Compilers Project

Simple Programming Language using Lex and Yacc

JUNE 2021

Team Members

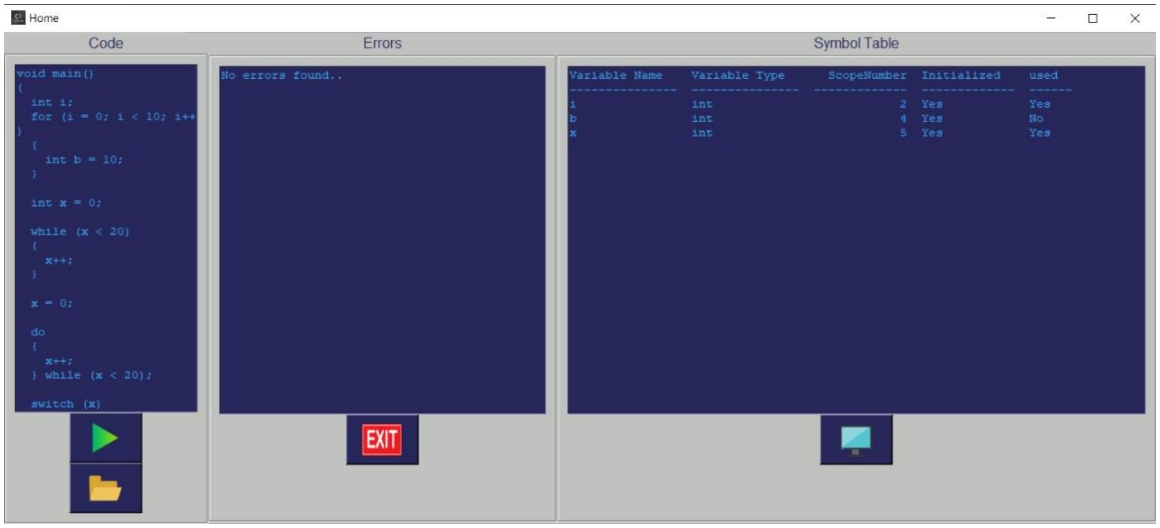
Aya Sameh Mohamed	1170107
Maram Mohamed Hosni	1170533
Heba Ali Hassan	1170134
Karim Wael Zain	1170058

Project Overview

The project implements a simple C-like programming language.

The language covers the following:

- Assignment statement
- If-Variables and constants declaration
- Mathematical and logical expressions
- then-else statement, while loops, repeat-until loops, for loops, switch statement
- Block structure
- Functions



Code compiled with no errors

Code compiled with errors

Tools and Technologies

- **Lex - A Lexical Analyzer Generator**

  Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream. It is well suited for editor-script type transformations and for segmenting input in preparation for a parsing routine.

- **Yacc - Yet Another Compiler-Compiler**

  Yacc provides a general tool for describing the input to a computer program. The user specifies the structures of his input, together with code to be invoked as each such structure is recognized. Yacc turns such a specification into a subroutine that handles the input process.

- **Tkinter**

  Tkinter is the standard GUI library for Python. It provides a fast and easy way to create GUI applications.

Tokens

Token	Description
TYPE_INT	To identify an integer type variable/identifier
TYPE_FLOAT	To identify a float type variable/identifier
TYPE_CHAR	To identify a char type variable/identifier
TYPE_BOOL	To identify a boolean type variable/identifier
TYPE_VOID	Void keyword for the main function
IDENTIFIER	Represents a variable's name or function's name
INTEGER	To identify an integer value assigned to variable/identifier
FLOAT	To identify a float value assigned to variable/identifier
CHAR	To identify a character value assigned to variable/identifier

BOOL	To identify a boolean value assigned to variable/identifier
CONST	To define constants
IF	If keyword for if-else statement
ELSE	Else keyword for if-else statement
SWITCH	Switch keyword for switch-case statement
CASE	Case keyword for switch-case statement
DEFAULT	Default keyword for switch-case statement
FOR	For keyword for for-loop statement
DO	Do keyword for do-while statement
WHILE	While keyword for do-while statement
BREAK	Break keyword for switch-case statement
RETURN	Return keyword for returning from a function
INC	Increment by 1 (++)
DEC	Decrement by 1 (--)
EQUAL	Equality check (==)
NOT_EQUAL	Inequality check (!=)
GREATER_EQUAL	Greater than or equal for comparing variables (>=)
LESS_EQUAL	Less than or equal for comparing variables (<=)
LOGICAL_AND	Logical and for comparison

LOGICAL_OR	Logical or for comparison
ASSIGN	Assignment operator for assigning variables
SEMICOLON	Semicolon for the end of most statements (;)
COMA	Comma for separating variables (,)
MAIN	Main keyword for the main function
FUNC	Function keyword for declaring a function
CALL_FUNC	Call function for calling a function
PLUS	Plus operation (+)
MINUS	Minus operation (-)
MULT	Multiplication operation (*)
DIVI	Division operation (/)
LEFTPARE	Left round bracket
RIGHTPARE	Right round bracket
CURLEFT	Left round bracket
RIGHTPARE	Right round bracket
LESS	Less that for comparing variables (<)
GREATER	Greater than for comparing variables (>)
COLON	Colon for switch-case statement (:)

