Week 8: Project 4

John Kucera

Prof. Renee McDonald

CMSC 430 Compiler Theory and Design

04 May 2021

Week 8: Project 4 Documentation

Modifying the Semantic Analyzer (C++ with Flex and Bison)

Approaching the Project

Just like the previous projects, I started by making sure I knew exactly what was going on in the skeleton code. This time, there were two new files types.cc and types.h with some modifications to parser.y. I learned that parser.y was modified to check for semantic errors. Types.cc held type checking functions that were to be called in parser.y, and types.h held the function definitions to accompany types.cc. I then went to make modifications to the other files, such as scanner.l and listing.cc, to include the additional tokens and error checking I added from Projects 1-2. I also modified parser.y to form a more complete and efficient tree of grammar productions.

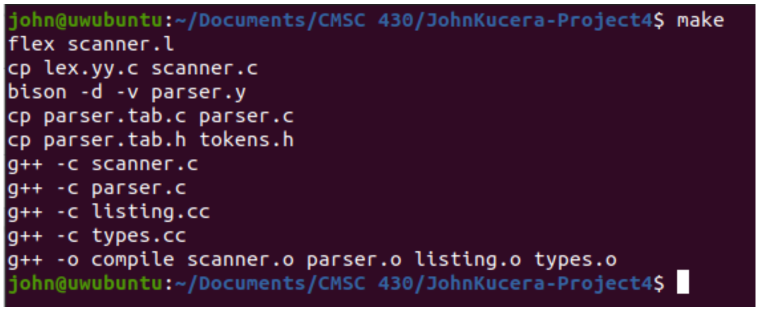
Getting into the new code, the first step I made was to modify symbols.h to include a function for checking duplicate variables. I call this in parser.y when a variable is declared, then it is checked in the symbol table to see if it already exists. If it does exist, the duplicate identifier error is raised and printed.

I then made modifications to the type-checking functions in types.cc to account for reals and booleans. First, I made sure to have errors raised when booleans were mixed with numeric types in either function returns or variable initialization. I also made sure that narrowing in BOTH function return and variable initialization raised errors, making it illegal to force a real value into an integer. Similarly, I made sure functions checkArithmetic, checkLogical, and checkRelational account for any type mismatches that occur with those operators. Numeric types are required for arithmetic operators, and boolean types are required for logical operators. I did the same with a checkRemainder function where I do not allow non-integers to be used with the rem operator.

Next, I made a function to store the return type for the input program. This is used later with the checkReturnType function to see if boolean and numeric types are getting mixed, OR a real is being returned when it should be an integer value (illegal narrowing). To account for if-then statements, I made a function that checks if an "if" expression is a boolean, and raises an error if it is not. Another function detects type mismatches between the "then" and "else" statements, raising an error if they do not match. For case statements, I made a function that checks if the case condition is an integer, and raises an error if it is not. Then, I made a function to store the first "when" statement that will be used to compare to the other "when" statements in another function. All of these are called in parser.y.

Test Cases

make: Successful.

