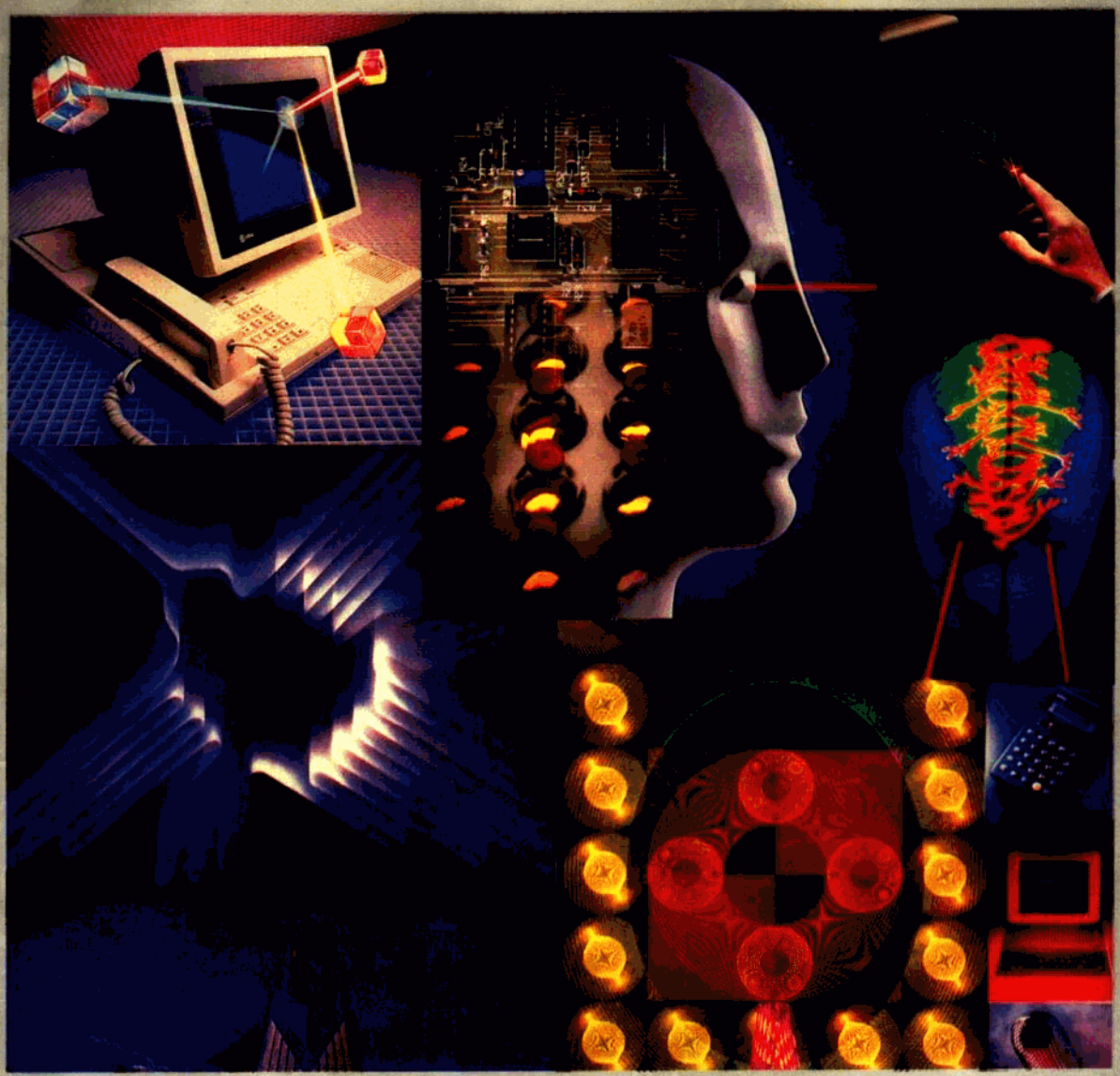
ONSPEC

3

技術叢書（共四册）

・控製軟件係統　・適用於 IBM PC 286・386計算機



HOPE 北京希望電腦公司

前　言

ONSPEC控制软件系统是美国HEURISTICS公司推出的工业控制软件系统，可在PC 286或PC 386等环境下运行。该控制软件是工业控制领域中应用得最广泛、最成熟的软件产品之一，因而受到用户的好评。

