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.

- Using Boolean Expressions with Arithmetic Operator
- Using Boolean Expressions with Relational Operator
- Using Arithmetic Expressions with Logical Operator
- Reductions containing nonnumeric types
- Remainder Operator Requires Integer Operands
- If-Then Type Mismatch
- Case Types Mismatch
- If Condition Not Boolean
- Case Expression Not Integer
- Narrowing Variable Initialization
- Variable Initialization Mismatch
- Undeclared Variable
- Duplicate Variable
- Narrowing Function Return

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          if a + 5 then
   6              2;
   7          else
   8              5;
   9          endif;
Semantic Error, If Expression Must Be Boolean
  10      c: real is 9.8 - 2 + 8;
  11      d: boolean is 7 = f;
Semantic Error, Undeclared f
  12  begin
  13      case b is
  14          when 1 => 4.5 + c;
  15          when 2 => b;
Semantic Error, Case Types Mismatch
  16          others => c;
  17      endcase;
  18  end;

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

You are to submit two files.

1. The first is a `.zip` file that contains all the source code for the project. The `.zip` file should contain the flex input file, which should be a `.l` file, the bison file, which should be a `.y` file, all `.cc` and `.h` files and a `makefile` that builds the project.
2. The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:

a. A discussion of how you approached the project
b. A test plan that includes test cases that you have created indicating what aspects of the program each one is testing and a screen shot of your compiler run on that test case
c. A discussion of lessons learned from the project and any improvements that could be made