第一章 Turbo PASCAL for Windows 简介

1.1 TPW 概貌

Borland 公司的 Turbo PASCAL 是为大家所熟悉的 PASCAL 语言开发软件，而 Turbo PASCAL for Windows（以下简称 TPW）则是 Borland 公司最近推出的新产品。TPW 是为开发 Microsoft Windows 的应用软件而设计的，并且采用了面向对象的程序设计技术。

MS Windows 由于其存储器管理能力、人机图形接口和不依赖于具体输出设备的输出方式而受到用户的欢迎。TPW 在 Windows 3.0 或更高的版本下运行。Windows 3.0 有三种模式：实模式(real mode)、标准模式(standard mode)和增强模式(enhanced mode)。实模式是模仿 DOS 的单任务工作环境，标准模式是基于 80386 CPU 的系统，能管理多达 16M 的内存；而增强模式则是基于 80386 CPU 的系统，并能建立虚拟内存(virtual memory)，其容量为机器中实际内存容量的四倍。标准模式和增强模式都是多任务的操作系统，TPW 要求在这两种模式下工作。

对于熟悉 Turbo PASCAL 的用户来说，使用 TPW 并不困难，因为 TPW 保留了传统 Turbo PASCAL 的数据结构、语法和大多数标准过程和函数。当然，TPW 也在原有的基础上增加了很多东西。下面我们将简单介绍 TPW 与传统的 Turbo PASCAL 不同的那些特点。

1.1.1 面向对象的程序设计技术

面向对象的程序设计技术(Object Oriented Programming，简称 OOP)是目前世界上流行的先进程序设计技术，TPW 和 Turbo PASCAL5.5 以上的版本都引入了这种技术。

TPW 定义了一种新的程序结构，称为对象(object)。对象可认为是完成某一特定任务所需的数据结构和访问这些数据结构所需算法的集合。我们可以将对象和传统 PASCAL 中的记录(record)相比较。记录是将各种各样的数据结构(如变量、数组、集合等)组织在一起，对象则不仅将这些数据组织在一起，还将对这些数据结构进行操作的函数和过程(统称为方法(method))也组织在一起。

TPW 定义的对象具有面向对象的程序设计技术的主要特点，即包裹性、继承性和多形性。

包裹性(Encapsulation)：是指一个对象将数据和对数据进行操作的方法统一在一起。一个对象中定义的数据，别的对象不能直接访问它。如果你在你定义的对象中提供了足够充分的方法，则当使用这个对象时，可以只传送信息给它，让它自己去完成某一特定任务，而不必去过问它内部定义了什么样的数据结构。也就是说，可以做到："告诉你做什么，怎么做是你的事，我不管。"

继承性(Inheritance)：一个对象可定义为另一个对象的子对象。在这种情况下，子对象继承了父对象所有的数据和方法，也就是说，子对象拥有父对象所有的数据和方法，如同于对象自己定义了这些数据和方法一样。继承性是可以延伸下去的，子对象又可以有自己的子对象。在 TPW 中，这种继承关系是一种树型结构，也就是说，一个对象可以有多个子对象，但每一个对象只能有一个父对象。

多形性(Polymorphism):对于 TPW 定义的对象,多形性的一种表现是子对象可替换(override)父对象的方法,即重新定义父对象中已定义的方法而仍然使用同样的名字。在面向对象的程序设计中,通常父对象提供完成某一任务所需的基本框架,包括定义基本的数据结构和定义基本方法,而父对象的每一子对象将完成更为具体的任务,子对象通常需要在父对象已提供的框架中加入更具体的内容。在这种情况下,子对象往往必须重定义父对象已定义的某些方法而不改变其命名,这是必要的,因为这样使得我们可以在程序中用同一个调用语句让父对象和各个子对象各自完成它们自己的任务,如同在程序中使用同一个变量而这个变量在不同情况下将有不同的值一样。多形性的另一种表现是父对象对子对象的兼容,程序中使用的对象类型的变量和参数,可被同类型的子对象替换。

TPW 的面向对象程序设计技术不象 SmallTalk 那样是"纯粹的"OOP,即:程序中所有的东西都是对象,通过传递信息给某个对象,然后由这个对象自行处理来完成程序中的某一任务,而不使用结构式程序设计的那种调用函数和过程的方法。TPW 的 OOP 象 C++那样是混合型的,即可以通过给对象传递信息来完成一个特定任务,也可以用调用函数和过程的方法来完成任务。因此,TPW 能与传统的结构式程序设计兼容。用传统的 Turbo PASCAL 开发出来的源程序,如果不使用部件 Crt 和 Graph 的话,一般只需做一些小改动即可在 TPW 中通过编译。

1.1.2 使用大容量的内存

传统的 Turbo PASCAL 为用户提供了 64K 的静态数据区,更多的数据则必须放入堆阵(heap)中。由于传统的 Turbo PASCAL 开发出来的软件是在 DOS 的环境下运行的,故受到 610K 内存容量的限制,堆阵能提供的存储区域决不会大于 640K。这使得那些需要使用大量数据的程序不得不将数据存在硬盘上,从而降低了运算速度。

在这一方面,TPW 就优越得多了。TPW 开发的是 MS Windows 的应用程序,在 MS Windows 3.0 或更高的版本上运行。MS Windows 3.0 的标准模式能管理多达 16M 的内存,而增强型模式则能建立虚拟内存,其容量为机器中实际内存容量的 4 倍。TPW 要求在标准模式或增强模式下工作,因此,用 TPW 开发出来的程序可使用大容量的内存。比如说,你的计算机安装了 2M 内存,工作于 Windows 的增强模式,则你将拥有多达 8M 的内存,Windows 系统本身占用了一部分内存,应用程序可使用 4M 左右的内存。

传统的 Turbo PASCAL 将内存分为四个区域,数据段、堆栈段、代码段和堆阵。数据段和堆栈段各自最多占内存为 64K,代码段没有限制,堆阵占据上述三个段之外的剩余内存空间。TPW 对内存的划分与传统的 Turbo PASCAL 不一样。TPW 将内存划分为数据段、堆栈段、代码段、内部堆阵(local heap)和公用堆阵(global heap)。其中数据段、堆栈段、和内部堆阵合在一起构成一个"自动数据段"(automatic data segment)。这个自动数据段不能超过 64K。这看起来似乎可使用的数据空间比传统的 Turbo PASCAL 还小,但事实上 TPW 使用的是面向对象的程序设计技术,大量的数据都在对象中定义,当程序运行时,对象可被装入到公用堆阵中,而公用堆阵使用当前的所有内存剩余空间,如前所述,这个空间是数以兆计的。

