MiniTS: A Compiler for a TypeScript-Inspired Class-Based Language: User Manual

Team Members:
Russell Liu
russl8@my.yorku.ca
219616150

Jeffrey Jiang
jeffj4@my.yorku.ca
218756668

Table of Contents

1 Input Languages

1.1 Structure of a Program

```
class Superclass {
      c : list[char] = "hello";
      i : int = 0;
}
class Overview extends Superclass {
      function f (i : int, j : int) : int {
            return i + j;
      }

      for (j : int = 1; j <= 5; j = j + 1) {
            i = f(i, j); // i = 1+2+3+4+5 = 15
      }

     j : int = 0;
     if(i == 15) {
          j = i;
     }
}
```

Figure 1: Structure of input program

Input Language:

A typical input program accepted by the compiler is class-based. Meaning that it consists of one or more class definitions, each of which can inherit from a superclass. Within these classes, users can:

1. Declare and assign variables
2. Define functions
3. Use control flow statements such as loops and conditionals.

Key Constructs

Classes and Inheritance

Programs are organized around classes. All statements and functions **must** be declared within a class. A class can inherit attributes and functions from another by extending it.

Variable Declarations and Assignments

Variables are strongly typed and can optionally be initialized during assignment. The supported types are ***int***, ***char***, ***bool***, and ***list*** parameterized by int, char, or bool.

Scoping

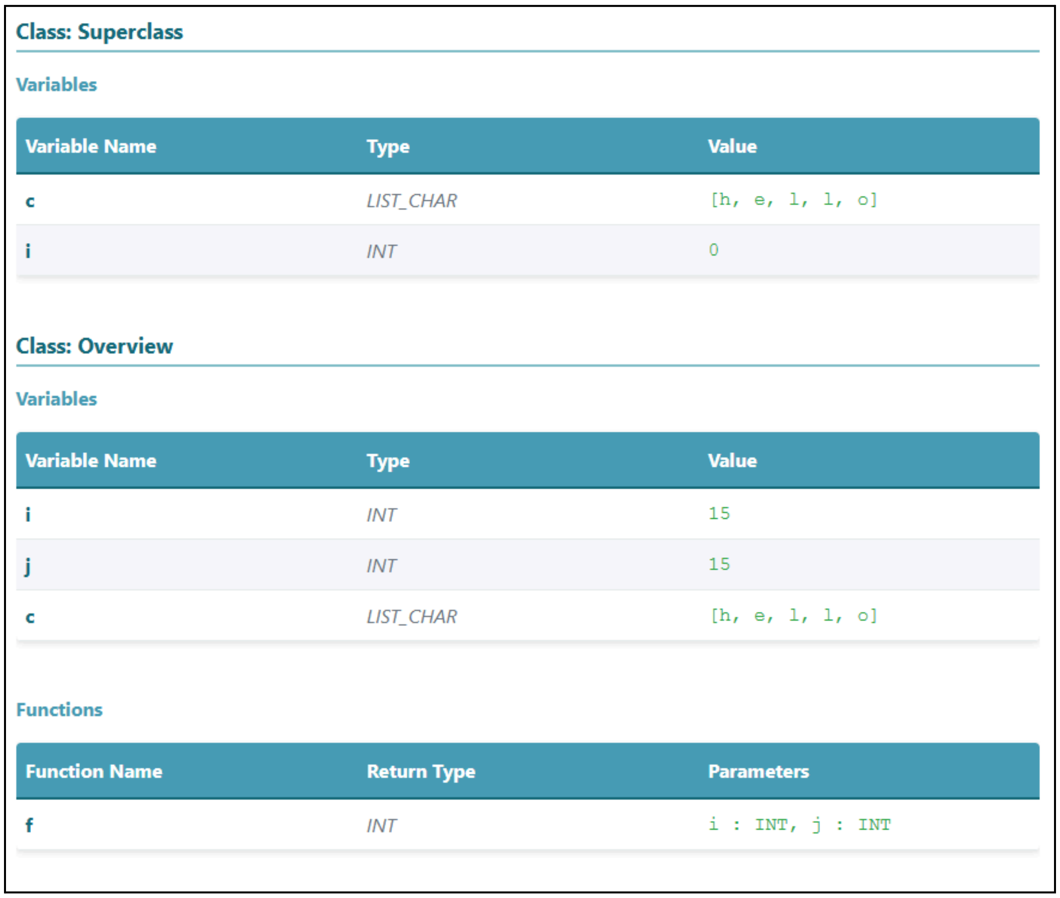
Variables follow lexical scoping rules: variables declared inside a function or block (such as a for loop or if statement) are local to that block. Class-level variables are accessible within all methods of the class and by its subclasses unless shadowed.

Functions

Functions support zero or more parameters and **must** specify a return type. All parameters and return values must be explicitly typed.

Control Flow

The language currently supports for loops, while loops, and if statements.



Class: Superclass

Variables

Variable Name	Type	Value
c	*LIST_CHAR*	`[h, e, l, l, o]`
i	*INT*	`0`

Class: Overview

Variables

Variable Name	Type	Value
i	*INT*	`15`
j	*INT*	`15`
c	*LIST_CHAR*	`[h, e, l, l, o]`

Functions

Function Name	Return Type	Parameters
f	*INT*	`i : INT, j : INT`

Figure 2: Evaluated variables of input program (Refer to Figure 1)

1.2 List of Advanced Programming Features

Feature 1: Inheritance

Example file(s): inventory.txt

Inheritance enables automatic access to ***all*** variables and functions defined in the superclass. To inherit attributes from another class, append the keyword *extends <ClassName>* to the class declaration.

```
class A {
    i : int = 0;
}
class B extends A { // Inherits attributes from class A
    j : int = i + 1; // j = 1
}
```

Feature 2: For-Loops

Example file(s): task_manager.txt

The for loop follows the structure:

```
for (<declaration>; <boolean_expression>; <assignment>) {
      // loop body
}
```

Example:

```
j : int = 0;
for (i: int = 0; i < 5; i = i + 1) {
    k : int = j + i;
}
```

Feature 3: Scoping

Example file(s): gaming.txt

Variables are scoped to the block in which they are declared. Identifiers can be reused in different scopes without conflict. Functions and loops both introduce new scopes.

a) Scoping in Loops

```
class Main {
    x: int = 100;

    function loopTest(): int {
        for (x: int = 0; x < 10; x = x + 1) {
            // 'x' here is local to the loop
        }
    }
    x = 10; // refers to the class-level 'x'
}
```

The loop-level x shadows the class-level x only within the loop body.

b) Scoping in Functions

```
class Main {
    x: int = 42;
    function print(x: int): int {
        return x; // Refers to the parameter
    }
    x = print(x); // x == 42
}
```

The parameter x shadows the outer x for the entire function body.

Feature 4: Function Declaration & Invocation

Example file(s): fleet.txt, student.txt

Functions are declared using the following syntax:

```
function functionName(param1: type, param2: type): returnType {
    // function body
}
```

Example:

```
function f(i: int): bool {
    return i == 0;
}
x: bool = f(1);
```

Feature 5: Lists

Example file(s): weather.txt

Lists can be declared and initialized using the following syntax:

```
var: list[type] = [...];
var = [...];
```

Example:

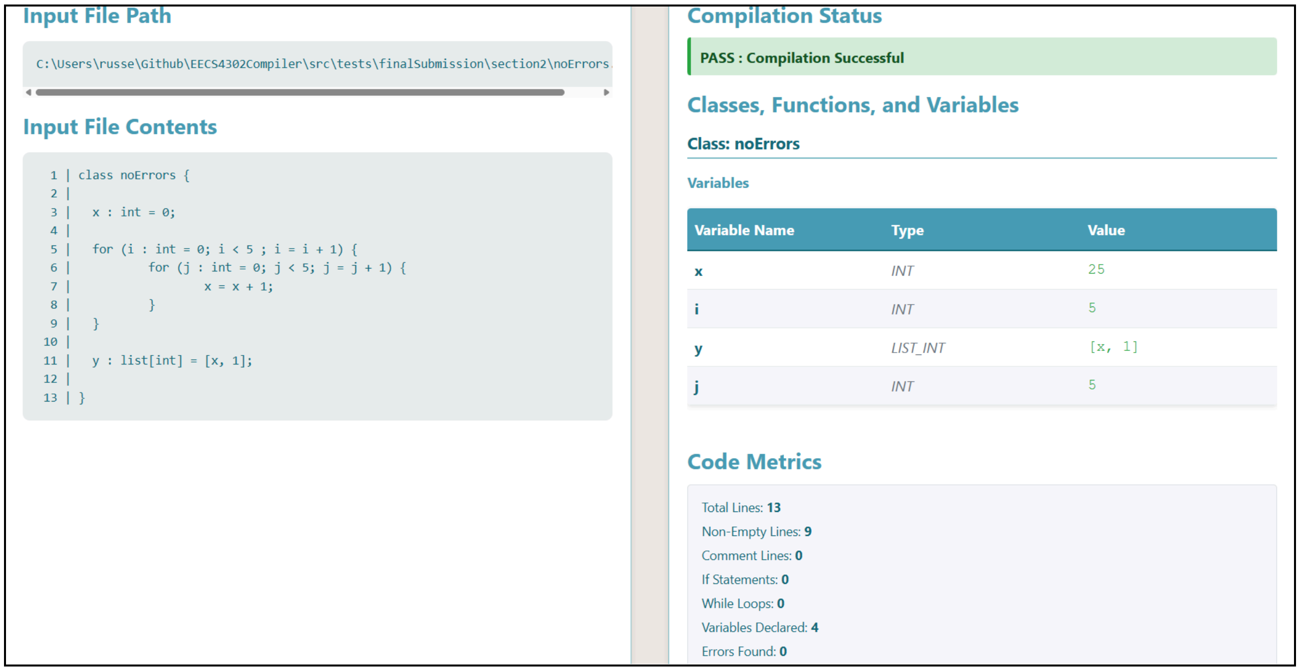
```
a: list[int] = [1, 2, 4];
b: list[char] = "this is a string";
c: list[char] = ['s', 't', 'r', 'i', 'n', 'g'];
```

Supported list types include list[int], list[char], and list[bool]. Also, it is worth pointing out that a character list can be declared in two different ways.

2 Output Structure

The HTML display differs depending on whether or not the program has semantic errors.

Program has no semantic errors



Input File Path

```
C:\Users\russe\Github\EECS4302Compiler\src\tests\finalSubmission\section2\noErrors
```

Input File Contents

```
class noErrors {

  x : int = 0;

  for (i : int = 0; i < 5 ; i = i + 1) {
          for (j : int = 0; j < 5; j = j + 1) {
                  x = x + 1;
          }
  }

  y : list[int] = [x, 1];

}
```

Compilation Status

PASS : Compilation Successful

Classes, Functions, and Variables

Class: noErrors

Variables

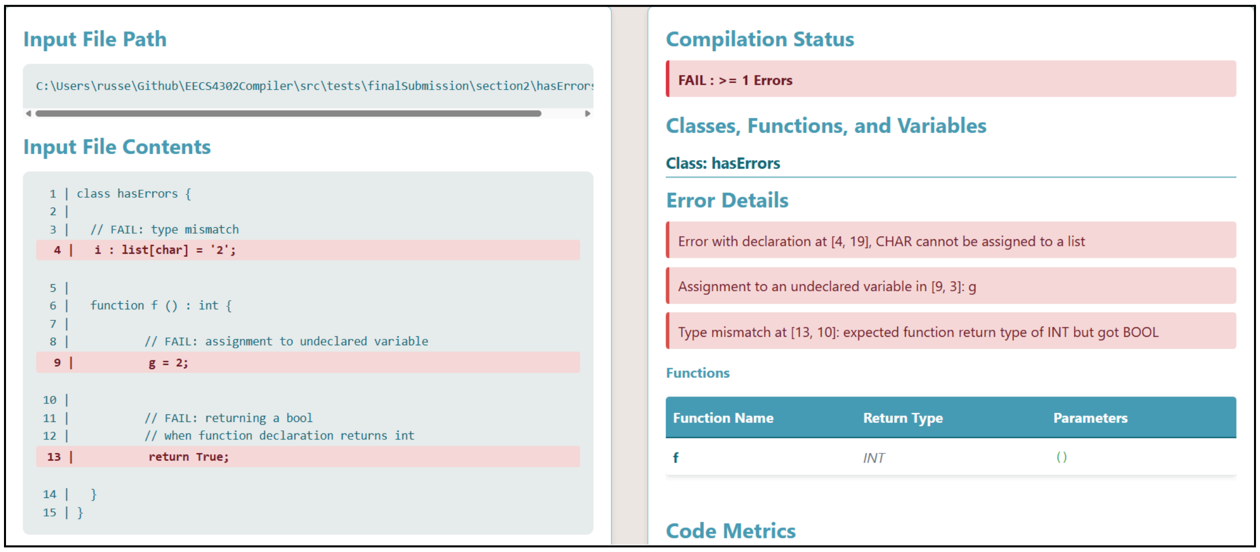
Variable Name	Type	Value
x	INT	25
i	INT	5
y	LIST_INT	[x, 1]
j	INT	5

Code Metrics

Total Lines: **13**
Non-Empty Lines: **9**
Comment Lines: **0**
If Statements: **0**
While Loops: **0**
Variables Declared: **4**
Errors Found: **0**

If the program is free of semantic errors, the compilation status will display **"PASS"**. The output will also list the final evaluated values of all variables and functions.

