val2 = false.
mvy_booleanOR(Val1,Val2,true):- Val1 = true; Val2 = true.
mvy_booleanOR(Val1,Val2,false):- Val1 = false, Val2 = false.


% Defining the start of the program with initializing block of the code.
e_muvypgm(begin(P)) :- muvy_block(P,_,_).

% Defining the block of the program.
muvy_block(seqnce(P),Var,EVar) :- e_seqnce_stmnts(P,Var,EVar).

% Defining the statements block of code.
e_seqnce_stmnts(cmd(A,B),Var,EVar) :- muvy_eval_c(A,Var,Var1),e_seqnce_stmnts(B,Var1,EVar).
e_seqnce_stmnts(P,Var,EVar) :- muvy_eval_c(P,Var,EVar).

% Defining the evaluation of increment and decrement operators
muvy_increment_operator(P,Var,Resultant,Result):-muvy_eval_integer(P,Value1),table1(Value1,Var,Val), Result is Val+1, muvy_insert(Value1,Result,Var,Resultant).
muvy_decrement_operator(P,Var,Resultant,Result):-muvy_eval_integer(P,Value1),table1(Value1,Var,Val), Result is Val-1, muvy_insert(Value1,Result,Var,Resultant).

% Defining the predicate for the ternary operator by evaluating the expression.
muvy_ternary_operator(ternary_op(A,B,_),Var,Resultant,Result):- muvy_boolean_comparison(A,Var,Var1,true),muvy_eval_expression(B,Var1,Resultant,Result).
muvy_ternary_operator(ternary_op(A,_,C),Var,Resultant,Result):- muvy_boolean_comparison(A,Var,Var1,false),muvy_eval_expression(C,Var1,Resultant,Result).


% Evaluating the boolean operations with Comparator operators.
muvy_boolean_comparison(mvy_booleanAND(A,B),Var,Resultant,Val) :- muvy_boolean(A,Var,Var1,Res1), muvy_boolean_comparison(B,Var1,Resultant,Res2),mvy_booleanAND(Res1,Res2,V
muvy_boolean_comparison(mvy_booleanOR(A,B),Var,Resultant,Val) :- muvy_boolean(A,Var,Var1,Res1), muvy_boolean_comparison(B,Var1,Resultant,Res2),mvy_booleanOR(Res1,Res2,Val
muvy_boolean_comparison(P,Var,Resultant,Result):- muvy_boolean(P,Var,Resultant,Result).

muvy_boolean(P,Var,EVar,Value):- muvy_eval_expression(P,Var,EVar,Value).
muvy_boolean(compare_not(P),Var,EVar,Value) :- muvy_boolean(P,Var,EVar,Value1),muvy_not(Value1,Value).
muvy_boolean(comp_equal(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_equal(Result1,Result2,Value).
muvy_boolean(comp_notequal(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_inequal(Result1,Result2,Value)
```

Evaluator



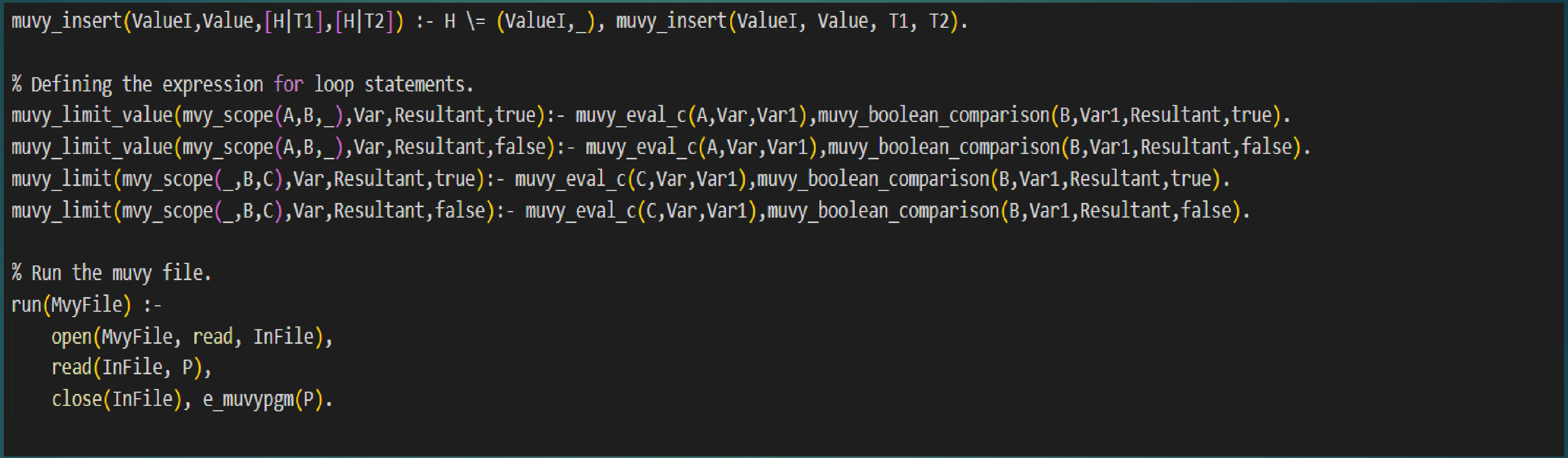
```
muvy_boolean(comp_less(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_less(Result1,Result2,Value).
muvy_boolean(comp_great(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_greater(Result1,Result2,Value)
muvy_boolean(comp_less(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_less(Result1,Result2,Value).
muvy_boolean(comp_great(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_greater(Result1,Result2,Value)
muvy_boolean(comp_lessOReq(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_lessequal(Result1,Result2,V
muvy_boolean(comp_greatOReq(A,B),Var,EVar,Value) :- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), muvy_greaterequal(Result1,Resul
muvy_boolean(true,Var,Var,true).
muvy_boolean(false,Var,Var,false).

