

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 |