User Guide
for
PL/0 Compiler

Important notes before beginning:

- This guide will cover the process of compiling and running the PL/0 compiler using terminal, the command line utility of unix based systems. Note that it will require some basic knowledge of using terminal but the scope of this guide will not include how to use terminal. Other guides can be found for this.
- Command prompt, the windows command line utility, can also be used, but this guide does not cover the use of command prompt. The commands are not interchangeable, user beware.
- The key to success in this process is proper naming. All compiled components (the runnable versions) of the PL/0 compiler MUST be named precisely. Also, the input to the compiler must be named accordingly.
- All output will can be found where the PL/0 compiler files are located and/or in the terminal window that the commands are being issued.

Compiling the Compiler

- All of the .c files, as well as the input file named “input” (with a .txt EXTENSION, not in the name) must be in the same location/folder/directory.
- When all of these things are in the same location, open a terminal window and change the directory to where these files are located.
- Once in the correct directory, you can begin compiling the components of the PL/0 compiler.
- Compile each of the 4 components using the GCC compiler and commands:

```
gcc CompileDriver.c -o CompileDriver
gcc Scanner.c -o Scanner
gcc Parser.c -o Parser
gcc Pmachine.c -o Pmachine
```

- After this is done, you should now have 4 files in the same location that are now unix executables, with the same names as their source code files.
- To now run the PL/0 compiler, with the input.txt file ready and in the same location, type

```
./CompileDriver
```

The following command line arguments can be appended to the end of the previous command:

-l : to print the list of lexemes/tokens (scanner output) to the screen
-a : to print the generated assembly code (parser/codegen output) to the screen
-v : to print virtual machine execution trace (virtual machine output) to the screen

Creating the input file

The following is the EBNF grammar for writing PL/0 code:

```
<program> ::= block "." .
<block> ::= <const-declaration> <var-
declaration> <statement> .

<const-declaration> ::= [ "const" <ident> "="
<number> { "," <ident> "=" <number> } ";" ]

<var-declaration> ::= [ "var" <ident> { ","
<ident>} ";" ]

<statement > ::= [ <ident> ":=" <expression>
| "begin" <statement> { ";" <statement> } "end" |
"if" <condition> "then" <statement>
|ε].

<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 times.
Terminal symbols are enclosed in quote marks.

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

Following are two examples of PL/0 programs:

```
const n = 13;

var i, h;

procedure sub;
        const k = 7;
        var j, h;
        begin
                j:=n;
                i:=1;
                h:=k;
        end;
begin
        i:=3;
```

```
        h:=0;
        call sub;
end;
———————————
const m = 7, n = 85;
var i, x ,y, z, q, r;
procedure mult;
        var a, b;
        begin
                a := x; b := y; z := 0;
                while b > 0 do
                begin
                        if odd x then z := z+a;
                                a := 2*a;
                                b := b/2;
                end
        end;
begin
        x := m;
        y := n;
        call mult;
end.
```