CS 8803: Compilers: theory and practice
Project Phase 1: Front end

Testing and output report

Team: Gang Liao, Rachna Saxena

This document describes the test output of Tiger language programs. In the first phase of project, front end components like parser table of Tiger, symbol table, semantic analysis and IR code generator are built and tested. This report contains test case outputs for parser output, symbol table and IR code generation. Below table describes the purpose and result of different each test cases.

Test cases	Test scenario	Status	Comment
Test1.tiger	Type and variable declaration, for loop, call to library function printi	Pass	
Test2.tiger	Type declaration, function definition, function call	Pass	
Test3.tiger	Variable declaration (float type), function definition, function call	Pass	
Test4.tiger	Various type and variable declarations, simple arithmetic expressions.	Pass	
Test5.tiger	Variable declaration, if-then-else statement	Pass	
Test6.tiger	Wrong function return	Pass	Negative test case, error gets generated
Test7.tiger	Wrong function return	Pass	Negative test case, error gets generated
Test8.tiger	Multiple function parameters	Pass	
Test9.tiger	Different data types in comparison	Pass	

Test10.tiger	Mismatch type of function parameters and calling parameters	Pass	Negative test case, error gets generated
Test11.tiger	Mismatch number of function parameters and calling parameters	Pass	Negative test case, error gets generated
Test12.tiger	Test multiple if-then-else statements	Pass	
Test13.tiger	Test nested if-the-else statements	Pass	
Test14.tiger	Test same type in comparison	Pass	
Test15.tiger	Test different types in comparison (int and float comparison).	Pass	Negative test case, error gets generated
Test16.tiger	Test if-then conditional statement	Pass	
Test17.tiger			
Test18.tiger			
Test19.tiger	Error nous comparison operator	Pass	Negative test case, error gets generated
Test20.tiger	Test for logical operators	Pass	
Test21.tiger	Test for break in for loop	Pass	
Test22.tiger	Test for array store operation	Pass	
Test23.tiger	Test for array access, storage and function call	Pass	
Test24.tiger	Complex test cases for multiple loops scenario	Pass	
Test25.tiger	Test for printi with float value	Pass	Negative test case, error gets generated
Test26.tiger	Calculate and print factorial of a number	Pass	

Test1.tiger

```
Let
/* Declare ArrayInt as a new type */
        type ArrayInt = array [100] of int;
/* Declare vars X and Y as arrays with initialization */
        type me = int;
        var X, Y : ArrayInt := 10;
        var i, sum : int := 0;
in
        for i := 0 to 100 do /* for loop for dot product */
                sum := sum + X[i] * Y [i];
        enddo;
        printi(sum); /* library call to printi to print the dot
product */
end
```

Output:

```
[ RUN ] parsing code...

let type id = array [ intlit ] of int ; type <13, "type"><45,
"ArrayInt"><36, "="><0, "array"><27, "["><46, "100"><28,
"]"><10, "of"><20, "int"><24, ";">id = int ; var <13,
"type"><45, "me"><36, "="><20, "int"><24, ";">
id , id : id := intlit ; var <14, "var"><45, "X"><22, ","><45,
"Y"><23, ":"><45, "ArrayInt"><44, ":="><46, "10"><24, ";">
id , id : int := intlit ; in <14, "var"><45, "i"><22, ","><45,
"sum"><23, ":"><20, "int"><44, ":="><46, "0"><24, ";">
for id := intlit to <46, "0"> intlit do <46, "100">
id := id + id [ id ] * id [ id ] ; <45, "sum"><44, ":="><45,
"sum"><32, "+"><45, "$t0"><34, "*"><45, "$t1">
enddo ; id ( id ) ; <45, "printi"><25, "("><45, "sum"><26,
")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: X
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: Y
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: i
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: sum
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: ArrayInt
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: me
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
```

```
[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, X, 100, 10
    assign, Y, 100, 10
    assign, i, 0,
    assign, sum, 0,
main:
    assign, i, 0,
loop_label1:
    brgt, i, 100, loop_label0
    array_load, $t0, X, i
    array_load, $t1, Y, i
    mult, $t1, $t0, $t2
    add, $t2, sum, $t3
    assgin, sum, $t3,
    add, i, 1, i
    goto, loop_label1, ,
loop_label0:
    call, printi, sum
    return, , ,
----------------------------------------
```

Test2. tiger

```
let

        type ArrayInt = array [100] of int;

        function print ( n : int) begin

                printi(n);

        end;

in

        print(5);

end
```

Output:

```
[ RUN ] parsing code...

let type id = array [ intlit ] of int ; function <13, "type"><45,
"ArrayInt"><36, "="><0, "array"><27, "["><46, "100"><28, "]"><10,
"of"><20, "int"><24, ";">
id ( id : int ) begin <6, "function"><45, "print"><25, "("><45,
"n"><23, ":"><20, "int"><26, ")">
id ( id ) ; <45, "printi"><25, "("><45, "n"><26, ")">
end ; in id ( intlit ) ; <45, "print"><25, "("><46, "5"><26, ")">
end
----------------------------------------
Table: Types
Name: ArrayInt
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: print
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [n]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
```

```
----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
print:
    call, printi, n
    return, , ,
main:
    call, print, 5
    return, , ,
----------------------------------------
```

