

第一课. 计算机的一般知识

1.1 电子计算机的发展及其基本结构

1.1.1 电子计算机的发展史

1945 年底,在美国首次研制成功人类第一台计算机,这台机器重 30 吨,占地面积达 167 平方米,加之它的工作原理,因此,人类后来的计算机并不是在这台机器的基础上发展起来的。

现代计算机理论的奠基人是图灵。在美国数学家冯·诺伊曼的主持下,1949 年诞生了第一台存储程序的计算机,又称第一代机,这台计算机为后来的计算机发展奠定了基础。

1959 年,第一台晶体管计算机问世,由此,计算机进入了第二代。1964 年,IBM 第一代 360 系列计算机问世,这是第一代通用计算机,为研制这种计算机,IBM 投资 50 亿美元,比二战期间美国政府投入到原子弹研究的钱(20 亿美元)还要多;由此,计算机进入了第三代。

进入到 80 年代以后,中大型计算机问世,于是开始了第四代计算机的时代。70 年代以后,出现了计算速度更快、存储量更大的巨型机。

70 年代微处理器的问世,标志着计算机的发展开始了又一场革命。1977 年 3 月苹果公司的个人用计算机问世,自此,计算机开始进入千家万户。

1.1.2 电子计算机的基本结构

计算机由硬件和软件组成。而硬件是由主机和外部设备组成;软件由系统软件和应用软件组成。

计算机硬件是软件的基础,是软件发挥功能的工作环境,而软件则是管理和利用硬件资源来实现计算机的功能,软件和硬件是相互促进和发展的。

硬件大体上有以下几个部分:控制器,运算器,内存储器(RAM、ROM),输入设备和输出设备。前三者又合称主机,后两者又称作外部设备。现在就这五个部分的功能,作一些简要说明。

1. 输入设备。输入设备是用来向主机输入原始数据和处理这些数据所使用的计算程序列的设备。输入设备的种类很多,但在微型机上不外乎下列几种:

(1) 终端键盘。利用手指击键方法向计算机输入信息。用户自己写的程序列化和准备处理的数据,都可由键盘上敲入。

(2) 磁盘。磁盘,实际上也是存储信息的,因为它们都是主机之外的设备,所以也称为外存储器。外存储器上的信息,也可以输入到机器中去。

(3) 模—数(A/D)转换器。它可以将连续变化的模拟量(如电压、电流、长度、角度等)转换为数字量,送入到机器中去。

此外,图形输入板、声音输入装置等,实际上是专用的模数转换器,同样可以为计算机输入信息。

2. 输出设备。输出设备是用来输出计算结果或其它信息的。常用的输出设备有:

(1) 显示终端。用以显示计算机的有关信息,占用户从键盘上敲入并为机器接收的字符、机器清单、机器向用户的提示,程序运行时的输给(包括数字、文字或图形)等。

(2) 打印机。显示终端上可显示的东西几乎都可以由打印机打印到纸上。

(3) 磁盘。可以用来存储程序和数据(包括数字、文字、图形和声音信息等)。

3. 主存储器,即内存储器(简称内存)。它用来存放原始数据、处理这些数据的程序以及计算结果(包括中间结果,也包括图形和声音信息等)。系统程序也存放在内存中。

内存储器分为一个个单元,好似一间间房子,并按顺序编了号码(从0号开始),通常又称为一个个地址。机器中的所有信息都以一定的规则存放在内存的一个个单元中。

对任何一个单元来说,它很象从左到右顺序安放的一排灯泡,每个灯泡代表一位数字:灯泡点亮代表1,灯泡熄灭代表0。于是这一排灯泡就可表示由0和1构成的一个数这排灯泡的个数,就称为位数或字长。目前,一般微型计算机的内存储器是用半导体器件组成的电路制成的,称为半导体存储器。字长,一般取决于微处理器的字长。

一般微型机的内存储器又分为两个部分。一部分是随机存储器(RAM),每个单元的数据是可以改变的,而且关机以后所有信息都会自动消失。这类存储器是用户可以使用的空间。另外,还有一部分是只读存储器(ROM),每个单元一信息是固化的,用户只可读使用,但无法使其改变。任何时候,只要接通电源,这些信息就建立好了。

4. 运算器。运算器是计算机进行信息加工的场所,所有算术运算逻辑运算等都在这里进行。就象用算盘做题时一样,它只能放当前被操作完的一个数据,中间结果一般配要送内存中保存起来,以备以后使用。所以,没有内存,单靠运算器是无所作为的。

5. 控制器。它是用来实现计算机各部分协调动作使计算过程自动进行的装置。也就是说,它是计算机内的指挥部。

控制器可以向计算机的其它部件发出信号,控制数据的传输与加工;同时,控制器也接收其它部件送来的信号,以便调整其控制功能。

所以,在计算机工作时,有两和种信息流:控制流与数据测定流,由控制流控制数据的传输与加工,完成所有的计算动作。

这里介绍的运算器、控制器、内存储器、输入输出设备等,都是一些看得见、摸得着的“硬”东西,所以又称它们为“硬件”或“硬设备”。相对地,那些在计算机工作过程中必不可少的数据以及对这些数据进行处理的控制命令等信息流都是摸不着的“软”东西,通常称之为“软件”或“软设备”。

易语言本身,属于软件的范畴。

1.2 计算机中数的表示

日常生活中,我们非常习惯使用十进制计数法,可是在日常生活里我们也还会遇到一些另的进制,如二进制(两只为一从)、十二进制(十二英寸为一英尺,十二个月为一年)、十六进制(中国老秤十六两为一斤)、二十四进制(二十四小时为一天)和六十进制(六十分为一小时,六十秒为一分等)等等。在计算机内部,则采用二进制计数法。

为什么计算机要使用二进制计数法呢?这是因为电气元件中两种状态最容易实现(如电路的通断、电位的高低等),也最为稳定,并且最容易实现对电路本身的控制。

在计算机里,一般以高电位代表1,低电位代表0。二进制的两个数基,就用0和1来表示。凡够2时,就向左进一位。比如十进制的2,用10表示;十进制的4,用100表示等。

用逻辑电路实现二进制数的运算,是极为方便的。

用不同进位制表示的数之间,可以根据一定的规则相互转换。

1.2.1 十六进制数、二进制数、八进制数

在计算机内部运算中常用的进位制有4种:

二进制：逢 2 进 1，由数字 0 和 1 组成，以下标 2 或后缀 B 表示。

八进制：逢 8 进 1，由数字 0 至 7 组成，以下标 8 或后缀 Q 表示。

十进制：逢 10 进 1，由数字 0 至 9 组成，以下标 10 后缀 D 表示，该后缀可以省略。

十六进制：逢 16 进 1，由数字 0 至 9 和字母 A 至 F 组成，以下标 16 或后缀 H 表示。

例如：二进制数 1001010 表示为 1001010 (B)、八进制数 234512 表示为 234512 (Q)、十六进制数 4523ADF 表示为 4523ADF (H)，十进制数的后缀可以省略。

用不同进位制表示的数之间，可以根据一定的规则相互转换。

1. 十六进制数、八进制数与二进制数之间的转换

一位十六进制数用四位二进制数表示，一位八进制数用 3 位二进制数表示。

二进制数转换为十六进制数时，以小数点位置为界，向两侧每四位分组，当两侧不足四位时补 0。例如：101010.010101 (B) = 0010 1010.0101 0100 (B) = 2A.54 (H)

二进制数转换为八进制数时，以小数点位置为界，向两侧每三位分组，当两侧不足三位时补 0。例如：101010.010101 (B) = 101, 010.010, 101 (B) = 52.25 (Q)

十六进制数转换为二进制数时，以小数点为界，每一位十六进制数转换为四位二进制数向两侧排列；八进制数转换为二进制数时，以小数点为界，每一位八进制数转换为三位二进制数向两侧排列。

1.2.2 十进制数和二进制数之间的转换

把一个十进制数转换为二进制数，方法如下：把这个十进制数反复地除以 2，直到商为零，所得的余数（从末一位读起）就是这个数的二进制表示。

如十进制的 11，反复用 2 除：

$$\begin{array}{r}
 2 \overline{) 11} \quad 1 \\
 2 \overline{) 5} \quad 1 \\
 2 \overline{) 2} \quad 0 \\
 2 \overline{) 1} \quad 1 \\
 \hline
 0
 \end{array}$$

用二进制表示，是 1011。

换句话说，把一个十进制数化成以 2 为底的指数形式，则它的系数（由高次到低次）就是其二进制表示的数。

像上面提到的十进制数 11，换成以 2 为底的指数形式为：

$$11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

把它的系数顺序排列起来，就是 1011，这就是它的二进制表示。

反之，一个二进制数它的十进制表示，就可以用

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$$

换言之，将一个二进制数转换成十进制数的方法是：将这个二进制数的最末一位乘以 2^0 ，倒数第 2 位乘以 2^1 ，……最后将各项相加即可。

1.2.3 十进制数与十六进制数的转换

在计算机内，所有的数都是用二进制表示的（电位的高低表示 1 或 0）。然而，如果让人们对于任何二进制数能象对十进制数那样一目了然，实在不是一件容易的事。这一方面由于

人们用二进制数终归没有用十进制数来得那么普遍与习惯；另一方面，用二进制表示一个数时，其位数较长（比如大于十进制 63 的数，用二进制表示至少要 6 位），且每位数只有 0 和 1 可资区别。

为此，人们引进了八进制和十六进制的表示方法。

必须指出，八进制或十六进制并不是机器中的某一位有八种状态或十六种状态，在机器内仍是以二进制为藉款础的（每一位只可有两种状态），只是为着输入或输出一个二进制数的方便，击机器内的二进制数从右向左每三位分作一组，每组便可表示 0-7 的某个数，显然最右一组满 8 时要向左邻一组进 1，于是这三位一组构成了八进制数。类似地，把二进制数从右到左每四位分作一组，每组表示 0-15 中的某个数，这一组一组就构成了十六进则数。目前用十六进制表示较为普遍。可以想见，把两个四位连在一起可以用两个十六进制数表示，其数值范围可为十进制数的 0 到 255，共 256 个数，选用它们来代表所有英文字符的内部编码（以使用二进制数代表字符）是足够使用了。因此又把两个十六进制数作为一个基本单位，称作“字节”。

十六进制数的表示法，0-9 仍沿用十进制中的 0-9，接下去的六个数依次用 A、B、C、D、E、F 表示。

因此，十进制的 0-16 表示成二进制数和十六进制数。如下表所示。

十进制表示	二进制表示	十六进制表示
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

十进制数和十六进制数之间的转换，类似于十进制数和二进制数之间的转换。

把一个十进制数反复地除以 16，直到商为 0，将所有余数（从末一个起）顺序排列起来，就是这个数的十六进制表示。

如，十进制的 269

$$\begin{array}{r}
 16 \overline{) 269} \quad 13 \text{ (D)} \\
 \underline{16} \\
 16 \overline{) 16} \quad 0 \\
 \underline{16} \\
 16 \overline{) 1} \quad 1 \\
 \underline{16} \\
 0
 \end{array}$$

转换为十六进制数为 10D，反之，把一个十六进制数的末位乘以 16^0 ，倒数第二位乘以 16^1 …再将各项累加，所得的数就是那个十六进制数的十进制表示。

如：

$$10D = 1 \times 16^2 + 0 \times 16^1 + 13 \times 16^0 = 269$$

关于八进制数及其与十进制数之间的转换，不再叙述，请读者自己考虑。

1.3 计算机语言

人们进行思维活动或人与人之间交流思想，要通过语言，计算机进行或人与计算机交流信息，也需要通过语言。这就是计算机语言。计算机语言是用来指明让计算机依次做些什么事情的，所以又称为程序设计语言，这种语言有其自身的特点和发展过程。

1.3.1 机器语言

前文提到，计算机的指挥中心是控制器，且计算机中的数都是用二进制数表示的。实际上，控制器也是用二进制数的 0 或 1（即低电平、高电平）来实现其控制功能的。

原来，任何一种电子计算机，都是一套指令系统，由若干条指令组成，每条指令都可指挥计算机实现某些功能。

一条指令，通常由操作码和操作数两部分构成。操作码由若干位二进制数组成，由于这若干位 0 和 1（即低电位、高电位）的不同组合，因而能使计算机产生不同的。操作数部分也是由若干位二进制数构成，它指明被操作的对象。一般地，它可以是被操作数的本身，也可被操作数在内存中存放的地址，甚至不指明操作数部分（有时是隐含的）。如果操作数部分指明的是地址，不同类型的机器待，又可能只指明一个地址（被操作数地址）、或指明两个地址（两个操作数的地址，或一个被操作数地址，一个操作结果存放地址），或指明三个地址（被操作的两个数的地址和操作结果存放地址）等等，它们分别被称为一地址、二地址或三地址指令。

控制器每遇到一个操作码，就会产生不同高低电位的组合，控制机器作相应的动作。地址部分的不同高低电位组合，又会指向相应的被操作对象。从而完成一条指令的功能。一条条指令执行完结，运算就告结束。

由于指令的操作码会直接使机器产生相应动作，操作数部分会直接指向被操作对象，所以用一条条指令写成的机器语言程序，能为机器直接认识。换之，机器语言是计算机能懂得的唯一语言。

用机器语言写的程序，在机器上运行时，速度是最快的。

值得说明的是，上述例子中，由七个连续单元中；十六进制数的 20 号单元中也得事先送入一个数；还有，结果存放的地址也是用户指定的。这就是说，内存中什么地方放程序，

什么地方放被操作的数，什么地方放结果，得由用户自行分配。短小的程序比较好办，长而复杂的程序，分配内存时得格外小心，以防止搞乱了。

显然，用机器语言编制计算程序，对用户来说有着很大的缺陷：

1. 难写、难读、难修改；
2. 由于每种机器的指令系统不一样，这种程序没有通用性；
3. 需要人工分配内存。

1.3.2 汇编语言

为着克服机器语言固有的缺陷，人们经过研究探讨，引进了一种汇编语言。

汇编语言实际是一种符号语言，它把人们难以记忆和辨认的操作码改用有意义的英文单词(或见长缩写)作为助记符来代替，而对操作数部分稍加改动，更易于理解。

然而，从机器语言到汇编语言是花了一定代价的。因为计算机只认识机器语言，为了让计算机能认识汇编语言，最简单的办法是在机器内部编制一个对照表，像一本字典一样，对任何汇编符号都能从这个对照表中查到相应的机器语言的指令操作码。进行这种对照查找，当然也得有一个用机器语言写的(以便机器能认识的)程序，即汇编程序。机器在执行这个汇编程序时会把一条条汇编语言翻译成机器语言，从而又为机器所能认识。

可见，执行汇编程序得额外花费机器运行时间，汇编程序和对照表也要占用内存空间。因而由机器语言改用汇编语言，是以降低机器运行速度和减小用户可用内存空间为代价的。

汇编语言可以克服机器语言的第一个缺陷，但其缺陷却依然存在。

1.3.3 算法语言

算法语言又称“高级语言”，它比较完满地解决了机器语言的所有缺陷，是程序设计语言的一大突破。

高级语言编写出的程序，称为源程序，而更是计算机不能直接认识的东西。让计算机认识并能执行高级语言源程序，需要一个翻译和执行的过程，它比汇编程序功能要强得多。

我们通常用高级语言写好一个程序，让机器运行它，这种说法并不十分确切。拿易语言写的程序来说，真正运行的并不是易语言的源程序，而是由易语言编译器将源程序编译后的机器码，从而完成了源程序所要做的一切。然而它还是顺着易语言程序一句句执行的。不同程序语句有不同方法和效果，所以我们以后还是说成运行(或执行)易语言源程序。

不言而喻，高级语言的优点是以降低运行速度和减小用户可用内存空间为代价的。运行速度大约只及机器语言的数百分之一。随着大规模集成电路技术的飞速发展，指令技术周期越来越短，存储器越来越便宜，这些代价也就显得不那么重要了。当然，对某些程序来说，降低运行速度可能是致命性的，所以近年来不少应用程序采用高级语言与机器语言相结合的办法，不是没有道理的。

以上提到的汇编程序、解释程序、编译程序以及操作系统、服务性程序、库程序等，一般称为系统程序；为解决特定问题而编写的程序，一般称为应用。这些都是属于软件的范畴。

1.4 习题

1. 将下列二进制数转换成十进制数：

- (1) 101 (2) 1100 (3) 11011001 (4) 101101100110

2. 将下列十进制数转换成二进制数：

(1) 26 (2) 54 (3) 129 (4) 25560

3. 把下列十进制数转换为十六进制数：

(1) 15 (2) 287 (3) 6438 (4) 39684

4. 把下列十六进制数转换成十进制数：

(1) B (2) 1C (3) A93 (4) FFFF

5. 把下列二进制数转换为十六进制数：

(1) 10100110 (2) 11000110 (3) 1011001110 (4) 101001011111010

6. 把下列十六进制数转换为二进制数：

(1) 8F (2) 3A4 (3) 6D35 (4) FFFF

第二课. 易语言的基本概念

本节将向读者介绍什么是易语言，构成易语言程序有那些最基本的东西。正象盖房子一样，这里仅仅是准备钢筋，水泥，砖，瓦，沙，石等材料。同时在本章中我们还要介绍一些易语言常用的命令和常规的上机操作方法。

2.1 易语言的特点

1. 全可视化

一般的可视化编程语言，仅支持图形用户界面的可视化设计操作，而易语言除了支持此类可视化操作，还支持程序流程的即时可视化呈视，极大地减少了程序录入错误。即：用户在编写程序的过程中，可以即时看到当前程序的运行流程及路线，有助于培养编程思路，提高解决编程问题的能力。

2. 全中文

作为一款由中国人自己开发的编程语言，易语言在中文处理方面有良好的支持。用户在编写程序的过程中，可以不接触任何英文。根本不懂英文的人使用中文编写代码没有任何障碍。

(1) 中文名称的快速录入。易语言内置四种名称输入法：首拼、全拼、双拼、英文。三种拼音输入法均全面支持南方音。使用这些输入法能够极大地提高中文代码的输入速度。

直接使用系统提供的输入法，如五笔字型、智能 ABC、紫光拼音、自然码等，同样可以进行程序的输入。

(2) 程序全部以中文方式显示，运算符号全部显示为对应的中文符号 ($\geq \leq \neq \approx \times \div$)，日期时间以中文格式呈现 (年月日时分秒)，以便于中文用户理解、阅读程序。

(3) 适合中国人的语言、思维习惯。对其它计算机编程语言的学习，总会感到某种限制，首先是语言环境的限制，有很多专业的术语字面上很难理解它的含义，而以中文编写出的程序代码，符合中国人的语法习惯和逻辑思维，可以做到见文思义，更加适合中国人使用。在以后复查程序时可以非常直观地分析；给其他人源代码学习时也会非常简单，相互交流变得更加容易。

易语言更提供了中文格式日期时间处理、汉字发音处理、全半角字符处理、人民币金额处理等功能支持。

3. 全编译与跨平台

易语言拥有自己独立的高质量编译器，中文源代码被直接编译为目的机器的 CPU 指令，高效且不存在任何速度瓶颈和安全隐患。

易语言现已同时支持 Windows 和 Linux 程序开发，不再依赖特定的操作系统。

4. 可扩充支持库

易语言由基本系统和运行支持库两部分组成,两者之间通过使用易语言自行定义的接口技术进行协作。运行支持库内提供了易语言的所有语言要素,如:命令、窗口和报表单元数据类型、普通数据类型、常量等等。可以通过安装外部支持库来扩充易语言基本系统。运行支持库还可以被随意增减、抽换或升级,基本系统对运行支持库提供了详细的版本控制。本技术给用户带来的最大好处是:

(1) 用户可以根据行业或自身需要定制易语言;

(2) 由于运行支持库的不断增多、升级,易语言的功能将被迅速扩充;

(3) 由于运行支持库可以仅包含声明而不包含实际的运行支持代码,并且可以随时被更新或抽换,这样可使人们通过国际互连网与服务器进行远程易语言交流(例如复杂型电子商务、远程控制等等)成为可能,这也是以后易语言互联网版本的发展方向。

5. 数据库支持

易语言相对其它编程语言的优势还在于易语言拥有自己的易数据库,并且用中文命令操作易数据库,简单方便。同时,易语言对外部数据库也有着非常好的支持,通过简单的组件和命令就可以实现易语言与各类数据库的连接,如 Oracle、MySQL、SQL Server、Access 等等。

6. OCX 组件、类型库 (TypeLib)、API 与 COM 对象

易语言可直接在程序中引用多种现有编程资源,极大的扩充了易语言的功能,并可对这些英文资源进行汉化处理,从而能够保持全中文的特点,让用户不用学习英文也能充分使用这些英文资源。

7. 与其它编程语言相互融合、互相补充

易语言支持当今先进的编程理念,例如面向对象编程、事件消息处理机制等,了解、学习和掌握易语言对掌握其它编程语言具有桥梁作用,同时,易语言可以和其它编程语言以标准 Win32 DLL 方式互相调用,保障了多种编程语言协同开发的需要。

8. 即时编译并自动规范语句格式的录入方式

在输入程序过程中,每条程序语句录入后,当光标离开该行,则对该行立即进行初步分析编译。如果该行输入正确,则该行的拼音简写会变成对应的汉字变量名或组件名,并呈现统一的字体间距和格式,因此任何人所编写的任何程序其格式都完全一致,这对于应用程序的协作开发、交流和维护非常有利。

9. 系统内置的自动名称管理器能够对用户所定义的各类名称进行跟踪管理

例如:假设程序中现存在一个名为“刷新内容”的子程序,而且在很多地方都调用了该子程序。现在用户根据需要想把该子程序更改为另外一个名称,在传统的编程语言中,用户更改子程序名称后,要搜寻整个应用程序,逐一找到使用了该子程序的地方,把名称相应地改变过来。在易语言中,用户只需更改该子程序名称,程序中其它所有使用了该子程序的地方,其名称都将被自动更改过来。

10. 贯穿全程的即时且全面的信息帮助

用户在进行任何操作的过程中,随时按 F1 帮助键,均能够在状态行上或提示夹中获得有关当前操作位置的详细相关信息。例如:用户将光标移动到某程序行上,然后按下 F1 键,马上就能够得到此程序行上所有命令的定义、参数、使用方法、所隶属的支持库等信息。

2.2 易语言程序的构成

下面对易程序的结构进行介绍。首先你的易程序需要有一个显示界面,一般是使用一个

窗口（也可以是控制台程序）作为启动画面，易语言中指定“_启动窗口”这个窗口是首先弹出的窗口，大家可以在这个窗口中放上其它的组件，以显示信息或美化程序界面。窗口显示时会有一系列的触发事件，如“创建完毕”事件、“尺寸被改变”事件等，但可能大家没有用到这些事件，因此不会进行任何的动作，只是显示一个窗口。若大家使用到了这些事件，就会形成事件子程序，这样就产生了子程序，而子程序是放在程序集中进行组织的，而每一个窗口对应一个程序集，大家也可以自己创建自己的程序集，程序集包含若干个子程序，子程序内输入程序代码。而程序代码就是各种命令和方法。

为配合命令书写，需要有存放内容的变量，为方便引用，可以建立常量，自定义数据类型，甚至可以建立图片或声音资源供引用。为了调用系统应用程序接口 API，使用更多的功能，需要进行 DLL 声明。这些操作可以在“程序面板”中完成。

为了重复利用程序资源，不必每次都重新写某段代码，除提供自定义子程序外，还提供易模块，供其它程序调用，也可以写标准动态链接库，供易语言及其他语言调用。

为了理解上述的程序结构，下面跟着本书写第一个易程序。

先从一个简单的例子说起。

假定某甲有人民币 15 元，某乙有人民币 20.5 元，求甲乙二人共有人民币多少元？

把这个问题写成易语言“Windows 控制台程序”，可以是下面的样子：

子程序名	返回值类型	公开	备注
_启动子程序	整数型		本子程序在程序启动后最先执行

变量名	类型	静态	数组	备注
甲	小数型			
乙	小数型			
和	小数型			

```

甲 = 15
乙 = 20.5
和 = 甲 + 乙
» 标准输出 ( 到文本 (和))
标准输入 ()
返回 (0) ' 可以根据您的需要返回任意数值
  
```

计算机在执行这个程序时，先让“甲”取得值 15，再让“乙”取得值 20.5，然后让“和”取得“甲”与“乙”之和，即 35.5；再把“和”的值显示出来。

例程中，几个数字，如 15、20.5，几个中文，如甲、乙、和。这些数字和中文，分别叫做常量和变量。

2.3 易语言界面、菜单介绍以及上机操作

2.2.1 易语言的界面

初次运行易语言后，首先会弹出对话框，询问创建何种类型的易程序。如下图所示。

若打开易语言界面后未新建程序，也可以通过菜单“程序”→“新建”来创建新的易程序。或点击窗口工具条中的新建按钮来新建易程序。



易语言启动对话框

易语言可创建以下 6 种程序:

1. **Windows 窗口程序:** 是支持在 Windows 下弹出窗口及组件等标准 WIN32 位程序, 也称易程序。
2. **Windows 控制台程序:** 是 WIN32 位无窗口界面的易程序。一般用于服务器等。
3. **Windows 动态链接库:** 可以生成 DLL 程序, 将在本书关于 DLL 章节中介绍。
4. **Windows 易语言模块:** 简称易模块, 是经过初步编译后的程序模块, 供其他程序重复调用, 将在本书关于模块章节中介绍。
5. **Linux 控制台程序:** 是支持 Linux 操作系统的无窗口命令行程序, 将在本书关于 Linux 程序章节中介绍。

6. **Linux 易语言模块:** 是支持 Linux 操作系统且经过初步编译后的程序模块。

选择“Windows 窗口程序”, 点击“确定”按钮, 就会创建一个相应的标准的 Windows 窗口程序, 并可以看到易语言的主界面。

易语言主界面的最上方是标题栏, 显示易语言系统当前打开的程序名称, 当前所支持的操作系统, 以及当前设计窗口名称。标题栏下方是菜单栏, 有易语言的常用菜单。菜单栏下方是快捷命令按钮工具条, 一些常用的操作都可以通过点击这些工具条中的按钮实现。

主界面的左边是易语言的工作夹, 其中有 3 个面板, 分别是“**支持库面板**”、“**程序面板**”和“**属性面板**”。

“**支持库面板**”的作用是: 显示支持库列表, 展开查看各支持库提供的命令、数据类型等信息。在程序编辑状态下, 可以通过双击此面板中的某个命令, 将其直接填充到光标处。若有窗口组件的方法也可以在这个列表中查看方法的用处。将光标移至某支持库根部, 按下 F1 后可查看此支持库的介绍信息。

“**程序面板**”的作用是: 相当于一个组织机构, 可以添加窗口, 或加载全局变量、常量、资源、DLL 命令申明、自定义数据类型等。也可用来在程序各操作界面间进行切换, 例如

可以直接找到某个创建的窗口中，或快速找到某个子程序，

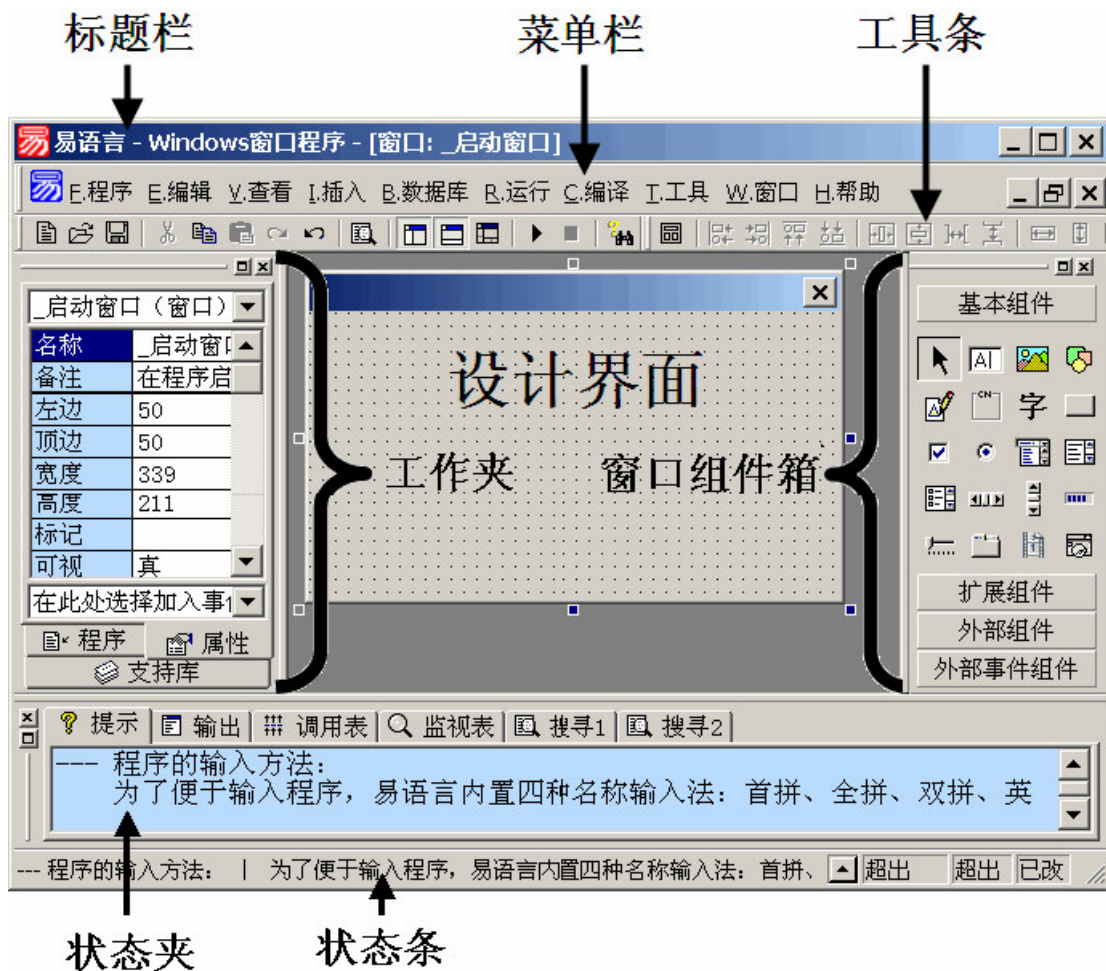
“属性面板”的作用是：属性表可查看和更改已添加组件的属性、组件列表列出所有组件并可快速选择所需组件，事件列表可生成此组件的事件子程序。