% Evaluating the arithmetic operations along with string operations.
muvy_eval_expression(operator_assign(A,B),Var,EVar,Result):- muvy_eval_expression(B,Var,Var1,Result),muvy_eval_integer(A,ValueI), muvy_insert(ValueI,Result,Var1,EVar).
muvy_eval_expression(str_reverse(P),Var,Var,Result):- muvy_eval_expression(P,Var,Var,Str),string(Str),string_to_list(Str,L),reverse(L,Rev),string_to_list(Result,Rev).
muvy_eval_expression(str_concat(A,B),Var,Var,Result) :- muvy_eval_expression(A,Var,Var,R1),muvy_eval_expression(B,Var,Var,R2),string(R1),string(R2),string_concat(R1,R2
muvy_eval_expression(increment_op(P),Var,EVar,Result):- muvy_increment_operator(P,Var,EVar,Result).
muvy_eval_expression(decrement_op(P),Var,EVar,Result):- muvy_decrement_operator(P,Var,EVar,Result).
muvy_eval_expression(addition(A,B), Var,EVar, Result):- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), Result is Result1 + Result2
muvy_eval_expression(subtract(A,B), Var,EVar, Result):- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), Result is Result1 - Result2
muvy_eval_expression(multiply(A,B), Var,EVar, Result):- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), Result is Result1 * Result2
muvy_eval_expression(divide(A,B), Var,EVar, Result):- muvy_eval_expression(A,Var,Var1,Result1),muvy_eval_expression(B,Var1,EVar,Result2), Result is Result1 / Result2.
muvy_eval_expression(P,Var,Var,Result):- muvy_data(P,Result).
muvy_eval_expression(P,Var,Var,Result):- muvy_eval_integer(P,ValueI),table1(ValueI,Var,Result).


% Defining the lookup by assigning values to variables.
table1(Value,[],_):- write(Value),fail.
table1(Value,[(Value,_,Value1)|_],Value1).
table1(Value1,[(Value2,_,_)|Value],Result):- Value1 \= Value2, table1(Value1,Value,Result).

% Defining the conditional and loop statements, if, if else, for loop, while loop, for range loop and printing the values.
muvy_eval_c(declaration(A,B),Var,EVar) :- muvy_eval_integer(B,ValueI),muvy_datatypes_initialize(A,Value),muvy_update(ValueI,A,Value,Var,EVar).
muvy_eval_c(declaration(A,B,C),Var,EVar) :- muvy_eval_integer(B,ValueI), muvy_data(C,Value),muvy_update(ValueI,A,Value,Var,EVar).
muvy_eval_c(assignment(A,B),Var,EVar):- muvy_eval_expression(B,Var,Var1,Result1),muvy_eval_integer(A,ValueI), muvy_insert(ValueI,Result1,Var1,EVar).
muvy_eval_c(assignment(A,B),Var,EVar):- muvy_ternary_operator(B,Var,Var1,Result1),muvy_eval_integer(A,ValueI), muvy_insert(ValueI,Result1,Var1,EVar).
muvy_eval_c(increment_op(P),Var,EVar):- muvy_increment_operator(P,Var,EVar,_).
muvy_eval_c(decrement_op(P),Var,EVar):- muvy_decrement_operator(P,Var,EVar,_).
muvy_eval_c(if(A,B),Var,EVar) :- muvy_boolean_comparison(A,Var,Var1,true),muvy_eval_c(B,Var1,EVar).
```

Evaluator