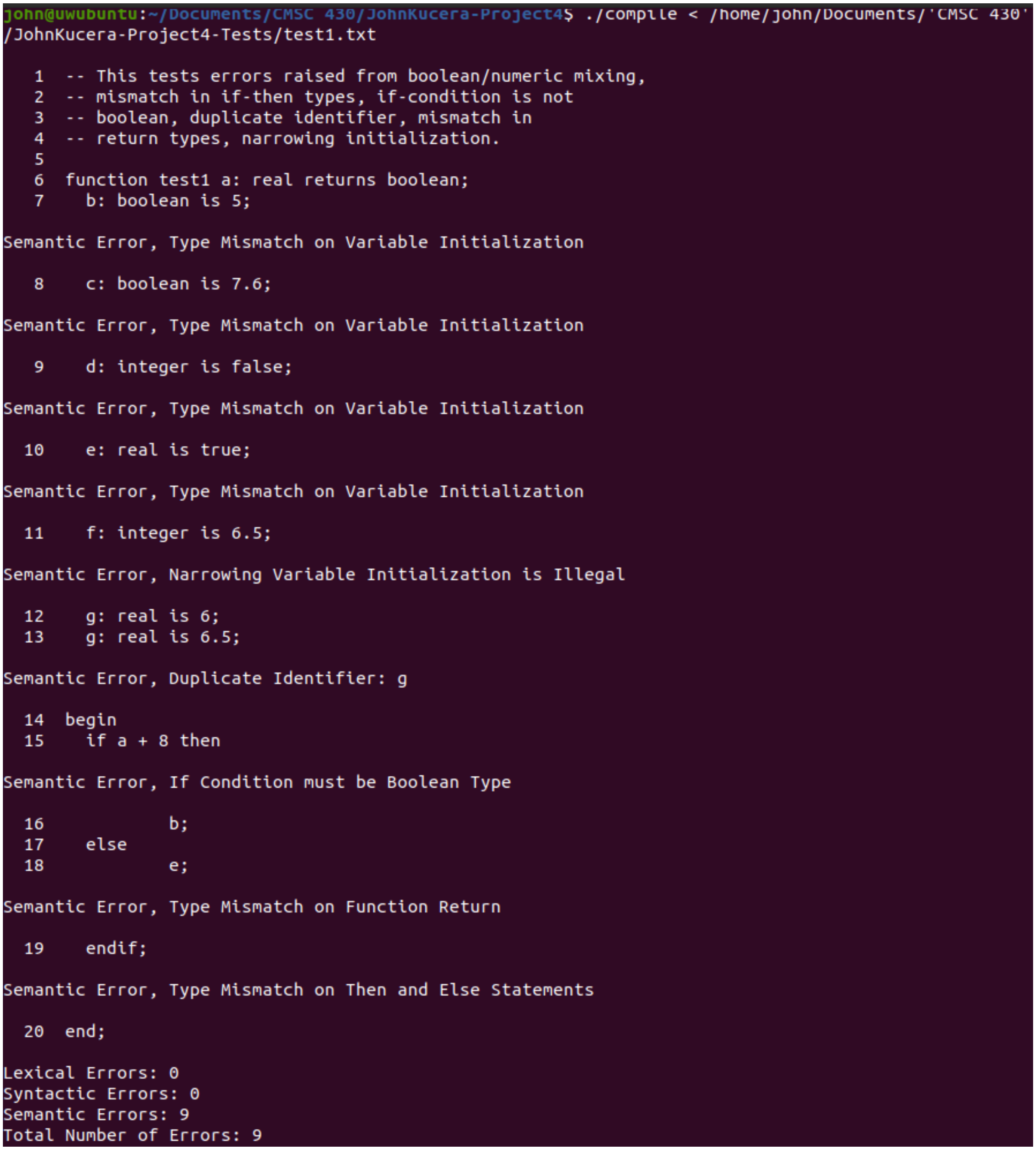


```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$ make
flex scanner.l
cp lex.yy.c scanner.c
bison -d -v parser.y
cp parser.tab.c parser.c
cp parser.tab.h tokens.h
g++ -c scanner.c
g++ -c parser.c
g++ -c listing.cc
g++ -c types.cc
g++ -o compile scanner.o parser.o listing.o types.o
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$
```

Week 8: Project 4

Test Case 1: test1.txt

Screenshot:



```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$ ./compile < /home/john/Documents/'CMSC 430'
/JohnKucera-Project4-Tests/test1.txt

   1  -- This tests errors raised from boolean/numeric mixing,
   2  -- mismatch in if-then types, if-condition is not
   3  -- boolean, duplicate identifier, mismatch in
   4  -- return types, narrowing initialization.
   5
   6  function test1 a: real returns boolean;
   7    b: boolean is 5;

Semantic Error, Type Mismatch on Variable Initialization

   8    c: boolean is 7.6;

Semantic Error, Type Mismatch on Variable Initialization

   9    d: integer is false;

Semantic Error, Type Mismatch on Variable Initialization

  10    e: real is true;

Semantic Error, Type Mismatch on Variable Initialization

  11    f: integer is 6.5;

Semantic Error, Narrowing Variable Initialization is Illegal

  12    g: real is 6;
  13    g: real is 6.5;

Semantic Error, Duplicate Identifier: g

  14  begin
  15    if a + 8 then

Semantic Error, If Condition must be Boolean Type

  16            b;
  17    else
  18            e;

Semantic Error, Type Mismatch on Function Return

  19    endif;

Semantic Error, Type Mismatch on Then and Else Statements

  20  end;

Lexical Errors: 0
Syntactic Errors: 0
Semantic Errors: 9
Total Number of Errors: 9
```