最右边是易语言的组件箱，里面列出了易语言提供的所有组件。分为四栏，“基本组件”栏可显示易语言最基本常用的组件，即核心支持库内的组件，在本书基本组件章节中进行介绍。“扩展组件”包含扩展支持库内的组件，在本书的后面有一些介绍。“外部组件”包含COM包装支持库所封装的ActiveX组件，此组件也称OCX组件。“外部事件组件”包含COM包装支持库所封装的COM事件组件。

主界面中间是设计区，在窗口设计时可自由向窗口中添加组件，进行程序界面设计；在程序代码编辑状态下可录入、修改程序代码。切换这两个工作状态可通过“窗口”菜单或“程序面板”等实现。

最下方是易语言的状态夹，可以查看帮助信息，查看调试文本等等。

易语言系统界面如下图所示。



易语言主界面

2.2.2 菜单项目详解

下面以易语言默认创建的“Windows 窗口程序”介绍各菜单项目的功能。

首先说明的是，菜单栏中各项目名称以及子项目菜单名称前面带有下划线的字母是此项菜单的快捷键。只要项目上的文字未变灰，就可使用此项菜单的功能。其使用方法是：点

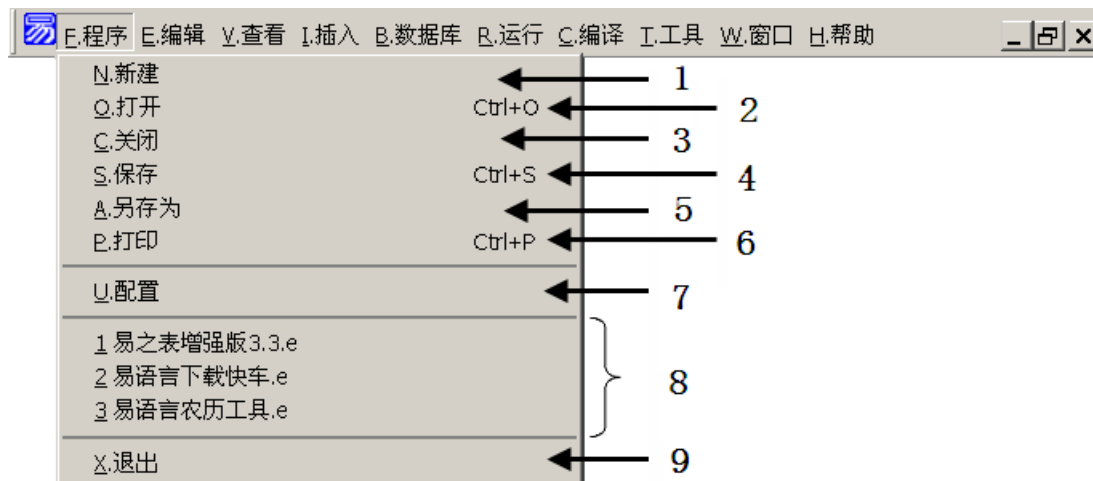
击键盘上的 **Alt** 键，易语言系统菜单栏中的第一个项“程序”被选中；点击菜单栏中任意一项名称前面的字母对应的键盘上的按键，此项菜单被弹出；最后点击子项目名称前面的字母，即可实现菜单功能。

而子项目菜单名称后面的提示是此项功能的快捷方式，比起快捷键来说，使用更方便快捷。如：**Ctrl + C** 键的功能是将所选内容复制到系统粘贴板中，方法就是同时按下键盘上的两个键或先按下 **Ctrl** 键然后再按下 **C** 键，而不需要弹出菜单。

在实际操作中，快捷键和组合键的使用会大大提高菜单功能的实现速度，减少鼠标的重复动作。

1. 程序

此菜单的功能是对程序文件项目的操作。



(1) 新建：建立新程序。

弹出标题为“新建：”的对话框，选择创建不同类型的程序。前面已经介绍过，这里就不多讲了。

(2) 打开：打开一个现有程序。

弹出标题为“请选择易程序文件：”的打开文件对话框，选择后缀为“*.e”的程序文件。

(3) 关闭：关闭当前程序。

关闭后“易语言”的程序设计窗口将被置空。

(4) 保存：保存当前程序。

新建程序没有保存过，将弹出“保存为：”对话框，提示编辑者选择程序的保存位置和程序的名称，以后此程序将默认这个保存位置，不会再弹出提示。

(5) 另存为：将当前程序以一个新文件名保存。

将弹出“另存为：”对话框，提示编辑者选择程序的新的保存位置或输入程序的新的名称进行保存，同时将位置指向新保存的程序。

(6) 打印：打印当前编辑窗口中的源程序。

使用打印机打印当前窗口中的源代码。

(7) 配置：配置本程序的环境及作者信息。

弹出“程序配置对话框”设置程序信息。该对话框可以将程序名称、程序备注、作者信息等信息保存在生成后的 EXE 文件中，当查看此 EXE 文件的属性时，这些信息会显示出来。并且可以在这里为自己的程序设置图标。

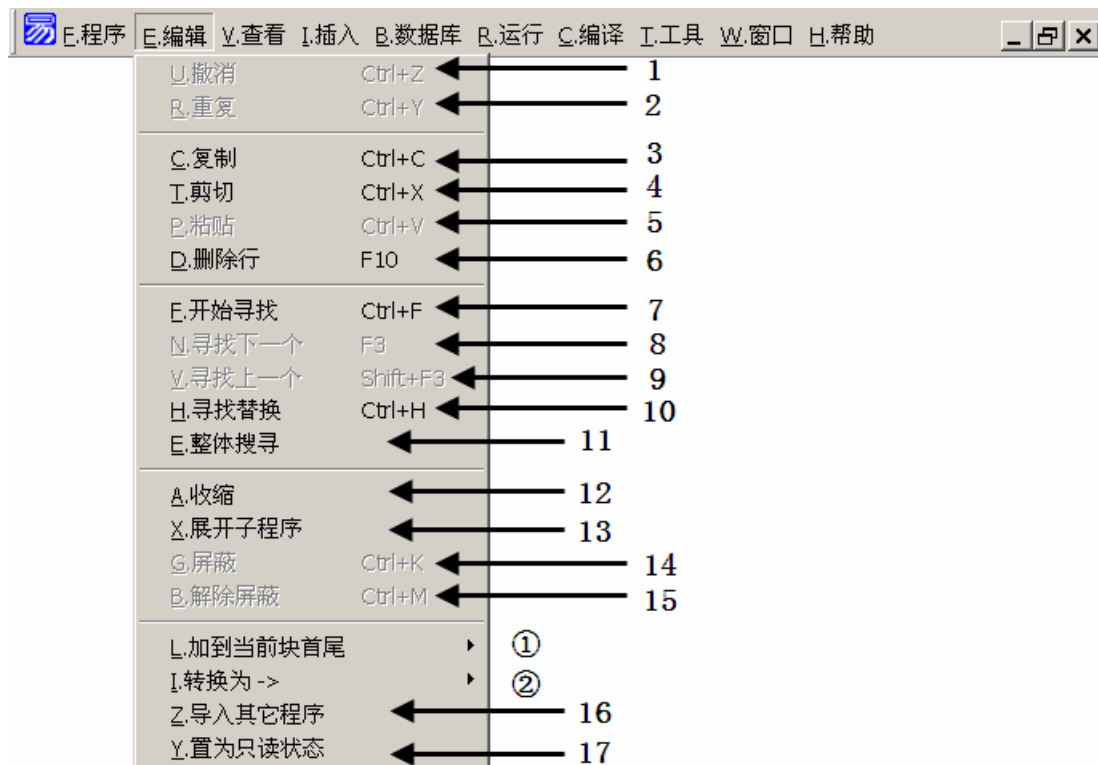
(8) 最近打开的程序，可用鼠标左键单击打开被选择程序，同时原有程序被关闭。

(9) 退出：退出系统；提示保存文档。

被更改过或未被保存过的程序，将弹出信息框提示编辑者保存程序，然后退出系统。

2. 编辑

此菜单中的项目可完成程序代码的即时编辑功能，多在代码设计区被选中时有效。



(1) 撤消：撤消最后一步操作。

一步一步撤消自创建或打开程序后对程序的修改。

(2) 重复：重新执行先前已撤消的操作。

一步一步还原自程序被创建或打开后的撤消的操作。

(3) 复制：复制被选块并将其置于粘贴板上。

复制被选中代码或窗体、窗体组件到粘贴板，其原有内容不会改变。

(4) 剪切：剪切被选块并将其置于粘贴板上。

相当于将被选中代码或窗体、窗体组件移动到粘贴板中，其原有内容被删除。

(5) 粘贴：插入粘贴板内容。

将粘贴板的内容插到程序中。其内容是程序代码，需要在代码设计区中进行插入；内容是窗体组件，需要选中窗体才能插入；如果是窗体，只需激活易语言系统，便可以将窗体插入到程序中。

(6) 删除行：删除当前所选择的块或光标当前所在的行。

(7) 开始寻找：开始在程序中寻找指定文本。

弹出“寻找对话框”，请求输入被寻找的文本。其寻找范围为当前程序集。

(8) 寻找下一个：在程序中寻找下一个指定文本。

以光标或已寻找到的文本为界，向代码下方寻找。其寻找范围为当前程序集。

(9) 寻找上一个：在程序中寻找上一个指定文本。

以光标或已寻找到的文本为界，向代码上方寻找。其寻找范围为当前程序集。

(10) 寻找替换：在程序中寻找替换指定的文本。

弹出“寻找替换对话框”，提示输入被替换和替换成的文本。以光标或已寻找到的文本为界，向下寻找或替换文本，也可以将当前程序集中所有找到的指定文本进行替换。

(11) 整体搜寻：在程序中寻找指定文本并列出所有找到的项目。

在全局中寻找指定文本，包括常量数据表、数据类型表等所有在代码设计区中以文本形式存在的指定项目。

注：以上 5 项功能遇到收缩的子程序时将跳过，不进入其内部寻找，忽略其中包括的指定文本。

(12) 收缩：将当前子程序或块内的所有语句收缩显示。

(13) 展开子程序：将当前选中块内的所有被收缩子程序展开显示。

(14) 屏蔽：屏蔽当前所选中的代码块。

把所选代码行或代码段设置为草稿，在调试和运行程序时不被执行。

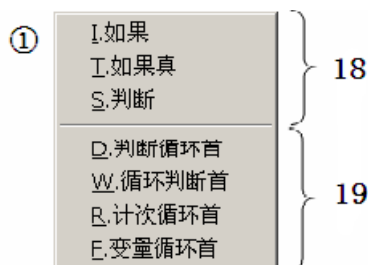
(15) 解除屏蔽：解除屏蔽当前所选中的代码块。

把草稿行或被屏蔽的代码设置为可执行代码。

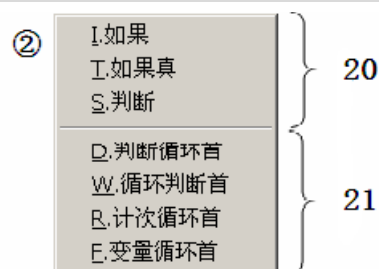
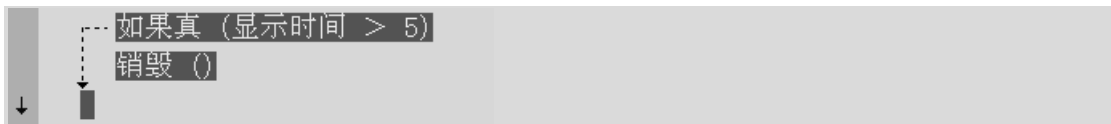
(16) 导入其它程序：将其它易程序中的内容全部导入到本程序中。

打开标题为“请选择易程序文件”对话框，选择程序文件插入到当前程序中。被导入程序的“_启动窗口”以及其它与当前程序重复的程序集名称、窗口名称后面将按顺序被加入数字加以区别。

(17) 置为只读状态：设置为只读状态后将不允许所有修改操作的发生。



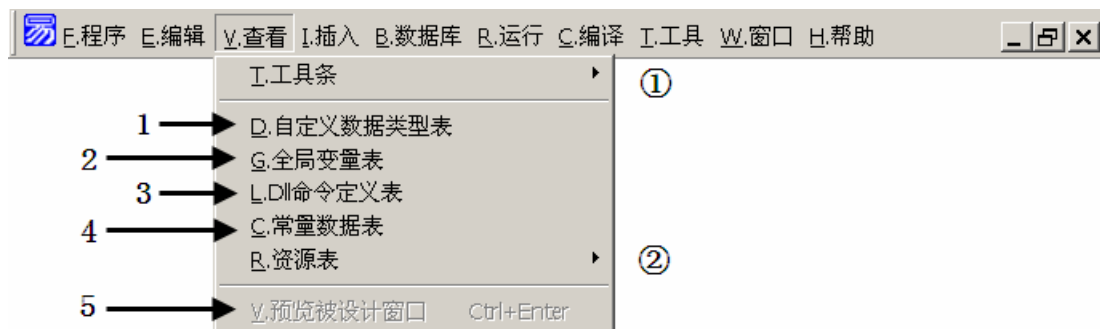
(18) (19) 添加一个新的流程控制命令，并将被选择代码块放到此命令中。要使菜单功能有效，选择代码块的方法是：选择两行或两行以上的单行代码；选择一个或多个分支流程控制命令，必须把流程线外的一行选中，如下图。其它选中方法无法实现。



(20) (21) 把选中的流程控制命令转换为别的流程控制命令，它们之间可以互相转换。但要注意的是，在转换过程中，原来的程序流向有可能发生改变。

3. 查看

显示各功能工具和程序相关资源定义表。



(1) 查看自定义数据类型表

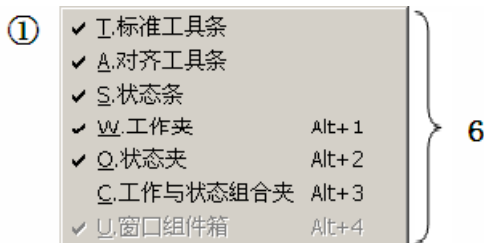
(2) 查看全局变量表

(3) 查看 Dll 命令定义表

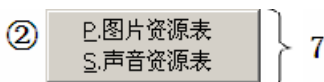
(4) 查看常量数据表

以上四项菜单的功能相同，都是将所选窗口放置到设计区的顶层。它们与工作夹中程序面板中的各项相互对应。

(5) 预览现行窗口，按 Esc 键退出预览
程序不能在此窗口中进行调试。



(6) 在易语言界面上显示和隐藏各功能工具。



(7) 与工作夹中程序面板中的相对项目功能相同。

4. 插入

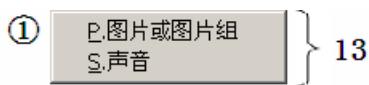
将所选项目自动插入到对应的设计区中，由编辑者按系统给定的格式填写代码。



(1) 根据现行编辑窗口的性质插入一个新子程序/数据类型/全局变量/Dll 命令/常量/资

源到当前位置。

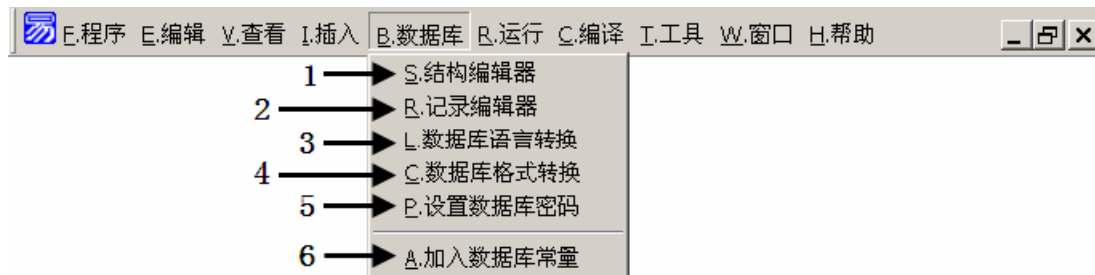
- (2) 插入一个新类模块。
- (3) 插入一个新程序集。
- (4) 插入一个新的子程序到当前位置的后面。
- (5) 插入一个新窗口。
- (6) 插入一个新的自定义数据类型到数据类型表。
- (7) 插入一个新的全局变量到全局变量表。
- (8) 插入一个新的 Dll 命令到 Dll 命令表。
- (9) 插入一个新的常量到常量数据表。
- (10) 插入一个新参数到程序中当前命令或子程序调用的参数表。
- (11) 插入一个新局部变量到子程序局部变量表。
- (12) 在当前编辑光标位置处插入所选择文件的全路径名称。



(13) 向资源表中添加数据资源，如：文本文件、声音图片文件以及其它类型文件。

5. 数据库

使用易语言系统提供的数据库工具，创建、编辑、转换、设置数据库相关内容。



(1) 浏览或修改指定数据库的结构。本功能由 dbmanger.e 编译后的程序提供，用户可以根据需要进行修改。

(2) 浏览或修改指定数据库的记录。本功能由 dbmanger.e 编译后的程序提供，用户可以根据需要进行修改。

(3) 将数据库中数据所使用的语言转换到另外一种。

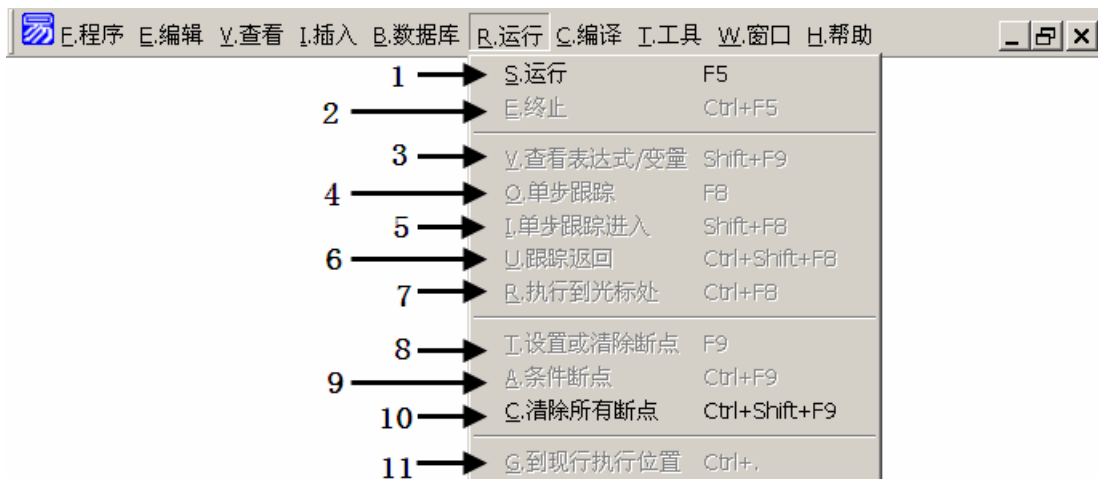
(4) 可以将其它类型的数据库通过 ODBC 转换为易数据库。

(5) 设置指定数据库的访问密码。

(6) 将指定数据库的名称及所有字段名作为文本常量加入到系统常量表，以便在程序中使用。

6. 运行

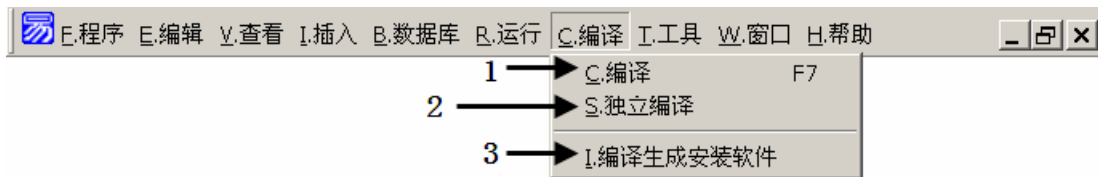
程序调试工具菜单。提供多种功能，用来查看、跟踪、挂起程序。具体使用方法请参见第七课。



- (1) 编译现行易程序的调试版本.EXE 文件后立即运行。
- (2) 终止现行易程序的运行。
- (3) 查看/修改指定表达式或变量的内容。
- (4) 在程序现行运行位置单步执行一程序行，如果此程序调用了子程序，系统不会跟踪到该子程序中去。
- (5) 在程序现行运行位置单步执行一程序行，如果此程序行调用了子程序，则跟踪进入子程序。
- (6) 在上级子程序调用现行子程序的语句后中断。
- (7) 运行易程序，在当前光标所处程序行处中断。
- (8) 设置或清除当前程序行处的断点。
- (9) 设置或修改当前程序行处的条件断点。
- (10) 清除掉程序中的所有断点。
- (11) 跳到现行即将被执行语句的位置。

7. 编译

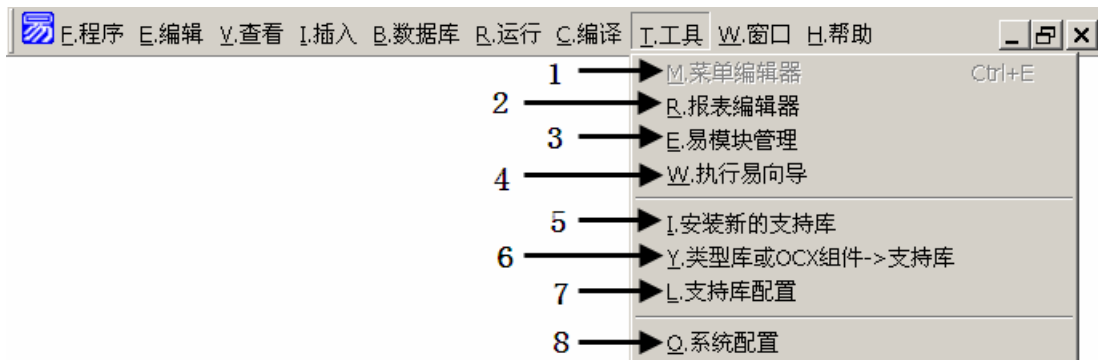
对已注册用户，提供三种不同的方法，将源代码创建成可执行的 EXE 程序文件。



- (1) 编译现行易程序的最终发布版本，创建对应的.EXE 可执行程序文件。
- (2) 编译出可执行 EXE 文件，该文件不依赖任何易语言系统文件，可以在未安装易语言系统的电脑上运行。
- (3) 制作当前易语言程序的安装软件，该软件不依赖任何易语言系统文件，可以在未安装易语言系统的电脑上运行并安装指定易语言程序。

8. 工具

易语言系统提供的多种附加工具，可用来管理和配置易语言的扩展功能。

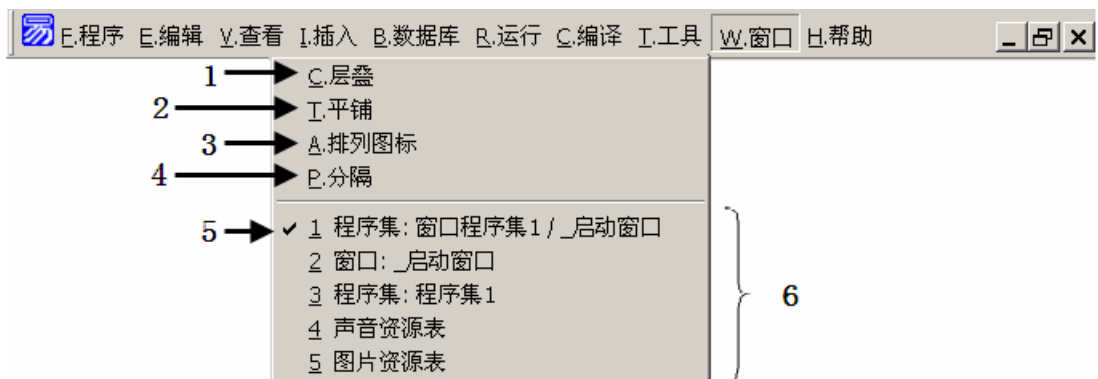


- (1) 调用菜单编辑器编辑修改当前窗口的菜单。
- (2) 编辑报表模板文件。
- (3) 管理系统中的易模块。
- (4) 执行指定的易向导文件。
- (5) 安装新的支持库或制作支持库安装包。
- (6) 本工具可以封装指定的 COM 类型库或 OCX 组件，使其能够在易语言中被使用。
- (7) 配置当前在系统中使用的支持库。
- (8) 设置与系统相关的配置信息。

可以打开易语言的系统配置对话框，通过调整该对话框中各项属性的参数，可以自定义界面各部位颜色，可以选择各种配色方案，还可以对内置输入法等很多方面进行配置。

9. 窗口

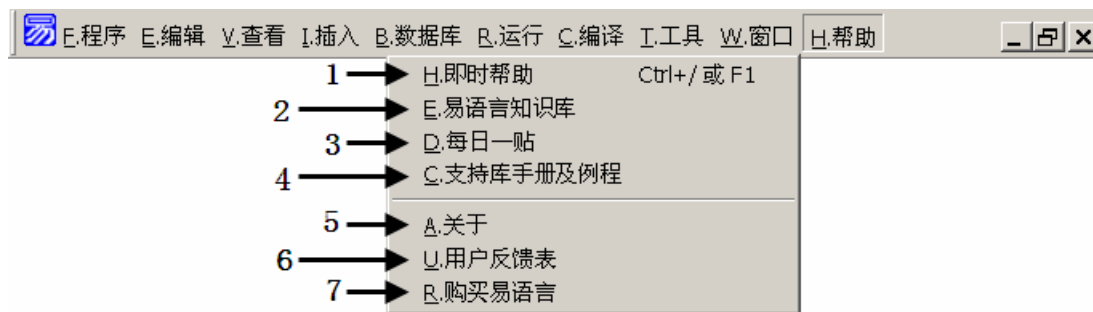
设置已被载入设计窗口的排列方式以及已被载入设计窗口的名称、隶属和类型。



- (1) 排列窗口成相互重叠。
- (2) 排列窗口成互不重叠。
- (3) 将图标排列在窗口底部。
- (4) 将活动的窗口分隔成窗格。
- (5) 已被激活的设计窗口。
- (6) 已被载入的设计窗口。

10. 帮助

易语言帮助和易语言系统信息。



(1) 在状态夹中显示有关当前位置的帮助信息。

(2) 打开并显示易语言知识库。

需要安装知识库文件，方可使用。

(3) 显示每日一贴。

(4) 提供有关易语言支持库的帮助信息。

(5) (6) (7) 显示程序和程序注册信息。

2.2.3 上机操作

1. 在设计窗口中添加组件

从组件组件箱中选出所需的组件添加在设计窗口中，只需要用鼠标左键在组件箱中点击欲添加的组件，使其处于选中状态，然后在设计窗口中左键单击或按住鼠标左键拖动，拉出一个组件即可。添加后的组件可以通过拖动鼠标改变其位置和大小，也可以使用方向键来微调组件的位置，还可以按住[Shift 键+方向键]来微调组件的大小。

2. 命令参数的输入

易语言提供的参数引导输入功能，减少了记忆量，更节省了编程的时间，极大降低了程序录入的错误。对于参数较多的命令，程序员不需要再花时间去查询参数的意义，可以直接将命令展开输入，方法：将光标停在欲展开的命令上，如果当前行没有通过编译，则不能展开命令，可以使用[Shift+Enter]键来编译当前行，然后按下[ALT+右方向键]，该命令就会被展开，各参数都列在了该命令的下面，可以直接在命令下的参数分支上输入。

3. 即时帮助信息

易语言编程环境在用户进行任何操作的同时，会将有关的支持信息在提示面板中显示出来，若不能即时提示，可以使用以下介绍的方法：随时按下“F1 键”使用可随时得到与主题相关的帮助信息。

即时帮助信息可显示系统中各运行支持库内的命令、库定义数据类型、库定义常量等等信息。直接在工作夹内的支持库面板中找到并单击欲查找信息的项目，此时所有的相关信息将会显示在状态夹的提示面板中。

如果欲将这些信息提取出来打印或者以后阅读，可以在相应项目上单击鼠标右键，在弹出菜单中选择“拷贝帮助文本到剪贴板”或者“写帮助文本到文件”，输出与该项目及该项目所有子项目相关的帮助信息，供电脑中浏览或打印出来阅读。

4. 备注和屏蔽的方法

备注是一行或一段代码的提示和说明。编写代码时要养成一个良好的习惯，就是给部分代码输入备注信息，这样一来，既方便了自己日后阅读，又方便其他人更快的理解程序代码的思路和功能。输入方法：在备注文字前加“'”号，则该符号后的本行文字变为备注，在输入代码时，可以在代码的旁边或代码的下方输入备注。

在任何情况下，如果想屏蔽一行代码，则在该行代码前加“'”号，和置为备注的方法相同，屏蔽后的代码在运行调试时不会被编译，调试程序寻找错误时，该方法会起到很大作用。将代码前的“'”号删除便可以解除屏蔽。还可以在代码上点击鼠标右键，弹出的菜单中也有“屏蔽”和“解除屏蔽”选项。也可使用[Ctrl+K 键]进行屏蔽，选中多行代码，然后使用[Ctrl+K 键]来屏蔽多行代码，然后可以使用[Ctrl+M 键]来解除屏蔽。

以上四种操作只是编写代码时的基本操作方法，其他细节上的使用，将在以后的相关地方加以详解。

2.4 变量

在程序运行中可以改变的量，称为变量。

变量实际上是常量的名字，只不过在程序运行的不同时刻可能代表不同的常量罢了。一个变量名好象旅馆的一个房间标记，今天可以住张三，明天可以住李四。

任何一个变量参与运算时，总是取它所代表的具体数据（即常量）来进行，可以代表一个具体的数据（即常量），或代表一组数据。

变量的声明方法：

子程序名	返回值类型	公开	备注
_启动子程序	整数型		本子程序在程序启动后最先执行

变量名	类型	静态	数组	备注
甲	小数型			
乙	小数型			
和	小数型			

Diagram illustrating the steps to create a variable declaration table:

1. Move the cursor to the subprogram table.
2. Press Ctrl+L to generate the blank table.
3. Press Enter to add a new blank row.
4. Input the variable name (e.g., 甲).
5. Input the data type (e.g., 小数型).
6. Press the space key to open the data type selection table.
7. Select the data type.
8. Add annotations (e.g., 静态, 数组, 备注).