Test3.tiger

```
let
      var X, Y : float := 0.0;
      function print (X : int) begin
            printi(X);
      end;
in
      print(5);
      X := 1.0;
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : float := floatlit ; function <14, "var"><45,
"X"><22, ","><45, "Y"><23, ":"><21, "float"><44, ":="><47,
"0.0"><24, ";">
id ( id : int ) begin <6, "function"><45, "print"><25, "("><45,
"X"><23, ":"><20, "int"><26, ")">
id ( id ) ; <45, "printi"><25, "("><45, "X"><26, ")">
end ; in id ( intlit ) ; <45, "print"><25, "("><46, "5"><26, ")">
id := floatlit ; <45, "X"><44, ":="><47, "1.0">
end

----------------------------------------
Table: Variables
Name: X
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: Y
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
```

```
----------------------------------------
Table: Functions
Name: print
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [X]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, X, 0.0,
    assign, Y, 0.0,
print:
    call, printi, X
    return, , ,
main:
    call, print, 5
    assgin, X, 1.0,
    return, , ,
----------------------------------------
```

Test4.tiger

```
let
      type First_Int = int;
      type Second_Int = First_Int;
      var X : First_Int := 0;
      var Y : Second_Int;
      var A : int := 0;
      var B : float := 0.1;
in
      Y := Y + X;
      A := A + B;
end
```

Output:

```
[ RUN ] parsing code...

let type id = int ; type <13, "type"><45, "First_Int"><36,
"="><20, "int"><24, ";">
id = id ; var <13, "type"><45, "Second_Int"><36, "="><45,
"First_Int"><24, ";">
id : id := intlit ; var <14, "var"><45, "X"><23, ":"><45,
"First_Int"><44, ":="><46, "0"><24, ";">
id : id ; var <14, "var"><45, "Y"><23, ":"><45,
"Second_Int"><24, ";">
id : int := intlit ; var <14, "var"><45, "A"><23, ":"><20,
"int"><44, ":="><46, "0"><24, ";">
id : float := floatlit ; in <14, "var"><45, "B"><23, ":"><21,
"float"><44, ":="><47, "0.1"><24, ";">
id := id + id ; <45, "Y"><44, ":="><45, "Y"><32, "+"><45,
"X">
id := id + id ; <45, "A"><44, ":="><45, "A"><32, "+"><45,
"B">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: A
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: B
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: X
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: Y
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: First_Int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: Second_Int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, X, 0,
    assign, Y, 0,
    assign, A, 0,
    assign, B, 0.1,
main:
    add, X, Y, $t0
    assgin, Y, $t0,
    add, B, A, $t1
    assgin, A, $t1,
    return, , ,
----------------------------------------
```

Test5.tiger

```
let
        var a, b : int := 0;
in


        if(a = b) then
                a := b + 2;
            else
                 a := 2;
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : int := intlit ; in <14, "var"><45, "a"><22,
","><45, "b"><23, ":"><20, "int"><44, ":="><46, "0"><24, ";">
if ( id = id ) then id := id + intlit ; <45, "a"><44, ":="><45,
"b"><32, "+"><46, "2">
else id := intlit ; <45, "a"><44, ":="><46, "2">
endif ; id ( id ) ; <45, "printi"><25, "("><45, "a"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    add, 2, b, $t1
    assgin, a, $t1,
    goto, if_label2, ,
if_label0:
    assgin, a, 2,
if_label2:
    call, printi, a
    return, , ,
----------------------------------------
```

Test6.tiger

```
let
        var x : int;
        function print ( n : int ) : int   /* integer return type */
            begin
            printi(n);
            return 10;
        end;
in
/* return value is captured in integer variable */
        x = print(5);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int ; function <14, "var"><45, "x"><23,
":"><20, "int"><24, ";">
id ( id : int ) : int begin id ( id ) ; <45,
"printi"><25, "("><45, "n"><26, ")">
return intlit ; <46, "10">
end ; in id = testCases/test-phaseI/test6.tiger line 9:
x = doesn't support token: =


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, x, 0,
print:
    call, printi, n
    return, , ,
main:
----------------------------------------
```

Test7.tiger

```
let
            var x : int;
/* return type is float, this should generate return type mismatch error */
        function print ( n : int ) : float
        begin
                printi(n);
                return 10.0;
        end;
in
/* return value is captured in int variable*/
        x = print(5);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int ; function <14, "var"><45, "x"><23,
":"><20, "int"><24, ";">
id ( id : int ) : float begin id ( id ) ; <45,
"printi"><25, "("><45, "n"><26, ")">
return floatlit ; <47, "10.0">
end ; in id = testCases/test-phaseI/test7.tiger line 9:
x = doesn't support token: =


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, x, 0,
print:
    call, printi, n
    return, , ,
main:
----------------------------------------
```

