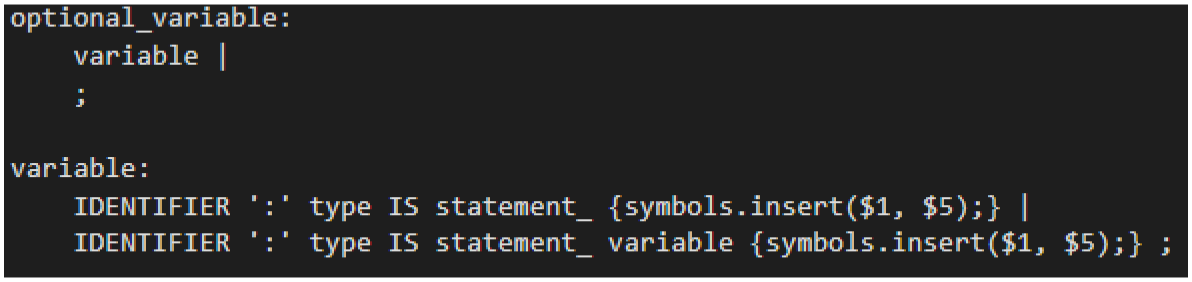
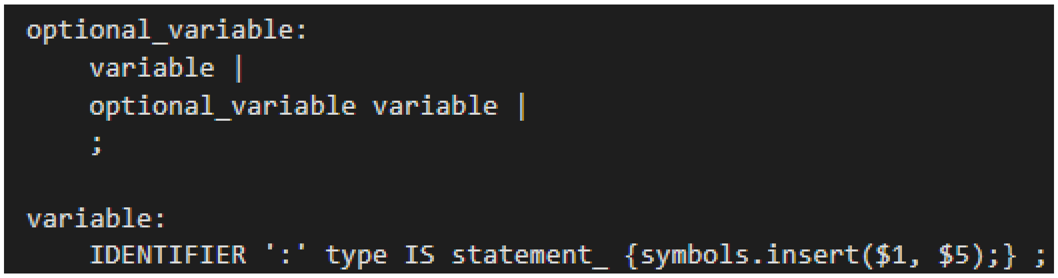
I started this project by messing around for quite a while to find out how the semantics/interpreter worked. Then after having a vague idea of what they did, I slowly started transplanting my code from project 2. By looking at the code, I had a rough idea of what each symbol meant so some semantics were easier to implement than others (copying && and changing it to || in expression). After much testing, I realized all of my 0 or more repetitions were not working as I had thought. I first noticed this with variable/optional_variable which started out as this:



```
optional_variable:
    variable |
    ;

variable:
    IDENTIFIER ':' type IS statement_ {symbols.insert($1, $5);} |
    IDENTIFIER ':' type IS statement_ variable {symbols.insert($1, $5);} ;
```

And turned into this:



```
optional_variable:
    variable |
    optional_variable variable |
    ;

variable:
    IDENTIFIER ':' type IS statement_ {symbols.insert($1, $5);} ;
```

The semantics weren't able to properly recursively. Though as I am writing this, I realize that if I the second option in variable in the first image was changed to this : "IDENTIFIER ':' type IS statement_ variable;" (removed symbols.insert) it probably would have worked as intended.

Next, I had quite a bit of trouble figuring out how to get the parameters to work. Given my minimal experience with C++, I'd never worked with arguments/the command line. Therefore, I had some trouble figure out what argc and argv did. After a quick google, I was able to figure out what they do. It took a few tries to get the parameters working properly, but what I eventually settled on was making a double array of argc size and copied everything from argv. Then, once parameters start. they skip index 0 which has the ./compile part and go for the first argument passed after it (index 0)

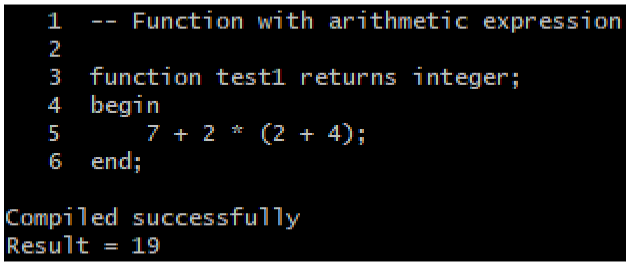
I was struggling with evaluating/interpreting booleans for a while when I realized I never added bool to scanner.l. Once I added it with the options for true and false, I was still having problems. I messed around a bit in scanner.l and moved {bool} above {real} and everything worked as intended.

Values.cc and values.h were simple enough to add other options to. It simply involved adding more case statements in evaulateReduction, Relational, and Arithmetic. There was some trouble adding subtract and divide to evaluateReduction. During trouble shooting I found out that at the start head is equal to 0 for case ADD, but every other case the head starts at 1. To fix this, I made it start at 0 for ADD and SUBTRACT. DIVIDE was a bit trickier, I had to add a boolean passed as a parameter to skip dividing the head by the tail for the first reduce (dividing 1/tail skew the results). Once the reduce statement has ended the boolean resets for later use. It took some time to figure out why and where evaluateBoolan and evaluateReal were necessary, but I eventually figured it out.

Lastly, I tried for a few hours to get case statements working but nothing I came up with worked. I procrastinated on them and implement everything else but eventually all I had left to implement was them. So after trying for a little while more and then giving up and googling for a bit I came across a solution from Sean Filer on github. It involve using isnan() to determine if values are valid numbers and if they are not valid they are NAN. Finally, the key component I couldn't figure out was having an interpretive statement/semantics within the recursive optional_case.