①将光标移到某个子程序中（包括子程序被声明的表格中或子程序中的空白处），使用“Ctrl+L 键”生成下面的空白表格；

②生成的空白表格；

③按“回车键”添加新的空白行；

④输入变量的名称，除“_”外不能有任何符号和标点；

⑤输入变量的数据类型，也可以按“空格键”弹出数据类型表从中选择；

⑥可按“空格键”选中/取消，如被选中，子程序第二次被调时变量值就不会自动还原到初始值；

⑦用来定义变量的数组维数和成员数（如：2，3），可以使变量保存一组数据，其默认（为空）只能保存一个具体的数据；

⑧注解变量的作用和在程序中相关信息。

一旦它被声明之后，就可以使用赋值运算符“=”，给它分配值（数据）。

2.4.1 变量的作用范围

从变量的使用范围来区分,可以将变量分为“局部变量”、“程序集变量”和“全局变量”。

局部变量,只能在其所在的子程序中才能被调用的变量,其他子程序都无法调用。因为子程序被调用的时候,这种变量才占用系统的内存,当子程序执行结束后,变量所占空间被系统收回,因此局部变量是非常节省系统内存的。

程序集变量,一般情况下仅在本程序集中被调用。

程序集变量所在的程序集中的所有子程序,都可以自由访问程序集变量,多个子程序都需要访问的数据,可以使用程序集变量来存储。属于静态变量。

全局变量,在程序运行后,所有程序集内子程序都可以使用的变量。也是覆盖范围最大的变量。这种变量在程序运行后即占用内存空间,在程序运行结束才从内存中清除,所以会长时间占用系统资源,建议根据程序的实际情况适当使用。

在选择使用变量的类型时,尽量选择符合该变量使用范围的变量类型,以节省系统内存。

2.4.2 变量的赋值

赋值语句是使变量取得数据的常用方法之一。

给变量赋值的时候要注意变量的数据类型,符合各数据类型的赋值规则即可。

变量的赋值还有几个需要注意的地方,例如:

1. 给数值型数据赋值时,数据会自动转换类型

任意数值类型的数据可以被写入到其它任意数值类型的变量中,系统将自动进行转换。例如将一个短整数写入到整数型变量中,将一个整数写入到小数型变量中等等,但是此时必须注意防止上一章中所提到的溢出问题。若将小数型变量写入到整数型变量时,会丢失小数点后的内容,请大家千万注意使用,最好转换类型是一一对应。

2. 使用“连续赋值()”命令给多个变量同时赋值

“连续赋值()”命令有2个参数,第一个参数是用做赋予的值和资源,第二个参数是被赋值的变量或变量数组,第二个参数可以重复添加,即可以添加多个被赋值的变量,例如:

连续赋值 (100, 变量 1, 变量 2, 变量 3, 变量 4)

命令运行后,将会给变量 1、变量 2、变量 3 和变量 4 同时赋值 100。这一行代码相当于以下四行代码:

变量 1 = 100

变量 2 = 100

变量 3 = 100

变量 4 = 100

以上代码是最基本也是最典型的赋值语句。其格式为:

变量名 = 表达式

功能是:把“=”号右端的表达式的值赋予“=”号左端的变量。或者说,让“=”号左端的变量取“=”号右端表达式的值。“=”号,称为赋值符号,不同于数学上的等于号。

3. 赋值程序例

在数学上,

变量 1=变量 1 + 1

是矛盾的方程,不存在这样的“变量 1”。

而在易语言中,这类语句是屡见不鲜的。它执行的结果是在旧“变量 1”取值的基础上让“变量 1”取得一个新值。也就是改变“变量 1”所代表的数值。上面那个语句执行的结

果就是让“变量 1”增大 1。

如果使用

变量 1+ 变量 2=100+100

或

变量 1+ 变量 2= 变量 3+ 变量 4)

都是错误的。因为赋值语句的格式中，要求赋值符号的左端必须为一个变量名。而这里的“变量 1+变量 2”不是变量名（它不符合变量名的构成规则）。

例 1. 某学生 5 门功课的考试成绩分别为 75，82，90，63，84。求该生的考试总分和平均分。“Windows 控制台程序”代码如下：

子程序名	返回值类型	公开	备 注
_启动子程序	整数型		本子程序在程序启动后最先执行

变量名	类 型	静态	数组	备 注
语文	小数型			
数学	小数型			
外语	小数型			
化学	小数型			
物理	小数型			
总分	小数型			
平均分	小数型			

```

» 标准输出 ( “请输入语文成绩：” )
语文 = 到数值 (标准输入 () )
» 标准输出 ( “请输入数学成绩：” )
数学 = 到数值 (标准输入 () )
» 标准输出 ( “请输入外语成绩：” )
外语 = 到数值 (标准输入 () )
» 标准输出 ( “请输入化学成绩：” )
化学 = 到数值 (标准输入 () )
↓ +» 标准输出 ( “请输入物理成绩：” )|
物理 = 到数值 (标准输入 () )
总分 = 语文 + 数学 + 外语 + 化学 + 物理
平均分 = 总分 ÷ 5
» 标准输出 ( “该学生的总分是：” + 到文本 (总分), #换行符, “      平均分是：” +
» 到文本 (平均分))
标准输入 ()
返回 (0) ’ 可以根据您的需要返回任意数值

```

按“F5 键”运行程序，分别将对应的分数输入，程序便将计算结果输出显示。

2.4.3 变量的初始值

变量的初始值即一种变量在程序运行时，没有赋予新值前的初始数据。每一种数据类型的变量初始值都有所不同。像文本型变量的初始值是一个空文本，表示为“”，数值型变量的初始值是 0 等等。下表所示。

变量的初始值

变量类型	变量初始值	初始值在代码中的表示方法
数值型	0	0
逻辑型	假	假
日期时间型	1899 年 12 月 30 日	[1899 年 12 月 30 日]
文本型	空文本	“ ”
字节集型	空字节集	{ }

大家可以制作一个简单的例程，来查看各种变量的初始值。新建一个“Windows 控制台程序”，在“_启动子程序”新建 10 个局部变量，并分别将变量名改为“字节变量”、“短整数变量”、“整数变量”、“长整数变量”、“小数变量”、“双精度小数变量”、“逻辑变量”、“日期时间变量”、“文本变量”、“字节集变量”，然后根据变量名定义相应的数据类型，并输入代码：

```
标准输出 (, 字节变量, #换行符, 短整数变量, #换行符, 整数变量, #换行符, 长整数变量, #换行符, 小数变量, #换行符, 双精度小数变量, #换行符, 逻辑变量, #换行符, 日期时间变量, #换行符, 文本变量, #换行符, 到文本 (字节集变量))
```

```
标准输入 ()
```

按“F5 键”试运行程序，控制台窗口将 10 个变量的值分行显示出来，由于没有对这些变量进行赋值，所以画板显示的是这些变量的初始值。

如一个变量未被赋值时，将取零值或空文本。

“Windows 控制台程序”代码如下：

子程序名	返回值类型	公开	备注
_启动子程序	整数型		本子程序在程序启动后最先执行

变量名	类型	静态	数组	备注
数值型	整数型			
文本型	文本型			

```
数值型 = 数值型 + 1
文本型 = 文本型 + “文本”
标准输出 (, “数值型的值:” + 到文本 (数值型), #换行符, “文本数据为:” + 文本型)
标准输入 ()
返回 (0) ' 可以根据您的需要返回任意数值
```

按“F5 键”运行程序，其中“数值型”的输出为“1”；“文本型”的输出值是“文本”。

```
数值型 = 数值型 + 1
文本型 = 文本型 + “文本”
```

以上两行代码在计算时相当于

```
数值型 = 0 + 1
文本型 = “” + “文本”
```

2.4.4 静态局部变量

“静态”属性是局部变量的重要属性。具有“静态”属性的局部变量称为“静态局部变量”。静态局部变量在子程序运行完毕后仍保留其内容；而非静态变量，即普通局部变量，

在每次进入子程序时都被重新初始化。

静态变量大致相当于“局部变量”和“全局变量”的结合：它具有局部变量的作用域，同时具有全局变量的生命周期。

静态变量的定义和取消定义的方法很简单，在欲定义的局部变量的静态属性上点击鼠标左键，当静态属性上出现“√”后，即表示定义了一个静态变量，当再次点击将“√”去掉，即表示取消定义。也可以在静态属性上按空格键，也可以定义和取消变量的静态属性。

下面编写一个简单的程序，来测试变量的静态属性。

新建一个“Windows 控制台程序”，并在“_启动子程序”中新建 2 个局部变量。一个变量名定义为“静态变量”；另一个变量名定义为“非静态变量”。都为整数型变量。将“静态变量”设置为静态属性。

输入代码：

子程序名	返回值类型	公开	备注
_启动子程序	整数型		本子程序在程序启动后最先执行

标准输出 ("输入数值：")
 输入值 = 到数值 (标准输入 ()) ' "输入值" 为程序集变量
 子程序 () ' 调用 "子程序"
 返回 () ' 可以根据您的需要返回任意数值

子程序名	返回值类型	公开	备注
子程序			

变量名	类型	静态	数组	备注
键值	整数型			
静态变量	整数型	✓		
非静态变量	整数型			

静态变量 = 静态变量 + 输入值
 非静态变量 = 非静态变量 + 输入值
 标准输出 (#换行符, "静态变量 = " + 到文本 (静态变量), #换行符, "非静态变量 = " + 到文本 (非静态变量), #换行符, #换行符, "输入1,继续计算。")
 键值 = 到数值 (标准输入 ())
 如果真 (键值 = 1)
 子程序 ()

按“F5 键”试运行程序，输入一个数，然后按“回车键”。

会显示 2 行内容，其中第二行的数字是不变的，显示的是非静态变量中的数据；而第一行的数字每次都递增，显示的是静态变量中的数据。

分析代码可以发现：非静态变量的值每次后，都会恢复它的初始值，所以代码：

非静态变量 = 非静态变量 + 输入值

其实等于代码：

非静态变量 = 0 + 输入值

静态变量会保存上次的值，所以总是递增的。

2.5 易语言的数据类型

在讲述变量之后和讲述常量之前，有必要介绍易语言中不同的数据类型。

数据类型包括：数值型、逻辑型、日期时间型等。

2.5.1 了解数据类型

易语言中基本数据类型有 6 种，包括数值型、逻辑型、日期时间型、文本型、字节集型、子程序指针型。

字节型。可容纳 0 到 255 之间的数值。

数值型中**整数型**数据，如：13556。

逻辑型数据，只能有 2 种值，即“真”或“假”。

日期时间型数据，用来记录日期及时间，如：[2002-2-2]。

文本型数据，可用来记录一段文本，如：“中文编程易语言”。在程序中表示一段文本数据，都要用双引号将文本引起来。

字节集型数据，用作记录一段字节型数据，表示为{23456754}。图片或 mp3 格式的文件是典型的字节集型数据，在程序中，存放此类数据的变量一定要定义为字节集型。

子程序指针型数据，是一个子程序在内存中的地址。

2.5.2 给变量正确赋值

定义了变量的数据类型后，要给变量赋值就应注意赋值的类型要和变量类型相同。例如：

文本变量 = “中文编程易语言” 、给文本变量赋值

整数变量 = 32342 、给整数变量赋值

日期时间变量 = [1982 年 1 月 1 日] 、给日期时间型的变量赋值

字节集变量 = # 图片 、给字节集变量赋值, 图片 1 图片资源表中的资源

这里要注意，给“子程序指针”类型的变量赋值，表示为“&”+要指向的子程序名。
例如：

变量 = &子程序 1

2.5.3 数据的比较

在编程中，经常会在各种数据间进行比较。同种数据类型之间进行比较，可以直接进行；而不同种数据之间进行比较，就要先进行数据类型的转换，将不同种的数据类型转换为同一种数据类型后才能进行比较，否则程序会报错。

例如：编辑框中输入了一个整数，要比较编辑框中的内容是否大于 50。由于编辑框中的内容是一个文本，首先要将编辑框中的内容转换成整数型数据后，再进行比较，输入以下代码：

到数值（编辑框 1. 内容）> 50

比较后，会返回一个逻辑值，如果大于 50 就会返回真，小于或等于 50 将返回假。

易语言中常用的数据类型间转换的命令有：

“到数值（）”命令，用来将一个通用型数据转换到整数型。

“到文本（）”命令，用来将一个通用型数据转换到文本型。

“到字节集（）”命令，用来将一个通用型数据转换到字节集型。

“从字节集转换（）”命令，用来将一个字节集型的数据转换成通用型数据，命令的第 2 个参数控制欲转换成的数据类型。

“到时间（）”命令，用来将一个文本型的数据转换成日期时间型。

使用这些数据类型间互相转换的命令，就可以进行不同数据类型间的比较了。

例如：比较 2 个编辑框中数的大小，用信息框显示出比较的结果，并用第 3 个编辑框显示出较大数减小数的结果。在窗口中添加 3 个编辑框组件和一个按钮组件，双击按钮组件，然后输入代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

```

-- 如果 (到数值 (编辑框1.内容) > 到数值 (编辑框2.内容))
    信息框 ("编辑框1中数大, 数值为:" + 到文本 (编辑框1.内容), 0, )
-- 编辑框3.内容 = 到文本 (到数值 (编辑框1.内容) - 到数值 (编辑框2.内容))
-- 如果 (到数值 (编辑框1.内容) = 到数值 (编辑框2.内容))
    信息框 ("2个编辑框中的数相等, 数值为:" + 到文本 (编辑框1.内容), 0, )
-- 信息框 ("编辑框2中的数大, 数值为:" + 到文本 (编辑框2.内容), 0, )
    编辑框3.内容 = 到文本 (到数值 (编辑框2.内容) - 到数值 (编辑框1.内容))
  
```

变量之间的比较也是一样，一定要注意变量的数据类型，不同数据类型的变量一定要转换成相同类型后再进行比较。

2.5.4 数据类型的存储字节与溢出

1. 数据类型的存储字节

各种数值型的数据都在内存中占用一定的存储空间。字节(byte)是系统中的基本存储单位。数据类型所占字节数越多，所能容纳数值的范围就越大。参见表 2-1。

表 2-1 常用数据类型

数据类型名称	占用字节数	取值范围
字节型	1	0 到 255
短整数型	2	-32,768 到 32,767
整数型	4	-2,147,483,648 到 2,147,483,647
长整数型	8	-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807
小数型	4	-3.4E38 到 3.4E38 (7 位小数)
双精度小数型	8	-1.7E308 到 1.7E308 (15 位小数)
逻辑型	2	“真”或“假”，
日期时间型	8	100 年 1 月 1 日到 9999 年 12 月 31 日
子程序指针	4	尺寸为 4 个字节。具有此数据类型的变量可以用来间接调用子程序。
文本型		由以字节 0 结束的一系列字符组成
字节集		一段字节型数据

从上表可以看出，数值型数据容纳的数值范围越大，占用的字节就越多。比如，短整数型的数值 3000 和整数型的数值 3000，都代表了数值 3000，但在系统中占用的空间却不同，即短整数型占 2 个字节，整数型占 4 个字节。所以，在实际应用时就要根据自己的需要来选

择使用的数据类型，避免存储空间的浪费。例如，存储的数据在-32768 至 32767 以内，就要采用短整数型；如果使用小数而对精度不高，就可以使用小数型而不用采用双精度小数型等等。

2. 数据的溢出

某数据类型存储的值超出了其所能容纳的范围，就会发生数据溢出错误。比如，让短整数型数据存放大于 32767 的数值，将会得到错误的结果。所以在选择数据类型时，除了要避免空间的浪费，又要防止数据的溢出。

可以做一个简单例题来测试一下数据的溢出，新建一个“Windows 控制台程序”，然后在“_启动子程序”中首先按下 Ctrl+L 键，新建一个变量，并定义变量名为“整数变量”，变量类型为整数型，然后输入代码：

子程序名	返回值类型	公开	备注
_启动子程序	整数型		本子程序在程序启动后最先执行

变量名	类型	静态	数组	备注
整数变量	整数型			

```

» 标准输出 ( “请输入一个大于10位的整数 (如：10000000000)：” )
↓ + 整数变量 = 到数值 (标准输入 () )
» 标准输出 ( 到文本 (整数变量))
标准输入 ()
返回 () ’ 可以根据您的需要返回任意数值
    
```

最后按“F5 键”试运行程序，输入一个很大的数（比如：10000000000），然后按“回车键”，如果超出整数型变量容纳范围而产生溢出，会显示出错误的结果。

2.5.5 自定义数据类型

除了使用易语言提供的数据类型以外，还可以根据需要自定义新的数据类型。例如要定义一个数据类型“矩形”，定义方法如下：

第一步，新建一个“Windows 窗口程序”，双击程序面板中的“自定义数据类型”。

第二步，在自定义数据类型界面按下 Ctrl+N 键，新建一个数据类型。然后将数据类型名定义为“矩形”，由于决定一个矩形的位置取决于矩形左上点的横纵坐标和矩形右下点的横纵坐标。所以，在“成员名”上按 4 次回车，加入 4 个成员。将 4 个成员名分别定义为“左上横坐标”、“左上纵坐标”、“右下横坐标”、“右下纵坐标”。

最后，就要来使用这个自定义的数据类型了。画板有一个方法是“画矩形 ()”，下面就用自定义的数据类型“矩形”来为画矩形方法填写参数。在窗口中添加一个画板组件和一个按钮组件，双击按钮组件，在“_按钮 1_被单击”子程序中新建一个变量，变量名为“矩形”，然后定义变量的数据类型为“矩形”，然后输入代码：

子程序名	返回值类型	公开	备注
_按钮1_被单击			

变量名	类型	静态	数组	备注
矩形	矩形			

矩形.左上横坐标 = 画板1.顶边

矩形.左上纵坐标 = 画板1.左边

矩形.右下横坐标 = 画板1.宽度

矩形.右下纵坐标 = 画板1.高度

画板1.画矩形 (矩形.左上横坐标, 矩形.左上纵坐标, 矩形.右下横坐标, 矩形.右下纵坐标)

代码输入后试运行程序，点击按钮，画板会以画板的边框大小画一个矩形。

2.6 常量

常量，其值在使用过程中不会发生变化的变量，称为常量。比如数字 15，不管程序如何变化，它永远是数字 15。

核心支持库中已经定义了大量常量，其它支持库通常也会定义一些常量，用户也可以在程序中定义自己的常量。

各种图片或声音等资源都被看作常量，要想在程序中随时调用，可以将其存放在资源表中，这样在编程时不但可以随时调用，而且资源表中的资源会和程序一起编译到可执行文件里面。

2.6.1 了解常量

常量是一个固定的量，其值不可以被改变。易语言中规定了一些常量，这些常量都有固定的值，例如易语言中的“#蓝色”代表了数值 16711680、“#F 键”代表了数值 70，所以在程序中使用“#蓝色”其实是调用了“16711680”这个颜色值。

核心支持库定义了许多常量，这些常量可以直接用#常量名即可调用，有数值型常量，如颜色值：#蓝色、#绿色；有文本型的常量，如：# 引号等等。扩展支持库也有许多常量的定义，并且新增加的支持库中，有的也会增加新的常量。

2.6.2 ASCII 码

ASCII 码是各种计算机通用的一种常量。例如字符 a 的 ASCII 码是 97、字符 b 的 ASCII 码是 98 等等，参见下表所示。可以使用易语言中的“字符 ()”和“取代码 ()”命令，在 ASCII 码和字符之间进行转换。例如：

信息框 (取代码 (“a” , 1), 0,)

信息框会显示 “a” 的 ASCII 码。

常用 ASCII 码表

ASC 值	对应字符	ASC 值	对应字符	ASC 值	对应字符	ASC 值	对应字符
032	(space)	056	8	080	P	104	h
033	!	057	9	081	Q	105	i

034	"	058	:	082	R	106	j
035	#	059	;	083	S	107	k
036	\$	060	<	084	T	108	L
037	%	061	=	085	U	109	M
038	&	062	>	086	V	110	N
039	'	063	?	087	W	111	O
040	(064	@	088	X	112	P
041)	065	A	089	Y	113	Q
042	*	066	B	090	Z	114	R
043	+	067	C	091	[115	S
044	,	068	D	092	\	116	T
045	-	069	E	093]	117	U
046	.	070	F	094	^	118	V
047	/	071	G	095	_	119	W
048	0	072	H	096	`	120	X
049	1	073	I	097	a	121	Y
050	2	074	J	098	b	122	Z
051	3	075	K	099	c	123	{
052	4	076	L	100	d	124	
053	5	077	M	101	e	125	}
054	6	078	N	102	f	126	~
055	7	079	O	103	g	127	□

2.6.3 常量的使用

下面就介绍常量的使用方法：

(1) 颜色值常量的使用

有颜色属性的组件，在颜色属性上都有一个颜色选择器，用来直接改变颜色，颜色选择器上可直接选择颜色的颜色值都作为常量提供，在调用的时候直接输入“#颜色名”即可，如：

标签 1. 背景颜色 = #天蓝

(2) “#换行符”的使用

一段文本尾部加入了一个“#换行符”，接在换行符后面的文本将另起一行，相当于在记事本中输入的回车键。如果想让编辑框显示一段文本并自动换行，就需要使用换行符，将“#换行符”加到欲换行文本的前面即可，如：

编辑框 1. 是否允许多行 = 真

编辑框 1. 内容 = “易语言” + #换行符 + “编程可视化”

(3) “#引号”、“#左引号”、“#右引号”

为了不和代码中表示文本数据的引号相冲突，程序中将文本的引号作为了一个文本常量，如果要想让编辑框显示出一个引号，就要使用“#引号”常量，要显示中国标点中的引号，就要使用常量“#左引号”、“#右引号”。例如：让编辑框显示出：“我爱易语言，我爱编程！”，需要输入以下代码：

编辑框 1. 内容 = #左引号 + “我爱易语言，我爱编程！” + #右引号

（4）键代码的使用

易语言中，将标准的 101 键盘上所有键的键代码都作为了核心支持库定义的常量，在程序中使用只需要输入“#”+键名，如键盘上的 F11 的键代码，在易语言中用常量表示为：#F11 键。例如，在向编辑框中输入内容的时候，想简单的屏蔽掉某个键，就可以在编辑框的“按下某键”事件子程序中输入代码。

（5）用常量填写参数

很多命令参数填入的都是常量，如：“时间到文本”命令，此命令将指定时间转换为文本并返回。第 1 个参数为“欲转换到文本的时间”，而第 2 个参数值可以为以下常量：1、#全部转换；2、#日期部分；3、#时间部分。在填写第二个参数时，可以填写数字，也可以直接填写常量名，如：

时间到文本 ([2004 年 3 月 16 日 5 时 11 分 11 秒], #日期部分)

2.6.4 枚举常量

枚举常量是一种非常方便的常量类型，它本身就是一个常量的集合，将多个常量以成员的形式，存放在一个常量中，使用的格式为

#枚举常量名. 成员名

这里要注意，枚举常量只是一种常量的表现形式，是由易语言支持库定义的常量，和普通常量相同，但只能由用户来调用，但不能自定义。

易语言中有很多支持库中使用了枚举常量，如核心支持库中定义的“变体类型”，“变体类型”提供变体型中所能够容纳数据类型的枚举值。如表 3-4 中的某类型枚举常量可存放于变体型中，通过“变体型. 取类型 ()”取回当前变体型对象的数据类型。

变体类型常量成员及常量值

变体类型 枚举常量集合类型

成员	描 述
未知	常量值为 -1
初空	常量值为 0
数值型	常量值为 1
文本型	常量值为 2
逻辑型	常量值为 3
日期型	常量值为 4
对象型	常量值为 5
错误值型	常量值为 6
数值型数组	常量值为 7
文本型数组	常量值为 8
逻辑型数组	常量值为 9
日期型数组	常量值为 10

对象型数组	常量值为 11
错误值型数组	常量值为 12
变体型数组	常量值为 13

表中,列出了“变体类型”常量的所有成员名及成员的常量值,在程序中,如果想调用“变体类型”常量中的“日期型”成员,该成员的常量值为 4,程序中调用该成员就等于调用了 4 这个整数,例如用信息框显示出该成员使用代码:

```
信息框 (#变体类型.日期型, 0, )
```

运行后,信息框将显示 4。

这里要注意,在核心支持库中还定义了“变体型”,“变体型”和“变体类型”不同,“变体型”是一种数据类型,可以将一个变量的类型定义成“变体型”,“变体型”的变量,可以加入成员和改变成员的值;而“变体类型”是一个常量,其值只可以调用而不可以改变。

“变体型”变量的成员和值,要通过调用命令来改变。

例如程序定义一个“变体型”变量,并加入一个文本型成员,然后给该成员赋值“,然后用信息框显示该成员,代码如下:

```


| 变量名 | 类 型 | 静态 | 数组 | 备 注 |
|-----|-----|----|----|-----|
| 变体型 | 变体型 |    |    |     |


变体型.创建数组 (6, 1)
变体型.赋值 (“Windows控制台程序”, 1)
信息框 (变体型.取文本 (1), 0, )

```

对“变体型”变量的操作的其他一些命令包括:清除()、取类型()、取数组成员数()、取文本()、取数值()、取逻辑值()、取日期()、取对象()、取变体型()、赋值()、创建数组()。程序中使用这些命令来操作“变体型”变量。

2.6.5 自定义常量

除了各支持库定义的常量以外,易语言中还可以自定义常量,自己来规定一个新的常量及其代表的值。自定义常量,可以定义一些固定值,编程中使用一些自定义常量还可以增加编程的灵活性,当程序中多个地方调用了某个自定义常量时,如果改变这个自定义常量的值,那这些调用该常量的地方将会自动调用改变后的新值,这样可以节省改写程序的时间。

定义了一个新的常量后,可以任意定义常量的名称,然后在“常量值”上输入该常量的值。自定义常量的使用方法和非自定义的常量的使用方法相同,用“#”+自定义常量的名称。

下面就用一个简单的例程来学习使用自定义常量。新建一个“Windows 窗口程序”,在窗口中添加 1 个编辑框组件、1 个标签组件和 1 个按钮组件。然后按照上面介绍的方法自定义一个常量,常量名叫“显示内容”,然后将常量值定义为“易语言”。这里要说明的是,如果定义数值型的常量,直接在“常量值”上输入数值即可;如果定义文本型常量要在欲定义的文本两端加双引号。

双击窗口中的按钮,在“_按钮 1_被单击”子程序中输入代码:

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

标签1.标题 = #显示内容
 编辑框1.内容 = #显示内容

最后运行程序，按下按钮后，可以看到标签和编辑框同时显示出“易语言”。可以试着在不改变代码的情况下，直接改变自定义常量的值，再次运行程序，可以看到改变常量值后，标签和编辑框显示的内容也跟着改变。

2.7 命令

在程序中，除了允许使用常量、变量之外，还允许使用事先定义了的命令，用户只要给出命令名与参数的值，就可以求出相应的值来。如求 5 的平方根，可调用命令“求平方根(5)”。

程序是由各种命令组合而成的，不同的命令完成不同的工作。易语言中提供了大量的命令，用户可以使用这些命令来实现预想的运行效果。一个程序可以实现一种或多种功能，而这些功能的实现都离不开程序内部调用的命令。命令是程序的基本组成部分，要学习易程序的编写，首先就要了解易语言所提供的命令。

