Assignment 3
Sarvansh Prasher

Prolog Code

```
% @author : Sarvansh Prasher
% @version 1.0
% @created on 03-19-2020

% Predicates used :

% "program" predicate will be the main function of any programming language
which contains
% the statements and is executable.

% "block" predicate will be containing all the declarations and commands
% and all the things which are necessary before starting any program.

% "declaration" predicate will be containing all the constant and variable
declarations and later use them in block.

% "command" predicate will be containing all the commands such as if
else,do while loop
% in which we will use the variables and doing operations inside them.

% "booleanExpression" predicate will be containing all the commands such as
true,false,
% using which we will determine if any variable is equal or not.

% "arithmeticExpression" predicate will be containing all the commands such
as subtraction operation.

% "additionOperation" predicate will be containing all the commands such as
addition operation.

% "multiplicationOperation" predicate will be containing all the commands
such as multiplication operation.

% "divisionOperation" predicate will be containing all the commands such as
division operation.
```

```
% "generalOperation" predicate will be containing all the general commands
using which we will assign values to variables.

% "identifier" predicate will be used for defining identifiers used in
block.

% "number" predicate will be used for numbers used in block.

:- use_rendering(svgtree).

:- table arithmeticExpression/3, subtraction/3, division/3,
    multiplication/3, paranthesis/3.

program(t_program(Z)) --> block(Z),[.].

block(t_block(Z,Z1)) --> [begin],declaration(Z),command(Z1),[end].

declaration(t_declaration(Z,Z1,Z2)) -->
[const],identifier(Z),[=],term(Z1),[;],declaration(Z2).
declaration(t_declaration(Z,Z1)) -->
[var],identifier(Z),[;],declaration(Z1).
declaration(t_declaration()) --> [].

command(t_assign(Z,Z1,Z2)) -->
identifier(Z),[:=],arithmeticExpression(Z1),[;],command(Z2).
command(t_ifCommand(Z,Z1,Z2,Z3)) -->
[if],booleanExpression(Z),[then],command(Z1),[else],command(Z2),[endif],[;]
,command(Z3).
command(t_whileCommand(Z,Z1,Z2)) -->
[while],booleanExpression(Z),[do],command(Z1),[endwhile],[;],command(Z2).
command(t_command(Z,Z1)) --> block(Z),[;],command(Z1).
command(t_assign(Z,Z1)) --> identifier(Z),[:=],arithmeticExpression(Z1).
command(t_ifCommand(Z,Z1,Z2)) -->
[if],booleanExpression(Z),[then],command(Z1),[else],command(Z2),[endif].
command(t_whileCommand(Z,Z1)) -->
[while],booleanExpression(Z),[do],command(Z1),[endwhile].

booleanExpression(t_boolean(true)) --> [true].
booleanExpression(t_boolean(false)) --> [false].
booleanExpression(t_boolean_equal(Z,Z1)) -->
arithmeticExpression(Z),[=],arithmeticExpression(Z1).
booleanExpression(t_boolean_not(Z)) --> [not],booleanExpression(Z).
```

```
arithmeticExpression(t_expr(Z,Z1)) -->
arithmeticExpression(Z),[+],subtraction(Z1).
arithmeticExpression(Z) --> subtraction(Z).

subtraction(t_subt(Z,Z1)) --> subtraction(Z),[-],multiplication(Z1).
subtraction(Z) --> multiplication(Z).

multiplication(t_mult(Z,Z1)) --> multiplication(Z),[*],division(Z1).
multiplication(Z) --> division(Z).

division(t_div(Z,Z1)) --> division(Z),[/],paranthesis(Z1).
division(Z) --> paranthesis(Z).

paranthesis(t_parant(Z)) --> ["("] , arithmeticExpression(Z) , [")"].
paranthesis(Z) -->term(Z).

term(Z) --> identifier(Z).
term(t_term(Z)) --> [Z] ,{number(Z)}.

identifier(t_id(x)) --> [x].
identifier(t_id(y)) --> [y].
identifier(t_id(z)) --> [z].
identifier(t_id(u)) --> [u].
identifier(t_id(v)) --> [v].


%---------------------------%%%%%%%%%%%%%%%%%%%%%%%-----------------------
-------

program_eval(t_program(Z),X,Y, Val) :- block_eval(Z,[(x,X),(y,Y)],FEnv),
                                       lookup(z,FEnv,Val).

