

PL/0 Lexical Considerations

Category	Lexeme	Token Name	Numerical Value
		nulsym	1
Literals and Identifiers	letter (letter digit)*	identsym	2
	(digit)+	numbersym	3
Arithmetic Operators	+	plussym	4
	-	minussym	5
	*	multsym	6
	/	slashsym	7
Comparisons	odd	oddsym	8
	=	equalsym	9
	<>	neqsym	10
	<	lessym	11
	<=	leqsym	12
	>	gtrsym	13
	>=	geqsym	14
Syntax and Assignment	(lparentsym	15
)	rparentsym	16
	,	commasym	17
	;	semicolonsym	18
	.	periodsym	19
	:=	becomesym	20
Reserved Words	begin	beginsym	21
	end	endsym	22
	if	ifsym	23
	then	thensym	24
	while	whilesym	25
	do	dosym	26
	call	callsym	27
	const	constsym	28
	var	varsym	29
	procedure	procsym	30
	write	writesym	31
	read	readsym	32
	else	elsesym	33

Comments are /* C-style */.

PL/0 Scanning Quick Reference

Source Code	Symbolic Tokens (not output)
var x, y; begin y := 3; /* Comment */ x := y + 56; end.	varsym identsym x commasym identsym y semicolonsym beginsym identsym y becomesym numbersym 3 semicolonsym identsym x becomesym identsym y plussym numbersym 56 semicolonsym endsym periodsym
Symbolic Tokens (not output)	Numeric Tokens
varsym identsym x commasym identsym y semicolonsym beginsym identsym y becomesym numbersym 3 semicolonsym identsym x becomesym identsym y plussym numbersym 56 semicolonsym endsym periodsym	29 2 x 17 2 y 18 21 2 y 20 3 3 18 2 x 20 2 y 4 3 56 18 22 19

Actual output:

File	Output
cleaninput.txt	var x, y; begin y := 3; x := y + 56; end.
lexemetable.txt	lexeme token type var 29 x 2 , 17 y 2 ; 18 begin 21 y 2 := 20 3 3 ; 18 x 2 := 20 y 2 + 4 56 3 ; 18 end 22 . 19
tokenlist.txt	29 2 x 17 2 y 18 21 2 y 20 3 3 18 2 x 20 2 y 4 3 56 18 22 19

PL/0 Grammar (extended BNF)

program	::= block "."
block	::= const-declaration var-declaration proc-declaration statement
const-declaration	::= ["const" ident "=" number { "," ident "=" number } ";"]
var-declaration	::= ["var" ident { "," ident } ";"]
proc-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"

Legend:

[The contents of brackets are optional]

{ The contents of braces are repeated zero or more times }

Terminal (i.e., literal) symbols are enclosed in "quote marks"

Parentheses and vertical bars act like they do in regular expressions

Useful C Declarations

```
typedef enum {
    nulssym = 1, identsym, numbersym, plussym, minussym,
    multsym, slashsym, oddsym, eqsym, neqsym, lessym, leqsym,
    gtrsym, geqsym, lparsym, rparsym, commasym, semicolonsym,
    periodsym, becomessym, beginsym, endsym, ifsym, thensym,
    whilesym, dosym, callsym, constsym, varsym, procsym, writesym,
    readsym, elsesym
} token_type;

#define MAX_SYMBOL_TABLE_SIZE 100

/* For constants, store kind, name and val
   For variables, store kind, name, L and M
   For procedures, store kind, name, L and M */

typedef struct symbol {
    int kind;           // const = 1, var = 2, proc = 3
    char name[12];      // name up to 11 chars
    int val;            // value
    int level;          // L level
    int addr;           // M address
} symbol;

symbol symbol_table[MAX_SYMBOL_TABLE_SIZE];
```