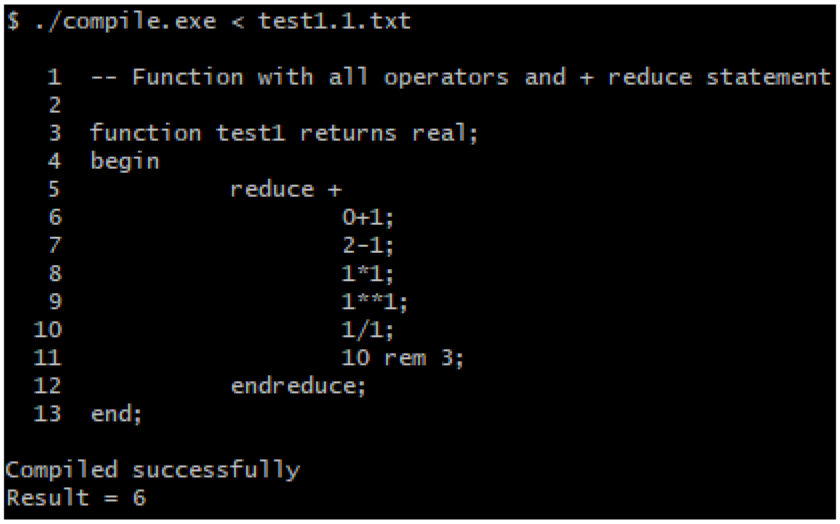
Overall, I was quite proud of all that I was able to figure out on my own. I was a little sad I couldn't figure out the answer to case statements on my own but it was a great feeling when it clicked and I figured out how exactly the solution worked and I knew why it worked. Some improvements that I'm predicting will be implemented in Project 4 are more checks preformed for valid types that match statements in variable/parameter. For example, currently if you declare c: boolean is 3 and then return c it will return 3. However, 3 shouldn't be a valid choice for boolan. It should only be true/false (maybe 0 or 1). Also, in the future I see when the first line says return boolean that maybe it will be able to return true/false rather than simply 0/1. And if it says return integer, it shouldn't be returning a boolean. Also, smarter checks for remainder might be necessary. For example, prevent (4 rem 10) since there is not remainder and it wouldn't be valid (4 / 10 would still be valid though). Finally, better chucks for parameters such as: recognizing more arguments passed than there are parameter (4 arguments passed with only 3 parameters), less arguments passed than parameters

(3 arguments passed with only 4 parameters), and invalid type passing (real variable gets value of true passed). Test cases of all aspects of the compiler start below:



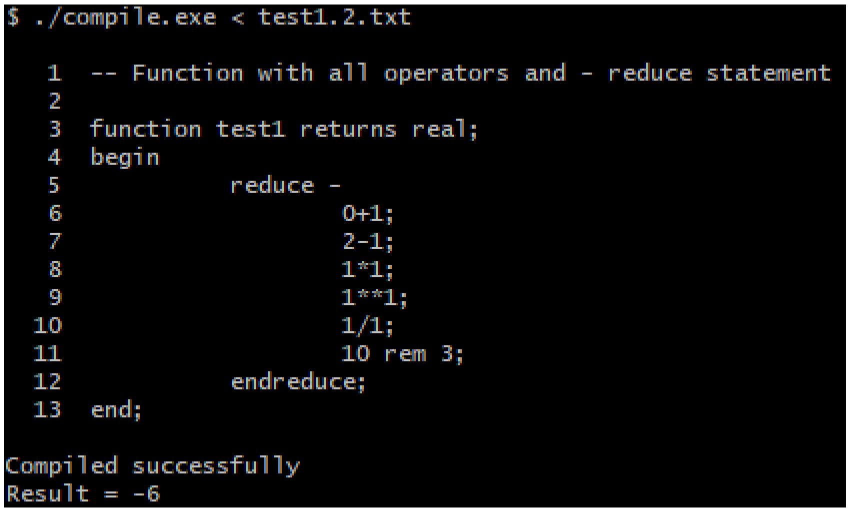
```
-- Function with arithmetic expression

function test1 returns integer;
begin
    7 + 2 * (2 + 4);
end;

Compiled successfully
Result = 19
```

Figure 1. Test case #1. Function with arithmetic expression.



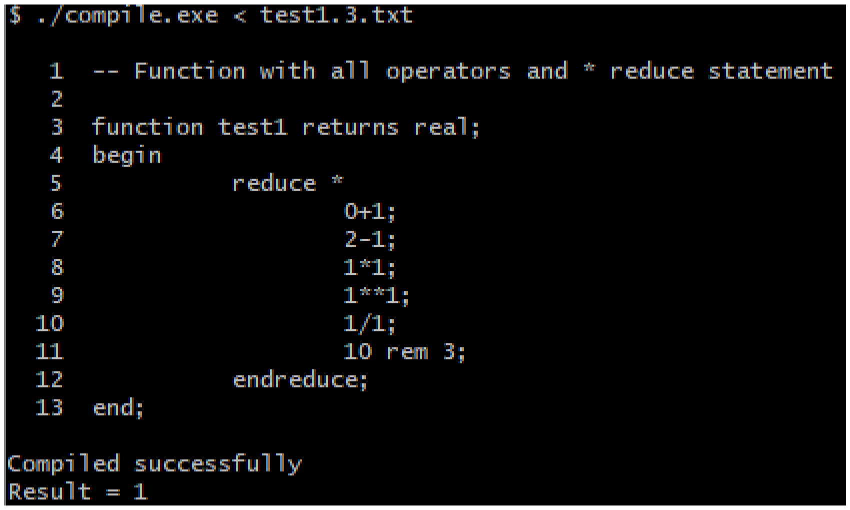
```
$ ./compile.exe < test1.1.txt

-- Function with all operators and + reduce statement

function test1 returns real;
begin
          reduce +
                  0+1;
                  2-1;
                  1*1;
                  1**1;
                  1/1;
                  10 rem 3;
          endreduce;
end;

Compiled successfully
Result = 6
```

Figure 2. Test case #2. All operators and reduce statement with ADD operator.



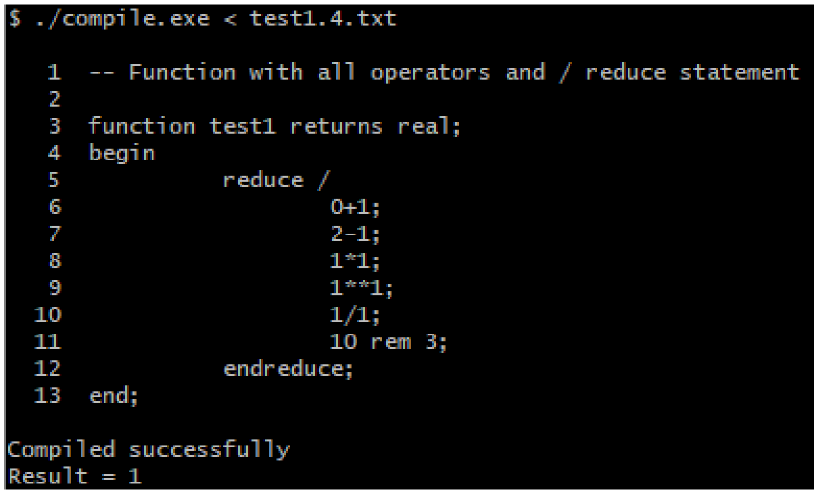
```
$ ./compile.exe < test1.2.txt

-- Function with all operators and - reduce statement

function test1 returns real;
begin
          reduce -
                  0+1;
                  2-1;
                  1*1;
                  1**1;
                  1/1;
                  10 rem 3;
          endreduce;
end;

Compiled successfully
Result = -6
```

Figure 3. Test case #3. All operators and reduce statement with SUBTRACT operator.



