

How I approached Project 4

According to the requirement PDF, there seemed to be a lot of modifications to be made so first thing I did was make a list of all the changes needed, and the rules (where most of it was common sense). When I checked the Project 4 skeleton code, I noticed that there were new additions to the Project 4 original code compared to the past skeleton codes. I began transferring some of my work - such as new operator additions, actions to take input, and additional parsing methods and grammar rules - from Project 2 and Project 3 and fitting them ~~by line~~ to the Project 4 skeleton code.

ANDOP, REMOP, EXPOP

I added the new tokens andop, remop, and expop to the parser and applied checking methods on them. Andop got its logic checked, while Expop and remop got its arithmetic checked.

Token types

I noticed that Project 4 skeleton code didn't have value and operator token types, just "type". Changed all the tokens <value> to <type> in the bison declarations section.

Semantic error messages

Then I focused on getting a program that compiled first before attempting anything else, though it might not actually do anything yet. Then I tried running the program on the test samples provided from the lecture, and the test case from the PDF to see what happens. For the latter case, the only errors that popped up were semantic errors but not exactly like in the PDF (pic included):

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test5.txt
1  -- Test of Multiple Semantic Errors from Project 4 PDF
2  -- error is a+5, should be boolean
3  -- error is undefined f in 7=f
4  -- error is case type mismatch as b is bool while c is real
5  -- total of 3 errs
6
7  function test a: integer returns integer;
8    b: integer is
9      if a + 5 then
10         2;
11      else
12         5;
13      endif;
Semantic Error, Type Mismatch on Variable Initialization
14    c: real is 9.8 - 2 + 8;
Semantic Error, Type Mismatch on Variable Initialization
15    d: boolean is 7 = f;
Semantic Error, Undeclared f
16  begin
17    case b is
18      when 1 => 4.5 + c;
19      when 2 => b;
20      others => c;
21    endcase;
22  end;
23
24
25