该软件系统包括以下三部份：

一、并发DOS系统

二、ONSPEC控制软件使用手册

三、ONSPEC—RPL实时编程语言使用手册

ONSPEC工业控制软件包在IBM PC 286，386及其兼容机上运行所需要的操作系统，不是单用户DOS操作系统，而是多用户多任务并发DOS操作系统（简称CDOS），CDOS支持大多数PC DOS和MS DOS的程序，因此它除具有DOS的全部功能以外，还具有一些更重要的特点，如多任务，多用户，多窗口，支持内存扩展，采用口令保护，菜单式用户界面等等。

为更有效地使用CDOS，由美国西雷公司提供《CDOS用户手册》，该手册由以下四部分组成：（1）CDOS安装指南；（2）CDOS用户指南；（3）CDOS参考指南；（4）CDOS运行应用指南。

参加编写这套资料的有顾良士高工，魏树铭副教授，韦众成副教授，金传祚高工。

白英彩教授审阅了该手稿並提出了许多有益的修改意见。

西雷公司驻沪办事处的夏明东先生为该书编写工作提供原稿並提出许多指导性建议。

借此机会向参加该书工作的所有朋友们致以衷心的感谢，並欢迎广大用户提出宝贵意见。

北京希望电脑公司

一九九一年六月

ONSPEC 控制软件

RPL 参考手册

目　录

1.0 RPL 程 序

1.1 程序结构

RPL(Real-time Programming Language实时编程语言)是块结构的语言。即将一个 或多个语句组成逻辑上相关的单元——块。每一块有一个块头、可选的说明和定义节、以及一组语句。在每个RPL程序中，最外一层块是主程序。