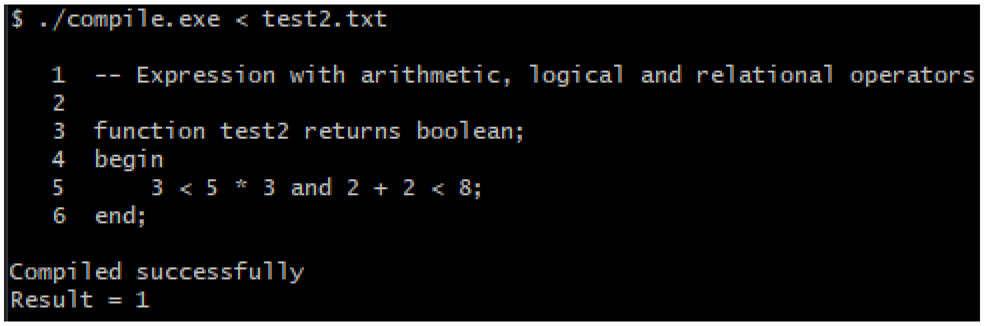
```
$ ./compile.exe < test1.3.txt

-- Function with all operators and * reduce statement

function test1 returns real;
begin
        reduce *
                0+1;
                2-1;
                1*1;
                1**1;
                1/1;
                10 rem 3;
        endreduce;
end;

Compiled successfully
Result = 1
```

Figure 4. Test case #4. All operators and reduce statement with MULTIPLY operator.



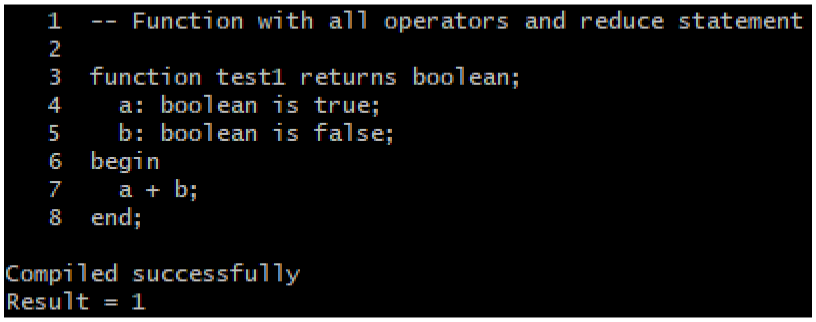
```
$ ./compile.exe < test1.4.txt

-- Function with all operators and / reduce statement

function test1 returns real;
begin
        reduce /
                0+1;
                2-1;
                1*1;
                1**1;
                1/1;
                10 rem 3;
        endreduce;
end;

Compiled successfully
Result = 1
```

Figure 5. Test case #5. All operators and reduce statement with DIVIDE operator.



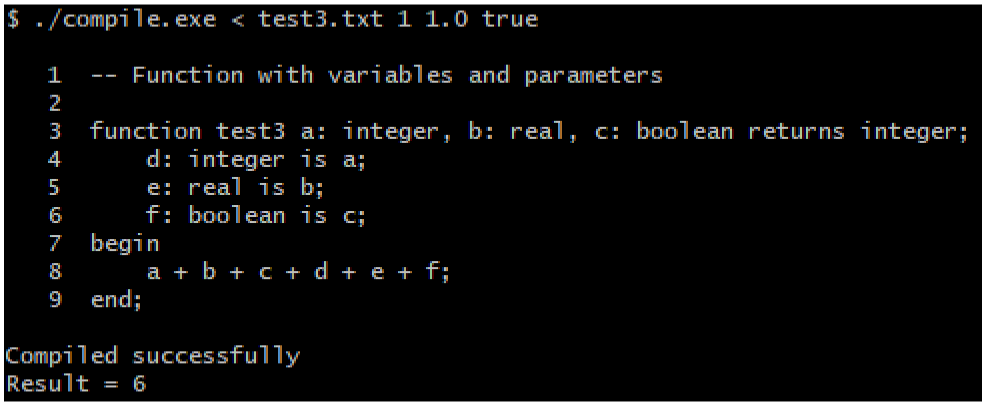
```
$ ./compile.exe < test2.txt

-- Expression with arithmetic, logical and relational operators

function test2 returns boolean;
begin
    3 < 5 * 3 and 2 + 2 < 8;
end;

Compiled successfully
Result = 1
```

Figure 6. Test case #6. Logical and relational operators.



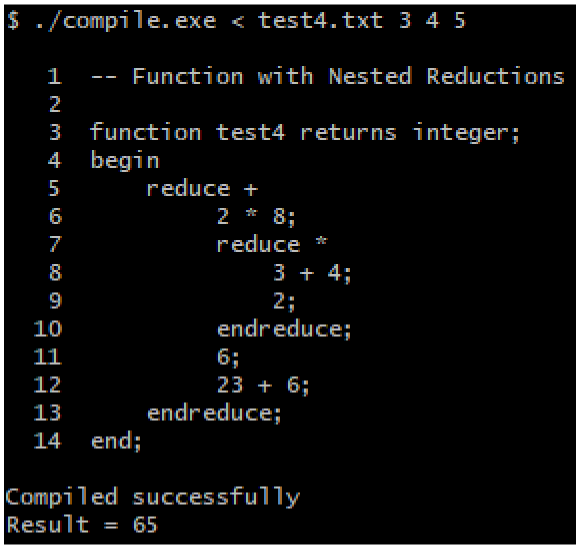
```
-- Function with all operators and reduce statement

function test1 returns boolean;
  a: boolean is true;
  b: boolean is false;
begin
  a + b;
end;

Compiled successfully
Result = 1
```

Figure 6.5. Test case #6.5. Showing the values that Booleans hold.



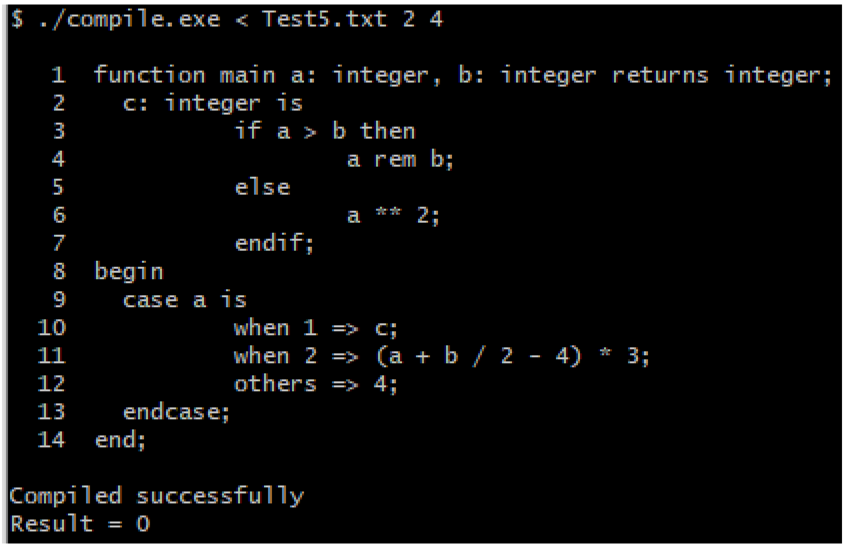
```
$ ./compile.exe < test3.txt 1 1.0 true

-- Function with variables and parameters

function test3 a: integer, b: real, c: boolean returns integer;
    d: integer is a;
    e: real is b;
    f: boolean is c;
begin
    a + b + c + d + e + f;
end;

Compiled successfully
Result = 6
```

Figure 7. Test case #7. Multiple parameters and variables.



