概　论

Borland 公司推出的 Turbo 系列语言 Turbo BASIC、Turbo C、Turbo Pascal、Turbo Prolog、Turbo Assembler、Turbo Debugger、Turbo C++和 Borland C++是一系列功能强大、速度很高的程序设计语言。在这些语言软件包中，还配套提供了一系列实用工具软件。使用它们，可以大大方便程序的开发工作，提高编程的效率。本书以某个版本的实用工具为蓝本，介绍这些工具软件的功能和用法。其他语言软件包和其他版本软件包中的同名实用程序的功能和用法，与本书介绍的可能略有差别，读者可以参考本书，具体用法可以查阅与软件配套的手册。下面是这些工具程序的功能简介及我们所依据的该实用程序的版本。

(1) BGIOBJ：图形驱动程序和字模的转换工具程序（2.0 版）。

(2) BINOBJ：将其它格式文件转换为.OBJ 文件（6.0 版）。

(3) CPP：C 语言程序预处理程序（2.0 版）。

(4) GREP：文件搜索工具（3.0 版）。

(5) IMPDEF：为动态连接库（DLL）建立一个模块定义文件。

(6) IMPLIB：为动态连接库（DLL）建立一个输入库（import library）。

(7) MAKE：独立的程序管理器。该工具程序能帮助程序员维护当前版本上的程序，保证它总是最新的（3.5 版）。

(8) OBJXREF：目标模块交叉参考器（3.0 版）。

(9) PRJ2MAK：将 Borland C++的工程文件转换为 MAKE 文件（2.0 版）。

(10) PRJCFG：从配置文件中修改一个工程文件中的选项，或将一个工程文件转换为一个配置文件（2.0 版）。

(11) PRJCNVT：将 Turbo C 的工程文件转换为 Borland C++的格式（2.0 版）。

(12) TCREF：源模块交叉引用实用程序（2.0 版）。

(13) THELP：Turbo Help 联机帮助工具程序（2.0 版）。

(14) TLIB：Turbo 库管理程序（3.01 版）。

(15) TLINK：Turbo 连接程序（4.0 版）。

(16) TOUCH：改变文件的日期和时间的工具（3.0 版）。

(17) TPUMOVER：Turbo Pascal 的单元管理程序（6.0 版）。

(18) TRANCOPY：从一个工程向另一个工程拷贝传送项（2.0 版）。

(19) TRIGRAPH：字符转换工具（1.0 版）。

另外，还有一个功能强大的可用于处理编辑器的宏语言，它称为 TEML（2.0 版）。

本书将详细地介绍每个工具程序，对每个工具程序都给出使用的例子，包括代码和命令行，还说明如何使用它们。

第一章 BGIOBJ

可以使用 BGIOBJ 来将图形驱动程序文件和字符集（笔划式字模文件）转换为目标（.OBJ）文件。在将它们转换后，就可以将它们连接到程序中，使它们成为可执行文件的一部分。这是对图形软件包动态装入方法的补充，在动态装入方法中，是在运行时从磁盘上装入图形驱动程序和字符集（笔划式字模）的。

将驱动程序和字模直接连接到程序的优点是很明显的，因为这样可执行文件就包含了所有（或大部分）需要的驱动程序和字模，而不必在运行时再访问磁盘上的驱动程序和字模文件了。但是，将驱动程序和字模连接到可执行文件中也将增加该文件的长度。

要将驱动程序或字模文件转换为一个可连接的目标文件，可以以如下语法使用实用工具程序 BGIOBJ.EXE：

```
BGIOBJ source_file
```

这里，source_file 是要转换为目标文件的驱动程序或字模文件。生成的目标文件具有与源文件相同的名字，但扩展名为.OBJ。例如，EGAVGA.BGI 将生成 EGAVGA.OBJ，SANS.CHR 生成 SANS.OBJ，依此类推。