Test8.tiger

```
let
        var x : int;
        /* integer return type; with multiple function arguments */
        function print ( n : int, m: int ) : int
        begin
                printi(n);
                printi(m);
                return 10;
        end;
in
/* return value is captured in integer variable */
        x := print(5,6);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int ; function <14, "var"><45, "x"><23, ":"><20,
"int"><24, ";">
id ( id : int , id : int ) : int begin id ( id ) ; <45,
"printi"><25, "("><45, "n"><26, ")">
id ( id ) ; <45, "printi"><25, "("><45, "m"><26, ")">
return intlit ; end ; in id := id ( intlit , intlit ) ; <45,
"x"><44, ":="><45, "print"><25, "("><46, "5"><22, ","><46,
"6"><26, ")">
end
----------------------------------------
Table: Variables
Name: x
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
----------------------------------------
Table: Functions
Name: print
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [n,m]
Parameter types: [int,int]
Parameter dimensions: [0,0]
Return type: int
----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------

[ OK ] successful parse...

----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, x, 0,
print:
    call, printi, n
    call, printi, m
    return, 10, ,
main:
    callr, x, print, 5, 6
    return, , ,
----------------------------------------
```

Test9.tiger

```
let
        var a : int := 0;
            var b : float := 0;
in
/* this should generate error a is int and b is float*/
        if(a = b) then
           a := b + 2;
        else
            a := 2;
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int := intlit ; var <14, "var"><45, "a"><23,
":"><20, "int"><44, ":="><46, "0"><24, ";">
id : float := intlit ; in <14, "var"><45, "b"><23, ":"><21,
"float"><44, ":="><46, "0"><24, ";">
if ( id = id ) then id := id + intlit ; <45, "a"><44, ":="><45,
"b"><32, "+"><46, "2">
Error: left and right type between assignment is mismatched!

else id := intlit ; <45, "a"><44, ":="><46, "2">
endif ; id ( id ) ; <45, "printi"><25, "("><45, "a"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    add, 2, b, $t1
    assgin, a, $t1,
    goto, if_label2, ,
if_label0:
    assgin, a, 2,
if_label2:
    call, printi, a
    return, , ,
----------------------------------------
```

Test10.tiger

```
let
            var x : int;
/* integer return type; with multiple function arguments */
        function print ( n : int, m: int ) : int
         begin
                printi(n);
                printi(m);
                return 10;
        end;
in
/* this should generate error. Function parameter type mismatch*/
        x := print(5,6.0);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int ; function <14, "var"><45, "x"><23, ":"><20,
"int"><24, ";">
id ( id : int , id : int ) : int begin id ( id ) ; <45,
"printi"><25, "("><45, "n"><26, ")">
id ( id ) ; <45, "printi"><25, "("><45, "m"><26, ")">
return intlit ; end ; in id := id ( intlit , floatlit ) ; <45,
"x"><44, ":="><45, "print"><25, "("><46, "5"><22, ","><47,
"6.0"><26, ")">

 Error: 6.0: float mismatched to function print parameter: int
```

Test11.tiger

```
let
        var x : int;
/* integer return type; with multiple function arguments */
        function print ( n : int, m: int ) : int
        begin
           printi(n);
           printi(m);
           return 10;
        end;
in
/* this should generate error. Number of Function parameters are different */
        x := print(5);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int ; function <14, "var"><45, "x"><23, ":"><20,
"int"><24, ";">
id ( id : int , id : int ) : int begin id ( id ) ; <45,
"printi"><25, "("><45, "n"><26, ")">
id ( id ) ; <45, "printi"><25, "("><45, "m"><26, ")">
return intlit ; end ; in id := id ( intlit ) ; <45, "x"><44,
":="><45, "print"><25, "("><46, "5"><26, ")">

Error: function print parameter numbers is not matched!
```

Test12.tiger

```
/* testing multiple if-then-else statement */
let
        var a, b : int := 0;
in
        if(a = b) then
                a := b + 2;
      else
      a := 2;
        endif;
        if(a >= b) then
                a := b + 2;
      else
      a := 2;
        endif;
        printi(a);
end
   printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : int := intlit ; in <14, "var"><45,
"a"><22, ","><45, "b"><23, ":"><20, "int"><44,
":="><46, "0"><24, ";">
if ( id = id ) then id := id + intlit ; <45, "a"><44,
":="><45, "b"><32, "+"><46, "2">
else id := intlit ; <45, "a"><44, ":="><46, "2">
endif ; if ( id >= id ) then id := id + intlit ; <45,
"a"><44, ":="><45, "b"><32, "+"><46, "2">
else id := intlit ; <45, "a"><44, ":="><46, "2">
endif ; id ( id ) ; <45, "printi"><25, "("><45,
"a"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
```

```
[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    add, 2, b, $t1
    assgin, a, $t1,
    goto, if_label2, ,
if_label0:
    assgin, a, 2,
if_label2:
    assign, $t2, 0,
    brlt, b, a, if_label4
    assign, $t2, 1,
if_label4:
    breq, $t2, 0, if_label3
    add, 2, b, $t3
    assgin, a, $t3,
    goto, if_label5, ,
if_label3:
    assgin, a, 2,
if_label5:
    call, printi, a
    return, , ,
----------------------------------------
```