```
muvy_eval_c(assignment(A,B),Var,EVar):- muvy_ternary_operator(B,Var,Var1,Result1),muvy_eval_integer(A,ValueI), muvy_insert(ValueI,Result1,Var1,EVar).
muvy_eval_c(increment_op(P),Var,EVar):- muvy_increment_operator(P,Var,EVar,_).
muvy_eval_c(decrement_op(P),Var,EVar):- muvy_decrement_operator(P,Var,EVar,_).
muvy_eval_c(if(A,B),Var,EVar) :- muvy_boolean_comparison(A,Var,Var1,true),muvy_eval_c(B,Var1,EVar).
muvy_eval_c(if(A,_),Var,EVar) :- muvy_boolean_comparison(A,Var,EVar,false).
muvy_eval_c(ifel(A,B,_),Var,EVar) :- muvy_boolean_comparison(A,Var,Var1,true),muvy_eval_c(B,Var1,EVar).
muvy_eval_c(ifel(A,_,C),Var,EVar) :- muvy_boolean_comparison(A,Var,Var1,false),muvy_eval_c(C,Var1,EVar).
muvy_eval_c(while(A,B),Var,EVar) :- muvy_boolean_comparison(A,Var,Var1,true),muvy_eval_c(B,Var1,Var2), muvy_eval_c(while(A,B),Var2,EVar).
muvy_eval_c(while(A,_),Var,EVar) :- muvy_boolean_comparison(A,Var,EVar,false).
muvy_eval_c(seqnce(P),Var,EVar):- muvy_block(seqnce(P),Var,EVar).
muvy_eval_c(for(A,B),Var,EVar) :- muvy_limit_value(A,Var,Var1,true),muvy_block(B,Var1,Var2),muvy_eval_c(forcommand(A,B),Var2,EVar).
muvy_eval_c(for(A,_),Var,EVar) :- muvy_limit_value(A,Var,EVar,false).
muvy_eval_c(forcommand(A,B),Var,EVar) :- muvy_limit(A,Var,Var1,true),muvy_block(B,Var1,Var2),muvy_eval_c(forcommand(A,B),Var2,EVar).
muvy_eval_c(forcommand(A,_),Var,EVar) :- muvy_limit(A,Var,EVar,false).
muvy_eval_c(forvalue(A,B,C,D),Var,EVar):-muvy_data(B,IValue),muvy_eval_integer(A,ValueI),muvy_insert(ValueI,IValue,Var,Var1),muvy_boolean(comp_less(A,C),Var1,Var2,true),m
muvy_eval_c(forvalue(A,B,C,_),Var,EVar):- muvy_data(B,Value),muvy_eval_integer(A,ValueI),muvy_insert(ValueI,Value,Var,Var1),muvy_boolean(comp_less(A,C),Var1,EVar,false).
muvy_eval_c(forRangeLoop(A,C,D),Var,EVar):-muvy_eval_integer(A,ValueI), muvy_boolean(comp_less(A,C),Var,Var1,true),muvy_block(D,Var1,Var2), table1(ValueI,Var2,Value), Res
muvy_eval_c(forRangeLoop(A,C,_),Var,EVar):- muvy_boolean(comp_less(A,C),Var,EVar,false).
muvy_eval_c(print(P),Var,Var):- muvy_eval_expression(P,Var,Var,Result),write(Result).
muvy_eval_c(printnl(empty),Var,Var):-writeln("").
muvy_eval_c(printnl(P),Var,Var):- muvy_eval_expression(P,Var,Var,Result),writeln(Result).

% Updates the environment with change or assign of the variable.
muvy_update(ValueI,Kind,Value,[],[(ValueI,Kind,Value)]) :- muvy_datatype(Kind, Value , correct).
muvy_update(_Id,Kind,Value,X,X) :- muvy_datatype(Kind, Value , incorrect),!,fail.
muvy_update(ValueI,Kind,Value,[H|P],[H|R]) :- muvy_update(ValueI, Kind, Value, P, R).