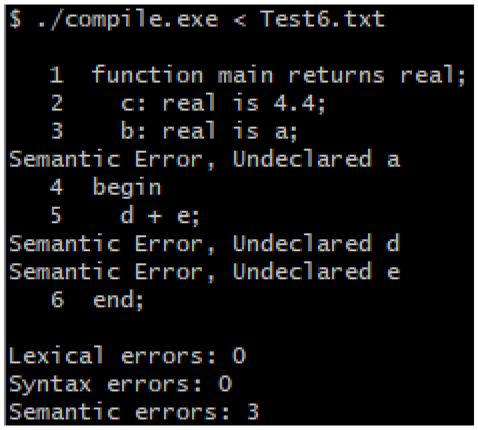
```
$ ./compile.exe < test4.txt 3 4 5

-- Function with Nested Reductions

function test4 returns integer;
begin
    reduce +
         2 * 8;
         reduce *
             3 + 4;
             2;
         endreduce;
         6;
         23 + 6;
    endreduce;
end;

Compiled successfully
Result = 65
```

Figure 8. Test case #8. Nested reduction statement.



```
$ ./compile.exe < Test5.txt 2 4

function main a: integer, b: integer returns integer;
  c: integer is
          if a > b then
                  a rem b;
          else
                  a ** 2;
          endif;
begin
  case a is
          when 1 => c;
          when 2 => (a + b / 2 - 4) * 3;
          others => 4;
  endcase;
end;

Compiled successfully
Result = 0
```

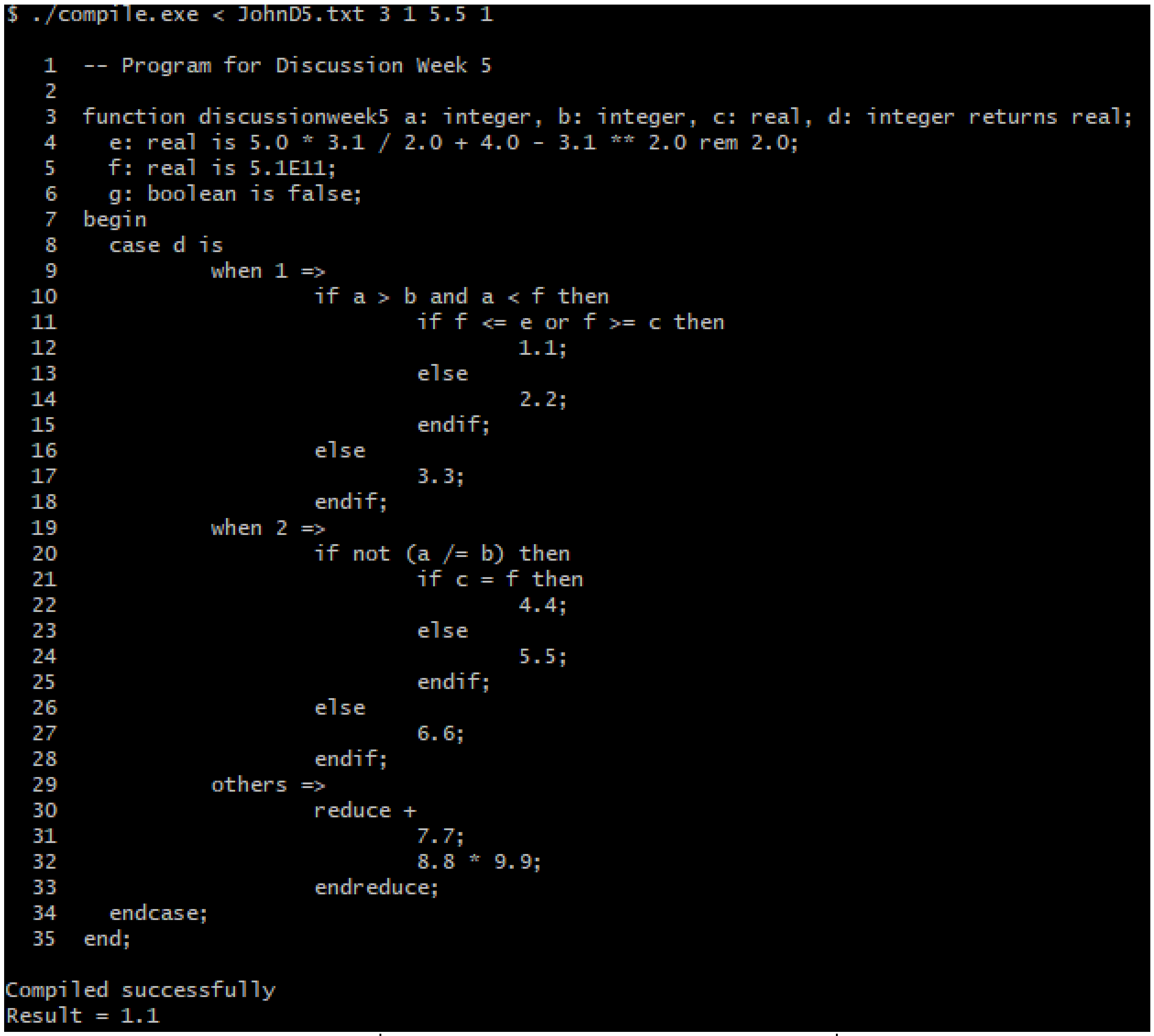
Figure 9. Test case #9. Testing the program given in the Project 3 handout.



```
$ ./compile.exe < Test6.txt

   1  function main returns real;
   2    c: real is 4.4;
   3    b: real is a;
Semantic Error, Undeclared a
   4  begin
   5    d + e;
Semantic Error, Undeclared d
Semantic Error, Undeclared e
   6  end;

Lexical errors: 0
Syntax errors: 0
Semantic errors: 3
```

Figure 10. Test case #10. Testing undeclared variables, including multiple on one line.

