在 FoxBASE+状态下显示彩色立体投影汉字的方法

宋立波　(辽宁省铁岭市委办公室)

目前在中文汉字系统 2.13 平台上广泛使用的 FoxBASE+数据库管理系统虽然支持汉字直接显示功能,但其不具有显示特殊效果的彩色汉字功能。如果能在 FoxBASE+的多种功能菜单状态下直接显示彩色立体投影等特殊效果的汉字,可使我们设计的系统用户界面更加丰富生动美观。

一、FoxBASE+与汇编语言的接口

FoxBASE+为我们提供了直接加载和执行二进制文件的功能调用接口,其调用格式如下:

1.LOAD FILENAME.BIN

其中 FILENAME.BIN 为被加载的二进制文件名,该项功能是直接将编译后的二进制文件加载到内存中;

2.CALL FILENAME.BIN

其中 FILENAME.BIN 为被执行的二进制文件名,该项功能是直接执行加载到内存的二进制文件。

利用这个功能调用接口,我们只要将自己编制的程序生成二进制文件后,即可直接在 FoxBASE+状态加载执行,于是我们就可以利用汇编语言或其他高级语言为其增加彩色空心或立体汉字的显示功能。

二、FoxBASE+和其他语言的二进制文件结构

1.FoxBASE+调用的二进制文件结构

COM 格式和 EXE 格式的可执行文件均可转换为二进制文件,FoxBASE+ 系统调用的二进制文件,除了具有一般二进制文件的格式外,还要求可加载执行的二进制文件必须从文件的第一个字节开始执行,文件以 RETF 远调用返回方式结束,文件中均以近调用 NEAR 方式调用各个子程序,即其必须是 COM 文件和 EXE 文件的混合格式;而一般情况下的 COM 文件和 EXE 可执行文件转换成二进制文件时,要求其原可执行文件的代码段、数据段和堆栈段的总长度小于 64KB 字节长度,即所有的 COM 文件均属于二进制文件,长度小于 64KB 没有堆栈段、没有数据段的 EXE 文件可转换为二进制文件。综上所述,只有利用特殊格式编制的可执行文件转换为二进制文件后方可被 FoxBASE+直接调用。

2.C 语言生成 FoxBASE 二进制文件的可能性

C 语言高级编译系统为我们提供了 6 种基本编译模式:Tiny.Medium.Small. Compact.Large.Huge。这 6 种编译模式中只有最小编译模式 Tiny 的代码段(CS)、数据段(DS)和堆栈段(SS)的总长度小于 64K 字节,并同时指向程序的相同地址,可直接转换成 COM 格式文件,即生成二进制文件。但这种二进制文件不是从文件的第一个字节开始执行的,也不是以远调用返回 RETF 方式结束文件的,即主函数 MAIN()的起始地址和结束方式不符合上述 FoxBASE+调用的二进制文件格式要求。虽然可以通过修改连接库文件 COT.OBJ 制作符合 FoxBASE+调用格式,使编译后的 Tiny 模式二进制文件符合 FoxBASE+格式的调用要求,但其修改和形成过程比较繁琐,对一般人员来讲实现起来比较困难。

3.用汇编语言编制 FoxBASE+二进制文件的方法

由于汇编语言的编程格式比较灵活直观,所以这里笔者提供了使用汇编形成 FoxBASE+调用格式二进制文件的方法和步骤:首先采用 COM 文件格式;然后去掉 COM 文件中的代码段、数据段和堆栈段的地址定义语句 ORG;最后在第一条可执行语句前面加上定义的远调用过程,并将程序退出部分用过程返加 RET 和远过程结束 ENDP 作为结束。上述具体步骤的详细过程请参见文件程序清单。然后使用 BIOS 中断 10H 的 OCH 子功能,直接在屏幕上写点即可完成具有各种效果汉字的显示功能。

三、实用程序范例及使用方法

根据上述程序设计的基本方法,本文提供了在 FoxBASE+ 状态下直接显示彩色立体投影汉字的实用程序。该程序直接使用 2。13 汉字系统中的楷体 24 点阵汉字字模作为被放大汉字字模数据,省去了字模的读取过程。其立体投影显示的算法与 C 语言中使用的算法相同,请读者参考有关方面的文章。程序中实现汉字立体投影显示的速度为原来汉字显示速度的 2 倍,这在用户菜单系统中完全可以满足要求。程序中有关参数的具体说明请参考注释部分。

二进制文件的生成方法如下:

C>MASM DISPLT
C>LINK DISPLT
C>EXE2BIN DISPLT.EXE DISPLT.BIN
在 FoxBASE+状态下的调用步骤为:
• LOAD "I=DILPLT.BIN"
• CALL"DISPLT.BIN"

该程序在 286 以上微机使用效果理想。程序可直接在 CGA/EGA 显示器的 04H-06H 图形模式下和 VGA/TVGA 的 0DH-12H、5B H-5FH 图形模式下正常使用。

```
;DISPLT. ASM
COLOR1  EQU 02H      ;汉字显示颜色
COLOR2  EQU 03H      ;立体显示颜色
STARTX  EQU 00H      ;汉字的起始列坐标
STARTY  EQU 20H      ;汉字的起始横坐标
COUNTD EQU 30H       ;汉字间距点数
COUNTLTEQU 04H       ;汉字立体点数
CODE    SEGMENT'CODE'
ASSUME  CS:CODE,DS:CODE,ES:CODE
DISP    PROC FAR ;FOXBASE++调用格式
START:  JMP BEGIN;楷体汉字"计算机"的24点阵字模
HZMODE DB 00H, 00H,00H,00H,20H,00H,00H,60H
       DB 00H,00H,60H,00H,00H,40H,60H,40H
       DB 0E3H,0F0H,71H,0FFH,0E0H,39H,0C1H,0C0H
       DB 38H,01H,80H,00H,23H,00H,00H,22H
       DB 00H,00H,20H,00H,00H,60H,00H,00H
       DB 60H,00H,0C0H,60H,00H,0FFH,0FFH,0FFH
       DB 7FH,0DFH,0FEH,00H,0C0H,00H,00H,0C0H
       DB 00H,00H,0C0H,00H,00H,0C0H,00H,00H
       DB 0C0H,00H,00H,40H,00H,00H,00H,00H
       DB 00H,00H,00H,00H,00H,00H,00H,00H
       DB 30H,00H,00H,30H,01H,00H,20H,02H
       DB 00H,21H,0CH,00H,23H,78H,80H,6EH
       DB 74H,0FFH,0FCH,17H,0EEH,0F0H,11H,4AH
       DB 40H,31H,5AH,40H,15H,52H,40H,19H
       DB 56H,40H,0F3H,07H,0FFH,0EBH,0FFH,0FFH
       DB 2DH,0F8H,0C0H,2CH,00H,80H,60H,00H
       DB  80H,60H,00H,80H,20H,01H,80H,00H
       DB 01H,0C0H,00H,00H,0C0H,00H,00H,00H
       DB 00H,00H,00H,00H,80H,40H,00H,0C1H
       DB 80H 00H,0C3H,00H,00H,8EH,00H,01H
       DB 0BCH,0CH,0FFH,0FFH,0FCH,0FFH,0FFH,0F8H
       DB 01H,18H,10H,03H,1CH,10H,03H,00H
       DB 60H,04H,01H,0C0H,07H,0FFH,80H,07H
       DB 0F0H,00H,06H,00H,00H,04H,0CH,00H
       DB 0FH,0FFH,0C0H,0FH,80H,0E0H,04H,00H
       DB 60H,00H,00H,30H,00H,00H,30H,00H
       DB 00H,70H,00H,07H,0F0H,00H,00H,0E0H
COUNT   DW 0003H             ;汉字个数
MODENUM DB 00H               ;汉字点阵列字节计数
LTNUM   DW 0000H             ;中间控制单元
DISPLT  PROC NEAR
COUNTL1:MOV MODENUM,03H ;一列3字节点阵数据
        PUSH BX
        PUSH DX
        MOV BH,00H           ;显示页面
COUNTL2:MOV BL,80H           ;点阵字模测试位
COUNTL3:TEST BL,DS           :[SI]
        JZ COUNTL5
        MOV AH,0CH           ;画点显示功能
        MOV AL,C9LOR2
        MOV LTNUM,COUNTLT
        ADD CX,0001H
        SUB DX,0001H
COUNTL4:INT 10H
        DEC DX
        INC CX
        DEC LTNUM
        CMP LTNUM,0000H      ;判断立体部分结束
        JNZ COUNTL4
        SUB CX,COUNTLT+1
        ADD DX,COUNTLT+1
        MOV AL,COLOR1
        INT 10H
COUNTL5: INC DX
        SHR BL,01H           ;逐位测试
        CMP BL,00H           ;所有点均画完否
        JNZ COUNTL3
        INC SI
        DEC MODENUM          ;3字节均显示完否
        CMP MODENUM,00H
        JNZ COUNTL2
        POP DX
        INC CX
        POP BX
        DEC BX               ;一个汉字显示完否
        CMP BX,00H
        JNZ COUNTL1
        RET
DISPLT  ENDP
BEGIN:  PUSH SP              ;保存现场
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        PUSH DI
        PUSH SI
        PUSH DS
        PUSH CS
        POP DS
        MOV SI,0FFSET HZMODE ;字模地址
        MOV CX,STARTX        ;显示位置的列坐标
COUNT6: MOV DX,STARTY        ;显示位置的行坐标
        MOV BX,24            ;显示一个汉字
        CALL DISPLT
        ADD CX,COUNTD        ;字间距调整
```

```
        DEC COUNT              ;判断多个汉字显示完否
        CMP COUNT,00H
        JNZ COUNT6
        POP DS                 ;恢复现象
        POP SI
        POP DI
        POP DX
        POP CX
        POP BX
        POP AX
        POP SP
        ;MOV AX,4C00H
        ;INT 21H
        IRET
DISP    ENDP                   ;远调返回
CODE    ENDS
        END START
```