block_eval(t_block(Z,Z1),Env,FEnv)
:-declaration_eval(Z,Env,Env1),command_eval(Z1,Env1,FEnv).

declaration_eval(t_declaration(),Env,Env).

declaration_eval(t_declaration(t_id(_),Z),Env,EnvOut) :-
                        declaration_eval(Z,Env,EnvOut).

declaration_eval(t_declaration(t_id(IdentifierNode),NumberNode,Z),Env,EnvOu
```

```
t)
:-update(IdentifierNode,NumberNode, Env, Env2),
declaration_eval(Z,Env2,EnvOut).

%-------------------------------------------------------------

command_eval(t_whileCommand(BooleanNode,_),Env,Env)
                      :- boolean_eval(BooleanNode,Env,Env,false).

command_eval(t_whileCommand(BooleanNode,CommandNode),Env,EnvOut)
                  :-boolean_eval(BooleanNode,Env,FEnv,Val),
                        Val = true,
                       command_eval(CommandNode,FEnv,FEnv1),

command_eval(t_whileCommand(BooleanNode,CommandNode),FEnv1,EnvOut).

%-------------------------------------------------------------

command_eval(t_assign(t_id(IdentifierNode),ExpressionNode,Z),Env,EnvOut) :-
                                    eval_expr(ExpressionNode,Env,Env2,Val),
                                    update(IdentifierNode,Val,Env2,FEnv),

command_eval(Z,FEnv,EnvOut).

x


%-------------------------------------------------------------
command_eval(t_command(Z),Env, Val) :- block_eval(Z,Env,Val).

%-------------------------------------------------------------


command_eval(t_ifCommand(BooleanNode,CommandNode,_),Env,EnvOut)
                             :-boolean_eval(BooleanNode,Env,FEnv,Val),
                     Val = true,
                     command_eval(CommandNode,FEnv,EnvOut).

command_eval(t_ifCommand(BooleanNode,_,CommandNode),Env,EnvOut)
                             :-boolean_eval(BooleanNode,Env,FEnv,Val),
                     Val = false,!,
```

```
command_eval(t_ifCommand(BooleanNode,_,CommandNode),FEnv,EnvOut).


%-----------------------------------------------------------

boolean_eval(t_boolean(false),Env,Env,false).

boolean_eval(t_boolean(true),Env,Env,true).

boolean_eval(t_boolean_equal(Expression,Expression1),Env,FEnv,Val)

:-eval_expr(Expression,Env,FEnv1,Val1),
                                    eval_expr(Expression1,FEnv1,FEnv,Val2),
                                              eval_equal(Val1,Val2,Val).

boolean_eval(t_booolean_not(Expression),Env,FEnv,Val):-
            eval_expr(Expression,Env,FEnv,BoolOutput),
                                           eval_not(BoolOutput,Val).

eval_not(true,false).
eval_not(false,true).

eval_equal(Val1,Val2,true) :- Val1 = Val2.
eval_equal(Val1,Val2,false) :- Val1 \= Val2.

eval_expr(t_expr(NumNode,TermNode),Env,FEnv,Val) :-
    eval_expr(NumNode,Env,Env1,Val1),eval_expr(TermNode,Env1,FEnv,Val2),
                                           Val is Val1+Val2.

eval_expr(t_subt(NumNode,TermNode),Env,FEnv,Val) :-
    eval_expr(NumNode,Env,Env1,Val1),eval_expr(TermNode,Env1,FEnv,Val2),
                                           Val is Val1-Val2.


eval_expr(t_mult(NumNode,TermNode),Env,FEnv,Val) :-
    eval_expr(NumNode,Env,Env1,Val1),eval_expr(TermNode,Env1,FEnv,Val2),
                                           Val is Val1*Val2.

eval_expr(t_div(NumNode,TermNode),Env,FEnv,Val) :-
    eval_expr(NumNode,Env,Env1,Val1),eval_expr(TermNode,Env1,FEnv,Val2),
                                           Val is Val1/Val2.
```

```
eval_expr(t_parant(NumNode),Env,FEnv,Val) :-
eval_expr(NumNode,Env,FEnv,Val).

eval_expr(t_term(Num),Env,Env,Num).

eval_expr(t_id(Identifier),Env,Env,Val) :- lookup(Identifier,Env,Val).

lookup(_,[],0).
lookup(Key, [(Key,Val)|_],Val).
lookup(Key, [_|Tail],Val) :- lookup(Key,Tail,Val).

update(Key,Val,[],[(Key,Val)]).
update(Key,Val,[(Key,_)|Tail],[(Key,Val)|Tail]).
update(Key,Val,[Head|Tail],[Head|FEnv]) :- Head \=
(Key,_),update(Key,Val,Tail,FEnv).
```

