1. Problem Description

In this assignment, we are asked to form a translator which converts the given language called MyLang to intermediate representation code that can be compiled by LLVM 3.3.

MyLang language is consists of,
- Assignment statements which have alphanumeric variables and expressions
- If and/or while statements which have expressions in their conditions
-Print statements which take variables as parameter

In this language, expression is operations of operands as variables and/or integers and operators as +, -, *, /.
If and while conditions are true if the expression is not equal to 0. Note that no nested if and/or while statements are allowed.
Hashtag (#), corresponds to comment sign.
There is a choose function which takes four expression parameters and returns the result of second expression if first expression is equal to 0, the result of third expression if first expression is greater than 0 and the result of fourth expression otherwise.
In case of syntax error, there is a warning output.
As a temporary variable we decided to use “gokberki” followed by an integer.

2. Problem Solution

First of all, we checked if there is a syntax error in the input by reading the input line by line. When we read the next line first task is to ignore all the comments in the line. After that if there is any choose statements, they need to be taken care of before we start checking other language constructs. Once we are free from choose calls, we decide what the syntax of that line should be depending on the first token (if,while,print statements) or existance of “=” operator in the line. Then we check the syntax of these lines token by token to see if there is a mistake. In the case of a mistake, syntaxCheck method returns the line which error occurs and we wrote the intermediate representation code which prints to the console “Line x: syntax error” when compiled by LLVM.

If there is no error detected, we wrote the intermediate representation code which prints to the console the result of the given input. Before the program starts, we made sure that necessary allocations are made and the variables are initiliazed to their initial value, 0. Reading the input line by line and deleting the whitespaces inside the line helped us to move forward.
If a line contains comment sign, it is rearranged by removing the comment part.
If a line contains choose call, it is rearranged by replacing the choose part with the temporary variable of the result of the choose function.
If a line is an assignment statement, it is processed by transforming infix notational expression to postfix notational expression, which will be assigned to the left hand side variable, when computed.
If a line contains if and/or while statement, it is processed by checking condition and directing the code by labeling.
If a line is a print statement, it is processed by reading the variable inside the parenthesis and giving it as parameter to called printf function which declared at the beginning of the program.

Now, let us take a look at methods and their usage.

int precedence(char c)
predence method is used when transforming an infix expression to postfix expression, decides which operation should come first. Returns an integer which decides the precedence for given character c.

string inToPost(string s)
InToPost method transforms an infix notational expression string to postfix notational expression string, note that postfix notational expression is parenthesis free. Returned postfix string will be checked for syntax errors

string spaceCanceller(string text)
spaceCanceller method gets rid of whitespaces and tab characters inside the strings, used for easy handle of lines.

bool syntaxPostFix(string str)
syntaxPostFix method checks the postfix expression, returns true if the expression is valid, false if not.

queue<string> evaluate(string postfix)
evaluate method stores the tokens of postfix notational expression to a queue and returns the queue. This queue will be used in expressionToIR method.

bool isVariable(string s)
isVariable method checks if given string is variable or not. Returns false if given string can be expressed as an integer, otherwise true.

bool isTemp(string s)
isTemp method checks if given string is temporary or not.

void expressionToIR(queue<string> q, stack<string> st , map<string, int> vars, bool fromAssignment,string variable_name)
expressionToIR method processes postfix notational expression which is stored in a queue. Forces LLVM to make necessary calculations with the help of a stack. During calculations if we come across a variable that has not been declared before, declare it and allocate memory. Last two parameters are for assignment statements. Depending on bool value makes necessary calculations.

int syntaxCheck(ifstream &inFile2)
syntaxCheck method checks the whole input, if there is any error detected returns the number of error occuring line.

string chooseSyntaxChecker(string str)
chooseSyntaxChecker method checks the syntax of choose calls, if there is no error detected method returns 1 to be replaced in the previous string by that choose call in order to indicate there exists no error. Note that replacing the whole choose statement with 1 does not have any effect on evaluating output since here we are only checking the syntax of the line. In the case of nested chooses, it works recursively until every choose is checked.

int choose(string text)
choose method is called while an input line makes a choose call. It replaces the choose call in that line by the variable name of the result of that choose call until there is no choose call left. Note that the variable name mentioned is of the form " 'datdiri3datdat1daaatdat'+i " where i is an integer. We picked that dummy name in order to prevent the conflicts with the input.

bool commacheck(string str)
commacheck method checks if the choose call has exactly four expressions as parameter.

3. Conclusion

After trying the code with the given testcases and more additional testcases that we have written, we are of the sense that the translator works well. The .ll files succesfully generates the wanted console outputs when compiled with LLVM 3.3. If we had more time and more knowledge we would use regex when checking syntax errors. Currently we check it using a lot of if statements and it decreases the readibility and overall beauty of the code. At the beginning, we decided to do syntax check and evaluation seperately which leads us to read the input twice. Correcting this would also be an improvement to do if we had more time.

Ali Kaan Biber 2018400069

Yavuz Samet Topçuoğlu 2019400285