汉字点阵显示程序实践

23090032047 计算机类 1 班 于景一

1. 实践思路

本题正确的输出结果可能有两种，一种是*横向输出*，另一种是*纵向输出*。由于*纵向输出*完全可以通过简单循环三次解决，并无技术难度，因此实践中一笔带过。对于*横向输出*，需要完整理解代码，是本次探究的重点。

1.1. 在 Windows 的准备工作

本次实验课强制要求使用 Visual Studio 2022 版本，由于本地未部署，应先安装。

做好如下准备工作：

① **从官网下载 Community 版：**https://visualstudio.microsoft.com/

② **如何配置 Visual Studio 中的 C 支持：**https://learn.microsoft.com/zh-cn/cpp/build/vscpp-step-0-installation

图 1.1-1 正确配置工作负载

1.2. 在 Linux 的准备工作

我将继续使用我的 Ubuntu 服务器作为实验平台。由于此发行版显然地内置了 gcc 支持，故惟一需要强调的问题是如何将文件上传到服务器。

在这里我*根据以往经验*（**保证原创**），提供两种解决方案：

① **使用 SFTP 协议：**对于任何一个并未配置 FTP、并未放行 FTP 端口的云服务器（通常为 20 和 21），只要它能通过 SSH 远程连接，我们都可以便利地使用 OpenSSH 软件包内

置的 SFTP（通过 SSH）进行文件传输操作。

我们知晓，SFTP 是使用 SSH 协议传输文件的，因此具有完整的认证信息，安全性很高（等同于连接 SSH 的安全性）。然而，由于复杂的加密解密技术，其文件传输效率要明显低于灵活的 FTP 协议。不过，这依然不妨碍我们方便地使用它。

那么，在实践中我们只需安装一个支持 SFTP 的客户端即可。我使用的 SSH 工具 Termius 本身便包含了 SFTP 功能。

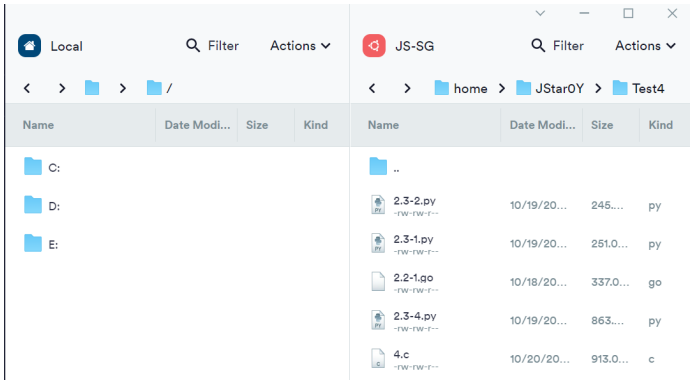

图 1.2-1 Termius 中的 SFTP 功能

② 使用 FTP（FTPS）协议：FTP 是非常出色的文件传输协议，总体来说具有“主动模式（Port）”和“被动模式（Passive）”两种传输模式。在一般情况下，如果 FTP 需要通过公网传输，我们需要配置 SSL/TLS 加密以提高安全性（也就是 FTP over TLS）。

所谓主动与被动，都是以服务端为中心语的。主动就是“服务器从自己的端口（20）**向**客户端的端口发起请求，**客户端同意后**，可以对这个端口进行文件传输操作”，而被动是“服务器随机打开一个端口，**被**客户端发起请求，**服务器同意后**，可以对这个端口进行文件传输操作”。

主动模式便于服务器管理，只需要服务器打开 20、21 端口即可。然而，服务器要求客户端打开端口的行为，可能收到防火墙阻碍。

被动模式便于客户端管理，客户端只需打开一个随机的端口。这种操作在客户端处不易受阻。然而，此操作会提高服务器负载，原因是在许多客户端同时操作的情况下，服务器需要为每个客户端开启不同的端口。

往昔在我对客户端配置的实践中，通常选用 FileZilla 这一开源程序。此程序支持 FTP、FTPS、SFTP 等多个协议。

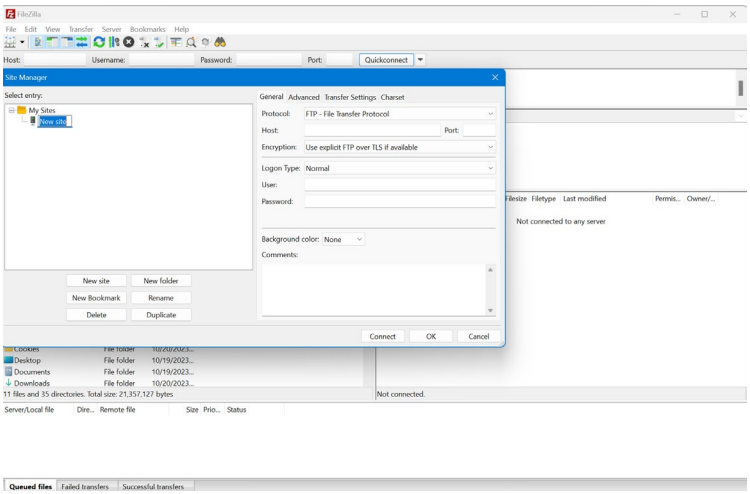

图 1.2-2 FileZilla 之快照

在服务端，可以使用 `proftpd` 或 `vsftpd` 等程序，在 Ubuntu 使用 apt 安装，并对相应的文件进行配置即可。唯一需要强调的是，由于文件传输通过公网，务必启用 SSL/TLS 认证。在此不加赘述。

1.3. 源码分析

我们应先对代码有一个整体的把握，再进行具体的实验操作。首先，该文件使用 GB 2313 编码，为正确显示中文文本，我们在 VS Code 使用 GB2312 重新打开。

这里要引起注意：程序一定要用 GB2312 书写的原因，必然与点阵字库有关。经查阅，chs16.fon 使用 GB2312 编码，是标准的汉字编码。若使用 UTF-8，并不能正确读取 GB2312 的汉字信息。故一般情况下，我们不能转换该程序源码的编码，而是继续使用 GB2312。