% Defining the values to be inserted if the datatype declared is valid.
muvy_insert(ValueI,_,[],_):- write(ValueI),fail.
muvy_insert(ValueI,Value,[(ValueI,Kind,_)|P],[(ValueI,Kind,Value)|P]):- muvy_datatype(Kind,Value,correct).
muvy_insert(ValueI,Value,[(ValueI,Kind,OldVal)|P],[(ValueI,Kind,OldVal)|P]):- muvy_datatype(Kind,Value,incorrect),writeln(ValueI),!,fail.
muvy_insert(ValueI,Value,[H|T1],[H|T2]) :- H \= (ValueI,_), muvy_insert(ValueI, Value, T1, T2).

% Defining the expression for loop statements.
muvy_limit_value(mvy_scope(A,B,_),Var,Resultant,true):- muvy_eval_c(A,Var,Var1),muvy_boolean_comparison(B,Var1,Resultant,true).
muvy_limit_value(mvy_scope(A,B,_),Var,Resultant,false):- muvy_eval_c(A,Var,Var1),muvy_boolean_comparison(B,Var1,Resultant,false).
```

Evaluator

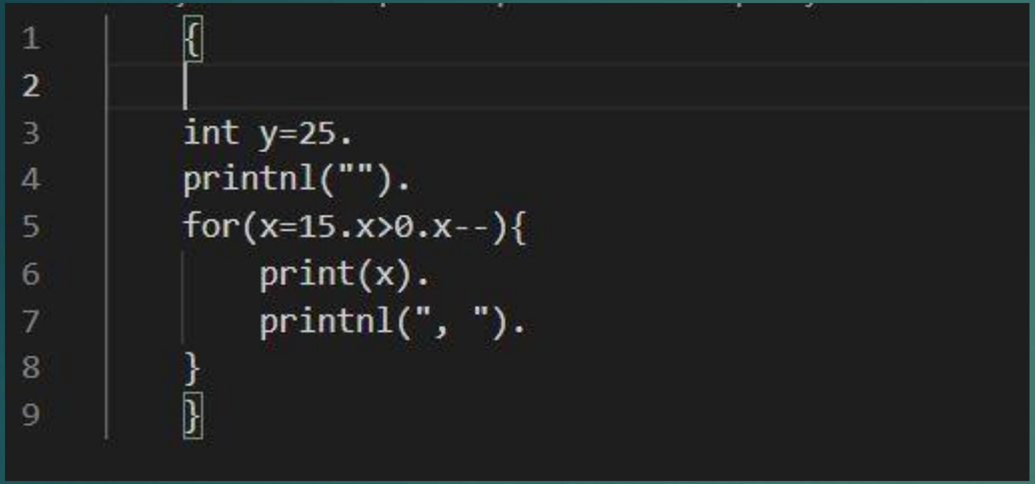


```
muvy_insert(ValueI,Value,[H|T1],[H|T2]) :- H \= (ValueI,_), muvy_insert(ValueI, Value, T1, T2).

% Defining the expression for loop statements.
muvy_limit_value(mvy_scope(A,B,_),Var,Resultant,true):- muvy_eval_c(A,Var,Var1),muvy_boolean_comparison(B,Var1,Resultant,true).
muvy_limit_value(mvy_scope(A,B,_),Var,Resultant,false):- muvy_eval_c(A,Var,Var1),muvy_boolean_comparison(B,Var1,Resultant,false).
muvy_limit(mvy_scope(_,B,C),Var,Resultant,true):- muvy_eval_c(C,Var,Var1),muvy_boolean_comparison(B,Var1,Resultant,true).
muvy_limit(mvy_scope(_,B,C),Var,Resultant,false):- muvy_eval_c(C,Var,Var1),muvy_boolean_comparison(B,Var1,Resultant,false).