2.7.1 命令的格式

命令格式如下：

命令名称（参数，.....）

大部分命令都需要填写参数，参数用括号括起来的，并用逗号分隔。部分命令无需参数，但括号不能省略，如“结束（）”命令。各种命令所要求参数的个数以及数据类型各有不同，由其语法所规定。例如“到文本（）”命令，该命令只需一个参数，参数内容为欲转换成文本的数据。有些命令的参数很多，如“子文本替换（）”命令的格式如下：

子文本替换（欲被替换的文本，欲被替换的子文本，[用作替换的子文本]，[进行替换的起始位置]，[替换进行的次数]，是否区分大小写）

2.7.2 命令的返回值

大多数命令执行完毕后都有返回值。有的命令返回运算结果，如“求正弦（）”命令，返回求得的正弦值；有的命令返回的执行的的结果，如“取文本左边（）”命令，返回取出的文本内容；有的命令返回运行是否成功的状态，如“创建目录（）”命令，创建成功则返回“真”，创建失败则返回“假”等等。大部分时候，当前命令的返回值对后续命令来说非常重要。例如一个命令如果运行成功了，就提示成功，否则提示失败，“Windows 控制台程序”的代码如下：

子程序名	返回值类型	公开	备 注
_启动子程序	整数型		本子程序在程序启动后最先执行

变量名	类 型	静态	数组	备 注
逻辑值	逻辑型			

```

-- 如果 (创建目录 ("c:\目录") = 真)
-- 逻辑值 = 真
-- 逻辑值 = 假
-- 标准输出 ( 逻辑值)
-- 标准输入 ()
-- 返回 (0) ' 可以根据您的需要返回任意数值
    
```

各命令的语法规定了其返回值的数据类型，在实际使用中，应当注意有可能需要对返回值的数据类型加以转换，例如，文本数据只接收文本型，因此要显示一个数字就可以使用“到文本 ()”命令将数字转换为文本形式显示，代码如下：

```

文本变量 = 到文本 (求平方根 (100))
    
```

“求平方根 ()”命令的返回值是一个数值型的，如果要以文本方式显示，就要用“到文本 ()”命令进行转换。

有些命令的返回值是一个通用型的数据，代表根据参数不同，其返回值数据类型也可以不同。例如“多项选择 (,)”命令。该命令有 2 个参数，第一个参数是索引值，第二个参数是待选项，待选项可以重复添加。待选项数据类型是通用型（表示参数 2 可以是任意数据类型）的，返回哪个待选项取决于第一个参数的索引值。索引值是 1 则返回第一个待选项；索引值是 2 则返回第二个待选项。所以，所选项是哪种类型的数据，返回值就为哪种类型的数据。下面就编写一个小程序来了解一下多项选择命令。

新建一个“Windows 控制台程序”，在“_启动子程序”中输入如下代码：

子程序名	返回值类型	公开	备 注
_启动子程序	整数型		本子程序在程序启动后最先执行

```

标准输出 ( "1," + "易语言", #换行符, "2," + 到文本 (3.14159), #换行符, "
3," + 到文本 (100000), #换行符, "4," + 时间到文本 ([2005年10月1日], ), #换行符,
"5," + 到文本 ({ 123456 }), #换行符)
-- 标准输出 ( "请输入数字，显示相应的项目。")
-- 子程序 ()
-- 返回 (0) ' 可以根据您的需要返回任意数值
    
```

按“F5 键”试运行程序，输入“4”后按“回车键”，将显示待日期时间“2005 年 10 月 1 日”。

有些命令无返回值，如“结束 ()”命令，此类无返回值的命令运行后不返回任何值，所以直接使用即可，例如：

子程序名	返回值类型	公开	备注
子程序			

变量名	类型	静态	数组	备注
输入值	整数型			

```

输入值 = 到数值 (标准输入 ())
--- 如果真 (输入值 ≥ 6)
    结束 ()
↓ + 标准输出 ( 多项选择 (输入值, “易语言”, 到文本 (3.14159), 到文本 (100000), 时间到
文本 ([2005年10月1日], ), 到文本 ({ 123456 })), #换行符)
子程序 ()
    
```

命令是否有返回值，返回值的数据类型，都可以通过即时帮助系统查找到。

2.7.5 文本操作类命令

在编写程序时免不了对大量的文本型的数据进行操作，文本操作类的命令比较全面，下面介绍常用文本操作命令。

1. “取文本长度 ()” 命令

获取指定文本的字节长度，半角数字和字符为 1 个字节的长度，汉字和全角标点符号为 2 个字节的长度，如：

取文本长度 (文本变量)

可以取出变量中文本的长度。

2. “取文本左边 ()”、“取文本右边 ()” 和 “取文本中间 ()” 命令

这 3 个命令可以取出一段文本中任意位置的文本。如：

取文本左边 (文本变量, 4)

可以将变量中的前 4 个字符取出来。

3. “寻找文本 ()” 和 “倒找文本 ()” 命令

从当前文本的指定位置开始寻找指定的文本，并返回最先找到该文本的位置。“寻找文本 ()” 是从指定文本的首部开始寻找，“倒找文本 ()” 相反。例如：

寻找文本 (文本变量, “:”, 1, 假)

代码运行后会返回找到的第一个 “:” 的位置。

4. “文本替换 ()” 命令

该命令可以将指定文本的某一部分用其它的文本替换。例如：

文本变量 = 文本替换 (文本变量, 4, 2, “xx”)

将 “文本变量” 中第 4 个位置开始的 2 个字符替换成 “xx”，并将结果保存到 “文本变量” 中。

2.7.6 时间操作类命令

时间操作类命令也是较常用的一类命令，可以对日期时间型数据进行操作。

1. “到时间 ()” 和 “时间到文本 ()” 命令

这 2 个命令用来在日期时间型数据和文本型数据之间转换。例如：

到时间 (“2004/2/2 12: 30: 25”)

“时间到文本”命令可以将指定的部分的时间转换成文本。例如：

时间到文本 (取现行时间 0, #日期部分)

2. “取现行时间 ()”命令

可以将当前系统的日期及时间取出，例如用“日期时间变量”保存当前的时间：

日期时间变量 = 取现行时间 ()

被保存时间是一个包括“年月日时分秒”的完整时间，如果想取出时间中的指定部分，需要使用其它时间操作类型命令来实现。

3. 取指定时间部分的命令

用来取出指定时间部分的命令有：“取时间部分 ()”、“取年份 ()”、“取月份 ()”、“取星期几 ()”、“取小时 ()”、“取分钟 ()”、“取日 ()”、“取秒 ()”、“取日期 ()”、“取时间 ()”。其中“取日期 ()”和“取时间 ()”命令返回值是日期时间型，其它返回值都为整数型。

“取时间部分 ()”命令可以取出日期时间型数据中的指定部分数值。例如：

(1) 取当前时间的年份：

取时间部分 (取现行时间 0, #年份)

(2) 用“日期时间变量”保存当前时间的“年、月、日”：

日期时间变量 = 取日期 (取现行时间 0)

2.7.7 位运算命令

位运算是指对数据进行二进制的逐位运算。计算机内部是采用二进制方式存储和处理数据的，输入到计算机的数字、字母、汉字等信息都以二进制的形式存储。

所谓二进制，就是以“逢二进一，借一当二”为原则，对数值进行计数的进位制，和我们日常使用的十进制类似，只不过十进制是“逢十进一”。

位的英文是 **Bit**，所以也常被称为比特位。

易语言中的位运算命令

(1) “位取反 ()”命令

“位取反 ()”命令对二进制数值每一比特位的值取反，即 0 变为 1，1 变为 0，返回值是转换后的十进制数。

例如：

文本变量 = 到文本 (位取反 (80))

代码运行后，“文本变量”将保存“位取反”运算结果“-81”。

(2) “位与 ()”命令

“位与 ()”命令对二进制数值的共同比特位进行“与”运算，即如两个或多个数值的共同位均为 1，则返回值的对应位也为 1，否则为 0，运算完毕后，返回值是转换后的十进制数。

比如：

一个二进制数的第 4 位为 1，另一个二进制数的第四位为 1，则返回值的第四位为 1；

一个二进制数的第 4 位为 0，另一个二进制数的第四位为 1，则返回值的第四位为 0；

一个二进制数的第 4 位为 1，另一个二进制数的第四位为 0，则返回值的第四位为 0；

一个二进制数的第 4 位为 0，另一个二进制数的第四位为 0，则返回值的第四位为 0；

例如：

文本变量 = 到文本 (位与 (56, 89))

运行后可以得出的结果为“24”。56 和 89 分别转换成二进制数为：0011 1000 和 0101 1001，进行与的运算后即会得出结果“0001 1000”即“24”。

(3) “位或 ()”命令

“位或 ()”命令对二进制数值进行“或”运算,并将运算后结果以十进制返回。如两个或多个数值的共同位均为 0,则返回值的对应位也为 0,否则为 1。运算完毕后,返回值是转换后的十进制数。

一个数值的第 4 位为 1,另一个数值的第四位为 1,则返回值的第四位为 1;

一个数值的第 4 位为 0,另一个数值的第四位为 1,则返回值的第四位为 1;

一个数值的第 4 位为 1,另一个数值的第四位为 0,则返回值的第四位为 1;

一个数值的第 4 位为 0,另一个数值的第四位为 0,则返回值的第四位为 0;

例如:

```
文本变量 = 到文本 (位或 (56, 89))
```

运行后的结果为“121”。56 和 89 分别转换成二进制数为:0011 1000 和 0101 1001,进行或的运算后即会得出结果“0111 1001”即“121”。

(4) “位异或 ()”命令

“位异或 ()”命令对二进制数值的共同比特位进行“异或”运算,并将运算结果以十进制返回。如果两个或多个数值的共同位相等(均为 0 或均为 1),则返回值的对应位就是 0,否则为 1。运算完毕后,返回值是转换后的十进制数。

比如:

一个数值的第 4 位为 0,另一个数值的第四位为 1,则返回值的第四位为 1;

一个数值的第 4 位为 1,另一个数值的第四位为 0,则返回值的第四位为 1;

一个数值的第 4 位为 1,另一个数值的第四位为 1,则返回值的第四位为 0;

一个数值的第 4 位为 0,另一个数值的第四位为 0,则返回值的第四位为 0;

例如:

```
文本变量 = 到文本 (位异或 (56, 89))
```

运行后的结果为“97”。56 和 89 分别转换成二进制数为:0011 1000 和 0101 1001,进行异或的运算后即会得出结果“0110 0001”即“97”。

2.7.8 其它常用命令

1. “读入文件 ()”和“写到文件 ()”命令

“读入文件 ()”命令将一个文件的所有数据读入程序,返回值是一个字节集型数据,在命令的参数中填入欲读入文件的全路径文件名,可以将读入的文件放在一个字节集型变量中,如:

```
字节集变量 = 读入文件 ( “C:\Downloads\echs.zip” )
```

对读入的文件数据经过处理后,用“写到文件 ()”命令写出至文件中。“写到文件 ()”命令的第一个参数指定写出文件的全路径文件名,文件的扩展名要和文件格式相匹配,写出的文件才能正常访问。例如将字节集型数据“字节集变量”中的内容写到文件中:

```
写到文件 ( “C:\echs.zip”, 字节集变量)
```

2. “寻找文件 ()”命令

“寻找文件 ()”命令可以在指定路径下寻找文件或目录,找到后就返回与条件匹配的文件名或目录名,如果没找到就返回一个空文本。命令的第一个参数为欲寻找的文件名,第二个参数为欲寻找文件的文件属性。例如寻找一个子目录:

```
寻找文件 ( “c:\目录”, #子目录)
```

3. “创建目录 ()”和“删除目录 ()”命令

“创建目录 ()”命令可以创建一个新目录,创建成功返回真,失败返回假。“删除目录 ()”命令可以用来删除一个目录,删除成功返回真,失败返回假。

“创建目录 ()”命令创建目录时,其父目录必须存在,否则会创建失败,即该命令不

能一次性创建多级目录。因此，如果要创建多级子目录，需要判断每一级目录的父目录是否存在，一层一层的创建各级子目录。

4. “打开文件（）”命令

“打开文件（）”用来打开一个指定的文件，成功返回被打开文件的文件号。虽然该命令只用来打开一个文件，并不对文件进行其它操作，但本命令取得的文件号，是很多文件操作类命令都要使用到的，如“读入文本（）”和“写出文本（）”命令、“读入数据（）”和“写出数据（）”命令等等。使用此命令可以比读入文件命令实现更多的操作。

例如将一个文本文件打开并用“读入文本（）”命令将读入的文本保存到“文本变量”中：

```
文本变量 = 读入文本（打开文件（“c:\帮助.txt”，，），）
```

2.8 运算符和表达式

将数据类型相同的常量、变量和函数用规定的运算符连接起来，就构成了表达式。表达式本身有一个值。

编写代码时，除了大量的使用命令或对组件的属性或方法进行操作，运算符的使用也非常重要。程序中所有涉及到的算术运算或关系比较运算等操作，都需要使用运算符。

易语言中提供了大量的运算符。例如赋值时使用的“=”号，就是赋值运算符，比较大小时使用的“>”和“<”号，是关系运算符等等。如表 2-2 所示。

易语言运算符

运算符分类	运算符	运算符含义	代码中显示
算术运算符号	+	加法运算，将加号两边的数相加	+
	-	减法运算，将减号两边的数相减；负号	-
	*	乘法运算，将乘号两边的数相乘	×
	/	除法运算，将除号两边的数相除	÷
	\	整除运算，将整除号两边的数整除	\
	%	求余数运算	%
关系运算符	>	判断是否大于	>
	<	判断是否小于	<
	= 或 ==	判断是否等于	=
	>=	判断是否大于等于	≥
	<=	判断是否小于等于	≤
	<> 或 !=	判断是否不等于	≠
	?=	判断是否约等于	≈
逻辑运算符	&& 或 QIE	逻辑与运算符，可以连接几个必须同时满足的条件	且
	或 HUO	逻辑或运算符，可以连接几个可选条件	或
赋值运算符	=	将等号后面的值赋值给等号前面的对象	=

程序中的运算符都有其优先级别，在程序运行的时候会按照符号的优先级别，从高到低依次运行。运算符的优先级别参见下表。

易语言常用运算符的优先级

运算符	优先级
() (小括号)	最高
*(乘) /(除)	↑
\ (整除)	
%(求余数)	
+(加) -(减)	
<(小于) <=(小于等于) >(大于) >=(大于等于) ==(等于) !=(不等于) ?=(约等于)	
&&(逻辑与)	
(逻辑或)	
=(赋值)	

在这里我们已看出：算术表达式中算术运算符的优先顺序，和在数学上是完全一样的：

(1) 如果有括号，必须先做括号内的。在程序中，没有大、中、小括号之分，一律使用应括号（），括号内可以套用括号，但不得超过 36 层。这就同数学上一样，先做最内层括号中的计算，层层向外脱，最后处理最外层括号中的计算。

(2) 无论括号内或括号外，函数计算优先于其它算术运算符。

(3) 乘、除运算优先于加、减运算。乘或除同时出现在一个表达式中时，先完成左边的后完成右边的(即以先后出现顺序为序)。

(4) 最后做加、减运算。加、减号同时出现在一个表达式中时，也以先后出现的顺序为序。

对于初学者来说，下列几点要格外注意：

1. 在数学上，代表两个数的字母相乘时，可以连写。如 A 代表一个数，B 代表一个数，则可用 AB 代表两个数相乘。在表达式中，这是绝对不允许的(这时它会把 AB 当作变量名)。必须写成 A*B。用 A.B 也是不允许的。

2. /号代表除号。

3. 左右括号必须成对出现。且不得用方括号[]或花括号{}代替。

2.8.1 算术运算符和算术表达式

1. 算术运算符

在程序中表示为：

＋，加法运算。如：3+2

－，减法运算或负值运算。如：10-2、-10

×，乘法运算。如：2×3

/，除法运算。如：20/12

\，整除运算。如：12\5，运算后会将保留一个整数，小数部分将被舍去

％，余数运算。还可以输入“求余数”，第一个参数填被除数，第二个参数填除数，第二个参数可以重复添加。如：1220%100、1220%100%120

2. 算术表达式

用算术符号和括号将运算对象连接起来的，符合易语言语法规则的式子，称易语言算术表达式。例如，下面是一个合法的易语言算术表达式：

```
变量 = ((6 × 12 + 16 ÷ 8) - 23) \ 10
```

表达式中运算的先后，是按照运算符的优先级别来进行判定的。

算式计算的结果可以被程序调用。

2.8.2 赋值运算符和赋值表达式

1. “=” 是赋值运算符，在程序中给变量赋值或用代码改变组件属性，大部分都是使用“=”进行赋值的，将等号后面的值赋值给等号前面的赋值对象。

2. 赋值表达式

一个正确的赋值表达式，一定要保证欲赋的值和被赋值的对象之间的数据类型相同，不同的数据类型要转换成相同的数据类型后再赋值。

3. 赋值运算符“=”和关系运算符“=”的区别。虽然2个运算符使用的是相同的符号，但含义却不同，赋值运算符“=”是用于赋值，将“=”右边的值（或变量）赋值给“=”左边的变量（或组件属性、数组成员、自定义数据类型成员）；关系运算符“=”，是比较符号两边的值是否相等，如果相等返回真，不相等返回假。

变量名	类 型	静态	数组	备 注
被比较值	整数型			
比较值	整数型			

```

--- 如果 (被比较值 = 比较值)
--- 标签1.标题 = “相等”
--- 标签1.标题 = “不相等”
↓

```

变量名	类 型	静态	数组	备 注
被比较值	整数型			
比较值	整数型			

```

--- 如果 (被比较值 = 比较值)
--- 标签1.标题 = “相等”
--- 标签1.标题 = “不相等”
↓

```

上述代码中，条件语句“如果（）”中的“被比较值=比较值”，是用关系运算符“=”进行比较，如果相等会返回“真”，不相等会返回“假”，如果返回“真”将会执行：标签1.标题=“相等”；如果返回“假”将会执行：标签1.标题=“不相等”，这2行给标签标题属性赋值的代码中，使用的就是赋值运算符“=”。

2.8.3 文本运算符与文本表达式

字符串在易语言中被称为文本，其运算符只有一个：

+ 连接运算符

其功能是将两个文本连接起来。

例如：

```
“ABCD” + “123”
```

将得到一个新的文本，其值为“ABCD123”。

可见，可用字符串运算符连接两个字符串，构成字符串表达式。

2.9 习题

1. 指明下列各数据，哪些是合法的常量。

- (1) 1E+39 (2) 359, 672001 (3) “ABC 34” (4) 29 • “AB” (5)

32.001

2. 指明下列各项，哪些是合法的变量名。

- (1) AA (2) A.5% (3) 2A\$ (4) AB\$% (5) BBC%

3. 将下列各数转换成常规计数法表示的数。

- (1) 8.2345E+05 (2) 2.3456E-06 (3) 1.259E+09

4. 写出下列各表达式中运算符的执行顺序。

- (1) $3*5+2*(68/(2+3))$ (2) $A+B*6/4+8\%165+2*8$ (3) $A+B-85\%3$

5. 指出下列各程序行中有哪些错误。

- (1) 变量 1=5, 变量 2=6 (2) 变量 1+2=变量 2+3 (3) “文本数据” (4)

3.1416*变量 1

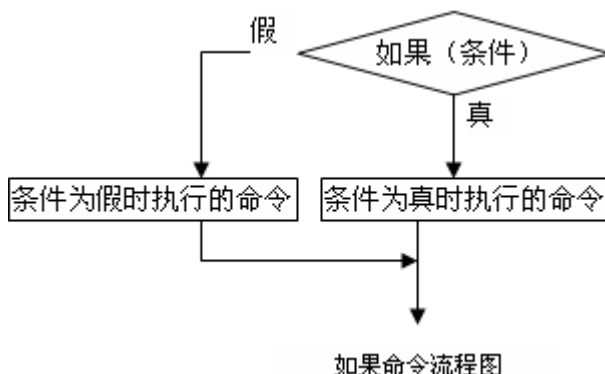
6. 给出下列表达式的逻辑值：

- (1) $5>3 \ \&\& \ 6>4$ (2) $3+5>6 \ \parallel \ 7=6$ (3) $A>A \ \&\& \ 4*4*2>2$ (4) $5\parallel 3<3$
(5) $5=6=7=0$

第三课. 程序流控制命令

3.1 “如果 ()” 命令

“如果 ()” 命令的参数为一个逻辑型数据，非真即假。若条件为真，则程序顺序执行后续代码；若条件为假，则程序跳转到左箭头所示的代码行继续运行。



新建一个“Windows 窗口程序”，在“_启动窗口”中添加 1 个编辑框组件和一个按钮组件。然后双击按钮，在“_按钮 1_被单击”子程序中输入代码：

```

--- 如果 (编辑框1.内容 = "")
--- 编辑框1.内容 = "Windows窗口程序"
--- 编辑框1.内容 = ""
  
```

运行程序。此时编辑框中没有任何内容，单击按钮，可以看到编辑框显示“Windows

窗口程序”；再单击按钮，此时编辑框中已有内容，则编辑框显示“”。

其实上面代码中的：

如果（编辑框.内容=“”）

解释为，编辑框中的内容是空文本（条件语句的值为真），程序可以向下顺序执行。

可以看出，条件语句的值只能是逻辑型的“真”或“假”。

“如果（）”命令参数中还可以填写多个条件，用“或”和“且”连接，例如：

```

--- 如果（比较值 = 100 或 比较值 = 50 或 比较值 = 25）
--- 比较值 = 0
--- 比较值 = 比较值 × 5

```

用“或”连接多个条件时，只要有一个条件成立时，整个条件参数就为真。上述代码中，当“比较值”等于 100、50 或 25 时，“比较值”被改为 0，否则就等于它自身的五倍。

变量名	类 型	静态	数组	备 注
比较值	整数型			

```

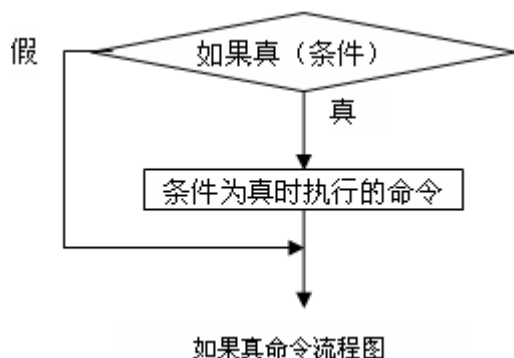
--- 如果（比较值 = 100 且 比较值 = 50）
--- 比较值 = 0
--- 比较值 = 比较值 + 1

```

用“且”连接的多个条件，必须所有条件都成立时，整个条件参数才为真。上述代码中，只有当“比较值”等于 100，并且“比较值”等于 50 时，“比较值”被改为 0，否则“比较值”会加 1。

3.2 “如果真（）”命令

“如果真（）”命令从流程线上就可以看出与“如果（）”命令的不同，“如果真（）”命令在条件成立的时候运行“如果真（）”命令下的代码，否则“如果真（）”命令没有任何动作。



例如：

```

--- 如果真（比较值 > 100）
--- 比较值 = 0

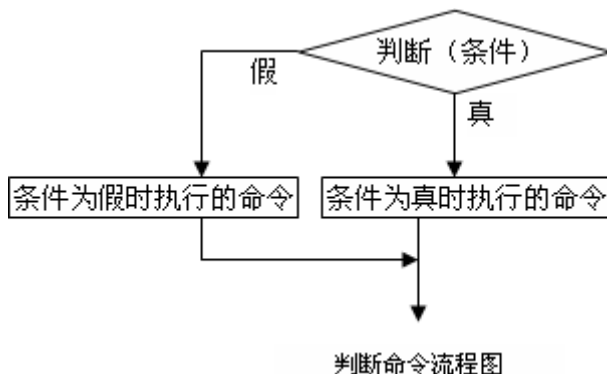
```

当“比较值”大于 100，则会运行“比较值=0”的代码，否则直接跳到判断结束后的代码继续运行。

3.3 “判断（）”命令

本命令根据提供的逻辑参数的值，来决定是否改变程序的执行位置，如果提供的逻辑参数值为“真”，程序将跳过同组后面的“判断”语句继续顺序向下执行，否则跳转到下一分支处去继续判断。

“判断（）”命令主要用于条件的分支选择，如下面2段代码运行效果相同，程序结构也相同，但使用“判断（）”命令，代码流程结构要清晰许多，而使用“如果（）”命令，不仅会使程序嵌套太多，程序代码难以看清楚，也降低了程序运行效率。



3.4 习题

1. 指出下列程序的运行结果：

变量名	类型	静态	数组	备注
整数变量	整数型			

```

整数变量 = 1
--- 如果真（整数变量 < 5）
    整数变量 = 整数变量 + 1
↓ 输出调试文本（到文本（整数变量）
  
```

2. 编写程序，由键盘输入弧度“输入值”值，按下列公式求“求出值”：

$$\text{求出值} = \begin{cases} \frac{\text{求正弦（输入值）} + \text{求余弦（输入值）}}{2} & \text{输入值} \geq 0 \\ \frac{\text{求正弦（输入值）} - \text{求余弦（输入值）}}{2} & \text{输入值} < 0 \end{cases}$$

3. 一个三角形的三个边长给出后，可通过下列公式求出三角形的面积：

$$\text{面积} = \sqrt{P(P - \text{边1})(P - \text{边2})(P - \text{边3})}$$

其中

$$P = \frac{\text{边1} + \text{边2} + \text{边3}}{2}$$

试编写此程序。要求程序能检查出：当给出的三个边长构不成三角形的情况，并提示数据错误，再由用户重新给出三个边长。

4. 某学校规定：期中考三门课，其中两门为主课。凡考分符合下列条件之一的学生，

发给“优良成绩奖”：

- (1) 三门课总分在 280 以上者；
- (2) 两门主课成绩均在 95 分以上，其余一门课考分在 80 分以上者；
- (3) 有一门主课为 100 分，另两门合计不低于 160 分者。

试编写法一个程序，对得奖者能打印出学号与各门课的考分。对无奖者只打印出学号来。

(学号、考分自选)

5. 试写一个程序，把下列数据中的每个负数都打印出来，并打印所有正数之和：

-2, 3, 15, -86, -7.3, 90, -80, 7, 42, -8

6. 写一个程序，用来产生一批从 0 到 20 的整型随机数，当得到的随机数正好为 15 时，结束程序。

第四课。程序的循环和跳转

本章将介绍循环程序设计的概念。循环语句是专用于这类程序设计的，由于它有着固定的格式和执行方法，使用起来十分方便。

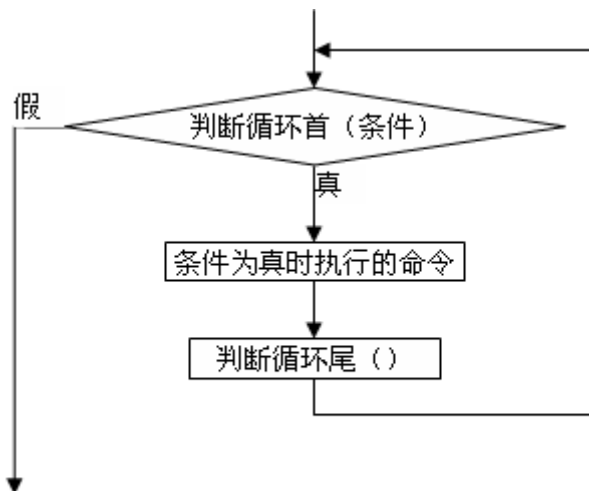
4.1 循环类流控制命令

循环类流程控制命令可以在一定条件下多次执行重复的代码。例如，将某数据库中前 100 条记录的“姓名”字段内容读出并显示在表格中，使用循环命令只需要几行代码即可实现。

循环流程类命令都由循环首和循环尾 2 部分组成，输入了循环首命令，循环尾就自动出现。循环首表示循环的开始，循环尾表示循环的结束，循环首和循环尾之间的代码，是循环执行的代码。

1. “判断循环首 ()” 和 “循环判断首 ()” 命令

“判断循环首 ()” 命令首先检查判断条件是否成立。如果不成立，直接跳到循环尾后的代码继续执行；如果条件成立，则进入循环。每次循环结束后，会再一次检查“判断循环首”中的条件，如果条件不成立了，就退出循环，执行后续代码。



判断循环首命令流程图

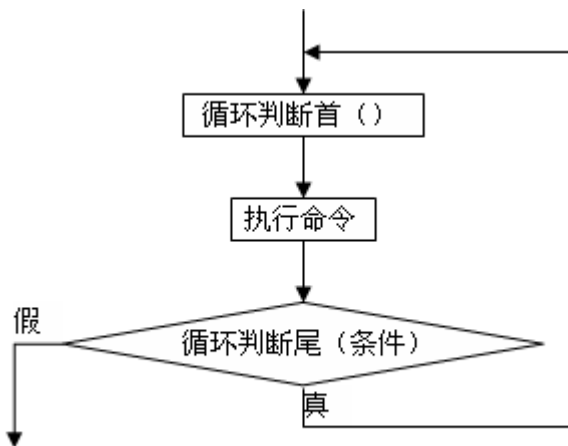
例如：让画板滚动写出 100 以内的偶数，“Windows 窗口程序”代码如下：

变量名	类 型	静态	数组	备 注
变量1	整数型			

```

--> 判断循环首 (变量1 < 100)
    变量1 = 变量1 + 2
--> 画板1.滚动写行 (变量1)
-- 判断循环尾 ()
    
```