用传统的 Turbo PASCAL 开发出来的应用程序是在 DOS 的环境下运行的。当一个应用程序运行时,内存中不可能有别的应用程序同时也在运行。而用 TPW 开发出来的应用程序是在 Windows 这个多任务的操作系统下运行的,当一个应用程序运行时,别的应用程序也能同时运行。一个应用程序的自动数据段是只能被这个应用程序访问的内存区域,别的应用程序不能访问它。每个应用程序都有自己的自动数据段,而公用堆阵则是所有同时运行的应用程序公用的

内存区域。

1.1.3 与 MS Windows 的接口

每个操作系统都有其应用程序接口。DOS 的应用程序接口就是 DOS 中断，应用程序通过 DOS 中断来进行文件管理，键盘输入，屏幕显示，打印机输出，通信等输入输出操作。当我们用传统的 Turbo PASCAL 开发 DOS 的应用软件时，我们几乎感觉不到这个接口的存在。这是因为编译程序已自动为用户将这些操作编译为 DOS 中断调用了，因此用户可以对 DOS 中断一无所知而照样开发自己的应用软件。然而 MS Windows 就要复杂得多。开发 MS Windows 的应用软件，不可避免地要直接跟 Windows 的应用程序接口打交道。这个接口称为"应用程序设计接口"(Application Programming Interface，简称 API)。

API 由 600 多个函数组成，它们来自 Windows 的三个模块：kernel.exe、GDI.exe 和 User.exe。这些函数按其功能可划分为三个函数集：窗口管理函数集、GDI 函数集和系统服务函数集。窗口管理函数集提供与窗口操作有关的函数，如建立和管理窗口，窗口文字滚动，在应用程序和 Windows 之间传送窗口消息(message)等。GDI(Graphic Device Interface，图形设备接口)函数集则为用户提供文字图形显示的函数集。这个函数集最突出的优点是与输出设备的相对独立性。不管用户是在屏幕上、在打印机上还是在绘图仪上输出文字和图形，用户使用的都是同样的函数。GDI 函数集提供了画直线、画圆、画多边形、画点阵图(bitmap)及显示各种字模(font)的子程序。系统服务函数集则为用户提供了通信、文件输入输出、系统资源管理等子程序。

为了调用 Windows API 中的函数，TPW 为用户提供了两个部件(Unit)：WinTypes 和 WinProcs。前者包括了所有 Windows API 中定义的数据结构和常数，后者则包括了 Windows API 中所有的函数。事实上这两个部件是"PASCAL 化"的 Windows API。用户通过使用这两个部件，可直接调用 Windows API 中的任何一个函数。

MS Windows 有一些典型的图形人机接口，如对话框(dialog)，组合框(combo box)，按钮(button)等，大多数 Windows 的应用程序都使用这些标准的图形接口，以便形成一个友好的人机界面。当使用这些图形接口时，用户必须给出一些具体描述。比如使用一个对话框，用户必须描述对话框的尺寸，类型，内部包含的控制框(controls)等，然后还要设计对话框的信息处理过程。为了方便用户使用 Windows 的图形接口，TPW 从两个方面为用户提供帮助：

一方面是提供了一个 Windows 的应用软件 WRT(Whitewater Resowrce Toolkit)。WRT 让用户用作图的方式描述 Windows 的图形接口的图形特征，如尺寸，彼此相对的位置，提示文字等。然后根据用户的操作生成相应的资源描述文件。用户用 WPT 画出的图形接口是什么形状，将来用户程序运行时在屏幕上出现的图形接口也就是什么形状，相当直观，而且易于修改。这显然比用文字资源文件来描述图形接口要方便得多。用 WRT 生成的资源文件，可在编译时加入源程序，也可直接加入经过编译生成的.exe 文件中。

另一个方面是提供了一个窗口对象库(Object Windows Library，简称 OWL)。在这个库中提供了一些管理常用的图形接口的对象。例如用来建立和管理一个窗口的对象 TWindow，在这个对象中定义了描述窗口尺寸，在屏幕上的位置、窗口配设的菜单等的数据结构，还提供了处理常规窗口操作的算法，如登记窗口类，建立和撤消一个窗口，处理窗口消息和窗口刷新等。由于 OOP 的可继承性，用户只需定义自己的对象为 OWL 中某个对象的子对象，就可不费力地得到处理常规操作所需的数据和算法，从而能集中精力来设计对象所需要处理的特殊任务。

1.1.4 事件驱动式的程序设计(Event Driven Programming)

什么是事件？大多数情况下，事件是终端用户的一个操作。例如从键盘输入一个字符，用键盘或鼠标器选择菜单中的某一项，按下屏幕上的某个"按钮"(一种图形接口)，移动光标等。每当这种操作发生时，Windows 就会产生一个相应的消息。Windows 为各种各样的事件定义相应的消息，很多消息还带有参数，用来给予更具体的描述。例如消息 wm_keydown，表示终端用户在键盘上按下了一个键，其参数则给出了这个键的代码。又如消息 wm_LButtonDown，表示终端用户在键盘上按下了鼠标器的左边按钮，其参数则给出了当前鼠标光标的位置，等等。对于终端用户的每一个有效操作，Windows 都会产生相应的消息并送给应用程序。但消息并不仅仅来源于终端用户的操作，应用程序和系统本身也可以产生消息。

Windows 为每一个应用程序建立一个消息队列。当一个事件发生时，Windows 就生成相应的消息并将之加到这个队列中。应用程序负责从队列中筛选出有用的消息并对之作出响应。Windows 对于所有的消息有一个缺省的处理过程，对于应用程序中未予处理的消息，则调用这个过程进行处理。

除非是很简单的应用程序，大多数应用程序都要跟终端用户的操作打交道。用 TPW 设计程序时，我们不必设计程序段去判断终端用户是否有了一个操作和这个操作是什么，我们只需检查消息队列，看其中是否有我们期待的消息。若有，则调用事先设计好的响应过程。特别是，若使用 OWL，则还可以使工作更为简化。因为 OWL 已定义了从消息队列中筛选出有用消息的算法，如果用户对某个消息定义了相应的响应方法，则 OWL 将自动调用该方法，否则交与 Windows 的默认处理过程去处理。因此，在用户的应用程序中，只需对所期待的各种消息设计相应的响应方法即可。

