Compilers Project

Ahmed Mohamed Saad	1190184
Ahmed Magdy AbdelAziz	1190200
Mahmoud Khaled Mahmoud	1190141
Ziad Ezzat Sedki	1190032

Project Overview:

This project involves the development of a compiler consisting of a lexical analyzer (lexer) and a syntax analyzer (parser) for C++ programming language. The lexer is responsible for breaking down the source code into tokens, the smallest units of meaning. The parser then uses these tokens to construct a syntax tree or validate the syntactic structure of the code according to the language's grammar.
We check for the syntax and semantic errors in the users code Having a symbol table to store our variables and identifiers and Quadruples to generate a code near to assembly code.

Tools and Technologies Used

- Programming Language: C++
- Compiler: G++
- Libraries/Tools:
  - Flex: Fast Lexical Analyzer for generating the lexer
  - Bison: GNU parser generator for creating the parser
  - PyQt5 for making the GUI part

Tokens Used:

Token	Pattern	Description
PLUS	"+"	Represents the addition operator.
MINUS	"-"	Represents the subtraction operator.
MULT	"*"	Represents the multiplication operator.
DIV	"/"	Represents the division operator.

Token	Pattern	Description
MOD	`"%"`	Represents the modulus operator.
EXP	`"^"`	Represents the exponentiation operator.
ASSIGN	`"="`	Represents the assignment operator.
DIV_EQ	`"/="`	Represents the division assignment operator.
PLUS_EQ	`"+="`	Represents the addition assignment operator.
MINUS_EQ	`"-="`	Represents the subtraction assignment operator.
MULT_EQ	`"*="`	Represents the multiplication assignment operator.
INC	`"++"`	Represents the increment operator.
DEC	`"--"`	Represents the decrement operator.
INT_VAL	`[0-9]+`	Represents an integer literal.
FLOAT_VAL	`[0-9]+\.[0-9]+`	Represents a floating-point literal.
CHAR_VAL	`['][a-zA-Z0-9][']`	Represents a character literal.
STRING_VAL	`(["][^"\\]*(\\.[^"\\]*)*["])|(['][^'\\]*(\\.[^'\\]*)*['])`	Represents a string literal.
TRUE_VAL	`"true"`	Represents the boolean literal `true`.
FALSE_VAL	`"false"`	Represents the boolean literal `false`.
INT	`"int"`	Represents the integer type keyword.
FLOAT	`"float"`	Represents the float type keyword.
CHAR	`"char"`	Represents the char type keyword.
BOOL	`"bool"`	Represents the boolean type keyword.
STRING	`"string"`	Represents the string type keyword.

Token	Pattern	Description
CONST	`"const"`	Represents the constant keyword.
WHILE	`"while"`	Represents the `while` loop keyword.
BREAK	`"break"`	Represents the `break` keyword.
CONTINUE	`"continue"`	Represents the `continue` keyword.
FOR	`"for"`	Represents the `for` loop keyword.
SWITCH	`"switch"`	Represents the `switch` keyword.
CASE	`"case"`	Represents the `case` keyword.
DEFAULT	`"default"`	Represents the `default` keyword.
IF	`"if"`	Represents the `if` keyword.
ELSE	`"else"`	Represents the `else` keyword.
EQUAL	`"=="`	Represents the equality operator.
NE	`"!="`	Represents the not equal operator.
GE	`">="`	Represents the greater than or equal to operator.
LE	`"<="`	Represents the less than or equal to operator.
AND	`"&&"`	Represents the logical AND operator.
OR	`"||"`	Represents the logical OR operator.
GREATER	`">"`	Represents the greater than operator.
LESS	`"<"`	Represents the less than operator.
NOT	`"!"`	Represents the logical NOT operator.
LBRACE	`"{"`	Represents the left brace `{`.
RBRACE	`"}"`	Represents the right brace `}`.
LPAREN	`"("`	Represents the left parenthesis `(`.
RPAREN	`")"`	Represents the right parenthesis `)`.
SEMICOLON	`";"`	Represents the semicolon `;`.
COMMA	`","`	Represents the comma `,`.

Token	Pattern	Description
COLON	`":"`	Represents the colon `:`.
REPEAT	`"repeat"`	Represents the `repeat` keyword.
UNTIL	`"until"`	Represents the `until` keyword.
VOID	`"void"`	Represents the `void` keyword.
RETURN	`"return"`	Represents the `return` keyword.
PRINT	`"print"`	Represents the `print` keyword.
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9_]*`	Represents an identifier (variable name).
NEWLINE	`"\n"`	Represents a newline character.
COMMENT	`[//].*`	Represents a comment.

QUADS Used:

Quadruple Operation	Description
ADD R0 R1 R0	Addition R0 and R1 Contents and save in R0
ALLOC x R1	Memory allocation of variable x in R1
AND T1 T2 T3	Logical AND operation between T1 , T2 to be saved in T3
ASSIGN 5 R6	Assignment operation of value 5 to register R6
CALL F11	Function call
DEC R0 R1	Decrement operation of R0 and saved in R1

DIV R1 R2 R1	Division operation of values stored in R1 and R2 and the result will be stored in R1
EQ_EQ T1 R0 T1	Equality check Between
EXP R0 R1 R1	Exponentiation operation R0 ^ R1 and result will be stored in R1
GE R5 R6 T5	Greater than or equal to check if R5 >= R6 and result is stored in T5
GREATER R5 R6 T5	Greater than or equal to check if R5 > R6 and result is stored in T5
INC R0 R1	Increment operation of result in R0 and store result in R1
JF T1 L0	Jump if T1 is false to label L0
JMP L1	Unconditional jump to label L1
JT T3 L5	Jump if T3 is true to L5
LE R0 R1 T4	Check if R0 <= R1 and result is stored in R4
LESS T1 T2 T5	Check if **T1** < **T2** and result is stored in **T5**
MOD R0 R1 R0	Modulus operation R0 = (R0%R1)
MUL R0 R1 R0	Multiplication operation R0 = R0*R1
NEG R0 R1	Negation operation R1 = -R0

NOT R0 R1	Logical NOT operation R1 = !R0
NOT_EQ R0 R1 T0	Not equal check T0 = (R0 != R1)
OR R0 R1 R2	Logical OR operation R2 = (R0 | R1)
RET	Return from function func
SUB T0 T1 R0	Subtraction operation R0 = T0 - T1

Workload:

Ahmed Mohamed Saad : Parser.y , Quadruples and CodeGeneration , GUI

Ziad Ezzat Sedki : Parser.y , Quadruples and CodeGeneration , GUI

Ahmed Magdy : Lexer.l SemanticAnalysis SymbolTable

Mahmoud Khaled : Lexer.l SemanticAnalysis SymbolTable