**­­­­**

PL/0 User’s Guide

Authors: Travis Le and Allysia Freeman

Table of Contents

Table of Contents 2

Index of Figures and Tables 3

1.0: How to Program with PL/0 4

1.1 Data Types 4

1.2 Operators 5

1.3 Expressions 6

1.4 Statements 7

1.5 Procedures 8

2.0: Build, Compile, and Execute PL/0 programs 9

2.1 Building the Compiler 9

It is assumed you are using a linuix-based terminal for all following instructions. 9

To build this compiler: 9

**1.** Make sure all files listed above are in the same directory, and named exactly as shown. 9

**2.** Open your terminal. 9

**3.** Run the command “**gcc compiler.c –o compile**” 9

You should now have a file called “**compiler.o**” in your directory, and are ready to execute. 9

2.2 Executing the Compiler 9

If you have closed your terminal, please re-open it in order to execute the compiler. 9

In order to execute the compiler, run the command “**./compile**” 9

Alternatively, the following flags are available for your use: 10

**1.** **–l** which instructs the compiler to display the lexeme table to the screen. This is the internal interpretation of the input file. 10

**2.** **–a** which instructs the compiler to display the generated machine code to the screen. The meaning of these codes and their properties are listed in section five. 10

**3.** **–v** whichinstructs the compiler to display a stack trace for the execution of the input file. The stack is also explained in section five. 10

3.0: EBNF Grammar of PL/0 11

4.0: Tokens and Reserved Words 12

5.0: Machine Code Instructions in PL/0 14

6.0: Error Codes 15

Index of Figures and Tables

**No table of figures entries found.**

[Table 1: Extended Backus–Naur Form (EBNF) for PL/0 10](#_Toc424478790)

[Table 2: Tokens and Reserved Words of PL/0 12](#_Toc424478791)

[Table 3: Machine Code Instructions for PL/0 13](#_Toc424478792)

[Table 4: Error Codes 15](#_Toc424478793)

# **1.0: How to Program with PL/0**

* 1. Data Types
     1. **Variable**
     2. **Integer**
     3. **Procedure**
  2. Operators
     1. **Assignment**
     2. **Relational Operators**
     3. **Mathematical Operators**
  3. Expressions
  4. Statements
     1. **Assignments**
     2. **Read and Write**
     3. **While Do**
     4. **If, Then, Else**
  5. Procedures
     1. **Declaration**
     2. **Calling**
     3. **Recursion**
     4. **Nested Procedures**

**2.0: Build, Compile, and Execute PL/0 programs**

The Compiler is composed of the following files:

1. **compiler.c**

The main driver of the program, this handles all the other files and executes them as needed.

1. **header.h**

This holds all the declarations used often throughout all files. It also includes a section for redefining filenames. If you don’t know how to use #define in C programming, please don’t change these names.

1. **scanner.h**

This scans your input file and generates a list of lexemes (kinds of symbols).

1. **parser.h**

This takes the result from scanner.h and analyzes it for errors. It then generates machine code for the given input file. If it is found to be correct, it sends the generated machine code to vm.h

1. **vm.h**

This takes the machine code from parser.h and executes it.

1. **input.pl0**

This is your PL/0 code. It must be named this exactly, including the file extension **.pl0**. If you would like to change the input filename, edit *nameCode* in header.h, making sure to include the file’s extension. Complex file extensions are not supported, and it is strongly suggested to use either a plaintext or simple code editor to write your PL/0 code.

1. 1. Building the Compiler

It is assumed you are using a linuix-based terminal for all following instructions.

To build this compiler:

1. Make sure all files listed above are in the same directory, and named exactly as shown.
2. Open your terminal.
3. Run the command “**gcc compiler.c –o compile**”

You should now have a file called “**compiler.o**” in your directory, and are ready to execute.

* 1. Executing the Compiler

If you have closed your terminal, please re-open it in order to execute the compiler.

In order to execute the compiler, run the command “**./compile**”

Alternatively, the following flags are available for your use:

* **–l** which instructs the compiler to display the lexeme table to the screen. This is the internal interpretation of the input file.
* **–a** which instructs the compiler to display the generated machine code to the screen. The meaning of these codes and their properties are listed in section five.
* **–v** whichinstructs the compiler to display a stack trace for the execution of the input file. The stack is also explained in section five.

**3.0: EBNF Grammar of PL/0**

|  |
| --- |
| **program** = block "**.**" **.**  **block** = const-declaration var-declaration procedure-declaration statement **.**  **const-declaration** = [“**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  | "**while**" condition "**do**" statement  | "**read**" ident  | "**write**" ident] **.**  **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**"|"**c**"|**...** |"**x**"|"**y**"|"**z**" **.** |

Table 1: Extended Backus–Naur Form (EBNF) for PL/0

The first word of each section is the name of a “syntactic class.”