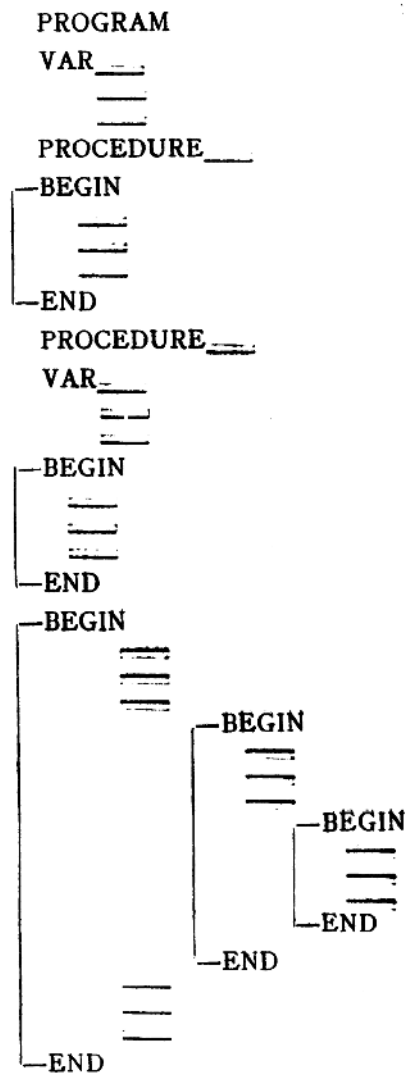


图 1—1 RPL的块结构

在程序中块可以嵌套，即可以在块中放置另一完整块，但两块不可重叠。过程和函数（见第6节）也可以嵌套。图1—1给出了RPL的典型块结构。程序1—1是一个有嵌套块的典型RPL程序。

```
PROGRAM FIRST_1;
CONST
     LIMIT = 10;
     MESSAGE = 'TESTING RPL';
VAR
     NAME : STRING;
PROCEDURE RESPOND (ST : STRING) ;
VAR
     I : INTEGER;
BEGIN
     FOR I : = 1 TO LIMIT DO
        BEGIN
           WRITELN(ST) ;
           ST : =CONCAT (', ST)  (*SHIFTS NAME TO RIGHT*)
        END
END;
BEGIN
     WRITELN (MESSAGE) ;
     WRITELN ('WHAT IS YOUR NAME? ') ;
     READLN (NAME) ;
     RESPOND (NAME) ;
     WRITELN ('FINISHED', MESSAGE)
END.
```

程序1—1　简单RPL程序

1.1.1 程序头

程序头形式如下：

PROGRAM〈程序名〉{〈程序参数〉}

〈程序名〉在程序中并没有什么特别之处，但此名在程序中不能再次被使用。可选项〈程序参数〉与其它版本的Pascal一样，在RPL中并无特别之处。

1.1.2 说明和定义

程序中的标识符在使用前必须先定义，除非此标识符是语言所预定义的。（见附录A）程序1—2是一个程序的说明和定义部分的实例，并例举说明了以下几种主要的说明类型。

1）LABEL标号说明

2）CONSTANT常数说明

3）TYPE类型定义

4）VAR变量说明

5）PROCEDURE和FUNCTION过程和函数定义

值得一提的是LABEL，CONSTANT，TYPE和VAR的说明次序可任定，且同一模块中每一种类型可出现多次。PROCEDURE和FUNCTION说明必须在最后，且每一模块中只能出现一次。

3.0节描述了各种数据类型的定义

```
LABEL
     34, 356, 755, 1000;
CONST
     TOP       =100;
     BOTTOM  =-TOP;
     LIMIT     =1.0E-16;
     MESSAGE=‘THANK YOU FOR NOT SMOKING’;
TYPE
     COLOR    = (RED, YELLOW, BLUE, GREEN, ORANGE) ;
     INDEX    =BOTTOM..TOP;
     PERPt    =^PERSON;
     PERSON   =RECORD
        NAME,
        ADDRESS   :STRING;
        PHONE     :STRING [8]
     END;
VAR
     COLR      :COLOR;
     I, J      :INTEGER;
     LIST      :ARRAY [INDEX] OF PERPT;
PROCEDURE ECHO (ST : STRING) ;
BEGIN
     WRITELN (ST, '', ST)
END;
```

程序1—2　说明和定义

1.1.3　语句体

在块中语句体的开始和结束由单词BEGIN和END标记，语句体可以有多个语句，也可以没有语句。对于主程序块，在单词END后必须跟句号。在语句体中，语句用分号分隔。

1.1.4　模块

一个模块是程序可以单独编译的部分，尔后可连接到主程序上。模块的一般形式与程序相同，其区别是没有主语句体。模块中可执行代码一定在过程和函数中。下面举例说明简单的单过程模块。

```
MODULE SIMPLE;
PROCEDURE MARK (CALL_NUM : INTEGER) ;
BEGIN
    WRITELN ('IN MODULE SIMPLE, CALLED FROM : ', CALL_
        NUM)
END;
MODEND
#
```

值得注意的是主语句体中的单词PROGRAM和#
在此用单词MODULE替代单词PROGRAM，用单词MODEND替代主语句体。
关于模块和模块化程序的更进一步的信息参考RPL编程指南。

1.2 作用域

RPL程序中的每一个标识符有其作用域。标识符的作用域是能正确访问此标识符的所有块的集合。标识符的作用域一般是从其定义开始直到块中的任何地方。

然而，如果嵌套块重新定义同一标识符，则里面的块不能访问外层块的变量，当同一标识符被多次定义时，总是里层的定义发生作用。

本手册使用术语全局的和局部的。程序中最外层的说明是全局说明。块中的说明局部于此块中。如果变量的说明在某一块中，则变量局部于此块。在嵌套块中，在蕴含块中说明的变量可以被嵌套块使用，但不是局部于此块的。在被蕴含块中，访问蕴含块中的变量叫做“上层访问”。