Week 8: Project 4

Aspect Tested	**Input (Line #)**	**Expected Output**
Mixing types: Initializing Boolean with an Integer value	Line 7: **b: boolean is 5;**	Error message: “Semantic Error, Type Mismatch on Variable Initialization”
Mixing types: Initializing Boolean with a Real value	Line 8: **c: boolean is 7.6;**	Error message: “Semantic Error, Type Mismatch on Variable Initialization”
Mixing types: Initializing Integer with a Boolean value	Line 9: **d: integer is false;**	Error message: “Semantic Error, Type Mismatch on Variable Initialization”
Mixing types: Initializing Real with a Boolean value	Line 10: **e: real is true;**	Error message: “Semantic Error, Type Mismatch on Variable Initialization”
Narrowing Variable Initialization: Assigning real value to an integer	Line 11: **f: integer is 6.5;**	Error message: “Semantic Error, Narrowing Variable Initialization is Illegal”
Permitting Widening: Assigning integer value to a real	Line 12: **g: real is 6;**	No error messages. Widening is permitted.

Duplicate Identifier	Line 12 & 13:
	g: real is 6;
	g: real is 6.5;
		Error message: “Semantic Error, Duplicate Identifier: g”

If Condition is not Boolean	Line 15: **if a + 8 then**	Error message: “Semantic Error, If Condition must be Boolean Type”

Type mismatch on Function return in then/else statement	Line 17 & 18:
	else
	e;
	(e is real, function is supposed to return boolean)
		Error message: “Semantic Error, Type mismatch on Function return”

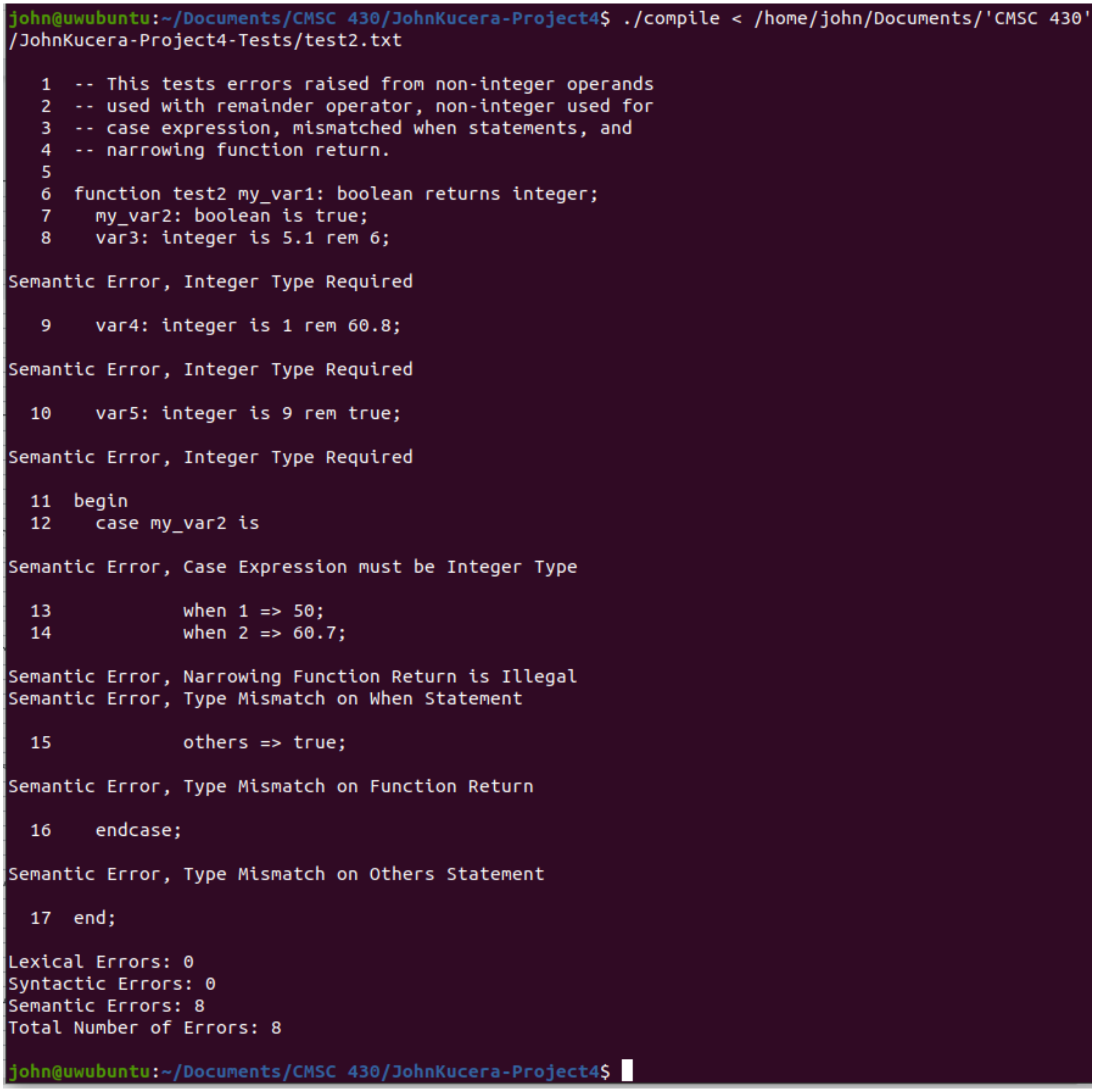
Type mismatch on Then and Else statements	Lines 15-20:
	if a + 8 then
	b;
	else
	e;
	endif
	(b is Boolean, e is real)
		Error message: “Semantic Error, Type mismatch on Then and Else Statements”