Program has semantic errors



Input File Path

```
C:\Users\russe\Github\EECS4302Compiler\src\tests\finalSubmission\section2\hasError
```

Input File Contents

```
 1 | class hasErrors {
 2 |
 3 |   // FAIL: type mismatch
 4 |   i : list[char] = '2';

 5 |
 6 |   function f () : int {
 7 |
 8 |           // FAIL: assignment to undeclared variable
 9 |           g = 2;

10 |
11 |           // FAIL: returning a bool
12 |           // when function declaration returns int
13 |           return True;

14 |   }
15 | }
```

Compilation Status

FAIL : >= 1 Errors

Classes, Functions, and Variables

Class: hasErrors

Error Details

Error with declaration at [4, 19], CHAR cannot be assigned to a list

Assignment to an undeclared variable in [9, 3]: g

Type mismatch at [13, 10]: expected function return type of INT but got BOOL

Functions

Function Name	Return Type	Parameters
f	INT	()

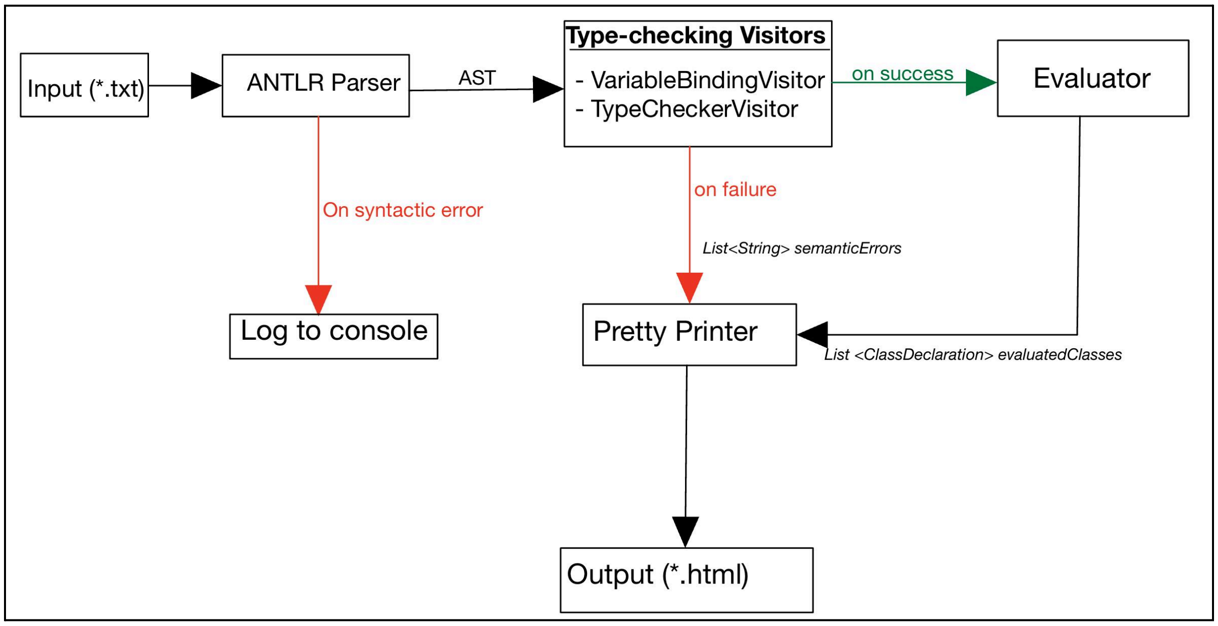
Code Metrics

The compiler checks for semantic errors such as:

- Type mismatches
- References to undeclared variables
- Variable redeclarations
- Inconsistent list types
- Other issues (as covered in Section 4)

Any detected errors will be displayed on the right-hand side of the UI. Corresponding lines in the input will be highlighted on the left for easy reference.

3 Compiler Workflow and Architecture



The compiler processes .txt files containing class-based programs and produces .html outputs. It begins with the ANTLR parser, which constructs an Abstract Syntax Tree (AST). If syntax errors are detected, the compiler logs them to the console and halts further processing.

If parsing succeeds, the AST (with the program node as root) is traversed by *two* semantic analysis visitors in sequence:

VariableBindingVisitor
Records declared variables in memory and ensures no undeclared variables are used.

TypeCheckerVisitor
Validates type correctness for each node, such as preventing boolean values from being assigned to integer variables.

If any semantic errors are found, evaluation is skipped, and the errors are passed to the Pretty Printer to generate an HTML summary.

If no errors are found, the AST is passed to the Evaluator, which assumes semantic validity and sends each class's variable summary to the Pretty Printer for final HTML generation.

4 Justification of Output

The output highlights several type checking criteria applied during compilation. In the code samples below, semantic errors are marked in red, and inline comments explain the reason behind each error.

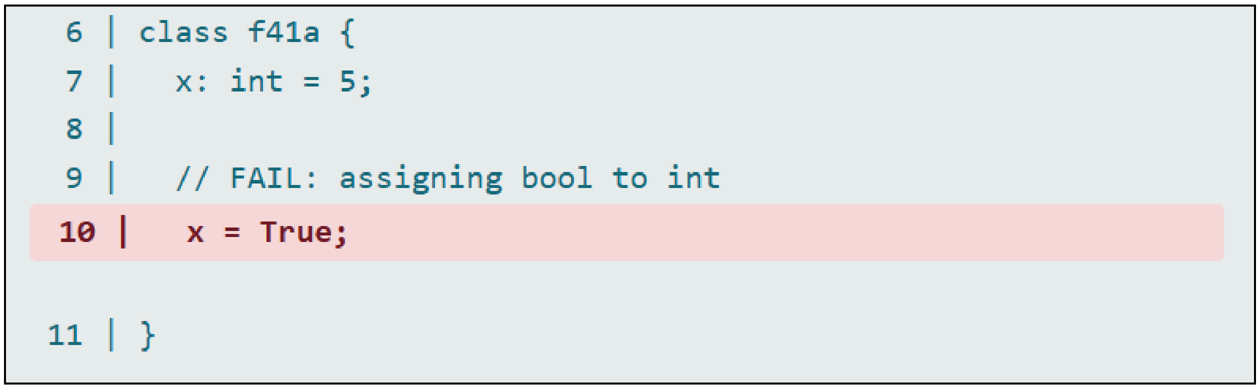
Below each code block, an error log is displayed, providing further detail about the type checking failure, such as:

- mismatched types
- undeclared variables
- incorrect usage of expressions

4.1 Variable declarations and assignments must match their declared type

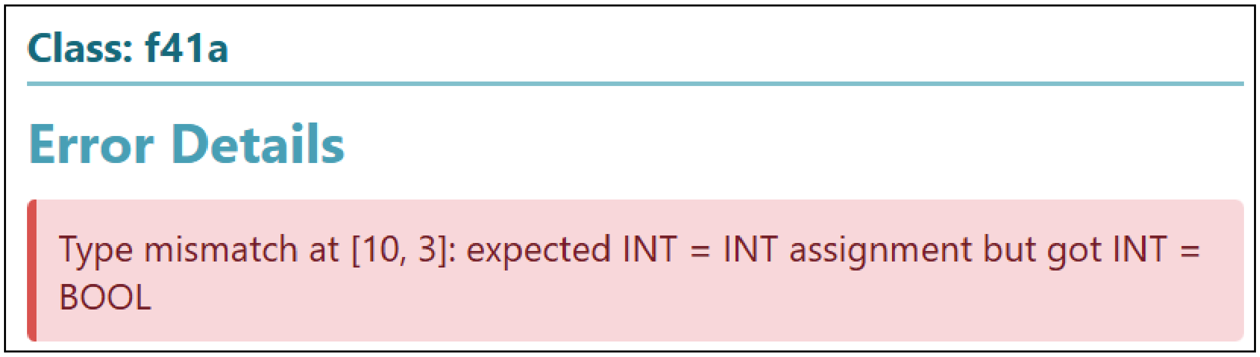
4.1.1 Primitave Variables

Declarations and assignments must match their declared types. For example, boolean expressions such as == and <= can only be assigned to boolean variables, and arithmetic expressions can only be assigned to integer variables.



```
 6 | class f41a {
 7 |   x: int = 5;
 8 |
 9 |   // FAIL: assigning bool to int
10 |   x = True;

11 | }
```



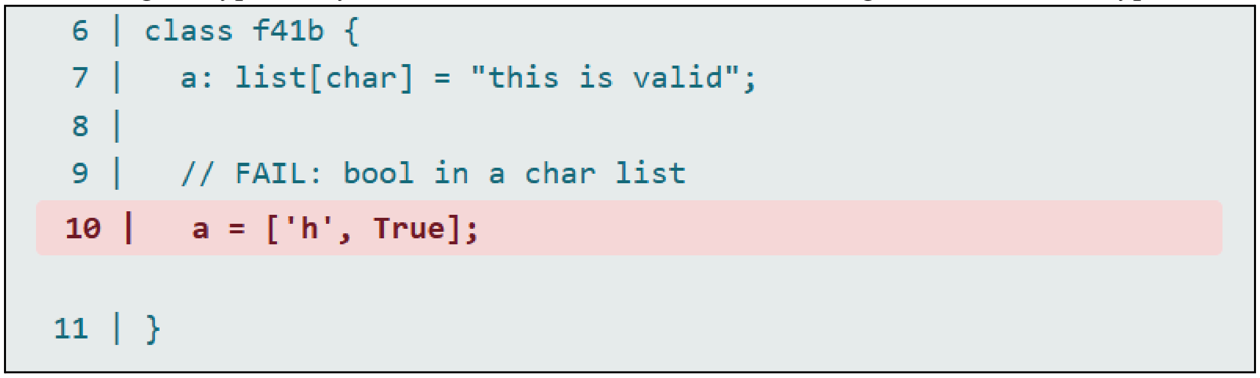
Class: f41a

Error Details

```
Type mismatch at [10, 3]: expected INT = INT assignment but got INT = BOOL
```

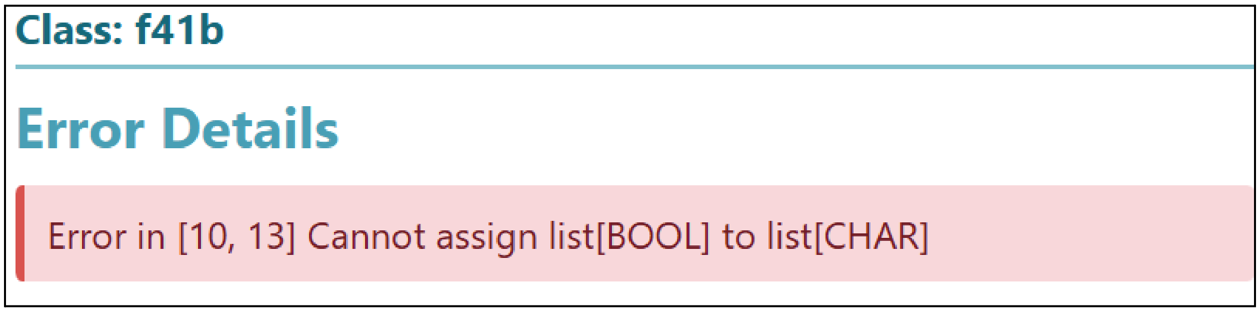
4.1.2 Lists

The same concept applies to list declarations and assignment statements. One difference is that for ensuring list type-safety, each item in the list must be checked against the declared type.



```
 6 | class f41b {
 7 |   a: list[char] = "this is valid";
 8 |
 9 |   // FAIL: bool in a char list
10 |   a = ['h', True];

11 | }
```