程序1—3中的程序包含嵌套块对同一标识符的多次定义。程序中的注释说明了在不同的地方哪些定义发生作用。

1.3 注　　释

在程序中空格可以出现的地方都可加载注释，编译程序会跳过这些注释。在程序中有两种方法写注释：

- 用花括号｛和｝将注释括起来。
- 用字符对（*和*）将注释括起来。

编译程序可以识别这两种不同的注释分隔符，因此注释可以嵌套。在程序中可以用一组注释分隔符作为一般注释，用另一组作为调试或开发代码节的注释符，下面给出示例程序段。

```
PROCEDURE WALKTREE (TREE : TREEPT) ;
BEGIN
    WITH TREE^DO
    BEGIN
        WALKTREE (LEFTREE) ; {PRE-ORDER WALK OF TREE }
        WRITELN (INFO.NAME) ;
```

```
(*****REMOVE THIS LINE FOR DIAGNOSTICS

WRITELN ('****IN WALKTREE****') ;
IF MARKED (NODE) THEN { LOOK FOR LOOPS IN TREE }
   BEGIN
      WRITELN ('LINK ERROR IN TREE') ;
      TREEDUMP (TREE, { WILL NOT RETURN }
   END
   ELSE
   MARK (NODE) ; { TREE OK SO FAR }

******REMOVE THIS LINE FOR DIAGNOSTICS*)

   WALKTREE (RIGHTREE)
END
END;
```

程序1—4 带注释的程序

2.0 标识符和常量

这一节描述RPL标识符，构成字符常量的规则，以及怎样定义命名常量。

2.1 标识符

RPL标识符可以代表变量、类型、常量、过程或函数，甚至整个程序。不论标识符代表何种对象，所有RPL标识符遵循相同规则。

RPL标识符的长度不限，只要能放在一行中即可。不过编译程序仅使用前8个字符区别不同的标识符。对于外部标识符只有前7个字符发生作用。

标识符可由字母、数字、下划线任意组合而成，但必须以字母开始，且不能包含空格。下划线和大小写被编译程序忽略。例如下面两个标识符被认为是同一标识符：

A_b_c和

abc

只要不用@编译可选项，在标识符中可用@作为第一个字母，但@不可用在标识符中。编译程序允许@字符是为了可以访问以@开头的标识符所命名的实时例行程序。然而如果选择@编译可选项，编译程序认为@字符是标准指针字符^，禁止@用于标识符。

下面是RPL标识符的合法实例：

```
X
@CPMRD
file_name
LA225prefix
Thisfile
Thisfile_for_91803_zip_only
```

编译程序不能区别最后两个标识符。下面是不合法的标识符实例：

```
X| 2      包含不合法字符
123x        以数字开头
program     使用保留字
STY@HM      @不是第一个字符
X␣22        包含空格
```

在程序中不能用象BEGIN和IF这样的保留字作为标识符，但可以使用象WRITELN和BOOLEAN这样的预定义的标识符来标识任意对象。在程序中预定义标识符是定义在全局层上一层的，因此在新定义的作用域中就无法访问预定义标识符所标识的原来的对象。

