Wanner HernandezR

CMSC430 Week 8 Project 4

Professor Jarc, Duane

May 10, 2021

**CMSC 430 Project 4**

The fourth project involves modifying the semantic analyzer for the attached compiler by adding checks for semantic errors. The static semantic rules of this language are the following:

Variables and parameter names have local scope. The scope rules require that all names be declared and prohibit duplicate names within the same scope. The type correspondence rules are as follows:

•  Boolean expressions cannot be used with arithmetic or relational operators.

•  Arithmetic expressions cannot be used with logical operators.

•  Reductions can only contain numeric types.

•  Only integer operands can be used with the remainder operator.

•  The two statements in an if statement must match in type. No coercion is performed.

•  All the statements in a case statement must match in type. No coercion is performed.

•  The type of the if expression must be Boolean.

•  The type of the case expression must be Integer

•  A narrowing variable initialization or function return occurs when a real value is being forced into integer. Widening is permitted.

•  Boolean types cannot be mixed with numeric types in variable initializations or function returns.

Type coercion from an integer to a real type is performed within arithmetic expressions.

You must make the following semantic checks. Those highlighted in yellow are already performed by the code that you have been provided, although you are must make minor modifications to account for the addition of real types and the need to perform type coercion and to handle the additional arithmetic and logical operators.

•  Remainder Operator Requires Integer Operands

•  If-Then Type Mismatch

•  Case Types Mismatch

•  If Condition Not Boolean

•  Case Expression Not Integer

•  Narrowing V ariable Initialization

•  Duplicate Variable

•  Narrowing Function Return

Using Boolean Expressions with Arithmetic Operator

page1image2645144880

• Using Boolean Expressions with Relational Operator

page1image2557369264

• Using Arithmetic Expressions with Logical Operator

page1image2559665952

• Reductions containing nonnumeric types

page1image2560205584

• Variable Initialization Mismatch

page1image2560436160

• Undeclared Variable

page1image2559862800

This project requires modification to the bison input file, so that it defines the additional semantic checks necessary to produce these errors and addition of functions to the library of type checking functions already provided in types.cc. You must also make some modifications to the functions provided. You need to add a check to the checkAssignment function for mismatched types in the case that Boolean and numeric types are mixed. You need to also add code to the checkArithmetic function to coerce integers to reals when the types are mixed and the error message must be modified to indicate that numeric rather than only integer types are permitted.

The provided code includes a template class Symbols that defines the symbol table. It already includes a check for undeclared identifiers. You need to add a check for duplicate identifiers.

Like the lexical and syntax errors, the compiler should display the semantic errors in the compilation listing, after the line in which they occur. An example of compilation listing output containing semantic errors is shown below:

1 -- Test of Multiple Semantic Errors  
2  
3 function test a: integer returns integer; 4 b: integer is  
5 ifa+ 5 then  
6 2;  
7 else  
8 5;  
9 endif;

Semantic Error, If Expression Must Be Boolean

1. 10  c: real is 9.8 - 2 + 8;
2. 11   d: boolean is 7 = f;

Semantic Error, Undeclared f

12 begin

13 14 15

case b is

when 1 => 4.5 + c;

when 2 => b;

Semantic Error, Case Types Mismatch

16 others => c;

17 endcase;

18 end;

Lexical Errors 0

Syntax Errors 0

Semantic Errors 3

**Course file:**

* listing.cc
* listing.h
* scanner.l
* parser.y
* makerfile
* types.cc
* types.h

**After Compile:**

* tokens.h
* parser.c
* parser.o
* scanner.o
* listing.o
* types.o
* compile

**Approach:**

* My initial approach to this project was I watched the 4 part videos series on Semantic Analyzers.
* I then read Chapter 7 of Mogensen’s Textbook.
* After going through all the assignment material, I started by importing my project 2 code to the project 4 skeleton code.
* I then made the necessary changes to types.h, types.cc, scanner.l and paser.y according to the project requirements.
* Starting from the elements evaluated from the bottom of the parse tree and working my way up
* I then check my specific test plan to see if it was working correctly, unfortunately I did ran into a trouble with case statement to work the correct way as I wanted.

**Instruction to Compile code/open iOS:**

* Open Terminal
* Make sure to have private access on terminal before proceeding
* Prompt command cd desktop (to go to the require files)
* Prompt command cd CMSC430\_Project4(File name)
* Prompt command ‘make’ (ls to check file, make to compile the files)
* Compile: scanner.o parser.o listing.o values.o

g++ -o compile scanner.o parser.o listing.o types.o

* Scanner.o: scanner.c types.h listing.h tokens.h

g++ -c scanner.c

* Scanner.c: scanner.l

flex scanner.l

mv lex.yy.c scanner.c

* Parser.o: parser.c types.h listing.h symbols.h

g++ -c parser.c

* Parser.c tokens.h: parser.y

bison -d -v parser.y

mv parser.tab.c parser.c

mv parser.tab.h tokens.h

* Listing.o: listing.cc listing.h

g++ -c lisitng.cc

* Values.o: values.cc values.h

g++ -c types.cc

* After doing that just start testing your files
* Prompt command ./compile <test1.txt(name of the test file)
* The code will work successfully

**Test**

**Test 1:** Using Boolean Expression with Arithmetic Operator



**Test 2:** Using Real Operands with Remainder Operator



**Test 3:** Using Boolean Expression with Relational Operator



**Test 4:** Using Arithmetic Expression with Logical Operator



**Test 5:** Conditional Expression Type Mismatch



**Test 6:** If Condition Not Boolean



**Test 7:** Narrowing Variable Initialization



**Test 8:** Variable Initialization Mismatch



**Test 9:** Undeclared Local

Graphical user interface, text, application

Description automatically generated

**Test 10:** Duplicated Identifier



**Teste 11:** Narrowing Function Return



**Test 12:** Test of Non-Integer Case Expression

Graphical user interface, application

Description automatically generated with medium confidence

**Test 13:** Test of Case Types Mismatch



**Test 14:** Using Real Operands with Remainder Operator



**Test 15:** Function with Boolean in Reduction List



**Test 16:** Multiple Semantic Factors



**Valid1:** Program with a Real Variable

**Graphical user interface

Description automatically generated with medium confidence**

**Lessons learn:**

This week project 4 I was able to pick up new skills like how to read and resolved compile error and how to approach the error and take the necessary steps to fix this error. I came up with a few warning mistakes I try my best to fix this mistake with case statement but had an unfortunate result, I was able to make my code work, but I was never able to make the case Statement work the correct way. I was to modify the semantic analyzer for Variables and parameter names have local scope. I wish to see the correct way of how the code should have look for this class so that I could learn and pick up on the mistake I made through this class. Thanks for this class because I was able to see my weakness and where I need to improve more, as programmer.