Multiple errors in one file	9 semantic errors, totaling to 9 errors	“Semantic Errors: 9 Total Number of Errors: 9”
TEST PASSED? YES. Successfully raised the errors properly.		

Week 8: Project 4

Test Case 2: test2.txt

Screenshot:



```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$ ./compile < /home/john/Documents/'CMSC 430'
/JohnKucera-Project4-Tests/test2.txt

   1  -- This tests errors raised from non-integer operands
   2  -- used with remainder operator, non-integer used for
   3  -- case expression, mismatched when statements, and
   4  -- narrowing function return.
   5
   6  function test2 my_var1: boolean returns integer;
   7    my_var2: boolean is true;
   8    var3: integer is 5.1 rem 6;

Semantic Error, Integer Type Required

   9    var4: integer is 1 rem 60.8;

Semantic Error, Integer Type Required

  10    var5: integer is 9 rem true;

Semantic Error, Integer Type Required

  11  begin
  12    case my_var2 is

Semantic Error, Case Expression must be Integer Type

  13            when 1 => 50;
  14            when 2 => 60.7;

Semantic Error, Narrowing Function Return is Illegal
Semantic Error, Type Mismatch on When Statement

  15            others => true;

Semantic Error, Type Mismatch on Function Return

  16    endcase;

Semantic Error, Type Mismatch on Others Statement

  17  end;

Lexical Errors: 0
Syntactic Errors: 0
Semantic Errors: 8
Total Number of Errors: 8

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$ 
```

Week 8: Project 4

Aspect Tested	Input (Line #)	Expected Output
Non-integers operands with Remainder: real	Line 8: **var3: integer is 5.1 rem 6;**	Error message: “Semantic Error, Integer Type Required”
Non-integers operands with Remainder: real	Line 9: **var4: integer is 1 rem 60.8;**	Error message: “Semantic Error, Integer Type Required”
Non-integers operands with Remainder: boolean	Line 10: **var5: integer is 9 rem true;**	Error message: “Semantic Error, Integer Type Required”
Case Condition is not Integer	Line 12: **case my_var2 is** (my_var2 is Boolean)_	Error message: “Semantic Error, Case Expression must be Integer Type”
Narrowing Function Return: Returning real value when integer is expected	Line 14: **when 2 => 60.7;** (integer is expected to be returned for this function)	Error message: “Semantic Error, Narrowing Function Return is Illegal”

Case When Statements have type mismatch	Line 13 & 14:
	when 1 => 50;
	when 2 => 60.7;
		Error message: “Semantic Error, Type Mismatch on When Statement”

Type mismatch on Function return in case statement	Line 15: **others => true;** (integer is expected to be returned for this function)	Error message: “Semantic Error, Type Mismatch on Function Return”