1.1 将新的.OBJ 文件加到 GRAPHICS.LIB 中

可以将驱动程序和字模目标模块添加到 GRAPHICS.LIB 中，以使连接器在连接图形例程时能找到它们。如果不将这些新的目标模块添加到 GRAPHICS.LIB 中，就要在工程（.PRJ）文件、BCC 命令行或在 TLINK 命令行中将它们添加到文件表中。要将这些目标模块添加到 GRAPHICS.LIB 中，可以以如下命令使用 TLIB：

```
tlib graphics +object_file_name [+object_file_name ...]
```

这里，object_file_name 是由 BGIOBJ.EXE 建立的目标文件名（如 CGA、EGAVGA、GOTH，等等）。扩展名.OBJ 是隐含的，因此不必写出。可以在一个命令行中写上几个文件，以节省时间。详见下节的例子。

1.2 登录驱动程序和字模

在将驱动程序和字模目标模块添加到 GRAPHICS.LIB 后，就要登录需要连接的所有驱动程序和字模，这可以通过在程序中调用 registerbgidriver 和 registerbgifont 实现，它们通知图形系统已经给出了所需的文件，以保证连接程序在建立可执行文件时能将它们连接到程序中。

每个登录例程带一个参数，这是一个在 graphics.h 中定义的符号名。如果成功地登录了驱动程序或字模，登录程序就返回一个非负值。

下面列出了 registerbgidriver 和 registerbgifont 使用的名字，它是 Borland C++中包

含的所有驱动程序和字模的完整列表。

驱动程序文件 (*.BGI)	registerbgidriver 符号名	字模文件 (*.CHR)	registerbgifont 符号名
CGA	CGA _driver	TRIP	triplex_font
EGAVGA	EGAVGA_ driver	LITT	small_font
HERC	Herc _driver	SANS	sansserif_font
ATT	ATT_driver	GOTH	gothic _font
PC3270	PC3270_driver		
IBM8514	IBM8514_driver		

1.3 例　子

假设要登录 CGA 图形驱动程序、gothic 字模和 triplex 字模时，将文件转换为目标模块，然后将它们连接到程序中。下面是执行的步骤：

(1) 用 BGIOBJ.EXE 将二进制文件转换为目标文件，使用以下命令行：

```
bgiobj  cga
bgiobj  trip
bgiobj  goth
```

这将建立三个文件：CGA.OBJ、TRIP.OBJ 和 GOTH.OBJ。

(2) 可以用以下 TLIB 命令行将这些目标文件添加到 GRAPHICS.LIB 中：

```
tlib  graphics  +cga  +trip  +goth
```

(3) 如果没有将这些目标文件添加到 GRAPHICS.LIB 中，就需要将目标文件名 CGA.OBJ、TRIP.OBJ 和 GOTH.OBJ 加到工程列表中（如果使用 Borland C++的集成环境），或者加到 BCC 的命令行中。例如，BCC 的命令行应象下面这样调用：

```
BCC  niftgraf  graphics.lib  cga.obj  trip.obj  goth.obj
```

(4) 在图形程序中以下面的方法登录这些文件（注意：如果在连接某些驱动程序或字模时出现连接错误“Segment exceeds 64K-段超过 64K”，就参见下一节）：

```
/* 声明 CGA_driver, triplex_font & gothic_font 的头文件 */
#include <graphics.h>

/* 登录并检测错误 */

if (registerbgidriver (CGA_driver) < 0) exit (1);
if (registerbgifont (triplex_font) < 0) exit (1);
if (registerbgifont (gothic_font) < 0) exit (1);
```

```
/* ... */

initgraph (...);                          /* initgraph 应在 registering 之后调用 */

/*   */
```

1.4 / F 选项

本节说明在将几个驱动程序和字模文件连接到程序中后（特别是在 samll 和 compact 模式下），在出现"Segment exceeds 64K（段超过 64K）"或相似的错误时，应该怎么处理。