所以，用 TPW 开发 Windows 的应用软件时，程序设计的主要线索是为期待的事件设计响应过程，然后等待事件发生。从这个意义上来说，应用程序本身不过是一个消息的"派发者"和"处理者"。

1.1.5 TPW 的标准部件(Standard Units)

TPW 提供了六个标准部件：System，WinDOS，WinCrt，Strings，WinTypes 和 WinProcs。

其中，WinTypes 和 WinProcs，如前所述，是"PASCAL 化"的 Windows API。

System 则类似于传统的 Turbo PASCAL 的部件 system。这是一个默认的部件，被自动连接到所有的应用程序上。TPW 的部件 system 基本上保留了传统的 Turbo PASCAL 的部件 system 中提供的所有过程和函数，只是去掉了一对与堆阵操作有关的过程：Mark 和 Relese。去掉它们的原因，显然是由于内存管理方式不同。传统的 Turbo PASCAL 应用软件是在 DOS 的单任务操作环境下运行的，一旦应用程序开始运行，所有的剩余内存空间都可由它支配。而 TPW 的应用程序是在 Windows 下运行的，剩余内存空间由 Windows 管理，并不全属于这个应用程序，而是和其它同时正在运行的应用程序共同使用。同样的原因，一些与堆阵操作有关的变量，如 HeapOrg，HeapPtr，FreePtr，FreeMin，在 TPW 的部件 system 中也不复存在。

在 TPW 的部件 system 中，还有两个问题要注意：一是所有在堆阵中申请使用内存和释放内存的过程和函数，如 New，GetMem，Dispose，FreeMem，其操作将是对于公用堆阵的，而不是对于局部堆阵的。也就是说，调用上述过程来存贮的数据，将被放在公用堆阵中。二是 TPW 取消了部件复盖(overlay)技术，因而取消了传统部件 system 中为部件复盖而定义的所有常数。

部件 WinDos 则类似于传统的 Turbo PASCAL 的部件 DOS，WinDos 提供的函数和过程基本上与 DOS 提供的一样，只是个别过程和类型常数的命名作了改动，以便符合 TPW 统一的命名规则。WinDos 还取消了一个使程序结束并驻留内存的过程 keep。

部件 WinCrt 则可以与 Turbo PASCAL 的部件 Crt 类比，但两者之间的差别比较大一些。Crt 中一些函数和过程是用来在屏幕上建立和管理窗口的，而 Windows 这个操作系统本身就具有这种功能，这一方面的操作可以而且应该通过调用 Windows API 中的函数来完成。因此 WinCrt 取消了所有与这类操作有关的过程和函数。事实上，用 TPW 开发出来的每个应用软件都有自己的主窗口，在这个主窗口上又可以开各式各样的子窗口。部件 WinCrt 的主要作用是提供一个类似于 DOS 环境的文字窗口。若一个应用软件使用部件 WinCrt，则这个软件运行起来时就象在 DOS 环境中的文字方式下运行一样，因而也就不能使用 Windows 的各种图形接口。因此，一个良好的 Window 应用软件不应该使用部件 WinCrt，WinCrt 只是在程序开发过程中为调试方便而使用。

部件 Strings 则引入一种字符串数据结构：空字符(null)结尾字符串，即一个字符串定义为一个字符数组，其长度可任意变化，最后用一个空字符来标志该字符串的结束。我们知道，传统的 Turbo PASCAL 字符串用第一个字节存放该串中实际有效的字符个数，在其后的字节中依次存放各个字符。这种字符串的长度必须事先定义，默认的长度为 255 个字节。这种字符串我们称为 PASCAL 字符串。遗憾的是，Windows API 不认识这种字符串，它只认识空字符结尾的字符串。为了便于应用程序调用 Windows API 中的函数，TPW 引入了这种字符串。部件 Strings 提供了一些过程和函数，用来对空字符结尾字符串进行操作，并在这种字符串和 PASCAL 字符串之间进行转换。

1.1.6 集成化的开发环境

仍然象传统的 Turbo PASCAL 一样，TPW 也提供了一个集成化的开发环境。所谓集成化，是指将源程序编辑、编译、运行和调试集中在一起。TPW 的集成化开发环境和传统的 Turbo PASCAL 一样，是一个菜单驱动式的应用软件，其菜单布局也彼此大同小异，但是 TPW 的这个开发软件是 MS Windows 的应用程序，在 MS Windows 的环境中运行。

TPW 的编译程序是用汇编语言写成的，其余部分则是用 C 语言写成的，由于目前发表的 TPW 是第一个版本，因此还没有一种适用于 Windows 的 PASCAL 语言可用来开发 TPW。

1.1.7 运行 TPW 的条件

如前所述，TPW 要求在 MS Windows 的标准模式或增强模式下工作。运行 TPW 的条件事实上也就是运行这两种模式所需要的条件：

(1) IBM PC，PS/2，或百分之百兼容的系统；

(2) 80286，80386，80386SX，80486 或等价的 CPU；

(3) 2M 以上内存；

(4) Microsoft Windows 3.0 或更高的版本；

(5) EGA，Hercules 或 VGA 显示适配器；

(6) 鼠标器或类似设备；

(7) 硬盘驱动器(Windows 3.0 和 TPW 都不能在软盘上运行)。

若将 TPW 的全部文件装入硬盘，大约占据 6.5M 的存储空间。

1.2 系统安装和启动

1.2.1 系统安装、启动和退出

在 TPW 的 1 号盘上有一个 Install 程序，执行这个程序即可自行安装整个系统。Install 必需在 Windows 下运行，用户可在 DOS 命令状态下从键盘键入 WIN A:INSTALL 来开始安装，也可以在启动 Windows 之后，选择菜单 File|Run，然后键入 A:INSTALL 并选择 OK.

若在安装过程中认可所有缺省的选择，则系统安装程序将在硬盘上建立一个子目录 TPW，在这个子目录下又分别建立 6 个子目录：OWL，OWLDEMOS，WINDEMOS，DOCDEMOS，DOC 和 UTILS.