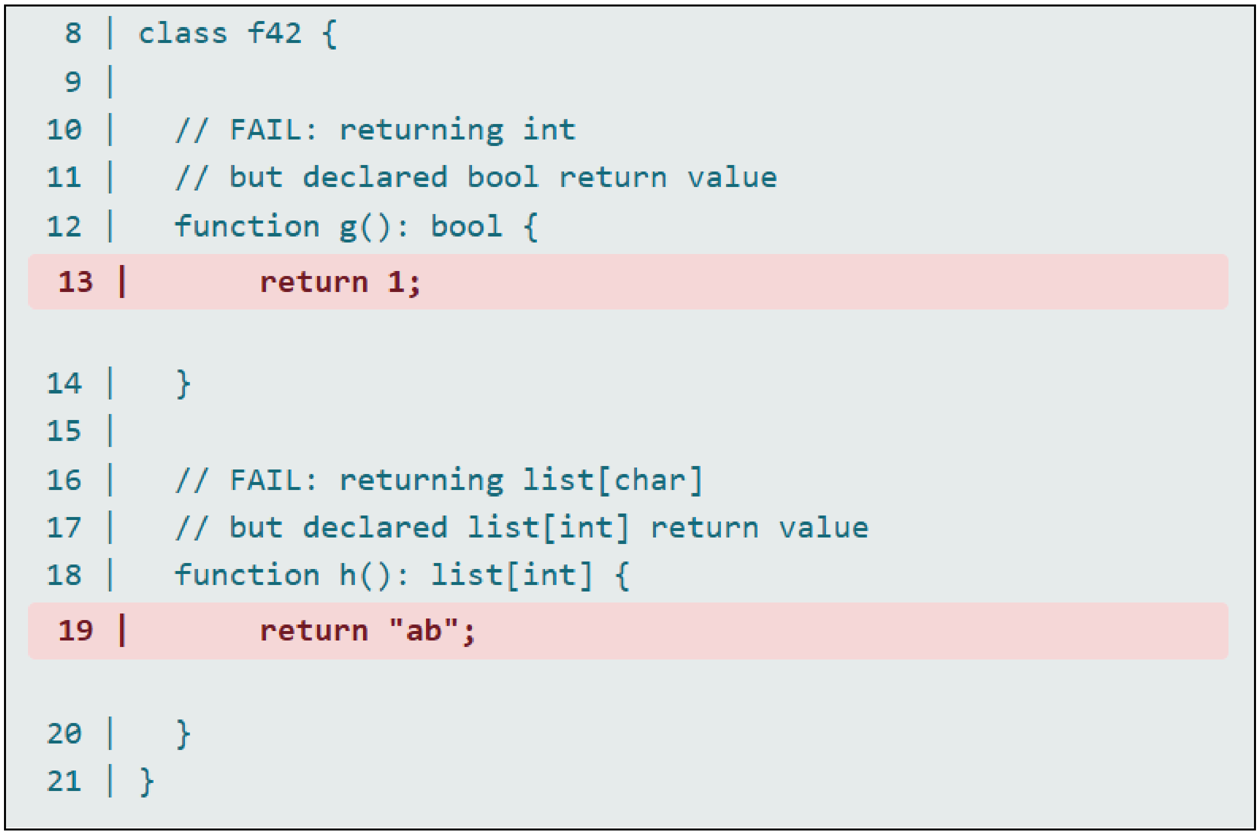
Class: f41b

Error Details

```
Error in [10, 13] Cannot assign list[BOOL] to list[CHAR]
```

4.2 Function return expressions must match their decleared return type

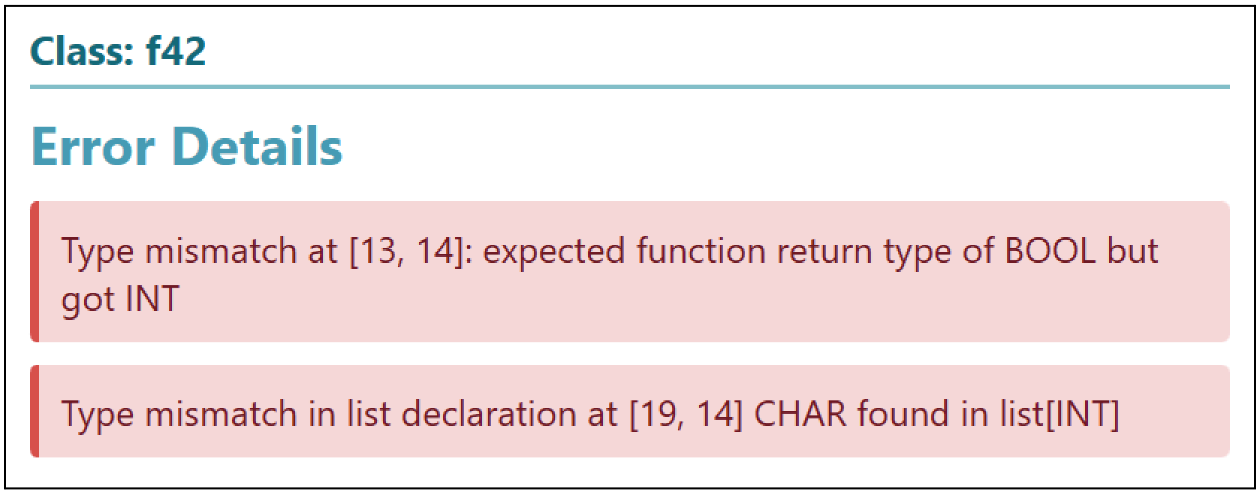
To ensure the type-safety of functions, the expression following their return statement is checked against the type it was declared with.



```
 8 | class f42 {
 9 |
10 |   // FAIL: returning int
11 |   // but declared bool return value
12 |   function g(): bool {
13 |       return 1;

14 |   }
15 |
16 |   // FAIL: returning list[char]
17 |   // but declared list[int] return value
18 |   function h(): list[int] {
19 |       return "ab";

20 |   }
21 | }
```



Class: f42

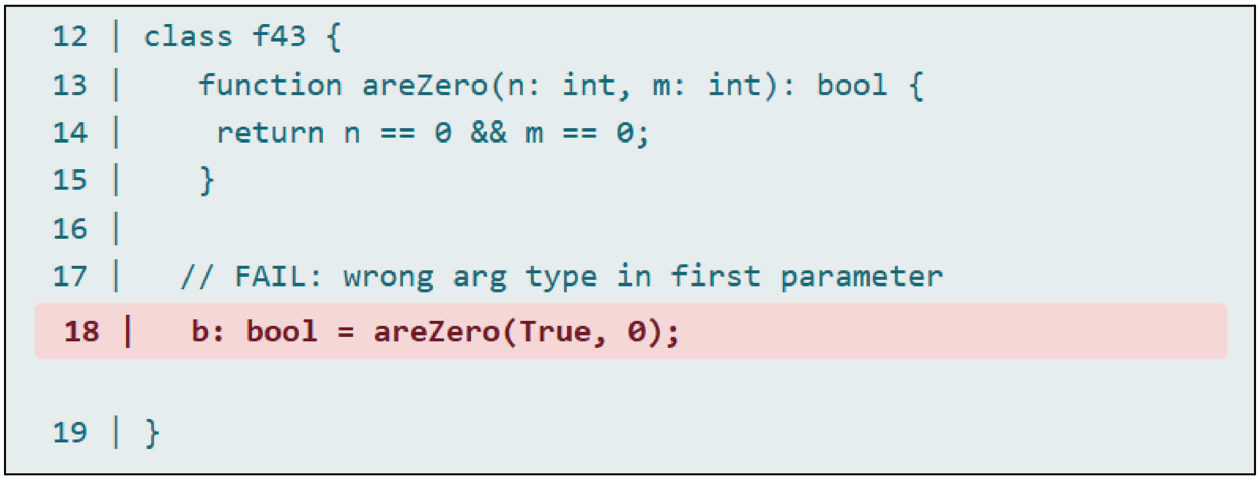
Error Details

Type mismatch at [13, 14]: expected function return type of BOOL but got INT

Type mismatch in list declaration at [19, 14] CHAR found in list[INT]

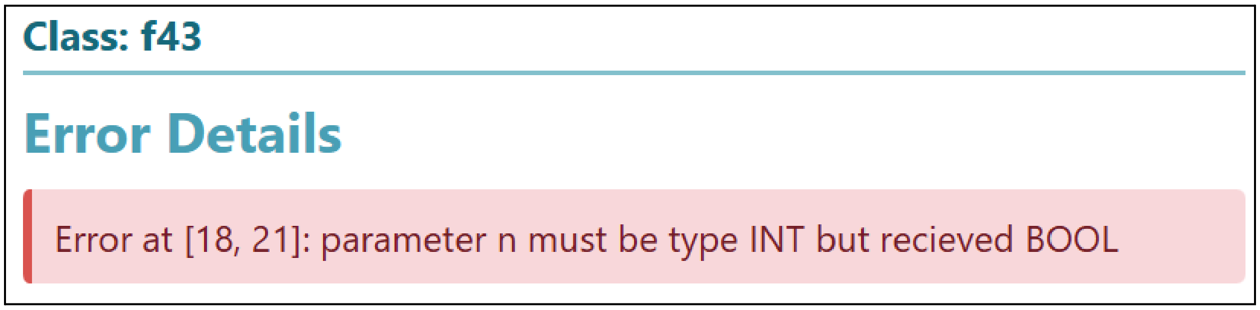
4.3 Function argument types must match parameter types

This check ensures that each parameter used in a function invocation matches the type specified in the declaration.



```
12 | class f43 {
13 |    function areZero(n: int, m: int): bool {
14 |     return n == 0 && m == 0;
15 |    }
16 |
17 |   // FAIL: wrong arg type in first parameter
18 |   b: bool = areZero(True, 0);

19 | }
```



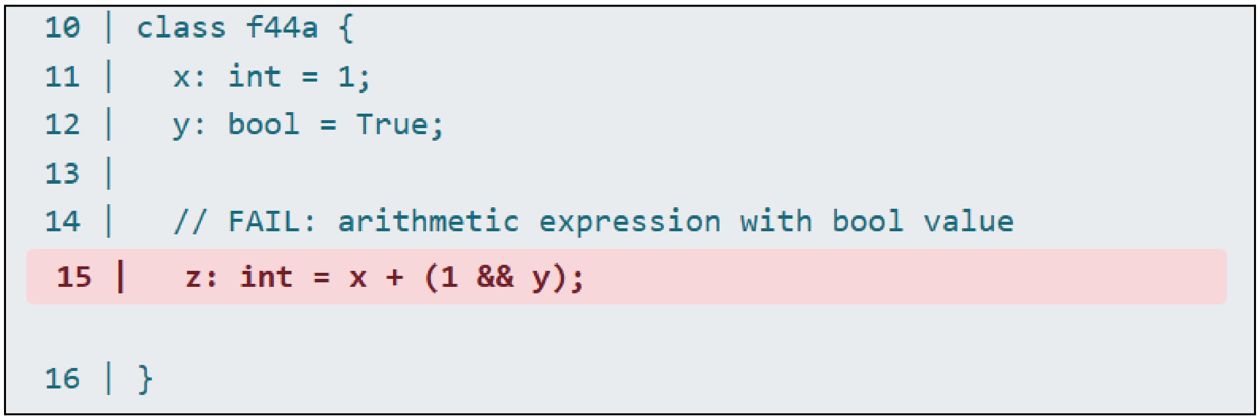
Class: f43

Error Details

```
Error at [18, 21]: parameter n must be type INT but recieved BOOL
```

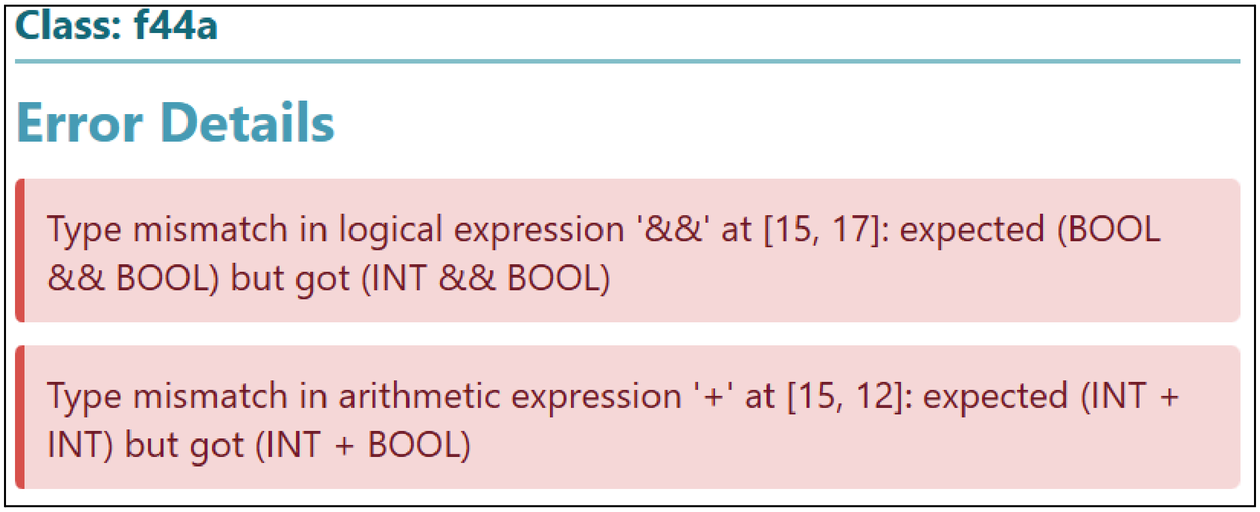
4.4 Binary and unary expressions must use valid operand types

4.4.1 Arithmetic (+, -, /, *, %): Both sides must be integers



```
10 | class f44a {
11 |   x: int = 1;
12 |   y: bool = True;
13 |
14 |   // FAIL: arithmetic expression with bool value
15 |   z: int = x + (1 && y);

16 | }
```