默认情况下，由 BGIOBJ.EXE 建立的文件都使用同一个段（称为__TEXT）。但是如果程序中连接了许多驱动程序和字模，或者在使用 small 或 compact 模式时，就会出问题。

为了解决这个问题，可以用 BGIOBJ 的 / F 选项来转换一个或多个驱动程序和字模。该选项使 BGIOBJ 使用一个名为"文件名__TEXT"的段，这样，默认段就不会对连接的所有驱动程序和字模（在 small 和 compact 模式的程序中为所有的程序代码）负担太重。例如，下面的两个 BGIOBJ 命令行让 BGIOBJ 使用名为 EGAVGA__TEXT 和 SANS__TEXT 的段。

```
bgiobj  / F  egavga
bgiobj  / F  sans
```

当选择 / F 选项时，BGIOBJ 还将 F 拼到结果目标文件名中（即 EGAVGAF.OBJ、SANSF.OBJ、等等），将__far 拼加到要被 registerfarbgidriver 和 registerfarbgifont 使用的名中（例如，EGAVGA__driver 变成了 EGAVGA__driver__far）。

对用 / F 建立的文件，必须使用远程登录例程，而不能使用常规的 registerbgidriver 和 registerbgifont。例如：

```
if  (registerfarbgidriver  (EGAVGA__driver__far)  <  0)  exit  (1);
if  (registerfarbgifont  (sansserif__font__far)  <  0)  exit  (1);
```

1.5 高级特征

本节讨论 BGIOBJ 的一些高级特征和 registerfarbgidriver 和 registerfarbgifont 例程。这一节是专门为熟练的用户准备的。

下面是 BGIOBJ.EXE 命令行的完整语法：

```
BGIOBJ  [/ F] source destination public-name seg-name seg-class
```

下表说明 BGIOBJ 命令行的每个组成部分。

组成部分	说　明
/F 或 -F	本选项让 BGIOBJ.EXE 使用另一个段名，而不是使用__TEXT（默认段名），并改变全局名和目标文件名。参见前一节的说明
<source>	这是要转换的驱动程序和字模文件，如果文件不是 Borland 公司提供的驱动程序或字模文件，则需要指定完整的文件名（包括扩展名）
<destination>	这是生成的目标文件名。默认的目标文件名为 source.OBJ，或 sourceF.OBJ（如果使用了/F 选项）
public-name	这是程序中在调用 registerbgidriver 或 registerbgifont（或对应的远程版本）来连接目标模块时将使用的名字。 公用名是被连接器使用的外部名，因此它应该是在程序中使用的名字前缀以下划线。如果程序使用了 Pascal 调用约定，就只能使用大写字母，并且不再加上下划线
seg-name	这是一个可选的段名；默认段为__TEXT（如果指定了/F 选项，则为"文件名__TEXT"）
seg-class	这是一个可选的段类；默认值为 CODE

除了 source 外的所有参数都是可选的。但是，如果要指定一个可选的参数，就必须指定它前面的所有参数。

如果选择使用自己的公用名，就要向程序中加上声明，形式如下：

```
void public__name(void);                    /* 如果没有使用/F，就使用默认的段名 */
extern int far public__name[];              /* 如果使用了/F，或使用了不是__TEXT 的段 */
```

在这些声明中，public__name 匹配在 BGIOBJ 转换时用的公用名。头文件 graphics.h 中包含了默认的驱动程序和字模的公用名的声明；如果使用这些默认公用名，就不必如刚才所说的声明它们了。

在这些声明后，需要在程序中登录所有的驱动程序和字模。如果没有使用/F 选项，没有改变默认的段名，就应用 registerbgidriver 和 registerbgifont 来登录驱动程序和字模；否则，就使用 registerfarbgidriver 和 registerfarbgifont。