Test13.tiger

```
/* testing nested if then else statement */
let
        var a, b, c : int := 0;
in
        if(a = b) then
      if (a > 0) then
                a := b + 2;
      else
      a := b;
            endif;
      else
      a := 2;
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id , id : int := intlit ; in <14, "var"><45,
"a"><22, ","><45, "b"><22, ","><45, "c"><23, ":"><20,
"int"><44, ":="><46, "0"><24, ";">
if ( id = id ) then if ( id > intlit ) then id := id +
intlit ; <45, "a"><44, ":="><45, "b"><32, "+"><46, "2">
else id := id ; <45, "a"><44, ":="><45, "b">
endif ; else id := intlit ; <45, "a"><44, ":="><46, "2">
endif ; id ( id ) ; <45, "printi"><25, "("><45, "a"><26,
")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: c
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
```

```
[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
    assign, c, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    assign, $t1, 0,
    brleq, 0, a, if_label3
    assign, $t1, 1,
if_label3:
    breq, $t1, 0, if_label2
    add, 2, b, $t2
    assgin, a, $t2,
    goto, if_label4, ,
if_label2:
    assgin, a, b,
if_label4:
    goto, if_label5, ,
if_label0:
    assgin, a, 2,
if_label5:
    call, printi, a
    return, , ,
----------------------------------------
```

Test14.tiger

```
/* testing complex expression */
let
        var a, b, c, d : int := 0;
in
        if(a = b) then
                a := b + ( 5 * a) - ( 3 + c);
       else
      a := (b / 2) + (d * 3);
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id , id , id : int := intlit ; in <14,
"var"><45, "a"><22, ","><45, "b"><22, ","><45, "c"><22,
","><45, "d"><23, ":"><20, "int"><44, ":="><46,
"0"><24, ";">
if ( id = id ) then id := id + ( intlit * id ) - (
intlit + id ) ; <45, "a"><44, ":="><45, "b"><32,
"+"><25, "("><46, "5"><34, "*"><45, "a"><26, ")"><33,
"-"><25, "("><46, "3"><32, "+"><45, "c"><26, ")">
else id := ( id / intlit ) + ( id * intlit ) ; <45,
"a"><44, ":="><25, "("><45, "b"><35, "/"><46, "2"><26,
")"><32, "+"><25, "("><45, "d"><34, "*"><46, "3"><26,
")">
endif ; id ( id ) ; <45, "printi"><25, "("><45,
"a"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t4
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t5
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t6
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t7
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: c
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: d
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int

----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
[ OK ] successful parse...
----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
    assign, c, 0,
    assign, d, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    mult, a, 5, $t1
    add, $t1, b, $t2
    add, c, 3, $t3
    sub, $t3, $t2, $t4
    assgin, a, $t4,
    goto, if_label2, ,
if_label0:
    div, 2, b, $t5
    mult, 3, d, $t6
    add, $t6, $t5, $t7
    assgin, a, $t7,
if_label2:
    call, printi, a
    return, , ,
----------------------------------------
```

Test15.tiger

```
/* testing complex expression of mixed types, int and float */
let
        var a, b : int := 0;
            var c, d  : float := 0;
in
        if(a = b) then
                a := b + ( 5 * a) - ( 3 + c);
        else
                a := (b / 2) + (d * 3);
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : int := intlit ; var <14, "var"><45,
"a"><22, ","><45, "b"><23, ":"><20, "int"><44,
":="><46, "0"><24, ";">
id , id : float := intlit ; in <14, "var"><45, "c"><22,
","><45, "d"><23, ":"><21, "float"><44, ":="><46,
"0"><24, ";">
if ( id = id ) then id := id + ( intlit * id ) - (
intlit + id ) ; <45, "a"><44, ":="><45, "b"><32,
"+"><25, "("><46, "5"><34, "*"><45, "a"><26, ")"><33,
"-"><25, "("><46, "3"><32, "+"><45, "c"><26, ")">
else id := ( id / intlit ) + ( id * intlit ) ; <45,
"a"><44, ":="><25, "("><45, "b"><35, "/"><46, "2"><26,
")"><32, "+"><25, "("><45, "d"><34, "*"><46, "3"><26,
")">
endif ; id ( id ) ; <45, "printi"><25, "("><45,
"a"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t4
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t5
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t6
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t7
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: c
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: d
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
[ OK ] successful parse...
----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
    assign, c, 0,
    assign, d, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    mult, a, 5, $t1
    add, $t1, b, $t2
    add, c, 3, $t3
    sub, $t3, $t2, $t4
    assgin, a, $t4,
    goto, if_label2, ,
if_label0:
    div, 2, b, $t5
    mult, 3, d, $t6
    add, $t6, $t5, $t7
    assgin, a, $t7,
if_label2:
    call, printi, a
    return, , ,
----------------------------------------
```