浅谈 Borland C++4.0 的异常处理机制

贾晓东　(上海大学)

摘要:本文从抛出异常,捕获异常,对异常的处理以及异常指定等几个方面详细地讨论了 *Borland C++4* 的异常处理机制,分析了其不足之处并对解决办法进行了探讨。

异常指的是程序中出现了异常情况,通常指出错。异常处理是指让程序的一部分检测及报告异常情况而另一部分处理它们。Borland C++4.0 定义了异常处理的标准,这个标准确保程序始终受到面向对象的支持。在这个标准中值得强调的是引入了虚函数和利用对象定义异常,虚函数确保了程序具有最少的运行时间,若无异常抛出则程序额外的时间耗费将是零。Borland C++4.0 同时也支持 ANSI 标准,支持中止异常处理模式,程序运行时若有异常情况发生程序将中止,但是抛出一个异常后从抛出点收集信息,这些信息在研究程序失败的原因时大有用处。也可以在异常处理中指定程序终止前要采取的措施。

Borland C++4.0 允许异常为任意类型,但是最好把异常定义成对象,这样做得有好处,异常对象将和其他对象一样对待。

Borland C++4.0 的异常处理机制要求用到三个关键字:try,catch,throw,由 try 关键字来标识的 try 块后必须紧跟着由 catch 标识的异常处理程序块。如果 try 块抛出一个异常,程序控制权就转让给适当的异常处理程序块。程序应当试图捕获由任何函数抛出的异常,不能这样做时,将导致程序的异常终止。一个异常带着信息从抛出点来到捕获点,这些信息将告诉程序使用者在运行程序时遭遇到的异常。下面我们从几个方面来具体谈谈 Borland C++4.0 的异常处理机制。

一、抛出异常

C++中异常处理技术建立在非局部转移这一概念的基础上,当程序遭遇到一个异常情况时程序就抛出此异常情况,抛出异常会使程序在继续运行之前执行一个非局部转移把控制权转让给程序中另外一部分来处理这种问题。异常的抛出一般发生在 try 块中,一个有可能产生异常的代码块必须含有关键字 try 并具有下面的形式:

```
try{
  / / Take some action;
  }
```

这将指示程序准备测试异常,如果异常产生了程序流将中断,程序搜索匹配的处理程序,如果找到了堆栈回绕到这点,程序控制转让给处理程序块,如果没有找到程序将调用 terminati()函数。如果没有异常抛出程序将按正常顺序执行。异常的抛出方式有四种:

1.throw throw __ object,这种方式指定了 throw __ object 是传给异常处理程序块的异常。

2,throw,这种方式简单地指定最后抛出的异常被再抛出,至少要有一个异常抛出否则程序调用 terminate 函数中止程序。

3.void my_func() throw(A,B)

```
{
  / / Body of function;
  }
```

这种方式给出了函数 my func()能抛出的异常的表,除表中的异常没有其他的异常能传出函数体。如果一个既非 A 又非 B 的异常在 my func()函数中产生了,它将使得程序的控制转交给 unexpected()函数。

4.void my_func() throw()

```
{
   / / Body of function;
}
```

这种方式说明 my_func()不会抛出任何异常,若 my_func()中调用的函数抛出了异常,该异常在 my_func()函数体外将不可见。