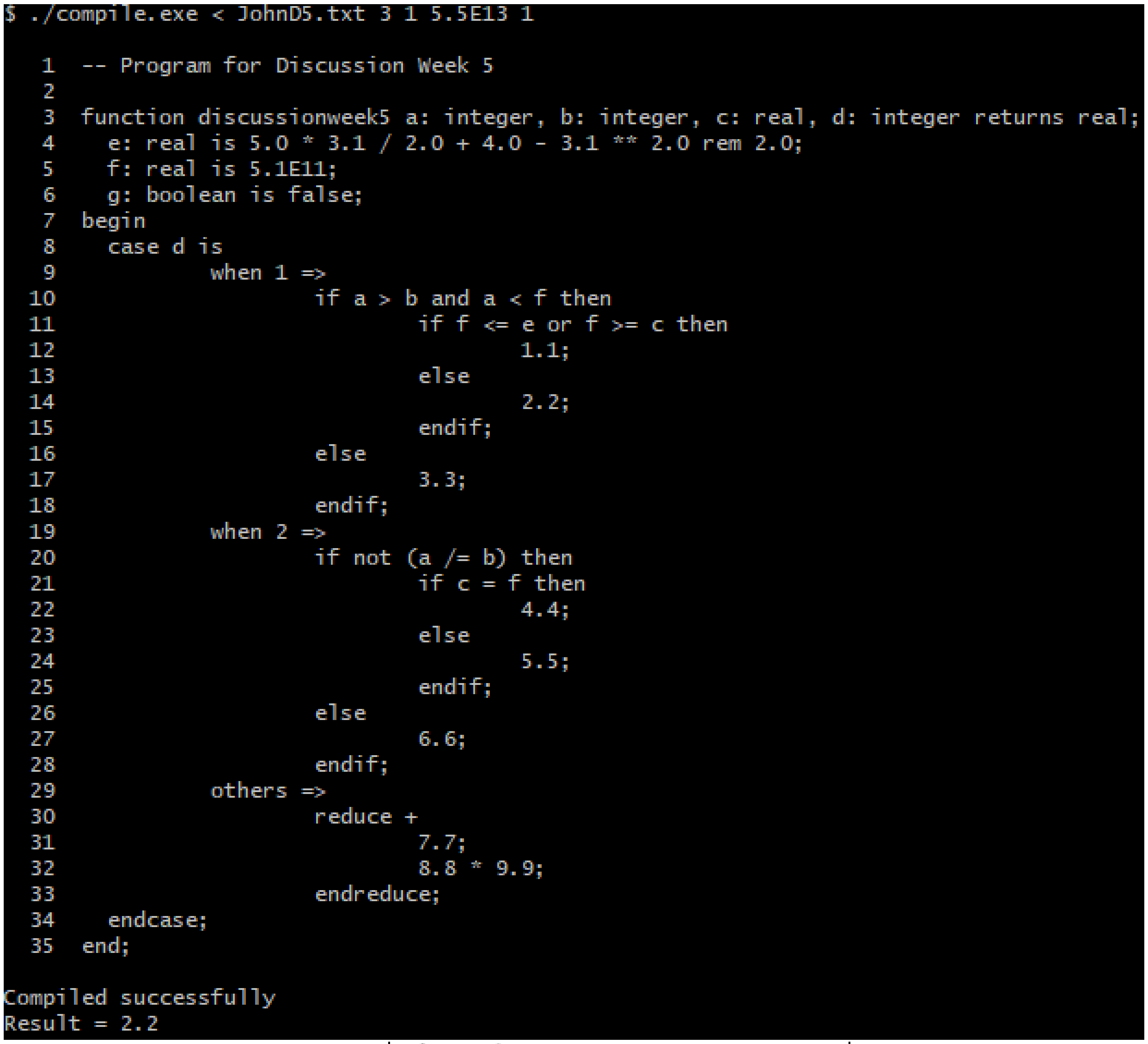
I really liked John Kucera's program so the rest of the test cases will involve those he came up with for discussion 5. I found his program was very easy to follow because everything returned is a set number. His program is also somewhat complicated and involves multiple nested statements, parameter and variable declarations, all operators)arithmetic, relational, logical) and all statements (if, case reduce). The test cases along with the expected/actual results start on the next page:



```
$ ./compile.exe < JohnD5.txt 3 1 5.5 1

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 1.1
```

a: 3, b: 1, c: 5.5, d: 1	when 1 =>
	if a > b and a < f then
	if f <= e or f >= c then
		1.1

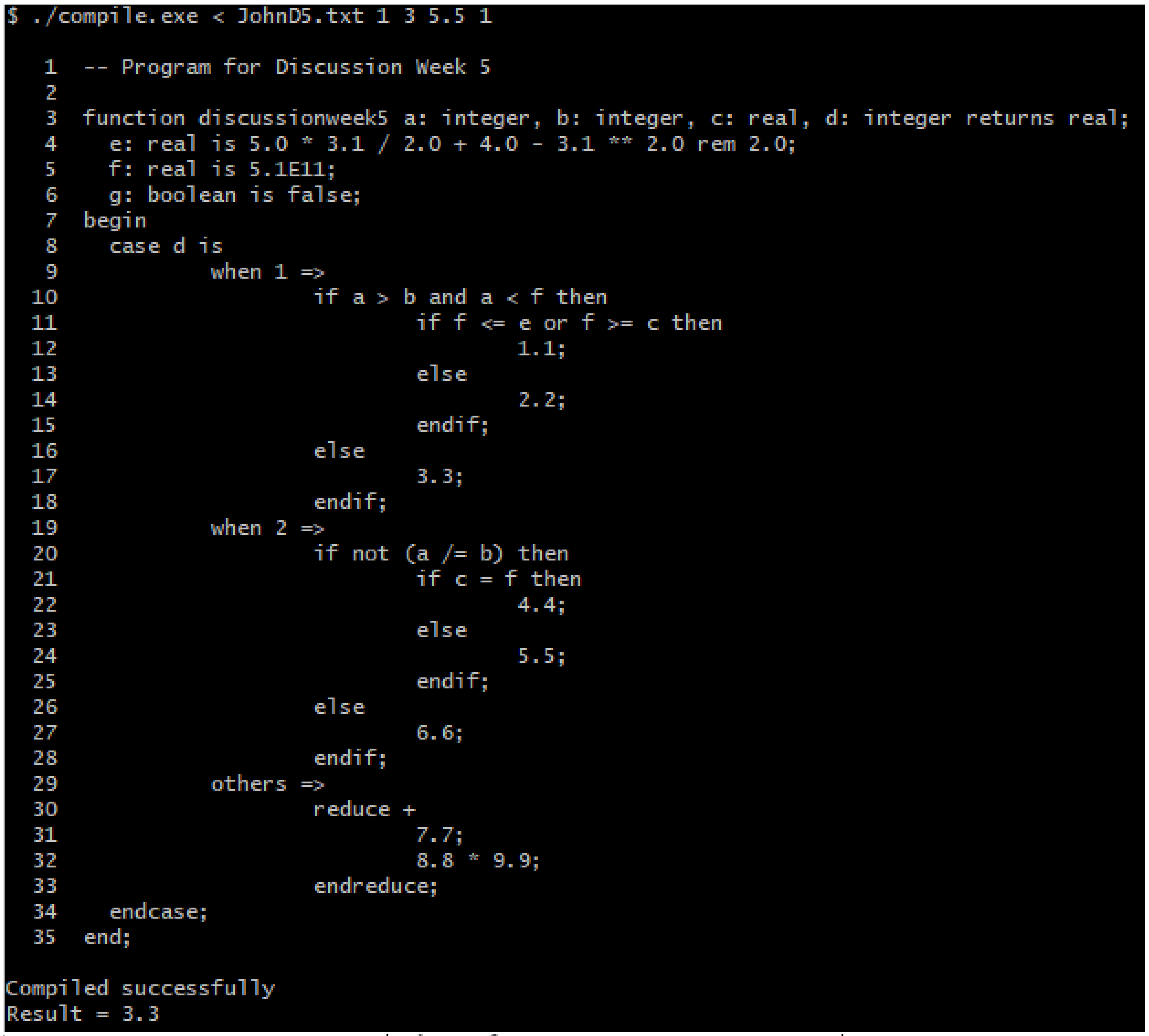
Figure 12. Test case #1 of John Kucera’s program. Results are as expected.



```
$ ./compile.exe < JohnD5.txt 3 1 5.5E13 1

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 2.2
```