下面是一个将字模文件装入到内存中的例子程序。

```
/* 将字模文件装入到内存中的例子 */

#include <graphics.h>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <process.h>
#include <alloc.h>

main()
{
```

```
void        * gothic_fontp;                    /* 指向内存中的字模缓冲区 */
int         handle;                            /* 用于输入输出的文件句柄 */
unsigned    fsize;                             /* 文件和缓冲区的大小 */

int errorcode;
int graphdriver;
int graphmode;

/* 打开字模文件 */
handle = open ("GOTH.CHR", O_RDONLY|O_BINARY);
if (handle == -1)
{
    printf ("unable to open font file 'GOTH.CHR'\n");
    exit (1);
}
/* 找出文件的大小 */
fsize = filelength (handle);
/* 分配缓冲区 */
gothic_fontp = malloc (fsize);
if (gothic_ fontp == NULL)
{
  printf ("unable to allocate memory for font file 'GOTH.CHR'\n");
  exit (1);
}
/* 将字模读入内存 */
if (read (handle, gothic_fontp, fsize) != fsize)
{
    printf ("unable to read font file 'GOTH.CHR'\n");
    exit (1);
}
/* 关闭字模文件 */
close (handle);
/* 登录字模 */
if (registerfarbgifont (gothic_fontp) != GOTHIC_FONT)
{
    printf ("unable to register font file 'GOTH.CHR'\n");
    exit (1);
}
/* 检测和初始化图形系统 */
graphdriver = DETECT;
initgraph (&graphdriver, &graphmode, "");
errorcode = graphresult();
if (errorcode != grOk)
{
    printf ("graphics error: %s\n", grapherrormsg (errorcode));
```

```
    exit (1);
  }
  settextjustify (CENTER_TEXT, CENTER_TEXT);
  settextstyle (GOTHIC_FONT, HORIZ_DIR, 4);
  outtextxy (getmaxx() /2, getmaxy() /2, "Borland Graphics Interface (BGI) ");

  /* 按任一键结束 */
  getch();
  /* 关闭图形系统 */
  closegraph();
  return (0);
}
```

第二章　BINOBJ

在 Turbo Pascal 中，有一个名叫 BINOBJ.EXE 的实用工具程序，它的作用与上一章介绍的 BGIOBJ 相似。它可以将其他文件转换为.OBJ 文件，以使该文件能作为一个"过程"连接到 Turbo Pascal 程序中。如果有一个必须驻留在代码段的二进制数据文件，或者有一个太大的无法放到类型常量数组的二进制数据文件时，这是非常有用的。例如，可以将 BINOBJ 用于 Graph 单元，以将图形驱动程序或字模文件直接连入.EXE 文件中。然后，在使用图形程序时，就只须有.EXE 文件即可。

BINOBJ 带三个参数，如下所示：

BINOBJ <source[.BIN]> <destination[.OBJ]> <public name>

这里，source 是要转换的二进制文件名，destination 是产生的.OBJ 文件名，而"public name"是在 Turbo Pascal 程序中被声明的过程的名字。

在下面的例子中，过程 ShowScreen 带一个指针参数，它将 4000 个字节的数据移到屏幕内存中。名为 MENU.DTA 的文件包含主菜单屏幕的映像（$80 \times 25 \times 2 = 4000$ 字节）。

以下是 MYPROG.PAS 的一个简单版本（无错误检测功能）：

```
program MyProg;

procedure ShowScreen (var ScreenData: Pointer);
{显示全屏数据，但无错误检测！}
var
  ScreenSegment: Word;

begin
  if (Lo (LastMode) = 7) then                          {单色？}
    ScreenSegment := $B000
  else
    ScreenSegment := $B800;
  Move (@ScreenData^,                                  {从指针}
    Ptr (ScreenSegment, 0)^,                           {移到视频内存}
    4000);                                             {80 × 25 × 2}
end;

var
  MenuP: Pointer;
  MenuF: file;
begin
  Assign (MenuF, 'MENU.DTA');                          {打开屏幕数据文件}
```

```
  Reset (MenuF, 1);
  GetMem (MenuP, 4000);                          { 在堆上分配缓冲区 }
  BlockRead (MenuF, MenuP^, 4000);               { 读屏幕数据 }
  Close (MenuF);
  ShowScreen (MenuP);                            { 显示屏幕 }
end.
```