附录A列出了RPL保留字和预定义标识符。RPL编程指南列出了实时入口点名，以及外部标识符信息。

注意：如果不小心用实时入口点名作为外部标识符，程序可能不会被正确连接。

2.2 常　　量

可将常量看做文字值，或给常量命名，然后可在任何需要此值的地方用此名字。RPL常量可以是字符串、整数、实数或纯量类型。

2.2.1 数字文字

数字文字可以是十进制整数、十六进制整数、长整数或实数。常量的格式决定了其类型。

注：在8位版的RPL中没有长整数。

整数文字是－32768到32767范围内的任意整数。整数文字不能有小数点或逗号。十六进制数用 $起头。下面是合法整数文字实例：

－3456

$FFOO

32767

$EFFF

长整数常量必须用#字符'#'起头，对于负数将负号放在#号前，下面是长整数文字实例：

#6234343

#0

－#678988

实数文字可以是定点或浮点格式。在定点格式中，至少各有一个数字分别在小数点前后。浮点文字格式小数点可有可无，再接E，然后再接可选的有符号整数。两种格式都不能有空格或逗号。下面是合法实数文字的例子：

64.78E－13

－65.3

－33.677E＋10

在浮点格式中，E是10的幂。例

6.3E5

是6.3乘以10的5次方，或630000。

2.2.2 字符串文字

字符串文字可包含任意个可打印字符数，只要一行放得下，字符串文字用单括号括起来。单括号中以所有字符包括空格是字符串的一部分。在字符串中用两个单引号代表一个单括号。在字符串中大、小写字母是不同的。

下面是合法字符串文字实例：

'＊＊＊INVALID EDIT COMMAND＊＊＊'

'Steve''s Program'

若所需定义的字符串比一行长或字符串中有控制字符请用第6章介绍的字符串函数。

2.2.3 命名常量

常量定义指定一个标识符做为常量的同义词。可在能用文字的任何地方使用命名常量。
下面是常量定义节的实例：

```
CONST
  message = 'VERSION 3.3';
  size    = 100;
  limit   = -size;
  esc     = $1B;
  conv-fact = 3.27E-3;
  null-str = '';
```

注：RPL允许空串。

3.0 变量和数据类型

这一节描述RPL所提供的数据类型。两种常见的数据类型是：简单类型和结构类型。简单数据类型，也叫标量类型，每一数据项只有一个元素。整数、字符、指针都是简单类型。

对于结构类型每一数据项包含一个以上元素。记录、串和数组都是结构类型。在此节中不讨论文件，关于文件的有关信息见7.0节。

3.1 类型定义

编译程序根据类型定义来确定怎样为变量分配空间。块中的类型定义节为某一类型定义赋予名字，如下例：

```
TYPE
    NUMBERS = ARRAY [1..10] OF INTEGER;
    STRPT   = ^STRING;
    LETTER  = 'A' ··· 'Z';
```

3.2 变量说明

变量说明赋予变量类型，确立其作用域。在程序中所有变量必须先说明后使用。下面是块中变量说明节的实例：

```
VAR
    X,Y,Z  : INTEGER;
    NAMES : LIST;
    NUM1  : 0..200;
    NUM2  : 0..200;
```

在上例中对于某一类型定义可以定义一组对象名，而且不仅可以用类型名，也可用直接类型定义。

对于强类型检查的编译程序，为了使变量相容，必须把这些变量用同一类型名来说明。强类型检查要求兼容变量有完全相同的类型名，而不仅仅是相同的内部结构。在上例中按强类型检查的要求NUM1和NUM2是不兼容的。为了使之兼容，必须使用如下说明：

```
NUM1, NUM2 : 0..200;
```

关于编译程序的强类型检查的更多信息请见编程指南。

RPL支持绝对变量，即用绝对变量说明确保变量被存于某一特定单元。有关细节参考编程指南。

RPL也支持外部变量。即可以在一个模块中说明变量而在另一模块中引用它。

3.3 简单类型

RPL预定义了几种简单数据类型，见表8—1。除了实数类型外，所有这些类型都是纯

量类型。纯量类型是指其每一个可能值都是可数的。ASCII字符集就是纯量类型的一个例子。

对于纯量类型可以定义它的枚举或子界数据类型。一个枚举类型是指直接枚举纯量类型的所有值的集合。子界类型是某些纯量类型的一个连续部分。

表3—1 预定义的数据类型

数据类型	位长	范围
CHAR（字符）	1个8位	0到255
BOOLEAN（布尔）	1个8位	真或假
INTEGER（整数）	2个8位	-32768到32767
LONGINT（长整数）	4个8位	$2^{32}-1$到2^{-32}
BYTE（字节）	1个8位	0到255
WORD（字）	2个8位	0到65535
BCDREAL（二—十进制实数）	10个8位	见编程指南
FLOATINGREAL（浮点实数）	8个8位	见编程指南