TPW 本身是 Windows 的一个应用软件，必须在 Windows 的环境中运行。因此，启动 TPW 首先要启动 Windows，这在 DOS 命令状态下只要键入 Win 然后按回车键即可。然后，可以用两种方法启动 TPW：

(1)选择程序处理器(Program Manager)的主菜单上的菜单命令 File，然后选择子菜单命令 Run，这一命令将打开一个对话框，在对话框的命令行(Command Line)中输入 TPW 及其所在的路径，例如，若 tpw.exe 在 C 盘根目录下的子目录 TPW 中，则输入

C:\TPW\tpw

即可启动 TPW；

(2)最简单的方法是找到 TPW 的图标(Icon)，将鼠标器光标移到这个图标上，连续按两次鼠标器即可。或者，如果你使用键盘，则首先用光标移动键选择 TPW 的图标，然后按回车键即可。

象所有的 Windows 应用程序一样，TPW 拥有自己的主窗口和控制菜单(Control menu)。在 TPW 主窗口左上角有个小方块，称为控制菜单框，用鼠标器选择这个小方块中的按钮即可打开控制菜单。该菜单中各项意义简单解释如下：

Restore：使 TPW 的主窗口恢复为先前的状态。这一项仅当 TPW 的主窗口放大到占据整个屏幕或缩小为一个图标时有效。

Move：用来改变 TPW 的主窗口在屏幕上的位置。当主窗口占据整个屏幕时，这一项将会无效。

Size：用来改变 TPW 主窗口的大小。当主窗口占据整个屏幕或缩小为一个图标时，这一项无效。

Minimize：使 TPW 的主窗口缩小为一个图标。

Maxmize：使 TPW 的主窗口占据整个屏幕。

Close：退出 TPW，其对应的加速键命令为 Alt+F4。

Switch To……：这一项将打开 Windows 的任务表(task list)，这个表列出了所有当前正在运行的应用程序名，选择其中某个应用程序将使 TPW 暂停运行而转去执行该应用程序。

退出 TPW 还有另一个方法：选择 TPW 的主菜单上的 File 这项，再选择其子菜单命令 Exit 即可退出 TPW，其对应的加速键命令是 Alt+X。

1.2.2 交换文件

如果 TPW 在 Windows 3.0 或更高版本的增强模式下运行，则系统将使用部分磁盘空间来建立虚拟内存。为提高系统的运行速度，可在磁盘上划出固定的一块存储空间供系统建立虚拟内存。在这种情况下，这块存储空间将以文件的形式存在于硬盘上，称为交换文件(Swap file)。用户可以为这个文件指定位置(驱动器号及路径)和大小。用户可按下列步骤建立交换文件：

(1)退出 Windows，在 DOS 状态下键入命令 Win/r，即启动实模式的 Windows。在这种模式下，Windows 不使用交换文件，因而用户可以修改它。

(2)实模式的 Windows 启动后，关闭所有的程序，只留下 Program manager。

(3)然后，键入 Swapfile 或者 C:\Windows\system\swapfile，即执行文件 Swapfile.exe，这个文件通常在 Windows 的 system 子目录下，按照程序的提示，用户可以建立一个 Swapfile 并指定其位置和大小。程序运行完毕之后，重新启动标准模式或增强模式的 Windows 即可。

究竟 Swapfile 多大才合适？这个问题很难回答。下列几点仅供参考：

(1)Swapfile 必须使用物理上连续的扇区。如果用户硬盘可分区，最好将某个较小的 DOS 分区(比如说，驱动器 G:)，指定为 Swapfile 专用。

(2)打开 Program manager 的 Help 菜单，选择 About Program manager 这一项，它将会显示当前操作模式和可用内存空间。据有关资料介绍，Free system Resource 的百分比应在百分之五十左右或更高。如果这个百分比低于 25%或磁盘操作明显太过频繁，则应考虑加大 swapfile。

(3)系统运行效果还和高速缓存(cache)的大小有关。若高速缓存太小而 swapfile 又很大，则将会导致太频繁的磁盘读操作。用户应在两者之间平衡。

1.2.3 DOS 初始化文件 ConFig.sys 和 AutoExec.bat

在 DOS 初始化文件 ConFig.sys 中，应包括如下两行：

```
device=himem.sys
device=Smartdrv.sys 768 256
```

第一行用来装入扩充存储器(Extended memory)的驱动程序，从而用户能使用 64K 常规内存之外的大量存储空间；第二行用来建立磁盘高速缓冲区(cache)，行末第一个数字指明普通缓冲区(Normal cache)的大小，即当系统不运行 Windows 时 cache 的大小，其单位为千字节(KB)；行末最后一个数字是指明最小的 cache，因为当运行 Windows 时，Windows 将会自动减少 cache 以获得更多的内存供系统使用，此处指明了允许的下限，单位为千字节(KB)。

为运行 TPW，在用户的 AutoExec.bat 文件中还应建立路径：

```
Path c:\windows; c:\dos; c:\tpw; c:\tpw\utils
```

其中，c:\windows 指明 windows 的存取路径，c:\tpw 和 c:\tpw\utils 则指明 TPW 的存取路径。若用户使用不同的子目录名，则上述路径名也应相应地改变。

1.2.4 TPW 的初始化文件

每次启动 TPW，TPW 都会从一个名称为 tpw.ini 的文本文件中读取初始化参数，关闭 TPW 时将会改写这个文件。tpw.ini 必须在用户的 Windows 子目录下，在这个文件中，至少有如下三行：

[Startup]

Cfgpath=c:\tpw\tpw.cfg

SigeOrg=1,7,8,685,562

其中第一行[startup]说明下列各行是初始化参数;第二行告诉 TPW 到什么地方去找它的初始化文件,其名称通常为 tpw.cfg。若用户改变了文件名或其存取路径,则应做相应的改变;第三行记录了 TPW 上次关闭时的状态。若第一个数字为 1,则 TPW 启动之后将会立刻缩小为一个图标(Icon),若第一个数字为 0,则 TPW 打开后将恢复上一次关闭时窗口的位置和大小。

另外,用户可以在 TPW 的集成化环境(Integrated Environment)中调用 Borland 公司提供的调试程序 TDW(Turbo Debugger for Windows)。在这种情况下,用户可能需要在 tpw.ini 文件中加入下列三行:

[Debugger]

Exepath=c:\tpw

Switches=—ds

其中,第一行说明下列各行为 TDW 的初始化参数,这一行和 tpw.ini 文件中其余各行至少应隔开一个空行;第二行告诉 TPW 到什么地方去找调试程序 tdw.exe;第三行"Switches="的后面是启动 tdw.exe 时的命令行参数。关于命令行参数的详细内容,读者可参考 Borland 公司的手册《Turbo Debugger for Windows》。若用户并不打算使用命令行参数,则第三行可删除。