Test16.tiger

```
/* testing if-then statement */
let
        var a, b : int := 0;
in
        if(a = b) then
                a := b + 2;
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : int := intlit ; in <14, "var"><45,
"a"><22, ","><45, "b"><23, ":"><20, "int"><44,
":="><46, "0"><24, ";">
if ( id = id ) then id := id + intlit ; <45, "a"><44,
":="><45, "b"><32, "+"><46, "2">
endif ; id ( id ) ; <45, "printi"><25, "("><45,
"a"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
main:
    assign, $t0, 0,
    brneq, b, a, if_label1
    assign, $t0, 1,
if_label1:
    breq, $t0, 0, if_label0
    add, 2, b, $t1
    assgin, a, $t1,
if_label0:
    call, printi, a
    return, , ,
----------------------------------------
```

Test19.tiger

```
/* error nous comparison operator */
let
        var a, b : int := 0;
        var c : int := 0;
in
        if(a = b = c) then
                a := b + 2;
       else
      a := 2;
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : int := intlit ; var <14, "var"><45,
"a"><22, ","><45, "b"><23, ":"><20, "int"><44,
":="><46, "0"><24, ";">
id : int := intlit ; in <14, "var"><45, "c"><23,
":"><20, "int"><44, ":="><46, "0"><24, ";">
if ( id = id = id ) then
Error: if boolean operation exists in if or while
condition statement,it must be the last operation in
this expression! for example, if (a + b >= c * d) is
correct.
```

Test20.tiger

```
/* test logical operators */
let
        var a, b : int := 0;
        var c : int := 0;
in
        if(a & c) then
                a := b | 2;
        else
                a := 2;
        endif;
        printi(a);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id : int := intlit ; var <14, "var"><45,
"a"><22, ","><45, "b"><23, ":"><20, "int"><44,
":="><46, "0"><24, ";">
id : int := intlit ; in <14, "var"><45, "c"><23,
":"><20, "int"><44, ":="><46, "0"><24, ";">
if ( id & id ) then id := id | intlit ; <45, "a"><44,
":="><45, "b"><43, "|"><46, "2">
else id := intlit ; <45, "a"><44, ":="><46, "2">
endif ; id ( id ) ; <45, "printi"><25, "("><45,
"a"><26, ")">
end
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: a
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: b
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: c
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -

----------------------------------------
Table: Functions
Name: flush
----------------------------------------
```

```
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
[ OK ] successful parse...
----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, a, 0,
    assign, b, 0,
    assign, c, 0,
main:
    and, c, a, $t0
    assign, $t1, 0,
    brneq, $t0, 0, if_label1
    assign, $t1, 1,
if_label1:
    breq, $t1, 0, if_label0
    or, 2, b, $t2
    assgin, a, $t2,
    goto, if_label2, ,
if_label0:
    assgin, a, 2,
if_label2:
    call, printi, a
    return, , ,
----------------------------------------
```

Test21.tiger

```
Let /* Declare ArrayInt as a new type */
   type ArrayInt = array [100] of int;
   type me = int;
/* Declare vars X and Y as arrays with initialization */
        var X, Y : ArrayInt := 10;
        var i, sum : int := 0;
in
        for i := 0 to 100 do /* for loop for dot product */
           sum := sum + X[i] * Y [i];
           if(i = 50) then
              break;
           endif;
        enddo;
        printi(sum); /* library call to printi to print the dot
product */
end
```

Output:

```
[ RUN ] parsing code...

let type id = array [ intlit ] of int ; type <13,
"type"><45, "ArrayInt"><36, "="><0, "array"><27, "["><46,
"100"><28, "]"><10, "of"><20, "int"><24, ";">
id = int ; var <13, "type"><45, "me"><36, "="><20,
"int"><24, ";">
id , id : id := intlit ; var <14, "var"><45, "X"><22,
","><45, "Y"><23, ":"><45, "ArrayInt"><44, ":="><46,
"10"><24, ";">
id , id : int := intlit ; in <14, "var"><45, "i"><22,
","><45, "sum"><23, ":"><20, "int"><44, ":="><46, "0"><24,
";"> for id := intlit to <46, "0">
intlit do <46, "100">
id := id + id [ id ] * id [ id ] ; <45, "sum"><44,
":="><45, "sum"><32, "+"><45, "$t0"><34, "*"><45, "$t1">
if ( id = intlit ) then break ; endif ; enddo ; id ( id ) ;
<45, "printi"><25, "("><45, "sum"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t4
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: X
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: Y
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: i
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: sum
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: ArrayInt
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: me
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
```

Test22.tiger

```
/* test case for array store operation */
Let
/* Declare ArrayInt as a new type */
    type ArrayInt = array [100] of int;
/* Declare vars X and Y as arrays with initialization */
    var X, Y : ArrayInt := 100;
    var sum :  ArrayInt := 0; /* Declare out array*/
    var i : int := 0;
in
        for i := 0 to 100 do /* for loop for dot product */
                sum[i] := X[i] * Y [i];
        enddo;
      for i := 0 to 100 do /* for loop for print */
/* library call to printi to print the dot product */
            printi(sum[i]);
     enddo;
end
```