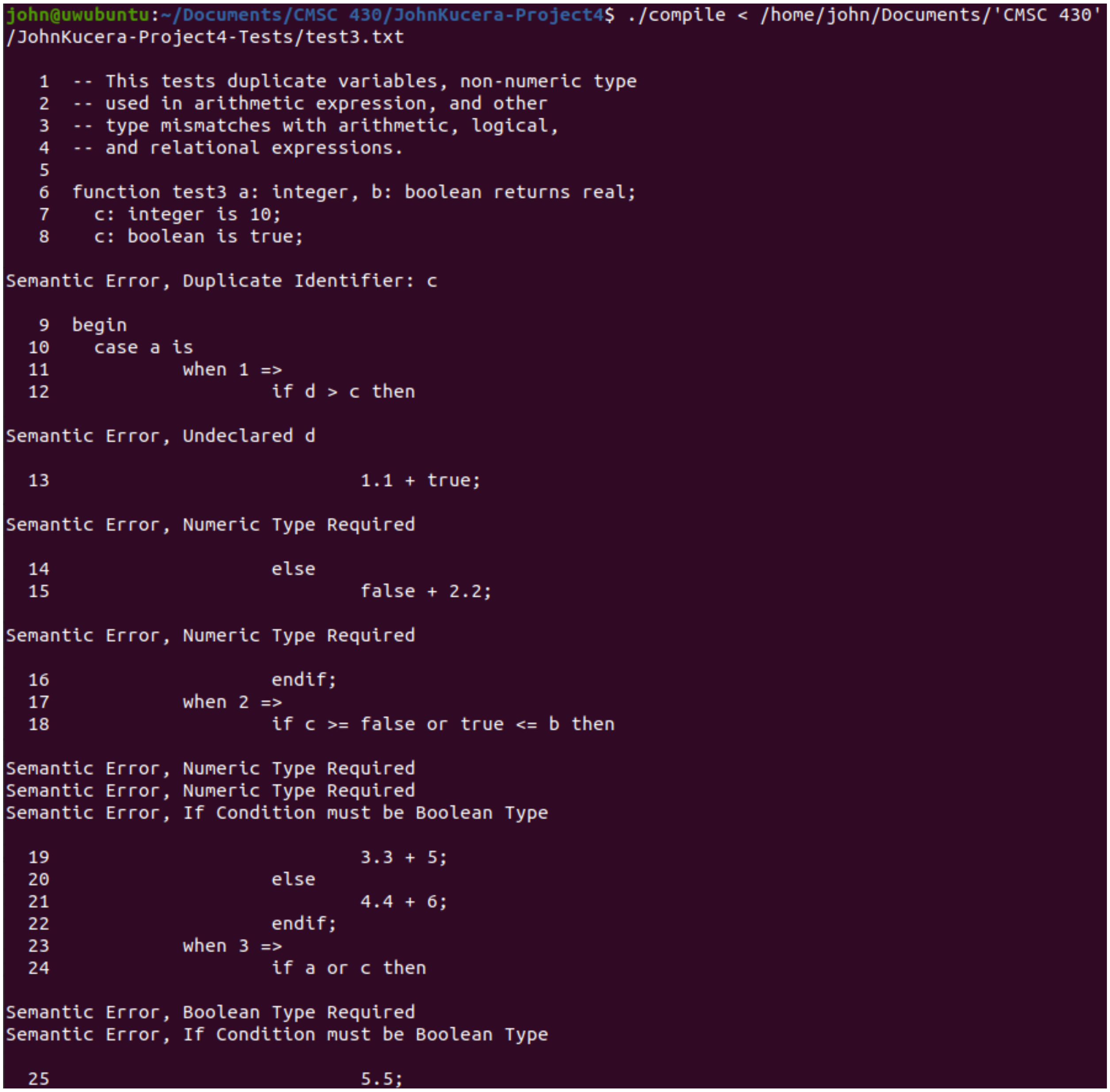
Case Others Statement have type mismatch	Line 13 - 15:
	when 1 => 50;
	when 2 => 60.7;
	others => true;
		Error message: “Semantic Error, Type Mismatch on Others Statement”

Multiple errors in one file	8 semantic errors, totaling to 8 errors	“Semantic Errors: 8 Total Number of Errors: 8”
TEST PASSED? YES. Successfully raised the errors properly.		

