Project 3

Terrence Jackson

CMSC 430: Compiler Theory and Design

Professor McDonald

September 24, 2024

Approach

I began this project by carefully setting up the provided skeleton code to ensure a solid foundation. This initial phase involved running the code and its associated test cases, which helped me gain a clear understanding of how semantic actions were being used to interpret the language. After that, I ported over my previous work from Project 2, merging it with the new structure while maintaining stability. This careful integration ensured that I wouldn't disrupt the existing functionality while incorporating the required enhancements. Incremental development was my strategy, allowing me to focus on specific features and guarantee that they worked before proceeding to the next phase.

Once I had my groundwork laid, I tackled the basic features like hexadecimal literals and escape characters. Implementing hexadecimal literals required me to write a custom atoh() function, which I then debugged to handle the proper conversion of values. The escape characters were tricky, but after experimenting with a few methods, I landed on using a switch statement to manage the conversion. My approach to adding new arithmetic and logical operators was methodical. For instance, exponentiation required using the C pow() function, and I had to ensure that division by zero errors were handled properly. The final phase involved adding the if and fold statements, where I tested each feature with both provided and custom test cases. When implementing fold statements, I faced the challenge of evaluating lists and applying operations, but breaking it down with a helper function made it manageable.

Throughout the process, I focused on preserving the integrity of my code. Each new feature was tested before moving on, preventing issues from snowballing. By the time I reached the final testing phase, I had refined and debugged the project extensively. Testing with all the provided cases and additional ones of my own allowed me to confidently deliver a robust final product.

Test Plan

Test Case	Input	Expected Output	Pass?

test1	// Function with Arithmetic Expression
	
	function main returns integer;
	begin
	  7 + 2 * (5 + 4);
	end;
		Compiled Successfully
		Result = 25
			Y

test2	// Function with When Statement
	
	function main returns integer;
	begin
	  when 3 < 2 & 6 < 7, 7 * 2 : 7 + 2;
	end;
		Compiled Successfully
		Result = 9
			Y

test3	// Function with a Switch Statement
	
	function main returns character;
	  a: integer is 2 * 2 + 1;
	begin
	  switch a is
	  case 1 => 'a';
	  case 2 => 'b';
	  others => 'c';
	  endswitch;
	end;
		Compiled Successfully
		Result = 99
		Note: ascii ‘c’ = 99
			Y

test4	// Function with a List Variable
	
	function main returns integer;
	  primes: list of integer is (2, 3, 5, 7);
	begin
	  primes(1) + primes(2);
	end;
		Compiled Successfully
		Result = 8
			Y

test5	// Function with Arithmetic Expression using Real Literals // and Hexadecimal Integer Literals function main returns real; begin	Compiled Successfully Result = 115.57	Y

	.83e+2 + 2.5E-1 + (4.3E2 + #aF2) * .01;
	end;		

test6	// Function with Character Literal Escape Characters
	
	function main returns character;
	lines: integer is 60;
	begin
	when lines < 60, '\n' : '\f';
	end;
		Compiled Successfully
		Result = 12
			Y

test7	// Tests All Arithmetic Operators
	
	function main returns integer;
	begin
	9 + 2 - (5 + ~1) / 2 % 3 * 3 ^ 1 ^ 2;
	end;
		Compiled Successfully
		Result = 5
			Y

test8	// Test Logical and Relational Operators
	
	function main returns integer;
	begin
	when !(6 <> 7) & 5 + 4 >= 9 | 6 = 5, 6 + 9 * 3 : 8 - 2 ^ 3;
	end;
		Compiled Successfully
		Result = 0
			Y

test9	// Function with an If Statement
	
	function main returns integer;
	a: integer is 28;
	begin
	if a < 10 then
	1;
	elsif a < 20 then
	2;
	elsif a < 30 then
	3;
	else
	4;
	endif;
	end;
		Compiled Successfully
		Result = 3
			Y

test10	-- Multiple Variable Initializations
	
	function main returns character;
	        b: integer is 5 + 1 - 4;
	        c: real is 2 + 3.7;
	        d: character is 'A';
	begin
	        if b + 1 - c > 0 then
	        d;
	        else
	        '\n';
	        endif;
	end;
		Compiled Successfully
		Result = 10
			Y

test11	// Single Parameter Declaration
	
	function main a: real returns real;
	begin
	        a + 1.5;
	end;
	
	./compile < test11.txt 6.8
		Compiled Successfully
		Result = 8.3
			Y

test12	// Two parameter declarations
	
	function main a: integer, b: real returns real;
	begin
	        if a < #A then
	        b + 1;
	        else
	        b - 1;
	        endif;
	end;
	
	./compile < test12.txt 16 15.9
		Compiled Successfully
		Result = 14.9
			Y

test13	// Test Right Fold
	
	function main returns integer;
	        values: list of integer is (3, 2, 1);
	begin
	        fold right - values endfold;
	end;
		Compiled Successfully
		Result = 2
			Y

test14	// Test Left Fold
	
	function main returns integer;
	begin
	    fold left - (3, 2, 1) endfold;
	end;
		Compiled Successfully
		Result = 0
			Y

