Documentation of mylang2ir Project

Atilla Türkmen – 2019400216

First, the program tokenizes the input that comes from the Scanner line by line and puts them in an Array List. Separator tokens are parenthesis, four operators, equals sign, curly brackets, commas, '#' sign and white lines. All white lines are ignored when tokens are being put into the list.

Then a Parser instance is created to process the tokens and print LLVM commands. Four functions are called from this class. With the first function, starting LLVM lines are printed onto the output file. These lines write the module ID; declares print.str constant for printing numbers, error.str constant for printing syntax error, printf function; defines and opens the main function.

Second function looks for variables ignoring tokens after comment symbol. Variables are tokens that are not keywords nor separators. If a variable contains non-alphanumeric characters or starts with a number function doesn't initialize it, continues to other tokens. LLVM code that allocates and stores zero in these variables are printed onto the file. All variable names are stored with "r" letter at the end to prevent errors when a variable is named "t1".

Third function processes tokens line by line and produces LLVM output according to instructions. Firstly, all tokens after # sign is removed from the line. Then It is checked whether the opening and closing brackets are equal. If they are equal, there are four options: print statement, assignment, if statement or while statement. All of these types of lines contains one or more expressions.

"computeExpression" function handles all expressions. If an expression contains only one token, it is either a number or a variable. If it is a number, the function returns that number directly. If it is a variable that variable is loaded to a temporary variable and the name of that temporary variable is returned. Names of temporary variables are in the form of t+number. This number is incremented when a temporary variable is created. If there is more than one token in the expression, that is an operation. Before turning to post-fix form; negative numbers are made into one token, empty parenthesis are checked and if there is one or more choose functions LLVM code that computes and stores the result in a temporary variable is created. Choose functions are replaced with that temporary variable. Then expression is turned into post-fix form. Operations are done from left to right with a stack. Result is stored in a temporary variable and pushed in stack. The last temporary variable which contains the result of all operations is returned.

When a print statement line is encountered, the expression inside parenthesis is sent to computeExpression function. Then LLVM command to print the answer is produced. When an assignment statement is encountered if left side is an acceptable variable name, right side is sent to computeExpression function and returned data is stored in variable on left side.

When start of an if statement is encountered, expression inside the parenthesis is evaluated, LLVM code that compares it with zero is produced. Two labels, ifbody+number and ifend+number are created. Number is incremented with every if statement. After ifbody label, evaluateInsideCurlyBrackets function is called, passing the list that contains all lines and current line number. This function creates a new array list and puts all tokens in it until a closing curly bracket is reached. These tokens are removed from the original list. Then produceOutput function is called on these lines. Thus, output of lines inside curly brackets are printed in ifbody label. After this process, program continues translating lines after curly brackets because they are removed. Handling of while statements is similar. The only difference is that; after whilebody, program is sent back to whilecond which is while condition label.

After all lines are finished, if there were no syntax errors, ret statement and curly bracket of main function is printed.