MOTOROLA

M68KPASC/L2

JANUARY 1983

TO: Users of Motorola Pascal on M68000 systems

This letter describes corrections to Cherry's book, Pascal Programming Structures for Motorola Microprocessors, as well as additional features of the M68000 Pascal compiler not mentioned in the Cherry publication.

Independent Compilation (Subprograms)

Level one procedure and function declarations may have their declaration and statement parts replaced by the Pascal directive forward, and then have their specific program parts treated as external and compiled separately within a subprogram. The subprograms are linked to the main program when the load module is created by the linker.

The form of a subprogram is similar to the form of a Pascal program. It consists of a subprogram heading and declaration part. However, it does not contain a statement part.

The subprogram heading contains, in the following order: (1) the symbol subprogram, (2) a subprogram name identifier, and (3) a subprogram parameter list. The subprogram parameters should be the same as the program parameters in type, number, and order.

The declaration part of a subprogram consists of the declaration of variables, procedures, and functions, and the definition of constants and types. In order to preserve recognition of global identifiers, all variables in the subprogram variable declaration part must agree in type, number, and order with those appearing in the program's variable declaration part.

Constants and types declared in the Pascal program may be duplicated in a subprogram.

Among the procedures and functions declared in the subprograms are those which were declared as external (forward) in the Pascal program. These externally-referenced procedures and functions must be declared at level one.

The final end statement of the final procedure or function declared in the subprogram is followed by a period (.), which terminates the subprogram.

The following is an example that shows the separate compilation of modules. The program "matrix" declares a procedure "multiply", which is treated as external because its declaration and statement parts are not included within the Pascal program module. In a separate module (the subprogram "multiply"), the procedure is then fully declared. Note that within the subprogram, the global variables (a, b, c, inmat, and outmat) are declared in the same order and have the same type as in the Pascal program.

Example:

Module One

```
program matrix (inmat, outmat);
type matrix = array [1..50,1..50] of integer;
var a,b,c : matrix;
    inmat,outmat : file of matrix;
procedure multiply (size:integer);forward;
begin
   a := inmat↑;
   get (inmat);
   b := inmat↑;
   multiply (20);
   outmat↑ := c;
   put(outmat)
end.
```

Module Two

```
subprogram multiply (inmat,outmat);
type matrix = array [1..50,1..50] of integer;
var a,b,c : matrix;
    inmat,outmat : file of matrix;
procedure mutiply (n:integer);
   var i,j,k, product:integer;
   begin
      for i := 1 to n do
         for j := 1 to n do
            begin
               product := 0;
               for k := 1 to n do
                  product := product + a[i,k]*b[k,j];
               c[i,j] := product
            end
   end.
```

Additional BNF Syntax Descriptions of Motorola Extensions

Note — The following symbols are meta-symbols belonging to the syntactic format, and not symbols of the Pascal programming language itself, except as noted:

< >	Enclose a symbol, called a syntactic variable.
::=	"is defined as"
|	"or"
[ ]...	Denote that the enclosed symbols are optional/repetitive — that is, occur zero or more times.
[ ]	Denote that the enclosed symbols are optional — that is, occur zero or one time. Exceptions are in <identifier item> definition, as noted.

1. Alphanumeric labels:

<label> ::= <unsigned integer>|<identifier>

<label declaration part> ::= label <label>[,<label>]...;

2. Exit statement:

<exit statement> ::= exit [<label>]

<simple statement> ::= <assignment statement>|<procedure statement>|
<goto statement>|<empty statement>|<exit statement>

3. Hex constants:

<hexdigit> ::= <digit>|A|B|C|D|E|F

<hexdigit sequence> ::= <hexdigit>[<hexdigit>]...

<unsigned integer> ::= [<digit sequence>#]<hexdigit sequence>

<unsigned number> ::= <unsigned integer>

4. Origin in variable declaration:

<identifier item> ::= <identifier>[[origin <unsigned integer>]]

NOTE: The inside brackets are symbols of the language; the outside brackets are meta-symbols.

