CMPE 230 Systems Programming Project 1 Report

Başak Önder – Sabri Mete Akyüz

Problem Description

In this Project, we implemented a translator, mylang2IR that takes the source code as an input and gives corresponding LLVM IR code as an output. Source code is an example of "MyLang" language that defines as follows:

1. It contains only integer variables, and their initial value is zero.
2. It may contain six types of lines: while statement lines, if statement lines, print statement lines, assignment lines, closed curly bracket lines and empty lines.
3. It contains choose function which takes four expressions as input and gives an output based on the value of first input
4. Expressions have operators *, /, +, - and parentheses. Precedence is the same as binary operand operations.
5. # sign is comment sign. After this sign, everything on that line must be considered as comments.

Problem Solution

We start the Project by writing the BNF structure of the expressions as follows:

<expr> → <term> <moreterms> | <term>

<moreterms> → + <term> <moreterms> | - <term> <moreterms>

<term> → <factor> <morefactors> | <factor>

<morefactors> → * <factor> <morefactors> | / <factor> <morefactors>

<factor> → (<expr>) | choose(<expr>,<expr>,<expr>,<expr>) | <var>

We implement this BNF part using right recursion to prevent infinite recursion loops with **isExpr(), isMoreterms(), isTerm(), isMorefactors(), isFactor()** methods. This part checks the given expression is legit or not, and also transforms it into postfix notation in order to evaluate easily by using stack structure. **exprCalc()** method is doing this calculating part by using postTokens vector from postfix notation and using a stack. Intermediate results are stored in temporary variables "veriveritemporari" in LLVM code.

In the second step, we take the input source code line by line. In order to tokenize it with stringstream, we wrote a **process()** method that takes the line as an input and place spaces in front and behind of some characters such as operators, parentheses, commas, and "#". After the tokenizing, we check this line to recognize which one of the six types corresponds to it. If it is a while or if line, we continue to take input lines in this section until encountering with a closed curly bracket line. Additionally, when a new variable is encountered, we initialize it into zero after than checking the variable name is alphanumeric.

At the final step, we implement the choose function. We take the most outside choose function and then we call **isExpr()** for its expressions one by one. We store the results of choose function in the result variables in LLVM code.

Conclusion

Program translate any source code that fits in the grammar rules into a LLVM IR code without an error. BNF and recursion parts make the implementation easier. However, choose function can be implemented by more efficient method somehow. While we are implementing it, we encountered with some errors and modify it regarding of them. We could plan our code design in a proper way. Apart from that, the program is **very cool** and working seamlessly.