PROGRAMMING LANGUAGE

Compilers Project

SUBMITTED TO:
ENG. SARAH RASHAD
ENG. YOUSSEF GHATTAS
BY:
AHMAD MUHAMMAD AL-MAHDI
KAREEM MOHAMED EISSA
KARIM MAGDY AMER
WESAM ADEL ABDUL-RAHMAN

Project Overview:

A programming language that supports basic data types (int, bool, char, float) and supports also assignment statements and mathematical and logical expressions, it also provides control statements and conditional statements. All syntax is C-Like.

Tools and Technologies Used:

- Flex (Lexical Analysis)
- Bison (Syntax Parsing)
- GNU Compiler Collection (ANSI C Compiler)
- Windows (Operating System)
- C# WinForms (Graphical User Interface GUI)

List of tokens:

Token	Description
INT	Integer data type
FLOAT	Floating point data type
BOOL	Boolean data type
CHAR	Character data type
CONST	Constant data type
FOR	For loop
WHILE	While loop
SWITCH	Switch keyword
CASE	Case (option in a switch statement)
DO	Do keyword
IF	If keyword (conditional)
DEFAULT	Default (default case in switch statement)
BREAK	Break keyword
ELSE	Else keyword
TRUE	True keyword (not equal to zero)
FALSE	False keyword (equal to zero)
MAIN	Main function
EQ_OP	==
RIGHT_OP	Shift right
LEFT_OP	Shift left
GE_OP	Greater than or equal
LE_OP	Less than or equal
AND_OP	Logical and
OR_OP	Logical or
XOR_OP	Exclusive or
CHAR_VAL	A char (example: ‘a’)
Identifier	An identifier
f_number	Floating point number (example: 1.2)
Number	Integer (example: 1, 2, 3)
[-+=;)(/*{}:><!]	Self-explained

Production Rules:

1) **factor** => number | f_number | identifier | TRUE | FALSE | CHAR_VAL | unary_exp | '(' expression ')';

2) **unary_exp** => '!' factor | '-' factor;
3) **mul_exp** => factor | mul_exp '/' factor | mul_exp '*' factor;
4) **add_exp** => mul_exp | add_exp '+' mul_exp | add_exp '-' mul_exp;
5) **shift_exp** => add_exp | shift_exp LEFT_OP | shift_exp RIGHT_OP add_exp;
6) **relational_exp** => shift_exp | relational_exp '<' shift_exp | relational_exp '>' shift_exp | relational_exp LE_OP shift_exp | relational_exp GE_OP shift_exp;
7) **equality_exp** => relational_exp | equality_exp EQ_OP relational_exp | equality_exp NE_OP relational_exp;
8) **xor_exp** => equality_exp | xor_exp XOR_OP equality_exp;
9) **and_exp** => xor_exp |and_exp AND_OP xor_exp;
10) **or_exp** => and_exp | or_exp OR_OP and_exp;
11) **conditional_exp** => or_exp;
12) **constant_expression** => conditional_exp;
13) **expression** => assignment;
14) **declaration** => INT identifier '=' assignment ';' | FLOAT identifier '=' assignment ';' | INT identifier ';' | FLOAT identifier ';' | BOOL identifier '=' assignment ';' | BOOL identifier   ';' | CHAR identifier '=' assignment ';' | CHAR identifier ';' ;
15) **constant_declaration** => CONST INT identifier '=' assignment ';' | CONST FLOAT identifier '=' assignment ';' | CONST CHAR identifier '=' assignment ';' | CONST BOOL identifier '=' assignment ';';
16) **statement** => labeled_statement | compound_statement | expression_statement | selection_statement | iteration_statement | declaration | constant_declaration | BREAK ';' | error ';' | error '}' | error_statement;
17) **labeled_statement** => CASE constant_expression ':' statement | DEFAULT ':' statement;
18) **compound_statement** => '{' '}' | '{' statement_list '}';
19) **statement_list** => statement | statement_list statement;
20) **expression_statement** => ';' | expression ';';
21) **selection_statement** => IF '(' expression ')' compound_statement | IF '(' expression ')' compound_statement ELSE compound_statement | SWITCH '(' expression ')' compound_statement;
22) **iteration_statement** => WHILE '(' expression ')' compound_statement | DO compound_statement WHILE '(' expression ')' ';' | FOR '(' expression_statement expression_statement ')' compound_statement | FOR '(' expression_statement expression_statement expression ')' compound_statement;
23) **error_statement** => WHILE expression ')' compound_statement | DO compound_statement WHILE expression ')' ';' | FOR expression_statement expression_statement ')' compound_statement | FOR expression_statement expression_statement expression ')' compound_statement | IF expression ')' compound_statement | IF expression ')' compound_statement ELSE compound_statement | SWITCH expression ')' compound_statement | expression | INT identifier assignment ';' | FLOAT identifier assignment ';' | BOOL identifier assignment ';' | CHAR identifier assignment ';';

List of quadruples:

Quadruple	Description
add Rx, a, b	Rx = a + b
sub Rx, a, b	Rx = a – b
mul Rx, a, b	Rx = a * b
div Rx, a, b	Rx = a / b
stor X, a	X = a
SHL Rx, a, b	Rx = a << b
SHR Rx, a, b	Rx = a >> b
GT Rx, a, b	Rx = true if (a > b)
LT Rx, a, b	Rx = true if (a < b)
GTE Rx, a, b	Rx = true if (a >= b)
LTE Rx, a, b	Rx = true if (a <= b)
EQ Rx, a, b	Rx = true if (a = b)
NE Rx, a, b	Rx = true if (a != b)
xor Rx, a, b	Rx = a xor b
and Rx, a, b	Rx = a and b
or Rx, a, b	Rx = a or b
mov Rx, a	Rx = a
Define INT x	int x;
Define BOOL x	bool x;
Define FLOAT x	float x;
Define CHAR x	char x;
JMP L	Unconditional jump to label L
JZ Rx, L	jump to L if Rx = zero
JNZ Rx, L	jump to L if Rx != zero