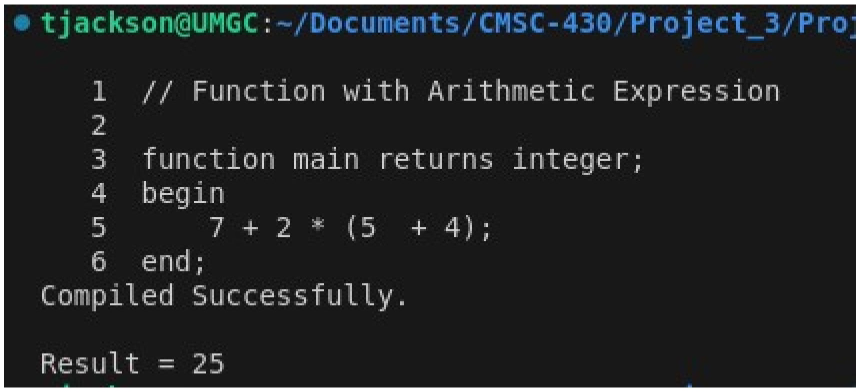
test15	// Test that Includes All Statements
	
	function main a: integer, b: real, c: character returns real;
	    d: integer is when a > 0, #A: #A0;
	    e: list of integer is (3, 2, 1);
	    f: integer is
	    switch a is
	    case 1 => fold left - e endfold;
	    case 2 => fold right - e endfold;
	    others => 0;
	    endswitch;
	    g: integer is
	    if c = 'A' & b > 0 then
	    a * 2;
	    elsif c = 'B' | b < 0 then
	    (a ^ 2) * 10;
	    else
	    0;
	    endif;
	begin
	    f + (d - 1) % g;
	end;
	
	./compile < test15.txt 1 2.5 65
		Compiled Successfully
		Result = 1
			Y

test16	// Nested if statements
	
	function main x: integer, y: integer returns character;
	begin
	    if x > 0 then
	    if y > 0 then
	    '1';
	    elsif y < 0 then
	    '4';
	    else
	    'Y';
	    endif;
		Compiled Successfully
		Result = 89
		Note: ascii 'Y' is 89
			Y

	elsif x < 0 then
	if y > 0 then
	'3';
	elsif y < 0 then
	'2';
	else
	'Y';
	endif;
	else
	if y <> 0 then
	'X';
	else
	'O';
	endif;
	endif;
	end;
	
	./compile < test16.txt -10 0		

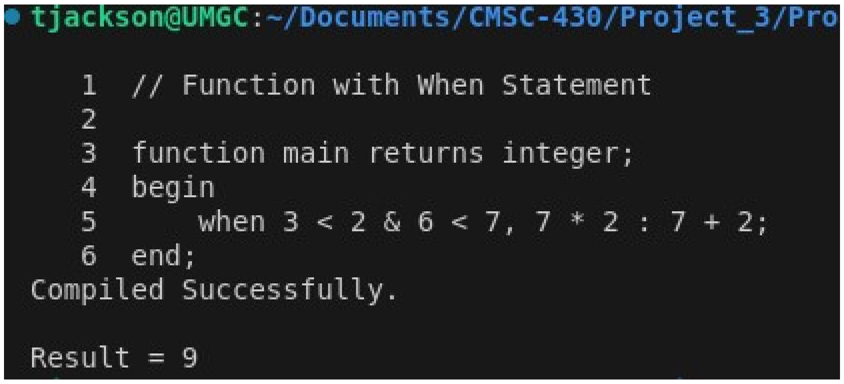
test17	// Tests Division by 0
	
	function main returns integer;
	begin
	9 / 0;
	end;
		1 Syntax Error	Y

Test Case 1:



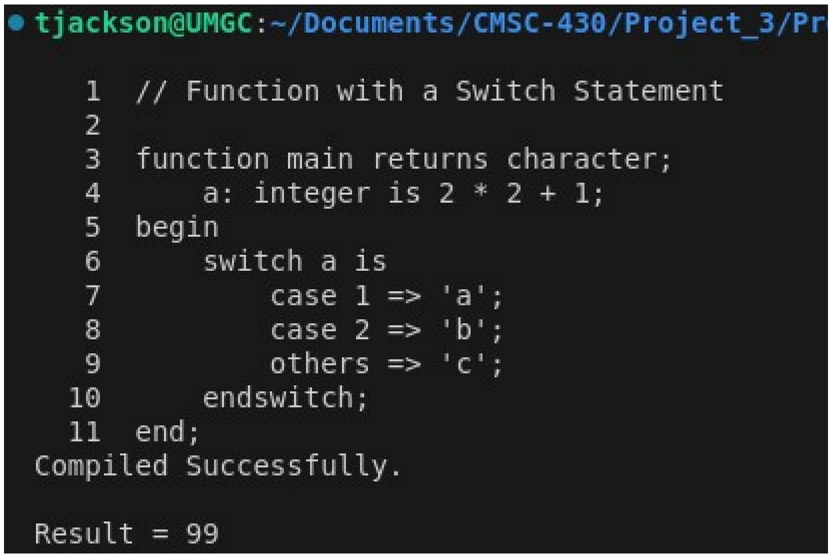
```
tjackson@UMGC:~/Documents/CMSC-430/Project_3/Pro

  // Function with Arithmetic Expression

  function main returns integer;
  begin
      7 + 2 * (5  + 4);
  end;
Compiled Successfully.

Result = 25
```

Test Case 2:



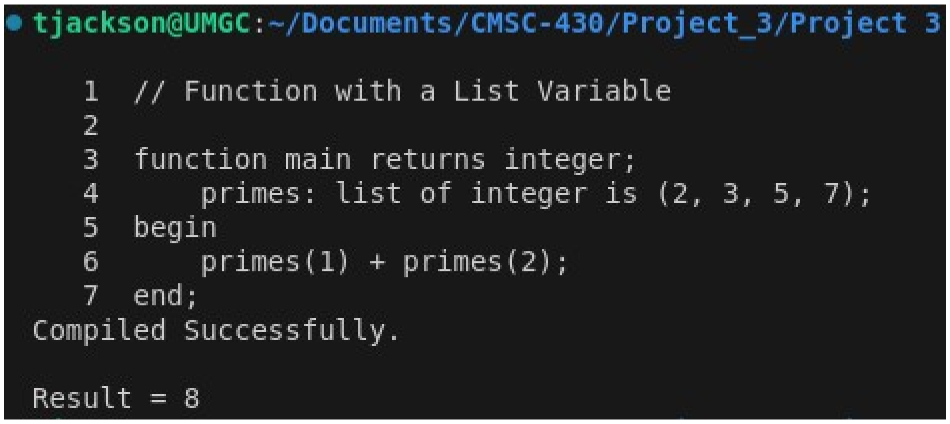
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Pro

// Function with When Statement

function main returns integer;
begin
    when 3 < 2 & 6 < 7, 7 * 2 : 7 + 2;
end;
Compiled Successfully.

Result = 9
```

Test Case 3:



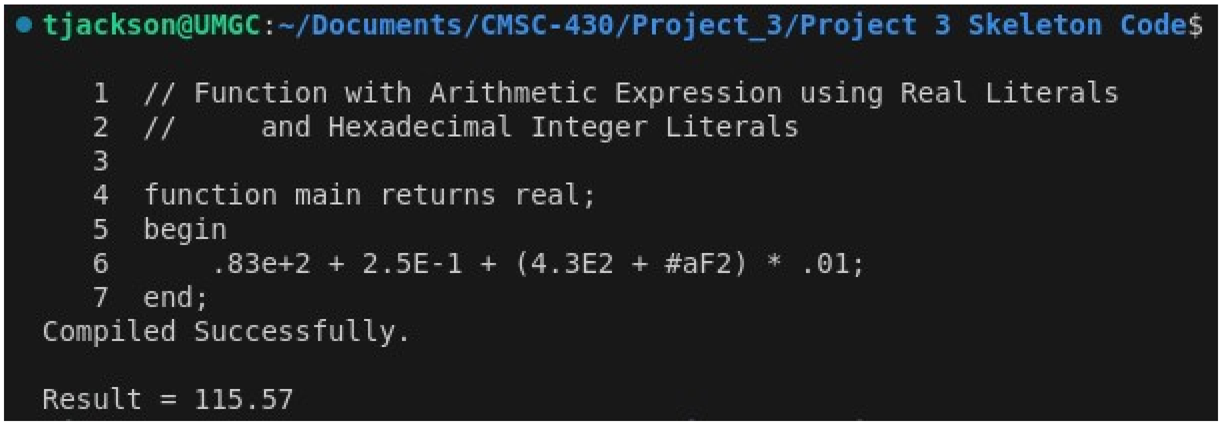
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Pr

// Function with a Switch Statement

function main returns character;
    a: integer is 2 * 2 + 1;
begin
    switch a is
        case 1 => 'a';
        case 2 => 'b';
        others => 'c';
    endswitch;
end;
Compiled Successfully.

Result = 99
```

Test Case 4:



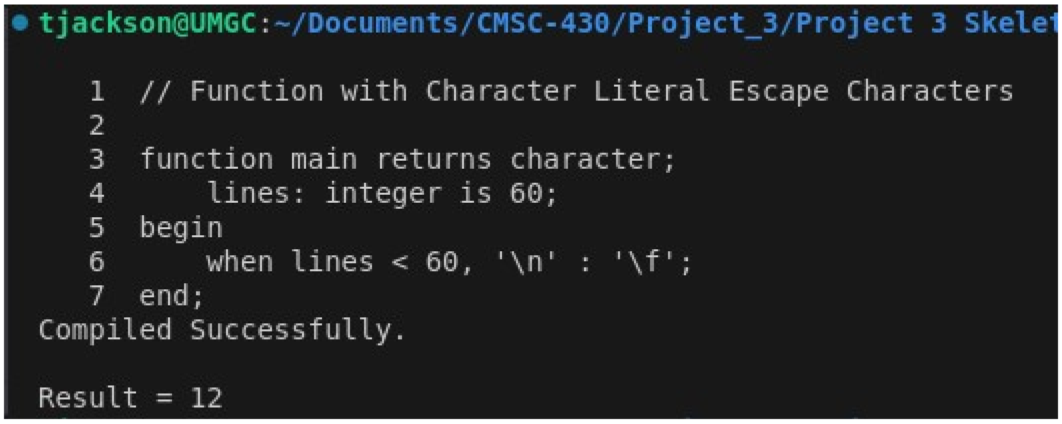
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3

// Function with a List Variable

function main returns integer;
    primes: list of integer is (2, 3, 5, 7);
begin
    primes(1) + primes(2);
end;
Compiled Successfully.