a: 3, b: 1, c: 5.5E13, d: 1	when 1 =>
	if a > b and a < f then
	if f <= e or f >= c then (else)
		2.2

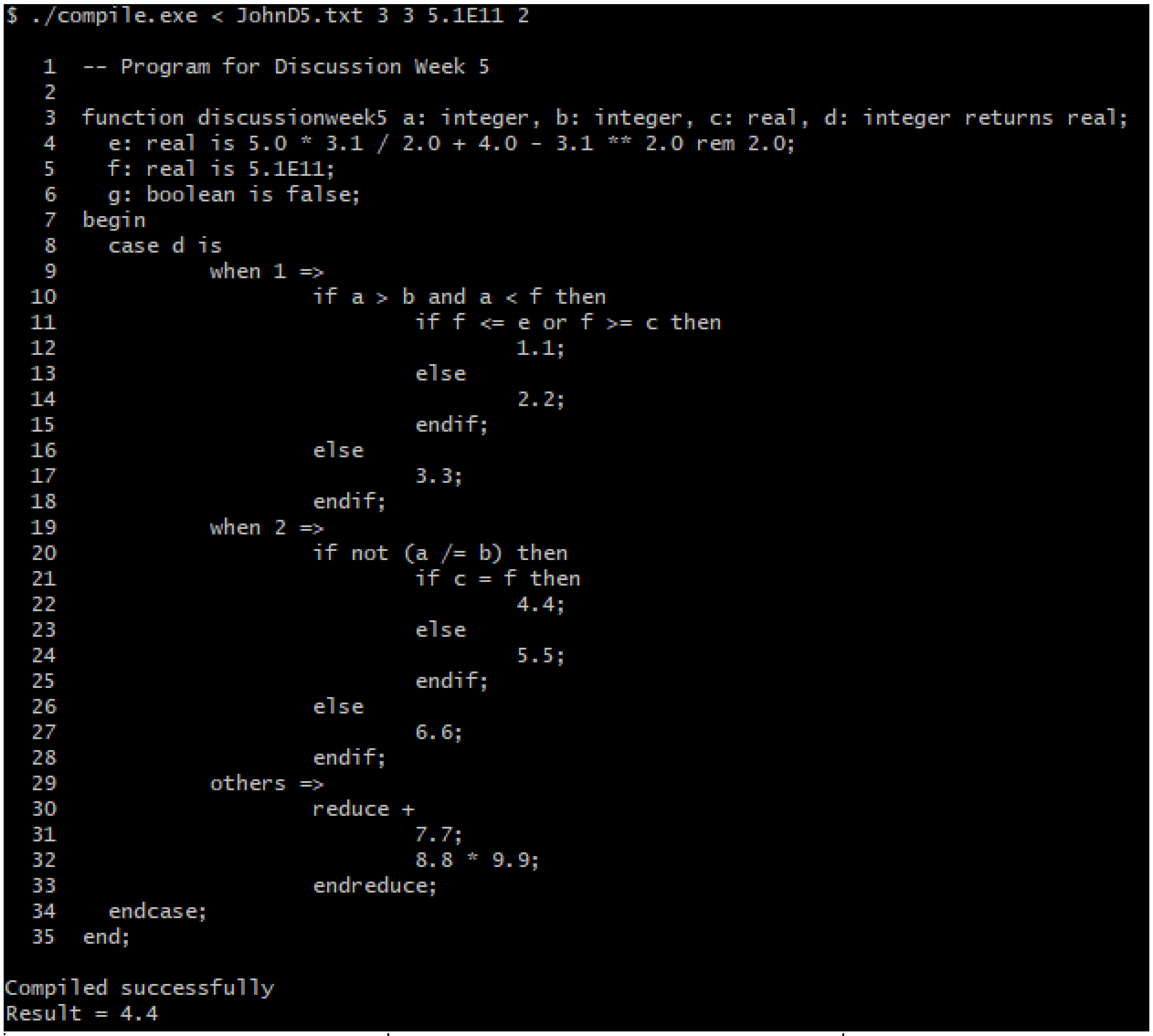
Figure 13. Test case #2 of John Kucera's program. Results are as expected.



```
$ ./compile.exe < JohnD5.txt 1 3 5.5 1

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 3.3
```

