
Cairo University

Faculty of Engineering

Computer Engineering Department

Fourth year

# Team #17

| Name | Section | B.N. |
|---|---|---|
| Nada Osman | 2 | 30 |
| Shredan Abdullah | 1 | 33 |
| Aya Ahmed | 1 | 14 |
| Salma Ragab | 1 | 30 |

# Project Overview:

The primary objective of this project is to design and implement a compiler for a programming language that is inspired by C++.

# Language Features:

- Variables and Constants.
- Basic data types (int, float, char, bool,string).
- Mathematical (+,-,*,/,%) and logical expressions(!,||,&&).
- Block structure (nested scopes).
- Control structures (if-else, while-loops, for-loops, Do-while loops, switch-cases).
- Functions.

# Tools and Technologies Used:

## ➢ Lex:

Lex is a lexical analyzer generator used to identify tokens in the source code. It helps in defining regular expressions for token patterns.

## ➢ Yacc:

Yacc (Yet Another Compiler-Compiler) is used to generate a parser that can process the tokens generated by Lex. It helps in defining the grammar and syntax rules for the language

# How to run the project:

- Run **./build.sh** script in the terminal.

# Tokens

| **Token** | **Regex** | **Description** |
|---|---|---|
| INT | [0-9]+ | Numbers from 0 to infinity. |

| | | |
|---|---|---|
| STRING | \"[^\n\"]*\" | Any string between double quotes. |
| CHAR | ['][a-zA-Z]['] | Any character between single quotes. |
| FLOAT | [0-9]+\.[0-9]+ | Any floating point number. |
| - | \n | New line. |
| - | \/\/[^\n]* | Single line comment. |
| - | \/*([^*]\|*[^\/])**\/ | Multi-line comment. |
| CONST | [A-Z]+ | Constant variable name |
| Mathematical-op | [/+*%-=] | Mathematical operators (+, -, *, /, %,=). |
| Logical-op | [&& \|\| !] | Logical operators (AND, OR, NOT). |
| Comparison-ineq | [>==<=!=] | Comparison inequalities [<=,>=, ==, =, >, <] |
| IDENTIFIER | [a-zA-Z_][a-zA-Z0-9_]* | Variable names |
| punctuators | [(){}:,] | Language punctuators. |
| BOOL | true | True value. |
| BOOL | false | False value. |

# Quadruples:

| **Quadruples** | **Description** |
|---|---|
| FUNC X | Start of a function, X is the function name |
| ENDFUNC X | End of a function, X is the function name |
| CALL X | Calls a function, X is the function name |
| RET | Return from a function. |
| PUSH X | Push to the stack, X is the identifier or expression |
| POP X | Pop from the stack, X is the identifier or expression |
| JMP Label | Unconditional jump to the label |
| JF Label | Jump to the label if the result of the last operation is false |
| NEG | Get the opposite sign of an expression |
| NOT | not of an expression |

| | |
|---|---|
| ADD | Add two numbers |
| SUB | Subtract two numbers |
| MUL | Multiply two numbers |
| DIV | Divide two numbers |
| MOD | Modulus two numbers |
| OR | Oring of two numbers |
| AD | Anding of two numbers |
| EQ | Check if two numbers are equal |
| NEQ | Check if two numbers are not equal |
| GT | Check if the first number is greater than the second |
| GEQ | Check if the first number is greater than or equal the second |
| LT | Check if the first number is less than the second |
| LEQ | Check if the first number is less than or equal the second |

# Production Rules:

- program:program statement | ;

- statement:
  declaration
  | function_declaration
  | assignment
  | return_ SEMICOLON
  | expression SEMICOLON
  | if_
  | while_
  | for_
  | do_while
  | switch_
  | block
  | BREAK SEMICOLON
  ;

- data_type:
  INT_TYPE
  | FLOAT_TYPE
  | BOOL_TYPE
  | CHARACTER_TYPE
  | STRING_TYPE
  | VOID
  ;

- declaration:
  data_type IDENTIFIER ASSIGNMENT expression SEMICOLON
  | data_type IDENTIFIER SEMICOLON
  |data_type CONST ASSIGNMENT expression SEMICOLON
  ;

- return_: RETURN | RETURN expression;

- switch_header: SWITCH IDENTIFIER COLON '{' ;

- switch_: switch_header CASES '}';

- default_case: DEFAULT COLON block ;

- CASES:CASE expression CASES | default_case;

- function_declaration: data_type IDENTIFIER function_arg_part block ;

- function_arg_part : '(' arguments ')' | '(' ')' ;

- arguments: arguments_declaration COMMA arguments | arguments_declaration;

- arguments_declaration: data_type IDENTIFIER ;

- if_header:IF expression ;

- if_body: COLON block ;

- if_: if_header if_body | if_header if_body ELSE block ;

- while_:
  WHILE expression COLON block ;

- do_while:
  DO block WHILE '(' expression ')' SEMICOLON ;

- for_:FOR '(' statement expression SEMICOLON assignment ')' {sharedData.loop=0;}
  block;

- assignment:IDENTIFIER ASSIGNMENT expression SEMICOLON
  |CONST ASSIGNMENT expression SEMICOLON ;

- block: '{' program '}' ;

- function_call: IDENTIFIER '( function_call_arguments ')' ;

- function_call_arguments: expression COMMA function_call_arguments | expression | ;

- expression:
  IDENTIFIER
  | INT
  | FLOAT
  | BOOL
  | CHARACTER
  | STRING
  | CONST
  | SUB expression
  | expression ADD expression
  | expression MOD expression
  | expression SUB expression
  | expression MUL expression
  | expression DIV expression
  |logics
  | function_call
  | '(' expression ')'
  ;

- logics:
  expression GREATER expression
  | expression LESS expression
  | expression LESS ASSIGNMENT expression
  | expression GREATER ASSIGNMENT expression
  | expression EQUAL_TO expression
  | expression NOT_EQUAL expression
  | expression AND expression
  | expression OR expression
  | NOT expression
  ;