LLVM Code Generator
Cmpe 230 Systems Programming – **HW1**

Muhammed Emin Güre

1. Problem and Solution

Program needs to convert a file that contains lines of expression with operators or not to an intermediate code LLVM as in project. There are lines for solving purpose and printing purpose. Program also needs to checks for any syntax error which is not convenient for the grammar. All operators are binary operators and there is no unary minus sign.

Programs reads the input file line by line and with the split_tokens function, line is splitted to the tokens. Then line that is tokenized is converted from infix form to postfix form as to make the expression solving easy. Converted line is solved by solvePostfix function and when an operator comes to an action, a line of llvm code is generated according to operator. If any line does not contain any assignment operator then program checks for line contain any operator. If there is an operator, firstly the expression is solved and a line of llvm code is generated as the printing purpose; if there isn't any operator, then the token is printed as in llvm code.

Program also checks for syntax errors such as undefined variable, missing paranthesis, extra operator, invalid characters, etc.

2. Software Architecture

Program is needed to be ran as the formar ./stm2ir [input-file]. Output file will be written in the input file's path and it will be named as input file's name with replacing the file type to .ll extension. The project is tested with llvm-3.5-runtime.

```
/**
 * Checks if c is a sign
 * @param c
 * @return  0 if c is ( or ). Returns 1 if c is +, -, *, /, =.
 */
int sign(char c);

/**
 * Splits the string str to tokens with checking signs between variables and
numbers and checks for syntax errors
 * @param str
 * @return a queue that contains tokens
 */
queue<string> split_tokens(string str);

/**
 * Returns a value to determine the precedence of the char a
 * @param a
 * @return -1 if a is +, 0 if a is -, 1 if a is * or /, 2 if a is (, 3 if a
is ).
 */
int get_precedence(char a);

/**
 * Solves the given postfix using stack
 * @param expr      queue that will be solved
 * @param line_num  to keep track the line number as throw error
 * @return          result of the postfix
 */
int solvePostfix(queue<string> expr, int line_num);

/**
 * @param expr  a queue of strings to be converted to postfix form
 * @return      a queue that contains the strings as infix form
 */
queue<string> infixToPostfix(queue<string> expr){
```