Result = 8
```

Test Case 5:



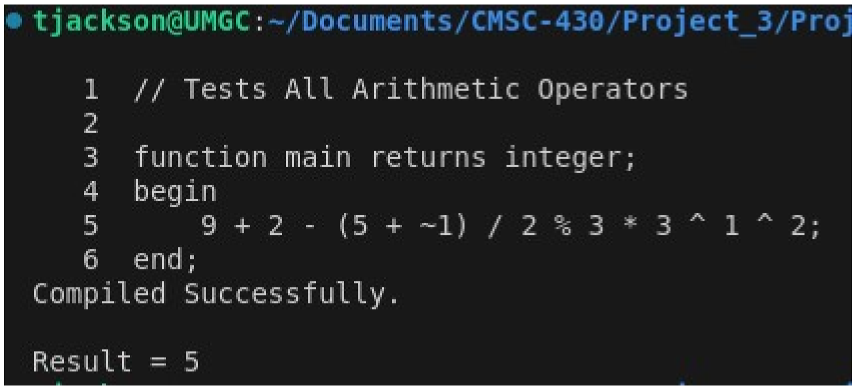
```
tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 Skeleton Code$

// Function with Arithmetic Expression using Real Literals
//     and Hexadecimal Integer Literals

function main returns real;
begin
    .83e+2 + 2.5E-1 + (4.3E2 + #aF2) * .01;
end;
Compiled Successfully.

Result = 115.57
```

Test Case 6:



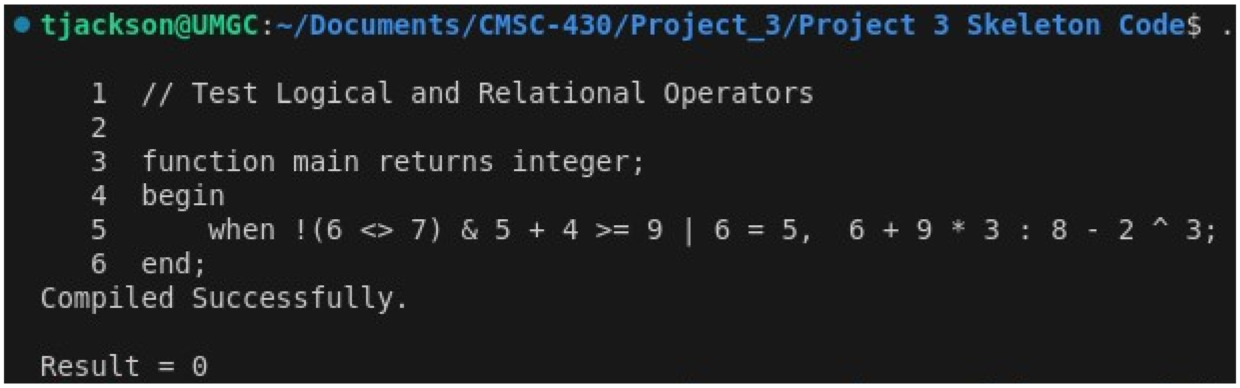
```
tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 Skelet

// Function with Character Literal Escape Characters

function main returns character;
    lines: integer is 60;
begin
    when lines < 60, '\n' : '\f';
end;
Compiled Successfully.

Result = 12
```

Test Case 7:



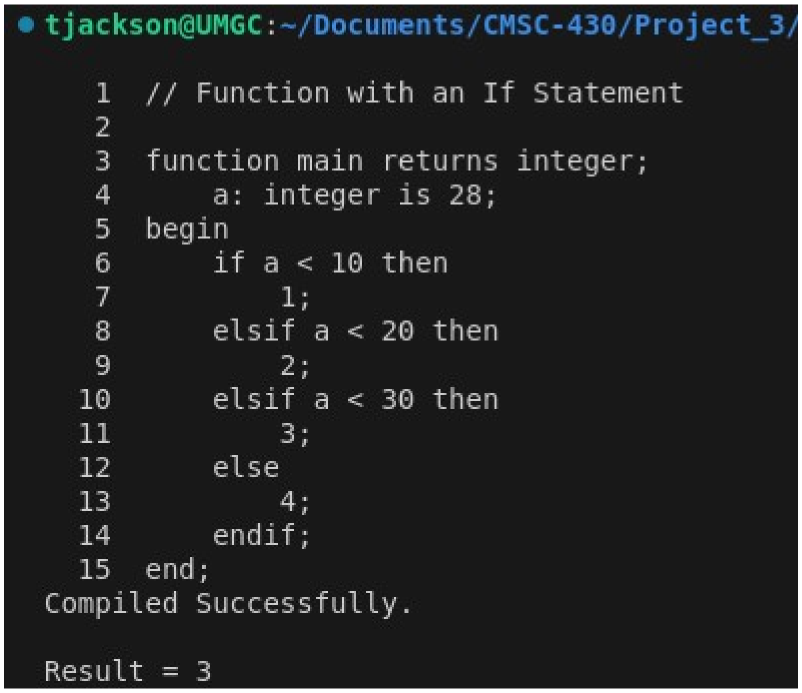
```
tjackson@UMGC:~/Documents/CMSC-430/Project_3/Proj

// Tests All Arithmetic Operators

function main returns integer;
begin
    9 + 2 - (5 + ~1) / 2 % 3 * 3 ^ 1 ^ 2;
end;
Compiled Successfully.