a: 1, b: 3, c: 5.5, d: 1	when 1 => if a > b and a < f then (else)	3.3

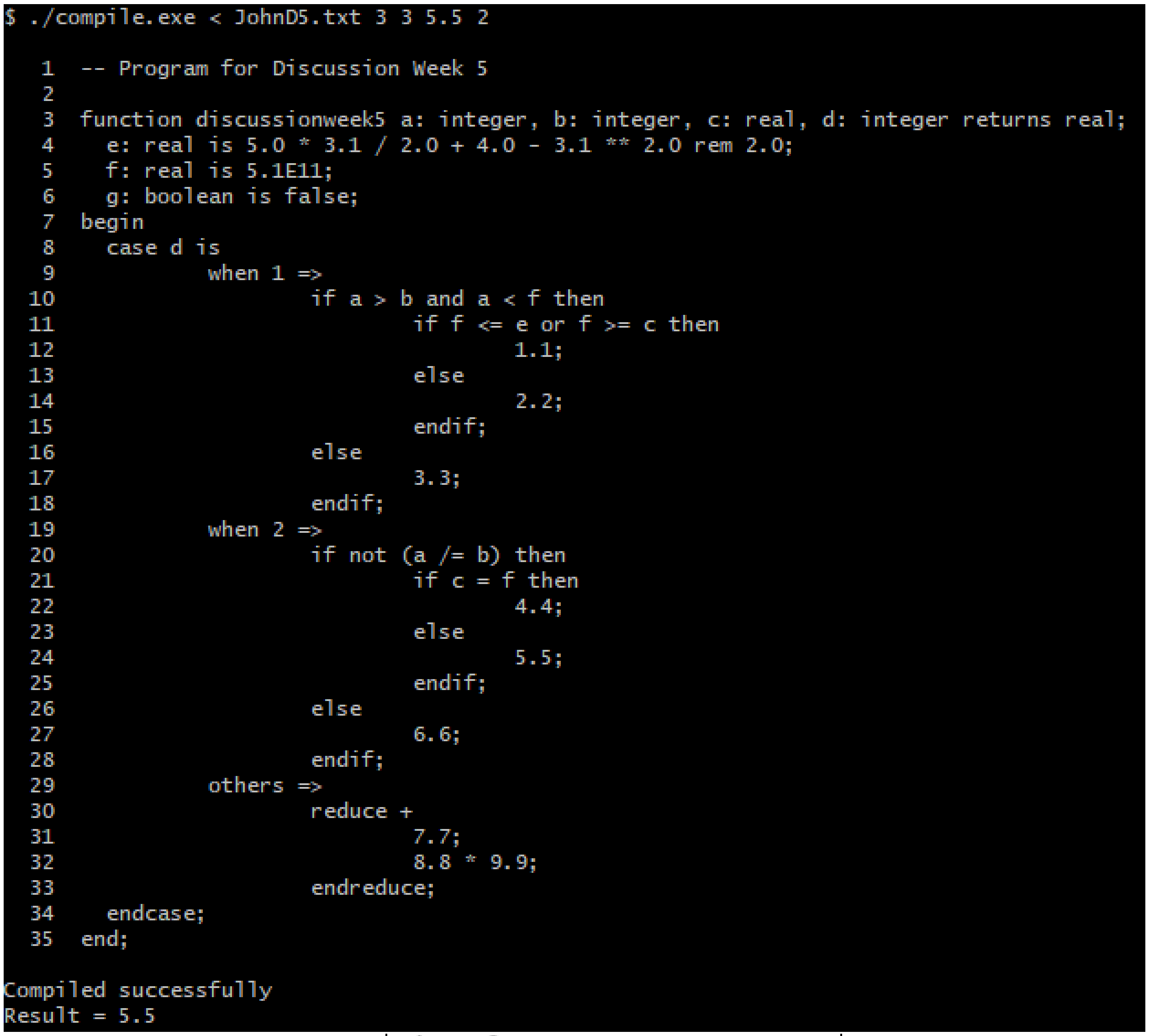
Figure 14. Test case #3 of John Kucera’s program. Results are as expected.



```
$ ./compile.exe < JohnD5.txt 3 3 5.1E11 2

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 4.4
```

a: 3, b: 3, c: 5.1E11, d: 2	when 2 =>
	if not a /= b then
	if c = f then
		4.4

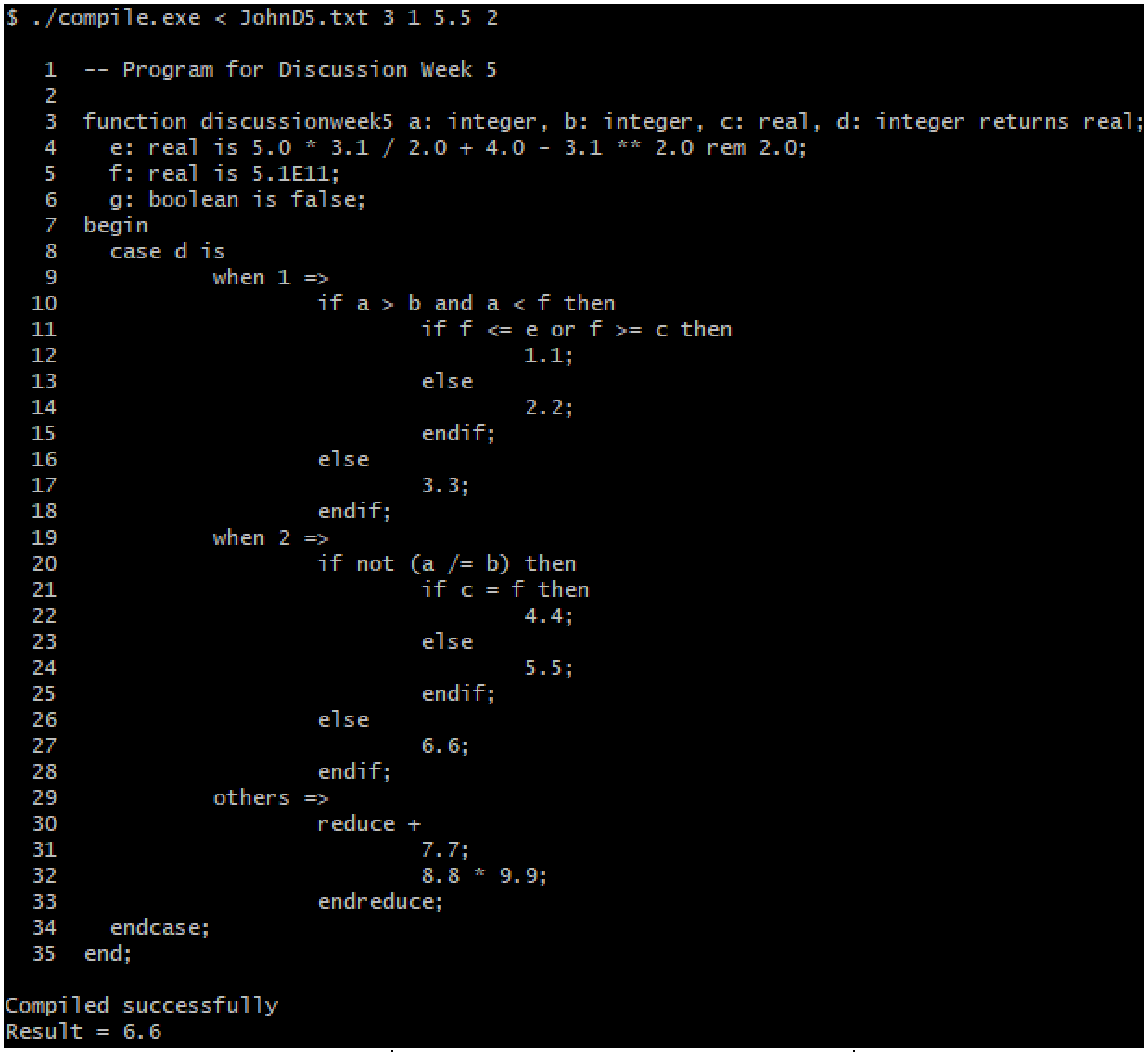
Figure 15. Test case #4 of John Kucera's program. Results are as expected.



```
$ ./compile.exe < JohnD5.txt 3 3 5.5 2

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 5.5
```

