CMPE 230 PROJECT 1 REPORT

IMPLEMENTING A COMPILER FOR MYLANG

Author: Kemal Berk Kocabagli
Student ID: 2013400045

In this project, given a source code in Mylang, a custom language that our instructor created, we were asked to translate it to working assembly a86 code and output the .asm file that creates an executable after being compiled in the a86 assembler.

To write the Mylang compiler, I followed three steps, which actually are interconnected to each other. I used C++ as my tool.

First, I read the input code with a global ifstream: input. This ifstream constitutes my tokenizer. I assumed in this step that every character in the given input file will be separated by at least one space. Once I read a token and it does not violate the Grammar of Mylang, I go on to the next token. Since the parsing is done in a recursive fashion, I needed the input to be global as my program goes through different functions while reading the input.

Second, I parse the given code according to the Grammar I was given. To achieve this, I used recursion. Hence, the parse tree is automatically formed, which will then be processed from bottom up. While the post-fix notation for arithmetic operations is handled by the recursion, the priority of division, mod and multiplication over addition and subtraction and the priority of expressions inside parenthesis are taken care of by the parse tree.

Thirdly and finally, the only thing left to do boils down to writing the a86 code to the output file. At this point, because my program already goes through the given Mylang code based on the parse tree, I implement the necessary a86 codes at the appropriate places. For instance, if my program is in the stm() function and the tokenizer has read an ID (this I check by another function called isID), I understand that this line of code should refer to a declaration/initialization. Therefore I insert the ID (variable name) into a global set where I keep all the variables in the Mylang code. I use a set here because I do not want to repeat a variable declaration in the a86 code. Additionally, to eliminate any unwanted coincidences concerning the variable names and built-in snippets of a86 code, I add the letter “v” in front of every variable name.

In the end, my program reads the input code written in Mylang and outputs its a86 assembly code equivalent. I especially made all of my recursive functions return an integer indicating whether the syntax is correct. If at any point an error in syntax is detected, the program outputs “SYNTAX ERROR. CANNOT COMPILE” and returns an empty output.asm.