Week 8: Project 4

Test Case 3: test3.txt

Screenshot:

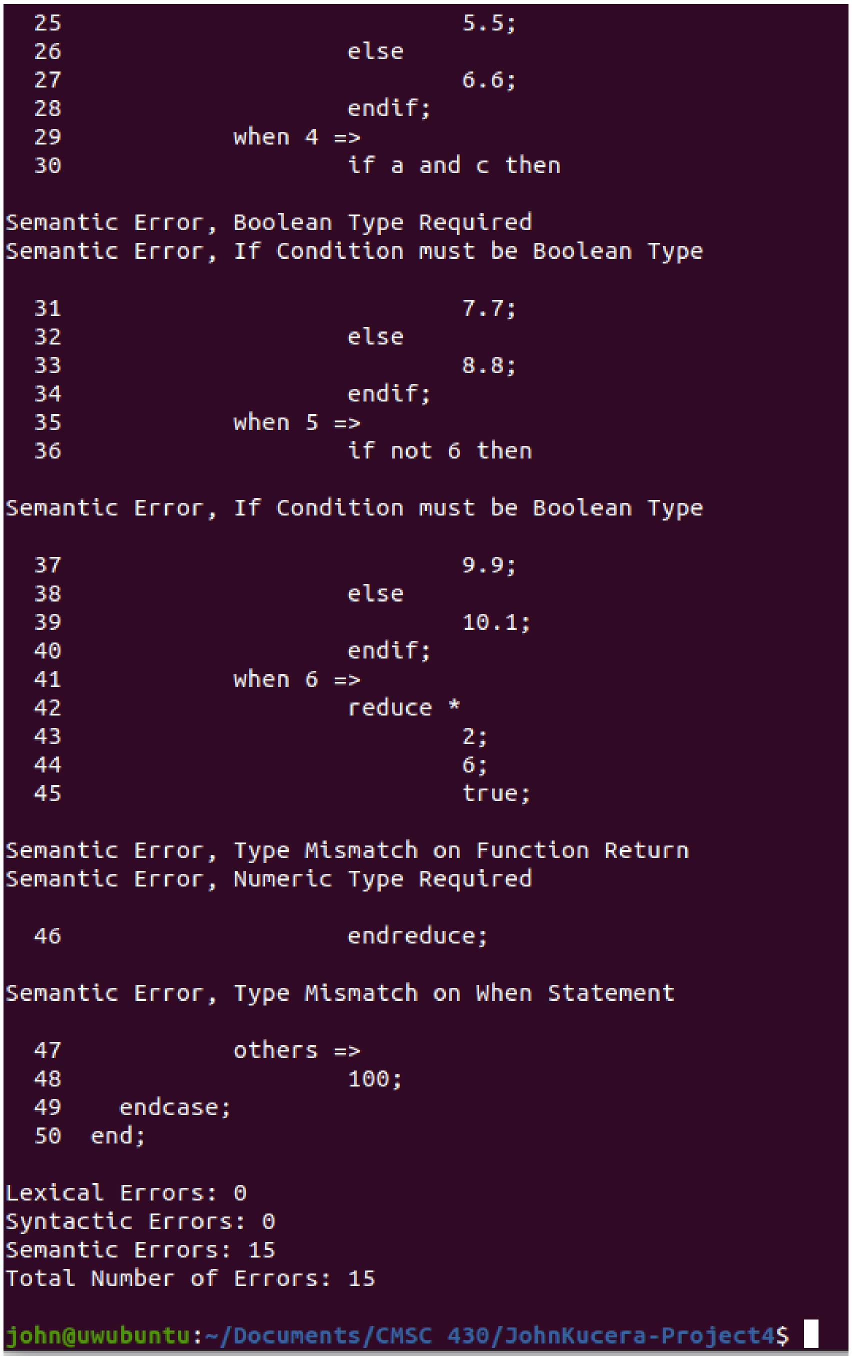


```
john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$ ./compile < /home/john/Documents/'CMSC 430'
/JohnKucera-Project4-Tests/test3.txt

   1  -- This tests duplicate variables, non-numeric type
   2  -- used in arithmetic expression, and other
   3  -- type mismatches with arithmetic, logical,
   4  -- and relational expressions.
   5
   6  function test3 a: integer, b: boolean returns real;
   7    c: integer is 10;
   8    c: boolean is true;

Semantic Error, Duplicate Identifier: c

   9  begin
  10    case a is
  11            when 1 =>
  12                    if d > c then

Semantic Error, Undeclared d

  13                            1.1 + true;

Semantic Error, Numeric Type Required

  14                    else
  15                            false + 2.2;

Semantic Error, Numeric Type Required

  16                    endif;
  17            when 2 =>
  18                    if c >= false or true <= b then

Semantic Error, Numeric Type Required
Semantic Error, Numeric Type Required
Semantic Error, If Condition must be Boolean Type

  19                            3.3 + 5;
  20                    else
  21                            4.4 + 6;
  22                    endif;
  23            when 3 =>
  24                    if a or c then

Semantic Error, Boolean Type Required
Semantic Error, If Condition must be Boolean Type

  25                            5.5;
```