PL/0 Parser Pseudocode 1/3

This pseudocode shows you how to walk through the syntax of a PL/0 program. It does *not* show you how to generate code.

```
procedure PROGRAM;
begin
  GET(TOKEN);
  BLOCK;
  if TOKEN != "periodsym" then ERROR
end;

procedure BLOCK;
begin
  if TOKEN = "constsym" then begin
    repeat
      GET(TOKEN);
      if TOKEN != "identsym" then ERROR;
      GET(TOKEN);
      if TOKEN != "eqsym" then ERROR;
      GET(TOKEN);
      if TOKEN != NUMBER then ERROR;
      GET(TOKEN)
    until TOKEN != "commasym";
    if TOKEN != "semicolonsym" then ERROR;
    GET(TOKEN)
  end;
  if TOKEN = "varsym" then begin
    repeat
      GET(TOKEN);
      if TOKEN != "identsym" then ERROR;
      GET(TOKEN)
    until TOKEN != "commasym";
    if TOKEN != "semicolonsym" then ERROR;
    GET(TOKEN)
  end;
  while TOKEN = "procsym" do begin
    GET(TOKEN);
    if TOKEN != "identsym" then ERROR;
    GET(TOKEN);
    if TOKEN != "semicolonsym" then ERROR;
    GET(TOKEN);
    BLOCK;
    if TOKEN != "semicolonsym" then ERROR;
    GET(TOKEN)
  end;
  STATEMENT
end;
```

PL/0 Parser Pseudocode 2/3

```
procedure STATEMENT;  
begin  
    if TOKEN = "identsym" then begin  
        GET(TOKEN);  
        if TOKEN != "becomessym" then ERROR;  
        GET(TOKEN);  
        EXPRESSION  
    end  
    else if TOKEN = "callsym" then begin  
        GET(TOKEN);  
        if TOKEN != "identsym" then ERROR;  
        GET(TOKEN)  
    end  
    else if TOKEN = "beginsym" then begin  
        GET TOKEN;  
        STATEMENT;  
        while TOKEN = "semicolon" do begin  
            GET(TOKEN);  
            STATEMENT  
        end;  
        if TOKEN != "endsym" then ERROR;  
        GET(TOKEN)  
    end  
    else if TOKEN = "ifsym" then begin  
        GET(TOKEN);  
        CONDITION;  
        if TOKEN != "thensym" then ERROR;  
        GET(TOKEN);  
        STATEMENT  
    end  
    else if TOKEN = "whilesym" then begin  
        GET(TOKEN);  
        CONDITION;  
        if TOKEN != "dosym" then ERROR;  
        GET(TOKEN);  
        STATEMENT  
    end  
end;  
end;
```

```
procedure CONDITION;  
begin  
    if TOKEN = "oddsym" then begin  
        GET(TOKEN);  
        EXPRESSION  
    else begin  
        EXPRESSION;  
        if TOKEN != "RELATION" then ERROR;  
        GET(TOKEN);  
        EXPRESSION  
    end  
end;  
end;
```

PL/0 Parser Pseudocode 3/3

```
procedure EXPRESSION;  
begin  
    if TOKEN = "plussym" or "minussym" then GET(TOKEN);  
    TERM;  
    while TOKEN = "plussym" or "minussym" do begin  
        GET(TOKEN);  
        TERM  
    end  
end;
```

```
procedure TERM;  
begin  
    FACTOR;  
    while TOKEN = "multsym" or "slashsym" do begin  
        GET(TOKEN);  
        FACTOR  
    end  
end;
```

```
procedure FACTOR;  
begin  
    if TOKEN = "identsym" then  
        GET(TOKEN)  
    else if TOKEN = NUMBER then  
        GET(TOKEN)  
    else if TOKEN = "(" then begin  
        GET(TOKEN);  
        EXPRESSION;  
        if TOKEN != ")" then ERROR;  
        GET(TOKEN)  
    end  
    else ERROR  
end;
```

Example PL/0 Parser Error Messages

1. Use = instead of :=.
2. = must be followed by a number.
3. Identifier must be followed by :=.
4. const, var, procedure must be followed by identifier.
5. Semicolon or comma missing.
6. Incorrect symbol after procedure declaration.
7. Statement expected.
8. Incorrect symbol after statement part in block.
9. Period expected.
10. Semicolon between statements missing.
11. Undeclared identifier.
12. Assignment to constant or procedure is not allowed.
13. Assignment operator expected.
14. call must be followed by an identifier.
15. Call of a constant or variable is meaningless.
16. then expected.
17. Semicolon or } expected.
18. do expected.
19. Incorrect symbol following statement.
20. Relational operator expected.
21. Expression must not contain a procedure identifier.
22. Right parenthesis missing.
23. The preceding factor cannot begin with this symbol.
24. An expression cannot begin with this symbol.
25. This number is too large.