若用户在 TPW 的集成环境中调用调试程序而系统报告找不到调试程序,则应检查 tpw.ini 文件中有无上述各行及所设置的路径是否正确。此外,DOS 环境变量 Path 中若未包括 TPW 的路径,系统也有可能找不到调试程序。

此外,Windows 的应用程序通常会使用典型的图形接口,如菜单、对话框、图标等。作为 Windows 应用程序的开发工具,TPW 提供了一个设计这些图形接口的工具——WRT(Whitewater Resource Toolkit)。WRT 本身是 Windows 的一个应用程序,用户用 WRT 可以用作图的方式直观地设计菜单、对话框、图标等。使用 WRT 时,WRT 需要在磁盘上建立大小至少为 700K 的临时文件,根据有关手册建议,用户最好在硬盘上留有 3M 的空余空间。

1.3 关于 TPW 菜单命令的简要介绍

1. 文件菜单 File

new:开始编辑一个新文件。

open:打开一个文件供用户编辑。这一项将打开一个对话框,用户可从对话框的文件表中选择一个文件或在输入框中直接输入文件名。

save:将当前正在编辑的文件存盘。

save as:将当前正在编辑的文件用另一个文件名存盘。这一项将打开一个对话框,用户可在对话框的子目录表中选择需要的子目录,在输入框中输入文件名。

SaveAll:将所有被修改过的文件存盘。

Print:打印当前正在编辑的文件内容。

Printer Setyup:这一项将打开一个对话框供用户设置打印机。

Exit：退出 TPW。

Dos 的 Turbo PASCAL 每次只能打开一个文件进行编辑，若用户要编辑另一个文件则必须关闭当前的文件。TPW 可以同时打开多个文件进行编辑，每个被打开的文件将拥有自己的文件窗口(Document Window)。若用户要暂停某个文件的编辑而转去编辑另一个文件，则可将该文件窗口缩小为一个图标，然后或者用 open 菜单项打开一个新文件，或者从下拉式菜单 File 末端的文件表中选取另一个文件，或者选择另一个文件对应的图标(当该文件已被打开时)，恢复该文件窗口。

在 Windows 中，每个应用程序都将有自己的主窗口，称为应用程序窗口(Application Window)，某些应用程序窗口又可以拥有多个文件窗口。TPW 的主窗口就是一个应用程序窗口，而每打开一个文件进行编辑就相应地打开一个文件窗口。文件窗口象主窗口一样有控制菜单，因此也可以进行移动、改变大小、缩小、放大、关闭等操作。用户打开一个新文件之前可以不关闭原来正在编辑的文件，从而可以方便地在多个文件之间交叉进行编辑，甚至同一个文件也可以被打开多次，这时每打开一次就相应地打开一个新的文件窗口，而用不同的窗口标题来区分它们。

打印正在编辑的文件内容时不要用打印屏幕键(Print Screen)，因为在 Windows 下，该键用来将整个屏幕的内容拷贝到剪辑板(Clipboard)上。用户可以用如下方法观察剪辑板的当前内容：

(1) 进入 Windows 的程序管理器(Progran manager)；

(2) 启动窗口 Main 中的 Clipboard 图标。

拷贝到剪辑板上的内容事实上是一个图形拷贝，包括按 Print Screen 键时屏幕上的任何图形和文字。拷贝屏幕的另一种做法是同时按 Alt 和 Print Screen，这样做将把当前活动窗口的内容拷贝到剪接板上。

2. 编辑菜单 Edit

该菜单用于文字编辑，进行文字编辑时，经常需要选择一段文字进行复制、移动、删除等操作。用户首先必须弄清楚选择一段文字的方法。可以用鼠标器或键盘选择一段文字。具体的操作方法不在本书讨论范围之内，读者可参考 Windows 的用户指南或 TPW 的用户指南。但如果用户用键盘的话，要注意 TPW 定义了两种不同的命令集：CUA (Common User Access) 和 Alternate。CUA 同时也是 Windows 使用的命令集，而 Alternate 则是 Borland 公司的软件通用的命令集。下面是这两个命令集中部分命令的对比：

操作	CUA	Alternate
选择一个文字块	Shift＋光标键	Ctrl＋KB Ctrl＋KK
从磁盘读一个文字块	Shift＋Ctrl＋R	Ctrl＋KR
写一个文字块到磁盘	Shift＋Ctrl＋W	Ctrl＋KW
文字块缩排(Indent)	Shift＋Ctrl＋V	Ctrl＋KI
文字块反缩排	Shift＋Ctrl＋V	Ctrl＋KV

用户可在两种命令集中选择一种。选择的方法是：

(1)选择 TPW 的选择项菜单 Option；

(2)在 Option 的下拉式菜单中选择 Preferences 一项；

(3)在打开的对话框中的 Command Set 一项选择 CUA 或 Alternate。

编辑菜单 Edit 中各子菜单的意义如下：

Undo：取消上一步的操作。例如，用户刚刚删除一个字符，则 Undo 将恢复该字符；若用户刚刚输入一个字符，则 Undo 删除该字符。Undo 可以连续进行，用 Undo 取消上一步的操作之后，再用 Undo 则将取消上一步的再上一步操作。

Redo：取消最近的 Undo 操作，即恢复 Undo 操作之前的状态。

Cut：将一个文字块从文件中移去并存于剪辑板上。

Copy：将一个文字块复制到剪辑板上。

Paste：将剪辑板上的内容标入到当前光标所在的位置。

Clear：删除一个文字块。

在 TPW 中，文字块的移动、复制都是通过剪辑板进行的，而剪辑板是为整个 Wjndows 系统所公用的。因此，用户可以将文字块方便地从一个文件移动或复制到另一个文件。

3. 寻找菜单 Search

Find：寻找一个给定的字符串。这一项将打开一个对话框，用户可在其中的输入框内输入需寻找的字符串。

Replace：寻找一个给定的字符串，并且用另一个给定的字符串来代替它。这一项将打开一个对话框，用户可在其中的输入框内输入要寻找的字符串和用来替换的字符串。

Search Again：重复最近一次 Find 或 Replace 操作。利用这一项用户可连续寻找或替换一个字符串。它是这一菜单项对应的热键，若用户选择 CUA 命令集，则为 F3；若选择 Alternate 命令集，则为 Ctrl+L。