屏幕数据文件（MENU.DTA）被打开，然后被读到堆上的缓冲区中，为了使该程序正确工作，必须提供 MYPROG.EXE 和 MENU.DTA 文件。可以用 BINOBJ 将 MENU.DTA 转换为一个.OBJ 文件（MENUDTA.OBJ），并告诉它将这些数据与名为 MenuData 的过程相连，然后可以声明外部过程 MenuData，其中包含着实际的屏幕数据。在将.OBJ 文件用 $L 编译指令连接到程序中后，MenuData 将为 4000 字节长，并且包含着屏幕数据。

首先，对 MENU.DTA 运行 BINOBJ，如下所示：

```
binobj MENU.DTA MENUDTA MenuData
```

第一个参数 MENU.DTA 为屏幕数据文件；第二个参数 MENUDTA 为要建立的.OBJ 文件名（在不指定扩展名时，将加上.OBJ）；最后一个参数 MenuData 为要在程序中声明的外部过程的名字。在将 MENU.DTA 转换为.OBJ 文件后，新的 MYPROG.PAS 程序将如下所示：

```
program MyProg;

procedure ShowScreen (ScreenData : Pointer);
{ 显示全屏幕数据，无错误检测！ }
var
  ScreenSegment : Word;
begin
  if (Lo (LastMode) = 7) then                    { 单色？ }
    ScreenSegment := $B000
  else
    ScreenSegment := $B800;
  Move (@ScreenData^,                            { 从指针 }
    Ptr (ScreenSegment, 0)^,                     { 移到视频内存 }
    4000);                                       { 80 × 25 × 2 }
end;

procedure MenuData; external;
{ $L MENUDTA.OBJ }
begin
  ShowScreen (@MenuData);                        { 显示屏幕 }
end.
```

注意，ShowScreen 一点没有改变，过程的地址使用@操作符传递。

下面我们再给出一个例子。

2.1 例　子

本例子程序（BGILINK.PAS）演示如何将图形驱动程序和字模文件连接到一个.EXE 文件中。BGI 图形驱动程序和字模文件是保存在磁盘上的独立文件中的，它们一般是在运行时被动态地连接。但是，有时候将所有的辅助文件都直接放到.EXE 文件中更适合些。在本程序中，它的 MAKE 文件（BGILINK.MAK）和两个单元（BGIDRIV.PAS 和 BGIFONT.PAS）将所有的驱动程序和字模直接连接到 BGILINK.EXE 文件中。

本例子要用到几个工具程序，必须将它们放在当前驱动器和目录下，或者设置适当的路径，使得能够引用它们。这些工具程序是：

MAKE.EXE	建立BGILINK.EXE的Make工具程序；
BINOBJ.EXE	将文件转换为.OBJ文件的工具程序。

在当前驱动器和目录下，还应有以下几个文件：

BGILINK.PAS	例子的主程序；
BGIDRIV.PAS	将连接所有BGI驱动程序的Pascal单元；
BGIFONT.PAS	将连接所有BGI字模的Pascal单元；
*.CHR	BGI字模文件
*.BGI	BGI驱动程序文件
BGILINK.MAK	建立BGIDRIV.TPU、BGIFONT.TPU和BGILINK.EXE的 make 文件。

步骤：(1) 在 DOS 提示符下键入以下命令，对 BGILINK.MAK 运行 MAKE 工具：

```
make -fBGIlink.mak
```

通过 BINOBJ.EXE，首先根据驱动程序文件（*.BGI）建立.OBJ 文件，然后调用 Turbo Pascal 编译 BGIDRIV.PAS。然后，将字模文件（*.CHR）转换为.OBJ 文件，并编译 BGIFONT.PAS。最后，编译 BGILINK.PAS，它要用到 BGIDRIV.TPU 和 BGIFONT.TPU。