% Run the muvy file.
run(MvyFile) :-
    open(MvyFile, read, InFile),
    read(InFile, P),
    close(InFile), e_muvypgm(P).
```

Evaluator

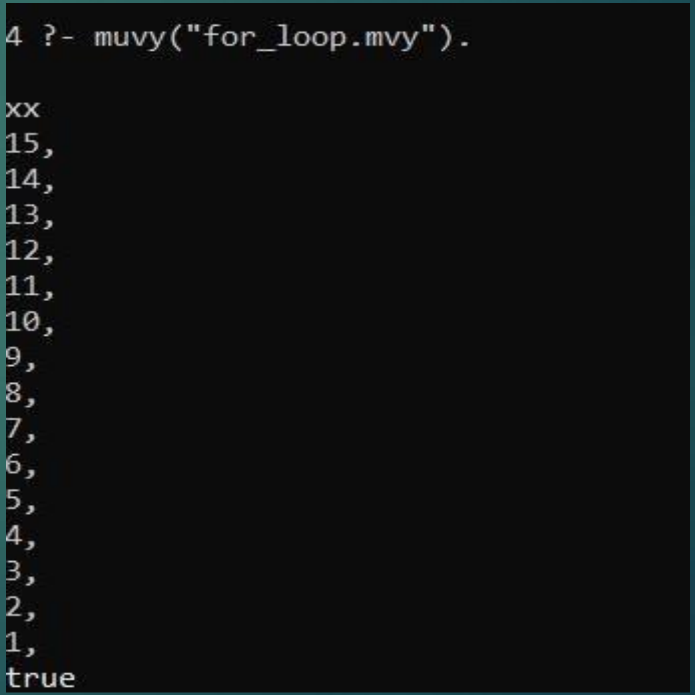
Sample run 1

Program demonstrating for Loop



```
{

int y=25.
printnl("").
for(x=15.x>0.x--){
    print(x).
    printnl(", ").
}
}
```

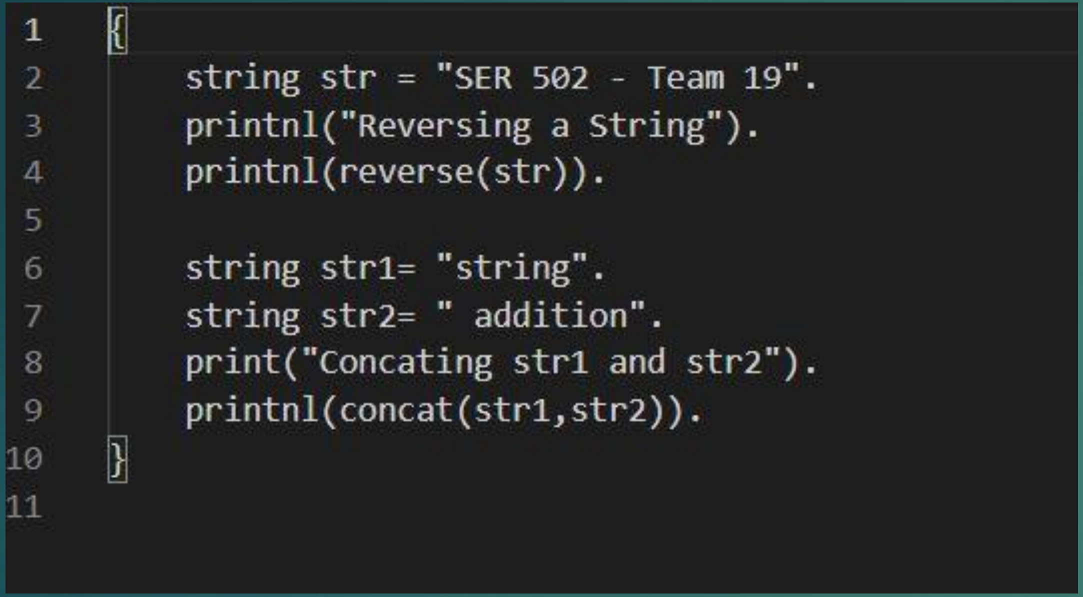
Output



```
4 ?- muvy("for_loop.mvy").

xx
15,
14,
13,
12,
11,
10,
9,
8,
7,
6,
5,
4,
3,
2,
1,
true
```

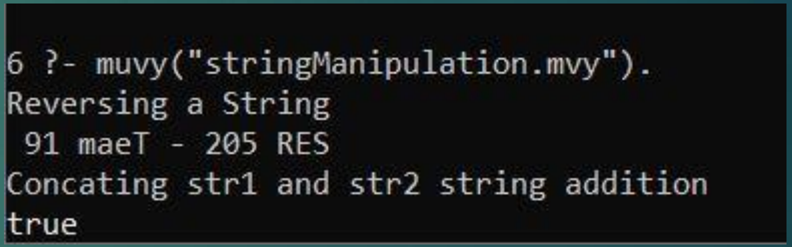
Sample run 2

Program for String manipulation