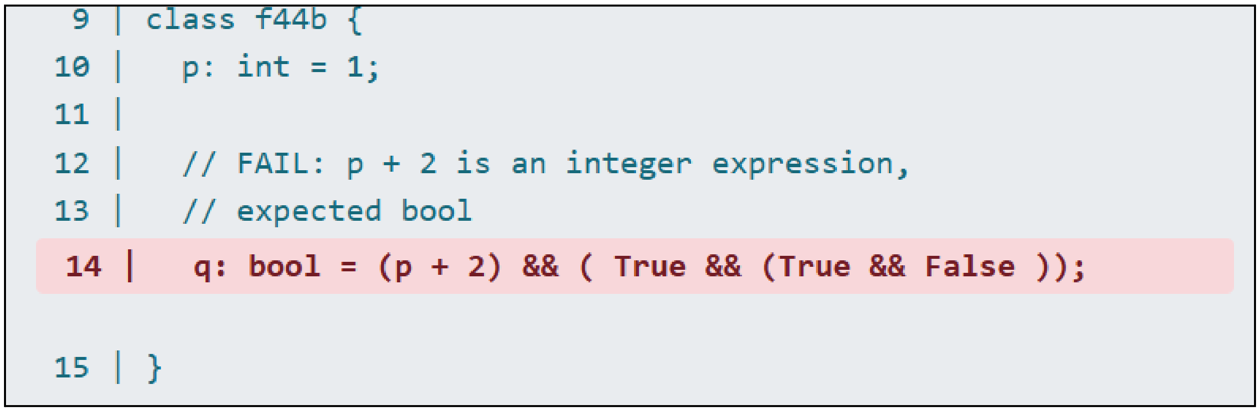
Class: f44a

Error Details

Type mismatch in logical expression '&&' at [15, 17]: expected (BOOL && BOOL) but got (INT && BOOL)

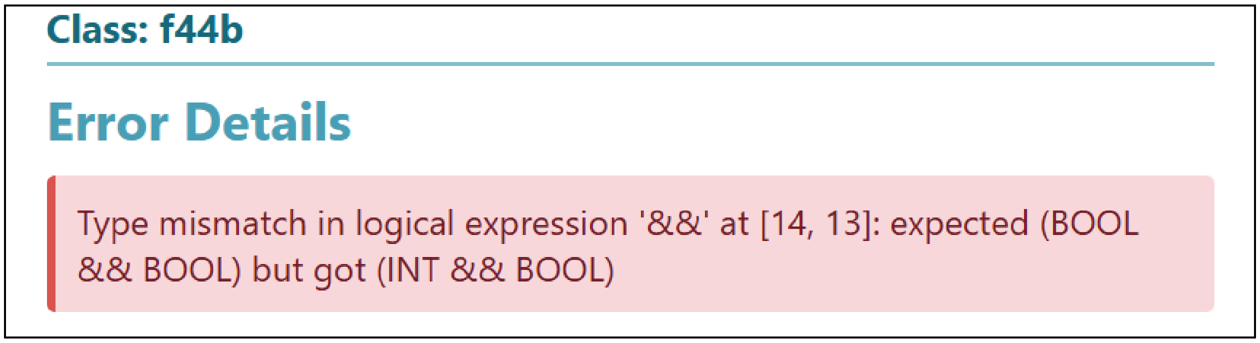
Type mismatch in arithmetic expression '+' at [15, 12]: expected (INT + INT) but got (INT + BOOL)

4.4.2 Logical (&&, ||, !): Operands must be boolean



```
 9 | class f44b {
10 |   p: int = 1;
11 |
12 |   // FAIL: p + 2 is an integer expression,
13 |   // expected bool
14 |   q: bool = (p + 2) && ( True && (True && False ));

15 | }
```

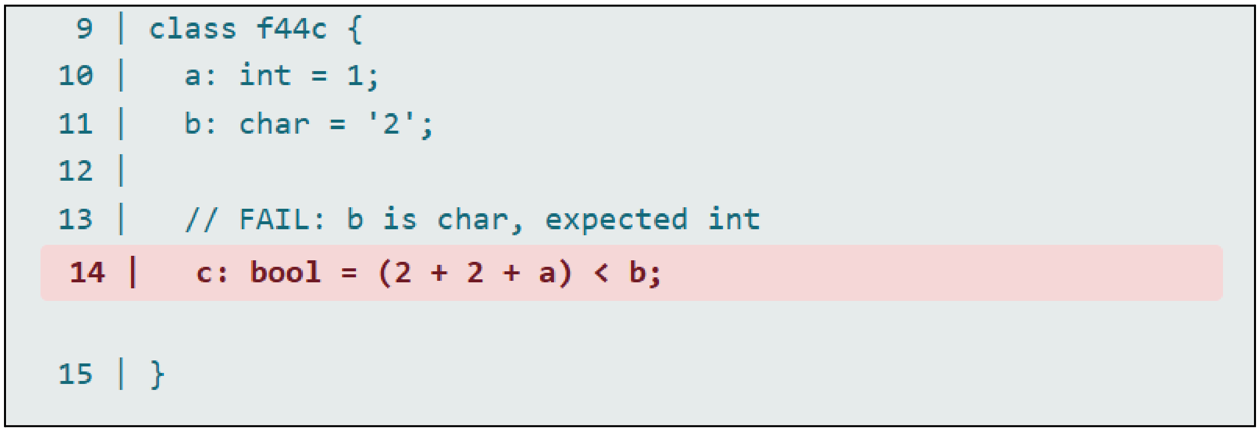


Class: f44b

Error Details

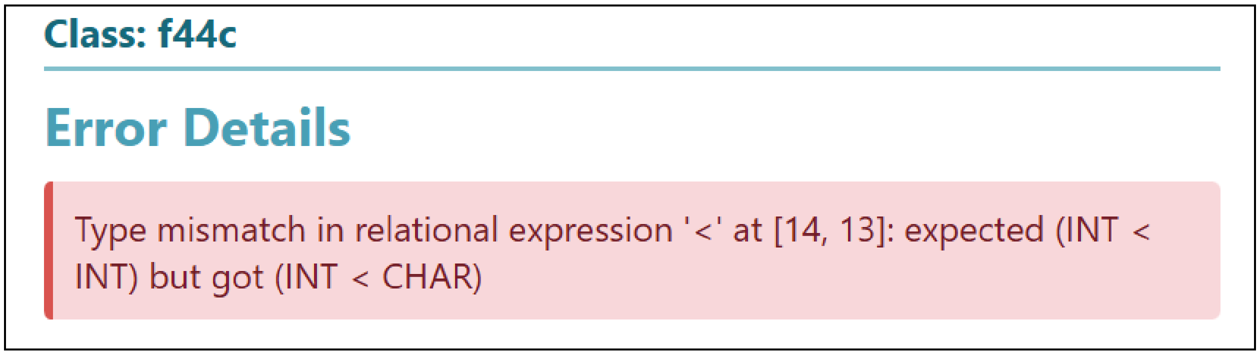
Type mismatch in logical expression '&&' at [14, 13]: expected (BOOL && BOOL) but got (INT && BOOL)

4.4.3 Relational (<=, <, >, >=): Both sides must be integers.



```
 9 | class f44c {
10 |   a: int = 1;
11 |   b: char = '2';
12 |
13 |   // FAIL: b is char, expected int
14 |   c: bool = (2 + 2 + a) < b;

15 | }
```



Class: f44c

Error Details

Type mismatch in relational expression '<' at [14, 13]: expected (INT < INT) but got (INT < CHAR)

4.4.4 Equality (==, !=) – Both sides must be of the same type

```
11 | class f44d {
12 |   x: int = 1;
13 |   y: bool = True;
14 |
15 |   // FAIL: expected that LHS
16 |   // and RHS are the same type
17 |   e: bool = x == (((!y)));

18 | }
```

Class: f44d

Error Details

```
Type mismatch in equality expression '==' at [17, 13]: expected (INT ==
INT) or (BOOL == BOOL) but got (INT == BOOL)
```

4.5 If-statement and while-loop conditions must be boolean

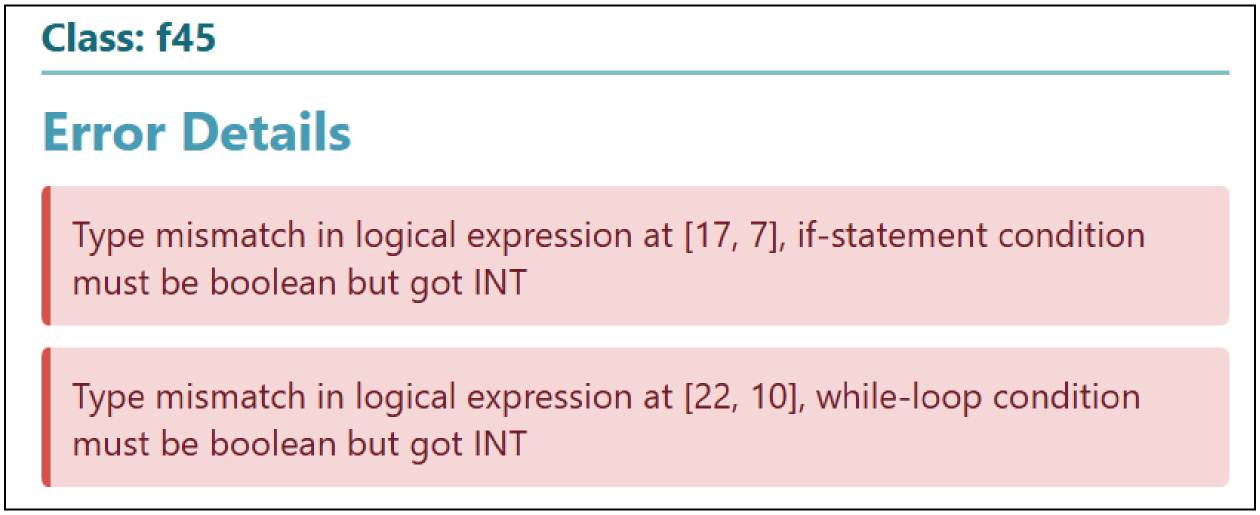
```
class f45 {

  // FAIL: condition not boolean
  if (1) { x: int = 1; }


  x: int = 0;

  // FAIL: condition not boolean
  while (5 + 5) {

    x = x + 1;
  }
}
```



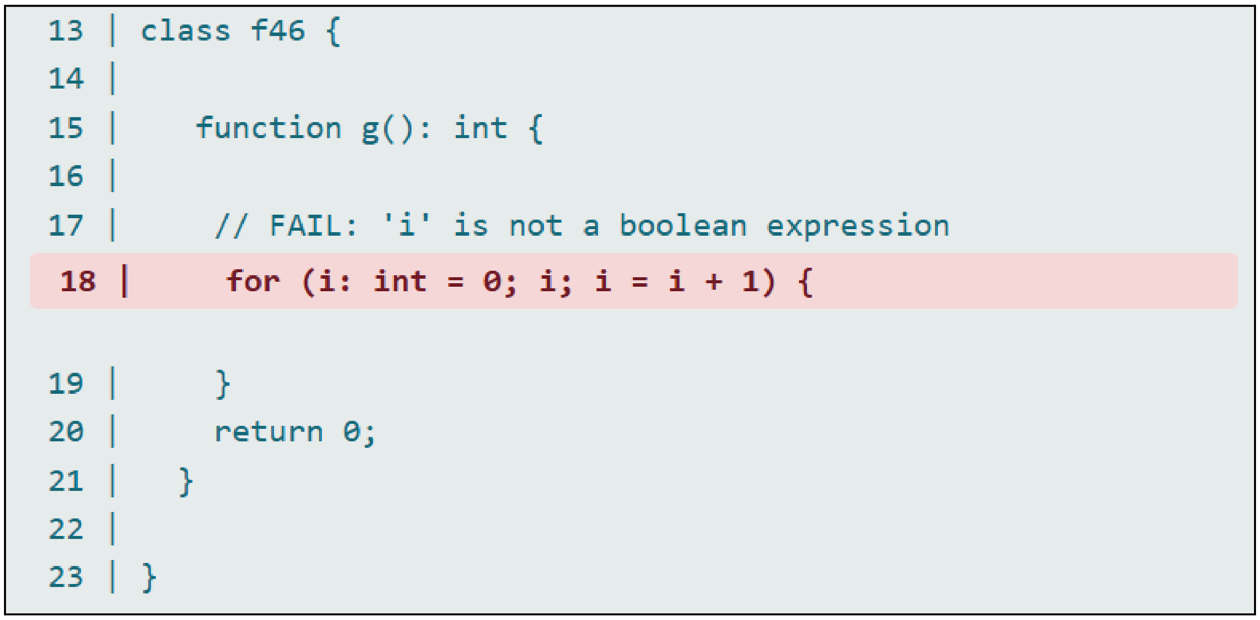
Class: f45

Error Details

Type mismatch in logical expression at [17, 7], if-statement condition must be boolean but got INT