Result = 5
```

Test Case 8:



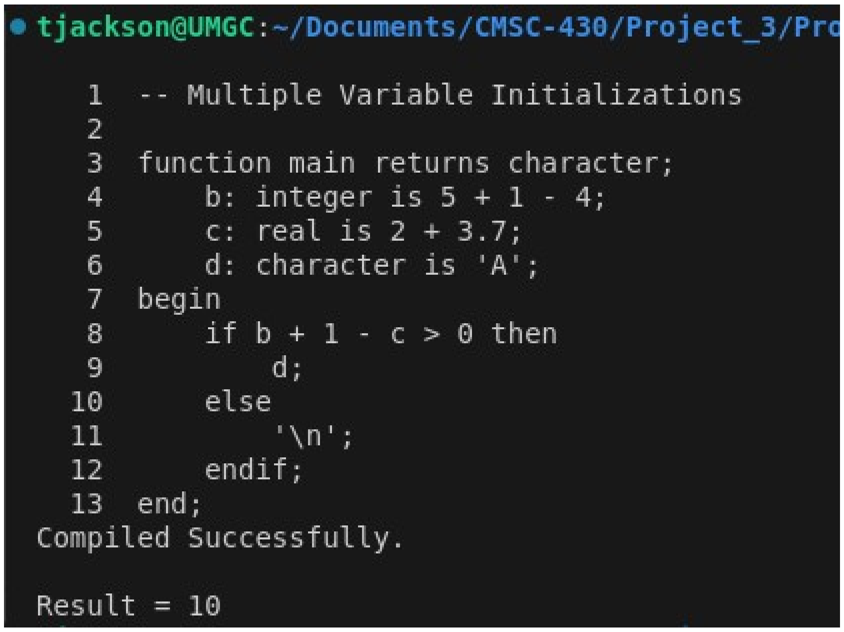
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 Skeleton Code$ .

  // Test Logical and Relational Operators

  function main returns integer;
  begin
      when !(6 <> 7) & 5 + 4 >= 9 | 6 = 5,  6 + 9 * 3 : 8 - 2 ^ 3;
  end;
Compiled Successfully.

Result = 0
```

Test Case 9:



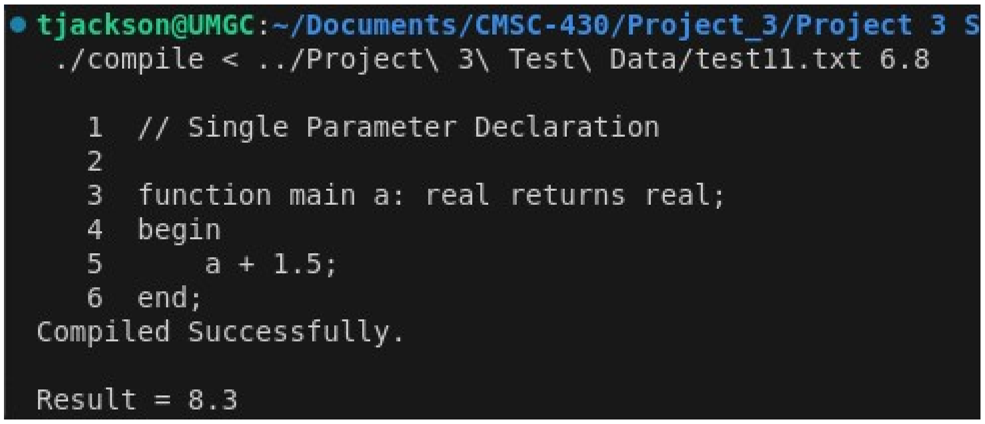
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/

  // Function with an If Statement

  function main returns integer;
      a: integer is 28;
  begin
      if a < 10 then
          1;
      elsif a < 20 then
          2;
      elsif a < 30 then
          3;
      else
          4;
      endif;
  end;
Compiled Successfully.

Result = 3
```

Test Case 10:



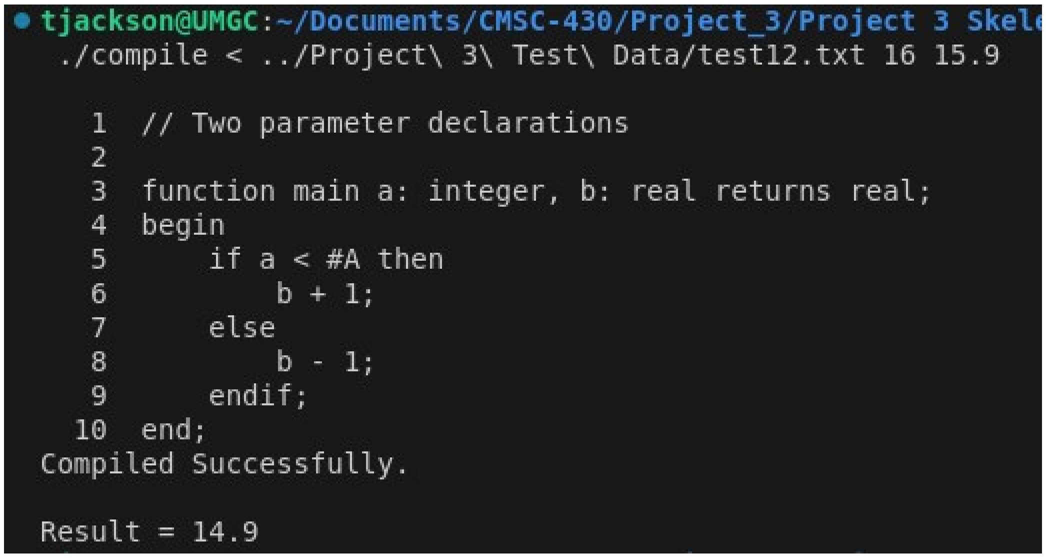
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Pro

-- Multiple Variable Initializations

function main returns character;
    b: integer is 5 + 1 - 4;
    c: real is 2 + 3.7;
    d: character is 'A';
begin
    if b + 1 - c > 0 then
        d;
    else
        '\n';
    endif;
end;
Compiled Successfully.

Result = 10
```