a: 3, b: 3, c: 5.5, d: 2	when 2 =>
	if not a /= b then
	if c = f then (else)
		5.5

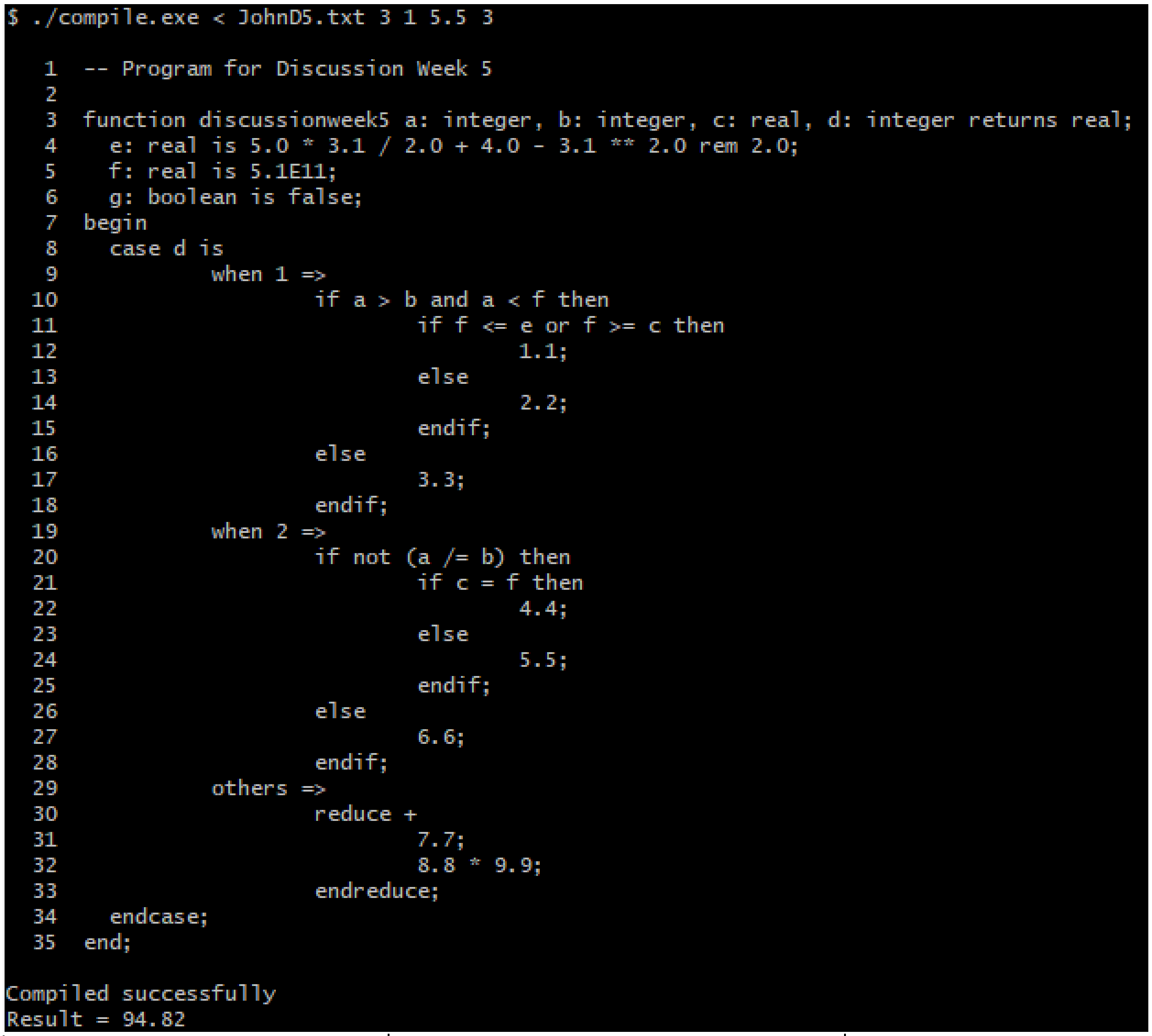
Figure 16. Test case #5 of John Kucera's program. Results are as expected.



```
$ ./compile.exe < JohnD5.txt 3 1 5.5 2

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 6.6
```

a: 3, b: 1, c: 5.5, d: 2	when 2 => if not a /= b then (else)	6.6

Figure 17. Test case #6 of John Kucera’s program. Results are as expected.



```
$ ./compile.exe < JohnD5.txt 3 1 5.5 3

-- Program for Discussion Week 5

function discussionweek5 a: integer, b: integer, c: real, d: integer returns real;
  e: real is 5.0 * 3.1 / 2.0 + 4.0 - 3.1 ** 2.0 rem 2.0;
  f: real is 5.1E11;
  g: boolean is false;
begin
  case d is
          when 1 =>
                  if a > b and a < f then
                          if f <= e or f >= c then
                                  1.1;
                          else
                                  2.2;
                          endif;
                  else
                          3.3;
                  endif;
          when 2 =>
                  if not (a /= b) then
                          if c = f then
                                  4.4;
                          else
                                  5.5;
                          endif;
                  else
                          6.6;
                  endif;
          others =>
                  reduce +
                          7.7;
                          8.8 * 9.9;
                  endreduce;
  endcase;
end;

Compiled successfully
Result = 94.82
```

a: 3, b: 1, c: 5.5, d: 3	others =>
	reduce +
	7.7;
	8.8 * 9.9;
		94.82

Figure 18. Test case #7 of John Kucera's program. Results are as expected.