

PL/0 Compiler User's Guide

COP 3402: SYSTEMS SOFTWARE

FALL 2014

Eric Peralli
Thomas Tavarez

The PL/0 Compiler

COMPONENTS

Our PL/0 consists of multiple components: The Lexicographical Analyzer, The Parser/Code Generator, and the Virtual Machine. Each of these components is described below.

Lexicographical Analyzer – The lexicographical analyzer traverses the PL/0 program and converts the code into numerical tokens.

Parser/Code Generator – The parser/code generator reads the tokens generated by the lexicographical analyzer, and determines if the program contains any grammatical errors. If no errors are encountered, the parser will generate the object code.

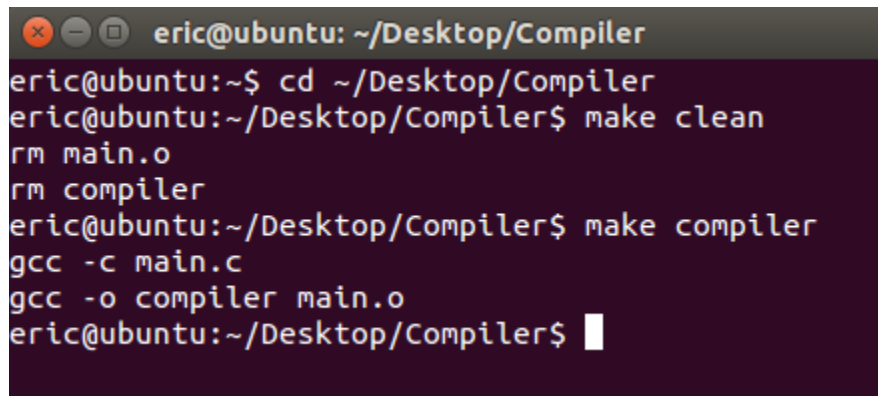
Virtual Machine – The virtual machine, which simulates an environment for the code to run efficiently, reads the object code and executes each line in order. After successfully executing the PL/0 program, the compiler generates an output file consisting of each component's output.

BUILDING THE COMPILER

The following instructions are meant to be executed in a UNIX based command-line interpreter (i.e. terminal). For demonstration purposes, we will execute these commands in a Linux Terminal session.

- Open your command-line interpreter and navigate to the compiler's directory using the "cd" command
- While in this directory, execute the "make clean" command to remove any preexisting executables
- Then, execute the "make compiler" to create new executables

An example of running the steps above is shown below in Figure 1:

A screenshot of a terminal window titled "eric@ubuntu: ~/Desktop/Compiler". The terminal shows a series of commands and their outputs. The user first changes the directory to ~/Desktop/Compiler, then runs "make clean" which removes "main.o" and "compiler". Next, the user runs "make compiler", which compiles "main.c" into "compiler" using "gcc -c main.c" and "gcc -o compiler main.o". The terminal ends with the prompt "eric@ubuntu:~/Desktop/Compiler\$".

```
eric@ubuntu: ~/Desktop/Compiler
eric@ubuntu:~$ cd ~/Desktop/Compiler
eric@ubuntu:~/Desktop/Compiler$ make clean
rm main.o
rm compiler
eric@ubuntu:~/Desktop/Compiler$ make compiler
gcc -c main.c
gcc -o compiler main.o
eric@ubuntu:~/Desktop/Compiler$
```

Figure 1 - Terminal Session

RUNNING THE COMPILER

Commands

The following commands are used to run the compiler:

- `make compiler`: This command compiles the program. And write the output to "Compiler_Output.txt".
- `./compiler`: This command will run the entire program, but display nothing to the screen
- `./compiler -v`: This command will display the virtual machine execution trace from the Virtual Machine.
- `./compiler -l`: This command will display the lexeme list from the Scanner.
- `./compiler -a`: This command will display the generated assembly code from the Parser/Code Generator.
- `make clean`: This command will delete any output files generated by the program. After executing this command, you must execute "make compiler" again.

The commands above are meant to be entered into a UNIX based command-line interpreter.

Input

By default, the compiler will read in the PL/0 code from the file "input.txt" located in the compiler's directory.

The PL/0 Language

Our COP 3402 compiler supports the PL/0 language, which is a language meant to demonstrate the construction of compilers.

FORMAT

A PL/0 program follows a strict format. The program must start with either variable or constant declarations, which can only be declared at the beginning of the main program or procedure. The body of the main program and procedures is called a block. Inside the block are statements, which can contain more declaration, and operations. An example of a full PL/0 program can be seen below.

```
const a=1, red= 6, blue=5;
var x;
procedure A;
begin
    if x = 1 then
        write red
    else
        write blue
end;
begin
    read x;
    call A;
end.
```

Block

Blocks are sections of the PL/0 code that are denoted by the reserved words “begin” and “end.”. After the “begin” command, you can find statements, procedure declarations, constant declarations, and variable declarations.

Identifiers

Variable – A variable is an identifier that contains a numerical value. Variables can be assigned a value using the “:=” operator.

Constant – A constant is a type of variable that can only have its value assigned once. Constants can only be declared at the top of a procedure, or the main code. Constants can be assigned values using the “=” operator.

Procedure – A procedure can be described as a program within a program. Procedures are declared as “Procedure x;”, with x representing the name of the procedure. Procedures contain blocks, and make use of the “begin” and “end;” reserved words to denote their blocks. They are called within the code using the “call x;” command, with x being the name of the procedure.

The following are examples of variables, constants, procedures, assigning them values, and the call function:

```
var a;
```

```
const b = 3;
```

```
Procedure c;
```

```
begin
```

```
    a:= 3 +4;
```

```
end;
```

```
call c;
```

```
end.
```

Conditional Statements

Conditional statements are sections in the PL/0 code that performs tasks based on the outcome of a logical statement. They used the reserved words “if”, “then, and “else”. Conditional statements check if a logical statement is true, and will perform the code following “then”. If the statement is false, then the “else” code is executed. An example of a conditional statement is show below.

```
var x, w;
```

```
begin
```

```
    x:= 4;
```

```
    read w;
```

```
    if w > x then
```

```
        w:= w + 1;
```

```
    else
```

```
        w:= x;
```

```
    write w;  
end.
```

Input/Output

The compiler handles input from a user using the reserved word “read x” with x denoting the variable that the input is written to. The compiler handles output using the reserved word “write x” with x denoting the variable, constant, or number that the user wants written to the screen. Below is an example of read and write in PL/0 code.

```
var a;  
read a;  
begin  
    write a;  
end.
```

Loops

A loop is a piece of code that will repeat itself until the exit condition is satisfied. PL/0 denotes loops using the reserved words “while” and “do”. As long as the while statement is true, the code in do will execute repeatedly. An example can be seen below.

```
const x=0,y=3;  
var i,j;  
begin  
    while i < y do  
        begin  
            j := 0;  
            i := i + 1;  
            while j < i do  
                begin
```

```
    write j;
```

```
    j := j + 1;
```

```
end;
```

```
write y
```

```
end
```

```
end.
```