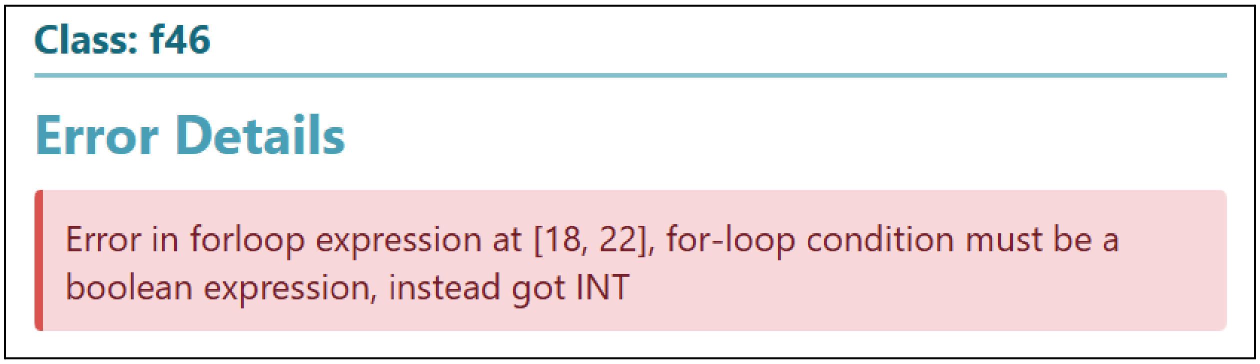
Type mismatch in logical expression at [22, 10], while-loop condition must be boolean but got INT

4.6 For-loop expressions must be of specific types



```
13 | class f46 {
14 |
15 |    function g(): int {
16 |
17 |     // FAIL: 'i' is not a boolean expression
18 |      for (i: int = 0; i; i = i + 1) {

19 |     }
20 |     return 0;
21 |   }
22 |
23 | }
```

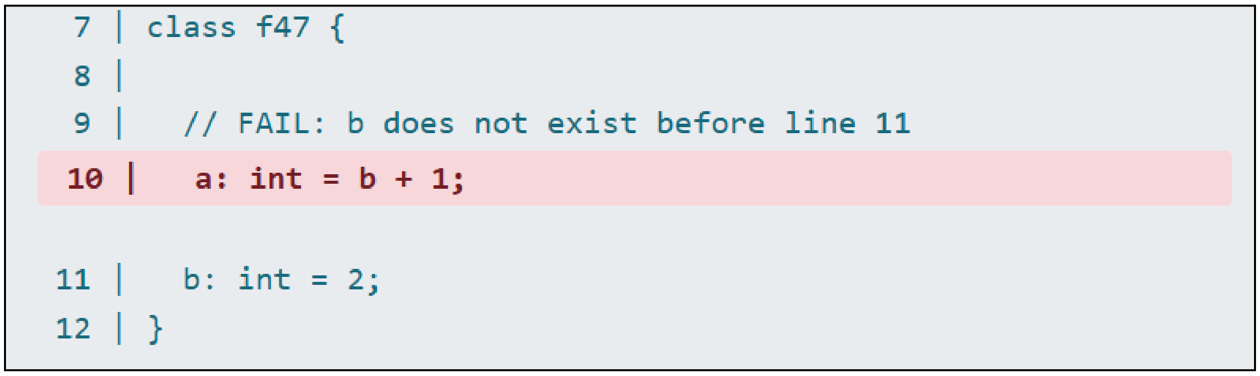


Class: f46

Error Details

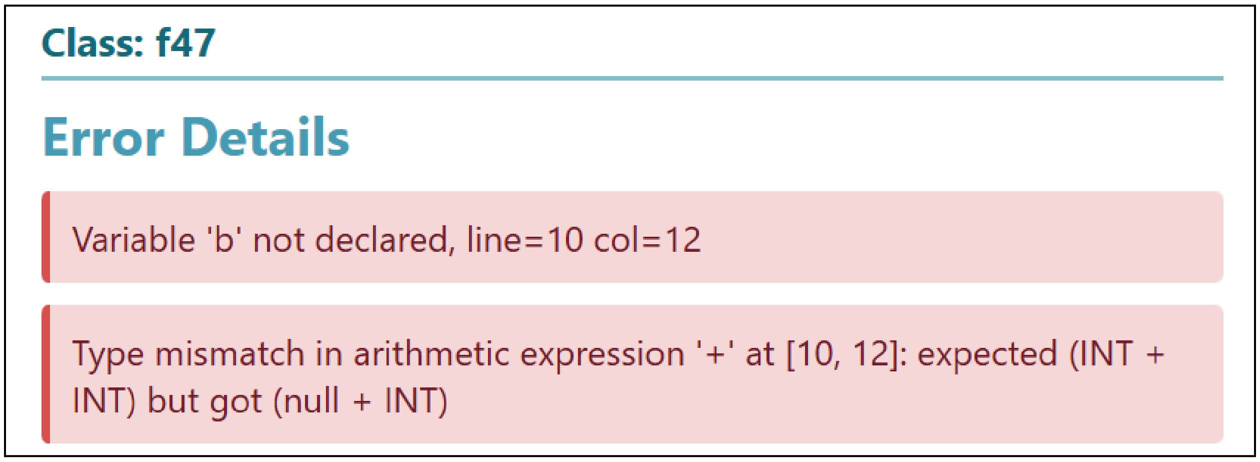
Error in forloop expression at [18, 22], for-loop condition must be a boolean expression, instead got INT

4.7 Variables must be declared before use



```
 7 | class f47 {
 8 |
 9 |   // FAIL: b does not exist before line 11
10 |   a: int = b + 1;

11 |   b: int = 2;
12 | }
```



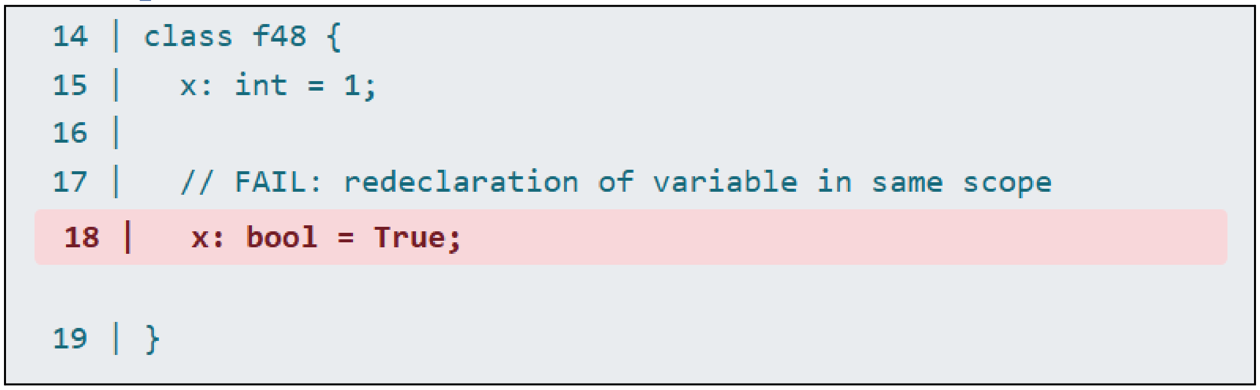
Class: f47

Error Details

Variable 'b' not declared, line=10 col=12

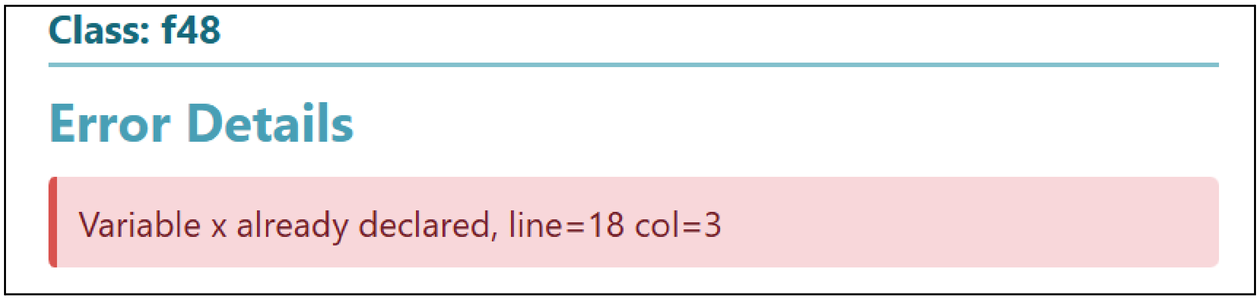
Type mismatch in arithmetic expression '+' at [10, 12]: expected (INT + INT) but got (null + INT)

4.8 Variables cannot be declared more than once in the same scope



```
14 | class f48 {
15 |   x: int = 1;
16 |
17 |   // FAIL: redeclaration of variable in same scope
18 |   x: bool = True;

19 | }
```



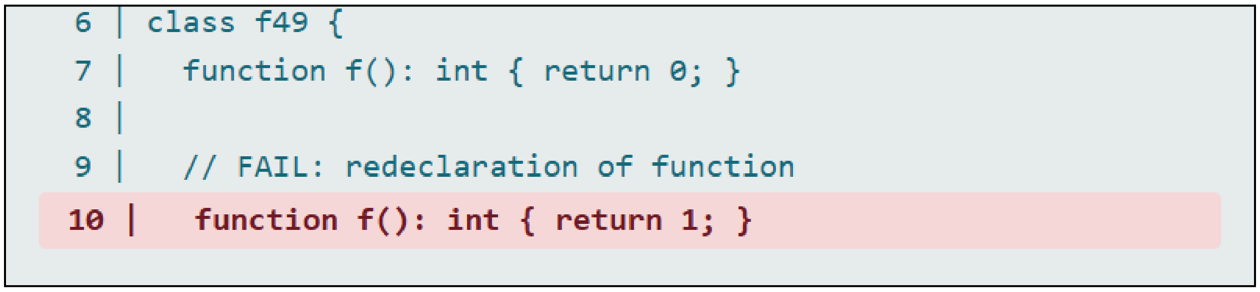
Class: f48

Error Details

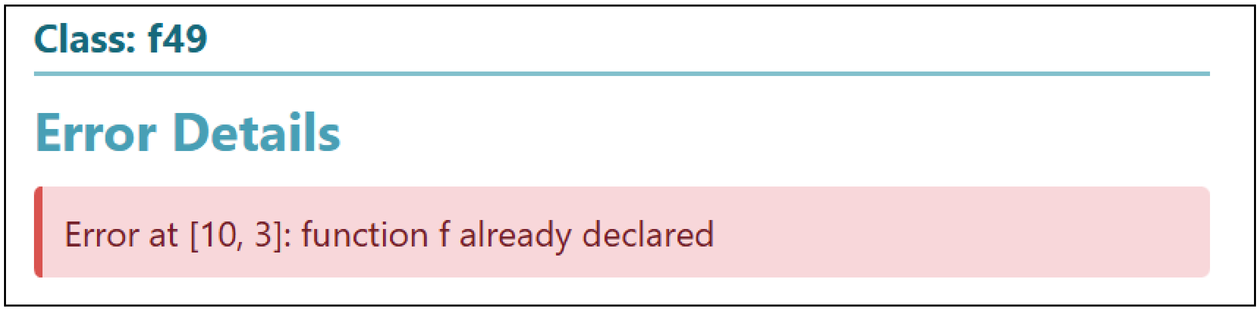
```
Variable x already declared, line=18 col=3
```

4.9 Functions and Classes cannot be redefined

4.9.1 Function redeclaration error



```
 6 | class f49 {
 7 |   function f(): int { return 0; }
 8 |
 9 |   // FAIL: redeclaration of function
10 |   function f(): int { return 1; }
```

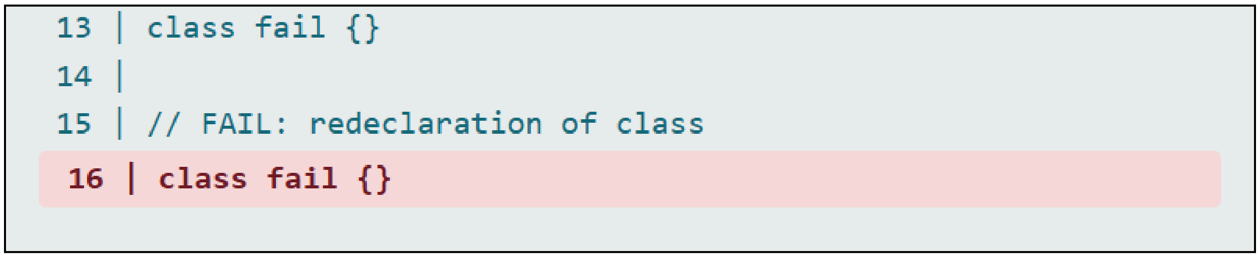


Class: f49

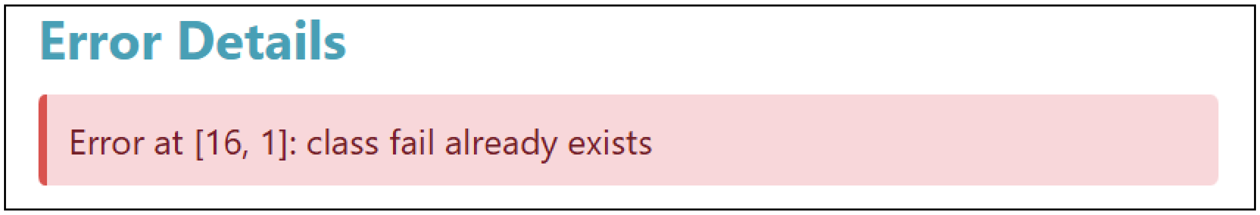
Error Details

```
Error at [10, 3]: function f already declared
```

