PL/0 User's Manual

Shawn Roxby

To compile and run PL/0 compiler

- To Compile
    - Download PL/0 compiler
        - Go to the correct directory
        - Type in gcc lexicalAnalyzer.c parserandcompiler.c vm.c –o compile
    - While running
        - After compilation and while in the correct directory, type in command ./compile [<your_PL\0_text_file.txt>] (any combination –v, –a, or – l)
        - -l The lexeme list created
        - -a The disassembled code from the code generator
        - -v The virtual machine execution stack trace
    - How to use after it is running:
        - On console output (if input file contains errors): Error (error number): description of the error
        - On console output: result of the PL/0 Code.

Simple PL/0 program

- There will be a number of examples throughout

```
var x, y;

begin

y := 3;

x := y + 56; end.
```

Begin and end

- Mark the beginning and end of a program of subroutine

```
begin  /* line 1 */

end.
```

Var and Constant

- Used in the declaration of variables and constants of so that the program can use them to hold data and aid in the programming process.
- Programming is virtually impossible without variables and constants.
- Any Complex program uses constants and variables to load and store data

```
var x;

 const y=4;

 begin

x := 56; x := y;

end.
```

If, then, do and while

- If and then are used in programming. Conditional statement used with a condition to make certain lines of code reachable and unreachable, these are necessary in the programming process to make the code react tor.

```
var x; const y=4; begin

x:=0;

if x = 0 then x :=y; end.
```

- While is another kind of condition used to make some lines of code repeat in code until a certain condition are met.

```
var x;

begin

    x:=100;

    while x > 0

    do x :=x - 1;

end.
```

Procedure

- Procedures are segments of code that help organizes, makes code readable, and create a preferable aesthetic to PL/0 code.
- Procedures can be called and the program will jump to that segment of code execute it and return back to where the original place where it is called.

```
var x;

procedure A;

        begin

                x := 5;

        end;

        begin

                 x:=3;

                call A;

         end.
```

<u>EBNF of PL/0:</u>

- 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
    - | "**call**" ident
    - | "**begin**" statement { "**;**" statement } "**end**"
    - | "**if**" condition "**then**" statement ["**else**" statement]
    - | "**while**" condition "**do**" statement
    - | "**read**" ident
    - | "**write**" expression
    - | **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**"**.**