Output:

```
[ RUN ] parsing code...
let type id = array [ intlit ] of int ; var <13, "type"><45,
"ArrayInt"><36, "="><0, "array"><27, "["><46, "100"><28,
"]"><10, "of"><20, "int"><24, ";">
id , id : id := intlit ; var <14, "var"><45, "X"><22,
","><45, "Y"><23, ":"><45, "ArrayInt"><44, ":="><46,
"100"><24, ";"> id : id := intlit ; var <14, "var"><45,
"sum"><23, ":"><45, "ArrayInt"><44, ":="><46, "0"><24, ";">
id : int := intlit ; in <14, "var"><45, "i"><23, ":"><20,
"int"><44, ":="><46, "0"><24, ";">
for id := intlit to <46, "0">
intlit do <46, "100"> id [ id ] := id [ id ] * id [ id ] ;
<45, "$t0"><44, ":="><45, "$t1"><34, "*"><45, "$t2">
enddo ; for id := intlit to <46, "0">
intlit do <46, "100"> id ( id [ id ] ) ; <45, "printi"><25,
"("><45, "$t4"><26, ")">
enddo ; end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t4
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: X
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: Y
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: i
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: sum
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: ArrayInt
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
```

```
[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, X, 100, 100
    assign, Y, 100, 100
    assign, sum, 100, 0
    assign, i, 0,
main:
    assign, i, 0,
loop_label1:
    brgt, i, 100, loop_label0
    array_load, $t0, sum, i
    array_load, $t1, X, i
    array_load, $t2, Y, i
    mult, $t2, $t1, $t3
    assgin, $t0, $t3,
    add, i, 1, i
    goto, loop_label1, ,
loop_label0:
    assign, i, 0,
loop_label3:
    brgt, i, 100, loop_label2
    array_load, $t4, sum, i
    call, printi, $t4
    add, i, 1, i
    goto, loop_label3, ,
loop_label2:
    return, , ,
----------------------------------------
```

Test23.tiger

```
/**
 * This file mainly tests array access and storage, as well as a
function call.
 * Its output isn't useful, but can easily be verified by running the
program mentally.
 */


let
      type ArrayInt = array[2] of int;


      var buffer : ArrayInt;
      var y : int := 7;
      var z : int := 20;


      function fillArray( firstInt : int, secondInt: int)
        begin
            buffer[0] := firstInt;
            buffer[1] := secondInt;
        end;
in
      fillArray(y, z);


      /* essentially, buffer[1] *= buffer[0]. just testing complex code
generation */
      buffer[1] := buffer[0 + 1 + 9 * 2 - 19] * buffer[1];


      printi(buffer[0]);
      printi(buffer[1]);
end
```

```
[ RUN ] parsing code...

let type id = array [ intlit ] of int ; var <13, "type"><45,
"ArrayInt"><36, "="><0, "array"><27, "["><46, "2"><28,
"]"><10, "of"><20, "int"><24, ";">
id : id ; var <14, "var"><45, "buffer"><23, ":"><45,
"ArrayInt"><24, ";">
id : int := intlit ; var <14, "var"><45, "y"><23, ":"><20,
"int"><44, ":="><46, "7"><24, ";">
id : int := intlit ; function <14, "var"><45, "z"><23,
":"><20, "int"><44, ":="><46, "20"><24, ";">
id ( id : int , id : int ) begin id [ intlit ] := id ; <45,
"$t0"><44, ":="><45, "firstInt">
id [ intlit ] := id ; <45, "$t1"><44, ":="><45, "secondInt">
end ; in id ( id , id ) ; <45, "fillArray"><25, "("><45,
"y"><22, ","><45, "z"><26, ")">
id [ intlit ] := id [ intlit + intlit + intlit * intlit -
intlit ] * id [ intlit ] ; <45, "$t2"><44, ":="><45,
"$t7"><34, "*"><45, "$t8">
id ( id [ intlit ] ) ; <45, "printi"><25, "("><45,
"$t10"><26, ")">
id ( id [ intlit ] ) ; <45, "printi"><25, "("><45,
"$t11"><26, ")">
end
----------------------------------------
Table: Variables
Name: $t10
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t11
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t4
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t5
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t6
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t7
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t8
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t9
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: buffer
----------------------------------------
Scope: 0
Type: int
Dimension: 2
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Variables
Name: y
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Variables
Name: z
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Types
Name: ArrayInt
----------------------------------------
Scope: 0
Type: int
Dimension: 2
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
```

```
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
Table: Functions
Name: fillArray
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [firstInt,secondInt]
Parameter types: [int,int]
Parameter dimensions: [0,0]
Return type: -
----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
----------------------------------------
```

```
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------
[ OK ] successful parse...
----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, buffer, 2, 0
    assign, y, 7,
    assign, z, 20,
fillArray:
    array_load, $t0, buffer, 0
    assgin, $t0, firstInt,
    array_load, $t1, buffer, 1
    assgin, $t1, secondInt,
    return, , ,
main:
    call, fillArray, y, z
    array_load, $t2, buffer, 1
    add, 1, 0, $t3
    mult, 2, 9, $t4
    add, $t4, $t3, $t5
    sub, 19, $t5, $t6
    array_load, $t7, buffer, $t6
    array_load, $t8, buffer, 1
    mult, $t8, $t7, $t9
    assgin, $t2, $t9,
    array_load, $t10, buffer, 0
    call, printi, $t10
    array_load, $t11, buffer, 1
    call, printi, $t11
    return, , ,
----------------------------------------
```