Output

1. ?- program(P, [begin, var, z, ; , var, x, ;, z, :=, x, end, .], []),write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(z), *t_declaration*(*t_id*(x), *t_declaration*())), *t_assign*(*t_id*(z), *t_id*(x)))),
Z = 2

2. ?- program(P, [begin, var, x,;, var, y,;, var, z,;, z,:=,x,+,y, end,.], []),write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(y), *t_declaration*(*t_id*(z), *t_declaration*()))), *t_assign*(*t_id*(z), *t_expr*(*t_id*(x), *t_id*(y))))),
Z = 5

3. ?- program(P, [begin, var, x,;, var, y,;, var, z,;, z,:=,'(',z,:=,x,+,2,')',+,y, end,.], []),write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(y), *t_declaration*(*t_id*(z), *t_declaration*()))), *t_assign*(*t_id*(z), *t_expr*(*t_parant*(*t_expressionOfExpression*(*t_id*(z), *t_expr*(*t_id*(x), *t_term*(2)))), *t_id*(y))))),
Z = 7

4. ?- program(P, [begin, var, x,;, var, y,;, var, z,;, if, x,=,y, then, z,:=,1, else, z,:=,0, endif, end,.], []), write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(y), *t_declaration*(*t_id*(z), *t_declaration*()))), *t_ifCommand*(*t_boolean_equal*(*t_id*(x), *t_id*(y)), *t_assign*(*t_id*(z), *t_term*(1)), *t_assign*(*t_id*(z), *t_term*(0))))),
Z = 0

5. ?- program(P, [begin, var, x,;, var, y,;, var, z,;, if, x, =, 0, then, z,:=,x, else, z,:=,y, endif, end,.], []), write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(y), *t_declaration*(*t_id*(z), *t_declaration*()))), *t_ifCommand*(*t_boolean_equal*(*t_id*(x), *t_term*(0)), *t_assign*(*t_id*(z), *t_id*(x)), *t_assign*(*t_id*(z), *t_id*(y))))) ,
Z = 3

6. ?- program(P, [begin, var, x,;, var, y,;, var, z,;, if, not, x,=,y, then, z,:=,x, else, z,:=,y, endif, end,.], []), write(P),program_eval(P, 2, 3, Z).
P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(y), *t_declaration*(*t_id*(z), *t_declaration*()))), *t_ifCommand*(*t_boolean_not*(*t_boolean_equal*(*t_id*(x), *t_id*(y))), *t_assign*(*t_id*(z), *t_id*(x)), *t_assign*(*t_id*(z), *t_id*(y))))),
Z = 2

7. ?- program(P, [begin, var, x,;, var, z,;, z,:=,0,;, while, not, x,=,0, do, z, :=, z,+,1,;, x,:=,x,-,1, endwhile, end,.], []),write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(z), *t_declaration*())), *t_assign*(*t_id*(z), *t_term*(0), *t_whileCommand*(*t_boolean_not*(*t_boolean_equal*(*t_id*(x), *t_term*(0))), *t_assign*(*t_id*(z), *t_expr*(*t_id*(z), *t_term*(1)), *t_assign*(*t_id*(x), *t_subt*(*t_id*(x), *t_term*(1)))))))),
Z = 2

8. ?-program(P, [begin, var, x,;, var, y,;, var, z,;, z,:=,1,;, u,:=,x,;, while, not, u, =, 0, do, z, :=,z,*,y,;, u,:=,u,-,1, endwhile, end,.], []), write(P),program_eval(P, 2, 3, Z).

P = *t_program*(*t_block*(*t_declaration*(*t_id*(x), *t_declaration*(*t_id*(y), *t_declaration*(*t_id*(z), *t_declaration*()))), *t_assign*(*t_id*(z), *t_term*(1), *t_assign*(*t_id*(u), *t_id*(x), *t_whileCommand*(*t_boolean_not*(*t_boolean_equal*(*t_id*(u), *t_term*(0))), *t_assign*(*t_id*(z), *t_mult*(*t_id*(z), *t_id*(y)), *t_assign*(*t_id*(u), *t_subt*(*t_id*(u), *t_term*(1)))))))),
Z = 9