```
{
    string str = "SER 502 - Team 19".
    printnl("Reversing a String").
    printnl(reverse(str)).

    string str1= "string".
    string str2= " addition".
    print("Concating str1 and str2").
    printnl(concat(str1,str2)).
}
```

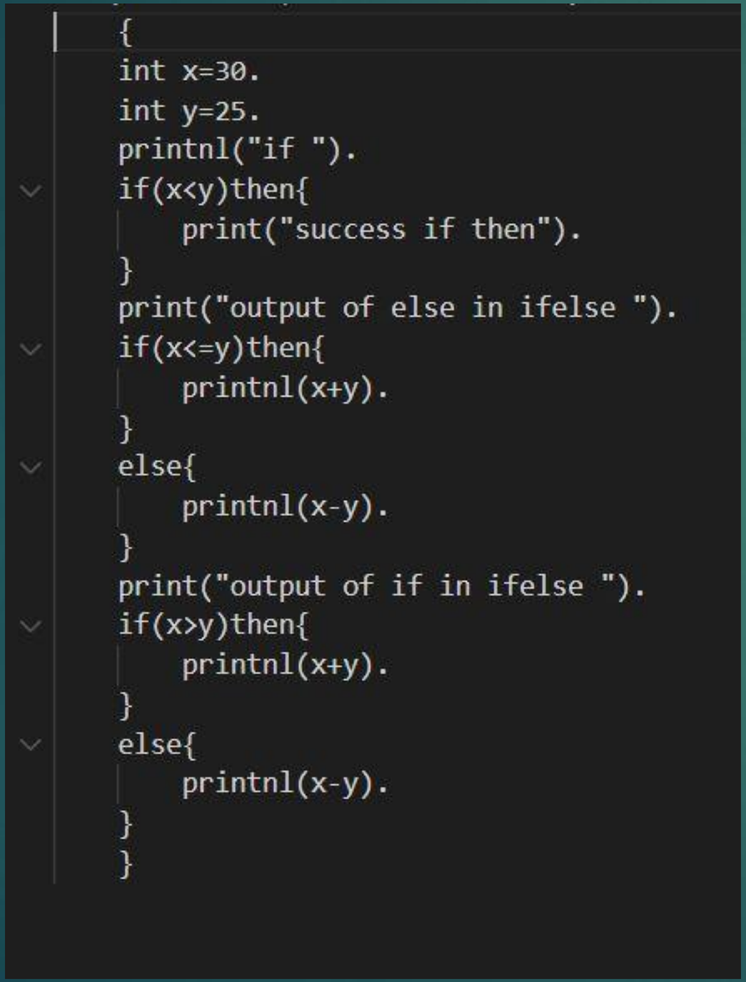
Output



```
6 ?- muvy("stringManipulation.mvy").
Reversing a String
 91 maeT - 205 RES
Concating str1 and str2 string addition
true
```

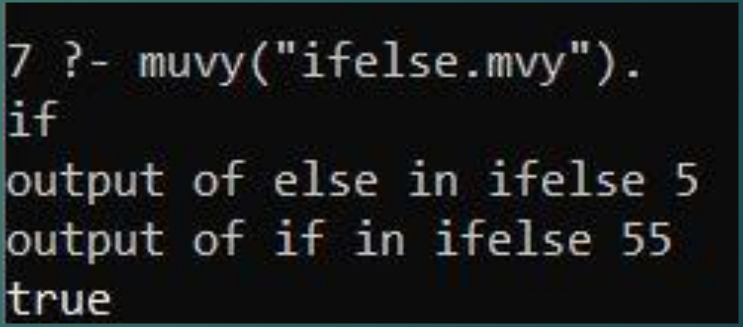
Sample run 3

Program for If else



```
{
int x=30.
int y=25.
printnl("if ").
if(x<y)then{
    print("success if then").
}
print("output of else in ifelse ").
if(x<=y)then{
    printnl(x+y).
}
else{
    printnl(x-y).
}
print("output of if in ifelse ").
if(x>y)then{
    printnl(x+y).
}
else{
    printnl(x-y).
}
}
```

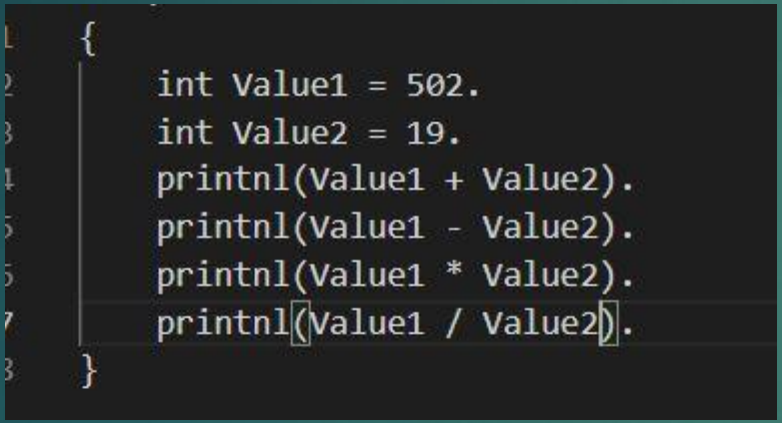
Output



