Kardelen Demiral 2018400027
Berfin Şimşek 2018400009

DOCUMENTATION

The problem was basically generating a translator to translate our language to the LLVM IR code. Since we are unfamiliar with low-level languages, it took some time to understand the syntax. After some research about compilers, it started to look less complex. We decided to write our code part by part to make it easier.

First of all, we print all the necessary lines like defining the main then we check all lines whether there is an error or not. There are 3 functions for error checking; “errorCatch”, “errorCatchForExpressions” and “IsValidVariableName”. First one checks whether while, if and print statements are written properly according to syntax. Second one checks the expressions and the last one checks the variable names. While checking, we also push our variables to a vector in order to allocate after checking. If there is an error, we print the line that has the error and finish the execution. In order to do that we also changed the print function declaration to write syntax errors. If not, we allocate the variables that we pushed to a vector and store 0 as their initial values.

After we get the lines one by one again in a while loop and determine the statement type, we print the necessary lines for each of them. For example, if it is a while loop or an if statement, we print the necessary lines for labeling. If it is while, if or print we took the string inside the paranthesis, if it is an assignment statement we took the right part of the “=” as a string and send to a function called “muko”. This function does all necessary things for all possible expressions. First, it calls the “infixToPostFix” function which separates every element of the expression and push them into a stack in the correct order of precedence. This function also consider “choose” as a separate element. If stack has only one element and it is not choose function, it returns the value of the element or the temporary value that it is loaded. If it is choose, it calls the “choose” function that makes the llvm program decide which expression should be returned according to the first expression and returned the temporary value as a string. We used llvm “select” operation here. If stack has more than one element, it means the expression includes aritmetic operations. So it takes the elements from the stack in twos and calls the “operation” function which basically prints the correct statements for arithmetic operations such as loading variables. After “muko” function returns, we print the other necessary lines according to the statement types. For instance, if it is a print statement, we print the llvm print function with the result that “muko” function returned. When this big while loop finishes, we print the return statement and this is the end of the code.

We had troubles when we were writing the choose function. First we tried to write like how we write the if statements but it gave error because we were loading a temporary variable to another temporary variable. Then we improved our program with llvm select operation which makes our lives easier and it solved the error. Another problem occurred in the allocation part. We were allocating the variables that at the left part of the assignment at the beginning then when we see a variable that is not allocated, we were allocating at that point. But we noticed that allocation in the while statements gives error so we changed our code to make all necessary allocation in another while at the beginning if there are not any error.

To criticise, our code is repetitive. For example, we have a function which deletes the spaces in a line and we used this function basically everywhere even when it is unnecessary. Also our code is quite long because of the repetitions. For example we used almost exactly the same code in our “choose” function when we divide choose function into parts according to the commas in the error checking function also. On the other hand, we considered every possible error in a line so we have very detailed

error checking functions. Also, we had a detailed research about llvm so we discovered the select operation which helped us handling the choose fuction.

If we had more time, we could write a more efficient program but since the time is limited we focused on just implementing the code properly. Additionally, we might have improved our code to be shorter and less repetitive.