Test24.tiger

```
let
      type ArrayInt = array[100] of int;
      var myArray : ArrayInt;
      var x : int := 0;
      var y : int := 7;
      var z : int := 20;
      var loopCounter : int;
      var loopCounter2 : int;
      var myFloat : float := 3.5;
      function doubleMe (input : int) : int
      begin
            return input + input;
      end;
      function quadrupleMe (input : int) : int
      begin
            input := doubleMe(input);
            input := doubleMe(input);
            return input;
      end;
in
      for loopCounter := 1 + 3 to 6 + 7
      do
            if (loopCounter & 1) then
                  x := x + y;
                  myFloat := x / 1.5;
            else
                  x := x + z;
                  myFloat := x / 1.5;
                  x := x - loopCounter;
                  x := x + loopCounter;
                  x := x + loopCounter;
                  for loopCounter2 := 1 to 4
                  do
                        x := x + loopCounter;
                  enddo;
            endif;
      enddo;
      x := quadrupleMe(x);
      myFloat := myFloat * 2;
      printi(x);
end
```

Output

```
[ RUN ] parsing code...

let type id = array [ intlit ] of int ; var <13, "type"><45,
"ArrayInt"><36, "="><0, "array"><27, "["><46, "100"><28,
"]"><10, "of"><20, "int"><24, ";">
id : id ; var <14, "var"><45, "myArray"><23, ":"><45,
"ArrayInt"><24, ";">id : int := intlit ; var <14, "var"><45,
"x"><23, ":"><20, "int"><44, ":="><46, "0"><24, ";">
id : int := intlit ; var <14, "var"><45, "y"><23, ":"><20,
"int"><44, ":="><46, "7"><24, ";">
id : int := intlit ; var <14, "var"><45, "z"><23, ":"><20,
"int"><44, ":="><46, "20"><24, ";">
id : int ; var <14, "var"><45, "loopCounter"><23, ":"><20,
"int"><24, ";">id : int ; var <14, "var"><45,
"loopCounter2"><23, ":"><20, "int"><24, ";">id : float :=
floatlit ; function <14, "var"><45, "myFloat"><23, ":"><21,
"float"><44, ":="><47, "3.5"><24, ";">id ( id : int ) : int
begin return id + id ; end ; function id ( id : int ) : int
begin id := id ( id ) ; <45, "input"><44, ":="><45,
"doubleMe"><25, "("><45, "input"><26, ")">id := id ( id ) ;
<45, "input"><44, ":="><45, "doubleMe"><25, "("><45,
"input"><26, ")">
return id ; end ; in for id := intlit + intlit to <46,
"1"><32, "+"><46, "3">
intlit + intlit do <46, "6"><32, "+"><46, "7">
if ( id & intlit ) then id := id + id ; <45, "x"><44,
":="><45, "x"><32, "+"><45, "y">
id := id / floatlit ; <45, "myFloat"><44, ":="><45, "x"><35,
"/"><47, "1.5">else id := id + id ; <45, "x"><44, ":="><45,
"x"><32, "+"><45, "z">id := id / floatlit ; <45,
"myFloat"><44, ":="><45, "x"><35, "/"><47, "1.5">id := id -
id ; <45, "x"><44, ":="><45, "x"><33, "-"><45,
"loopCounter">id := id + id ; <45, "x"><44, ":="><45,
"x"><32, "+"><45, "loopCounter">id := id + id ; <45, "x"><44,
":="><45, "x"><32, "+"><45, "loopCounter">for id := intlit to
<46, "1">
intlit do <46, "4">id := id + id ; <45, "x"><44, ":="><45,
"x"><32, "+"><45, "loopCounter"> enddo ; endif ; enddo ; id
:= id ( id ) ; <45, "x"><44, ":="><45, "quadrupleMe"><25,
"("><45, "x"><26, ")">
id := id * intlit ; <45, "myFloat"><44, ":="><45,
"myFloat"><34, "*"><46, "2">
id ( id ) ; <45, "printi"><25, "("><45, "x"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t1
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t10
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t11
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Variables
Name: $t12
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t13
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Variables
Name: $t2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t3
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Variables
Name: $t4
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t5
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t6
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t7
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: $t8
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: $t9
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: loopCounter
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Variables
Name: loopCounter2
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Variables
Name: myArray
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Variables
Name: myFloat
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: x
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: y
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: z
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: ArrayInt
----------------------------------------
Scope: 0
Type: int
Dimension: 100
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -

----------------------------------------
Table: Functions
Name: doubleMe
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [input]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
```

```
----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -


----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
```

```
----------------------------------------
Table: Functions
Name: quadrupleMe
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [input]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int
----------------------------------------



[ OK ] successful parse...


----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, myArray, 100, 0
    assign, x, 0,
    assign, y, 7,
    assign, z, 20,
    assign, loopCounter, 0,
    assign, loopCounter2, 0,
    assign, myFloat, 3.5,
doubleMe:
    add, input, input, $t0
    return, $t0, ,
quadrupleMe:
    callr, input, doubleMe, input
    callr, input, doubleMe, input
    return, input, ,
main:
    add, 3, 1, $t1
    add, 7, 6, $t2
    assign, loopCounter, $t1,
loop_label1:
    brgt, loopCounter, $t2, loop_label0
    and, 1, loopCounter, $t3
    assign, $t4, 0,
    brneq, $t3, 0, if_label1
    assign, $t4, 1,
if_label1:
    breq, $t4, 0, if_label0
    add, y, x, $t5
    assgin, x, $t5,
    div, 1.5, x, $t6
    assgin, myFloat, $t6,
    goto, if_label2, ,
if_label0:
    add, z, x, $t7
    assgin, x, $t7,
    div, 1.5, x, $t8
    assgin, myFloat, $t8,
    sub, loopCounter, x, $t9
    assgin, x, $t9,
    add, loopCounter, x, $t10
    assgin, x, $t10,
    add, loopCounter, x, $t11
    assgin, x, $t11,
    assign, loopCounter, $t1,
loop_label3:
    brgt, loopCounter, $t2, loop_label2
    add, loopCounter, x, $t12
    assgin, x, $t12,
    add, loopCounter, 1, loopCounter
    goto, loop_label3, ,
loop_label2:
if_label2:
    add, loopCounter, 1, loopCounter
    goto, loop_label1, ,
loop_label0:
    callr, x, quadrupleMe, x
    mult, 2, myFloat, $t13
    assgin, myFloat, $t13,
    call, printi, x
    return, , ,
----------------------------------------
```