Go to line number：将光标移至指定的行。这一项将打开一个对话框供用户输入行数。在 TPW 的主窗口的最下面一行的左方，有两个数字，中间用冒号隔开，它们表示当前光标所在位置的行数和列数。

Show Last Compile Error：将光标移至最近一次编译时产生错误的地方，并显示错误代码和错误信息。

Find Error：寻找产生运行错误的位置。当程序运行产生运行错误时，TPW 的编译程序将会显示产生错误的地址。记下这个地址，然后选择这一菜单项，在打开的对话框中输入这个地址，于是编译程序将重新编译用户的源程序，并在源程序中产生运行错误的位置停下来。

4. 运行菜单 Run

Run：运行用户的程序。

Debugger：这一项将调出 TPW 专用的调试程序 TDW（Turbo Debugger for Windows）。TDW 也是一个配置了菜单的应用软件。启动 TDW 之后，用户通过选择相应的菜单项，即可进行设置断点、单步运行、检查变量内容等调试工作。调试完毕按 Alt-X 返回 TPW。

Parameters：若用户的应用程序带有命令行参数，则这一项用来输入命令行参数。

5. 编译菜单 Compile

Compile：编译当前正在编辑的文件。

Make：编译主程序并检查所有被主程序使用的部件（units），若某个部件在最后一次编译之后被修改过，则重新编译该部件。

Build：重新编译主程序和所有被主程序使用的部件，不论它们是否修改过。

Primary File：这一项将打开一个对话框供用户输入主程序的文件名。一旦用户指定了主程序，则可在编辑主程序所使用的任何一个部件后直接运行该程序，而不必调出主程序文件后再运行。

Clear Primary File：清除掉已指定的主程序文件名。

Information：显示最近一次编译后的有关信息，如源程序行数，生成的代码字节数等。

6. 选择项菜单 Option

Option 用来指定编译系统的设置，其中各项意义如下：

Compiler：这一项将打开一个对话框，用来指定与编译有关的各种参数。其中，选择框 Options、Align data、String var checking 和 Boolean evaluation 用来指定编译开关伪指令(switch directives)，Memory Sizes 用来指定堆栈和数据段内部堆阵的字节数。有关它们的意义请读者参考 TPW 的程序员指南(Programmer's guide)或打开 TPW 的帮助(Help)窗口(按 Alt+H)，选择 Compiler Directives 这一项，参考其中的说明。这些选择框的选择相当于在程序中用如下语句指定编译伪指令：

{$A+,B-,D+,F-,G-,I+,L+,N-,R+,S+,V+,W+,X+}

{$M 8192,8192}

但是程序中的指定具有更高优先权。也就是说，一旦程序中设置了某条开关伪指令，则无论 Option 选择框中对该条伪指令如何设置，编译程序都将按照程序中的指定来编译。用户可用 Ctrl+O(CUA 命令集)或 Ctrl+OO(Alternate 命令集)将选择框中的设定转化为上述的语句，加在程序的开头。这样做可以使程序"记住"编译时的条件，避免再次编译程序时，由于开关伪指令重新设置而使程序产生不同的运行效果。

Linker：设置程序连接时的一些选择项。这一菜单项将打开一个对话框，其中有三个选择框 Mapfile，Link buffer file 和 Debug info。Mapfile 用来指示编译程序产生一个关于连接时的内存地址信息的文件，其后缀为 map，文件名则与程序文件名相同。Mapfile 是一个多选一的选择框，若选择其中的选择项 off，则不产生.map 文件，选择另外的三项 Segments，Publics 和 Detailed 中的任何一项都将产生.map 文件。选择 Segments 将在.map 文件中记录程序起始地址，程序使用的段地址表和程序联接时的错误信息(如果有错误的话)。选择 Publics 则除了在.map 文件中记录上述信息外，还记录了所有公用符号(public symbols)及其地址。选择 Detailed 则在.map 文件中记录上述的所有信息以及源程序中各行的起始地址。.map 文件可用于汇编语言级的调试。若用户不打算深入到汇编语言级，则应选择 off，这也是系统的缺省选择。

选择框 Link buffer file 用来指定使用内存或磁盘作为联接缓冲区，缺省选择为内存。

选择框 Debug info 用来指示编译程序是否将调试信息加到用户的.exe 文件中去。若用户需要用 TDW 调试程序，则应设置这一项。

Directories：这一菜单项将打开一个对话框供用户指定编译时各种文件的存取路径，它们分别是：

ExE and TPU directry：指定访问.exe 和.tpu 文件的子目录；

Include directories：指定用户标准包含文件的存取路径；

Unit directories：部件文件的存取路径；

Object directories：目标文件(.obj 文件)的存取路径；

Resource directories：资源文件的存取路径。资源文件是一种描述 Windows 的图形接口的数据文件。

除了第一项 EXE and TPU Directry 之外，其它各项都可输入多条路径，各条路径之间用";"隔开。

Preferences：这一菜单命令将打开一个对话框，供用户按个人喜好选择编译系统的一些处

理方式。该对话框有 6 个选择框，其中选择框 Editor options 供用户选择一些文字处理方式，如文字是否自动缩排(indent)，存文件时是否自动建立备份文件等。选择框 Auto Save 有二个选择项，其中 Editor files 指示系统是否在运行程序之前保存源程序，Desktop 指示系统是否在退出 TPW 时保存当前窗口状态，以便下次进入 TPW 时即进入这种状态；Configuration 指示系统是否在退出 TPW 时保存用户在菜单 Options 中所做的所有设置。选择框 Font 用来选择系统字模，Tab size 用来指定一个 Tab 键对应几个字符位置，Right mouse button 指示系统是否将鼠标器右边按钮用于条目搜寻；Command set 则用来选择 CUA 或 Alternate 命令集。

7. 窗口菜单 Windows

这一菜单项用来安排系统文件窗口。其中：

Tile：使系统窗口成为迭瓦式窗口，即同时打开的多个文件窗口并排排列。

Cascade：使系统窗口成为重迭式窗口，即一个打开的文件窗口重迭在另一个打开的文件窗口上面。

Arrange Icons：每次用户将一个已打开的文件缩小为一个图标后，该图标将处在 TPW 主窗口的左下角，若有多个文件打开后缩小为图标，则这些图标将从左到右排列。用户可用鼠标器移动这些图标，菜单命令 Arrange icons 用来将这些图标均匀排列。