注意：所有程序源码将附在 *4.程序源码* 中。

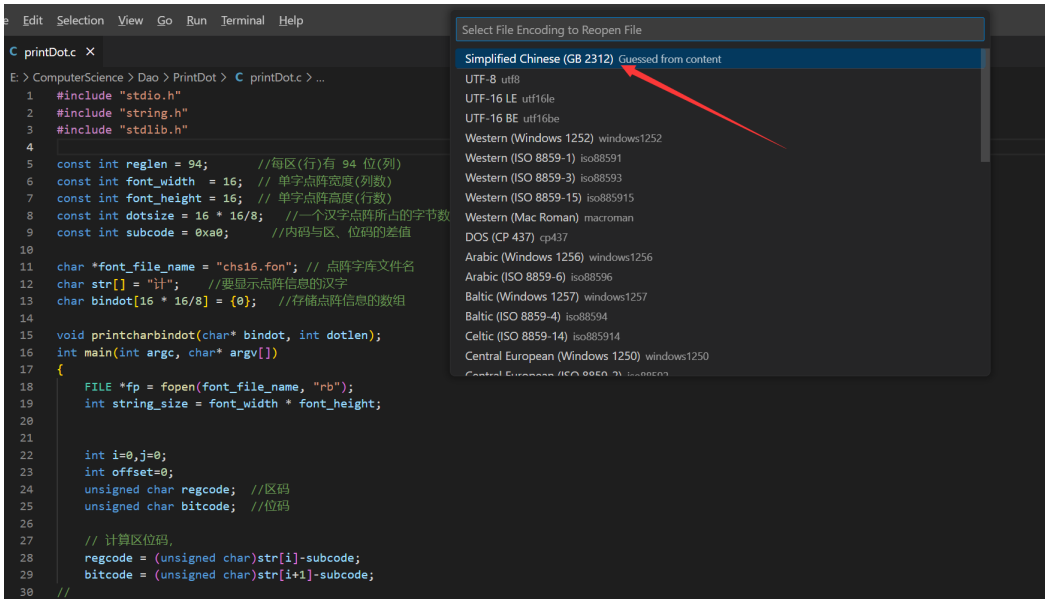

图 1.3-1 VS Code 修改编码以正确打开源码

① 思想：

打开字库文件，计算汉字在字库中的位置，读取该汉字的点阵数据（二进制数），并以字符串的形式存储到一个字符数组中【由于一个字节可存储 8 个二进制数，因此我们在定义字节数时会对点阵数（二进制）进行/8（变为字节）的操作】，最后依序输出该字符数组的二进制值，注意每 16 位须换行以正确显示点阵。

② 关键常量与变量

✧ 汉字点阵所占字节常量：`dotsize = 16 * 16/8`

✧ 一维字符数组：`bindot[16 * 16/8]`

✧ 存储待显示汉字：`str[]`

注：字符型数据在内存中是以 8 位为一个字节存储。因此如果我们要用 char 来存储一个 16*16 的点阵，该点阵由 0 和 1 组成，便需要将二进制的 16*16 分为 8 个一组存储，即实际存储的总字节数为 16*16/8。

③ 关键函数：

printcharbindot(char* bindot, int len)中

依次输出存储在字符串 bindot 中的每一个字符

for (charnum =0; charnum < len; ++charnum)

解读每一个字符（每个字节）中的二进制值，按从高到低的位次

for(bitindex = 7;bitindex>=0; --bitindex)

每位 bitvalue = ((bindot[charnum]>>bitindex) & 0x1):

bitvalue 存储了应该输出的数值，通过 bitvalue+'0'转化为字符输出。

charnum 用来标记并读取 bindot 字符数组的某项。

bitindex 是使该字节的二进制值，向右偏移并丢弃的位数。

对于>>运算符，我们认识到他是向右按位移动并丢弃。如：

a>>b 其中 a=60 (60 = 0011 1100); b=2.

那么 a>>b=15 (15 = 0000 1111).

对于&按位与运算符，我们认识到：

对二进制右移操作后的值，我们只保留其二进制的最后一位（最右侧）的特征（若为 1，则为 1；若为 0，则为 0），而其他位由于 0x1 除最后一位都为 0，结果就必为 0.

当 bitnum 满 16，即已输出过 16 个二进制值（2 字节）时，进行换行。

④ **修改方案：**

a) **纵向输出：**只需要分别读取并输出字符，即可得到目标结果。方案如下：

每次输出都调整读入的汉字；循环进行三次读取、输出操作即可。

```
for(int i = 0; i < 3; i++) {
    if(i==0) str[] = ‘于’; //在此处以我的姓名为例
    else if(i==1) str[] = ‘景’;
    else str[] = ‘一’;
    FILE *fp = fopen(font_file_name, "rb"); …… //main()中读取、输出操作
    fclose(fp);
```

当然，我们可以采取先统一读取，再统一输出的方式进行。其实现方式如下：

首先重新扩大字符串变量的容量：二维字符串数组。

```
char str[3][4] = {"于", "景", "一"};
```

定义二位数组取代 bindot，存储点阵信息。

```
char binlib[3][16*16/8] = {0};
```

创建一个 readbindot()函数，传入变量 q 使每次写入、读取对应到 binlib[q]

```
void readbindot(int q)
```

修改 fread()函数。

```
fread(binlib[q], 32, 1, fp);
```

在主函数的实现即：

```
for(int w=0; w<3; w++) {
      redingote(w);
      //输出其点阵信息
      printcharbindot(binlib[w], dotsize);
}
```

b) **横向输出：**读取三个汉字后，依序统一输出。方案如下：

基于对源码的理解和上面纵向输出的实现，我们可以首先分别读入三个汉字，存储到binlib 这个二维数组中。然后，每行依序输出这三个字符数组的 16 个二进制数后再换行。如此即可实现横向输出。

我们已经新定义了 binlib 这个二维数组，因此无需在 printcharbindot()函数中传入 bindot 了。改为:

```
void printcharbindot(int dotlen);
```

由于此方法需要统一读取、统一输出。故在主函数中:

```
for(int w=0; w<3; w++) readbindot(w);
//输出其点阵信息
printcharbindot(dotsize);
```

需要深度修改 printcharbindot()函数，修改后的代码如下:

```
void printcharbindot(int len)
{

    int charnum = 0;  //当前字节号
    int bitnum  = 0;  //已读取的位数
    int bitindex = 0;  //当前位号
    int bitvalue;     //当前位的值
    for (charnum = 0; charnum < len;) //在这里 charnum 只用来确定是否输出了三个字的每个字节
    {
        for(int n = 0; n < 3; n++)  //共输出三次，每次对应一个汉字
        {
            //从高到低顺次输出一个字节的每位信息
            for(int m = 0; m < 2; m++)  //一个汉字每行需要输出两次“8 个二进制位”
            {
                for(bitindex = 7; bitindex>=0; --bitindex)  //遍历一个字节的每一位
                {
                    //输出当前字节第 bitindex 位的值
                    bitvalue = ((binlib[n][charnum]>>bitindex) & 0x1 );
                    if(bitvalue==1) printf("■");
                    else printf("  ");
                }
                charnum++; //暂时提升到第二个字节
            }
            charnum-=2; //清除暂时提升，返回第一个字节
            printf("  "); //满 16 位（两个字节）输出两个空格
        }
        charnum += 2;  //每行三个字都遍历完两个字节后，提升到下一行的第一个字节
        printf("\n"); //输出过三个字后换行
    }
```

```
}
```

c) **强化显示效果：**1 和 0 的不易人眼区分，需要提高可辨识度。方案如下：

已知 1 代表黑色，0 代表白色，那么在输出时用“■■”替换“1”；用“　”（两个空格）替换“0”即可。输出两个字符的原因是换行的单行行高总大于一个字符的宽，使用两个字符可以调节横竖比例。由于编码原因，我们需要在 terminal 中同样使用 GB2312 才能正确显示“■”符号，如 PowerShell 中，我们使用：

```
> chcp 936
```

