Delivery 3 - CSCI 465 - Sam Dressler
Submit 12.15.2020
Program LOC: ~3500
TIME spent: hard to guess but I'm thinking around 50 hours

FCG = Final Code Generator
ICG = Intermediate Code Generator
Note:
I did not include the output from the lexical analyzer since it has been included in the previous two deliveries. I also did not include any of the error cases as seen from the lexical analyzer.

You have also seen the output in the terminal during my presentation so screenshots of that will not be included in this delivery except for errors.

For this delivery:

1. Types of integer and char are allowed, and variables can be declared. Type checking is used when variables
are assigned a value

2. full expressions are accepted including parenthesis. The expressions generate three address code using
temporary variables that are stored in the symbol table.

3. Write, read, writeln, readln are allowed and generate three address codes for each that can be used for final code generation.

4. If and If/Else statements are implemented. Both conditions as well as labels to mark the blocks are added to the three-address code section where it will be processed by the final code generator

5. While statements are generated into three address code where the final code generator will process it into MIPS assembly. The final code generator allows for characters as well as integers to be read into and stored. I am very proud of what I was able to accomplish for this last part of the project.

6. Beginning and end keywords are allowed for the program as well as in conditional statements.

7. Arrays can be declared for type integer. the implementation creates an entry in the symbol table for each index. When an array index is assigned a value, it checks to see if the index is in the range by checking if the value exists in the symbol table.

8. Machine code is generated into a form of MIPS. MIPS is a Reduced instruction set assembly language that I learned how to use in spring of 2020 while taking CSCI 370.
The MIPS that is generated is in the correct MIPS syntax and should run after being generated. two passes of the three-address code are ran, The first will find all print labels and labels for conditional statements and will prepare them for the second run through. The second run will generate the appropriate mips assembly code for each statement.
The generated MIPS assembly was assembled and ran using MARS v4.5.

9. Outputs for good and bad programs are in the zip file. The output for the bad program includes the errors
that would be generated while using the bad program.

Overall this delivery was very challenging but in terms of architecture, the second delivery was more difficult. There was a lot of time spent writing, debugging, and designing this compiler.
After taking this course there is a new-found resepect for all of the makings of the compiler. As something
that I use nearly every day working on programs for other classes, going through the process of building and understanding the program will be very beneficial in the rest of my computer science career.

Thank you,
Sam Dressler

Delivery 2 - CSCI 465 - Sam Dressler
Submit 11.19.2020
Program LOC : 1917

This program combines the lex, syntax analyzer, and intermediate code generator to produce
3-address code for expressions, IO statements, assignment statements, variable declarations,
and procedure and program definitions.

The amount of effort spent on the parts for delivery two approach 25 hours.

The program can successfully handle expressions of various complexities.
The program will output errors in the consol indicating missing semicolons,
undefined references to variables, missing parenthesis, etc.

The program is compiled using a makefile which generates an executeable "s++"
(Thought this would be a fitting name since the program was developed in c++
and compiled using g++ by a developer whose name starts with an 'S')

Design Descisions:
Symbol Table - vector of tuples which contain : ID,
Scope(temp, procedure, program, etc),
Scope ID (name of program or procedure, for temp it is the name of the procedure it is being set in)
type: variable type
value: string value or reference to temp var in the table

Parameter handling for procedures or programs:
- My goal with the design I went for was to be able to create a new node in alinked list for each new procedure, program, or function.
- This node contains information about the function as well as a pointer to the head of a seperate linked list
- The second linked list contains the parameters (id, type)

Scoping for variable declaration:
- Each procedure, program, or function with a var section will have their variables assigned

in that specific scope;
- variables that are declared in one scope cannot be accessed in another scope without causing an undefined reference error to be thrown by the compiler
- each begin statement prints the current scope in the terminal
- each end statement will display the previous scope after popping it off the scope stack.

Challenges:
-handling different forms of expressions was tedious
- dealing with the linked lists required a lot of extra code and were difficult to debug
(spent the first whole day of development on these for the program and procedure handling)
- fatigue - such a massive program requires an incredible amount of time spent sitting.
- started late - I'd say this is about half my fault, other courses have had non stop assignments as well so finding time to sit down and develop this was scarce.