4.9.2 Class redeclaration error



```
13 | class fail {}
14 |
15 | // FAIL: redeclaration of class
16 | class fail {}
```

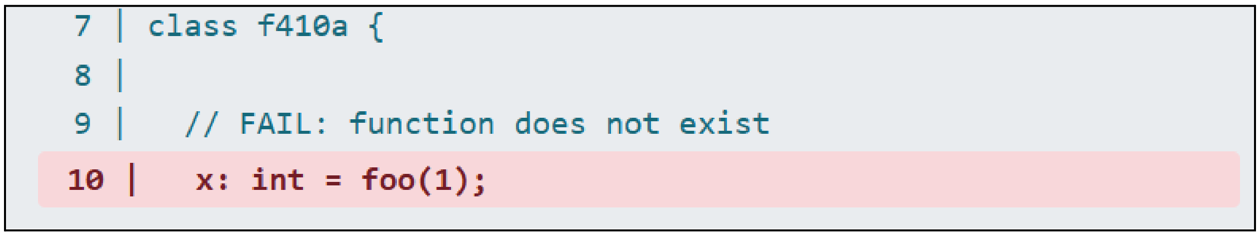


Error Details

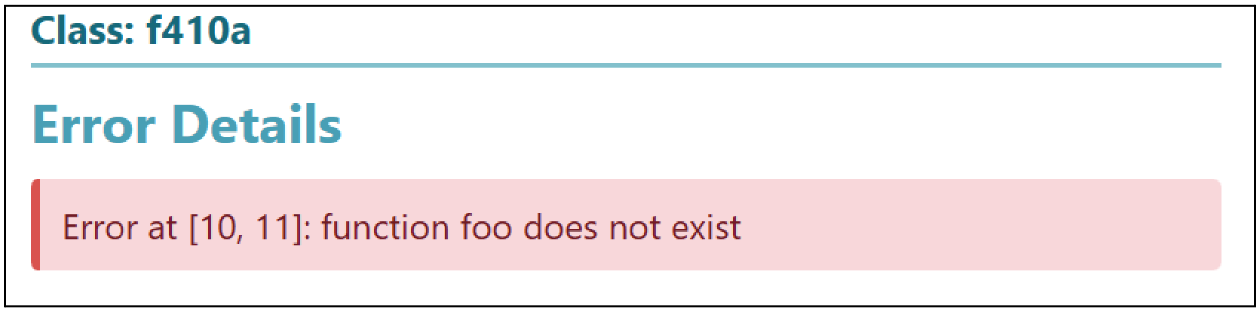
```
Error at [16, 1]: class fail already exists
```

4.10 Function calls must match their declarations

4.10.1 Functions must be declared to be invoked



```
 7 | class f410a {
 8 |
 9 |   // FAIL: function does not exist
10 |   x: int = foo(1);
```

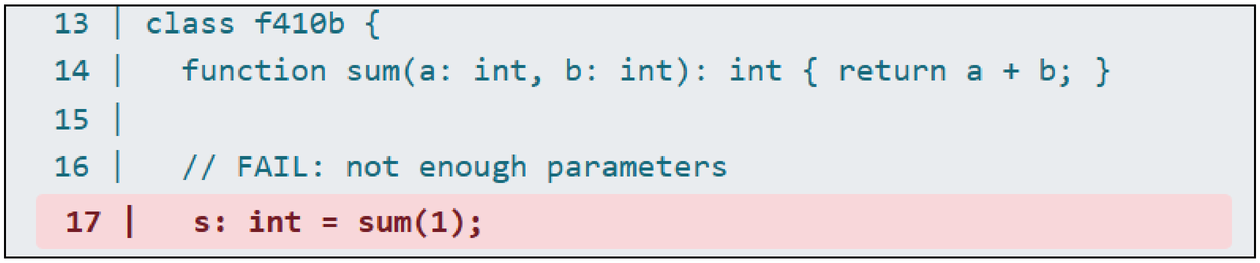


Class: f410a

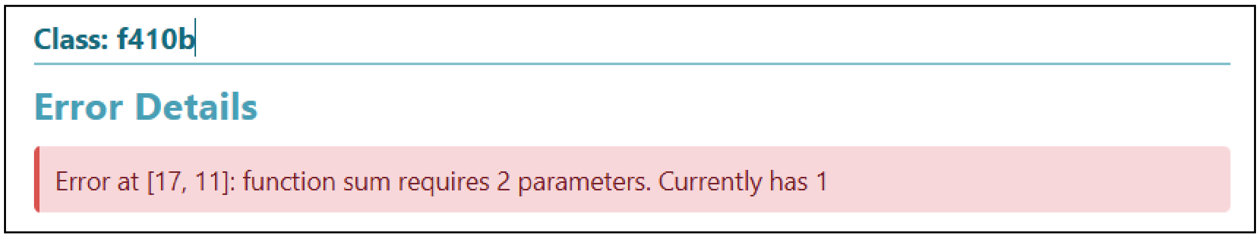
Error Details

Error at [10, 11]: function foo does not exist

4.10.2 Function calls must include all parameters from declaration



```
13 | class f410b {
14 |   function sum(a: int, b: int): int { return a + b; }
15 |
16 |   // FAIL: not enough parameters
17 |   s: int = sum(1);
```

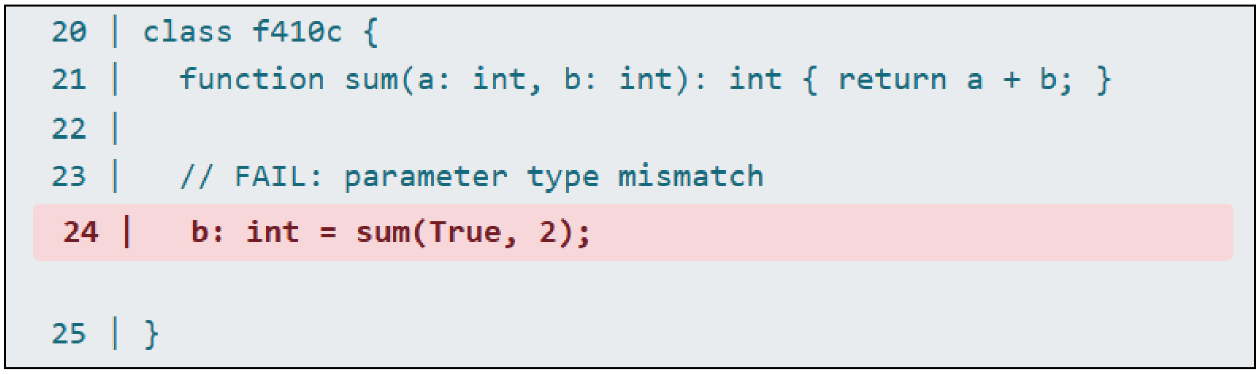


Class: f410b

Error Details

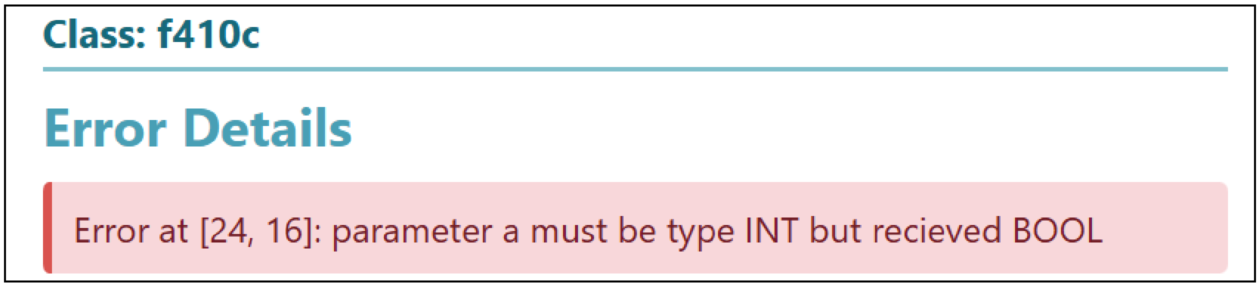
Error at [17, 11]: function sum requires 2 parameters. Currently has 1

4.10.3 Function parameters must match declration parameter types



```
20 | class f410c {
21 |   function sum(a: int, b: int): int { return a + b; }
22 |
23 |   // FAIL: parameter type mismatch
24 |   b: int = sum(True, 2);

25 | }
```



Class: f410c

Error Details

```
Error at [24, 16]: parameter a must be type INT but recieved BOOL
```

5 Summary of Submitted Examples

The examples are contained in **src/tests/finalSubmission/section5**.
Many of the input files cover multiple items. In general all files include:

- Variable Declaration
- Variable Assignment
- Class Declaration(s)
- Type-checking
- Expression evaluation

5.1 Highlights of Example Input 1

- bank_account.txt
    - Variable Comparison Operators, and logic

5.2 Highlights of Example Input 2

- calculator.txt
    - Decision Nodes: If statements, conditional expressions, and branching

5.3 Highlights of Example Input 3

- employee.txt
    - Boolean logic, truth values like True/False, logical operations

5.4 Highlights of Example Input 4

- fleet.txt
    - Variable scoping, function declaration, if statements, conditional execution, variable modification

5.5 Highlights of Example Input 5

- gaming.txt
    - Scoping of local vs class variable

5.6 Highlights of Example Input 6

- inventory.txt
    - Inheritance, parent-child class relationship

5.7 Highlights of Example Input 7

- library.txt
    - while-loop, boolean loop and exit condition

5.8 Highlights of Example Input 8

- student.txt
    - Variable operations, function call, function returning

5.9 Highlights of Example Input 9

- task_manager.txt
    - for-loop, incrementing loop condition verification

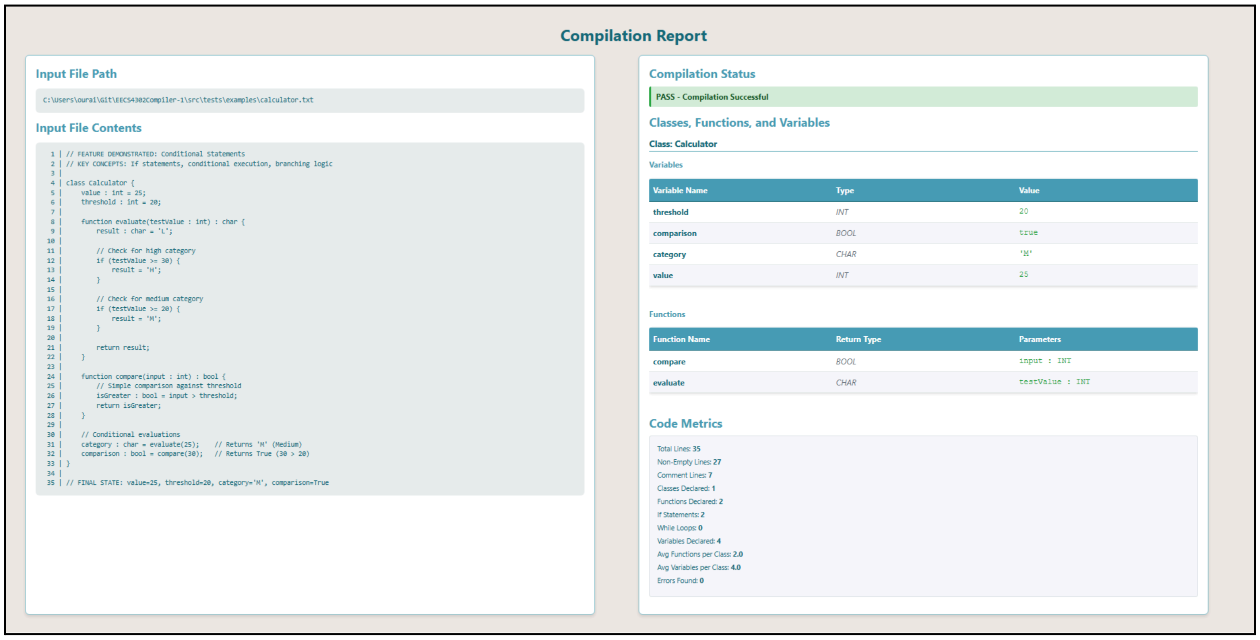
5.10 Highlights of Example Input 10

- weather.txt
  - Character list, list assignments, list manipulation

6 Miscellaneous Features

6.1 Formatted HTML Report

- Clean, easy to read report with clear results
- Original input file, file path, variables, functions, and classes are listed
- Code Metrics



Compilation Report

Input File Path

```
C:\Users\ourai\Git\EECS4302Compiler-1\src\tests\examples\calculator.txt
```

Input File Contents

```
// FEATURE DEMONSTRATED: Conditional Statements
// KEY CONCEPTS: If statements, conditional execution, branching logic

class Calculator {
    value : int = 25;
    threshold : int = 20;

    function evaluate(testValue : int) : char {
        result : char = 'L';

        // Check for high category
        if (testValue >= 30) {
            result = 'H';
        }

        // Check for medium category
        if (testValue >= 20) {
            result = 'M';
        }

        return result;
    }

    function compare(input : int) : bool {
        // Simple comparison against threshold
        isGreater : bool = input > threshold;
        return isGreater;
    }

    // Conditional evaluations
    category : char = evaluate(25);    // Returns 'M' (Medium)
    comparison : bool = compare(30);   // Returns True (30 > 20)
}

// FINAL STATE: value=25, threshold=20, category='M', comparison=True
```