更改 Code Page 为 GB2312。

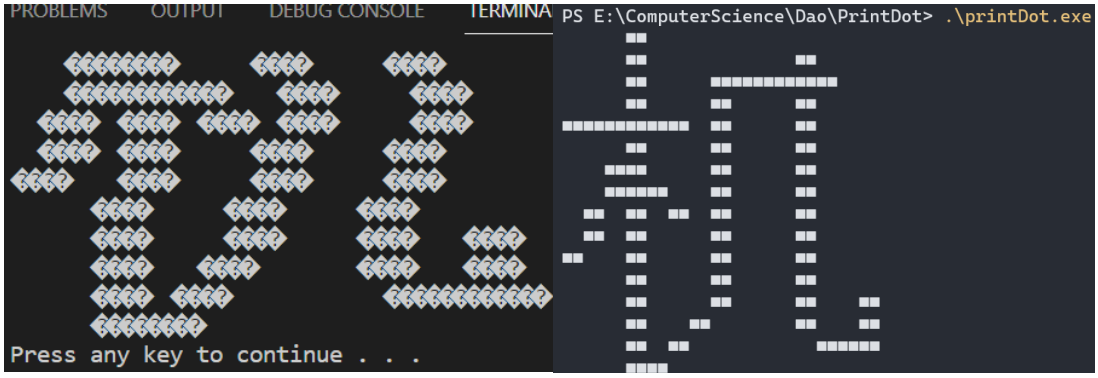

图 1.4.c-1~2 乱码显示与正常显示

代码实现非常简单，将 `printf("%c", bitvalue+'0');` 替换即可：

```
if(bitvalue==1) printf("■■");
else printf("  ");
```