“循环判断首 ()”命令是先循环再判断，即首先运行一次循环首和循环尾之间的代码，再判断条件是否成立。如果“循环判断尾 ()”中的条件为真，就跳到循环首处继续循环，如果条件不成立，则循环终止。将上面的代码中的“判断循环首”直接转换为“循环判断首”，运行结果是相同的。可以看出，这两个命令在一定情况下是可以互换的，但由于两个命令的判断位置不同，有可能对循环体内的运行结果造成影响，在实际应用中要注意区分。



循环判断首命令流程图

下面编写用列表框组件列出 C 盘根目录下所有文件的程序。

首先，新建一个“Windows 窗口程序”，然后添加 1 个列表框组件和 1 个按钮组件。双击按钮组件，在“_按钮 1_被单击”子程序中输入代码：

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

变量名	类 型	静态	数组	备 注
文件名称	文本型			

```

文件名称 = 寻找文件 ("c:\*.*", )
--> 判断循环首 (文件名称 != "")
    列表框1.加入项目 (文件名称, )
    文件名称 = 寻找文件 ( )
-- 判断循环尾 ()
    
```

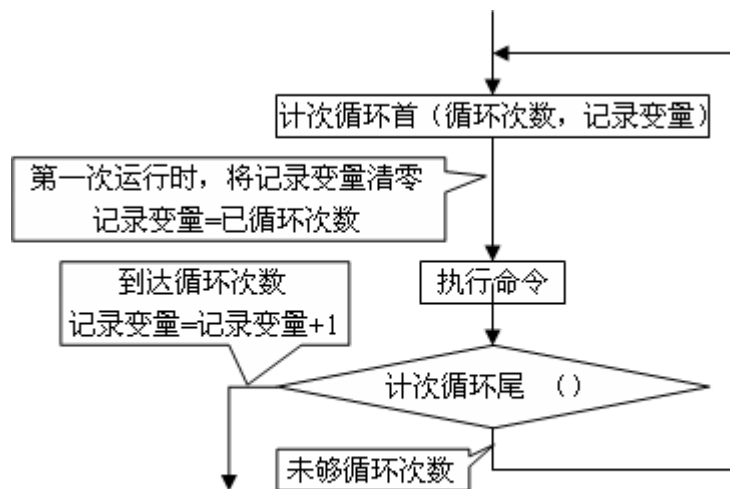
这里用到了“寻找文件 ()”命令，当使用“寻找文件 ()”命令在指定目录连续检索相同条件的文件（非子目录）时，第二次调用无需填写参数，该命令会自动继续向下寻找。

运行程序，程序会将 C 盘根目录中所有的文件，包括被隐藏文件和系统文件都显示在列表框中。

可尝试将“判断循环首 ()”命令转换成“循环判断首 ()”命令，看运行结果是否有所不同。

2. “计次循环首 ()”命令

“计次循环首 ()”命令可以指定循环的次数。命令的第二个参数可以填入一个变量，用来记录已循环次数，第一次循环时变量值为 1，第二次循环时变量值为 2，依此类推。



计次循环首命令流程图

下面编写一个将 1 到指定范围内的所有整数相加的程序。

创建“Windows 窗口程序”，在“_启动窗口”中添加一个编辑框组件和一个按钮组件。

然后将编辑框的“输入方式”属性设置为“整数文本输入”，双击按钮，在“_按钮 1_被单击”的子程序中输入代码：

变量名	类型	静态	数组	备注
循环次数	整数型			
相加	整数型			

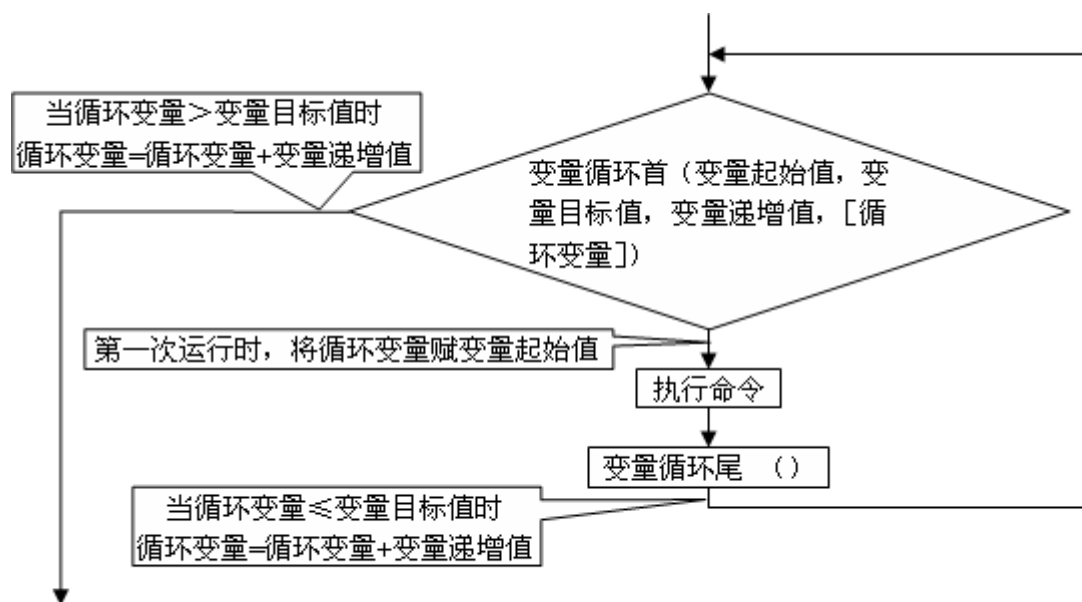
```

--> 计次循环首 (到数值 (编辑框1.内容), 循环次数)
    相加 = 相加 + 循环次数
-- 计次循环尾 ()
    
```

最后，试运行程序，在编辑框中输入一个大于 1 的整数，然后点击按钮，可以求出 1 到编辑框中的数值范围以内所有整数的和。但要注意的是，如果输入的数过大，就会导致循环的时间过长，程序暂时失去响应；如果输入的数值范围太大，可能会出现数据溢出情况。

3. “变量循环首 ()”命令

该命令用做一个变量的内部循环，有四个参数，规定了变量的起始值，目标值和递增值，每次循环，变量的起始值都会增加规定的递增值，直到达到目标值，退出循环。第四个参数记录当前起始值，类似“计次循环首 ()”命令的记录循环次数的变量。



变量循环首命令流程图

例如，可以求出 100 到 200 之间的整数和，用以下代码：

变量名	类型	静态	数组	备注
循环次数	整数型			
相加	整数型			

```

--> 变量循环首 (100, 200, 1, 循环次数)
    相加 = 相加 + 循环次数
-- 变量循环尾 ()
编辑框1.内容 = 到文本 (相加)
    
```

循环结束后的“相加变量”就是求得的结果。最后将结果显示在编辑框中。

4.2 循环的套用

循环流程命令可以嵌套使用。

在循环体内再次使用循环语句，构成了循环语句的嵌套使用。大循环内套一个小循环，小循环内还可以再套一个循环。一个套一个，所套的次数，叫做循环嵌套深度。

多重循环在解决应用问题中，是非常有用的。

例如，下面就用易语言做一个简单的程序：

打印乘法表的 $1 \times 1 = 1$, $1 \times 2 = 2$, $1 \times 3 = 3$, ..., $1 \times 9 = 9$

可以使用“计次循环首 ()”命令。新建“Windows 控制台程序”。程序代码如下：

变量名	类 型	静态	数组	备 注
乘数	整数型			
被乘数	整数型			
乘积	整数型			

```

--> 计次循环首 (9, 乘数)
    --> 计次循环首 (乘数, 被乘数)
        乘积 = 被乘数 × 乘数
        标准输出 (, 到文本 (被乘数) + "*" + 到文本 (乘数) + "=" +
            到文本 (乘积) + " ")
    -- 计次循环尾 ()
-- 标准输出 (, #换行符)
-- 计次循环尾 ()
标准输入 ()
返回 (0) ' 可以根据您的需要返回任意数值
  
```

这个程序在输出格式上作了一些考虑。在每输出一次运算结果(内层循环中的输出语句)的后面加上两个空格,以便使下次输出能与之保留距离。在外层循环中的输出语句是使以后的输出能另起一行。这样就可以显示出格式正确的九九表。

双重循环语句的执行过程中,程序总是先遇到最外层那个循环。这时你就可以把内层循环当作一个语句来看待。最外层循环执行过程中,是将其内部的每条语句顺序执行的,而遇到内层循环时,我们既把它当作一个语句,也一样要执行这个语句。然而它毕竟是一个循环语句,所谓执行它,还是按一般循环语句的执行方法。因此,就事论事刚刚列出的显示九九表的程序而言,每对“乘数”循环一次,就要对“被乘数”完整的循环当前“乘数”变量中的次数。

三重循环、四重循环等,执行过程依此类推。

4.3 跳转类流程控制命令

有了跳转类流程控制命令,可以更加方便的控制循环,“返回 ()”和“结束 ()”命令可以控制子程序和整个程序的结束。

1. “到循环尾 ()”和“跳出循环 ()”命令

这两个跳转类流程控制命令都是用来控制循环的。当一个循环中运行了到“到循环尾 ()”命令,将会直接跳到循环尾的代码处,当一个循环中运行了“跳出循环 ()”命令,那么当前的循环就会结束,然后继续运行循环体以后的代码。例如:

(1) 让一个编辑框中只显示 10 以内的奇数 “1, 3, 5, 7, 9”,代码如下:

变量名	类 型	静态	数组	备 注
循环变量	整数型			

```

--> 计次循环首 (9, 循环变量)
    -- 如果真 (循环变量 % 2 = 0)
        到循环尾 ()
    编辑框1.内容 = 编辑框1.内容 + 到文本 (循环变量) + ", "
-- 计次循环尾 ()
  
```

当循环变量等于不想显示出的值,就会执行“到循环尾 ()”命令,跳过显示到编辑框

的代码。

(2) 如果在循环过程中，当某个条件产生，想提前结束该循环，可以使用“跳出循环 ()”命令，如：

变量名	类型	静态	数组	备注
循环变量	整数型			

```

--> 计次循环首 (9, 循环变量)
  -- 如果真 (循环变量 = 8)
    跳出循环 ()
    编辑框1.内容 = 编辑框1.内容 + 到文本 (循环变量)
  -- 计次循环尾 ()
  
```

当循环变量等于 8，就运行“跳出循环 ()”命令，循环即会结束，所以编辑框会显示“1234567”。

2. “返回 ()”命令

“返回 ()”命令被执行后，会退出当前子程序。当前子程序中“返回 ()”命令之后的代码将不再被执行，程序将自动跳转到调用本子程序语句的下一条语句处继续执行。

由于“返回 ()”命令后面的代码不被运行，所以“返回 ()”命令也经常被用于程序的侦错，在后面的章节会有介绍。有返回值的子程序必须使用“返回 ()”命令来返回执行结果，要注意实际返回值的数据类型要和子程序定义的返回值数据类型相匹配。例如返回日期时间型的数据：

```

返回 ([2004 年 11 月 15 日])

```

变量名	类型	静态	数组	备注
变量1	整数型			

```

--> 计次循环首 (20, 变量1)
  -- 如果真 (变量1 = 10)
    返回 ()
    编辑框1.内容 = 到文本 (变量1)
  -- 计次循环尾 ()
  
```

当变量 1 等于 10 的时候，就运行“返回 ()”命令，当前循环和当前子程序都会被终止。

3. “结束 ()”命令

“结束 ()”命令是结束当前易程序的运行。不论当前易程序有多少个已载入窗口，都会被自动关闭。可以用于程序的关闭按钮。“结束 ()”命令没有返回值也没有参数。若只想关闭当前的某一个窗口，应使用“窗口”的“销毁 ()”方法。

例如，在按下“关闭按钮”后，就将结束程序：

子程序名	返回值类型	公开	备注
_关闭按钮_被单击			

```

结束 ()

```

注：建议尽量在程序中使用销毁所有已载入窗口的方法来结束程序。

4.5 习题

1. 下列程序的运行结果是什么？

变量名	类 型	静态	数组	备 注
变量1	整数型			
计数变量	整数型			

→ 变量循环首 (1, 10, 1, 计数变量)

计数变量 = 计数变量 + 1

变量1 = 变量1 + 1

→ 变量循环尾 ()

» 输出调试文本 (到文本 (计数变量) + “ ” + 到文本 (变量1))

2. 改正下列程序中的错误：

变量名	类 型	静态	数组	备 注
变量1	整数型			
计数变量	整数型			

→ 变量循环首 (1, 5, 1, 计数变量)

变量1 = 1

变量1 = 变量1 × 计数变量

→ 变量循环尾 ()

» 输出调试文本 (“从1到5的阶乘是： ” + 到文本 (变量1))

3. 编写程序，用来计算并打印 100 个数中正数的平方根的算术平均值。（它们可有正、有负）

4. 编写一个程序，求出乘积为 399 的两个相邻奇数。

5. 鸡兔同笼。共有头 M 个，脚 N 只，编写一个程序，求鸡兔各多少只。

6. 100 匹马驮 100 担货，规定大马一匹驮 3 担，中马一匹驮 2 担，小马 2 匹合驮 1 担。要 100 匹马正好驮完这 100 担货，且不许有一匹马驮不够规定数。试编写一个程序，求大、中、小马各应有几匹？

7. 用循环语句写一个程序，以计算把 100 元人民币换成零钱有多少种换法。

第五课. 子程序

搭积木时把一个个不同形状，不同大小的木块作不同的组合，可以搭出各种各样的模型来。把一个个具有特定功能的程序段作为若干个独立块，再把它们作不同的组合，也可以构成一个完整的程序，这种方法称作模块化结构程序设计方法。

当一个具有特定功能的程序段在一个程序中多次被使用到时，如果在每次使用它的地方都写一遍，不仅写起来麻烦，输入机器时也费时间，还容易增加出错机会。而且这种程序较长，占用内存空间大，读起来也讨厌。可以设想，如果能只把它们写入一次，需要使用它们时就调用它们一次，那不就简单了吗？

其实子程序就是用来对一系列命令进行封装，实现模块化、重用及抽象，从而有利于程序的结构化开发，使程序结构更清晰。合理划分代码结构是软件成功的基本保障。

5.1 子程序的分类

易语言中的子程序可以分为两大类：“事件子程序”和“用户自定义子程序”。

对应组件所发生事件的子程序，称作组件的“事件子程序”。例如：按钮有被单击、被双击、左键被按下等事件，选中按钮后，在属性面板中的事件列表中选择这些事件，就可以

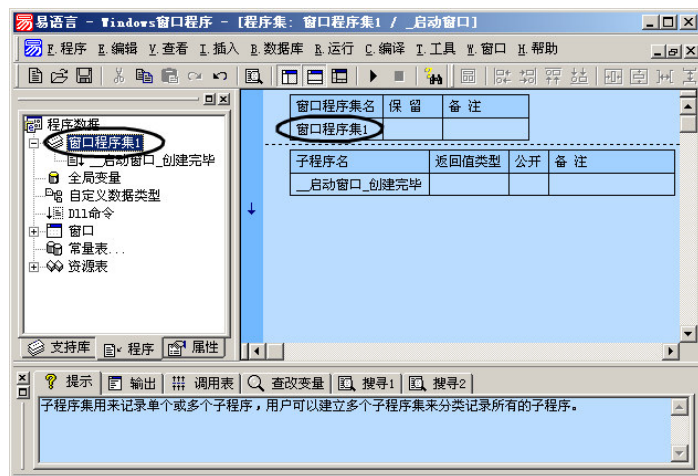
生成对应的事件子程序，大家可以在这些事件子程序中写入具体执行的代码。运行时，一旦这些事件产生，就会自动执行相应的子程序。事件子程序的名称、返回值类型和参数个数都是系统定义的，不允许用户任意修改。

“用户自定义子程序”是由用户创建，其参数个数和返回值都由用户自行定义的程序。用户可以根据需要在程序设计时对其任意修改。

使用子程序的好处很多，例如有段代码在程序中多处被重复调用，此时就可以将其编写到一个子程序中，不仅减少了代码输入的重复劳动，而且需要修改这段代码时，只要修改一个地方即可，而不用在程序中逐个修改，即实现重用；再例如要实现一个相对复杂的功能，如果全部代码写在一起，发现错误就无法确认问题来源，此时可将问题分解为多个简单问题，使用子程序逐个实现，这样有利于提高代码的正确性和清晰度。

5.2 用户自定义子程序的创建

子程序是存在于程序集中的，由程序集分组管理。若一个新建的易程序还没有进行任何操作是没有程序集的，这时可以使用菜单：“插入”→“程序集”来新建一个程序集放置自己所写的用户自定义子程序；或通过在窗体空白处双击鼠标，会自动创建该窗口对应的窗口程序集，并自动生成“_启动窗口_创建完毕”子程序（创建完毕事件子程序可包含本窗口创建时被执行的程序代码）；或双击按钮等组件，也会创建该窗口对应的窗口程序集及组件对应缺省事件子程序。此时就可以在当前窗口程序集中添加新的子程序了。如下图所示。



创建窗口程序集

用户自定义子程序可以放在窗口程序集中，或自定义的程序集中，当然，建议您将当前窗口用到的用户自定义子程序放在当前窗口程序集中，这样就可以非常方便地引用程序集变量和窗口中的组件，而且查找起来也较为方便。

2. 若程序集已建立，可通过程序面板切换到程序集中操作，或通过窗口菜单切换。

定义一个新的子程序可以通过在程序编辑界面按下“Ctrl+N”键；或在程序编辑界面点击鼠标右键，在弹出菜单中选择第一项“新子程序”；或者选择易语言菜单“插入”→“新子程序”来新建子程序。

创建后的子程序可以根据需要修改其名称，建议在定义子程序名称时，尽量选择能体现

其功能的名称，如“求和”、“统计字符”等。

5.3 子程序的调用

子程序的调用方法和命令的调用方法相同，输入子程序的名称就可以调用该子程序。如果子程序要求提供参数，在括号中要按照子程序参数定义的数据类型和参数个数顺序填写参数；如果需要用变量保存子程序返回值，必须使用和子程序返回值相同的数据类型变量。

5.4 返回值和参数的定义

使用“返回（）”命令，可以将子程序处理后的结果返回给调用它的程序使用。

为子程序定义返回值，需要在子程序的“返回值类型”单元格中定义其返回值的数据类型，同时，在子程序任意一个结束分支中，都必须使用“返回（）”命令将欲返回的值回传。

为子程序增加参数定义，首先在子程序名称上按回车键增加一个空白参数定义，修改该参数的名称和数据类型即可。参数的“类型”如果为空，系统默认为整数型参数。如果某项参数的“可空”类型未被设置，在调用该子程序时就必须为此参数提供初始值。

子程序的参数在当前子程序内可以当作变量引用。例如：

子程序名	返回值类型	公开	备 注		
相加运算	整数型				
参数名	类 型	参考	可空	数组	备 注
加数1	整数型				
加数2	整数型				

返回 (加数1 + 加数2)

上述子程序定义中，子程序名为“相加运算”，返回值类型为“整数型”，子程序有 2 个整数型参数，分别为“加数 1”和“加数 2”。

该子程序可以对参数做相加运算，并将运算结果回传。调用该子程序的方法如下：

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

编辑框1.内容 = 到文本 (相加运算 (1345462, 33456730))

5.5 子程序嵌套

与循环嵌套类似，子程序中再调用另一个子程序，叫子程序嵌套。而调用子程序本身，则被称作递归调用。

5.6 子程序指针

易语言中部分命令必须以子程序在内存中的起始地址作为参数，如“启动线程（）”命令；使用某些外部 API 函数时，也同样需要知道调用的子程序在内存中的起始地址。在易语言中，以子程序指针型变量记录子程序在内存中的起始地址。赋值方法为“&”+子程序名。如：

变量 1 = &子程序 1

启动线程（变量1，）

赋值之后，“变量1”（类型为“子程序指针型”）存储的就是“子程序1”的可执行代码首地址。

5.7 子程序参数的“参考”属性

子程序参数的“参考”属性用于设置系统为当前子程序参数传递数据时是否为传递指向数据的指针。如果所传递过来的参数数据为数组、用户定义数据类型、库定义数据类型、文本型、字节集型数据，则无论此属性是否为“真”（选中），都将传递指针。如果所传递过来数据的类型与相应位置处参数的数据类型不一致但可以相互转换。例如将“整数型”数据传递到“小数型”的参数中，则在数据被实际传递前，系统将首先自动将“整数型”数据转换为“小数型”数据，然后再进行传递。因此在这种情况下，即使本属性为“真”，系统也无法传递指向原数据的指针，只能传递数据本身。如果系统将数据指针成功地传递过来，那么在子程序中对此参数内容的更改将会相应地反映到调用子程序时所提供的参数数据上。

给子程序参数定义“参考”属性，只要将属性相应位置打上“√”即可。

下面通过一个例程来观察“参考”属性的作用。

新建一个“Windows 窗口程序”，然后在“_启动窗口”中添加一个画板组件和一个按钮组件。

双击按钮组件，在代码编辑界面新建一个子程序，定义子程序名为“测试”，为该子程序定义2个参数，均为整数型；参数名分别设为“参考参数”和“非参考参数”，将“参考参数”的“参考”属性上打上“√”：

子程序名	返回值类型	公开	备 注		
测试					
参数名	类 型	参考	可空	数组	备 注
参考参数	整数型	√			
非参考参数	整数型				

参考参数 = 1000

非参考参数 = 1000

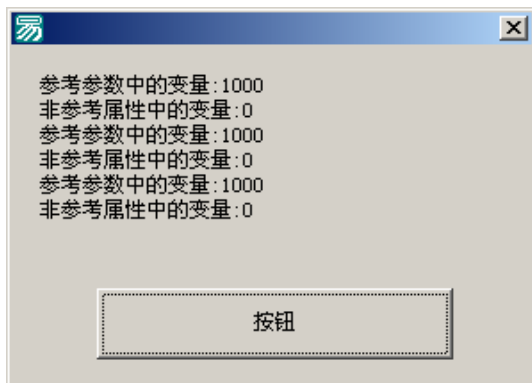
在“_按钮1_被单击”子程序中输入代码：

子程序名	返回值类型	公开	备 注		
_按钮1_被单击					
变量名	类 型	静态	数组	备 注	
变量1	整数型				
变量2	整数型				

测试（变量1，变量2）

» 画板1.滚动写行（“参考参数中的变量:” + 到文本（变量1），“非参考属性中的变量:” + 到文本（变量2））

运行程序，观察运行结果。如下图所示。



“参考”属性对变量的影响比较

上述程序中，每次点击按钮，就会调用“测试”子程序对2个参数进行修改。从画板的显示结果可以看出，对应“参考参数”的变量1，因子程序内部对相应参数的操作而被修改，而对应“非参考参数”的变量2没有发生任何变化。

5.8 子程序参数的“可空”属性

易语言自带的支持库命令中有一些参数可以被省略，一般查找帮助时可以看到可省略的部分用中括号括起来了。如“时间到文本（）”命令的第二个参数是可以省略的，它也可有一个默认的数据，此命令帮助如下：

调用格式：〈文本型〉 时间到文本（欲转换到文本的时间，[转换部分]）

将指定时间转换为文本并返回。

参数<1>的名称为“欲转换到文本的时间”，类型为“日期时间型（date）”。

参数<2>的名称为“转换部分”，类型为“整数型（int）”，可以被省略。参数值可以为以下常量：1、#全部转换；2、#日期部分；3、#时间部分。如果省略了本参数，默认为“#全部转换”。

用户自定义子程序也可以实现这样的功能。

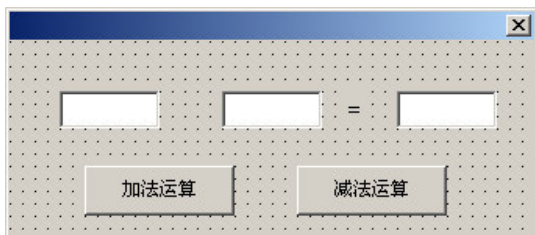
在设计子程序的时候，有些参数可能经常用到相同的数据。如果每次调用该子程序，均需手工指定此数据作为参数初始值，那是一件非常繁琐的事情。在这种情况下，可以将此数据作为该参数的默认值，内置在子程序中，并将参数属性设为“可空”类型。当子程序被调用时，该参数没有指定具体的初始值，就为该参数赋予默认值参与运算。如果某项参数的“可空”类型未被设置，在调用该子程序时就必须为此参数提供数据。

定义“可空”属性的方法和定义“参考”属性相同，将该属性打上“√”即可。如果需要检测当前子程序被调用时，该参数是否传递了数据，可用“是否为空（）”命令进行确认。

下面编写一个可以进行加法或减法运算的子程序，来了解“可空”属性的实际应用。

本例程将加减运算的功能制作成一个子程序，通过第三个参数确定是作加法计算还是减法计算，如果第三个参数为空进行加法运算，不为空则进行减法运算。

第一步，新建一个易程序，在窗口中添加3个编辑框组件，2个按钮组件和1个标签组件，将按钮组件的标题分别改为“加法运算”和“减法运算”，标签组件的标题改为“=”。如下图所示。



加减法运算例程界面

第二步，切换到代码编辑界面，新建一个子程序，将子程序名改为“加减运算”。给该子程序定义3个整数型参数，参数名分别为“被运算数”、“运算数”和“进行的运算”，将“进行的运算”参数的“可空”属性被设置，在子程序中输入如下程序代码：

子程序名	返回值类型	公开	备 注		
加减运算	整数型				
参数名	类 型	参 考	可 空	数 组	备 注
被运算数	整数型				
运算数	整数型				
进行的运算	整数型		✓		

```

--- 如果 (是否为空 (进行的运算))
--- 返回 (被运算数 + 运算数)
--- 返回 (被运算数 - 运算数)

```

子程序中，“进行的运算”参数控制子程序进行何种运算，该参数为空进行加法运算，该参数不为空则进行减法运算。

第三步，回到“_启动窗口”双击“减法运算”按钮，然后在“_按钮1_被单击”和“_按钮2_被单击”子程序中分别输入代码：

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

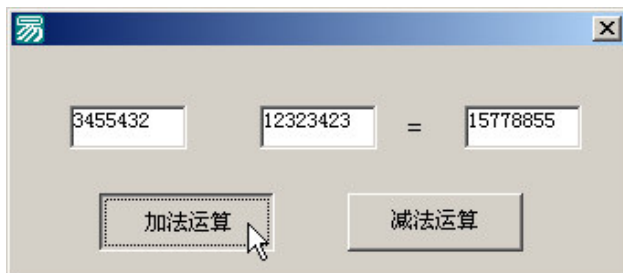
```
编辑框3.内容 = 到文本 (加减运算 (到数值 (编辑框1.内容), 到数值 (编辑框2.内容), ))
```

子程序名	返回值类型	公开	备 注
_按钮2_被单击			

```
编辑框3.内容 = 到文本 (加减运算 (到数值 (编辑框1.内容), 到数值 (编辑框2.内容), 1))
```

可以看到，在“_按钮1_被单击”子程序中，省略了“加减运算”子程序第3个参数的填写。

最后，运行程序，查看运行后的效果。如下图所示。



加减法运算例程的运行效果

程序运行时点击左边按钮，程序运行的“加减运算”子程序第三个参数没有填参数，因此进行的是加法计算。点击右边按钮，程序运行的“加减运算”子程序第三个参数设置参数为 1，因此进行的是减法计算。本例程可以参考随书光盘中的例程“加减运算.e”。

5.9 子程序参数的“数组”属性

子程序参数也可以接收数组，如系统的“播放 MP3 (,)”命令的第二个参数就可以接收数组。而用户自定义子程序也可以实现这个功能，接收一个数组，在子程序内部进行处理。

子程序参数定义了“数组”属性后，就可以给这个参数中填写一个数组变量。

子程序的参数如果是“数组”型变量，就自动具有了“参考”属性。

定义参数的“数组”属性，在该参数的数组属性上打“√”。

下面就编写一个例程，来了解数组参数的作用。

第一步，新建一个易程序，在窗口中添加一个画板和一个按钮组件。

第二步，双击按钮组件，在程序设计界面新增一个子程序，然后将子程序名改为“测试数组属性”，给该子程序定义一个参数，将参数的名称改为“数组”，并将参数的数组属性选中：

子程序名	返回值类型	公开	备 注		
测试数组属性					
参数名	类 型	参考	可空	数组	备 注
数组	整数型			√	

数组 = { 100, 200, 300 }

第三步，在按钮 1 被单击的子程序中输入代码：

子程序名	返回值类型	公开	备 注		
_按钮1_被单击					

变量名	类 型	静态	数组	备 注
数组	整数型		3	

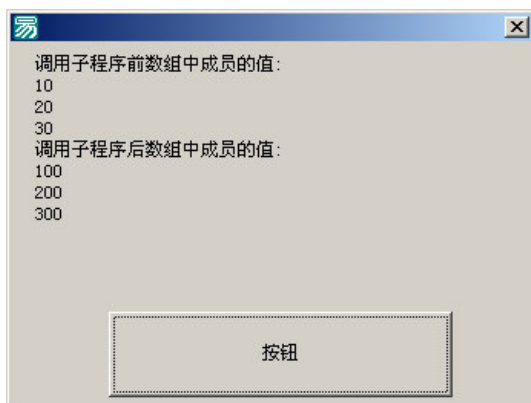
数组 = { 10, 20, 30 }

” 画板1.滚动写行 (“调用子程序前数组中成员的值:”，到文本 (数组 [1]),
到文本 (数组 [2]), 到文本 (数组 [3]))