(2) 运行 BGILINK.EXE。它包含了所有的驱动程序和所有的字模，因此它能在被 Graph 支持和带图形卡（CGA、EGA、EGA 64 K、EGA 单色、Hercules 单色、VGA、MCGA、IBM 3270 PC 和 AT&T 6400）的任何系统上运行。

说明：BGILINK.PAS 在 uses 语句中用到了 BGIDRIV.TPU 和 BGIFONT.TPU，如下所示：

```
uses BGIDriv, BGIFont;
```

然后，它登录要使用的驱动程序，此时为所有的驱动程序，因此它能在任何图形卡上运行；接着，它登录所有要使用的字模；最后，画出非常朴素的图形。

很容易为各种应用修改 BGILINK.PAS 程序，并且在程序对 RegisterBGIdriver 和 RegisterBGIfont 的调用中删去不用的驱动程序和字模。

关于登录和连接驱动程序和字模的详细说明，请读者参阅 Turbo Pascal 的有关资料。

下面是 BGILINK.PAS 程序的清单：

```
program BgiLink;

uses  Graph,                                          { 图形例程库 }
      BGIDriv,                                        { 所有的BGI驱动程序 }
      BGIFont;                                        { 所有的BGI字模 }
var
  GraphDriver,  GraphMode,  Error : integer;

procedure Abort  (Msg : string);
begin
  Writeln  (Msg,  ' ',  GraphErrorMsg  (GraphResult));
  Halt  (1);
end;

begin
  { Register all the drivers }
  if RegisterBGIdriver  (@CGADriverProc)  < 0 then
    Abort  ('CGA');
  if RegisterBGIdriver  (@EGAVGADriverProc)  < 0 then
    Abort  ('EGA / VGA');
  if RegisterBGIdriver  (@HercDriverProc)  < 0 then
    Abort  ('Herc');
  if RegisterBGIdriver  (@ATTDriverProc)  < 0 then
    Abort  ('AT&T');
  if RegisterBGIdriver  (@PC3270DriverProc)  < 0 then
    Abort  ('PC 3270');

  { Register all the fonts }
  if RegisterBGIfont  (@GothicFontProc)  < 0 then
    Abort  ('Gothic');
  if RegisterBGIfont  (@SansSerifFontProc)  < 0 then
    Abort  ('SansSerif');
  if RegisterBGIfont  (@SmallFontProc)  < 0 then
    Abort  ('Small');
  if RegisterBGIfont  (@TriplexFontProc)  < 0 then
    Abort  ('Triplex');

  GraphDriver  =  Detect;                             { 自动检测硬件 }
  InitGraph  (GraphDriver,  GraphMode,  '');          { 激活图形 }
  if GraphResult  < >  grOk then                      { 有错否？ }
```

```
  begin
    Writeln ('Graphics init error: ', GraphErrorMsg (GraphDriver));
    Halt (1);
  end;

  MoveTo (5, 5);
  OutText ('Drivers and fonts were ');
  MoveTo (5, 20);
  SetTextStyle (GothicFont, HorizDir, 4);
  OutText ('Built ');
  SetTextStyle (SmallFont, HorizDir, 4);
  OutText ('into ');
  SetTextStyle (TriplexFont, HorizDir, 4);
  OutText ('EXE ');
  SetTextStyle (SansSerifFont, HorizDir, 4);
  OutText ('file!');
  Rectangle (0, 0, GetX, GetY + TextHeight ('file!') + 1);
  ReadIn;
  CloseGraph;
end.
```