This is read using the following rules:

1. **|** means “or, but not both”
2. **[ ]** means an optional item
3. **{ }** means repeat zero or more times
4. Special symbols are enclosed in quote marks
5. A period is used to indicate the end of the definition of a syntactic class

**4.0: Tokens and Reserved Words**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Name** | **Value** | **Usage** |
|  | nulsym |  | n/a |
|  | identsym |  | used for constant, procedure, and variable names |
|  | numbersym |  | used for numbers |
| **+** | plussym |  | adds |
| **-** | minussym |  | subtracts |
| **\*** | multsym |  | multiplies |
| **/** | slashsym |  | divides |
| **odd** | oddsym |  | test if an expression is odd |
| **=** | eqlsym |  | constant definition or check is two expressions are equal |
| **<>** | neqsym |  | test that two expressions do not equal eachother |
| **<** | lessym |  | tests that left expression is less than right expression |
| **<=** | leqsym |  | tests that left expression is less than or equal to right expression |
| **>** | gtrsym |  | tests that left expression is greater than right expression |
| **>=** | geqsym |  | tests that left expression is greater than or equal to right expression |
| **(** | lparentsym |  | begin factor |
| **)** | rparentsym |  | end factor |
| **,** | commasym |  | separates identifiers in declarations |
| **;** | semicolonsym |  | ends a statement |
| **.** | periodsym |  | ends the program |
| **:=** | becomessym |  | assigns a value to a variable |
| **begin** | beginsym |  | begins a block of statements |
| **end** | endsym |  | ends a block of statements |
| **if** | ifsym |  | begins if statement, followed by condition |
| **then** | thensym |  | follows then, followed by statement |
| **while** | whilesym |  | begins while loop, followed by condition |
| **do** | dosym |  | follows while, followed by statement |
| **call** | callsym |  | calls a procedure |
| **const** | constsym |  | begins a constant declaration |
| **var** | varsym |  | begins a variable declaration |
| **procedure** | procsym |  | begins a procedure declaration |
| **write** | writesym |  | prints a value to the screen |
| **read** | readsym |  | asks the user to enter a value |
| **else** | elsesym |  | may follow if statements |

Table 2: Tokens and Reserved Words f PL/0

**5.0: Machine Code Instructions in PL/0**

* + 1. **The Stack**
    2. **Machine Code Meanings**

|  |  |  |
| --- | --- | --- |
| **1** |  |  |
| **2** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |
| **6** |  |  |
| **7** |  |  |
| **8** |  |  |
| **9** |  |  |

Table 3: Machine Code Instructions for PL/0

**6.0: Error Codes**

|  |  |  |
| --- | --- | --- |
| Error Number | Error Message | Explanation |
| **0** | Program is syntactically correct. | n/a |
| **1** | Invalid file input | Make sure the input file is named correctly or edit header.h's filename definitions |
| **2** | Use “=” not “:=” | Don’t use := when you’re not assigning a value to a variable. |
| **3** | Use “:=” not “=” | Always use := when assigning a value to a variable. |
| **4** | “=” expected after const declaration | When declaring a constant, it must be given a value. |
| **5** | Number expected after “=” with const | Constant declarations must be followed by =. |
| **6** | "then" expected after "if" | “then” must follow after an “if” statement. |
| **7** | "do" expected after "while" | “do” must follow after a “while” statement. |
| **8** | const, var, and procedure must be followed by an identifier | You must give a name to every constant, varable, and procedure when declaring it. |
| **9** | “:=” expected after identifier | Missing “:=” after a variable name. |
| **10** | Ident expected after “call” | The proper syntax is “call name;” |
| **11** | Relational operator expected | Missing a relational test. |
| **12** | Assignment to constants and procedures not allowed | You cannot assign a value to procedure names or constants. They are static. |
| **13** | Semicolon needed between statements | Missing a semicolon after a statement. |
| **14** | Cannot begin statement with this symbol | Check your program’s syntax. |
| **15** | Undeclared variable detected | Declare the variable before using it. |
| **16** | Unclosed parenthesis detected | All parenthesis must be closed. |
| **17** | Invalid operator | See operators section for correct implementation. |
| **18** | Invalid symbol | The symbol is not supported by PL/0. |
| **19** | ";" expected | Missed a “;” in code |
| **20** | Number too long | A number is too long. Make it shorter. |
| **21** | Identifier too long | The name of a variable is too long. Make it shorter. |
| **22** | Generated code too long | Your program is too complicated for the compiler. Shorten or simplify your code. |
| **23** | Compiler has run out of memory | n/a |
| **24** | Period expected. | A period must be at the end of the program. |
| **25** | Var or const detected more than once | You can only declare variables and constants once per level. Merge the declarations into one. |

Table 4: Error Codes