测试数组属性 (数组)

” 画板1.滚动写行 (“调用子程序后数组中成员的值:”，到文本 (数组 [1]),
到文本 (数组 [2]), 到文本 (数组 [3]))

最后试运行程序，查看运行效果，如下图所示。



数组属性例程运行效果

运行后，画板显示出调用子程序前后数组变量中各成员的值，在调用子程序后，可以看出数组参数自动具有了参考属性，数组变量中成员的值发生了改变。

5.10 习题

1. 用子程序方法写出求 3 的阶乘加 4 的阶乘加 5 的阶乘的程序来。
2. 利用程序流控制命令，控制程序对一组数据的不同的处理：对读入的零，只统计次数（读入多少次 0）；对读入的正数，求出均值；对读入的负数，只找其两个边界值。数据总数不会多于 10000 个。作为标志：当读到-999 时，控制程序结束。还要求当读入的数据不为数字数据时，能做出错处理。并将各功能组织为不同的子程序。

第六课．数组

到现在为止，我们在前几章中所用到的各种变量，都有一个特点，就是在程序运行的任何时刻，每个变量名都能代表一个数据。这种变量名，我们叫做简单变量。

用简单变量固然可以编写出各种各样的程序。但是如果在程序中有成组地出现的数据时，程序的编写与处理，将是十分麻烦的。

假定我们要处理 200 个学生的考试成绩，每人考 6 门课，共有 1200 个分数。这 1200 个分数如果要反复使用，最好的办法是把这 1200 个数一次读入 1200 个变量，否则每次使用得从头读一遍。1200 个变量在内存中，随时可以使用，这当然很方便，然而这是不可能的。而对于一个数组中的多个不同元素，只需取多个不同的下标即可。而且下标又可以通过程序的运行予以改变，这就给编写程序提供了较大的方便。

数组变量可以存放一组数据，数组的成员数可以自行定义，并且在程序运行后还可以动态改变，赋值方法也多种多样。即可以动态的管理数组变量。所以数组变量是使用非常灵活、应用非常广泛的一种变量。

数组变量又分为“单维数组变量”和“多维数组变量”。

单维数组变量，成员的表示形式为：

数组变量名 [数组成员下标]

如：变量 [2]，表示该数组中的第 2 个成员。注意，易语言中的数组下标是从 1 开始的，根本不存在“变量[0]”这个数组成员。

多维数组变量，可以看作是一个特殊的单维数组，它的成员也可以看作是多维数组。多维数组成员的表示形式为：

多维数组变量名 [数组成员下标] [数组成员下标]

如：变量 [1] [2]，意思是一个二维变量中的第二个成员。

6.1 数组变量的定义及赋值

1. 数组变量的定义方法

在欲定义变量的数组属性上输入要定义的成员数。例如，要将“变量”定义为有 3 个成员的数组变量，就在“变量”的数组属性上输入 3，定义后的“变量”就有变量[1]、变量[2]和变量[3]这三个数组成员。如下图所示。

变量名	类型	静态	数组	备注
变量	整数型		3	

定义数组变量

2. 多维数组变量的定义方法

在欲定义的“变量”的数组属性上输入：成员数，成员数，…。例如，要定义一个 2 维数组变量，每个维有 3 个成员，就在“变量”的数组属性上输入“3, 3”如下图所示。

变量名	类型	静态	数组	备注
变量			3, 3	

二维数组定义

这个 2 维数组，可以看作是 3 个拥有 3 个成员的单维数组组成，这个数组中各个成员，按成员顺序排列，表示方法如下：

变量[1][1]、变量[1][2]、变量[1][3]、变量[2][1]、变量[2][2]、变量[2][3]、变量[3][1]、变量[3][2]、变量[3][3]

在易语言中，多维数组成员也可以以单维数组的表示方法来访问，上边的多维数组有 9 个成员，按成员顺序排列也可以表示为：

变量[1]、变量[2]、变量[3]、变量[4]、变量[5]、变量[6]、变量[7]、变量[8]、变量[9]

数组变量也可以直接称之为数组，本书中以后如没有特别声明，所提到的数组都是指的数组变量。

3. 数组变量的赋值

(1) 直接赋值

数组变量的赋值，就是给数组中的成员赋值，每个成员都有独立的存储空间。数组中的每个成员都可以看作是单独的变量，可以使用给变量赋值的方法来给数组的成员赋值，例如，给一个有 2 个成员的整数型数组赋值，让每个成员都为 100，程序代码为：

```
变量[1] = 100
```

```
变量[2] = 100
```

例如，给一个 2 维的整数数组赋值，每个维有 2 个成员，每个成员都赋值 100：

```
变量[1][1] = 100
```

```
变量[1][2] = 100
```

```
变量[2][1] = 100
```

变量 [2] [2] = 100

(2)连续赋值

给数组变量赋值还有一个十分简便的方法,就是使用一对大括号将要赋予的值括起来,每个值都用“,”号隔开,被隔开的值赋予数组中的对应位置的成员,例如上面讲的给有 2 个成员的数组赋值,每个成员都赋值 100,就可以使用下面的方法:

变量 = {100,100}

使用这种方法给数组成员很多的数组赋值,尤为方便,如给一个有 10 个成员的整数数组,就可以输入:

变量 = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}

(3)命令赋值

可以使用“连续赋值 ()”命令给数组赋值,“连续赋值 ()”命令可将指定的常数、常数集、常量、资源、对象或者变量赋予到一系列变量或变量数组中去。参数<1>的名称为“用作赋予的值或资源”,参数<2>的名称为“被赋值的变量或变量数组”,命令参数表中最后一个参数可以被重复添加。具体也可以参考即时帮助,例程如:

局部变量: 变量 1 数据类型: 整数型 数组: 3

连续赋值 (100, 变量 1 [1], 变量 1 [2], 变量 1 [3])

(4)直接用命令的返回值给数组赋值

有些命令的返回值就是一个数组,所以可以直接使用该返回值给数组赋值。赋值的时候首先要注意,根据命令返回数组的数据类型来给数组定义数据类型;还要注意数组的成员数是可变的,并且命令返回的数组成员数也不固定,所以可以定义接收返回值的数组成员数为 0,当该数组接收了命令的返回值后,会自动定义成员数。例如:“分割文本 ()”命令的返回值,就是一个文本型的数组,在程序中可以表示为:

变量名	类型	静态	数组	备注
文本数组	文本型		0	

文本数组 = 分割文本 (取运行目录 0, “\”,)

代码运行后,如果运行目录是“d:\Program Files\”,那分割后返回后子文本就是“d:”、“Program Files”、“”,即文本数组就有了 3 个成员,每个成员的值就是返回来的子文本。

(5)命令参数是数组

有些命令或组件方法的参数是数组,此类命令或方法就需要根据帮助信息,来了解参数中的数组所代表的意义。例如画板的画多边形方法,第一个参数中的添加的数组顺序记录多边形各顶点的横向及纵向坐标值,坐标是成对出现,所以,数组中每 2 个成员就代表了要画出多边形的 1 个点,要让画板画出一个三角形就可以用以下代码:

变量名	类型	静态	数组	备注
三角形坐标数组	整数型		6	

三角形坐标数组 [1] = 50

三角形坐标数组 [2] = 10

三角形坐标数组 [3] = 0

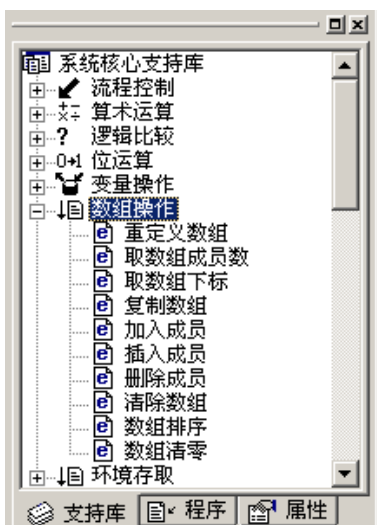
三角形坐标数组 [4] = 100

三角形坐标数组 [5] = 100

三角形坐标数组 [6] = 100

画板1.画多边形 (三角形坐标数组,)

6.2 动态管理数组变量



数组变量的最大特点就是它的成员数也是可变的，这样就有非常大的灵活性，在程序运行中，可以动态的为数组添加成员、删除成员等等，这样就达到了对数组的动态管理。

动态管理数组主要通过命令来实现，易语言提供了 10 个专门针对数组操作的命令，如图所示。

命令解释参见下表所示。

数组操作命令

命令名	命令解释
重定义数组	可以重新定义指定数组的维数及各维的上限值。
取数组成员数	取指定数组变量的全部成员数目，如果该变量不为数组，返回-1，
取数组下标	返回指定数组维可用的最大下标（最小下标固定为 1）
复制数组	将数组数据复制到指定的数组变量
加入成员	将数据加入到指定数组变量的尾部，自动增加其成员数目
插入成员	将数据插入到指定数组变量的指定位置，自动增加其成员数目
删除成员	删除指定数组变量中的成员，自动减少其成员数目
清除数组	删除指定数组变量中的所有成员，释放这些成员所占用的存储空间
数组排序	对指定数值数组变量内的所有数组成员进行快速排序
数组清零	将指定数值数组变量内的所有成员值全部设置为零

下面就结合实例来了解如何动态的管理数组。首先新建一个易程序，在窗口中添加 2 个编辑框组件和 1 个列表框组件，再添加 5 个按钮组件。

添加列表框组件是因为对数组的操作是不可见的，是在程序内部进行操作，这里将对数组的操作过程用列表框显示出来；2 个编辑框一个用来填写向数组中添加的内容，另一个用来填写“插入成员”和“删除成员”的成员位置，2 个编辑框的“输入方式”属性都设置成“整数文本输入”。5 个按钮组件的标题用 5 个常用的组操作命令名来命名。

首先双击“加入成员”按钮，切换到程序编辑界面，然后新建一个程序集变量，变量名为“被操作数组”，数据类型为“整数型”，然后定义为 0 个成员的数组变量。如下图所示。

窗口程序集名	备 注		
窗口程序集1			
变量名	类 型	数 组	备 注
被操作数组	整数型	0	

定义一个数组

1. “计次循环首 ()” 和 “取数组成员数 ()” 命令

“计次循环首 ()” 命令经常和 “取数组成员数 ()” 命令一起使用，用一个计次循环，就可以轻松的调用数组中的每一个成员。方法如下：首先用 “取数组成员数 ()” 命令将数组中的成员数取出来，然后用此成员数限定 “计次循环首 ()” 命令循环次数，每次循环，“循环次数变量” 就会递增 1，所以就可以用：

数组名 [循环次数变量]

用这行代码就可以依次访问数组中的每个成员。在本例程中首先要考虑：要用一个列表框显示出数组中的每个成员，所以要将数组中的每个成员依次取出，并添加到列表框中，并且每次对数组操作后都要重新显示数组中的成员。新建一个子程序，这个子程序专门用来让列表框刷新显示数组内容的，这样可以节省很多代码。

在程序设计界面用 **Ctrl+N** 来新建一个子程序，将子程序名改为 “刷新列表框”，然后在子程序中输入代码：

子程序名	返回值类型	公开	备 注
刷新列表框			

变量名	类 型	静态	数 组	备 注
循环次数变量	整数型			

列表框1. 清空 ()

```
--▶ 计次循环首 (取数组成员数 (被操作数组), 循环次数变量)
    列表框1. 加入项目 (到文本 (被操作数组 [循环次数变量]), )
--▶ 计次循环尾 ()
```

程序中的计次循环，第一次循环，“循环次数变量” 是 1，就将 “被操作数组[1]” 加入了列表框；第二次循环，“循环次数变量” 是 2，就将 “被操作数组[2]” 加入了列表框，依此类推，就将数组中所有的成员加入了列表框。由于被操作数组成员是整数型，所以要转换成文本型后再加入列表框。

2. “加入成员 ()” 命令

“加入成员 ()” 命令很简单，第一个参数填写欲加入成员的数组名，第二个参数填写欲加入的成员值。例程中是将编辑框 1 中的内容加入数组，所以用鼠标双击 “加入成员” 按钮后产生的 “_按钮1_被单击” 子程序中输入代码：

子程序名	返回值类型	公开	备 注
_按钮1_被单击			

加入成员 (被操作数组, 到数值 (编辑框1. 内容))
刷新列表框 ()

刷新列表框 () 是用来调用刚才新建的 “刷新列表框” 子程序，子程序被调用后就会运行子程序中的代码。以后每次对数组操作后都将调用这个子程序，用来重新显示被操作后的数组成员情况。

3. “插入成员 ()” 命令

“插入成员（）”命令，其实就是可以在数组中的指定位置加入成员。命令的第二个参数填写欲插入成员的位置。例程中编辑框2用来填写插入成员的位置，双击“插入成员”按钮，在产生的子程序中输入代码：

子程序名	返回值类型	公开	备注
_按钮2_被单击			

插入成员 (被操作数组, 到数值 (编辑框2.内容), 到数值 (编辑框1.内容))
刷新列表框 ()

4. “删除成员（）”命令

“删除成员（）”命令可以将数组中指定位置的成员删除，命令的第二个参数，用来填写欲删除的成员位置。例程中用编辑框2来填写欲删除的成员位置，双击“删除成员”按钮，在产生的子程序中输入代码：

子程序名	返回值类型	公开	备注
_按钮3_被单击			

删除成员 (被操作数组, 到数值 (编辑框2.内容),)
刷新列表框 ()

5. “清除数组（）”和“数组清零（）”命令

“清除数组（）”命令用来删除指定数组中的所有成员；而“数组清零（）”命令是将数值型数组的所有成员都设置为0。这里要注意，“数组清零（）”命令只能对数值型数组进行操作。例程中只使用到了“清除数组（）”命令，双击“清除数组”按钮，在产生的子程序中输入代码：

子程序名	返回值类型	公开	备注
_按钮4_被单击			

清除数组 (被操作数组)
刷新列表框 ()

6. “数组排序（）”命令

数组排序命令也只能对数值型数组进行操作，可以将一个数值型数组进行从大到小或从小到大的重新排列。双击例程中的“数组排序”按钮，在产生的子程序中输入代码：

子程序名	返回值类型	公开	备注
_按钮5_被单击			

数组排序 (被操作数组,)
刷新列表框 ()

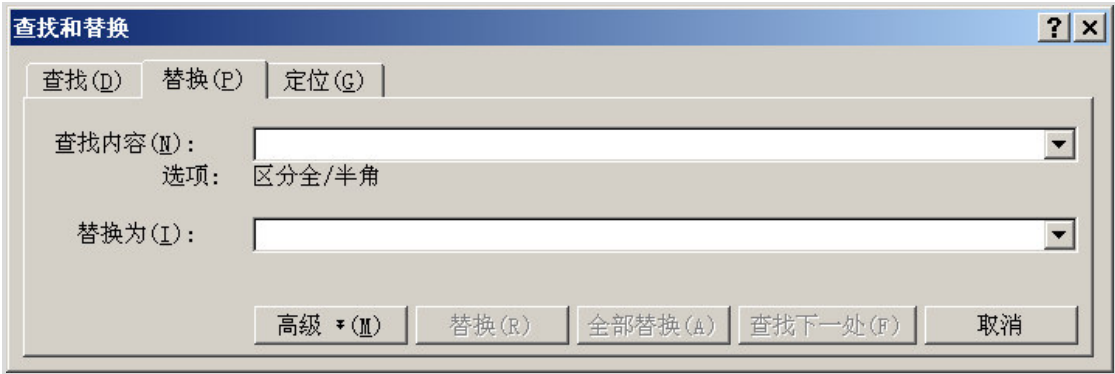
还有“重定义数组（）”、“复制数组（）”、“取数组下标（）”命令，都是对数组操作非常方便的命令，这里就不详细介绍了。

6.3 文本与数组操作实例

6.3.1 查找和替换文本

无论程序的大小，几乎都有查找和替换文本的工具。例如在 Microsoft Office Word 中，

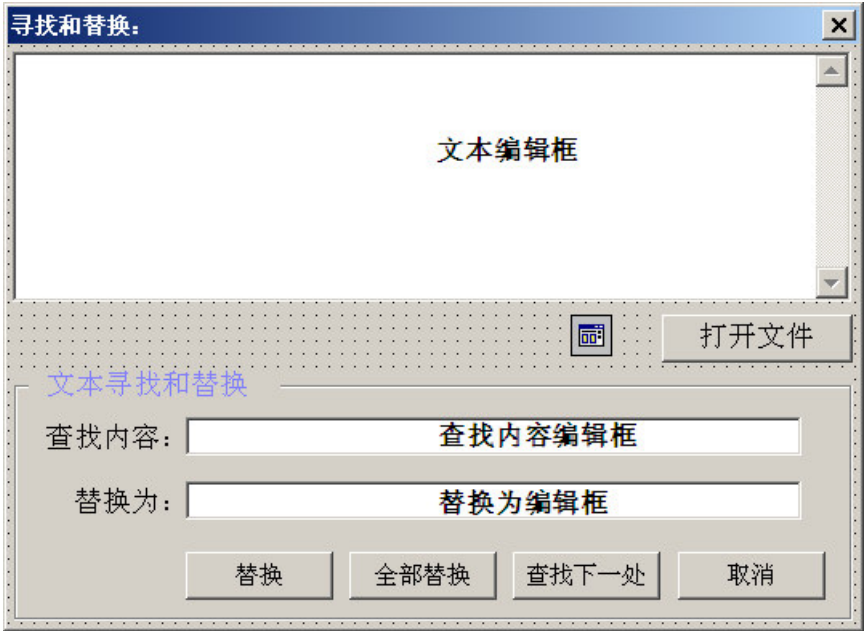
使用[Ctrl + F]组合键，可以弹出此功能的对话界面。下图所示：



其中“替换”子项中的功能更具代表性，所以下面编写的示例主要以此为范例，加以简单的解释。

首先，介绍下图中常用功能的使用方法。查找内容：在“查找内容”框内输入要查找的文字，单击“查找下一处”。替换文字：可自动替换文字，例如，将“你”替换为“您”。

新建“Windows 窗口程序”，程序界面设计如下图所示：



文本编辑框主要属性设置如下：

是否允许多行	真
滚动条	纵向滚动条
输入方式	只读方式

编写代码如下所示：

窗口程序集名	保 留	备 注	
寻找文本窗口程序集			
变量名	类 型	数 组	备 注
文件号	整数型		
文本位置	整数型		

第一步，添加程序集变量，如上。

子程序名	返回值类型	公开	备注
_打开按钮_被单击			

通用对话框1.过滤器 = “文本文件 (*.txt) | *.txt”

--- 如果真 (通用对话框1.打开 () = 假)

返回 ()

文件号 = 打开文件 (通用对话框1.文件名, ,)

--- 如果真 (文件号 = 0)

返回 ()

文本编辑框.内容 = 读入文本 (文件号,)

移到文件首 (文件号)

文本位置 = 0

第二步, 打开对话框, 选择文本文件, 并将文件的读写位置移动到文件的首部。如上。

子程序名	返回值类型	公开	备注
_全部替换按钮_被单击			

文本编辑框.内容 = 子文本替换 (文本编辑框.内容, 查找内容编辑框.内容, 替换为编辑框.内容, , , 真)

第三步, 使用“子文本替换 ()”命令, 将“文本编辑框”中的指定文本用另一个文本替换掉, 并重新把替换后的文本放到“文本编辑框”中。

子程序名	返回值类型	公开	备注
_查找按钮_被单击			

文本位置 = 寻找文本 (文本编辑框.内容, 查找内容编辑框.内容, 文本位置, 真)

--- 如果真 (文本位置 \neq -1)

文本编辑框.获取焦点 ()

文本编辑框.起始选择位置 = 文本位置

文本编辑框.被选择字符数 = 取文本长度 (查找内容编辑框.内容)

文本位置 = 文本位置 + 取文本长度 (查找内容编辑框.内容)

第四步, 查找指定的文本, 并选中。

子程序名	返回值类型	公开	备注
_替换按钮_被单击			

文本编辑框.内容 = 文本替换 (文本编辑框.内容, 文本位置 - 取文本长度 (查找内容编辑框.内容), 取文本长度 (查找内容编辑框.内容), 替换为编辑框.内容)

_查找按钮_被单击 ()

第五步, 使用“文本替换 ()”命令, 替换查找到的当前一个文本, 然后把替换后的文本重新放到“文本编辑框”中, 最后调用“_查找按钮_被单击 ()”子程序查找下一个匹配的文本, 并选中。

子程序名	返回值类型	公开	备注
_取消按钮_被单击			

结束 ()

如果编辑顺利, 可按“F5 键”运行。

6.3.2 创建 CSV 文本文件

CSV 是逗号分隔值文件的缩写, 这是一种使用逗号来分隔各个元素的文本文件。例如, 假设你具有一个由用户的单位、姓名和职位组成的 CSV 文件, 那么这个文件的内容可能类似如下:

```
三川实业有限公司, 刘小姐, 销售代表
国皓, 黄雅玲, 市场经理
世邦, 黎先生, 采购员
```

先来看一段代码, 然后介绍一个较为实用的示例。以下示例可将文本“文,”、“武,”、“双,”、“全”写入名为“文本文件.csv”的文本文件中, 代码如下:

变量名	类 型	静态	数组	备 注
文件号	整数型			

```
文件号 = 打开文件 ( "D:\Documents and Settings\Administrator\桌面\文本文件.csv", #改写, )
--- 如果真 (文件号 = 0)
    返回 ()
--> 如果真 (写出文本 (文件号, "文,", "武,", "双,", "全") = 假)
    信息框 ("写出文本失败。", 0, )
```

不管在什么时候使用“打开文件 ()”你都需要使用适当的到参数, 这取决于你想要读取、写入或是追加一个文件。使用“打开文件 ()”命令来创建一个名为“文本文件.csv”的新文本文件。(注意: 如果没有指定路径, 这个文本文件会创建在源代码所在的相同的文件夹中。也可以指定一个完整路径 (例如 D:\Documents and Settings\Administrator\桌面\文本文件.csv)。

从文本“文,”开始写入 (注意: 创建 csv 文件时我们必须手动写入逗号), “写出文本 ()”命令会写入指定的第一个文本数据, 并将当前读写位置置于当前位置, 因此, 下一次此命令添加“武,”文本时, 恰好位于“文,”文本的右侧, 后面的添加依此类推。最后得到的文本文件的内容如下:

```
文, 武, 双, 全
```

注意逗号和下一个项目之间没有任何空格。一个项目结束时, 另一个项目马上开始。来看另一个示例。

上面的代码虽然可以写出 csv 文件, 但在实际操作中是不现实的, 这样就需要重新编写代码。首先创建“Windows 窗口程序”, 在“启动窗口”上添加三个“编辑框”组件和一个“按钮”组件; 在“_按钮_被单击”事件子程序中添加如下代码:

变量名	类 型	静态	数组	备 注
文本数据	文本型			
文件号	整数型			

```

↓ + 文件号 = 打开文件 ( "D:\Documents and Settings\Administrator\桌面\文本文件.csv", #改写, )
    ... 如果真 (文件号 = 0)
        返回 ()
    文本数据 = 检索 (单位编辑框.内容, 姓名编辑框.内容, 职位编辑框.内容)
    移到文件尾 (文件号)
    ... 如果真 (写文本行 (文件号, 文本数据) ≠ 真)
        信息框 ( "写出失败。", 0, )
    关闭文件 (文件号)

```

使用“写出文本 ()”命令将检索返回的数据写到指定的文件中。

使用“检索”检索信息，其子程序代码如下：

子程序名	返回值类型	公开	备 注		
检索	文本型				
参数名	类 型	参考	可空	数组	备 注
单位	文本型				
姓名	文本型				
职位	文本型				

变量名	类 型	静态	数组	备 注
返回值	文本型			

返回值 = 单位 + “,” + 姓名 + “,” + 职位
返回 (返回值)

如你所见，在子程序中的代码与前一个程序有了很大的差别，其中不再使用类似“文，”、“武，”、“双，”、“全”的固定编码值，而是使用类似于“单位”的参数和变量。同样可以写出相同格式的文本文件。

但要注意的是，当使用包含逗号的数据可会遇到，如一个用户的职位为“销售代表”，而另一用户的职位为“市场助理，销售经理”这样，文本文件应该类似如下：

万海，林小姐，销售代表

嘉元实业，刘小姐，市场助理，销售经理

其第一行包含三段信息：万海/林小姐/销售代表；由于职位中包含逗号，第二行中却有四个段信息：嘉元实业/刘小姐/市场助理/销售经理。

处理这种嵌入逗号的方法是使用一个双引号包含每段信息；当文件被其它程序（Microsoft Office Access、Microsoft Office Excel 以及 Windows 系统中的通讯簿）使用时，嵌入的逗号就会被忽略。文件类似如下所示，

"万海", "林小姐", "销售代表"

"嘉元实业", "刘小姐", "市场助理，销售经理"

为解决双引号的问题，必须在“检索”子程序中将双引号加进去，代码修改如下：

```

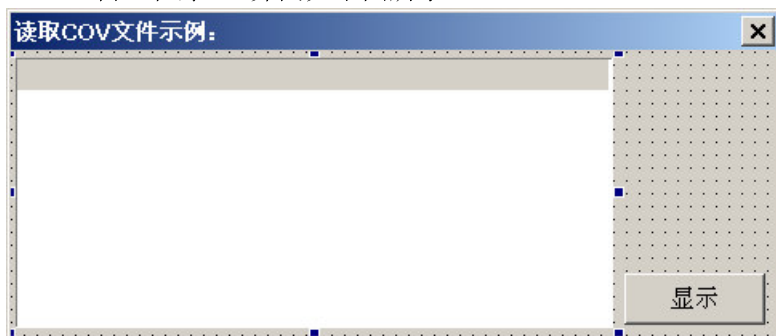
返回值 = #引号 + 单位 + #引号 + “,” + #引号 + 姓名 + #引号 + “,” + #引号 + 职位 + #引号
返回 (返回值)

```

6.3.3 读取 CSV 文本文件

前节创建的程序，可以将输入的客户信息直接转换到 cov 文件格式，并能使其默认程序正确的打开，虽然很方便，但毕竟不是自己编写的程序。那为了能在易语言中方便的读取和使用 cov 文件，下面就提供一此代码来实现这个功能。

创建“Windows 窗口程序”，界面如下图所示：



示例程序界面

“超级列表框”组件的主要属性设置如下：

类型	报表列表框
整行选择	真

双击“按钮”组件，在其事件子程序中添加代码如下：

变量名	类 型	静态	数组	备 注
读入的文本	文本型			
文件号	整数型			
单行数组	文本型		0	
循环变量	整数型			
去除长度	整数型			
列已存在	逻辑型			
表项索引	整数型			

以上是在编写程序中所声明的变量。

```

文件号 = 打开文件 ( "D:\Documents and Settings\Administrator\桌面\文本文件.csv", , )
如果真 (文件号 = 0)
    返回 ()
如果真 (移到文件首 (文件号) = 假)
    返回 ()
    
```

上面代码的功能是，打开上节中创建的 cov 文件。

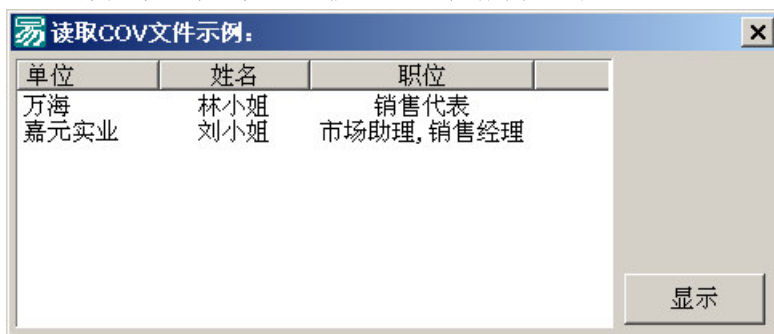
下面代码实现读取文本文件中的各行文本，并分析和去除各段数据的多余文本。其中还是使用了一些硬性代码，所以对其它方面的支持要差强一点，但这是为了更容易的理解这段代码。这里就不多做解释了。

```

--> 循环判断首 ()
读入的文本 = 读入一行 (文件号)
-- 如果真 (取文本右边 (读入的文本, 1) = “,”)
    去除长度 = 3
-- 如果真 (取文本右边 (读入的文本, 1) = #引号)
    去除长度 = 2
读入的文本 = 取文本中间 (读入的文本, 2, 取文本长度 (读入的文本) - 去除长度)
单行数组 = 分割文本 (读入的文本, #引号 + “,” + #引号, )
-- 如果真 (列已存在 = 假)
    超级列表框1. 插入列 (-1, “单位”, 140, #中间对齐, , )
    超级列表框1. 插入列 (-1, “姓名”, 60, #中间对齐, , )
    超级列表框1. 插入列 (-1, “职位”, 140, #中间对齐, , )
    列已存在 = 真
表项索引 = 超级列表框1. 插入表项 (-1, , , , , )
-- 计次循环首 (取数组成员数 (单行数组), 循环变量)
    超级列表框1. 置标题 (表项索引, 循环变量 - 1, 单行数组 [循环变量])
-- 计次循环尾 ()
清除数组 (单行数组)
-- 循环判断尾 (是否在文件尾 (文件号, ) = 假)