下面是 BGIDRIV.PAS 的程序清单:

```
unit BGIDrv;

interface

procedure ATTDriverProc;
procedure CgaDriverProc;
procedure EgaVgaDriverProc;
procedure HercDriverProc;
procedure PC3270DriverProc;

implementation

procedure ATTDriverProc; external;
{$L ATT.OBJ}

procedure CgaDriverProc; external;
{$L CGA.OBJ}

procedure EgaVgaDriverProc; external;
{$L EGAVGA.OBJ}

procedure HercDriverProc; external;
```

```
{$L HERC.OBJ }

procedure PC3270DriverProc;  external;
{$L PC3270.OBJ }

end.
```

以下是 BGIFONT.PAS 的程序清单：

```
unit BGIFont;

interface

procedure GothicFontProc;
procedure SansSerifFontProc;
procedure SmallFontProc;
procedure TriplexFontProc;

implementation

procedure GothicFontProc;  external;
{$L GOTH.OBJ }

procedure SansSerifFontProc;  external;
{$L SANS.OBJ }

procedure SmallFontProc;  external;
{$L LITT.OBJ }

procedure TriplexFontProc;  external;
{$L TRIP.OBJ }

end.
```

最后，我们给出 BGILINK.MAK 的内容如下：

```
#建立使用 BGIFONT.TPU 和 BGIDRIV.TPU 的样本程序
bgilink.exe: bgidriv.tpu bgifont.tpu
  tpc bgilink /m

#建立所有要连接的字模的单元
bgifont.tpu: bgifont.pas goth.obj litt.obj sans.obj trip.obj
  tpc bgifont
goth.obj: goth.chr
  binobj goth.chr goth GothicFontProc
```

```
litt.obj: litt.chr
  binobj litt.chr litt SmallFontProc
sans.obj: sans.chr
  binobj sans.chr sans SansSerifFontProc
trip.obj: trip.chr
  binobj trip.chr trip TriplexFontProc

#建立所有要连接的驱动程序的单元
bgidriv.tpu: bgidriv.pas cga.obj egavga.obj herc.obj pc3270.obj att.obj
  tpc bgidriv
cga.obj: cga.bgi
  binobj cga.bgi cga CGADriverProc
egavga.obj: egavga.bgi
  binobj egavga.bgi egavga EGAVGADriverProc
herc.obj: herc.bgi
  binobj herc.bgi herc HercDriverProc
pc3270.obj: pc3270.bgi
  binobj pc3270.bgi pc3270 PC3270DriverProc
att.obj: att.bgi
  binobj att.bgi att ATTDriverProc
```

第三章　CPP

CPP 在一个文件中生成 C 语言源程序的列表。这些源程序的包含文件和#define 宏被扩展。对正常的 C 程序编译，并不需要使用本工具程序。

许多时候，当编译器报告在宏内或在包含文件中有一个错误时，如果能看到包含文件或宏扩展的结果，就可以更详细地了解发生了什么错误。在许多多遍编译器中，有一个独立的遍执行这项工作，并且对这遍的结果进行检查。由于 Borland C++使用了一个集成的单遍编译器，因此单独给出了在其他编译器中作为一遍功能的 CPP。

可以象使用 BCC 一样使用 CPP，独立的编译器 CPP 从 TURBOC.CFG 中读取默认选项，接受与 BCC 相同的命令行。

与 CPP 无关的 BCC 选项将被 CPP 忽略，如果读者想了解 CPP 能处理的参数表，可以在 DOS 提示符下敲入 CPP。

只有一个例外，在 CPP 命令行上列出的文件名与在 BCC 命令行上同样看待，并且允许使用通配符。这个例外是所有的文件都被当作 C 语言源程序看待。对.OBJ、.LIB 或.ASM 文件不进行特殊处理。

对每个被 CPP 处理的文件，输出被写到当前目录的一个文件中（或写到-n 选项命名的输出目录中），输出文件与源名相同，只是扩展名被改为.I。

输出文件是包括源文件的每一行和所有包含文件的文本文件。其中删去了所有的预处理指令，同时所有的条件文本行也在编译时被删除。如果不用命令行选项指定其他情况，文本行将前缀以源文件或包含文件的文件名和行号。在文本行内，所有的宏都被以扩展文本代替。

