Cairo University
Faculty of Engineering
Credit Hours System

Compilers Project Report

Submitted To: ***Eng. Lydia Wahid***

Eng. Sandra Wahid

Name:	*Amira Ahmed Amer*	**1142324**
	Dina Adib Fouad	**1142007**
	Salma Hanafy	**1142049**
	Rana Amr Afifi	**142046**
Date:	**13-05-2018**	

1. Project Overview:

The aim of the project is to implement a compiler for a simple C++ like language. This language covers

- Variables and Constants declaration.
- Mathematical and logical expressions.
- Assignment statement.
- If-then-else statement, while loops, repeat-until loops, for loops, switch statement.

Project also has a simple IDE for the language.

2. Tools and Technologies Used:

a. Lex

Lex helps write programs whose control flow is directed by instances of regular expressions in the input stream. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions. The Lex source file associates the regular expressions and the program fragments. As each expression appears in the input to the program written by Lex, the corresponding fragment is executed.

b. YACC

Yacc provides a general tool for describing the input to a computer program. The Yacc 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; frequently, it is convenient and appropriate to have most of the flow of control in the user's application handled by this subroutine.Yacc is written in portable C. The class of specifications accepted is a very general one: LALR (1) grammars with disambiguating rules.

c. C# Windows Forms

C# was used to implement the IDE's GUI.

3. Tokens and their description

Token	Description
IF	If for if statement
ELSE	Else for if statement
DO	Do for do while statement
WHILE	While for while loops and do while
FOR	For loop statement
SWITCH	Switch for switch case statement
CASE	Case for switch case statement
DEFAULT	Default for switch case statement
BREAK	Break for switch case statement

CONST	Const keyword to define constants
INT	Int keyword to define integer variables
FLOAT	Float keyword to define float variables
CHAR	char keyword to define character variables
BOOL	Bool keyword to define boolean variables
TR	True
FL	False
GE	Greater than or equal comparison operator (<=)
LE	Larger than or equal comparison operator (>=)
EQ	Equal to comparison operator (==)
NE	Not equal to comparison operator (!=)
INC	Increment (++)
DEC	Decrement (--)
PLUSEQ	+= assignment operator
MINEQ	-= assignment operator
DIVEQ	/= assignment operator
MULEQ	*= assignment operator
OR	Logical OR used in comparison (||)
AND	Logical AND used in comparison (&&)
SEMI	Semicolon for ending statements
NEG	Minus (-) operator
PLUS	Plus (+) operator
MULT	Multiply (*) operator
DIV	Divide (/) operator
OPENBRAC	Open round bracket
CLOSEBRAC	Closed round bracket
LESS	Less than comparison operator (>)
GREAT	Greater than comparison operator (<)
NOT	Logical NOT used in comparison (!)
EQU	Assignment (=) operator
CURLOPBR	Open curly braces
CURLCLBR	Closed curly braces
COLON	Colon (:) used with case and default
CHARACTER	Character value assigned to a variable or constant
INTEGER	Integer value assigned to a variable or constant
FLOT	Decimal value assigned to a variable or constant
VARIABLE	Variables' name

4. Language Production Rules:

program: function

function: function stmt | epsilon

stmt: SEMI |post_prefix SEMI |CURLOPBR stmt_list CURLCLBR |decleration |assign_stmt |loop |condition

loop: DO stmt WHILE OPENBRAC logi_expr CLOSEBRAC SEMI |WHILE OPENBRAC logi_expr CLOSEBRAC stmt |FOR OPENBRAC init_cond logi_expr SEMI loop_exp CLOSEBRAC stmt

condition: SWITCH OPENBRAC VARIABLE CLOSEBRAC CURLOPBR case_stmt CURLCLBR | IF OPENBRAC logi_expr CLOSEBRAC stmt | IF OPENBRAC logi_expr CLOSEBRAC stmt ELSE stmt

stmt_list: stmt_list stmt |epsilon

case_stmt: CASE case_exp COLON stmt_list BREAK SEMI case_stmt |DEFAULT COLON stmt_list BREAK SEMI |epsilon

case_exp: VARIABLE |CHARACTER|INTEGER |FLOT |TR |FL

decleration: type VARIABLE SEMI |CONST type VARIABLE EQU val SEMI

assign_stmt: VARIABLE ass_op val SEMI |type VARIABLE EQU val SEMI

val:post_prefix |logi_expr

ass_op: EQU |DIVEQ |MULEQ |MINEQ |PLUSEQ

post_prefix: VARIABLE INC |VARIABLE DEC |INC VARIABLE |DEC VARIABLE

math_expr: math_expr PLUS math_expr |math_expr NEG math_expr |math_expr MULT math_expr |math_expr DIV math_expr|term

term: INTEGER|VARIABLE |NEG math_expr |FLOT|OPENBRAC logi_expr CLOSEBRAC |CHARACTER |STR

logi_expr:logi_expr OR and_expr|and_expr

and_expr: and_expr AND equ_expr|equ_expr

equ_expr: equ_expr EQ rel_stmt |equ_expr NE rel_stmt|rel_stmt

rel_stmt: rel_stmt GREAT not_expr |rel_stmt LESS not_expr |rel_stmt GE not_expr|rel_stmt LE not_expr |TR |FL |not_expr

not_expr: NOT math_expr |math_expr

init_cond: assign_stmt |SEMI

loop_exp: post_prefix |VARIABLE ass_op val |epsilon

type: INT |BOOL |CHAR |FLOAT |STRING

5. Quadruples:

Quadruples	Description
MOV Rx, Ry	Rx = Ry

ADD R1, R2, R3	R1 = R2 + R3
SUB R1, R2, R3	R1 = R2 - R3
MUL R1, R2, R3	R1 = R2 * R3
DIV R1, R2, R3	R1 = R2 / R3
AND R1, R2, R3	R1 = R2 && R3
OR R1, R2, R3	R1 = R2 || R3
DIVEQ R1, R2	R1 /= R2
MULEQ R1, R2	R1 *= R2
MINEQ R1, R2	R1 -= R2
PLUSEQ R1, R2	R1+= R2
NOT R1	! R1
NEG R1	- R1
INC R1	R1 = R1+ 1
DEC R1	R1 = R1 - 1
CMPE R1, R2, R3	R1 = true if R2 == R3
CMPNE R1, R2, R3	R1 = true if R2!= R3
CMPGE R1, R2, R3	R1 = true if R2>= R3
CMPLE R1, R2, R3	R1 = true if R2 <= R3
CMPG R1, R2, R3	R1 = true if R2 > R3
CMPL R1, R2, R3	R1 = true if R2 < R3
CMPG R1, R2, R3	R1 = true if R2 > R3
JMP Label_X	Unconditional jump to Label_X
JT Rx, Label_X	Jump to Label_X if Rx is true
JF Rx, Label_X	Jump to Label_X if Rx is false