Close all：关闭所有已打开的文件。

在菜单 Windows 的下拉式菜单末尾有一个当前已打开的文件名表，选择其中一个文件将使其文件窗口成为活动的窗口(即光标所在的窗口)。

8. 菜单 Help

用来打开帮助窗口。TPW 的帮助窗口具有和 Windows 其它应用软件的帮助窗口一样的风格，用户可以方便地在各个帮助条目之间跳转。TPW 的帮助窗口几乎就是一本使用说明书，有关 TPW 各个方面的内容都可以从中找到相应的说明。同时，TPW 的帮助系统又是一个在线(on line)帮助系统，任何时候用户只要将光标移到 TPW 的某个关键词、常数、函数或过程名的位置上，然后按 Ctrl+F1，即可得到有关的帮助信息。

第二章　面向对象的程序设计方法

2.1　对象的定义

对于 TPW 来说，对象(object) 可认为是数据和对这些数据进行操作的函数和过程的集合体。下面是一个使用“对象”这种程序结构的一个简单程序(只是为了说明问题方便而已，从实用角度来说，这个程序完全是“小题大作”)。

```
Program OOPDemo1;                                    {line 1}

Uses WinCrt;                                         {line 3}

Type TMorning=object                                 {line 5}
     TextStr:string;
     Procedure Init;
     Procedure AssignStr;
     Procedure Display;
     End;

Procedure TMorning.Init;                             {line 12}
Begin
     AssignStr;
End;

Procedure TMorning.AssignStr;                        {line 17}
Begin
     TextStr:='Good morning!';
End;

Procedure TMorning.Display;                          {line 22}
Begin
     GotoXY(25,10);
     write(TextStr);
End;

Var Morning:TMorning;                                {line 28}

Begin                                                {line 30}
     Morning.Init;
```

```
    Morning.Display;
End.
```

第 1 行说明这是一个程序而不是一个部件(Unit)。

第 3 行说明本程序使用部件 WinCrt。如前所述，WinCrt 提供一个类似 DOS 文本方式的窗口。因为我们的程序只需要显示一行文字，因此选择这种最简单的窗口。

第 5 行开始到第 10 行是类型定义。我们定义了一个对象 TMorning，此处第一个字母 T (Type)表示这个名字是一种对象类型的名字，这是 TPW 的一种命名习惯。在 TPW 的对象库中，所有对象类型的名字第一个字母总是 T。遵从这种命名习惯有利于改善程序的可读性。当然，从语法上说，任何符合标识符命名规则的字符串都可作为一个对象的名称。对象 TMorning 包含了一个数据(字符串 TextStr)和三个对 TextStr 进行运算的过程 Init，AssignStr 和 Display。熟悉 Turbo PASCAL 的读者会发现，对象的定义和记录(Record)的定义很类似，只有两点不同，一是关键词 Record 被 Object 取代，二是记录中不能定义过程和函数，而对象中可以而且通常需要定义过程和函数。对象中的过程和函数统称为方法(method)。

在对象的定义中，并没有说明方法 Init，AssignStr 和 Display 的具体内容是什么。这些必须在其后的过程说明中给出。从第 12 行开始到 26 行是这三个方法的定义，语法上几乎与传统的 Turbo PASCAL 的过程说明完全一样，只是在每个过程名字的前面有 TMorning.，用以指出这个过程是哪个对象中定义的过程。

第 12 行到第 15 行说明过程 Init 的作用是调用另一个过程 AssignStr，而过程 AssignStr 的作用是将字符串 'Good Morning!' 赋给对象 TMorning 中的字符串 TextStr，而过程 Display 的作用则是在窗口的中间位置显示这个字符串。

接下来的 28 行定义了一个变量 Morning，其类型是 TMorning。就象 Integer 只是一种数据类型，你不能给它赋值或拿它来进行运算一样，TMorning 也只是一种对象类型，必须给它一个实体，才能对它进行运算。这种实体在 TPW 中称为"实例"(Instance)。此处变量 Morning 就是 TMorning 的一个实例。一个对象类型可以有多个实例。例如，你可以定义：

```
Var M1,M2,M3:TMorning;
```

于是 M1，M2，M3 成为 TMornimg 的三个实例，它们将各自拥有 TMorning 中定义的数据 TextStr。也就是说，在内存中将有三个变量 M1.TextStr，M2.TextStr 和 M3.TextStr。但并非 M1，M2，M3 都将有过程 Init，AssignStr 和 Display 的执行代码。事实上，程序运行时，内存中将只有一段这三个过程的执行代码，TMorning 的所有实例将共享这段代码。一般地说，每种对象类型可以有多个实例，所有的实例将各自拥有该对象类型所定义的所有数据，但它们将共享该对象类型所定义的所有过程、函数的执行代码。

从第 30 行开始才是程序的执行部分。第 31 行调用对象 Morning 的过程 Init，这个过程调用另一个过程 AssignStr，其结果是赋予字符串变量 TextStr 一个字符串"Good Morning!"。第 32 行调用对象 Morning 的另一个过程 Display，其作用是在窗口中间位置显示 TextStr 的内容。

这个程序虽然很简单，但已描述了定义和使用对象的基本方法。定义一个对象包括两部分：数据类型的说明和方法的说明。数据类型说明部分给出一个对象所使用的所有数据，这些数据可以是 PASCAL 中合法的任何一种数据结构，包括数组、集合、枚举类型等，还可以是另外的对象。方法说明部分则定义对这些数据进行运算的所有过程和函数。在方法说明中只给出这些过程和函数的名称，对于这些过程和函数具体内容的说明则和一般的过程、函数的说明

几乎完全一样，只是在这些过程或函数名前要给出其所属对象类型的名称。一个对象类型定义完毕之后，要给它一个实例，然后才能使用这个对象。

现在我们可以开始运行上面这个小程序。首先启动 TPW，输入上述程序，然后按 Ctrl＋F9，我们看到屏幕上开了一个窗口，在窗口中央有一行文字"Good Morning!"。事实上，当我们看到这个窗口时，程序已经运行完毕，但属于这个程序的窗口仍然留在那里，等待终端用户去关闭。你只要按一下 Alt＋F4 即可关闭这个窗口。在某些场合，也许你还需要留着这个窗口。比如说，你编了一个程序进行一系列复杂的计算，程序运行完毕后在窗口上留下了计算的结果，你可能需要留着它供后面的工作参考。这时你可以这样做：按 Alt 加空格键，这时在窗口左上角将会显示该窗口的控制菜单（或称为系统菜单；Control menu or System menu），选择 Minimize 这一项，按回车键，该窗口就缩小成了一个图标（Icon）。你可能看不见它，因为它可能跑到屏幕上当前可见部分的下方去了。没关系，下次当你需要看这个窗口的内容时，按 Ctrl－Esc 键，Windows 的当前任务表（Task list）就会出现在屏幕上。选择你的程序名（在它的前面会有 Inactive，说明该程序已运行完毕），按回车键，该窗口又重新出现。