2. 进行实验

2.1. 在 Windows 安装 Visual Studio

在准备环节我们已确定了安装版本，在这里我们直接启动安装向导。

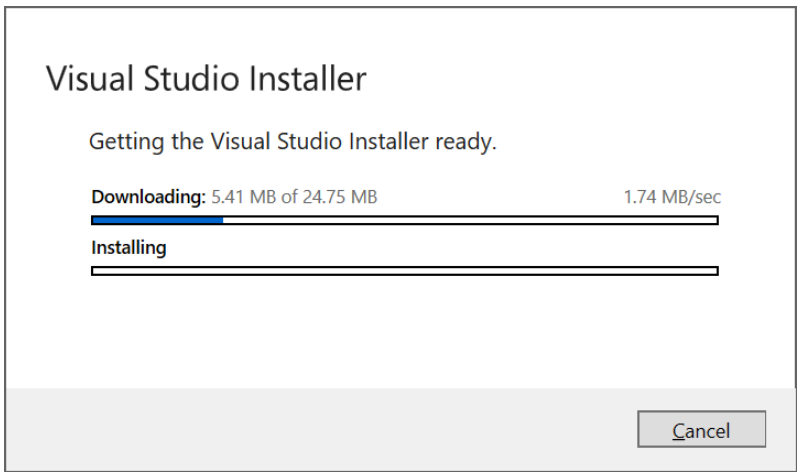

图 2.1-1 开启安装向导

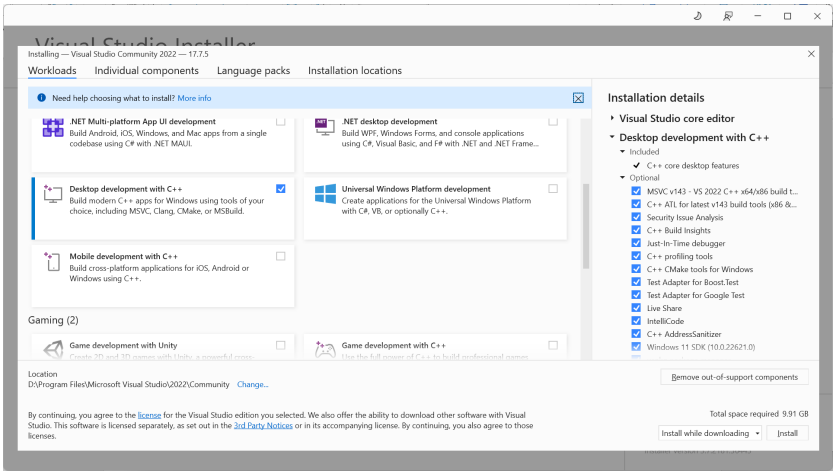

图 2.1-2 根据 1.1.2 配置工作负载

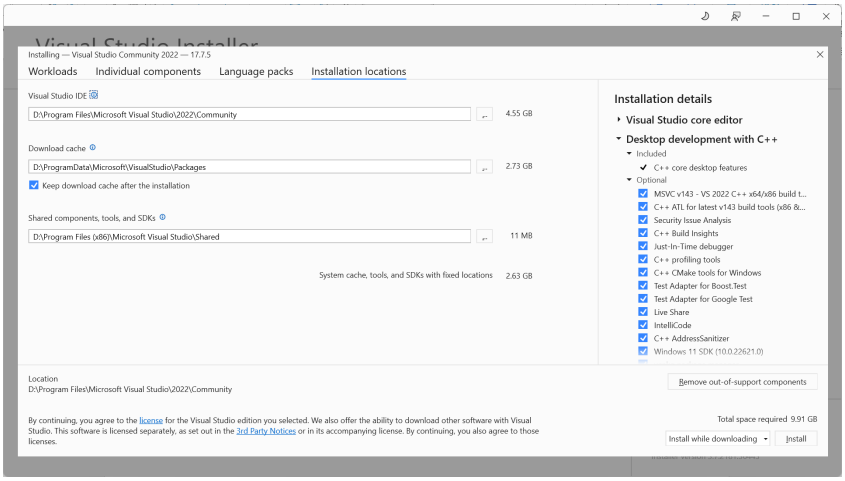

图 2.1-3 配置安装目录

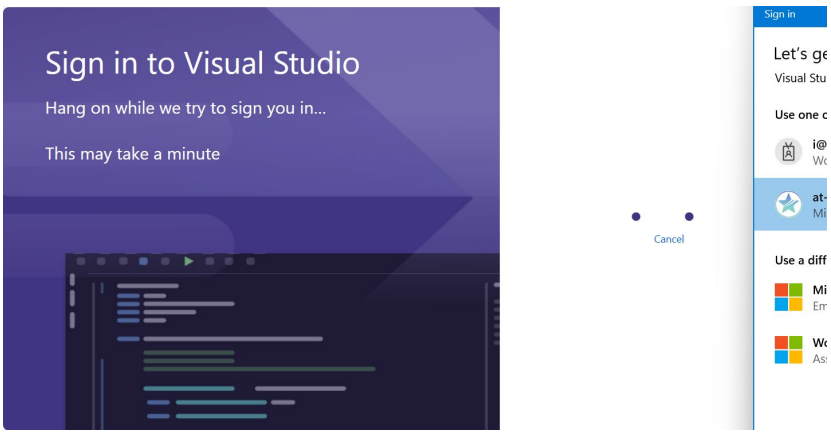

图 2.1-4 登录并启动

2.2. 使用 VS 2022 调试代码

由于在第一节我们已编辑好代码，我们直接新建空白工程，导入 C 相关源码文件。

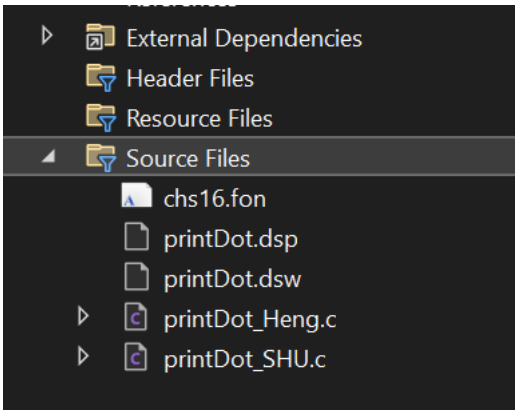

图 2.2-1 导入文件

在同一工程下，默认所有 C 源码文件将都编译，然后连接成一个完整的 C 程序，这也是 C 语言的性质。意味着同时导入多个独立的源码并不能被正确编译。解决方法是将其中一个文件暂时移除：Exclude From Project

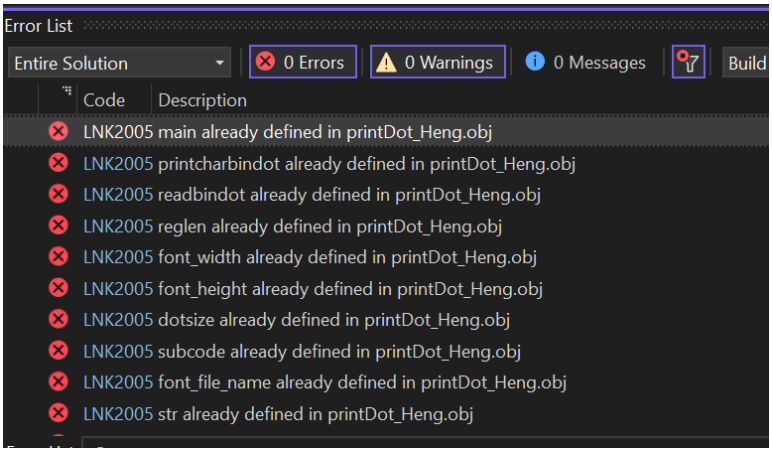

图 2.2-2 编译多个文件报错

直接编译代码遭报错：

```
Error C4996 'fopen': This function or variable may be unsafe. Consider
using fopen_s instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS.
```

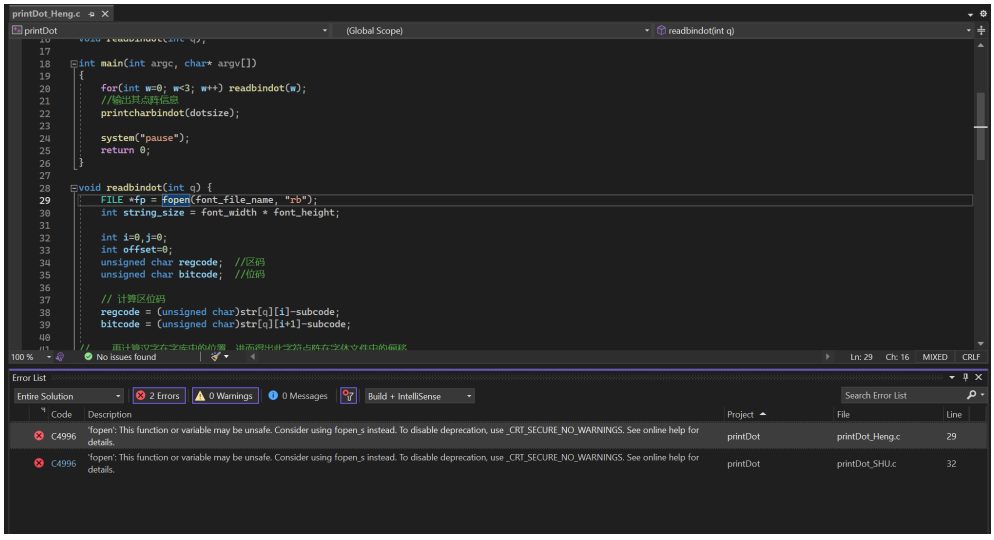

图 2.2-3 直接编译报错

这个错误是 Microsoft 编译器引起的，它认为 fopen()函数不安全。在这里我们考虑两种解决方法：

① 使用 fopen_s()函数代替

② 禁用警告：在 Project > Properties > Configuration Properties > C/C++ > Preprocessor > Preprocessor Definitions 添加“_CRT_SECURE_NO_WARNINGS”字样以屏蔽警告。

出于对代码完整性的考虑，我们采用第二种解决方法。

编译后运行，依然报错：

```
Unhandled exception at 0x00007FFA3477829C (ucrtbased.dll) in PrintDot-1.exe: An invalid parameter was passed to a function that considers invalid parameters fatal.
```