Test Case 11:



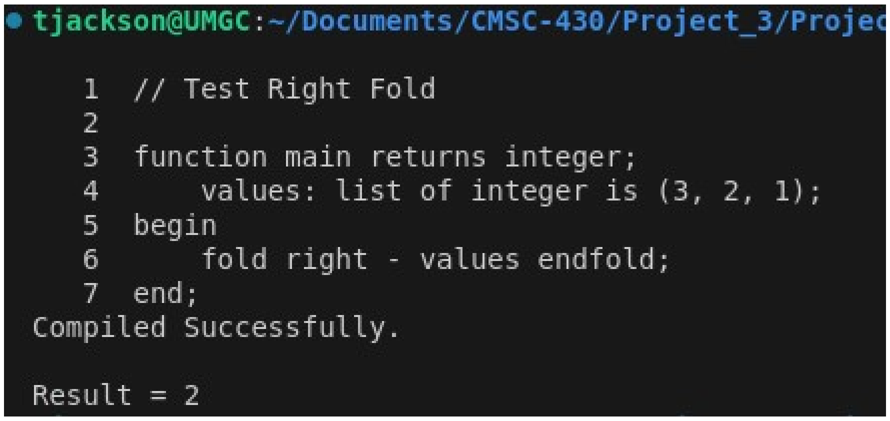
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 S
  ./compile < ../Project\ 3\ Test\ Data/test11.txt 6.8

// Single Parameter Declaration

function main a: real returns real;
begin
    a + 1.5;
end;
Compiled Successfully.

Result = 8.3
```

Test Case 12:



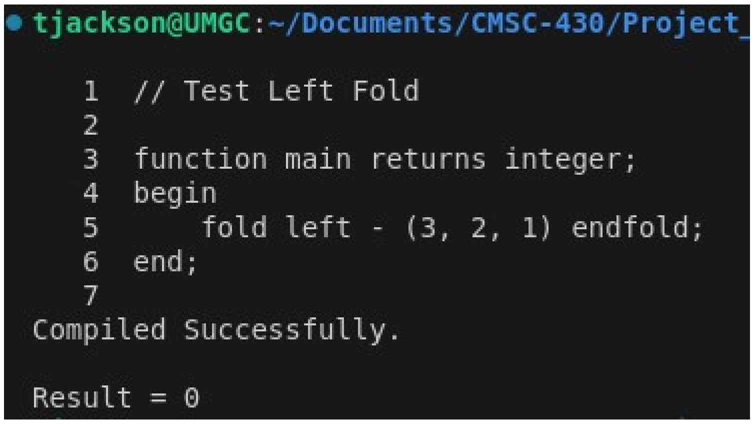
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 Skele
  ./compile < ../Project\ 3\ Test\ Data/test12.txt 16 15.9

   // Two parameter declarations

   function main a: integer, b: real returns real;
   begin
       if a < #A then
           b + 1;
       else
           b - 1;
       endif;
   end;
 Compiled Successfully.

 Result = 14.9
```

Test Case 13:



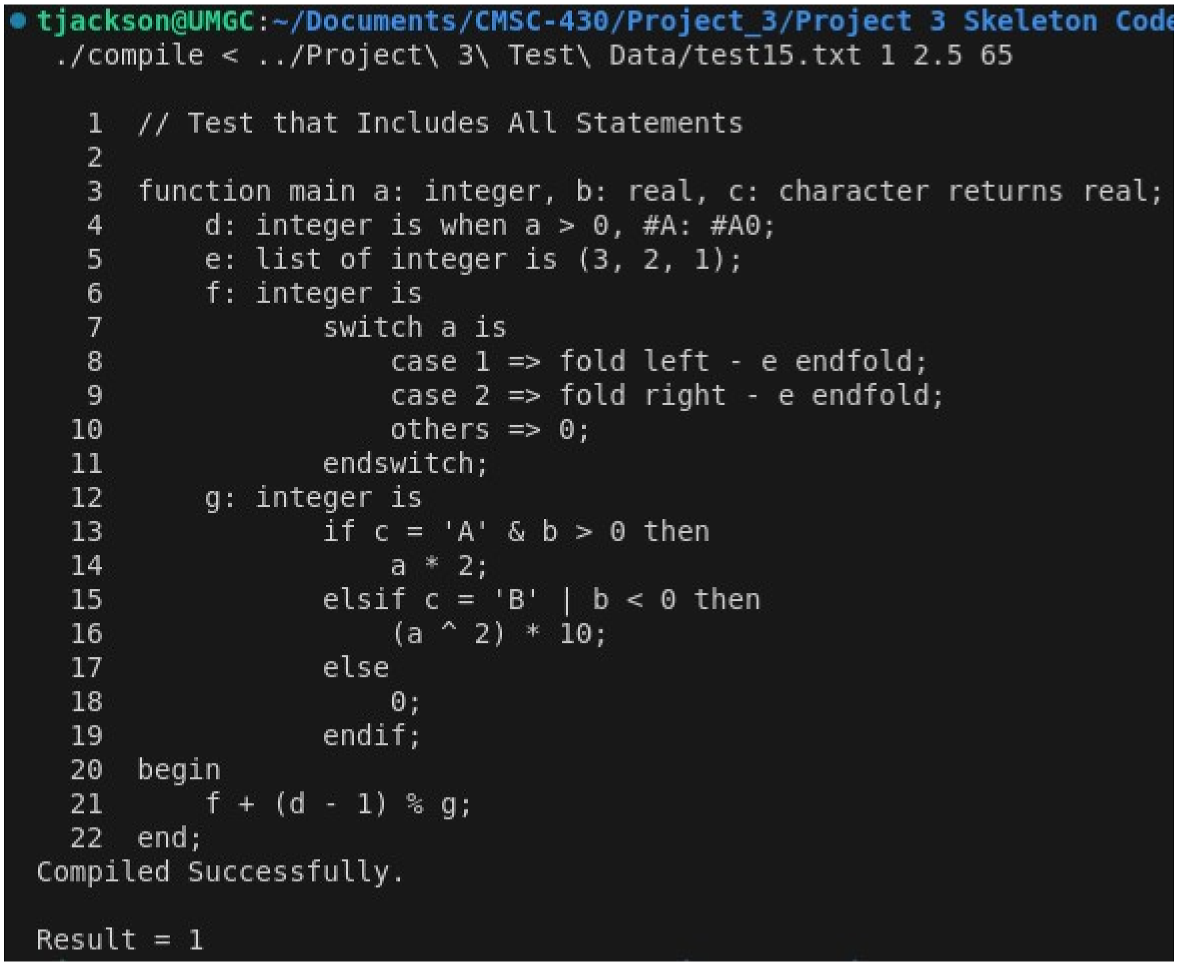
```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Projec

   // Test Right Fold

   function main returns integer;
       values: list of integer is (3, 2, 1);
   begin
       fold right - values endfold;
   end;
 Compiled Successfully.

 Result = 2
```

Test Case 14:



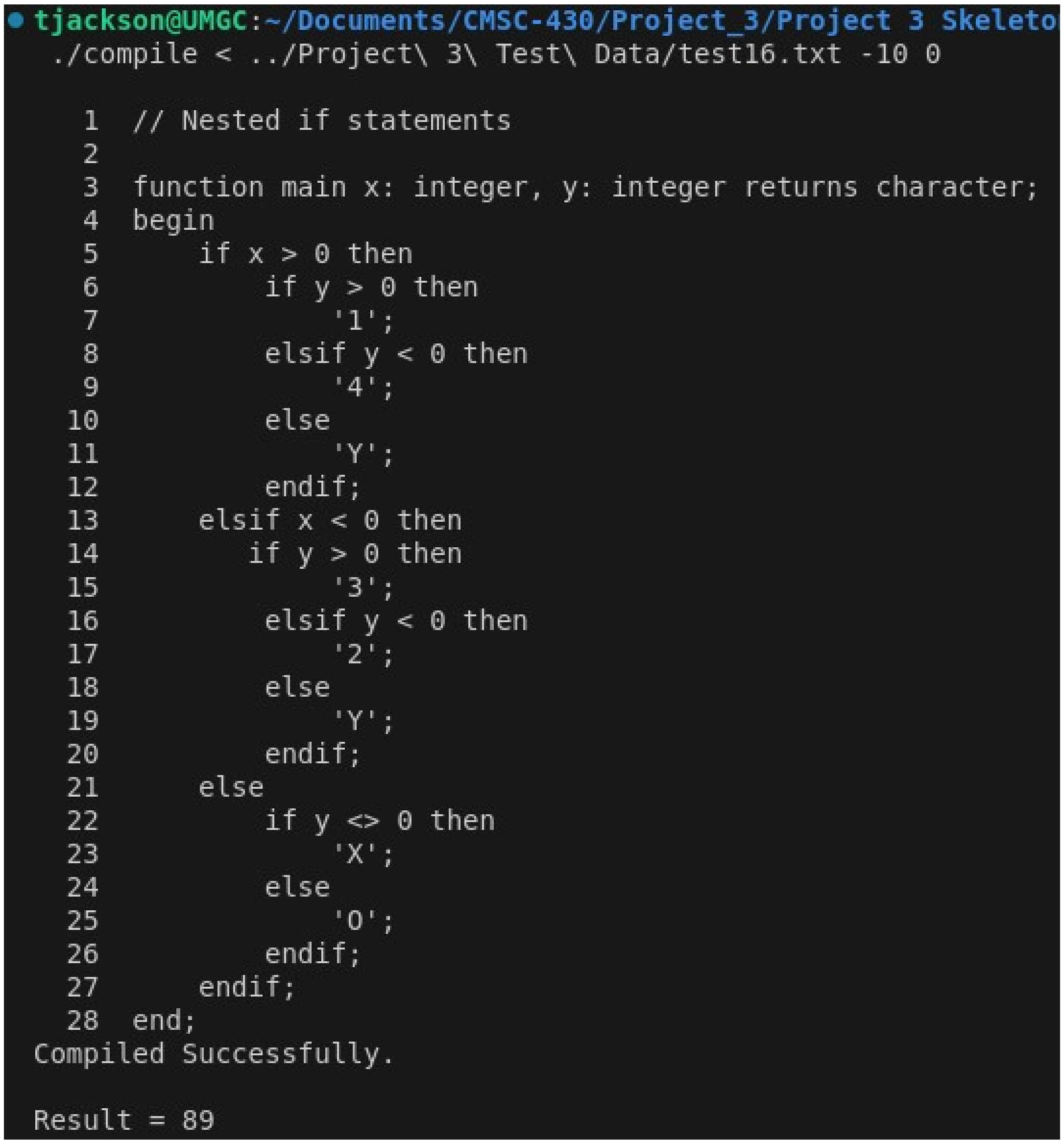
```
tjackson@UMGC:~/Documents/CMSC-430/Project_

// Test Left Fold

function main returns integer;
begin
    fold left - (3, 2, 1) endfold;
end;

Compiled Successfully.

Result = 0
```