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 2
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test1.txt
```

The semantic error messages were stored in listing.cc. This gave me an idea on how to start modifying the checking methods.

Checking methods

I noticed that the skeleton code included “types.cc” that had checking methods in them, and I saw how they applied to the tokens and their grammar rules in the parser. There was check arithmetic, logical, and relational.

The first error that popped on the initial run on the PDF test case was “type mismatch”, which came from the checkAssignment method in types.cc. “Variable Initialization” was from the variable method in the parser, which means that the parser found error in the variable statement in the PDF test case, in the “if else” part. For the checkAssignment, it compares the types of the left side and the right side. So in this test’s case, “integer” would be the left side, and the result of “if...else” would be the right side. The result of “if else” should be an integer too, but it can’t come to that conclusion because the expression “a +5” should be a boolean before “if else” can come up with an integer result, and “a” in this case is an integer according to the parameter. I needed a way to let the program detect this error. Hence, in the checkAssignment method, there needed to be more specific scenarios written for specific problems rather than the general semantic message.

Check to the checkAssignment function for mismatched types in the case that Boolean and numeric types are mixed.

The original checkAssignment function returned a generic error message, but not the specific kinds. So I nested three scenarios for the checkAssignment function: IF there is an error, then check: if a narrowing variable initialization occurs from integer to real (but real to integer is okay), if a boolean is expected but the result isn’t a boolean (such as in the test case mentioned before), or if a real or integer type is expected but a boolean occurs. I was going to write a scenario for if an integer is expected but a boolean appears, but then I realized that boolean can look like an integer (0 or 1) so it would mess up the program. I didn’t include it.

If-then checking methods

I assumed I needed to make new checking methods for the “if then” and “case is” grammar rules in order for the test case from the PDF to provide specific errors messages. I made a void method “checkIf” to extract the expression, then, and else statements from the “if then” statement so it can be used in the types.cc file. Then I mentioned checkIf in the types.h file. If expression must either be true or false, and the “then” and “else” statements should both be the same types.

Case-is checking method

This one was more complicated, but since I tried parsing the case statement back in Project 3, I took a similar route. I assumed that the result of the case types from all the “case” options and the “other statement” should all match the case expression’s type. I made a void function to extract the case expression’s type from the case statement first, so it can be used for reference

and comparison in types.cc when checking the type for the “case” options and the “other statement”. I made three functions: caseExpression to get the case expression’s type, checkCase for the case options, and checkOtherStatement for the other statement.

Addition to checkArithmetic function to coerce integers to reals when the types are mixed

Coerce means convert. In a math function, integers can turn into real for calculation when the types are mixed. In order to do this, I made two “if” statements where “if left type is real and right type is integer, return real type” and vice versa. Otherwise, return integer as usual. Then I edited the “integer type required” to “numeric type required”.

Addition of check to symbols.h for duplicate identifiers

Duplicate is like find, because there is either a duplicate or not - similar to “bool find”, where it’s either found or it’s not. I used “find” in the parser to check for duplicates by adding it to the variable method’s list of actions to check for duplicates, and copy pasted the action from the “primary” method’s options, and changed the appended error from “undeclared” to “duplicate_identifier”.

REMOP checking function

The last thing I worked on was the remainder check, where the function should check whether the remainder is an integer and not a real number itself. The rem operator should check that both the whole number and the remainder are integers, meaning both the left and right operands on each side of the “rem” operator should be integers.

Test Plan:

I made 14 test files (included in the ZIP file): 4 from the video lectures and variations of them, 6th from the Project Requirement PDF, and 6b and 6c are variations of the PDF test case to specifically test certain parts of the program. Test 5 is self made to test non-integer remainders and duplicates. To see if my modification of the skeletal codes worked, I compared my modified code to the skeletal code.

Test case	Expected output	Met expectation?
	Compiled Successfully	Yes
	Compiled Successfully	
1c, but returns real	Compiled Successfully	
	Compiled Successfully	Yes
	Throw error due to real literal instead of integer	
2c, but "b: boolean is 5" and mismatch of types	Throw 2 errors: due to invalid boolean value and mismatch of types	
3, from video lecture	Compiled Successfully	Yes
	Throw error due to invalid boolean value	Yes
	Throw error due to invalid value for relational operation	Yes
4, from video lecture	Compiled Successfully	Yes
	Throw 3 errors due to non-integer operands around the "rem" operator, and a duplicate identifier	Yes
6 from Project req. PDF	Throw 3 errors exactly like the PDF sample	Yes
	Throw error as the then and else statements are mismatched types	
6c, like Project PDF test case but without any issues	Compiled Successfully	Yes

Test1:

This tests if the parser reads and passes valid arithmetic expressions and operators.

```
1  -- Function with arithmetic expression
2
3  function test1 returns integer;
4  begin
5      7 + 2 * (2 + 4);
6  end;
```

Compilation for test 1:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test1.txt

1  -- Function with arithmetic expression
2
3  function test1 returns integer;
4  begin
5      7 + 2 * (2 + 4);
6  end;

Compiled Successfully
```

This compiled successfully as there were no issues due to no semantic errors

Test 1b:

This one is like Test 1, but has a real number wedged in the operation. It should pass because coercion from integer to real number is fine in this case.

```
-- Function with arithmetic expression

function test1 returns integer;
begin
    7 + 2 * 3.5;
end;
```

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test1b.txt

1  -- Function with arithmetic expression
2
3  function test1 returns integer;
4  begin
5      7 + 2 * 3.5;
6  end;

Compiled Successfully
```

Test 1c: This is like 1b, but returns real. Again should compile.

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test1c.txt

1  -- Function with arithmetic expression
2
3  function test1 returns real;
4  begin
5      7 + 2 * 3.5;
6  end;

Compiled Successfully
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ make
```

Test 2:

This should compile as there are no semantic issues. Everything is integer type.

```
File Edit Format View Help
|-- Function with a Variable

function test2 returns integer;
    b: integer is 9 * 2 + 8;
begin
    b + 2 * 8;
end;
```

Compilation for test 2:

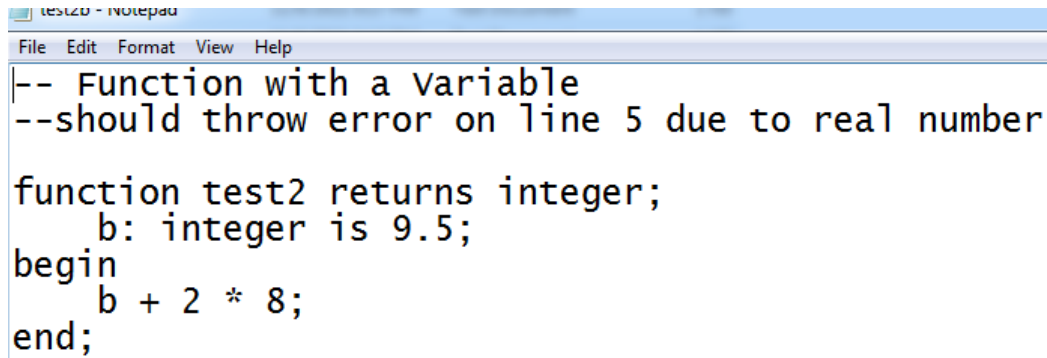
```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test2.txt

1  -- Function with a Variable
2
3  function test2 returns integer;
4      b: integer is 9 * 2 + 8;
5  begin
6      b + 2 * 8;
7  end;

Compiled Successfully
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ make
```

Test 2b:

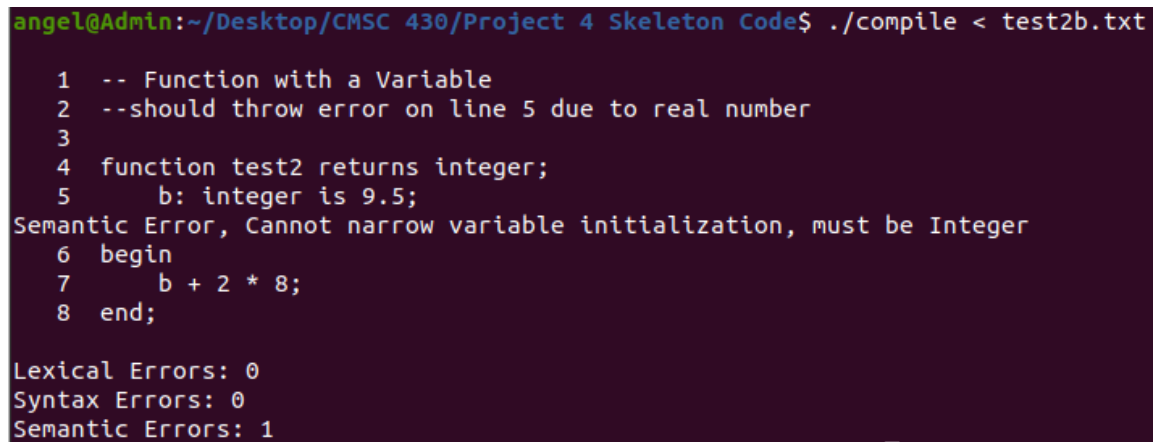
This one says b:integer but the value 9.5 is a real number. It should throw an error stating it should be an integer and that narrow variable initialization is not allowed. *This shows that the program generates semantic error on narrowing initialization.*



```
-- Function with a Variable
--should throw error on line 5 due to real number

function test2 returns integer;
    b: integer is 9.5;
begin
    b + 2 * 8;
end;
```

Compilation:



```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test2b.txt

1  -- Function with a Variable
2  --should throw error on line 5 due to real number
3
4  function test2 returns integer;
5      b: integer is 9.5;
Semantic Error, Cannot narrow variable initialization, must be Integer
6  begin
7      b + 2 * 8;
8  end;

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 1
```

Type 2c:

This should throw an error because "5" is not a boolean value, and another error because b is not real or an integer type to be calculated with numeric types. *This shows that the program generates semantic error when if condition is not Boolean, and that boolean expressions cannot be used with arithmetic or relational operators.*

```
File Edit Format View Help
-- Function with a Variable
--should throw error on line 5 since it's not boolean, and line 7 since b
is not numeric type

function test2 returns boolean;
    b: boolean is 5;
begin
    b + 2 * 8;
end;
```

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test2c.txt

1  -- Function with a Variable
2  --should throw error on line 5 since it's not boolean, and line 7 since b
is not numeric type
3
4  function test2 returns boolean;
5      b: boolean is 5;
Semantic Error, must be Boolean
6  begin
7      b + 2 * 8;
Semantic Error, Numeric Type Required
8  end;

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 2
```

Test3:

This should compile with no issue as all returned types are boolean.

```
File Edit Format View Help
-- Function with an Boolean Variable

function test3 returns boolean;
    b: boolean is 5 < 2;
begin
    b and 2 < 8;
end;
```


Compilation for test 3:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test3.txt

1  -- Function with an Boolean Variable
2
3  function test3 returns boolean;
4      b: boolean is 5 < 2;
5  begin
6      b and 2 < 8;
7  end;

Compiled Successfully
```

Test 3b:

This should throw error since "5" is not a boolean value. This shows that the program generates semantic error when if condition is not Boolean.

```
File Edit Format View Help
|-- Function with an Boolean Variable
|-- should throw error on line 5 due to integer

function test3 returns boolean;
    b: boolean is 5;
begin
    b and 2 < 8;
end;
```

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test3b.txt

1  -- Function with an Boolean Variable
2  -- should throw error on line 5 due to integer
3
4  function test3 returns boolean;
5      b: boolean is 5;
Semantic Error, must be Boolean
6  begin
7      b and 2 < 8;
8  end;

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 1
```

Test 3c:

This should throw an error because "2" is not a boolean value.

```
-- Function with an Boolean Variable
-- should throw error on line 7 due to integer

function test3 returns boolean;
    b: boolean is 5 < 2;
begin
    b and 2;
end;
```

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test3c.txt

1  -- Function with an Boolean Variable
2  --should throw error on line 7 due to integer
3
4  function test3 returns boolean;
5      b: boolean is 5 < 2;
6  begin
7      b and 2;
Semantic Error, Boolean Type Required
8  end;

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 1
```

Test 4:

This test should compile with no problem, and it shows that reductions work.

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test4.txt

1  -- Function with a Reduction
2
3  function test4 returns integer;
4  begin
5      reduce +
6          2 * 8;
7          6;
8          23 + 6;
9      endreduce;
10 end;

Compiled Successfully
```

Test5:

This is a test to see if all the non-integer operands for the "rem" operand are caught, and to catch the duplicate "a". There should be 3 errors: two for invalid operands for "rem", and one for duplicate. *This shows that the program generates semantic error when a remainder operator has non-integer operands, and that it generates semantic error for duplicate variables.*

```
-- Function with non-integer remainders, duplicate
```

```
function test4 returns integer;
    a: integer is 5;
    b: real is 4 rem 3;
    c: real is 4.5 rem 3;
    d: real is 4 rem 3.5;
    a: integer is 2;
begin
    a + 10;
end;
```

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test5.txt

1  -- Function with non-integer remainders, duplicate
2
3  function test4 returns integer;
4      a: integer is 5;
5      b: real is 4 rem 3;
6      c: real is 4.5 rem 3;
Semantic Error, Both whole and remainder should be Integers
7      d: real is 4 rem 3.5;
Semantic Error, Both whole and remainder should be Integers
8      a: integer is 2;
Semantic Error, Duplicate Identifier: a
9  begin
10     a + 10;
11 end;

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 3
```

Test6:

