Ian Harvey

Ia678792

COP3402

Montane

The User’s Guide for the PL/0 Compiler

1. How to compile and run the PL/0 compiler.

Using the PL/0 compiler is easy. The compiler is all contained in one file called “hw4compiler.c” which itself can be compiled with the following command:

gcc hw4compiler.c

This creates an executable file that we can use to compile our programs written in PL/0. To compile a text file we have written in PL/0 use the following command:

./a.out <filename.txt>

The compiler also supports a few directive flags that you can add before the filename field.

-l prints out the lexeme list that the scanner produces.

-a prints out the generated assembly code.

-v prints out the PL/0 virtual machine execution trace.

-s prints out the source code to screen.

-d prints out additional debug information including the symbol table and token traces following some error messages.

With these flags the command to compile and run your programs looks like this:

./a.out [-l] [-a] [-v] [-s] [-d] <filename.txt>

Where “[]” denotes an optional field and “<>” denotes a placeholder.

1. How to use the PL/0 language.

The PL/0 language is an ancient language that in the early days of computing was used as training wheels for the more sophisticated PL/1. It’s structure is unlike any modern language mainly due to its limited support of structures we are oh so used to. Every program starts with a block and ends with a period. The structure of a block is as follows:

Const-declaration var-declaration procedure-declaration statement.

Where constant, variable and procedure declaration are optional.

To declare constants, type the keyword “const” followed by a valid identifier name and then assign the constant a value as this is the only time you will be able to and all constants must be initialized. A valid identifier starts with a letter and is then followed by either a number or letter, up to 10 times (making the max length 11). To declare multiple constants, separate the values by a comma. At the end of constant declaration, you put a semicolon to denote you are done declaring constants. An example is as follows:

const a = 1, nice = -1, var21 = 10;

Next is variable declaration. It is like constant declaration except you are not allowed to assign values to variables when they are declared. We use the “var” keyword to start this. An example is as follows:

var x, y, z;

Next is procedure declaration. To declare a procedure, use the “procedure” keyword followed by a valid identifier to identify the procedure followed by a semicolon followed by another block structure followed by another semicolon. I.e.:

procedure foo;

const a = 0;

var x;

begin

x = a;

write a;

end;

“What’s that stuff after the variable declaration?” you may ask. It is a statement, the last part of a block. If you only have one statement you do not need a “begin” and “end” but to have multiple statements they must be surrounded by a begin and end. Every statement inside a begin and end must end with a semicolon. I.e.:

var a;

begin

read a;

write a;

end.

There are many statements that I will begin to go over now.

The call statement is structured as follows:

call <ident>

this is used to call your previously declared procedures and jump to the bit of code that they generate under their own statement.

The if then statement is structured as follows:

if <condition>> then <statement> [else <statement]

where the condition is either

odd <expression>

where the odd function checks if the following expression is odd or not. Or the condition is two expressions with a relational operator in the middle. The relational operators are as follows:

= two expressions equal?

!= two expressions not equal?

< left less than right?

<= left less than or equal to right?

> left greater than right?

>= left greater than or equal to right?

An expression is any combination of “+”,”-“,”\*”,”/”,variables, constants, and parenthesis that logically makes sense. I.e.:

(x – 1) \* y

1 + 2 + 3 \* 5

MAX / 2 + 1

After the condition we follow with the “then” keyword and then a statement. After this statement we then can add an optional “else” keyword to add another statement if the condition is false. An example of a full if then else is as follows:

If x > 0 then x := x – 1 else y := y + 1

You may notice there is assignment here. The structure for assignment is as follows:

<ident> := <expression>

Where a previously declared constant or variable is assigned the value in expression by the “:=” operator.

The while do statement is structured as follows:

while <condition> do <statement>

This is your only loop structure in the language. When the loop gets to the end of the statement after “do” it loops back to check the condition again and keeps on repeating until the condition is false. An example is as follows:

var x;

begin

x := 5;

while x > 0 do

begin

write x;

x := x – 1;

end;

end.

This small program prints out numbers from 5 to 1.

We also have “read” and “write” statements. A read consists of the “read” keyword and then a valid variable and then takes an integer value from stdin and stores it to the variable. A write consists of the “write” keyword and then an expression to write to screen. You can write variables, constants, and expressions that evaluate to numbers. More formally, these are declared below:

read <ident>

write <expression>

A statement also can be left blank meaning the following program is valid:

var x;

.

Although you rarely will be leaving the statement part blank as it is the only way to really do stuff in your program.

Thus ends the lengthy guide to programming in PL/0. In short, don’t. If you have to and this guide did not answer all your questions search online “EBNF of PL/0” for a formal declaration of the language or search online for other guides to this language as it is literally over 40 years old and was made when programming was but a baby to the world.