Compilation Status

PASS - Compilation Successful

Classes, Functions, and Variables

Class: Calculator

Variables

Variable Name	Type	Value
threshold	*INT*	`20`
comparison	*BOOL*	`true`
category	*CHAR*	`'M'`
value	*INT*	`25`

Functions

Function Name	Return Type	Parameters
compare	*BOOL*	`input : INT`
evaluate	*CHAR*	`testValue : INT`

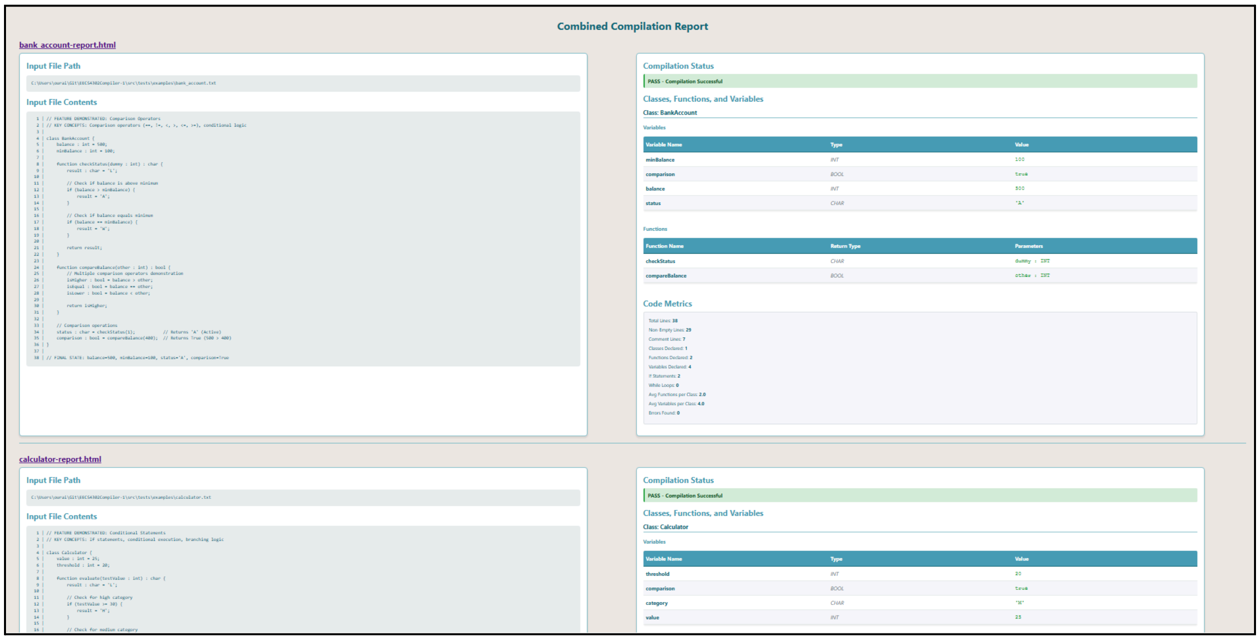
Code Metrics

```
Total Lines: 35
Non-Empty Lines: 27
Comment Lines: 7
Classes Declared: 1
Functions Declared: 2
If Statements: 2
While Loops: 0
Variables Declared: 4
Avg Functions per Class: 2.0
Avg Variables per Class: 4.0
Errors Found: 0
```

6.2 Multiple Input File Support

- Able to call a directory containing multiple files, and compile all these files at once
- The files are processed sequentially, and all reports are generated at the end

- A combined report is also generated, which shows all the files on one report, with hyperlinks to each individual report if needed



Combined Compilation Report

bank_account-report.html

Input File Path

```
C:\Users\ourai\Git\EECS4302Compiler-1\src\tests\examples\bank_account.txt
```

Input File Contents

```
// FEATURE DEMONSTRATED: Comparison Operators
// KEY CONCEPTS: Comparison operators (==, !=, <, >, <=, >=), conditional logic

class BankAccount {
    balance : int = 500;
    minBalance : int = 100;

    function checkStatus(dummy : int) : char {
        result : char = 'L';

        // Check if balance is above minimum
        if (balance > minBalance) {
            result = 'A';
        }

        // Check if balance equals minimum
        if (balance == minBalance) {
            result = 'W';
        }

        return result;
    }

    function compareBalance(other : int) : bool {
        // Multiple comparison operators demonstration
        isHigher : bool = balance > other;
        isEqual : bool = balance == other;
        isLower : bool = balance < other;

        return isHigher;
    }

    // Comparison operations
    status : char = checkStatus(1);           // Returns 'A' (Active)
    comparison : bool = compareBalance(400);  // Returns True (500 > 400)
}

// FINAL STATE: balance=500, minBalance=100, status='A', comparison=True
```

Compilation Status

PASS - Compilation Successful

Classes, Functions, and Variables

Class: BankAccount

Variables

Variable Name	Type	Value
minBalance	INT	100
comparison	BOOL	true
balance	INT	500
status	CHAR	'A'

Functions

Function Name	Return Type	Parameters
checkStatus	CHAR	dummy : INT
compareBalance	BOOL	other : INT

Code Metrics

Total Lines: **38**
Non Empty Lines: **29**
Comment Lines: **7**
Classes Declared: **1**
Functions Declared: **2**
Variables Declared: **4**
If Statements: **2**
While Loops: **0**
Avg Functions per Class: **2.0**
Avg Variables per Class: **4.0**
Errors Found: **0**

calculator-report.html

Input File Path

```
C:\Users\ourai\Git\EECS4302Compiler-1\src\tests\examples\calculator.txt
```

Input File Contents

```
// FEATURE DEMONSTRATED: Conditional Statements
// KEY CONCEPTS: If statements, conditional execution, branching logic

class Calculator {
    value : int = 25;
    threshold : int = 20;

    function evaluate(testValue : int) : char {
        result : char = 'L';

        // Check for high category
        if (testValue >= 30) {
            result = 'H';
        }

        // Check for medium category
```

Compilation Status

PASS - Compilation Successful

Classes, Functions, and Variables

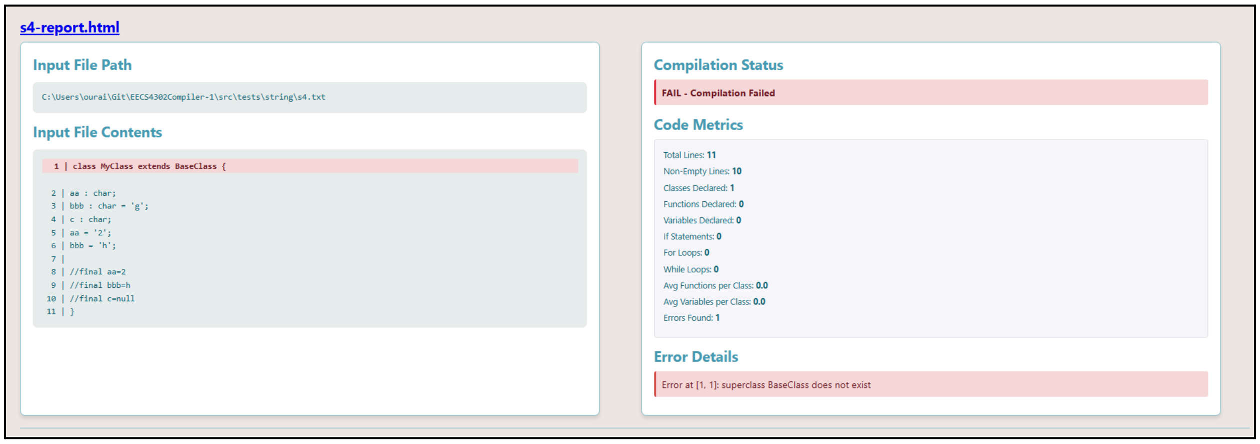
Class: Calculator

Variables

Variable Name	Type	Value
threshold	INT	20
comparison	BOOL	true
category	CHAR	'M'
value	INT	25

6.3 Error Highlighting

- Semantic errors are highlighted in red, so they are easier to identify in the input text



s4-report.html

Input File Path

```
C:\Users\ourai\Git\EECS4302Compiler-1\src\tests\string\s4.txt
```

Input File Contents

```
class MyClass extends BaseClass {

aa : char;
bbb : char = 'g';
c : char;
aa = '2';
bbb = 'h';

//final aa=2
//final bbb=h
//final c=null
}
```

Compilation Status

FAIL - Compilation Failed

Code Metrics

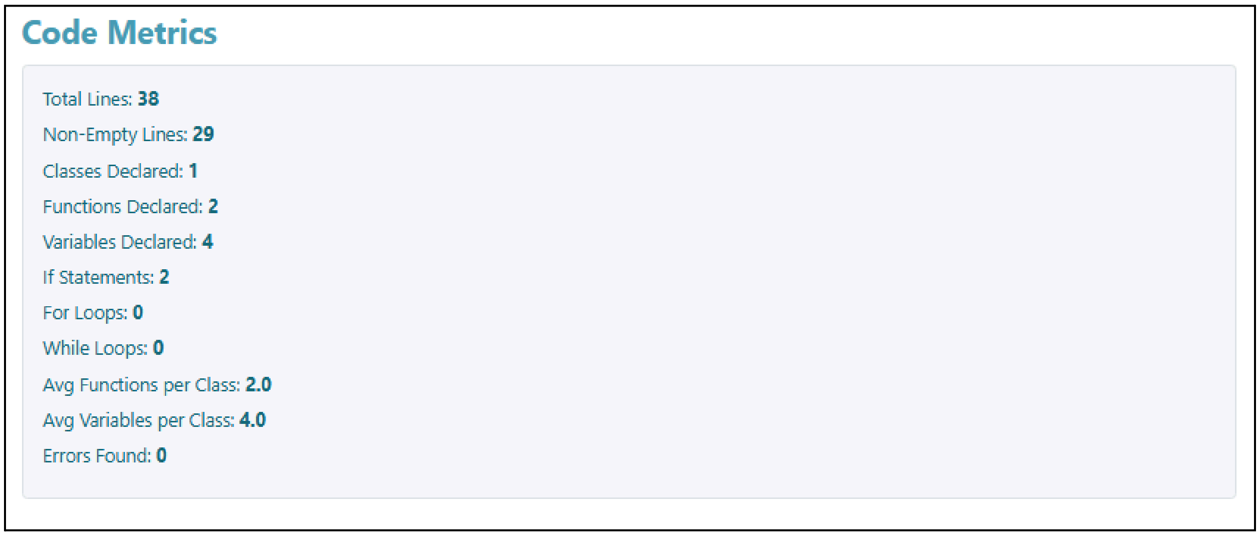
Total Lines: **11**
Non-Empty Lines: **10**
Classes Declared: **1**
Functions Declared: **0**
Variables Declared: **0**
If Statements: **0**
For Loops: **0**
While Loops: **0**
Avg Functions per Class: **0.0**
Avg Variables per Class: **0.0**
Errors Found: **1**

Error Details

Error at [1, 1]: superclass BaseClass does not exist

6.4 Code Metrics

- The following metrics are tracked on a per file basis, this is calculated at report generation time
- Total Lines
- Non-Empty Lines
- Comment Lines
- Classes Declared
- Functions Declared
- Variables Declared
- If Statements
- For Loops
- While Loops

