University of Central Florida

|  |  |
| --- | --- |
|  | PL/0 Compiler User Guide |
|  |  |
|  | Anthony Dinh, Alvin Lam  Fall 2014  11/9/14 |

Department of Electrical Engineering & Computer Science

COP 3402 System Software

Homework 4 (PL/0 Compiler)

**Compiling the PL/0 Compiler**

This guide assumes that the user is on the UCF Eustis server.

There are several ways to compile the PL/0 compiler

1. Using the Makefile that we have included in the .zip file.

If you use the Makefile, type these commands into the console

\* “make scanner”

\* “make ParserCodeGenerator”

\* “make P-Machine”

\* “make compiler”

Please be aware that if you use the Makefile, the name of the executable for the CompilerDriver will be called "compiler" by default.

2. The good ol fashion way, typing everything by hand

If you choose to use this method to compile the tiny compiler, please use these commands.

\* “gcc -o scanner scanner.c”

\* “gcc -o P-Machine P-Machine.c”

\* “gcc -o ParserCodeGenerator ParserCodeGenerator.c”

\* “gcc -o compiler CompileDriver.c”

**Running the PL/0 Compiler**

You have several options for running the PL/0 compiler.

If you simply wish to compile and run the PL/0 code, type the following into your console.

* “./compiler”

Please ensure that you have an input PL/0 file labeled “input.txt” or the compiler will give you an error.

It is also possible run the compiler with several flags. Adding these flags allow you to see the output of the various components of the compiler outputted to your screen.

There are 3 valid flags

\* The -l argument prints the lexemelist to the screen, which is the output of the Lexical Analyzer

\*The -a argument prints the Machine code to the screen, which is the output of the Parser/CodeGenerator

\*The -v argument prints the Stack to the screen, which is the output of the P-Machine

Running the Compile Driver without any arguments will only print whether the code is synatically correct and program out such as the write command

Regardless of the arguments given, the Compiler Driver will always produce an output.txt file which contains all the output of the Scanner, Parser/CodeGenerator, and the P-Machine

The program will only accept 3 arguments from the user, if the user inputs more than 3 commands, the CompileDriver will crash.

A sample command would be "./compiler -l -v", this command will print the lexeme list and the stack trace of the P-Machine, but not the Machine code from the Parser/Code Generator.

**How to use the PL/0 Language**

A PL/0 program consists of a block (the actual program) and a period (denotes the end of the program).

The only thing required in the PL/0 program is the period at the end. So the following is valid PL/0 code

.

The **block** contains a constant declaration, variable declaration, procedure declaration and a statement. They must be declared in this order.

The **constant declaration** contains all the constants in the program, values that will not change during the course of the program. Constants are optional, you do not need to include constants in your program.

They look like this [“**const”** ident “=” number {“,” ident “=” number} “;”]

The following is a valid constant declaration in PL/0

Const a = 5;

“**const”** is the keyword used to tell the compiler that you are declaring a constant.

Ident is the name of the constant .

The “=” symbol tells the compiler that you are setting the constant to a number.

Number denotes the value that you are setting the constant to.

{“,” ident “=” number} denotes that you have the option of declaring more than one constant; but if you declare more than one constant, you must separate the constants with “,”.

The “;” symbol tells the compiler that you are finished declaring constants. If you declare constants, you **must** end the constant declarations with “;”.

The **variable declaration** contains all the variables in the program, values that can change during the course of the program. Variables are optional, you do not need to include variable in your program.

They look like this [“**var** "ident {"**,**" ident} “**;**"]

The following is a valid variable declaration in PL/0

var a;

"**var**" is the keyword used to tell the compiler that you are declaring a variable.

Ident is the name of the variable that you are declaring.

{"**,**" ident} denotes that you have the option of declaring more than one variable; but if you declare more than one variable, you must separate the variables with “;”.

The “;” symbol tells the compiler that you are finished declaring variables. If you declare variables, you **must** end the variable declarations with “;”.

The **procedure declaration** contains all the procedures in the program, these are known as function in the C language. Procedures are optional, you do not need to include procedures in your program.

They look like this { "**procedure**" ident "**;**" block "**;**" }

The following is a valid procedure declaration in PL/0

procedure a;;

“**procedure”** is the keyword used to tell the compiler that you are declaring a procedure.

Ident is the name of the procedure that you are declaring.

The “;” symbol tells the compiler that you are finished declaring the identifier for the procedure.

block refer to the fact that a procedure are essentially self-contained programs. Procedures can contain their own constant, variable, and procedure declarations.

The “;” symbol tells the compiler that you are finished declaring the procedure.

The **statement** contains the majority of the program. A statement can contain a variable assignment, a call to another procedure, a begin statement, an if statement, while statement, read, write, or e.

A variable assignment is defined as ident “:=” expression

Ident is the name of the variable being changed

The “:=” symbol denotes that the variable is being changed to another value.

expression is the value that the variable is being changed to.

A valid variable assignment would be.

a := 5

Where a is a variable that was previously defined.

**call** represents a call to another procedure.

A call is defined as “**call”** ident

The following is a valid call

Call jimmyJohns

Where jimmyJohns was a procedure that was previously defined in the program.

“**call”** is the keyword used to tell the compiler that you are calling another procedure.

Ident is the name of the procedure that you are calling. This procedure must have already been previously defined.

A beginstatement is defined as "**begin**" statement { "**;**" statement } "**end**"

The “**begin”** must be followed by a statement, you may also include multiple statements with your begin statement, but you must separate them with “;”. At the end of the begin statement you must include “**end”** to denote the end of the begin statement. The following is a valid begin statement.

begin

x := 5

end

An if statement is defined as "**if**" condition "**then**" statement ["**else**" statement]

The “**if”** keyword must be followed by a condition, followed by the “**then”** keyword, a statement must follow the “**then”** keyword. You have the option of including an else statement which executes if the if statement is not true. The else statement must include the keyword “**else”** followed by a statement. The following is a valid if statement

If n =0 then call fact

A while statement is defined as "**while**" condition "**do**" statement

The “**while”** keyword must be followed by a condition, then followed by the “**do”** keyword, a statement must follow the “**do”** keyword. The following is a valid while statement

while x > 5 do x := x -1

A read statement is defined as "**read**" ident

Ident is the variable that you are reading the value to.

Read x

A write statement is defined as "**write**" ident

Ident is the variable you are printing to the screen.

Write x

A **condition** is defined as expression rel-op expression

An **expression** is defined as a positive or negative term which may be added or subtracted from another term

A **rel-op**  is defined as a relation operator. There are 6 relational operators.

* “=” – equality
* “<>”- not equal to
* “<” – less than
* “<=” – less than or equal to
* “>” – greater than
* “>=” - greater than or equal to

A **term** is defined as a factor which may be multiplied or divided by another factor.

A **factor** is defined as an identifier, number, or expression

A **number** is a one or more digits

An **ident** is defined as a letter followed by 0 or many letters or digits

A **digit** is defined as “0” – “9”

A **letter** is defined as “a” – “ Z”