CMSC 430 Project 3

The third project involves modifying the attached interpreter so that it interprets programs for the complete language.

When the program is run on the command line, the parameters to the function should be supplied as command line arguments. For example, for the following function header of a program in the file `text.txt`:

```
function main x: integer, y: integer returns character;
```

One would execute the program as follows:

```
$ ./compile < test.txt 10 -10
```

In this case, the parameter `x` would be initialized to `10` and the parameter `y` to `-10`. An example of a program execution is shown below:

```
$ ./compile < test.txt 10 -10

   1  // Determines the quadrant of a point on the x-y plane
   2
   3  function main x: integer, y: integer returns character;
   4  begin
   5      if x > 0 then
   6          if y > 0 then
   7              '1';
   8          elsif y < 0 then
   9              '4';
  10          else
  11              'Y';
  12          endif;
  13      elsif x < 0 then
  14         if y > 0 then
  15              '3';
  16          elsif y < 0 then
  17              '2';
  18          else
  19              'Y';
  20          endif;
  21      else
  22          if y <> 0 then
  23              'X';
  24          else
  25              'O';
  26          endif;
  27      endif;
  28  end;

Compiled Successfully

Result = 52
```

After the compilation listing is output, the value of the expression which comprises the body of the function should be displayed as shown above.

The existing code evaluates some of the arithmetic, relational and logical operators together with the case statement and decimal integer and real literals only. You are to add the necessary code to include all of the following:

- Hexadecimal integer and character literals that include escape characters
- All additional arithmetic operators
- All additional relational and logical operators
- Both `if` and `fold` statements
- Functions with multiple variables
- Functions with parameters

The `fold` statement repeatedly applies the specified operation to the list of values, producing one final value. A left `fold` associates the operator left to right and a right fold right to left. For example, the following left `fold`:

```
fold left - (3, 2, 1) endfold;
```

would be evaluated as ((3 – 2) – 1) = 0, but using a right `fold`:

```
fold right - (3, 2, 1) endfold;
```

It would be evaluated as (3 – (2 – 1)) = 2. For operations that are associative, the result would be the same whether it is as folded to the left or right.

This project requires modification to the bison input file, so that it defines the additional the necessary computations for the above added features. You will need to add functions to the library of evaluation functions already provided in `values.cc`. You must also make some modifications to the functions already provided.

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