<variable declaration> ::= <identifier item>[,<identifier item>]...:<type>

<variable declaration part> ::= var <variable declaration>
[;<variable declaration>]...;

5. Otherwise in case statement:

<case statement> ::= case <expression> of <case list element>
[;<case list element>]...[;otherwise <statement>]end

6. Subprograms:

<subprogram> ::= <subprogram heading>[<declaration part>]...<period>

<subprogram heading> ::= subprogram <identifier>(<subprogram parameters>);

<subprogram parameters> ::= <identifier>[,<identifier>]...

<declaration part> ::= <label declaration part>|<constant definition part>|
<type definition part>|<variable declaration part>|
<procedure declaration>|<function declaraton>

7. Underscores in identifiers:

<identifier> ::= <letter>[<letter or digit>]...

<letter> ::=

A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|
k|l|m |n|o|p|q|r|s|t|u|v|w|x|y|z|_

Corrections

1. Both the circumflex (^) and the at sign (@) are permitted for pointers and file buffers.

2. The text, page 78, incorrectly states that the argument is not needed for standard file input. However, omitting the argument generates compiler errors on the M68000 Pascal system. The "eof" and "eoln" functions need a factor in Motorola Pascal.

   Example: eof (input)

3. The "halt" statement, pages 313-314, is not a statement. It is a pre-defined standard procedure.

4. The examples on the bottom of pages 78 and 79 are incorrect for the sample data at the top of page 79. Both examples will cause "read past end of file".

5. The compiler can not pass parameters declared as "string" to a procedure. The problem can be avoided by generating a type whose simple type is string.

Examples (based on example on page 318 of Cherry's book):

```
   1(      -16) 0)— program duncan(input,output);
   2(      -98) 0)— var   instring : string[80];
   3(      -98) 0)—
   4(        0) 1)— procedure reverse (var s:string);
                                                    ^2,7
**Error—        0***
   5(       -4) 1)— var   lastch:string[1];
   6         1  1)A- begin
   7         2  1)—     if length(s) <= 1
   8         3  1)—        then s := s
   9            1)B-       else begin
  10         4  1)—          lastch := copy(s,length(s),1);
  11         5  1)—          s := delete(s,length(s),1);
  12         6  1)—          reverse(s);
                                    ^126
***Error—       4***
  13         7  1)—          s := insert(lastch,s,1)
  14            1)-B         end;
  15            1)-A   end;
  16            1)—
  17         8  0)A- begin
  18            0)B-    repeat
  19         9  0)—        writeln(' input string? ');
  20        10  0)—        readln(instring);
  21        11  0)—        writeln;  writeln;  writeln;
  22        14  0)—        reverse (instring);
                                    ^126
***Error—      12***
  23        15  0)—        writeln (instring)
  24        16  0)-B    until (instring = 'tiuq')
  25            0)-A end.
```

```
 1(    -16) 0)-- program duncan(input,output);
 2(    -16) 0)-- type  str = string[80];
 3(    -98) 0)-- var   instring : str;
 4(    -98  0)--
 5(      0) 1)-- procedure reverse (var s:str);
 6(      4) 1)-- var   lastch:string[1];
 7       1  1)A- begin
 8       2  1)--    if length(s) <= 1
 9       3  1)--       then s := s
10          1)B-       else begin
11       4  1)--          lastch := copy(s,length(s),1);
12       5  1)--          s := delete(s,length(s),1);
13       6  1)--          reverse(s);
14       7  1)--          s := insert(lastch,s,1)
15          1)-B          end;
16          1)-A    end;
17          1)--
18       8  0)A- begin
19          0)B-    repeat
20       9  0)--       writeln(' input string? ');
21      10  0)--       readln(instring);
22      11  0)--       writeln;  writeln;  writeln;
23      14  0)--       reverse (instring);
24      15  0)--       writeln (instring)
25      16  0)-B    until (instring = 'tiuq')
26          0)-A end.
```

6. The table of source program options in Cherry, pages 305-307, is superceded by those of Table 2-1 in M68KPASC, the M68000 Resident Pascal User's Manual.

7. Similarly, the listing error messages given in Cherry, Appendix G, are superceded by those in Appendix D in M68KPASC.