RPL提供4种“伪函数”或类型转换运算符将一个简单类型转换成另一种。这些伪函数并不产生任何代码，仅指引编译器将下列8或16位数据项当作不同的类型。这4个伪函数是：

- CH(X)　将给出的表达式X的值当作ASCII码返回相应的字符。
- ORD(X)　返回表达式的序数值。字符的序数值是其ASCII的数值表示。
- ODD(X)　若表达式的值是奇数，返回布尔值真，否则返回布尔值假。
- WORD(X)　指示编译程序将给出的表达式当作宿主机字。

3.3.1 布尔类型

布尔类型有两个值：真和假。假的序数是0，真的序数是1。

一个布尔变量使用一个字节，即使在压缩结构（见3.4节）也是如此。

在此字节中，确定其值时只有最低位起作用。若此位被设置为1，则变量的值为真，否则其值为假。但是，逻辑运算使用整个字节。

3.3.2 字符类型

字符类型变量使用一个字节。字符的内部表示是字符的ASCII值。字符变量的范围是CHR(0)到CHR(255)。

在程序中表达字符值时，若字符是可打印的，用单引号将字符括起来，或使用CHR伪函数。用两个单引号表示单引号字符。

下面示例程序说明了CHR和ORD。

```
PROGRAM CHR_ORD;
VAR
    I, J   : INTEGER;
    C,D    : CHAR;
```

```
    BELL : CHAR;
BEGIN
    I : = 7;
    C : = '8';
    D : = CHR(I + ORD('0')); (* '0'的ASCII值是48 *)
    J : = ORD(C) - ORD('0');
    BELL : = CHR(7)
END
```

3.3.3 整数类型和长整数类型

整型变量为2字节长。整数的范围是－32768到32767，在0到255范围内的整型文字在代码中仅用一个字节。

长整型变量为4字节长，长整数的范围是2＼u－32＼d到2＼u32＼d－1。对于长整型文字只能用十进制，象普通的整型文字一样书写，但必须用字符#开头

例如

```
#6234343
```

可以定义长整型的子界类型，但不能用于数组的下标。

长整型和其它数据类型的转换有三个函数：

```
FUNCTION SHORT (L : LOGINT) : INTEGER;
FUNCTION LONG  (S : SHORT)  : LONGINT;
FUNCTION XLONG (S : SHORT)  : LONGINT;
```

短（short）数据类型是任意8或16位类型，例如字符型，布尔型，整型或字型。函数LONG把短位数的值扩展成长实数时填零，而函数XLONG将其值的符号位扩展到高位字中。关于整数和长整数数据类型的内部表示的特定信息参考编程指南。注意：在8位版的RPL中是没有长整数类型的。

3.3.4 实数类型

针对不同的应用RPL提供了两种不同的方法来处理实数。

- 针对商业领域应用的二进制编码的十进制。
- 针对科学计算和工程应用的二进制浮点数。

命令行上的可选项通告编译程序使用那一种格式。

实数的内部表示及其范围依处理器而定。关于实数的内部表示的细节参考编程指南。

下面是实数文字的实例，其解释见第2节。

```
212.3E－16
－22.454
2.0E＋4
```

3.3.5 字节类型和字类型

字节类型使用单个字节。对于表达式和赋值语句，字节类型与字符类型和整型兼容。字节类型可以是任意的位模式，在处理控制字符和字符运算中非常有效。字数据类型使用宿主