PL/0 USER GUIDE

Christopher Donoso

Components of the PL/0 Compiler

The PL/0 Compiler is composed of three major parts: Lexical Analyzer, Parser, and the Virtual Machine. The lexical analyzer reads in the program and converts it into tokens that is read by the Parser. The parser checks these tokens for possible errors and create assembly code for the Virtual Machine. The Virtual Machine then acquires the code and executes the program.

How to Compile and Run the PL/0 Compiler

Before compiling and running the compiler, make sure to have the program file, header file, and input file in the same folder. To compile, open your terminal and direct it to the appropriate path and type: gcc compiler.c. To run, type: ./a.out.

To run with flags, you must set it us as or any variation of: ./a.out –l –v –a. The definition of each flag is as follows:

-l (Lexical Analyzer): prints the list of lexemes/tokens to the screen

-a (Parser): prints the generated assembly code to the screen

- v (Virtual Machine: prints the virtual machine execution trace to the screen.

How to Use the PL/0 Compiler Once it is Running

Once it is running, it will ask you to type the name of the input file. When doing so, make sure to type in the text extension, such as: source.txt. Based on what flags you set up, the text on the screen will vary, but the output will always show (if your program has an output.)

Understanding PL/0 Language

Block

The main of the PL/0 is the block and it contains 4 sections: constant declaration, variable declaration, procedure declaration, and statement.

Constant Declaration

A constant holds a value that cannot be changed. It is declared as:

```
const x = 1;
```

If multiple constants are declared, you would add commas between declaration:

```
const = x = 1, y = 2, z = 3;
```

Variable Declaration

When a variable is first declared, it is only declared with identifiers and it is not initialized. The program should initialize the variable later on. Therefore, it is declared as:

```
var x;
```

It multiple variables are declared, you would also add commas between declarations

```
var x, y, z;
```

Procedure Declaration

When declaring a procedure, it must be followed by a name:

```
procedure sub;
```

The procedure will contain its own begin and end and the variables utilized in between will only exist for that procedure and will not be used in main.

Statement

The statement is the core of the PL/0 and it is where the chunk of the program will be processed. It also creates the assembly code for the Virtual Machine.

If, Then, Else

If an “if” statement is called, it must include a “then” statement and it may include an else statement, but it is not necessary. With this statement, if a certain condition is met, then it will execute a certain instruction. An example of a program utilizing “if” and “then” statement:

```
var x;
begin
    if x > 10 then x := x* 5;
end.
```

In this program, if the variable x is greater than 10, its new value would be x * 5.

An example of the programing utilizing “else” statement:

```
var w;
begin
    if w > 10 then w := w * 5;
    else w := 21;
end.
```

In this program, if w is greater than 10, its new value would be w * 5. Otherwise, its value would be 21.

While Do

If a “while” statement is called, it must include a “do” statement. This statement is a loop and will execute the instructions inside the body until a certain condition is broken. An example of a program utilizing “while-do” statement:

```
var a;
begin
    a := 0;
    while a < 10 do a := a + 1;
end.
```

In this program, a is 0 and will increment itself by 1 until it is less than 10.

Call

Since call initiates a procedure, whenever it is called, it must be followed by the procedure name.

```
const c = 8;
var x;
procedure p;
    var y;
    begin
        y := c;
```

```
        x := x + y;
    end;
begin
    x := 5;
    call p;
    write x;
end.
```

In this program, procedure p is called after initializing x to 5. Therefore, the x will be used in procedure p.

Write

This is a simple statement that writes the output onto the screen.

```
const c = 8;
var x;
procedure p;
    var y;

    begin
        y := c;
        x := x + y;
    end;
begin
    x := 5;
    call p;
    write x;
end.
```

This is the same program from before and since the result is 13, it will output 13 onto the screen.

Read

This is also a simple statement since it acquires the users input.

EBNF of Tiny PL/0:

program ::= block "**.**" **.**
block ::= const-declaration var-declaration procedure-declaration statement**.**
constdeclaration ::= ["**const"** ident "**=**" number {"**,**" ident "**=**" number} "**;**"]**.**
var-declaration ::= ["**var**" ident {"**,**" ident} "**;**"]**.**
procedure-declaration ::= { "**procedure**" ident "**;**" block "**;**" }
statement ::= [ident "**:=**" expression
| "**begin**" statement { "**;**" statement } "**end**"
| "**if**" condition "**then**" statement
| "**while**" condition "**do**" statement
| "**read**" ident
| "**write**" ident
| **e**] **.**
condition ::= "**odd**" expression
| expression rel-op expression**.**
rel-op ::= "**=**"|"**<>**"|"**<**"|"**<=**"|"**>**"|"**>=**"**.**
expression ::= ["**+**"|"**-**"] term { ("**+**"|"**-**") term}**.**
term ::= factor {("*****"|"**/**") factor}**.**
factor ::= ident | number | "**(**" expression "**)**"**.**
number ::= digit {digit}**.**
ident ::= letter {letter | digit}**.**
digit ;;= "**0**" | "**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**"**.**
letter ::= "**a**" | "**b**" | … | "**y**" | "**z**" | "**A**" | "**B**" | … |"**Y**" | "**Z**"**.**

Based on Wirth's definition for EBNF we have the following rule:

[ ] means an optional item.
{ } means repeat 0 or more items.
Terminal symbols are enclosed in quote marks.

A period is used to indicate the end of the definition of a syntactic class.

Possible Errors

1. Use = instead of :=.
2. = must be followed by a number.
3. Identifier must be followed by =.
4. **const**, **var**, **procedure** must be followed by identifier.
5. Semicolon or comma missing.
6. Incorrect symbol after procedure declaration.
7. Statement expected.
8. Incorrect symbol after statement part in block.
9. Period expected.
10. Semicolon between statements missing.
11. Undeclared identifier.
12. Assignment to constant or procedure is not allowed.
13. Assignment operator expected.
14. **call** must be followed by an identifier.
15. Call of a constant or variable is meaningless.
16. **then** expected.
17. Semicolon or **}** expected.
18. **do** expected.
19. Incorrect symbol following statement.
20. Relational operator expected.
21. Expression must not contain a procedure identifier.
22. Right parenthesis missing.

23. The preceding factor cannot begin with this symbol.
24. An expression cannot begin with this symbol.
25. This number is too large.

Tokens Value

nulsym = 1

identsym = 2

numbersym = 3

plussym = 4

minussym = 5

multsym = 6

slashsym = 7

oddsym = 8

eqlsym = 9

neqsym = 10

lessym = 11

leqsym = 12

gtrsym = 13

geqsym = 14

lparentsym = 15

rparentsym = 16

commasym = 17

semicolonsym = 18

periodsym = 19

becomesym = 20

beginsym = 21

endsym = 22

ifsym = 23

thensym = 24

whilesym = 25

dosym = 26

callsym = 27

constsym = 28

varsym = 29

procsym = 30

writesym = 31

readsym = 32

elsesym = 33