注意：CPP 的输出结果不能被编译，因为在每一行的前面都缀上了文件名和行号。

3.1　将 CPP 用作宏预处理器

CPP 的-P 选项告诉它，在每一行的前面都缀以源文件的行号。如果给出-P-（即关闭此选项），CPP 就省略此行号信息。在关闭此选项后，就可以将 CPP 用作宏预处理器；生成的结果.I 文件就可以被 BC 或 BCC 编译了。

3.2　例　　子

下面的简单程序演示 CPP 如何预处理一个文件，首先选择-P，然后再选择-P-。

下面是源文件 HELLOAJ.C 的内容：

```
#define NAME "AJ McInnis"
#define BEGIN {
```

```
#define END    }

main()
BEGIN
   printf ("%s\n",  NAME);
END
```

下面是将 CPP 当作预处理器激活的命令行:

```
CPP HELLOAJ.C
```

输出为:

```
HELLOAJ.c 1:
HELLOAJ.c 2:
HELLOAJ.c 3:
HELLOAJ.c 4:
HELLOAJ.c 5: main()
HELLOAJ.c 6: {
HELLOAJ.c 7:     printf ("%s\n",  "AJ McInnis");
HELLOAJ.c 8: }
```

下面是将 CPP 作为宏预处理器激活的命令行:

```
CPP -P- HELLOAJ.C
```

输出为:

```
main()
{
   printf ("%s\n",  "AJ McInnis");
}
```

第四章 GREP

GREP（Global Regular Expression Print——全局正则表达式打印）是一个源于UNIX系统的强有力的实用查找程序，它可以快速在多个文件或在其标准输入流中查找所需的文本。

本节我们将给出几个各种情况的例子，以帮助读者学习如何使用GREP。例如，如果想要看看在哪个源文件中调用了Setupmodem函数，可以用GREP来查找目录下所有.TXT文件的内容，以寻找Setupmodem串。如下所示：

```
GREP  Setupmodem  *.TXT
```

GREP将显示每个包含"Setupmodem"串的文件行的列表。由于在默认情况下，GREP是大小写敏感的，所以字符串"Setupmodem"和"SETUPMODEM"被认为是不同的字符串。可以设置适当的查找选项，以忽略大小写的区别。

GREP除了查找固定的字符串外，还可以做许多其他事情。在下面的几个小节中，我们将学习如何用GREP来查找匹配不同模式的字符串。

GREP遵循的命令行语法如下：

```
GREP  [查找选项]  查找串  文件名  < 文件名  文件名…文件名>
```

命令：

```
GREP ?
```

将打印一个显示GREP命令行选项、特殊字符和默认值的帮助屏幕。关于如何改变GREP默认值的信息，读者可以参见下面的-u命令行选项。

4.1 GREP的选项

在命令行中，选项是以连字符号"-"打头的若干个字符，它们可以改变GREP的行为。每个单独的字符都是一个可打开或关闭的开关，在该字符后打入加号就打开该选项，打入减号就关闭掉该选项。

缺省状态为打开，因此加号"+"是可选的。例如，-r的意义与-r+的意义相同。可把多个选项一个个地列出（如-i，-d，-l），也可把它们组合起来（如-idl或-il -d等），它们对GREP来说作用是相同的。

下面列出GREP所用的选项字符及它们的意义：

-c 只计数　只打印匹配行的数目。对每个至少含有一个匹配行的文件，GREP就打印出文件名和匹配行的数目，而不打印匹配行。本选项默认为关闭。

-d 子目录　对命令行上所指定的每个文件名，GREP将查找与文件名相匹配的所有文件，查找的目录是所指定的目录及所指定目标下面的所有子目录。如果所给的文件名不带路径，GREP就假