PL/0 Compiler

User Guide

To compile:

(On Eustis)

cd into the same directory as the source code.

Run `make` to compile.

To use:

Make an input.txt containing your PL/0 code is in the same directory as pl0c.exe before running it.

(On Eustis)

Make sure you're in the same directory and enter `./pl0c.exe`

pl0c will output several different files:

cleaninput.txt

Input that has been cleared of comments.

lexemetable.txt

A table of detected lexemes that were converted internally into tokens.

lexemelist.txt

The list of lexemes the compiler will try to parse through for error checking and code generation.

symlist.txt

List of symbols detected, along with relevant debugging information.

mcode.txt

List of generated cpu instructions for your given input that runs through the VM.

stacktrace.txt

Lists the generated cpu instructions alongside the state of the stack after every instruction executed.

Example usage for PL/0:

To declare a variable...

```
var x;
```

Variable declarations must be made at the top of the block for a given scope.

To declare multiple variables in a given scope…

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

To assign a variable to a value…

```
x := 3;
```

```
x := 3 + 4;
```

```
x := y + (y + z * z);
```

The variable must already be defined before you can assign values to (or from) it.

Evaluates statements according to basic order of operations.

Variable assignments go in between `begin` and `end` statements.

To declare a constant…

```
const x = 3;
```

Constants must appear BEFORE variables of the same scope.

Example program:

```
const y = 5;
var x;
begin
      x := y;
end.
```

Whatever logic you want your program to do will go in between `begin` and `end`.

The final `end` must have a period after it.

To declare a procedure…

```
procedure MyProc; /* This is a comment. Anything between these
symbols will not be read by the program */
	var procVar;
	begin
		procVar := 7;
		write procVar; /* Prints to the screen */
	end;
begin
	call MyProc;
end.
```

Procedures have their own variable declarations and `begin`/`end` statements.

You can access variables from an outer scope within procedures if it doesn't exist in the procedure's scope, but this doesn't work the other way around.

For procedures, `end` must have a semicolon after it.

Logic constructs:

```
if someVar > otherVar then begin /* If true, do this */
	call MyProc;
end;

if someVar < otherVar then /* If true, do this. If not, do that. */
	call ThisProc  /* This is formatted this way because it */
else                /* looks neat, not because it's necessary */
	call ThatProc;

while someVar < otherVar do /* Loops while true */
begin
	read readValue; /* Asks for input, then stores it in readValue */
	someVar := someVar + readValue;
end;
```