上面这个小程序只是从一个侧面反映了 OOP 的包裹性（Encapsulation）。也许读者会觉得 TMorning 中定义的两个过程 Init 和 AssignStr 完全是多余的，取消它们，将 31 行的语句换为

```
Morning.TextStr := 'Good Morning!'
```

即可。确实如此，对于这样的一个微不足道的小程序，你有充分的理由这样做。但对于大型的程序，这种做法就不可取了。因为 OOP 的一个原则是：尽量避免直接访问对象中定义的数据，对这些数据的操作应由对象本身提供的算法来完成。

2.2 对象的继承性和虚方法

OOP 的另一个特点是继承性（Inheritance）。应用程序可以定义一个对象为一个已定义的对象的子对象（descendant object），而前者则称为其父对象（ancestor object）。子对象继承了父对象所有的数据和方法。现在我们为对象 TMorning 定义一个子对象，将上述程序修改如下：

```
Program OOPDemo1;                          {line 1}

Uses WinCrt;                               {line 3}

Type TMorning=object                       {line 5}
     TextStr:string;
     Procedure Init;
     Procedure AssignStr;
     Procedure Display;
     End;

TAfternoon=object(TMorning)                {line 12}
     Procedure Init;
     Procedure AssignStr;
     End;
```

```
Procedure TMorning.Init;                    {line 17}
Begin
    AssignStr;
End;

Procedure TMorning.AssignStr;               {line 22}
Begin
    TextStr := 'Good morning!';
End;

Procedure TMorning.Display;                 {line 27}
Begin
    GotoXY(25,10);
    write(TextStr);
End;

Procedure TAfternoon.Init;                  {line 33}
Begin
    AssignStr;
End;

Procedure TAfternoon.AssignStr;             {line 38}
Begin
    TextStr := 'Good afternoon!';
End;

Var Afternoon: TAfternoon;                  {line 43}

Begin                                       {line 45}
    Afternoon.Init;
    Afternoon.Display;
End.
```

从第 12 行到第 15 行定义了一个对象 TAfternoon。第 12 行 TAfternoon = Object (TMorning)说明 TAfternoon 是 TMorning 的子对象，于是 TAfternoon 就自动拥有了 TMorning 的所有数据和方法。也就是说，TAfternoon 一样有一个字符串变量 TextStr 和三个过程 Init，AssignStr 和 Display。但是在 10 和 11 行，我们看到 TAfternoon 又重新定义了 Init 和 AssignStr。这种做法称为替换(override)。子对象往往需要对父对象所定义的方法做一些修改，在这种情况下，子对象可重新定义这些方法而不改变其命名(OOP 的多形性)。在上例中，重新定义 Init 和 AssignStr 是为了给 TextStr 赋予一个不同的字符串：'Good afternoon!'

第 17 行到 41 行是两个对象中所有方法的说明部分。其中 33 行到 36 行是 TAfternoon 的方法 Init 的说明部分，和 TMorning.Init 的说明一模一样。第 38 行到 41 行是 TAfternoon 的方法 AssignStr 的说明部分，这里我们赋予 TextStr 一个不同的字符串。

43 行给了对象类型 TAfternoon 一个实例 Afternoon。45 到 48 行是程序的执行部分，调用 Afternoon 的两个过程 Init 和 AssignStr。

再次运行这个程序，我们看到窗口上显示的是“Good afternoon!”而不是“Good morning!”。这是因为，当程序执行 46 行语句 Afternoon. Init 时，该过程将调用 AssignStr，由于 AssignStr 已被重新定义，故程序调用 TAfternoon 的 AssignStr 而不是 TMorning 的 AssignStr，结果 TextStr 的内容就是'Good afternoon!'而不是'Good morning!'。其后的语句是 Afternoon. Display，过程 Display 由 TMorning 继承而来。

细心的读者会发现，TAfternoon. Init 和 TMorning. Init 的定义是一模一样的，既然 TAfternoon 已继承了过程 Init，而且无需改动，为什么还要再定义它？让我们去掉第 43 行，同时把 33 到 36 行也去掉。然后编译这个程序，语法上毫无问题，编译顺利通过。让我们运行这个程序，奇怪的是，这一回我们看到的是“Good morning!”而不是“Good afternoon!”。

问题在编译程序那里。当编译程序编译 TMorning 的 Init 时，该过程调用 AssignStr，于是编译程序按编译原则寻找 AssignStr 在最近的范围内的定义，自然是找到了 TMorning. AssignStr 的定义，于是编译该过程的代码供 Init 调用。当程序运行时，第 46 行语句是调用 TAfternoon 的 Init 过程，该过程是由 TMorning 继承来的，于是调用 TMorning 的 Init 过程。如前所述，该过程调用的将是 TMorning 的 AssignStr 过程，从而在 TextStr 中存放的将是字符串“Good morning!”。所以，在上例中 TAfternoon 重新定义了 Init，这使得编译程序去寻找当前对象中定义的过程 AssignStr，从而避免了上述情况发生。

但是，这只是为了说明问题而故意这样做。在一个实用的程序中，一个对象中定义的方法可能被该对象中很多个其它的方法所调用，若程序替换了其中一个方法就必须把所有调用它的方法都重新定义一遍，岂不麻烦透顶？实际上我们有简单得多的方法来解决这问题。这就是虚方法(virtual method)。

现在我们将程序修改如下：

```
Program OOPDemo1;                                    {line 1}

Uses WinCrt;                                         {line 3}

Type TMorning=object                                 {line 5}
     TextStr:string;
     Constructor Init;
     Procedure AssignStr; virtual;
     Procedure Display; virtual;
     End;

  TAfternoon=object(TMorning)                        {line 12}
     Procedure AssignStr; virtual;
     End;

Constructor TMorning.Init;                           {line 16}
Begin
     AssignStr;
```