```
7 ?- muvy("ifelse.mvy").
if
output of else in ifelse 5
output of if in ifelse 55
true
```

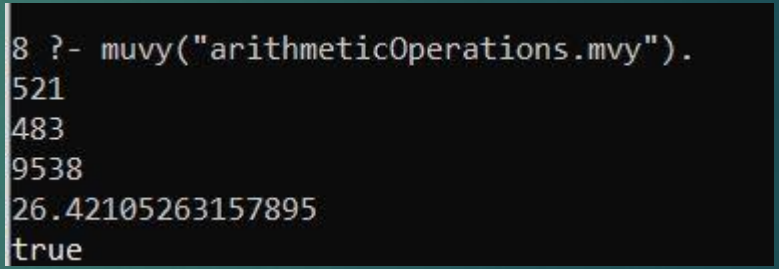
Sample run 4

Program for Arithmetic Operations



```
{
    int Value1 = 502.
    int Value2 = 19.
    printnl(Value1 + Value2).
    printnl(Value1 - Value2).
    printnl(Value1 * Value2).
    printnl(Value1 / Value2).
}
```

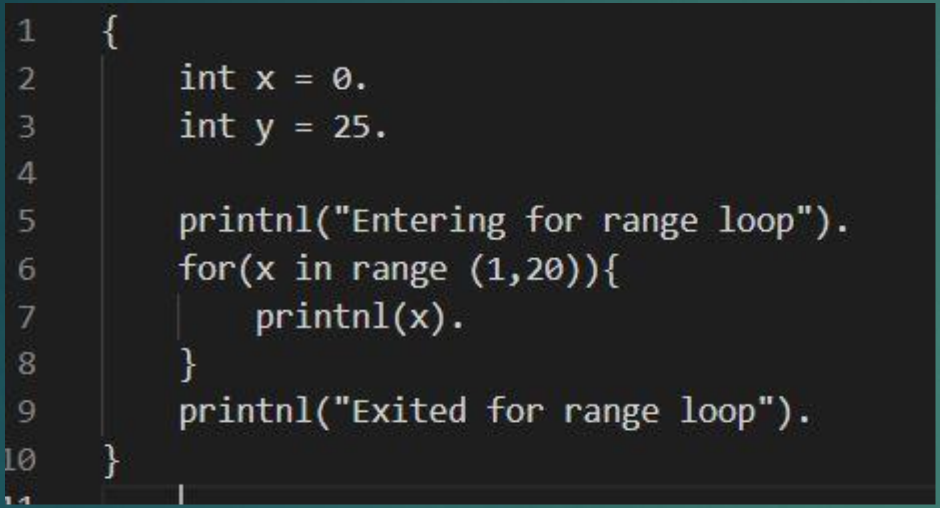
Output