Test25.tiger

```
let
      var A, B, C, D, E : float;
in
      A := 1.5;
      B := 2.5;
      C := 3.5;
      D := 4.5;
      E := 5.5;
      printi(A);  /* Error only support print integer */
      printi(B);
      printi(C);
      printi(D);
      printi(E);
end
```

Output:

```
[ RUN ] parsing code...

let var id , id , id , id , id : float ; in <14, "var"><45,
"A"><22, ","><45, "B"><22, ","><45, "C"><22, ","><45,
"D"><22, ","><45, "E"><23, ":"><21, "float"><24, ";">
id := floatlit ; <45, "A"><44, ":="><47, "1.5">
id := floatlit ; <45, "B"><44, ":="><47, "2.5">
id := floatlit ; <45, "C"><44, ":="><47, "3.5">
id := floatlit ; <45, "D"><44, ":="><47, "4.5">
id := floatlit ; <45, "E"><44, ":="><47, "5.5">
id ( id ) ; <45, "printi"><25, "("><45, "A"><26, ")">

 Error: A: float mismatched to function printi parameter:
int
```

Test26.tiger

```
/**
 * This program calculates and prints the factorial of the 'number'
variable.
 */
let
      var number : int := 8;
      var loopCounter : int;
      var result : int := 1;
in
      for loopCounter := 1 to number
      do
            result := result * loopCounter;
      enddo;
      printi(result);
end
```

Output:

```
[ RUN ] parsing code...

let var id : int := intlit ; var <14, "var"><45,
"number"><23, ":"><20, "int"><44, ":="><46, "8"><24, ";">
id : int ; var <14, "var"><45, "loopCounter"><23, ":"><20,
"int"><24, ";">
id : int := intlit ; in <14, "var"><45, "result"><23,
":"><20, "int"><44, ":="><46, "1"><24, ";">
for id := intlit to <46, "1">
id do <45, "number">
id := id * id ; <45, "result"><44, ":="><45, "result"><34,
"*"><45, "loopCounter">
enddo ; id ( id ) ; <45, "printi"><25, "("><45,
"result"><26, ")">
end
```

```
----------------------------------------
Table: Variables
Name: $t0
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: loopCounter
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: number
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Variables
Name: result
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -
```

```
----------------------------------------
Table: Types
Name: float
----------------------------------------
Scope: 0
Type: float
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Types
Name: int
----------------------------------------
Scope: 0
Type: int
Dimension: 0
Parameters: -
Parameter types: -
Parameter dimensions: -
Return type: -


----------------------------------------
Table: Functions
Name: exit
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -


----------------------------------------
Table: Functions
Name: flush
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: []
Parameter types: []
Parameter dimensions: []
Return type: -
```

```
----------------------------------------
Table: Functions
Name: not
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: int


----------------------------------------
Table: Functions
Name: printi
----------------------------------------
Scope: 0
Type: -
Dimension: -
Parameters: [i]
Parameter types: [int]
Parameter dimensions: [0]
Return type: -
----------------------------------------

[ OK ] successful parse...

----------------------------------------
Generate IR CODE ...
----------------------------------------
    assign, number, 8,
    assign, loopCounter, 0,
    assign, result, 1,
main:
    assign, loopCounter, 1,
loop_label1:
    brgt, loopCounter, number, loop_label0
    mult, loopCounter, result, $t0
    assgin, result, $t0,
    add, loopCounter, 1, loopCounter
    goto, loop_label1, ,
loop_label0:
    call, printi, result
    return, , ,
----------------------------------------
```