Begüm Arslan-2018400189
Buse Tolunay- 2018400195

mylang2ir for MyLang Project

Our Project was to implement a compiler-like program that translates a source code to llvm code which is a low level language. Source was a simple programming language whose rules is set by Professor Özturan.
There were some problems to handle in the Project:

- We read line by line and at each line deleted redundant spaces before start.

1. Expressions
   How to handle expressions actually has been taught. First we need to turn expression to postfix. Fort his we used CFG that is provided. I think the most complicated part was factor, because a factor is the simplest part of an expression. The one that has the most precedence. Expressions can also have choose function called inside them. So returned value of choose also a factor. More details are on the choose part. The problems of expressions:
   a. We need to check if factors -that are not choose or expression inside paranthesis- are one token, and if not a number they are appropriate variable names. VariableCheck and tokenCheck handled these.
      - We decided “if”, “while”, “choose”, “print” can’t be variable names.
      - tokenCheck uses another function deletedoublespace. This function deletes one of the consecutive blank spaces and blank spaces at the each end of the expression.
      - Deletedoublespace also used for each line to get rid of redundant blank spaces.
   b. Second, -actually first- problem was the operators that are inside the paranthesis. We replace inside paranthesis with blank spaces before searching for operators.(discardParanthesis function)
   c. Third problem was evaluating postfixes, its method was also given. We used a stack. After calculations we put the result on a temporary variable. So we can use it in assignment or if- while statements etc.(evaluate function)
   d. We need to allocate if we encounter a non-declared varible, so did we.

2. Assignments
   If we saw a equal sign, this means there is an assignment. Expressions are evaluated here , than resulting temp is stored at the variable written to the left of the equal sign.

3. If – while branches
   If-while branches was difficult in terms of syntax errors. There was so much possibility. There is three part of if-while:
   a. Condition:

Condition was actually an expression. We calculated it. If expression and general if-while syntax is correct, we just write to the llvm file.

b. Body:
Inside a if-while body there can be print, assignment statements. Body ends with a "}", so we continued to get lines until we reach it. If we can't, this is an error. Also other syntax checks are handled here. The only difference between if and while is the end of the body. For while llvm should go to condition, for if should g oto end.

c. End:
A simple line that we write to llvm file.

4. Choose
Choose is teh function that is given to us. We decided to implement as a function on llvm. So we write this function at the top of the file, then call it when necessary. Choose also a factor. In factor function, it is checked syntactically and also 4 expressions seperated by commas sent to the returnedValueOfChoose.

a. Choose function
Prints the llvm code of print function which defines the choose function with 4 variables. Definition of llvm code is that ;if the first term equals to 0 function returns the second term,if the it is greater than 0 functionreturns the third expression ,else returns fourth variable.

b. returnedValueOfChoose
In here first, expressions of choose are calculated via expression and evaluation. Then we simply write results of these expressions to the call of choose in llvm as parameters.

5. Error handling
We used a global boolean variable named "error". We did several checks in our implementation to see if the error is true, that is there is a syntax error. If any, code is terminated. We also used another global variable to know which line has the error(lineNum).

6. Comments
Comments are cut off in both main when we read lines, also when we get lines of if-while statements.

The ups and downs of our code/approach and limitations

1. We made choose with a higher functionality than writing it to llvm with if-else branches each time it is called. Implementing choose as a function of llvm was an up
2. We tried make it efficient even though there was no time constraint. Doing extra checks for example if an expression has unbalanced paranthesis, we did not wait untill the factor function call and took early action without doing extra work.
3. We used a toOutfile string so we can make declarations before entering any loop. This saved us from traversing several times. But a string was not too much reliable in terms of space. We still used it.

4. discardParanthesis and deletedoublespace were not efficient functions. But we couldn't find a better way to tokenize expressions.
5. We succesfully used CFG, including choose with expression, term, moreTerms, factor, moreFactors.

How could we improve our program if we had time

1. We didn't use any classes. Maybe OOP would help us.
2. We could have find a better way to tokenize.
3. We could have used an array instead of a string for the output file. But we took some risk.