```
8 ?- muvy("arithmeticOperations.mvy").
521
483
9538
26.42105263157895
true
```

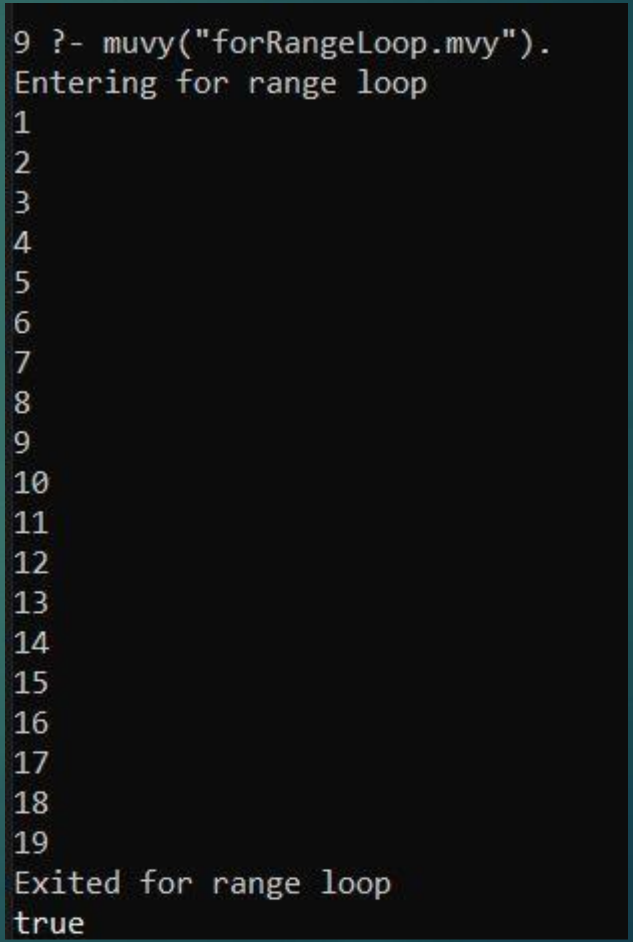
Sample run 5

Program demonstrating for in range



```
{
    int x = 0.
    int y = 25.

    println("Entering for range loop").
    for(x in range (1,20)){
        println(x).
    }
    println("Exited for range loop").
}
```

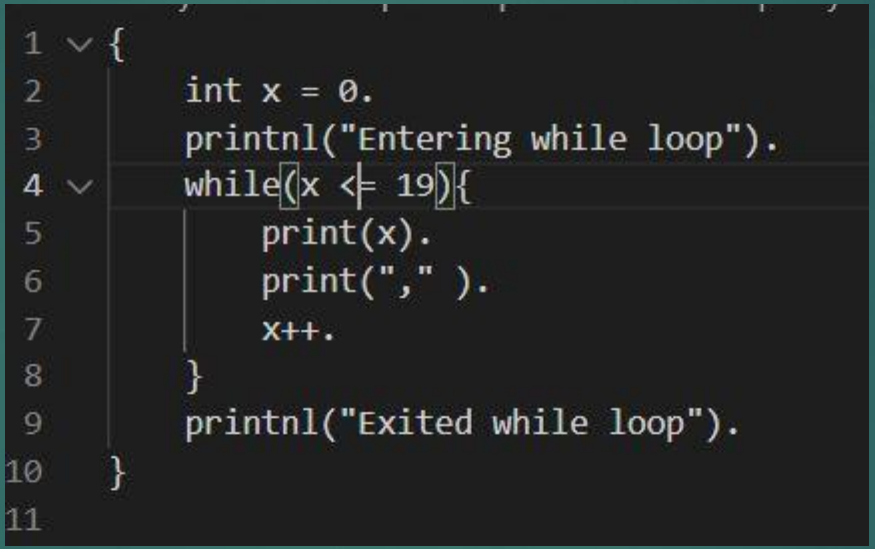
Output



```
9 ?- muvy("forRangeLoop.mvy").
Entering for range loop
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
Exited for range loop
true
```

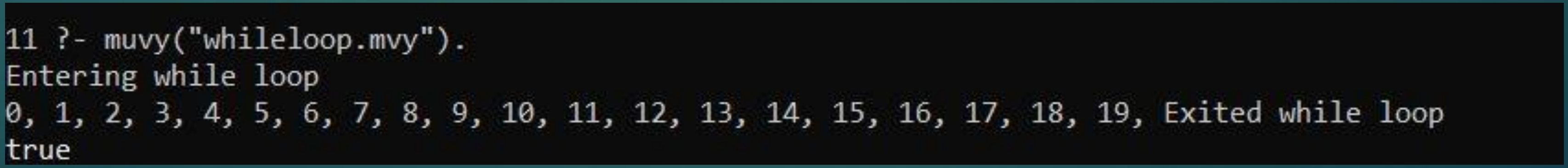
Sample run 6

Program demonstrating while loop



```
{
      int x = 0.
      println("Entering while loop").
      while(x <= 19){
          print(x).
          print("," ).
          x++.
      }
      println("Exited while loop").
}
```

Output



```
11 ?- muvy("whileloop.mvy").
Entering while loop
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, Exited while loop
true
```

THE END

GITHUB LINK : HTTPS://GITHUB.COM/HARSHAMUPPARAJU21/SER502-SPRING2023-TEAM19