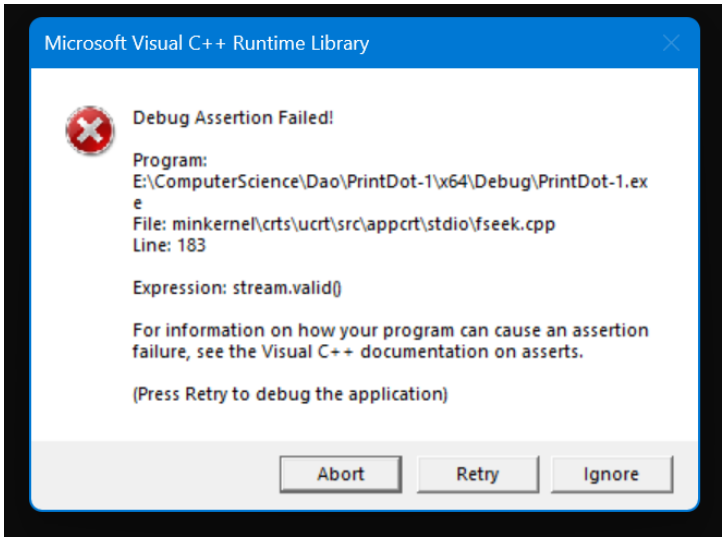

图 2.2-4-1 运行程序抛出异常

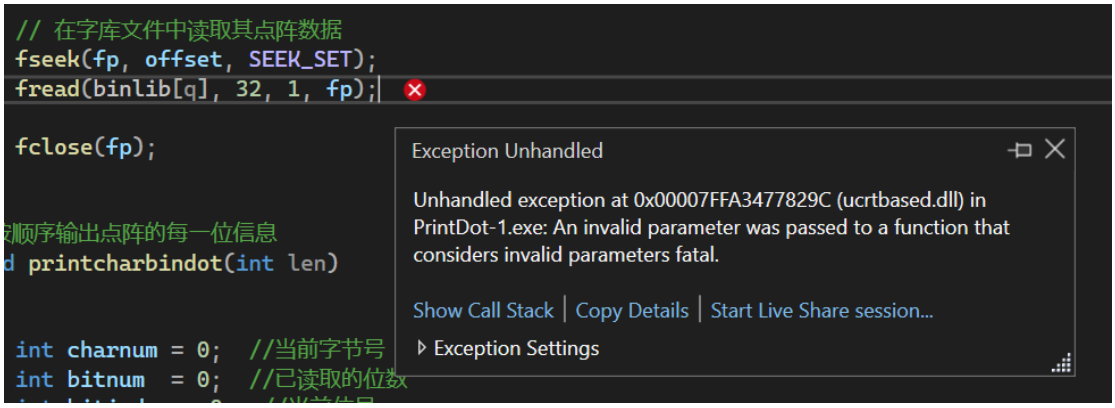

图 2.2-4-2 Debug 发现在 49 行抛出异常

经过分析知道，传递了一个无效的参数。那么无效的参数是什么呢？既然程序可以正常在 VS Code 中编译运行，意味着致错点在 Visual Studio 的不同的环境。

Visual Studio 为 Debug 提供了独立的编译后的输出路径，在./x64/Debug/中。意味着我们定义的点阵字库与编译后的程序并不在同一个目录了。

应修改 `char *font_file_name = "chs16.fon"` 该行代码，改用绝对路径，如：

```
char *font_file_name = "C:/chs16.fon"
```

