Compilers Project

Submitted to:
Eng. Mahmoud Khaled Ali

Submitted by:
Team 2

Ahmed Essam Eldin	SEC: 1	BN: 2
Philopateer Nabil Atia	SEC: 2	BN: 4
Mazen Amr Fawzy	SEC: 2	BN: 8
Mahmoud Ahmed Sebak	SEC: 2	BN: 18

Project Overview

The Project is a simple compiler for a language similar to C++ Language.

Supported features:

1. Data types: **int**, **char**, **bool** and **double.**
2. Single character Variable declaration e.g.:
    a. int a';
3. Constant declaration e.g.:
    a. Const int a = 10;
4. Mathematical expressions (+, -, *, /, %) e.g.:
    a. x = x + 1;
5. Logical expressions (>, <, <=, >=, ==, &&, ||) e.g.:
    a. bool x = a && b;
6. Assignment e.g.:
    a. x = 10; or char c = 'a';
7. If-else statement e.g.:
    a.
```
if(a===1)
{
        x = 1;
}
else
{
        x = 2;
}
```
8. While loop e.g.:
    a.
```
while(x>=1)
{
        x = x+1;
}
```
9. Do-while loop e.g.:
    a.
```
do
{
        x=x+1;
}while(x>=1);
```
10. For loop e.g.:
    a.
```
for(i=0;i<n;i=i+1)
{
        x=x+1;
}
```

11. Block structure e.g.:

```
{
        int x=1;
        {
                int y;
        }
}
```

12. Switch case e.g.:

```
switch(a)
{
        case 1:
                x=1;
                break;
        case 2:
                x=2;
                break;
        default:
                x=2;
}
```

13. Function e.g.:

```
int m(int a, int b)
{
        int x=a+b;
        return x;
}
```

14. Syntax and semantic errors.

Tools and Technologies used

In this project we used Flex for building the lexical analyzer and Bison for building the grammar and the syntax analyzer.

Tokens

Token	Description
[0-9]+	Positive integer number
(0|([1-9][0-9]*))(\.[0-9]+)?	Positive double number
\'.\'	Character
true|false	Boolean values
[-+()=/*,:;<>{}]	Allowed symbols
">=", "<=", "==", "!=", "&&", "||"	Logical Operators
"int", "char", "double", "bool", "const"	Data types
"If", "else", "while", "do", "for", "switch", "case", "break", "default", "void", "return"	Reserved words
[_a-zA-Z]	Identifier
[\s\t]+	Whitespace

Main Production Rules

Production rule	Description
data_type IDENTIFIER '=' expr ';'	Declaration with assignment
IDENTIFIER '=' expr ';'	Assignment
data_type IDENTIFIER ';'	Declaration without assignment
CONST data_type IDENTIFIER '=' expr ';'	Constant declaration
IF '(' expr ')' open_bracket stmt closed_bracket ELSE open_bracket stmt closed_bracket	If-else statement
IF '(' expr ')' open_bracket stmt closed_bracket	If statement
WHILE '(' expr ')' open_bracket stmt closed_bracket	While Loop
DO open_bracket stmt closed_bracket WHILE '(' expr ')' ';'	Do while loop

FOR '(' for_statement ';' opt_expr ';' for_statement ')' open_bracket stmt closed_bracket	For loop
open_bracket stmt_list closed_bracket	Block structure
SWITCH '(' IDENTIFIER ')' open_bracket case_list default_stmt closed_bracket	Switch statement
RETURN opt_expr ';'	Return statement
INTEGER | '-' INTEGER | CHARACTER | BOOLEAN | DOUBLE_VALUE | '-' DOUBLE_VALUE | IDENTIFIER | expr '+' expr | expr '-' expr | expr '*' expr | expr '/' expr | expr '%' expr | expr AND expr | expr OR expr | expr '>' expr | expr '<' expr | expr LE expr | expr GE expr | expr EQ expr | expr NE expr | '(' expr ')'	Expressions
CASE '(' INTEGER ')' ':' stmt_list BREAK ';' | CASE '(' DOUBLE_VALUE ')' ':' stmt_list BREAK ';' | CASE '(' CHARACTER ')' ':' stmt_list BREAK ';' | CASE '(' INTEGER ')' ':' stmt_list | CASE '(' DOUBLE_VALUE ')' ':' stmt_list | CASE '(' CHARACTER ')' ':' stmt_list	Case statement in switch case
VOID IDENTIFIER '(' paramter_list ')' open_bracket stmt_list closed_bracket | data_type IDENTIFIER '(' paramter_list ')' open_bracket stmt_list closed_bracket	Function statement

Quadruples

S1: last element in the stack and S2: the second last element in the stack.

Quadruple	**Description**
add<type>	S1 <- S1 + S2
sub<type>	S1 <- S1 - S2
mul<type>	S1 <- S1 * S2
div<type>	S1 <- S1 / S2
mod<type>	S1 <- S1 mod S2
neg<type>	S1 <- -S1
and<type>	S1 <- S1 AND S2
or<type>	S1 <- S1 OR S2
gt<type>	S1 <- S2 > S1 (boolean)
gte<type>	S1 <- S2 >= S1 (boolean)
lt<type>	S1 <- S2 < S1 (boolean)
lte<type>	S1 <- S2 <= S1 (boolean)
eq<type>	S1 <- S2 = S1 (boolean)
neq<type>	S1 <- S2 != S1 (boolean)
<type>_to_<type>	Convert S1 from the LHS type to the RHS type
push<type> <value>	Push <value> to the stack
pop<type> <value>	Pop S1 from stack and save it to <dest>
jmp<label>	Unconditional jump to the label
jnz<type> <label>	Jump to the label if S1 is not equal to zero
jz<type> <label>	Jump to the label if S1 is equal to zero
proc<ident>	Define a procedure
call<ident>	Call a procedure
ret	return from a procedure

Task Distribution

Member	work
Ahmed Essam Eldeen	- Building Parse tree (writing actions for the production rules) - Generating quadruples
Philopateer Nabil	- Designing and implementing symbol table (handling nested blocks) - Semantic analysis (handling semantic errors)
Mahmoud Ahmed Sebak	- Building Parse tree (writing actions for the production rules) - Generating quadruples
Mazen Amr	- Designing and implementing symbol table (handling nested blocks) - Semantic analysis (handling semantic errors)