- Avg Functions per Class
- Avg Variables per Class
- Errors Found
- 

Code Metrics

Total Lines: **38**

Non-Empty Lines: **29**

Classes Declared: **1**

Functions Declared: **2**

Variables Declared: **4**

If Statements: **2**

For Loops: **0**

While Loops: **0**

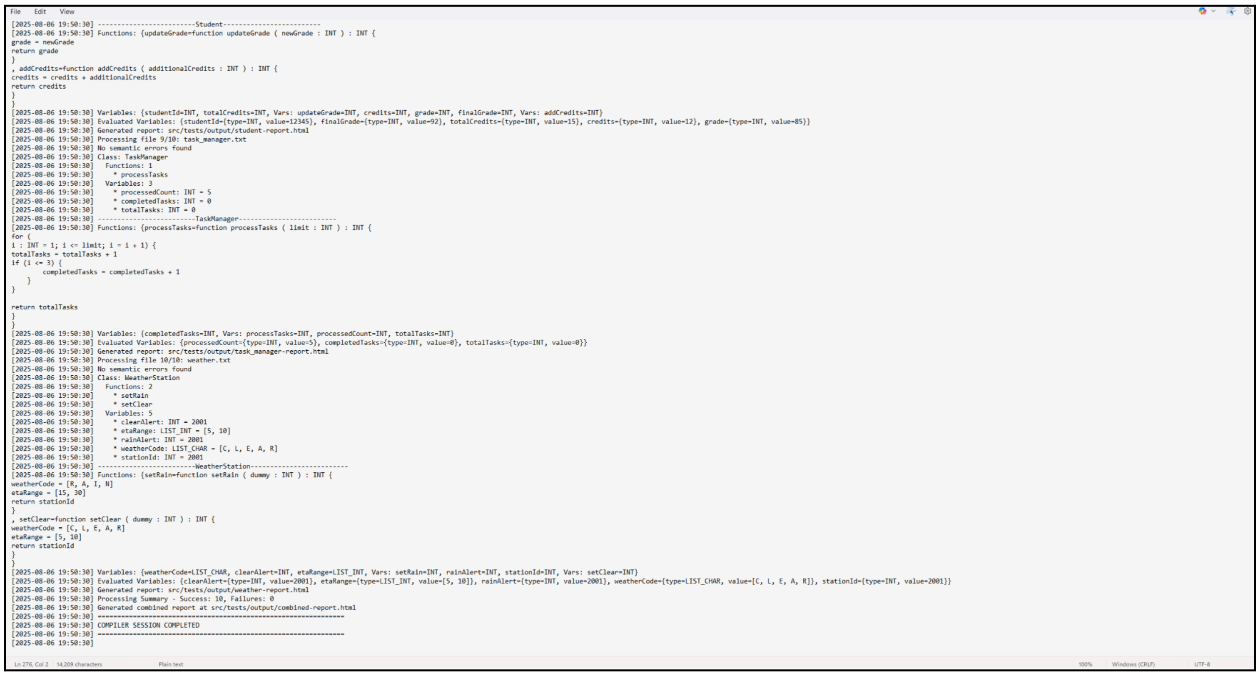
Avg Functions per Class: **2.0**

Avg Variables per Class: **4.0**

Errors Found: **0**

6.5 Log Retention

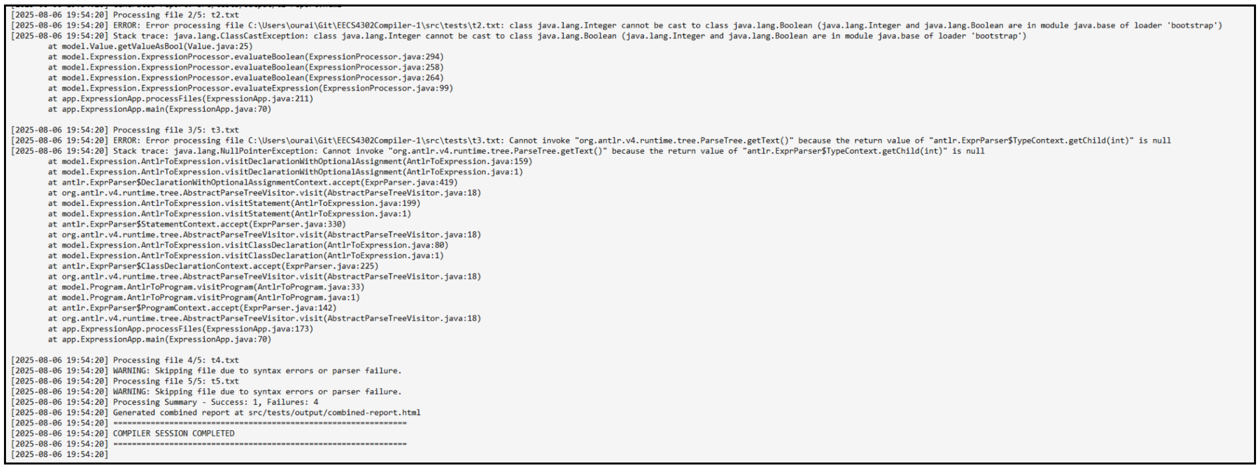
- For more detailed analysis, we have created a history.log file within the tests/output folder
- This is a rolling file, which appends the console outputs and includes details about each file similar to the report, but does not get overridden, and retains a record of old executions.



```
[2025-08-06 19:50:30] ------------------------Student------------------------
[2025-08-06 19:50:30] Functions: {updateGrade=function updateGrade ( newGrade : INT ) : INT {
grade = newGrade
return grade
}
, addCredits=function addCredits ( additionalCredits : INT ) : INT {
credits = credits + additionalCredits
return credits
}
}
[2025-08-06 19:50:30] Variables: {studentId=INT, totalCredits=INT, Vars: updateGrade=INT, credits=INT, grade=INT, finalGrade=INT, Vars: addCredits=INT}
[2025-08-06 19:50:30] Evaluated Variables: {studentId={type=INT, value=12345}, finalGrade={type=INT, value=92}, totalCredits={type=INT, value=15}, credits={type=INT, value=12}, grade={type=INT, value=85}}
[2025-08-06 19:50:30] Generated report: src/tests/output/student-report.html
[2025-08-06 19:50:30] Processing file 9/10: task_manager.txt
[2025-08-06 19:50:30] No semantic errors found
[2025-08-06 19:50:30] Class: TaskManager
[2025-08-06 19:50:30]   Functions: 1
[2025-08-06 19:50:30]     * processTasks
[2025-08-06 19:50:30]   Variables: 3
[2025-08-06 19:50:30]     * processedCount: INT = 5
[2025-08-06 19:50:30]     * completedTasks: INT = 0
[2025-08-06 19:50:30]     * totalTasks: INT = 0
[2025-08-06 19:50:30] ------------------------TaskManager------------------------
[2025-08-06 19:50:30] Functions: {processTasks=function processTasks ( limit : INT ) : INT {
for (
i : INT = 1; i <= limit; i = i + 1) {
totalTasks = totalTasks + 1
if (i <= 3) {
        completedTasks = completedTasks + 1
    }
}

return totalTasks
}
}
[2025-08-06 19:50:30] Variables: {completedTasks=INT, Vars: processTasks=INT, processedCount=INT, totalTasks=INT}
[2025-08-06 19:50:30] Evaluated Variables: {processedCount={type=INT, value=5}, completedTasks={type=INT, value=0}, totalTasks={type=INT, value=0}}
[2025-08-06 19:50:30] Generated report: src/tests/output/task_manager-report.html
[2025-08-06 19:50:30] Processing file 10/10: weather.txt
[2025-08-06 19:50:30] No semantic errors found
[2025-08-06 19:50:30] Class: WeatherStation
[2025-08-06 19:50:30]   Functions: 2
[2025-08-06 19:50:30]     * setRain
[2025-08-06 19:50:30]     * setClear
[2025-08-06 19:50:30]   Variables: 5
[2025-08-06 19:50:30]     * clearAlert: INT = 2001
[2025-08-06 19:50:30]     * etaRange: LIST_INT = [5, 10]
[2025-08-06 19:50:30]     * rainAlert: INT = 2001
[2025-08-06 19:50:30]     * weatherCode: LIST_CHAR = [C, L, E, A, R]
[2025-08-06 19:50:30]     * stationId: INT = 2001
[2025-08-06 19:50:30] ------------------------WeatherStation------------------------
[2025-08-06 19:50:30] Functions: {setRain=function setRain ( dummy : INT ) : INT {
weatherCode = [R, A, I, N]
etaRange = [15, 30]
return stationId
}
, setClear=function setClear ( dummy : INT ) : INT {
weatherCode = [C, L, E, A, R]
etaRange = [5, 10]
return stationId
}
}
[2025-08-06 19:50:30] Variables: {weatherCode=LIST_CHAR, clearAlert=INT, etaRange=LIST_INT, Vars: setRain=INT, rainAlert=INT, stationId=INT, Vars: setClear=INT}
[2025-08-06 19:50:30] Evaluated Variables: {clearAlert={type=INT, value=2001}, etaRange={type=LIST_INT, value=[5, 10]}, rainAlert={type=INT, value=2001}, weatherCode={type=LIST_CHAR, value=[C, L, E, A, R]}, stationId={type=INT, value=2001}}
[2025-08-06 19:50:30] Generated report: src/tests/output/weather-report.html
[2025-08-06 19:50:30] Processing Summary - Success: 10, Failures: 0
[2025-08-06 19:50:30] Generated combined report at src/tests/output/combined-report.html
[2025-08-06 19:50:30] ============================================================
[2025-08-06 19:50:30] COMPILER SESSION COMPLETED
[2025-08-06 19:50:30] ============================================================
[2025-08-06 19:50:30]
```

- This is the way the user can debug syntax errors, as they do not show up in the html report.



```
[2025-08-06 19:54:20] Processing file 2/5: t2.txt
[2025-08-06 19:54:20] ERROR: Error processing file C:\Users\ourai\Git\EECS4302Compiler-1\src\tests\t2.txt: class java.lang.Integer cannot be cast to class java.lang.Boolean (java.lang.Integer and java.lang.Boolean are in module java.base of loader 'bootstrap')
[2025-08-06 19:54:20] Stack trace: java.lang.ClassCastException: class java.lang.Integer cannot be cast to class java.lang.Boolean (java.lang.Integer and java.lang.Boolean are in module java.base of loader 'bootstrap')
        at model.Value.getValueAsBool(Value.java:25)
        at model.Expression.ExpressionProcessor.evaluateBoolean(ExpressionProcessor.java:294)
        at model.Expression.ExpressionProcessor.evaluateBoolean(ExpressionProcessor.java:258)
        at model.Expression.ExpressionProcessor.evaluateBoolean(ExpressionProcessor.java:264)
        at model.Expression.ExpressionProcessor.evaluateExpression(ExpressionProcessor.java:99)
        at app.ExpressionApp.processFiles(ExpressionApp.java:211)
        at app.ExpressionApp.main(ExpressionApp.java:70)

[2025-08-06 19:54:20] Processing file 3/5: t3.txt
[2025-08-06 19:54:20] ERROR: Error processing file C:\Users\ourai\Git\EECS4302Compiler-1\src\tests\t3.txt: Cannot invoke "org.antlr.v4.runtime.tree.ParseTree.getText()" because the return value of "antlr.ExprParser$TypeContext.getChild(int)" is null
[2025-08-06 19:54:20] Stack trace: java.lang.NullPointerException: Cannot invoke "org.antlr.v4.runtime.tree.ParseTree.getText()" because the return value of "antlr.ExprParser$TypeContext.getChild(int)" is null
        at model.Expression.AntlrToExpression.visitDeclarationWithOptionalAssignment(AntlrToExpression.java:159)
        at model.Expression.AntlrToExpression.visitDeclarationWithOptionalAssignment(AntlrToExpression.java:1)
        at antlr.ExprParser$DeclarationWithOptionalAssignmentContext.accept(ExprParser.java:419)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visit(AbstractParseTreeVisitor.java:18)
        at model.Expression.AntlrToExpression.visitStatement(AntlrToExpression.java:199)
        at model.Expression.AntlrToExpression.visitStatement(AntlrToExpression.java:1)
        at antlr.ExprParser$StatementContext.accept(ExprParser.java:330)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visit(AbstractParseTreeVisitor.java:18)
        at model.Expression.AntlrToExpression.visitClassDeclaration(AntlrToExpression.java:80)
        at model.Expression.AntlrToExpression.visitClassDeclaration(AntlrToExpression.java:1)
        at antlr.ExprParser$ClassDeclarationContext.accept(ExprParser.java:225)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visit(AbstractParseTreeVisitor.java:18)
        at model.Program.AntlrToProgram.visitProgram(AntlrToProgram.java:33)
        at model.Program.AntlrToProgram.visitProgram(AntlrToProgram.java:1)
        at antlr.ExprParser$ProgramContext.accept(ExprParser.java:142)
        at org.antlr.v4.runtime.tree.AbstractParseTreeVisitor.visit(AbstractParseTreeVisitor.java:18)
        at app.ExpressionApp.processFiles(ExpressionApp.java:173)
        at app.ExpressionApp.main(ExpressionApp.java:70)

[2025-08-06 19:54:20] Processing file 4/5: t4.txt
[2025-08-06 19:54:20] WARNING: Skipping file due to syntax errors or parser failure.
[2025-08-06 19:54:20] Processing file 5/5: t5.txt
[2025-08-06 19:54:20] WARNING: Skipping file due to syntax errors or parser failure.
[2025-08-06 19:54:20] Processing Summary - Success: 1, Failures: 4
[2025-08-06 19:54:20] Generated combined report at src/tests/output/combined-report.html
[2025-08-06 19:54:20] ===============================================================
[2025-08-06 19:54:20] COMPILER SESSION COMPLETED
[2025-08-06 19:54:20] ===============================================================
[2025-08-06 19:54:20]
```

7 Limitations

Not Supported:

- **Method overloading, overriding:** For simplicity's sake, every function name must be unique.
- **Nested functions:** Functions cannot be declared inside other functions or within loops.
- **List operations:** Only list declarations and direct assignments are supported; dynamic operations like adding or removing elements are not implemented.
- **Character expressions:** Equality and relational operations involving char types are not supported. These expressions currently only work with int and bool types.
- **No anonymous/lambda functions:** All functions must be named and declared at the class level.
- **No string operations or methods:** Strings are allowed ("...") but the compiler does not currently support operations like concatenation or indexing.
- **No operator precendence enforcement in grammar**

Just Out of Scope (due to time constraints):

These features were not implemented but could be supported in future iterations:

- Object creation and dot notation (e.g., new ClassName(), object.field)
- Private variables and access modifiers
- else if / else branches
- do-while loops
- print statements