Language Production Rules

Rules
Program:mainStmt | functions mainStmt
Functions: FUNC function | functions FUNC function
mainStmt: TYPE_VOID Main LEFTPARE RIGHTPARE stmt_block
stmts: stmt | stmts stmt | stmt_block | stmts stmt_block
stmt_block: CURLEFT CURRIGHT | CURLEFT stmts CURRIGHT
Stmt: varDeclarationStmt SEMICOLON | postPrefixExpr SEMICOLON | ifStmt | forStmt | whileStmt | switchstmt | dowhileStmt | CALL_FUNC functionCall SEMICOLON
varDeclarationStmt: variableDecl | multiVariableDecl | variableAssign
varType: TYPE_INT | TYPE_FLOAT | TYPE_CHAR | TYPE_BOOL
dataType: BOOL | INTEGER | FLOAT| CHAR
variableDecl: varType IDENTIFIER | varType IDENTIFIER ASSIGN expression | CONST varType IDENTIFIER ASSIGN expression | varType IDENTIFIER ASSIGN CALL_FUNC functionCall | CONST varType IDENTIFIER ASSIGN CALL_FUNC functionCall
multiVariableDecl: variableDecl COMA IDENTIFIER | variableDecl COMA IDENTIFIER ASSIGN expression | multiVariableDecl COMA IDENTIFIER | multiVariableDecl COMA IDENTIFIER ASSIGN expr
variableAssign: IDENTIFIER ASSIGN variableAssignTypes

variableAssignTypes: expression | CALL_FUNC functionCall;
expression: logicExpr
mathExpr: mathExpr PLUS mathExpr | mathExpr MINUS mathExpr | mathExpr MULT mathExpr | mathExpr DIVI mathExpr | expr
relationExpr: relationExpr LESS relationExpr | relationExpr GREATER relationExpr | relationExpr EQUAL relationExpr | relationExpr NOT_EQUAL relationExpr | relationExpr GREATER_EQUAL relationExpr | relationExpr LESS_EQUAL relationExpr | mathExpr | NOT mathExpr
logicExpr: logicExpr LOGICAL_AND logicExpr | logicExpr LOGICAL_OR logicExpr | relationExpr
Expr: dataType | IDENTIFIER
postPrefixExpr: IDENTIFIER INC | INC IDENTIFIER | IDENTIFIER DEC | DEC IDENTIFIER
ifStmt: ifstmt_start else_if optional_else | ifstmt_start optional_else
Ifstmt_start: IF LEFTPARE logicExpr RIGHTPARE body
Else_if: else_if ELSE IF LEFTPARE logicExpr RIGHTPARE body | ELSE IF LEFTPARE logicExpr RIGHTPARE body
optional_else: ELSE body | /*empty*/
Body: stmt_block
switchstmt: SWITCH LEFTPARE IDENTIFIER RIGHTPARE CURLEFT casestmt CURRIGHT
Casestmt: CASE expr COLON stmts BREAK SEMICOLON | DEFAULT COLON stmts BREAK SEMICOLON | DEFAULT COLON BREAK SEMICOLON | /*empty*/

forStmt: FOR LEFTPARE initStmt logicExpr SEMICOLON loopCond RIGHTPARE stmt_block
initStmt: variableAssign SEMICOLON | SEMICOLON
loopCond: postPrefixExpr | IDENTIFIER ASSIGN mathExpr
whileStmt: WHILE LEFTPARE logicExpr RIGHTPARE stmt_block
dowhileStmt: DO stmt_block WHILE LEFTPARE expression RIGHTPARE SEMICOLON
functionCall: IDENTIFIER LEFTPARE functionArgumentsPassed RIGHTPARE
functionArgumentsPassed: expression | functionArgumentsPassed COMA expression
function: functionHeader functionBody
functionHeader: varType IDENTIFIER LEFTPARE functionArgumentsDecl RIGHTPARE
functionArgumentsDecl: varType IDENTIFIER | functionArgumentsDecl COMA varType IDENTIFIER
functionBody: CURLEFT stmts returnStmt CURRIGHT | CURLEFT returnStmt CURRIGHT
returnStmt: RETURN expression SEMICOLON | RETURN functionCall SEMICOLON

Quadruples

Token	Description
MOV Rd, Rs	`Rd ← Rs`
ADD Rd, Rs1, Rs2	`Rd ← Rs1 + Rs2`
SUB Rd, Rs1, Rs2	`Rd ← Rs1 - Rs2`
MUL Rd, Rs1, Rs2	`Rd ← Rs1 * Rs2`
DIV Rd, Rs1, Rs2	`Rd ← Rs1 / Rs2`
CMPL Rd, Rs1, Rs2	`Rd ←  Rs1 < Rs2`
CMPLE Rd, Rs1, Rs2	`Rd ←  Rs1 <= Rs2`
CMPG Rd, Rs1, Rs2	`Rd ←  Rs1 > Rs2`
CMPGE Rd, Rs1, Rs2	`Rd ←  Rs1 >= Rs2`
CMPE Rd, Rs1, Rs2	`Rd ←  Rs1 == Rs2`
CMPNE Rd, Rs1, Rs2	`Rd ←  Rs1 !=  Rs2`
AND Rd, Rs1, Rs2	`Rd ← Rs1 && Rs2`
OR Rd, Rs1, Rs2	`Rd ← Rs1 || Rs2`
INC Rx	`Rx ← Rx + 1`
DEC Rx	`Rx ← Rx - 1`
JF Rx, label	Jump to label if Rx if false
JT Rx, label	Jump to label if Rx if true
JMP label	Jump to label
CALL functionName	Used to call a function and it makes the required context switching

RETURN	To return from a function back to the address that is determined by CALL when the function was called

Test Cases Modifications

1. semantic-error.cpp
    - The language does not support strings. So, it was replaced with a character.
2. loops.cpp
    - Semicolons were added.
    - The counter of the for loop (i) should be initialized before the for statement.
    - The for loop sequence is doing the check then the loop condition and finally the loop.
3. function.cpp
    - Modified the method that a function is declared or called.