修改后，能正确运行程序，见下图：

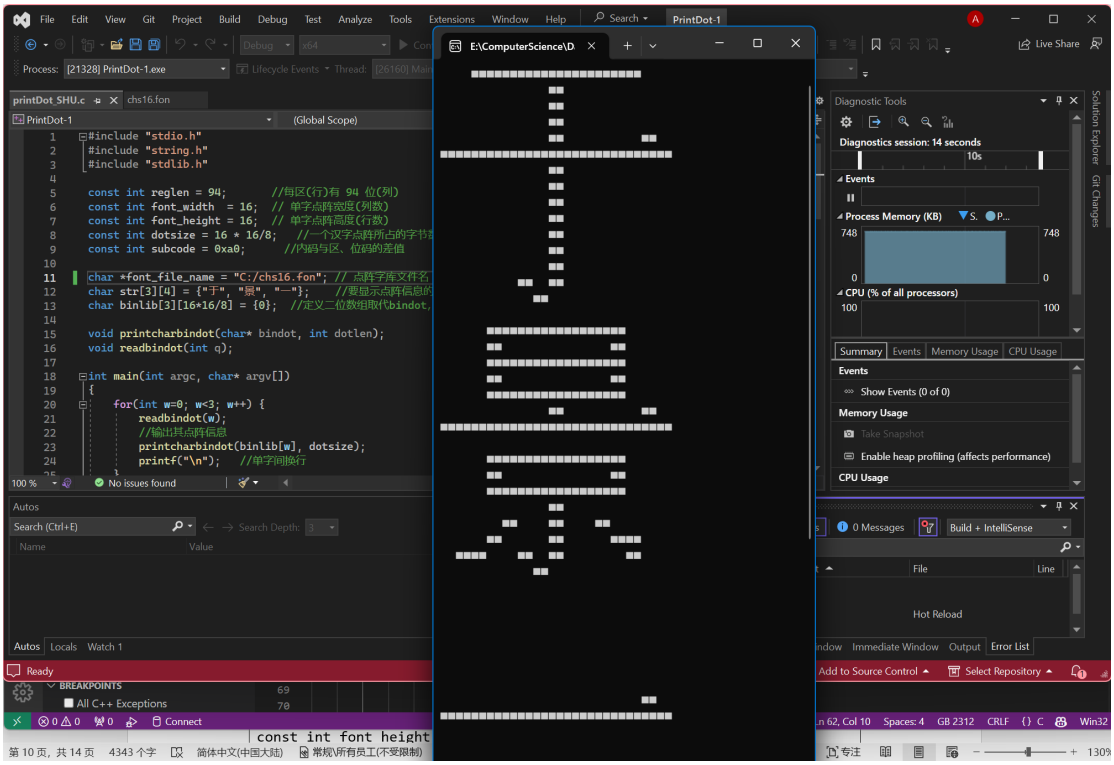

图 2.2-5-1 竖排运行结果

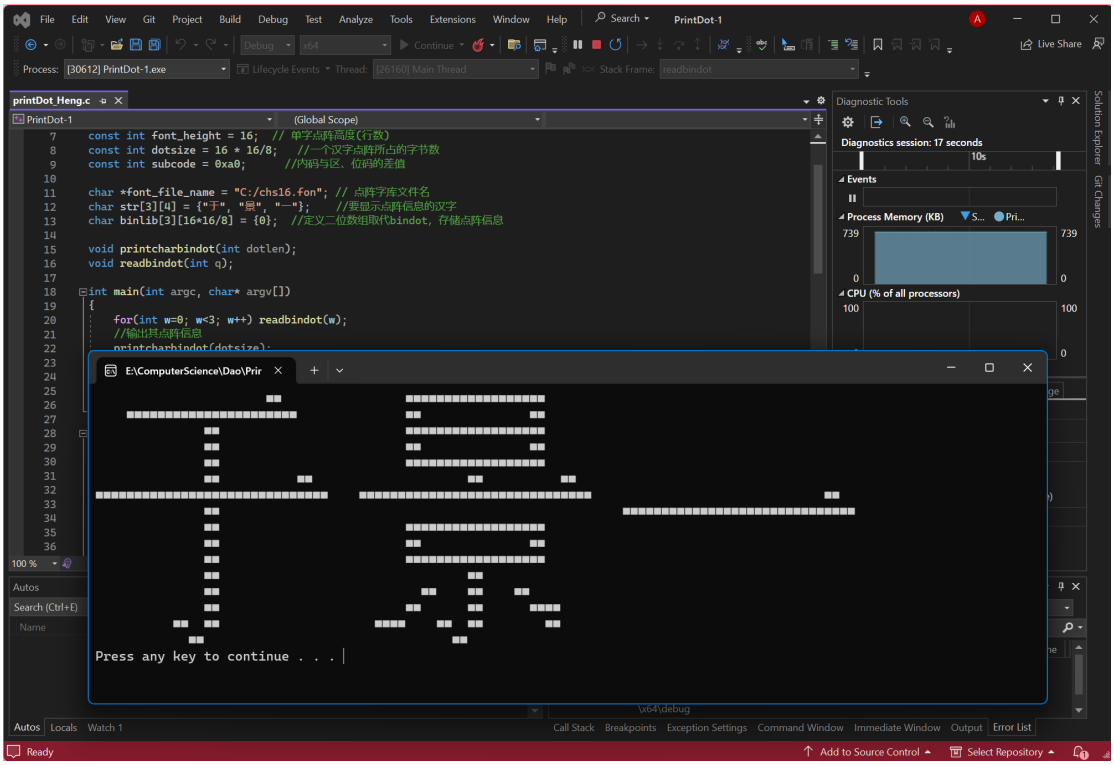

图 2.2-5-2 横排运行结果

2.3. 在 Linux 编译运行相关代码

使用 SFTP 上传源代码文件。

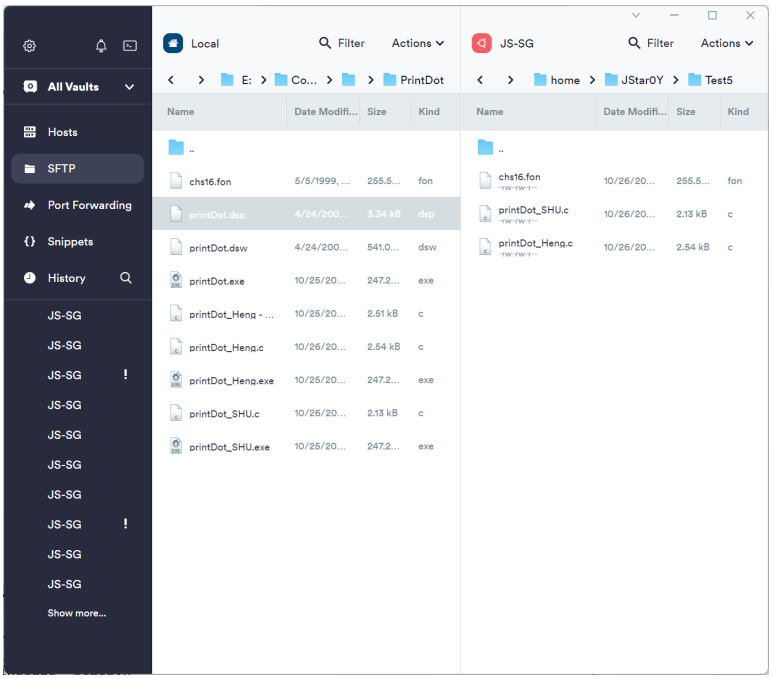

图 2.3-1 SFTP 上传源码文件

在编译命令的编写中，需要指定编码格式。在代码分析中我们已确定使用 GB2312 编码，那么 gcc 指令应该这样写：

```
gcc -finput-charset=GB2312 -fexec-charset=GB2312 -o OUTPUT_NAME
INPUT_NAME.c
```

