Wanner HernandezR

CMSC430 Week 6 Project 3

Professor Jarc, Duane

April 27, 2021

**CMSC 430 Project 3**

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

You may convert all values to double values, although you can maintain their individual types if you wish.

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 a: integer, b: integer returns integer;

One would execute the program as follows:

$ ./compile < test.txt 2 4  
In this case, the parameter a would be initialized to 2 and the parameter b to 4.

An example of a program execution is shown below:

$ ./compile < test.txt 2 4

1

2

3

4

5

6

7

8 begin

9 case a is

function main a: integer, b: integer returns integer; c: integer is

ifa> b then a rem b;

else  
a \*\* 2;

endif;

1. 10  when 1 => c;
2. 11  when 2 =>(a+b/2-4)\*3;
3. 12   others => 4;
4. 13  endcase;

14 end;

Compiled Successfully

Result = 0

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 reduction statement and integer literals only. You are to add the necessary code to include all of the following:

* •  Real and Boolean literals
* •  All additional arithmetic operators
* •  All additional relational and logical operators
* •  Both if and case statements
* •  Functions with multiple variables
* •  Functions with parameters

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.

**Course file:**

* listing.cc
* listing.h
* scanner.l
* parser.y
* makerfile
* symbols.h
* values.cc
* values.h

**After Compile:**

* tokens.h
* parser.c
* parser.output
* parser.o
* scanner.o
* listing.o
* values.o
* compile

**Approach:**

* My initial approach to this project was downloading the Project 3 Skeleton Code and merging parser.y and scanner.l with what I had previously added to the lexical and syntactical analyzers.
* I then added the necessary enumerations to values.h.
* Once I was sure that my productions were in the correct bottom-up order in parser.y, I began to add more cases to the evaluation functions in values.cc.
* I then added a functioning if/else production and worked on accepting parameters from the command line.
* Finally, I worked on the case statement and made the necessary changes to make my project work correctly and efficient.

**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\_Project3(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 values.o

* Scanner.o: scanner.c values.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 values.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 values.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:** Arithmetic operators

Text, letter

Description automatically generated

**Test 2:** Relational and logical operators

Text

Description automatically generated with medium confidence

**Test 3:** Conditional expression

Graphical user interface

Description automatically generated with medium confidence

**Test 4:** Conditional expression

Graphical user interface

Description automatically generated with medium confidence

**Test 5:** Function with a reduction

Graphical user interface

Description automatically generated with medium confidence

**Test 6:** Multiple integer variable initialization

Graphical user interface

Description automatically generated with low confidence

**Test 7:** Variable initialization with real and Boolean variables

Text

Description automatically generated with low confidence

**Test 8:** Single parameter declaration

Graphical user interface

Description automatically generated with low confidence

**Test 9:** Two parameter declarations

Graphical user interface

Description automatically generated with medium confidence

**Test 10:** Nested If

Graphical user interface

Description automatically generated with medium confidence

**Teste 11:** Nested Reduction

Graphical user interface, application

Description automatically generated with medium confidence

**Test 12:** Nested Reduction

Graphical user interface, application

Description automatically generated

**Test 13:** Two parameter declarations

Graphical user interface

Description automatically generated with medium confidence

**Test 14:** Nested if

Graphical user interface, application

Description automatically generated with medium confidence

**Test 15:** Variable initialization with real and Boolean variables

**Text

Description automatically generated with low confidence**

**Lessons learn:**

This week project 3 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 small warning mistakes I was able to work through it and made the necessary changes to make those mistakes not impact the outcome of my code. I was able to learn more about converting values to double and had a better understanding on how to modify the attached interpreter so that it could interprets programs for the complete language. I was able to learn a few more tricks that was able to make me advance throughout this week problems and I am looking forward to keep Improving my skills.