(continues on next page)

Week 8: Project 4

(continued from previous screenshot)



```
  25                            5.5;
  26                    else
  27                            6.6;
  28                    endif;
  29            when 4 =>
  30                    if a and c then

Semantic Error, Boolean Type Required
Semantic Error, If Condition must be Boolean Type

  31                            7.7;
  32                    else
  33                            8.8;
  34                    endif;
  35            when 5 =>
  36                    if not 6 then

Semantic Error, If Condition must be Boolean Type

  37                            9.9;
  38                    else
  39                            10.1;
  40                    endif;
  41            when 6 =>
  42                    reduce *
  43                            2;
  44                            6;
  45                            true;

Semantic Error, Type Mismatch on Function Return
Semantic Error, Numeric Type Required

  46                    endreduce;

Semantic Error, Type Mismatch on When Statement

  47            others =>
  48                    100;
  49    endcase;
  50  end;

Lexical Errors: 0
Syntactic Errors: 0
Semantic Errors: 15
Total Number of Errors: 15

john@uwubuntu:~/Documents/CMSC 430/JohnKucera-Project4$ 
```

Week 8: Project 4

Aspect Tested	**Input (Line #)**	**Expected Output**
Undeclared variable	Line 12: **if d > c then** (d has no declaration beforehand)	Error message: “Semantic Error, Undeclared d”
Arithmetic operator: Boolean operand	Line 13: **1.1 + true;**	Error message: “Semantic Error, Numeric Type Required”
Arithmetic operator: Boolean operand	Line 15: **false + 2.2;**	Error message: “Semantic Error, Numeric Type Required”
Relational operator: Boolean operand	Line 18: **if c >= false or true <= b then**	Error message x2: Error message: “Semantic Error, Numeric Type Required”
Logical operator OR: Numeric operand	Line 24: **if a or c then** (a and c are integers)	Error message: “Semantic Error, Boolean Type required”
Logical operator AND: Numeric operand	Line 30: **if a and c then** (a and c are integers)	Error message: “Semantic Error, Boolean Type required”

Reduction: Non-numeric operand	Line 42-46:
	reduce *
	2;
	6;
	true;
	endreduce;
		Error message: “Semantic Error, Numeric Type Required”

Multiple errors in one file	15 semantic errors, totaling to 15 errors	“Semantic Errors: 15 Total Number of Errors: 15”
TEST PASSED? YES. Successfully raised the errors properly.		

Week 8: Project 4

Lessons Learned

One lesson I learned in the process of writing this project was to not brush aside old code just because I think it is complete and done with. In this case, those files specifically were listing.cc and listing.h. Since the beginning, Listing.cc already handled a few of the semantic errors, such as undeclared identifier and duplicate identifier. When I was looking through types.cc and parser.y to figure out how undeclared identifiers were being checked, I did not find much and was so confused on where it was going to come from. I only checked listing.cc later because it was code that I finished writing a long time ago and forgot about it. A lot of time would be saved if I reviewed it and made sure I knew everything that was in it. So, after realizing that undeclared and duplicate identifiers had their own error types, writing my own checks for duplicate identifiers became easier.

Another lesson I learned, similar to last project, was to pay more attention to incorporating the C++ code with the code used for flex and bison. There were many times when I spent too much time wondering "How am I going to write this function in the bison/flex language? I still don't know much about it." It took me a while to realize that I can just write the functions in C++ in another file, then call them in the bison/flex files. In that sense, even though we are using multiple languages, it allows code to be written simpler and with more efficiency since C++ is what I (and many others) know better than these linux tools.

Overall, learning about all the parts of a compiler this semester (lexical analyzer, syntactic analyzer, interpreter, semantic analyzer) definitely benefited my confidence as a computer science student. I can now discuss and analyze compilation in more depth and have a clearer view on the differences between lexical errors, syntax errors, and semantic errors. The whole process of writing a compiler was very interesting to me and I feel a sense accomplishment from learning how to write one.