其中`-finput-charset`是源码的编码，而`-fexec-charset`是执行程序的编码。

虽然这样可以正确读取并输出，但我们忽略了终端程序（SSH）的编码格式，导致输出的■符号不能正确显示，如下图。

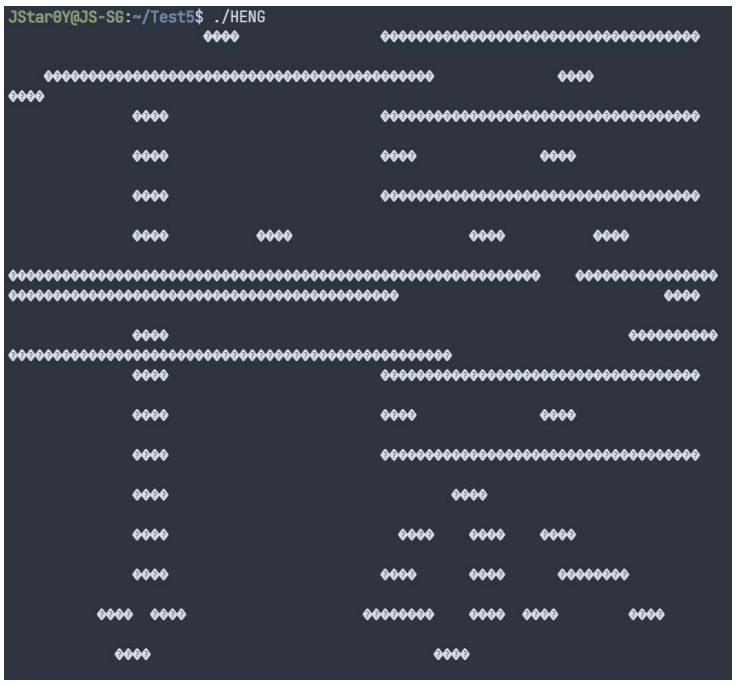

图 2.3-2 “■”字符显示错误

在这里，我们将终端字符集改为 GB18030（是 GBK，GB2312 标准的升级版），即可正

确显示输出结果。

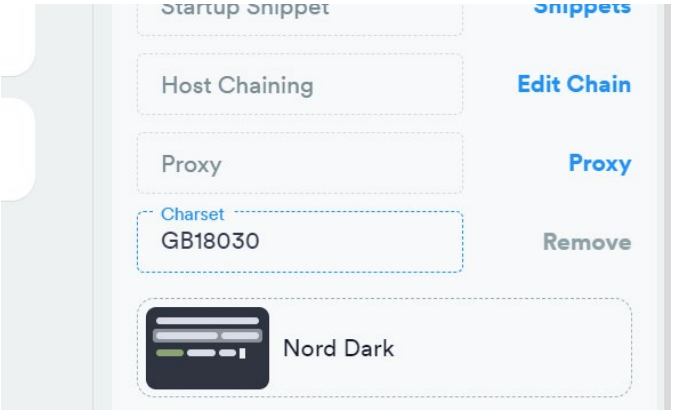

图 2.3-3 修改终端字符集

正确的输出结果如下图。

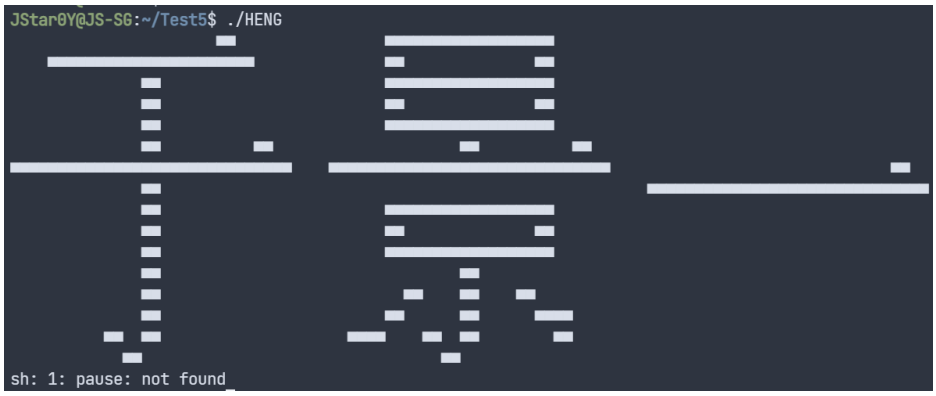

图 2.3-4 正确的输出

我们发现，在 Linux 下并不存在 pause 这个命令，意味着 `system(pause)`失效了。我们给出修改方法，这种方法是多端都适用的：

```
printf("Press any key to continue . . .");
getchar();
```