```

按“F5 键”运行程序，单击其上的按钮，显示结果如下图：



显示结果

6.4 习题

1. 写一个矩阵转换程序。例如将方阵

```

1  2  3
4  5  6
7  8  9

```

转换成一个新方阵

```

1  4  7
2  5  8
3  6  9

```

要求转换过程中不要借别的矩阵的协助。最后要把新方阵显示出来。

2. 写一个矩阵加法程序。并将新矩阵的元素显示出来。例如：

矩阵1 =

12	18	-50
-7	325	60

矩阵2 =

58	-30	6
18	72	-9

求矩阵 3=矩阵 1+ 矩阵 2。矩阵 3 的结果应是

矩阵3 =

70	-12	-44
11	397	51

3. 有若干个整数，要求分别按奇数和偶数由小到大的顺序显示出来，并显示出奇数之和与偶数之和。请编写此程序。具体整数个数及数据，可自行设定。

第七课. 程序调试

几乎每一个稍微复杂一点的程序都必须经过反复的调试、修改，才能最终完成。很显然，只有正确找出错误的地方才可以将其改正。出错以后怎样找出错误的地方就变得很重要了，下面就谈一些查错的方法。

7.1 调试工具

1. 易语言内部侦错

输入程序代码时有明显的语法错误存在，易语言会提示错误原因，而且当使用 **Ctrl+Enter** 键编译当前行时，错误代码无法通过编译。当用户写完一段代码后，试运行程序，易语言会对所有可能执行到的语句进行初步编译，当编译时发现明显的格式或语法错误时，易语言会将光标自动移动到错误行，并提示错误原因，用户可以根据光标提示和错误原因很快改正错误。如图 1 所示。

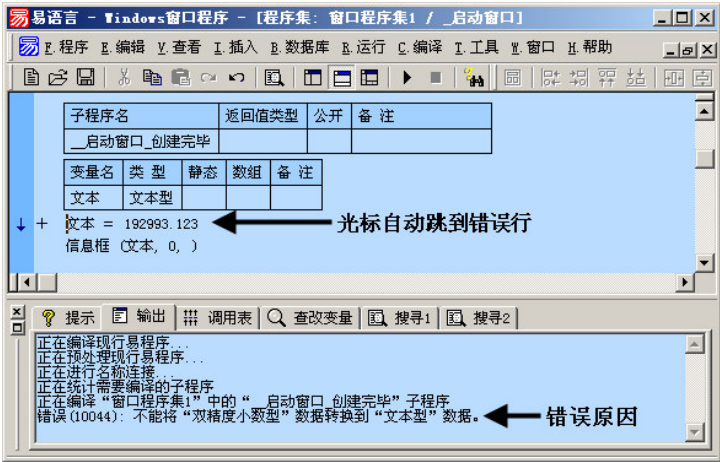


图 1 内部侦错

虽然易语言提供的内部侦错功能十分强大，但也只能指出输入上的格式错误，如数据类型不匹配、指定的变量或组件未找到等。而对于那些程序员编程时，逻辑上的错误，易语言就无能为力了。这就需要其他调试方法和调试命令的配合使用，快速而准确的找到错误代码。

2. 跟踪调试方法解释

易语言提供的跟踪调试方法，如图 2 所示。

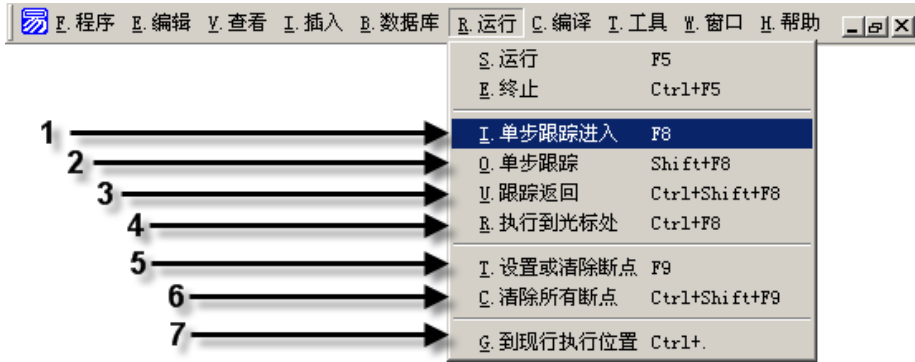


图 2 易语言调试方法

各调试项目在易语言状态条中的解释如下：

- ①在程序现行运行位置单步执行一程序，如果此程序行调用了子程序，则跟踪进入子程序。
- ②在程序现行运行位置单步执行一程序，如果此程序行调用了子程序，系统不会跟踪到该子程序中去。
- ③在上级子程序调用现行子程序的语句后中断。
- ④运行易程序，在当前光标处程序行处中断。
- ⑤设置或清除当前程序行处的断点。
- ⑥清除掉程序中的所有断点。
- ⑦跳到现行即将被执行语句的位置。

3. 跟踪法

编写一个比较简单的程序，看看程序是如何调试的。例程“调试程序.e”代码如下：

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

变量名	类型	静态	数组	备注
局部计次变量	整数型			

```
--> 变量循环首 (1, 300, 1, 局部计次变量)
-- 如果真 (局部计次变量 % 2 = 0)
--   -- 如果真 (局部计次变量 % 3 = 0)
--     -- 如果真 (局部计次变量 % 5 = 0)
--       编辑框1.内容 = 编辑框1.内容 + 到文本 (局部计次变量) + #换行符
--     --
--   --
--  --
-- 变量循环尾 0
```

该程序是输出 300 以内同时能被 2，3，5 整除的整数。 现在开始调试。调试有多种方

法，先介绍一种不需要设置断点就可以调试程序的方法。

首先选中要被调试的程序代码行。然后选择“执行到光标处”子菜单项目。如图3所示。

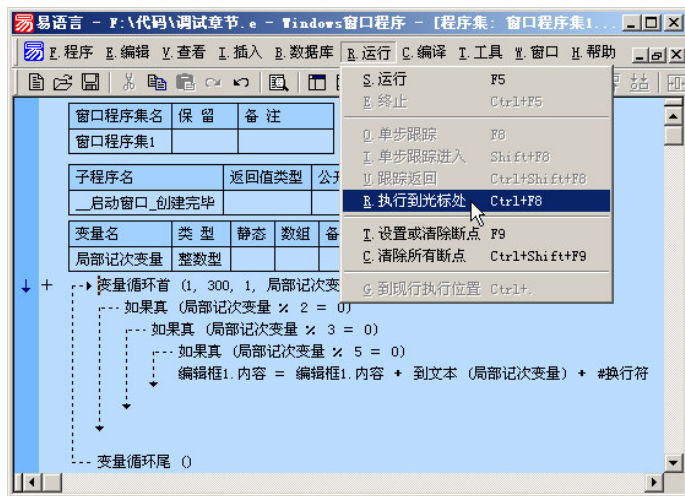
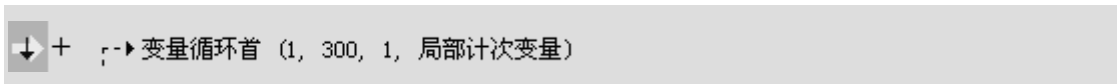


图3 程序跟踪

按“F5”键运行程序，当程序执行到被选中代码时，系统便将程序挂起，并用黄色箭头指示代码执行位置。



通常用“F8”键就可以实现把程序每一步执行的情况都反映出来的功能。此方法就是模拟计算机一步步执行程序的过程。

当前程序已经被运行，不断按“F8”就可以使程序一步一步执行，直到最后一行代码被执行完毕。

为了更直观的了解程序的运行过程，在跟踪程序的同时打开易语言的“查改变量”面板。选中“局部计次变量”如图4所示。

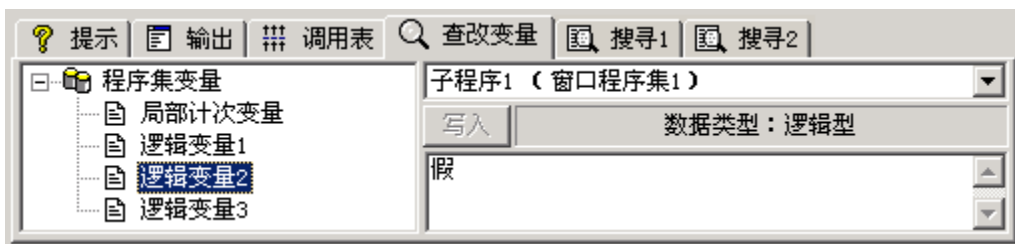


图4 查改变量面板

不断按“F8”的同时，“局部计次变量”的值也不断的改变。

下面介绍图4中标题为“写入”按钮的使用方法。

此按钮的作用是改写被选中变量的值，使被调试程序直接运行符合此变量被写入值的代码行处。

首先选中变量值，输入将被写入的值，按“写入”按钮。然后使程序设计界面获得焦点，再按“F8”键跟踪程序。

例程“调试程序 1.e”，使用改写逻辑变量的值可以更直观的看到程序的运行效果。

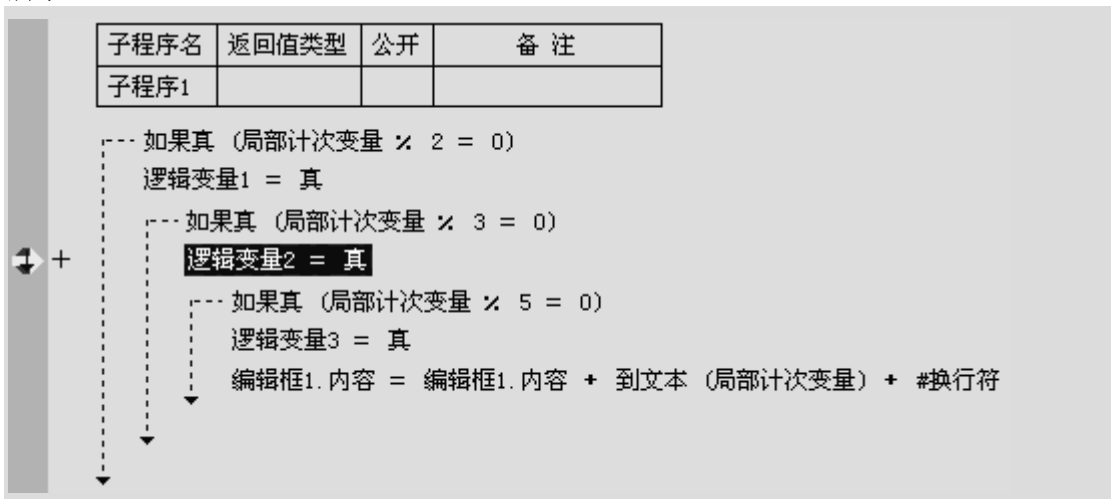
例程“调试程序 2.e”，按照前面的步骤，在“_启动窗口_创建完毕”事件子程序和“子程序 1”中选中代码行进行跟踪，比较各个跟踪方法。

4. 断点法

在前面已经学习了基本的程序调试方法。但也有一个缺点，就是在遇到循环次数比较多或者语句比较多时，用起来比较费时，下面用一种新的也是常用的调试方法：断点法。

所谓断点法，就是在程序执行到某一行时，调试器自动“中断”程序运行，并保留这时各变量的状态，方便检查、校对。还用“程序调试 2.e”为例，具体操作如下：

选中被调试程序代码，按下“F9”键，这时发现，该行首部加入红点，这表明该行已经被设置成断点行，当每次运行到此行的时候，程序都会自动停下来供编程人员调试。如下所示：



请记住，计算机是执行到断点行之前的一行，断点行并没有执行，所以这时“逻辑变量 2 = 真”这一句并没有执行，其值还是为假。如图 5 所示。



图 5 断点代码行未被执行

“调用表”面板显示当前被执行代码所在程序集。如图 6 所示。

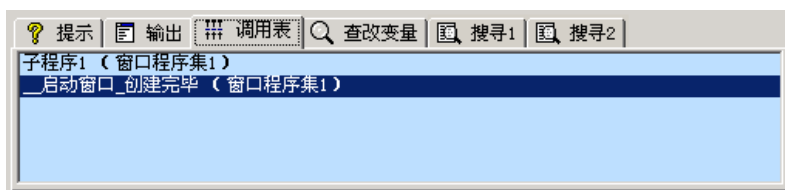


图 6 调用表面板

断点除了有以上用处之外，还有另外一个重要用处：它方便大家判断某个语句有没有执行或者是不是在正确的时刻执行，因为有时程序由于人为的疏忽，可能在循环或者递归时出现无法预料的混乱，这时候通过断点法，就能够判断程序是不是依照预期的顺序执行。

7.2 调试输出命令

1. “输出调试文本（）”

仅在易程序的调试版本中被执行，在发布版本中将被直接跳过；使用本命令可以在易语言调试系统的“输出”面板中输出指定的文本行以帮助调试，该文本之前被自动加上一个星号（*），之后被自动加上回车换行符。

如果觉得“改写变量”面板只能显示当前被调试程序变量的一个值，无法与前面或后面程序运行时变量的值进行对比，十分的不方便。就可以用“输出调试文本（）”命令实现这个功能。在“__启动窗口_创建完毕”事件子程序中添加“输出调试文本（）”命令如下：

子程序名	返回值类型	公开	备注
__启动窗口_创建完毕			

```

--> 变量循环首 (1, 300, 1, 局部计次变量)
    逻辑变量1 = 假
    逻辑变量2 = 假
    逻辑变量3 = 假
    子程序1 ()
    输出调试文本 (到文本 (逻辑变量1) + “ ” + 到文本 (逻辑变量2) +
        “ ” + 到文本 (逻辑变量3))
-- 变量循环尾 ()
    
```

注意提供的参数数据类型必须为文本型。

按“F5”键运行程序，在“输出”面板中显示，如图7所示。

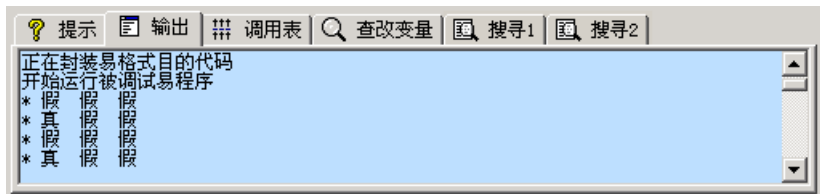


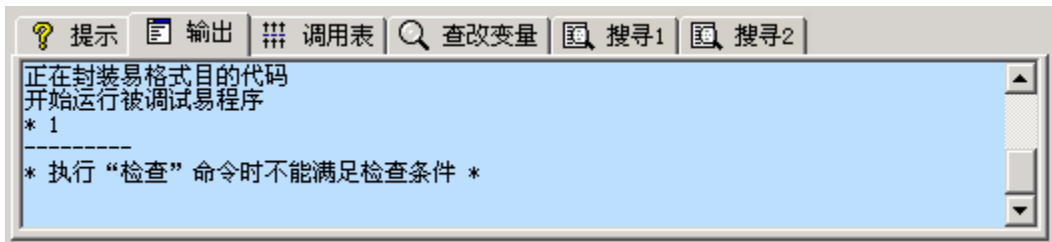
图7 输出面板显示调试文本

如果调试时使用到多个“输出调试文本（）”命令，就必须提供被显示变量值的变量名或变量值的解释文本。在“子程序”中的最后一个如果真语句中添加“输出调试文本（）”命令如下：

```

-- 如果真 (局部计次变量 % 5 = 0)
    逻辑变量3 = 真
    输出调试文本 (“逻辑变量3: ” + 到文本 (逻辑变量3) + “ ” +
        “局部计次变量: ” + 到文本 (局部计次变量))
    编辑框1.内容 = 编辑框1.内容 + 到文本 (局部计次变量) + #换行符
    
```

显示结果如图8所示。



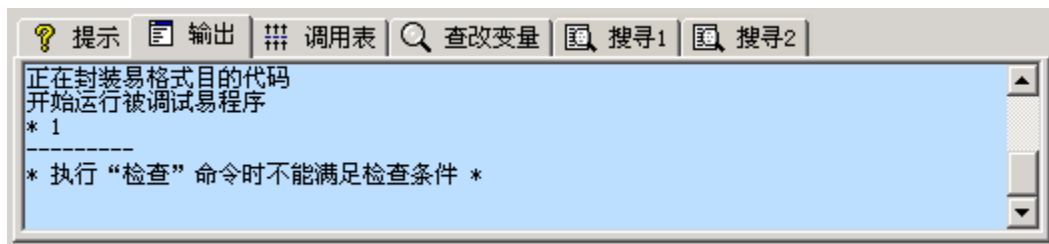


图 8 标记调试文本

可以很清楚的看到，只有“局部计次变量”的值为“30”时程序才会执行到最后一个如果真语句中，并使“逻辑变量 3”的值为真。

2. “暂停 ()”

仅在易程序的调试版本中被执行，在发布版本中将被直接跳过；可以在子程序中的任何地方放置此命令，使用此命令，就相当于在程序代码中设置断点。

3. “检查 ()”

仅在易程序的调试版本中被执行，在发布版本中将被直接跳过；执行本命令时，如果给定参数的条件值被计算后结果为假，易程序的执行将被暂停且警示。可以在子程序中的任何地方放置此命令，使用此命令，就相当于在程序代码中设置条件断点。

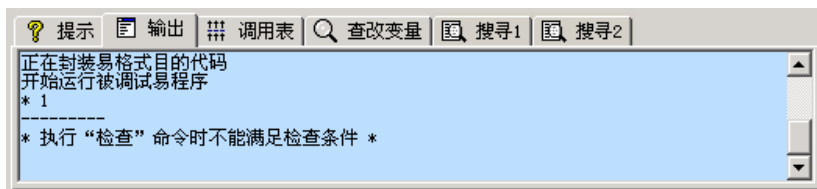
为方便了解“检查 ()”命令的作用，用“输出调试文本 ()”命令配合使用。首先将前面添加的两个“输出调试文本 ()”命令置为草稿。新添加调试命令代码如下：

子程序名	返回值类型	公开	备注
_启动窗口_创建完毕			

```

--> 变量循环首 (1, 300, 1, 局部计次变量)
    逻辑变量1 = 假
    逻辑变量2 = 假
    逻辑变量3 = 假
    子程序1 ()
    检查 (逻辑变量1 = 假)
    输出调试文本 (到文本 (局部计次变量) + #换行符 + "-----")
-- 变量循环尾 ()
    
```

按“F5”键运行程序，当“检查 ()”命令中的条件不成立时，程序被暂停。提示如下图所示。



“检查 ()”命令被执行

分析提示文本，

当“局部计次变量”为“1”时，程序无法进入“子程序 1”中的第一个如果真条件语句中执行，所以“逻辑变量 1”的值保持不变，为假，“检查 ()”命令没有被执行。而“输出调试文本 ()”命令正常输出文本（“局部计次变量”的值和“-----”）。当“局部计次变

量”为“2”时，符合“子程序1”中第一个如果真条件语句，其中的代码被执行，“逻辑变量1”的值为真，“检查（）”命令的条件不成立，“检查（）”命令被执行并输出提示。

4. “是否为调试版（）”

如果当前 EXE 易程序执行文件为易语言编辑环境调试运行程序时编译出来的调试版本，返回真。否则表明为发布版本，返回假。

5. “信息框（）”

在对话框中显示信息，等待用户单击按钮，并返回一个整数告诉用户单击哪一个按钮。

“信息框（）”是软件中必不可少的组成部分，是用户与软件之间的一种通信通道。但在程序的调试过程中也被经常使用到，而且可以达到多种调试方法的功能。如：断点、暂停（）、输出调试文本（）等方法和命令。将程序中的调试命令全部置为草稿，添加“信息框（）”，“子程序1”代码如下：

```

--- 如果真 (局部计次变量 % 2 = 0)
    逻辑变量1 = 真
    --- 如果真 (局部计次变量 % 3 = 0)
        逻辑变量2 = 真
        --- 如果真 (局部计次变量 % 5 = 0)
            逻辑变量3 = 真
            信息框 (局部计次变量, 0, )
            编辑框1.内容 = 编辑框1.内容 + 到文本 (局部计次变量) + #换行符

```

按“F5”运行程序，信息框提示“局部计次变量”为“30”，信息框被第一次执行，程序被暂停。如下图所示。



信息框调试

使用“信息框（）”显示数据不需要考虑是文本、数值、逻辑值或日期时间。如果同时显示不同数据类型变量和数据值，必须先转换到文本型。

注意：“信息框（）”不适合在“时钟周期”事件中和循环次数比较多的代码中使用。

7.2 调试总结

程序中出现的错误通常分为“语法错误”和“逻辑错误”。

所谓“语法错误”是指程序代码不符合易语言语法。这种错误最容易发现和修改：首先在代码输入的时候，系统就会检查并发现一部分语法错误；其次在程序运行的时候，系统执行到有语法错误的代码行，也会发现并指出错误原因。由此可见，系统会帮助找出所有的语法错误，只要按照其提示信息进行相应的修改即可。

所谓“逻辑错误”是指程序流程上、处理上的错误。含有“逻辑错误”的程序能够正常执行，只是执行结果不正确。这类错误系统是不可能发现的，只有靠编程者自己去寻找。

实际应用中，通常是将“断点”“单步跟踪”“查改变量”（以及调试输出）等调试方式结合起来使用。

一般的程序调试步骤是这样的：

1. 运行程序，执行所有的程序功能，从而找出并修改所有“语法错误”；
2. 通过分析判断，找到可能有“逻辑错误”的代码段；
3. 在“有逻辑错误的代码段”前面设置“断点”；
4. 运行程序，待程序在“断点”处中断后，使用“单步跟踪[Shift+F8]”“单步跟踪进入[F8]”“跟踪返回[Ctrl+Shift+F8]”“执行到光标处[Ctrl+F8]”等调试命令跟踪程序的运行。跟踪过程中，随时观察各变量值的变化（通过状态夹中的“查改变量”，可看到所有全局和局部变量），必要时使用调试输出命令将变量值输出。通过跟踪，一般能发现程序出错的原因。
5. 终止程序运行，修改代码，继续调试。

第八课. 常用算法

概括地说，“算法”是指解题方案的准确而完整的描述。

对于一个问题，如果可以通过一个计算机程序，在有限的存储空间内进行有限长的时间而得到正确的结果，则称该问题是算法可解的。但算法不等于程序，也不等于计算方法。程序可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。通常，程序设计的质量不可能优于算法的设计。

通常求解一个问题可能会有多种算法可供选择，选择的主要标准是算法的正确性和可靠性，简单性和易理解性。其次是算法所需要的存储空间少和执行更快等。

算法设计是一件非常困难的工作，经常采用的算法设计技术主要有列举法、递推法、贪算法、回溯法、分治法、动态规划法等等。另外，为了更简洁的形式设计和藐视算法，在算法设计时又常常采用递归技术，用递归描述算法。

8.1 列举(穷举、枚举)法

列举法的基本思想是根据提出的问题，列举所有可能情况，并用问题中提出的条件检验哪些是需要的，哪些是不需要的。因此，列举法常用于解决“是否存在”或“有多少种可能”等类型的问题。例如，求解不定方程的问题可以采用列举法。

列举法的特点是算法比较简单，但当列举的可能情况较多时，执行列举算法的工作量将会很大。因此，在用列举法设计算法时，应该重点注意使方案优化，尽量减少运算工作量。通常，只要对实际问题作详细的分析，将与问题有关的知识条理化、完备化、系统化，从中找出规律，或对所有可能的情况进行分类，引出一些有用的信息，列举量是可以减少的。

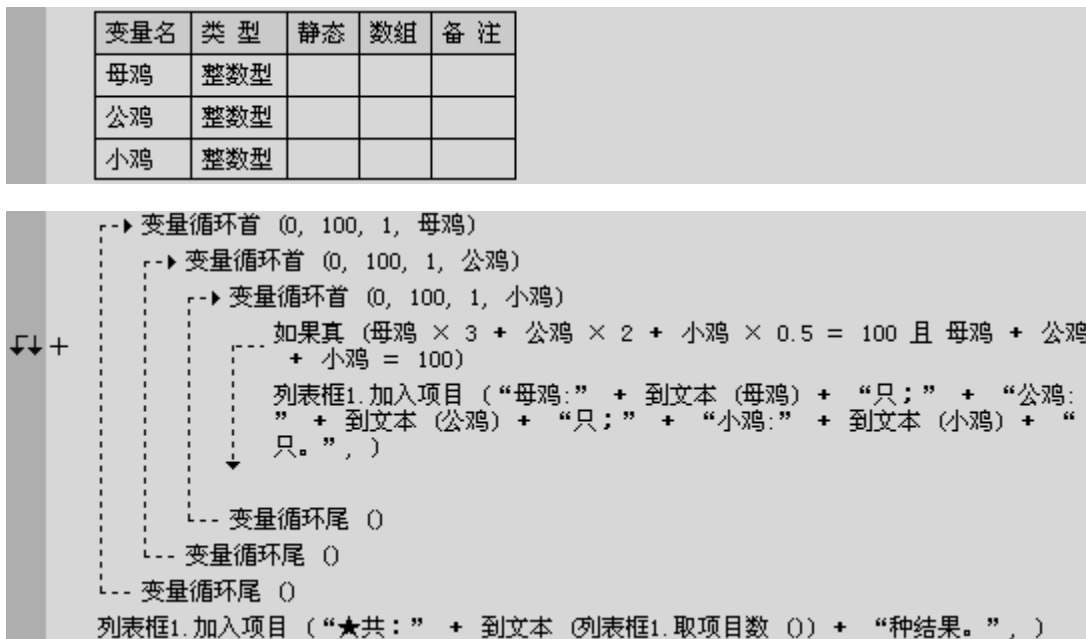
先介绍“列举法”在易语言中实现的方法。

列举法，顾名思义是列举出所有可能的情况，并用问题的条件来一一检验，并从中找出符合要求的解。特点比较简单，但是在写列举算法时尽量过滤掉一些不必要的情况，优化算法。下面举一个例子，在易语言中编写代码实现，来说明列举法的基本思想，以及如何减少列举量。

大家都知道百鸡的问题吧！母鸡每只 3 元，公鸡每只 2 元，小鸡每只 0.5 元，计算一下如何 100 块钱买 100 只鸡。

计算中，只有以下两个条件成立，才输出结果。

母鸡数量+公鸡数量+小鸡数量=100 只
 $3 \times \text{母鸡数量} + 2 \times \text{公鸡数量} + 0.5 \times \text{小鸡数量} = 100$ 元
 列举所有可能的条件的代码如下：



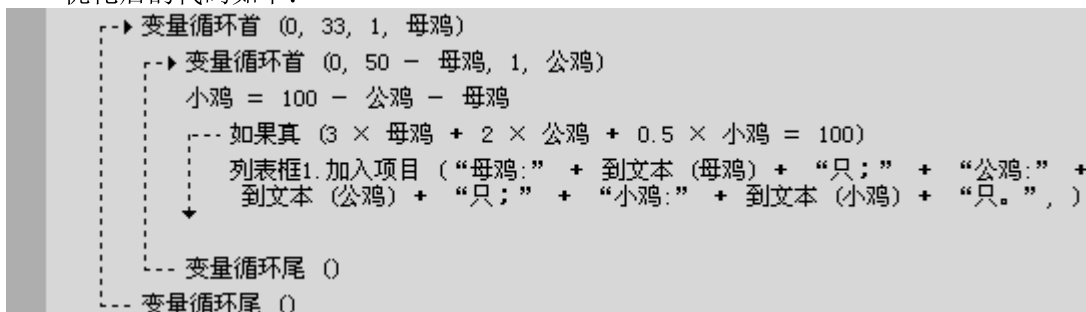
可以看到，各层循环均需循环 101 次，因此总循环次数为 101^3 ，列举量太大。但只对问题稍加分析，很容易发现这个算法还可以改进，减少不必要的循环次数。

首先，考虑到母鸡为 3 元一只，因此，最多只能买 33 只母鸡，即算法中的外循环只需要从 0 到 33 就可以了，没有必要从 0 到 100。

其次，考虑到公鸡为 2 元一只，因此，最多只能买 50 只公鸡。又考虑到对公鸡的列举是在算法的第二层循环中，因此，在第一层循环中已确定买母鸡数量的情况下，公鸡最多只能买 50-母鸡数量，即第二层中对“公鸡”只需要从 0 到 50-母鸡数量就可以了。

最后，考虑到总共买 100 只鸡，因此，在第一层循环中已确定买母鸡的数量，且第二层已确定买公鸡的情况下，买小鸡的数量只能是 100-母鸡数量-公鸡数量，即第三层的循环已没有必要了。并且，在这种情况下，条件：母鸡数量+公鸡数量+小鸡数量=100 只自然已经满足。

优化后的代码如下：



列举所有可能情形，最直观的是联系循环的算法。但，循环层数的改变，就影响了这一问题的解，即没有一般性。例如：有多种鸡，而每种鸡的价格又不同，每次根据一个固定的金额购买某几种鸡。

8.2 递推法

所谓递推,是指从已知的初始条件出发,逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定,或是通过对问题的分析与化简后确定。递推本质上属于归纳法。许多递推公式,实际上是通过通过对实际问题的分析与归纳而得到的,因此,递推是归纳的结果。

也就是利用问题本身所具有的一种递推关系求问题解的一种方法。设要求问题规模为 N 的解,当 $N=1$ 时,解或者已知,或者能非常方便地得到解。能采用递推法构造算法的问题有重要的递推性质,即当得到问题规模为 $i-1$ 的解后,由问题的递推性质,能从已求得的规模为 $1, 2, \dots, i-1$ 的一系列解,构造出问题规模为 I 的解。这样,程序可从 $i=0$ 或 $i=1$ 出发,重复地,由已知至 $i-1$ 规模的解,通过递推,获得规模为 i 的解,直至得到规模为 N 的解。

递推法中最具代表的就是阶乘法,下面采用分段乘法进行大数求阶乘。

在计算阶乘的时候,随着数字的增加,计算的结果会以极快的速度激增。当使用整数型数据储存结果时,当超过 12 时,计算结果将溢出;当使用长整数型数据储存结果时,当超过 20 时,计算结果将溢出。

为了能够计算较大的整数的阶乘并得到准确的结果,必须考虑适当的算法,分段乘法是其中的一种,将每一步的中间结果使用数组分段储存,数组的一个成员储存一段数据,由于我们经常使用的是 10 进制,因此以 10 的 n 次方为标准进行分段是可行的。同时需要考虑分组相乘时,每组是否会出现溢出,因此数组的类型采用长整数型。

这里采用该算法写了一个模块,循环和数组配合,虽然能计算出正确的结果,但速度较慢,计算 10000 的阶乘需要 50 秒(最快的在 1 秒左右,大家可以使用 Windows 的计算器试试),计算 50000 的阶乘,耗时 24 分钟,估计不会有人尝试计算更大的数值。

分段相乘完毕后还要考虑每组的数值是否超过分段的标准,如果超过要进位。代码参见例程。

8.3 递归算法

描述递归定义的函数或求解递归问题的过程称为递归算法。一个递归算法,本质上是将较复杂的处理归结为较简单的处理,直到最简单的处理。因此,递归的基础也是归纳。

下面学习易语言中递归算法的运用。

在易语言中许多地方可以用到递归算法,典型特征是自己调用自己。逐步到达条件边界,直到目的完成,如:“寻找文件 ()”等,递归算法结构比较简练,清晰易读,但,缺点是执行效率比较抵。

下面来练习寻找的的写法。

首先需要作一个单独的寻找子程序,如:名字取为“寻找”,参数为“寻找路径”文本型。

由于“寻找文件 ()”命令很特殊,好多初学者不好理解,因为寻找文件每次只返回一个匹配对象,所以一定要用循环才可以找到所有的匹配条件者,而第一次使用寻找到一个匹配条件者,再继续寻找其他文件就不需要指定寻找文件条件了。

代码可分为三部分,代码如下:

子程序名	返回值类型	公开	备 注		
寻找					
参数名	类 型	参 考	可 空	数 组	备 注
寻找路径	文本型				

变量名	类 型	静 态	数 组	备 注
文件名	文本型			

处理事件 0

```

--- 如果真 (取文本右边 (寻找路径, 1) ≠ "\")
    寻找路径 = 寻找路径 + "\"

```

第一部分，判断指定“寻找路径”参数右边是否带“\”，如果不带，就自动加上。

```

文件名 = 寻找文件 (寻找路径 + "*.tmp", )
--> 判断循环首 (文件名 ≠ "")
    列表框1.加入项目 (寻找路径 + 文件名, )
    标签1.标题 = 寻找路径 + 文件名
    文件名 = 寻找文件 ( )
-- 判断循环尾 0

```

第二部分是关键代码，首先调用

```
文件名 = 寻找文件 (寻找路径 + "*.tmp", )
```

因为只返回一个文本，所以用一个判断循环，判断“文件名”是否为空，如果不空 继续寻找，否则.....