Test Case 15:



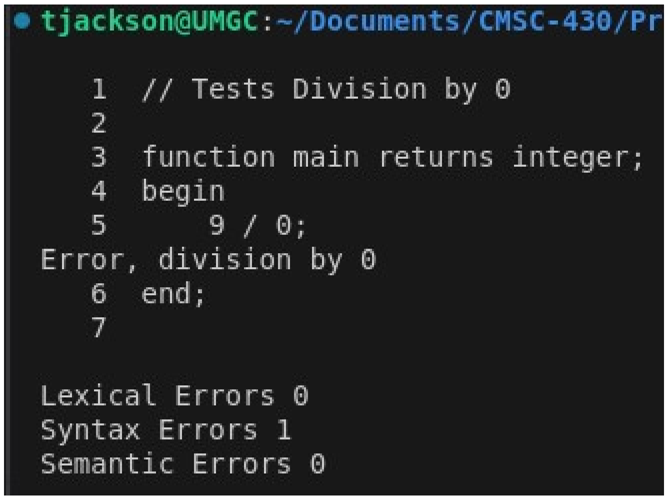
```
tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 Skeleton Code
 ./compile < ../Project\ 3\ Test\ Data/test15.txt 1 2.5 65

// Test that Includes All Statements

function main a: integer, b: real, c: character returns real;
    d: integer is when a > 0, #A: #A0;
    e: list of integer is (3, 2, 1);
    f: integer is
           switch a is
               case 1 => fold left - e endfold;
               case 2 => fold right - e endfold;
               others => 0;
           endswitch;
    g: integer is
           if c = 'A' & b > 0 then
               a * 2;
           elsif c = 'B' | b < 0 then
               (a ^ 2) * 10;
           else
               0;
           endif;
begin
    f + (d - 1) % g;
end;
Compiled Successfully.

Result = 1
```

Test Case 16:



```
● tjackson@UMGC:~/Documents/CMSC-430/Project_3/Project 3 Skeleto
  ./compile < ../Project\ 3\ Test\ Data/test16.txt -10 0

  // Nested if statements

  function main x: integer, y: integer returns character;
  begin
      if x > 0 then
          if y > 0 then
              '1';
          elsif y < 0 then
              '4';
          else
              'Y';
          endif;
      elsif x < 0 then
         if y > 0 then
              '3';
          elsif y < 0 then
              '2';
          else
              'Y';
          endif;
      else
          if y <> 0 then
              'X';
          else
              'O';
          endif;
      endif;
  end;
Compiled Successfully.

Result = 89
```

Test Case 17:



```
• tjackson@UMGC:~/Documents/CMSC-430/Pr

    1  // Tests Division by 0
    2
    3  function main returns integer;
    4  begin
    5      9 / 0;
 Error, division by 0
    6  end;
    7

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

Lessons Learned

One of the key lessons I took away from this project is the value of incremental development and rigorous testing. By approaching the project feature by feature, I was able to maintain stability and catch issues early, before they became deeply embedded in the code. This approach also allowed me to tackle more complex tasks, such as implementing fold statements, without feeling overwhelmed. Testing at each step ensured that I could isolate problems and fix them quickly. Moving forward, I will continue to rely on this methodical approach to coding, as it has proven to be invaluable in minimizing errors.

I also learned the importance of trusting simpler solutions when appropriate. I initially resisted using a switch statement for handling escape characters, overthinking the problem and experimenting with more complicated approaches. In hindsight, choosing the simpler option saved me time and mental energy. This experience taught me to recognize when straightforward solutions can be just as effective, and to avoid over-engineering problems. Similarly, my time spent debugging the hexadecimal conversion function reinforced the importance of taking a break when stuck. Walking away from the project for a few days helped me return with fresh eyes, allowing me to quickly identify the issue and resolve it.

The project further highlighted the challenges of error handling in complex systems. When implementing division by zero error handling, I struggled initially to connect the arithmetic operations with the error-handling functions located in different files. However, once I understood how to leverage helper functions and manage cross-file communication, I was able to implement the necessary functionality. This was a rewarding aspect of the project, as it deepened my understanding of how different parts of a system must communicate effectively to handle errors smoothly. Overall, this project strengthened my problem-solving skills, and I now have a more confident grasp of how to tackle future challenges in compiler construction and interpretation.

References

Type Inference in C++. GeeksforGeeks. (2024, July 29). https://www.geeksforgeeks.org/type-inference-in-c-auto-and-decltype/

Vector in C++ STL. GeeksforGeeks. (2024, July 5). https://www.geeksforgeeks.org/vector-in-cpp-stl/