以下是修改并编译后的程序运行截图。

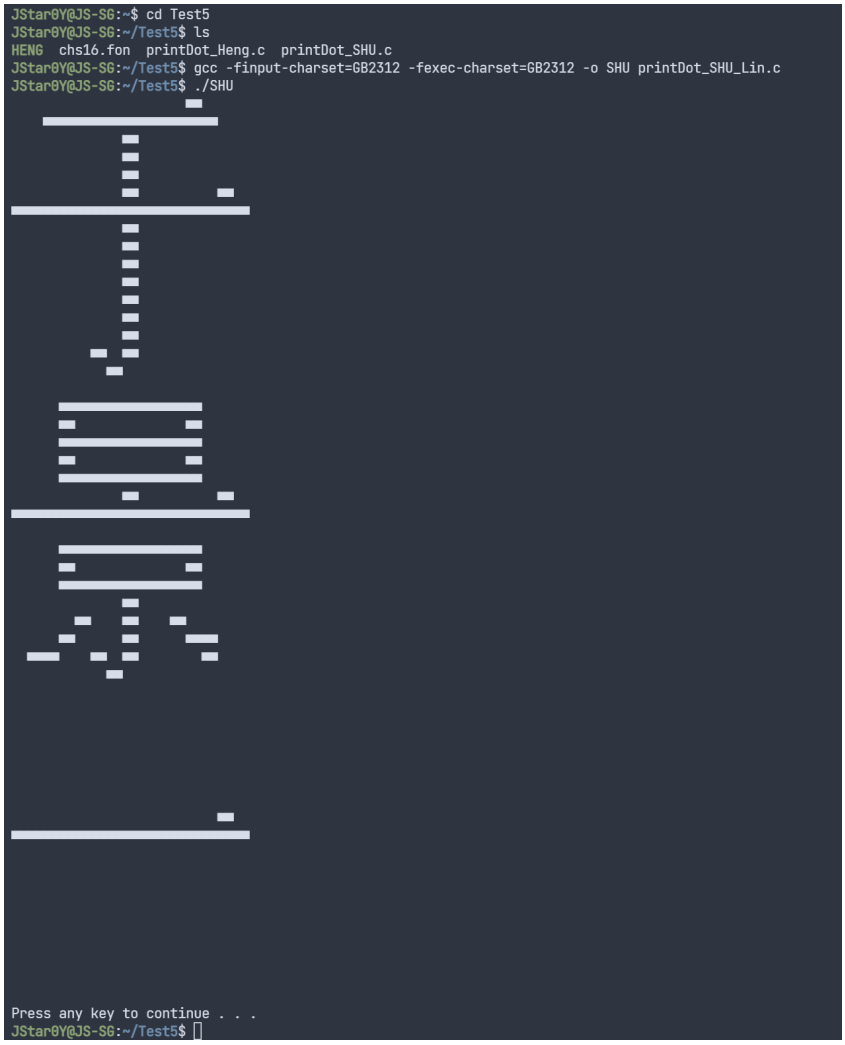

图 2.3-5-1 竖排输出结果

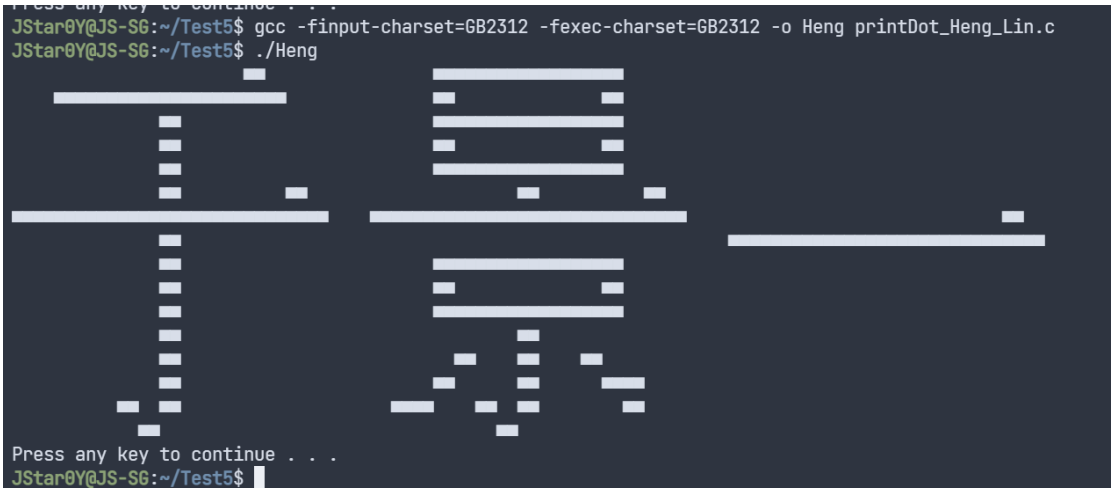

图 2.3-5-2 横排输出结果

3. 问题解决

我的解决方案写在文中了，在此一一列出，您可以用 Ctrl+左键跳转到对应位置。

① 如何配置 Visual Studio 中的 C 支持

② 如何将文件上传到服务器（SFTP 与 FTP）

4. 程序源码

4.1. 竖排输出的一种实现方式

```
#include "stdio.h"
#include "string.h"
#include "stdlib.h"

const int reglen = 94;       //每区(行)有 94 位(列)
const int font_width  = 16;  // 单字点阵宽度(列数)
const int font_height = 16;  // 单字点阵高度(行数)
const int dotsize = 16 * 16/8;   //一个汉字点阵所占的字节数
const int subcode = 0xa0;      //内码与区、位码的差值

char *font_file_name = "chs16.fon"; // 点阵字库文件名
char str[3][4] = {"于", "景", "一"};     //要显示点阵信息的汉字
char binlib[3][16*16/8] = {0};  //定义二位数组取代 bindot，存储点阵信息

void printcharbindot(char* bindot, int dotlen);
void readbindot(int q);

int main(int argc, char* argv[])
{
    for(int w=0; w<3; w++) {
        readbindot(w);
        //输出其点阵信息
        printcharbindot(binlib[w], dotsize);
        printf("\n");   //单字间换行
    }
```

```
    system("pause");
    return 0;
}

void readbindot(int q) {
    FILE *fp = fopen(font_file_name, "rb");
    int string_size = font_width * font_height;

    int i=0,j=0;
    int offset=0;
    unsigned char regcode;  //区码
    unsigned char bitcode;  //位码

    // 计算区位码
    regcode = (unsigned char)str[q][i]-subcode;
    bitcode = (unsigned char)str[q][i+1]-subcode;

//    再计算汉字在字库中的位置，进而得出此字符点阵在字体文件中的偏移
    offset=((regcode-1)*reglen+bitcode-1)*dotsize;

    // 在字库文件中读取其点阵数据
    fseek(fp, offset, SEEK_SET);
    fread(binlib[q], 32, 1, fp);

    fclose(fp);
}

//按顺序输出点阵的每一位信息
void printcharbindot(char* bindot, int len)
{

    int charnum = 0;  //当前字节号
    int bitnum  = 0;  //已读取的位数
    int bitindex = 0;  //当前位号
    int bitvalue;     //当前位的值
    for (charnum = 0; charnum < len; ++charnum)
    {
        //从高到低顺次输出一个字节的每位信息
        for(bitindex = 7; bitindex>= 0; --bitindex)
        {
            //输出当前字节第 bitindex 位的值
            bitvalue = ((bindot[charnum]>>bitindex) & 0x1 );
            if(bitvalue==1) printf("■■");
            else printf("  ");
```

```
            //满 16 位输出一行
            if ((++bitnum %16) == 0)
                printf("\n");
        }
    }
}
```

4.2. 横排输出的一种实现方式

```
#include "stdio.h"
#include "string.h"
#include "stdlib.h"

const int reglen = 94;       //每区(行)有 94 位(列)
const int font_width  = 16;  // 单字点阵宽度(列数)
const int font_height = 16;  // 单字点阵高度(行数)
const int dotsize = 16 * 16/8;   //一个汉字点阵所占的字节数
const int subcode = 0xa0;      //内码与区、位码的差值

char *font_file_name = "chs16.fon"; // 点阵字库文件名
char str[3][4] = {"于", "景", "一"};    //要显示点阵信息的汉字
char binlib[3][16*16/8] = {0};  //定义二位数组取代 bindot，存储点阵信息

void printcharbindot(int dotlen);
void readbindot(int q);

int main(int argc, char* argv[])
{
    for(int w=0; w<3; w++) readbindot(w);
    //输出其点阵信息
    printcharbindot(dotsize);

    system("pause");
    return 0;
}

void readbindot(int q) {
    FILE *fp = fopen(font_file_name, "rb");
    int string_size = font_width * font_height;

    int i=0,j=0;
    int offset=0;
    unsigned char regcode;  //区码
    unsigned char bitcode;  //位码
```

```
    // 计算区位码
    regcode = (unsigned char)str[q][i]-subcode;
    bitcode = (unsigned char)str[q][i+1]-subcode;

//    再计算汉字在字库中的位置，进而得出此字符点阵在字体文件中的偏移
    offset=((regcode-1)*reglen+bitcode-1)*dotsize;

    // 在字库文件中读取其点阵数据
    fseek(fp, offset, SEEK_SET);
    fread(binlib[q], 32, 1, fp);

    fclose(fp);
}

//按顺序输出点阵的每一位信息
void printcharbindot(int len)
{

    int charnum = 0;  //当前字节号
    int bitnum  = 0;  //已读取的位数
    int bitindex = 0;  //当前位号
    int bitvalue;     //当前位的值
    for (charnum = 0; charnum < len;) //在这里 charnum 只用来确定是否输出了三个字的每个字节
    {
        for(int n = 0; n < 3; n++)  //共输出三次，每次对应一个汉字
        {
            //从高到低顺次输出一个字节的每位信息
            for(int m = 0; m < 2; m++)  //一个汉字每行需要输出两次“8 个二进制位”
            {
                for(bitindex = 7; bitindex>=0; --bitindex)  //遍历一个字节的每一位
                {
                    //输出当前字节第 bitindex 位的值
                    bitvalue = ((binlib[n][charnum]>>bitindex) & 0x1 );
                    if(bitvalue==1) printf("■■");
                    else printf("  ");
                }
                charnum++; //暂时提升到第二个字节
            }
            charnum-=2; //清除暂时提升，返回第一个字节
            printf("  "); //满 16 位（两个字节）输出两个空格
```

```
        }
        charnum += 2;  //每行三个字都遍历完两个字节后，提升到下一行的第一个字节
        printf("\n"); //输出过三个字后换行
    }
}
```

对 Linux 平台，特别将 system(pause)修改为多端都适用的方法：

```
printf("Press any key to continue . . .");
getchar();
```