这部分只寻找一个目录里所有匹配者，不会寻找里面文件夹里的匹配者，当然不能放过它们，下面的代码就是：

```

文件名 = 寻找文件 (寻找路径 + ".*", #子目录)
--> 判断循环首 (文件名 ≠ "")
    --- 如果真 (取文本左边 (文件名, 1) ≠ ".")
        寻找 (寻找路径 + 文件名)
        文件名 = 寻找文件 (, #子目录)
    -- 判断循环尾 0

```

第三部分是递归算法，等于子程序调用自己

```
寻找 (寻找路径 + 文件名)
```

就是这句，你也许会问

```
如果真 (取文本左边 (文件名, 1) ≠ ".")
```

是什么意思，其实就是判断文件名是不是文件夹，再 XP 命令提示符输入“dir”，一看就知道了。“.”是本级目录，“..”是上一级目录。

一定不要忘了在这个子程序里面加上

```
处理事件 0
```

否则运行后机器会假死。

下面是另一个运用递归算法求最大公约数的例程，核心部分只用了 6 行代码，代码如下：

子程序名	返回值类型	公开	备 注
辗转相除			

变量名	类 型	静态	数组	备 注
余数				

```

--- 如果 (较大数 % 较小数 = 0)
--- 编辑框3.内容 = 到文本 (较小数)
▶ 余数 = 较大数 % 较小数
  较大数 = 较小数
  较小数 = 余数
  辗转相除 ()
↓

```

注意：% 是求余数的运算符，不是除号，也不是百分号。

此例中求最大公约数，没有使用常用的短除算法，使用辗转相除法更便于编程。

8.4 查找运算

顺序表上的查找运算有很多种。如顺序查找法、二分法、分块查找法等。

我们先来介绍顺序查找法。顺序查找法，就是在表中顺序对比节点和欲查找数据的值，相同则返回相应的位置值。其过程，就是对表的顺序扫描。这是一种方法简单，但效率不高的查找方式。代码如下：

子程序名	返回值类型	公开	备 注
查找数据	整数型		

参数名	类 型	参考	可空	数组	备 注
欲操作数组	整数型	✓		✓	
欲查找数据	整数型				
查找位置	整数型	✓	✓		

变量名	类 型	静态	数组	备 注
计次变量	整数型			

```

---▶ 计次循环首 (取数组成员数 (欲操作数组), 计次变量)
--- 如果真 (欲操作数组 [计次变量] = 欲查找数据)
    查找位置 = 计次变量
    返回 (查找位置)
↓
--- 计次循环尾 ()
返回 (0)

```

8.5 对分（二分）查找

对分查找又称折半查找，它是一种效率高的查找方法。

对分查找的基本思想是：

- 1.首先确定该区的中点位置；
- 2.然后将待查的值与中间值进行比较，若相等，则查找成功并返回此位置，否则须确定新的查找区间，继续二分查找，具体方法如下：

（1）若中间值大于待查的值，则由表的有序性可知，中间值右表中的值均大于待查值，因此若表中存在等于待查值的值，则该值必定是在中间值左边的子表中，故新的查找区是左子表。

(2) 类似地，若中间值小于待查值，则要查找的值必在中间值的右子表中，即新的查找区间是右子表。下一次查找是针对新的查找区间进行的。

因此，从初始的查找区间开始，每经过一次与当前查找区间的中点位置上的节点的比较，就确定是否成功，不成功则当前的查找区间就缩小一半。这一过程重复直至找到中间值的值，或者直至当前的查找区间为空（即查找失败）时为止。

数据越多越能体现对分查找法的好处。

其实，每次插入都是比较大小安排插入位置，为了插到大于和小于数值的中间。如 4、6、8、50、55、59，如果给一个 56，当然知道插入到 55 后面，以前的代码是从 4 开始一一比较，而我们可以直接把数组对分，比一下是比前一部分大，还是比后一部分大，再在剩余结果对分比较，再对分……。逐一比较平面需要比较 5 次，对分只需要比较 2 次，数据越多，越能体现。代码如下：

变量名	类型	静态	数组	备注
排序变量	整数型		0	
当前查找次数	整数型			
数据成员数量	整数型			
插入位置	整数型			
当前数据	整数型			
有序成员数	整数型			
当前位置为奇数	整数型			
中间位置	整数型			
中间数据	整数型			

```

↓ + --> 计次循环首 (数据成员数量, 当前查找次数)
    当前位置 = 1
    当前分区成员数 = 取数组成员数 (排序变量)
    当前数据 = 到数值 (列表框1.取项目文本 (当前查找次数 - 1))
    -- 判断 (当前查找次数 = 1)
    -- 插入成员 (排序变量, 1, 当前数据)
    -- ' 加入第一个成员
    -- 到循环尾 ()
    -- 判断 (当前数据 ≥ 排序变量 [当前分区成员数])
    -- 插入成员 (排序变量, 当前分区成员数 + 1, 当前数据)
    -- ' 用无序数据比较数组的最大位
    -- 到循环尾 ()
    -- 判断 (当前数据 ≤ 排序变量 [1])
    -- ' 用无序数据比较数组的最小位
    -- 插入成员 (排序变量, 1, 当前数据)
    -- 到循环尾 ()
    -- 到循环尾 ()
    
```

```

--> 判断循环首 (当前位置 < 当前分区成员数 - 1) ' 条件成立, 继续拆, 直至找到
    "当前数据" 等于 "中间数据" 或者直至当前的查找区间为空 (即失败) 为止。
' 二分排序的结构
中间位置 = (当前位置 + 当前分区成员数) \ 2 ' 取有序区中间位置, 将分区
减半进行查找
中间数据 = 排序变量 [中间位置] ' 取有序区中间位置的数据
-- 如果真 (当前数据 = 中间数据)
    当前位置 = 中间位置
    跳出循环 ()
-- 如果 (当前数据 < 中间数据)
    当前分区成员数 = 中间位置
    ' 其插入位置在左分区
    当前位置 = 中间位置
    ' 其插入位置在右分区
-- 判断循环尾 ()
插入位置 = 当前位置 + 1
插入成员 (排序变量, 插入位置, 当前数据)
' 插入位置 = 1
-- 计次循环尾 ()

```

8.6 冒泡排序

冒泡排序是最简单也是最经典的。这种方法的基本思想是, 将待排序的元素看作是竖着排列的“气泡”, 较小的元素比较轻, 从而要往上浮 (相反, 较大的元素比较重, 从而要往下沉)。在冒泡排序算法中我们要对这个“气泡”序列处理若干遍。所谓一遍处理, 就是自底向上检查一遍这个序列, 并同时注意两个相邻的元素的顺序是否正确。如果发现两个相邻元素的顺序不对, 即“轻”的元素在下面, 就交换它们的位置。显然, 处理一遍之后, “最轻”的元素就浮到了最高位置 (或最重的元素沉到最低位置); 处理两遍之后, “次轻”的元素就浮到了次高位置。在第二遍处理时, 由于最高位置上的元素已是“最轻”元素, 所以不必检查。一般地, 第 i 遍处理时, 不需检查第 i 高位置以上的元素, 因为经过前面 $i-1$ 遍的处理, 它们已正确地排好序了。

关键在于每循环一遍都记录交换的次数, 避免下次不必要的比较, 还要记住最后交换的位置, 这个位置以后的数据下次可以在之前比较。

比如现在有一组数据, 第一次先比较所有的数据, 而后记下最后比较位, 而每次如果有数据交换, 就继续循环比较。而比较的最后位置则是循环的最大值。代码如下:

变量名	类型	静态	数组	备注
交换	逻辑型			
未比数据数量	整数型	✓		
交换变量	整数型			
位置变量	整数型			

↓ + 列表框2. 列表项目 = { }

交换 = 真 ' 如果不为真, 下面不会进入判断循环

未比数据数量 = 列表框1. 取项目数 () ' 也就是下面变量循环第一次的变量目标值

```

交换 = 真 ' 如果不为真, 下面不会进入判断循环
未比数据数量 = 列表框1.取项目数 () ' 也就是下面变量循环第一次的变量目标值
--> 判断循环首 (交换 = 真)
    交换 = 假
    --> 变量循环首 (1, 未比数据数量 - 1, 1, 位置变量) ' 比较的次数比数据数量少一
        如果真 (数据组 [位置变量] > 数据组 [位置变量 + 1]) ' 前一个数大于后
            一个数, 就将两数的位置交换
            交换变量 = 数据组 [位置变量] ' 数据组可以是变量、文件、数据库 (这里
                用的是变量)
            数据组 [位置变量] = 数据组 [位置变量 + 1]
            数据组 [位置变量 + 1] = 交换变量 ' 以上3行代码是交换用的
            交换 = 真
        --> 变量循环尾 ()
        未比数据数量 = 未比数据数量 - 1
    --> 判断循环尾 ()

```

8.7 直接插值排序

直接插入排序的基本思想是：每次把一个待排序的节点，按其关键字的大小插入到前面已排序好节点中，直到全部节点插入完毕。

两个重要概念。有序区：是指顺序表中已排序好的部分。无序区：即表中未进行排序的区域。例如，一维数组[2, 5, 6, 2, 9, 5, 47] 有序区的范围是：【1, 3】(从1到3节点)，无序区是：【4, 7】(从4到7节点)。对于任何一个节点为 n ($n > 1$) 的顺序表,其初始有序区的范围是【1, 1】无序区的范围是【2, n 】。插入排序的基本思想就是不断增大有序区，减小无序。直到无序区消失。

直接插入排序的原理：每次将无序区的第一个节点插入到有序区中。

例如，无序数组[8, 5, 6, 4, 3, 9, 4]，其初始有序区【1, 1】，无序区【2, 7】。第一次插入运算，将无序区第一个节点“5”插到前面的有序区中。运算后数组为[5,8,6,4,3,9,4]，有序区为【1, 2】无序区为【3, 7】。第二次插入运算，将无序区的第一个节点“6”插入到前面的有序区中，运算后的数组为[5,6,8,4,3,9,4]，其有序区为【1, 3】，无序区减小成【4, 7】。如此重复，经过6次插入运算，有序区成为【1, 7】，无序区消失。此数组成为有序数组。代码如下：

```

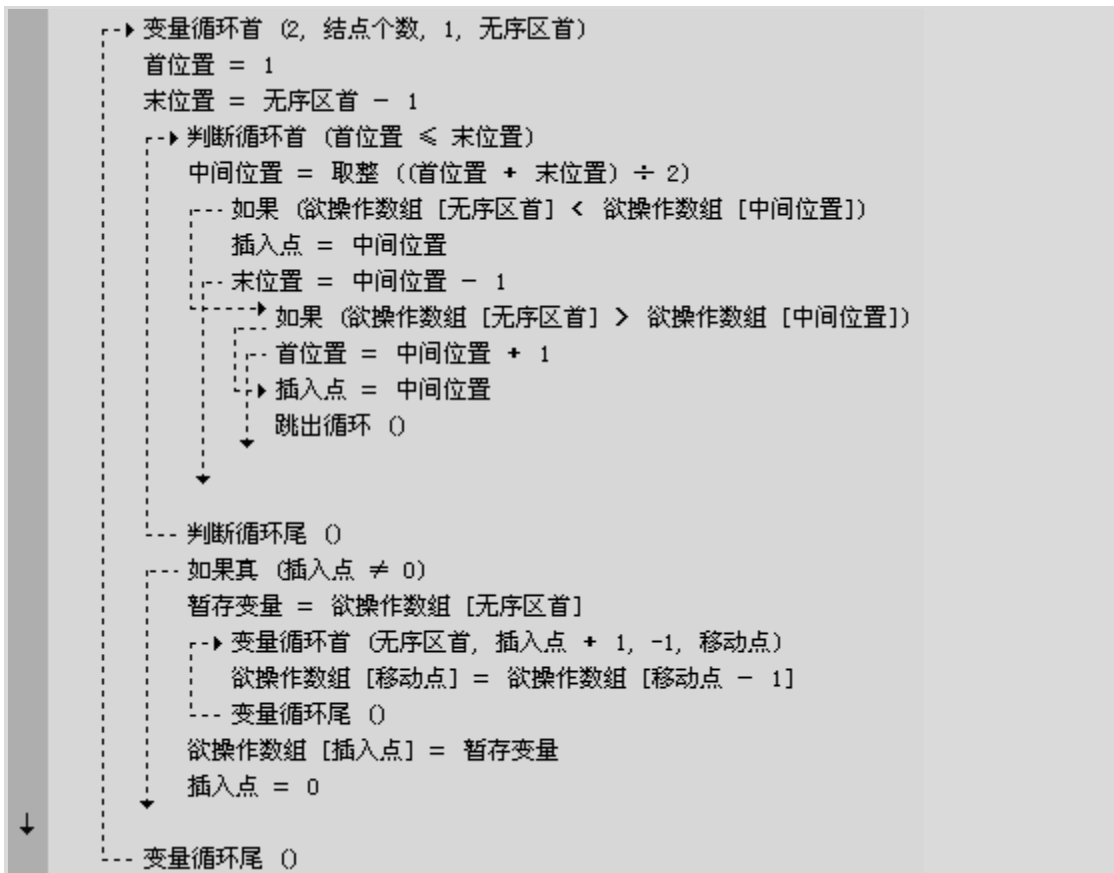
--> 变量循环首 (2, 结点个数, 1, 无序区首) ' 需要进行n-1次插入运算。
    --> 计次循环首 (无序区首 - 1, 插入点) ' 在有序区中寻找插入位置。
        如果真 (欲操作数组 [无序区首] <= 欲操作数组 [插入点]) ' 此为升序排序
            的插入条件。
            跳出循环 () ' 找到了插入位置。
        --> 计次循环尾 ()
        暂存变量 = 欲操作数组 [无序区首]
        --> 变量循环首 (无序区首, 插入点 + 1, -1, 移动点) ' 插入运算
            欲操作数组 [移动点] = 欲操作数组 [移动点 - 1]
        --> 变量循环尾 ()
        欲操作数组 [插入点] = 暂存变量
    --> 变量循环尾 ()

```

本段代码用到了查找算法和插入算法。

直接插入排序法的代码在执行过程中，大部分时间是消耗在查找插入点位置和移动节点

上。节点的移动是必须的，此类算法是无法减少移动次数的，除非是用新的算法。而每次都从无序区首（即表的第一个节点）开始寻找插入点的过程是完全没有必要的，可以用我们讲过的二分查找法优化此过程。以下是实现代码。



上面对直接插入排序进行了优化，我们暂且把优化算法称为二分直接插入排序，这种算法仅是对寻找插入点的过程做了优化。如果我们能找到另一种方法，可同时减少寻找插入点的次数和移动节点的次数，就可显著的提高排序的运算速度。

以下介绍的希尔排序算法是直接插入排序法的一种变种，它能有效的减少这两个方面的运算量，从而显著的提高排序操作的运算速度。

原理：先取一个小于节点数的整数 k 作为分组单位，把表上的所有相距 k 的节点逻辑上看成一组。在个组内进行直接插入排序；减小 k 的取值，把表上的所有相距 k 的节点逻辑上看成一组。在个组内进行直接插入排序；重复以上步骤；最后一次 k 取 1，即对整个表做直接插入排序。

其中 k 称为增量，增量的取值集合称为增量表。

例，无序数组[4,5,8,2,6,8,9,2,4,5,1,6,1,5,4,2] 增量表我们取成“5, 3, 1”(k 的取值集合)。

(1) 当 $k=5$ 时，表上的逻辑分组情况为：【1, 6, 11, 16】【2, 7, 12】【3, 8, 13】【4, 9, 14】【5, 10, 15】(【 】内均是数组的下标值)。 $k=5$ 时组内进行直接插入排序过程为：

第一组：数组成员分别为“4”，“8”，“1”，“2”。排序后的数组[1,5,8,2,6,2,9,2,4,5,4,6,1,5,4,8]

第二组：数组成员分别为“5”，“9”，“6”。排序后的数组[1,5,8,2,6,2,6,2,4,5,4,9,1,5,4,8]

第三组：数组成员分别为“8”，“2”，“1”。排序后的数组[1,5,1,2,6,2,6,2,4,5,4,9,8,5,4,8]

第四组：数组成员分别为“2”，“4”，“5”。排序后的数组[1,5,1,2,6,2,6,2,4,5,4,9,8,5,4,8]

第五组：数组成员分别为“6”，“5”，“4”。排序后的数组[1,5,1,2,4,2,6,2,4,5,4,9,8,5,6,8]

此次运算后数组为[1,5,1,2,4,2,6,2,4,5,4,9,8,5,6,8]

(2) 当 $k=3$ 时，表上的逻辑分组情况为：【1, 4, 7, 10, 13, 16】【2, 5, 8, 11, 14】【3, 6, 9, 12, 15】(【 】内均是数组的下标值)。 $k=3$ 时组内进行直接插入排序过程为：

第一组：数组成员分别为 “1”，“2”，“6”，“5”，“8”，“8”
排序后的数组[1,5,1,2,4,2,5,2,4,6,4,9,8,5,6,8]

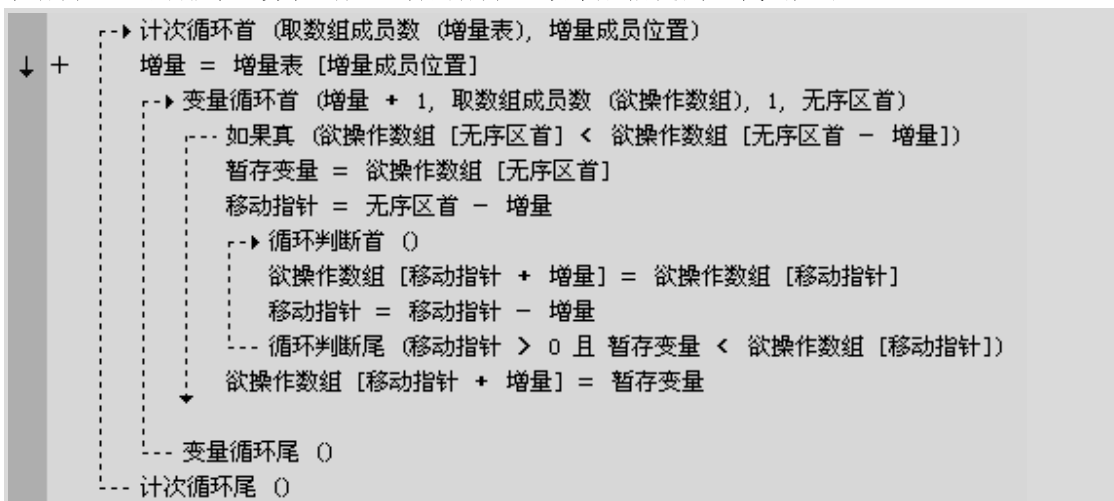
第二组：数组成员分别为 “5”，“4”，“2”，“4”，“6”
排序后的数组[1,2,1,2,4,2,5,4,4,6,5,9,8,6,6,8]

第三组：数组成员分别为 “1”，“2”，“4”，“9”，“6”
排序后的数组[1,2,1,2,6,2,6,2,4,5,4,6,8,5,9,8]

此次运算后数组为[1,2,1,2,6,2,6,2,4,5,4,6,8,5,9,8]

(3) 当 $k=1$ 时，表上的逻辑分组情况为：【1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16】(【 】内均是数组的下标值)。即整个表分为一个组，此时对整个表做直接插入表排序，可以发现此次运算时节点移动是非常少的。此次运算后数组为有序数组。

通过以上详细的运算过程描述，我们可以发现：当 k 值很大时，组很小，查找和移动量都很小，随着 k 值增大，组虽然增大，查找运算量增大了，但最耗时间的移动节点运算量却在减小。总的排序运算耗时是显著的减小，效率成倍提高。代码如下：



8.8 线表插值排序

在易语言中，一维数组上插入一个数据可以直接调用“插入成员(欲插入成员的数组变量，欲插入的位置，欲插入的成员数据)”系统命令。我们来研究它是怎么实现的。

原理很简单，当向一维数组的第 i 个位置插入一个数据时，第 i 个位置及它以后的数据依次都向后移动，将第 i 个位置腾出来后，把所要插入的数据写入该位置。

具体的实现步骤如下：

(1) 首先增加数组的长度。在易语言中可以用“加入成员 ()”和“重定义数组 ()”两种方法。前者可能更快些。

(2) 移动数据。不破坏数据，当然是从数组的最后位置向前移动了。可设一个位置变量“数据位置”指向最后位置。“数组”是定义的一维数组。“数组[数据位置]”，就是我们插入数组尾部的空数据。

数组[数据位置]=数组[数据位置-1]

把数组的倒数第二个数据移到尾部。然后

数据位置 = 数据位置 - 1

“数据位置”指向数组的倒数第二个数据,

数组[数据位置]=数据[数据位置-1]

把数组的倒数第三个数据移到数组的倒数第二个位置上,依次移动, 最后把数组的第 i 个位置上的数据移到 $i+1$ 的位置上。

(3) 把需要插入的数据插入到“数组[i]”。

代码如下:

子程序名	返回值类型	公开	备 注		
插入数据	逻辑型				
参数名	类 型	参 考	可 空	数 组	备 注
数组	整数型	✓		✓	
欲插入数据	整数型				
插入位置	整数型				

变量名	类 型	静 态	数 组	备 注
数据位置	整数型			

```

--- 如果真 (插入位置 < 1 或 插入位置 > 取数组成员数 (数组))
    返回 (假)
加入成员 (数组, 欲插入数据)
数据位置 = 取数组成员数 (数组)
--> 判断循环首 (数据位置 > 插入位置)
    数组 [数据位置] = 数组 [数据位置 - 1]
    数据位置 = 数据位置 - 1
--- 判断循环尾 ()
数组 [插入位置] = 欲插入数据
返回 (真)
  
```

8.9 数组插值排序

随机生成数字或文本。按从大到小或从小到大排序。

现在我们用数组排序。如果是数值可以用数组的“数组排序 (,)”。代码如下:

变量名	类 型	静 态	数 组	备 注
排序变量	整数型		0	
数据成员位置	整数型			
数据成员数量	整数型			

```

--> 重定义数组 (排序变量, 假, 0)
列表框2.列表项目 = { }
数据成员数量 = 列表框1.取项目数 ()
--> 计次循环首 (数据成员数量, 数据成员位置)
    加入成员 (排序变量, 到数值 (列表框1.取项目文本 (数据成员位置 - 1)))
--- 计次循环尾 ()
数组排序 (排序变量, 真) ' 这个子程序的核心代码
  
```

上面的代码比较简短,但是在某些时候,比如多项排序,就不好控制了,多位数组动态加入会自动变成单维的。下面是排序的标准写法,当然只是个框架,代码如下:

变量名	类 型	静态	数组	备 注
排序变量	整数型		0	
数据成员位置				
数据成员数量				
插入位置				
数据				

列表框2.列表项目 = { }

→ 重定义数组 (排序变量, 假, 0)

数据成员数量 = 列表框1.取项目数 ()

→ 计次循环首 (数据成员数量, 数据成员位置)

数据 = 到数值 (列表框1.取项目文本 (数据成员位置 - 1))

--- 如果 (数据成员位置 = 1)

--- 插入成员 (排序变量, 1, 数据)

→ 计次循环首 (取数组成员数 (排序变量), 数据成员位置)

--- 如果 (数据 ≥ 到数值 (排序变量 [数据成员位置]))

--- 插入位置 = 插入位置 + 1

→ 跳出循环 0

--- 计次循环尾 0

插入成员 (排序变量, 插入位置, 数据)

插入位置 = 1 ' 把位置改成首位

--- 计次循环尾 0

上面只是数值排序，大家可以练习一下文本、时间、逻辑。

8.10 删除运算

在易语言中，一维数组中数据（节点）在逻辑上是顺序相连的，并且它们在计算机中内存中也是顺序存储的，我们一般称顺序存储的线性表为顺序表。注意，顺序表是从存储方式上来描述表结构的；线性表则是从逻辑上描述表结构的。表结构从逻辑上可分为线性表和非线性表。线性表和非线性表都可以用两种方式存储，即顺序存储（如数组），和链式存储。表的链式存储方式需要指针。

在易语言中顺序表是建立在一维数组基础上的表结构。本节讲述顺序表上的删除运算：

删除运算也是通过移动节点来完成的。删除*i*节点时，我们只要把*i*后面的节点依次向前移动。在一维数组中把欲删除的位置后面的数据依次前移，把最后位置的数组成员删除，便完成了此运算。只有删除最后位置上的数据时不需要移动数据，直接删除该成员。代码如下：

子程序名	返回值类型	公开	备 注		
删除数据	逻辑型				
参数名	类 型	参考	可空	数组	备 注
欲操作数组	整数型	✓		✓	
删除位置	整数型				

变量名	类 型	静态	数组	备 注
前位置	整数型			


```

-- 如果真 (删除位置 < 1 或 删除位置 > 取数组成员数 (欲操作数
-- 组))
-- 返回 (假)
前位置 = 删除位置 ' 从删除位置处，将其后位置的数据向前移动
  变量循环首 (删除位置, 取数组成员数 (欲操作数组) - 1, 1, )
  -- ' 最后一个成员不需要向前移动，所以循环到的最大值是数组成员
  -- 数减1
  欲操作数组 [前位置] = 欲操作数组 [前位置 + 1] ' 把后一位
  置的成员移到前一成员位置处
  前位置 = 前位置 + 1 ' 记录将被移动成员的位置
-- 变量循环尾 ()
删除成员 (欲操作数组, 取数组成员数 (欲操作数组), 1) ' 删除最后
一位的成员
返回 (真)

```