The test case is the sample from the Project 4 requirement PDF. The program generates semantic error when case types don't match and that it generates semantic error when case expression is not integral along with other criteria. I wrote more notes in the text file but otherwise the errors should be exactly the same:

```
function test a: integer returns integer;
    b: integer is
        if a + 5 then
            2;
        else
            5;
        endif;
    c: real is 9.8 - 2 + 8;
    d: boolean is 7 = f;
begin
    case b is
        when 1 => 4.5 + c;
        when 2 => b;
        others => c;
    endcase;
end;
```

Compilation:

```
7 function test a: integer returns integer;
8     b: integer is
9         if a + 5 then
10             2;
11         else
12             5;
13         endif;
Semantic Error, If Expression must be Boolean
14     c: real is 9.8 - 2 + 8;
15     d: boolean is 7 = f;
Semantic Error, Undeclared f
16 begin
17     case b is
18         when 1 => 4.5 + c;
19         when 2 => b;
20         others => c;
21     endcase;
Semantic Error, Case Type Mismatch in Other statement
22 end;
23
24
25

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 3
```

Test 6b:

This is a variation of the project PDF test case, but edited to focus on finding a mismatch error in the types for the "if then" statement and the "else" statement (one is a real ~~type~~ ^{value}). *It shows that the program generates semantic error when if and then types don't match.*

```
-- Test of Multiple Semantic Errors from Project 4 PDF
-- type mismatch err on line 9 as it's not integer

function test returns integer;
  a: integer is 3;
  f: integer is 7;
  b: integer is
    if a > 1 then
      2.5;
    else
      5;
    endif;
  c: integer is 2 + 3;
  d: boolean is 7 = f;
begin
  case b is
    when 1 => 4;
    when 2 => b;
    others => c;
  endcase;
end;
```

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test6b.txt

1  -- Test of Multiple Semantic Errors from Project 4 PDF
2  -- type mismatch err on line 9 as it's not integer
3
4  function test returns integer;
5    a: integer is 3;
6    f: integer is 7;
7    b: integer is
8      if a > 1 then
9        2.5;
10     else
11       5;
12     endif;
Semantic Error, Then and Else statements must be the same type
13    c: integer is 2 + 3;
14    d: boolean is 7 = f;
15  begin
16    case b is
17      when 1 => 4;
18      when 2 => b;
19      others => c;
20    endcase;
21  end;

Lexical Errors: 0
Syntax Errors: 0
Semantic Errors: 1
```

Test 6c:

This is the test case from the PDF but without any errors. It should compile.

```
-- The test case from PDF but without any errors

function test returns integer;
    a: integer is 3;
    f: integer is 7;
    b: integer is
        if a > 1 then
            2;
        else
            5;
        endif;
    c: integer is 2 + 3;
    d: boolean is 7 = f;
begin
    case b is
        when 1 => 4;
        when 2 => b;
        others => c;
    endcase;
end;
```

Compilation:

```
angel@Admin:~/Desktop/CMSC 430/Project 4 Skeleton Code$ ./compile < test6c.txt

1  -- The test case from PDF but without any errors
2
3  function test returns integer;
4      a: integer is 3;
5      f: integer is 7;
6      b: integer is
7          if a > 1 then
8              2;
9          else
10             5;
11         endif;
12     c: integer is 2 + 3;
13     d: boolean is 7 = f;
14 begin
15     case b is
16         when 1 => 4;
17         when 2 => b;
18         others => c;
19     endcase;
20 end;

Compiled Successfully
```

Lessons Learned and any improvements that could be made

A lot of my lesson learned is in the "how I approached the project" portion above. I would say that my biggest lesson here was how to adapt code, especially the new methods I created in Project 2 and Project 3. For example, the semantic parsing methods of the Project 4 skeleton were similar but different to Project 3's. Since I did Project 3 and particularly the creation of methods to check and analyze the "if else" and "case is" statements, Project 4 wasn't as difficult as I expected but there had to be a lot of attention to detail and adapting whatever I learned in Project 3 to Project 4 - but parsing semantics instead of syntax. I felt like the criteria for Project 4 were more specific, "it needs to do, not that" and I made more test cases to test every criteria. copy pasted parts of the criteria from the PDF to the test cases here.