PL/0 User's Manual

Authors: Tuan Pham, Gregory White

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

A. Compiling PL/0 compiler

1. Download PL/0 compiler
2. Open terminal window
3. Navigate to PL/0 compiler directory
4. Terminal command: make

B. Running PL/0 compiler

1. Place text file named 'input.txt' containing PL/0 code into project directory
2. Open terminal window
3. Navigate to compiled PL/0 compiler directory
4. Terminal command: ./compiler

II. How to use the PL/0 compiler once it is running

A. Using PL/0 compiler:

After a user has compiled and ran the PL/0 compiler the only user action required is any input required from the PL/0 code contained in 'input.txt'.

1. Console output: lexeme list used during code compilation process
2. Console output (if 'input.txt' contains errors): Error # and description of error
   Revise 'input.txt' file and re-run PL/0 compiler
3. Console output (if 'input.txt' contains NO errors): "No errors, program is syntactically correct."
4. Console input: Numerical input required from PL/0 code in 'input.txt'
5. Console output: Answer according to PL/0 code in 'input.txt'

III. How to use the PL/0 language

PL/0 uses the EBNF Grammar convention (Section IV-A) for writing code.
General Program structure:

1) Define constants
2) Declare variables
3) Declare procedures (uses same structure as programs)
4) Declare statements
5) A period marking the end of a program

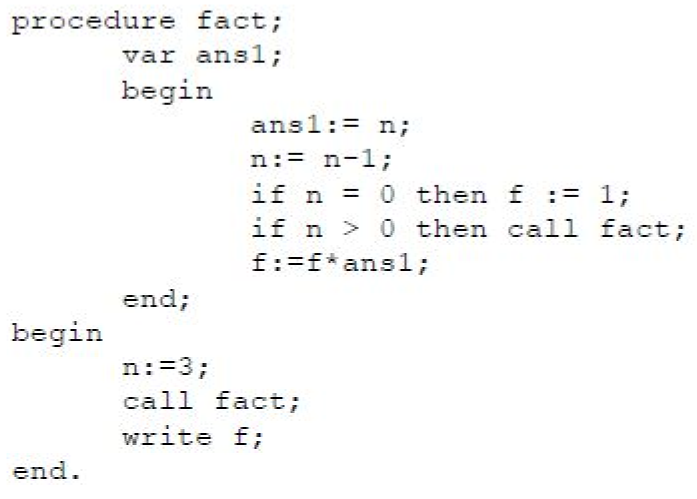
A. Datatypes

All PL/0 datatypes are integers. They have a maximum digit length of 5. PL/0 datatypes must be identified with an identifier. Identifiers have a maximum length of 11. They must start with an alphabetic character and may contain alphanumeric characters and underscores.

1. Constants
   Constants can **NOT** be changed after they are defined. Multiple constants can be defined.
   Ex: const foo = 0, bar = 1, foo1 = 12345;
2. Variables
   Variables can be changed after they are defined. The only variable type PL/0 supports are integers. Multiple integers can be declared.
   Ex: int foo, bar, foo1

B. Procedures

Procedures are identified by an identifier. They are called by the reserved word call. Procedures follow a similar structure to programs. PL/0 supports multiple procedures and recursion.
Procedure Call Ex:



```
procedure fact;
      var ans1;
      begin
             ans1:= n;
             n:= n-1;
             if n = 0 then f := 1;
             if n > 0 then call fact;
             f:=f*ans1;
      end;
begin
      n:=3;
      call fact;
      write f;
end.
```

Recursive Procedure Ex:

```
procedure fact;
      var ans1;
      begin
             ans1:= n;
             n:= n-1;
             if n = 0 then f := 1;
             if n > 0 then call fact;
             f:=f*ans1;
      end;
```

C. Statements

PL/0 supports a variety of statements including repetition and selection. Statements must be separated by semi-colons.

1. Repetition

   To repeat instructions multiple times a while loop can be used.

   Ex:

   ```
   while i > loopEnd do
   begin
           s:= s+i;
           i:= i-1;
   end;
   ```

2. Selection

   To execute different sets of code according to certain conditions an if, then, else set of statements may be used.

   ```
   if x<0 then
           y:=0-x
   else
           y:=x;
   ```

IV. References

A. PL/0 EBNF Grammar

```
program ::= block "." .
block ::= const-declaration  var-declaration  procedure-declaration statement.
constdeclaration ::= ["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 ["else" statement]
                | "while" condition "do" statement
                | "read" ident
                | "write" ident
                | e ] .
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" | … | "y" | "z" | "A" | "B" | ... | "Y" | "Z".
```

B. Lexical Conventions

Reserved Words: const, var, procedure, call, begin, end, if, then, else, while, do, read, write, odd.
Special Symbols: ‘+’, ‘-‘, ‘*’, ‘/’, ‘(‘, ‘)’, ‘=’, ’,’ , ‘.’, ‘ <’, ‘>’, ‘;’ , ’:’ .
Identifiers: identsym = letter (letter | digit)*
Numbers: numbersym = $(\text{digit})^+$
Invisible Characters: tab, white spaces, newline
Comments denoted by: /* . . .   */