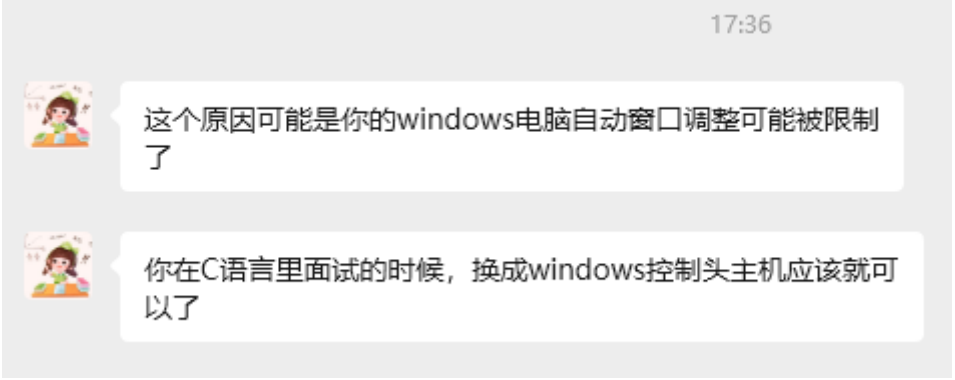
Win32 API

平常我们运行起来的黑框程序其实就是控制台程序

我们可以使用cmd命令来设置控制台窗口的长宽：设置控制台窗口的大小，30行，100列

```
mode con cols=100 lines=30
```

> 出师不利，刚开始就卡住了，cmd不能使用API命令？
>
> 
>
>
> 先接着往下学吧，为什么别人电脑可以。

解决win11 mode con无效的过程

① 打开“真正的”旧版命令窗口

1. 同时按键盘上的 `Win + R`（Win 就是左下角窗户图标键）。
2. 弹出的小方框里输入

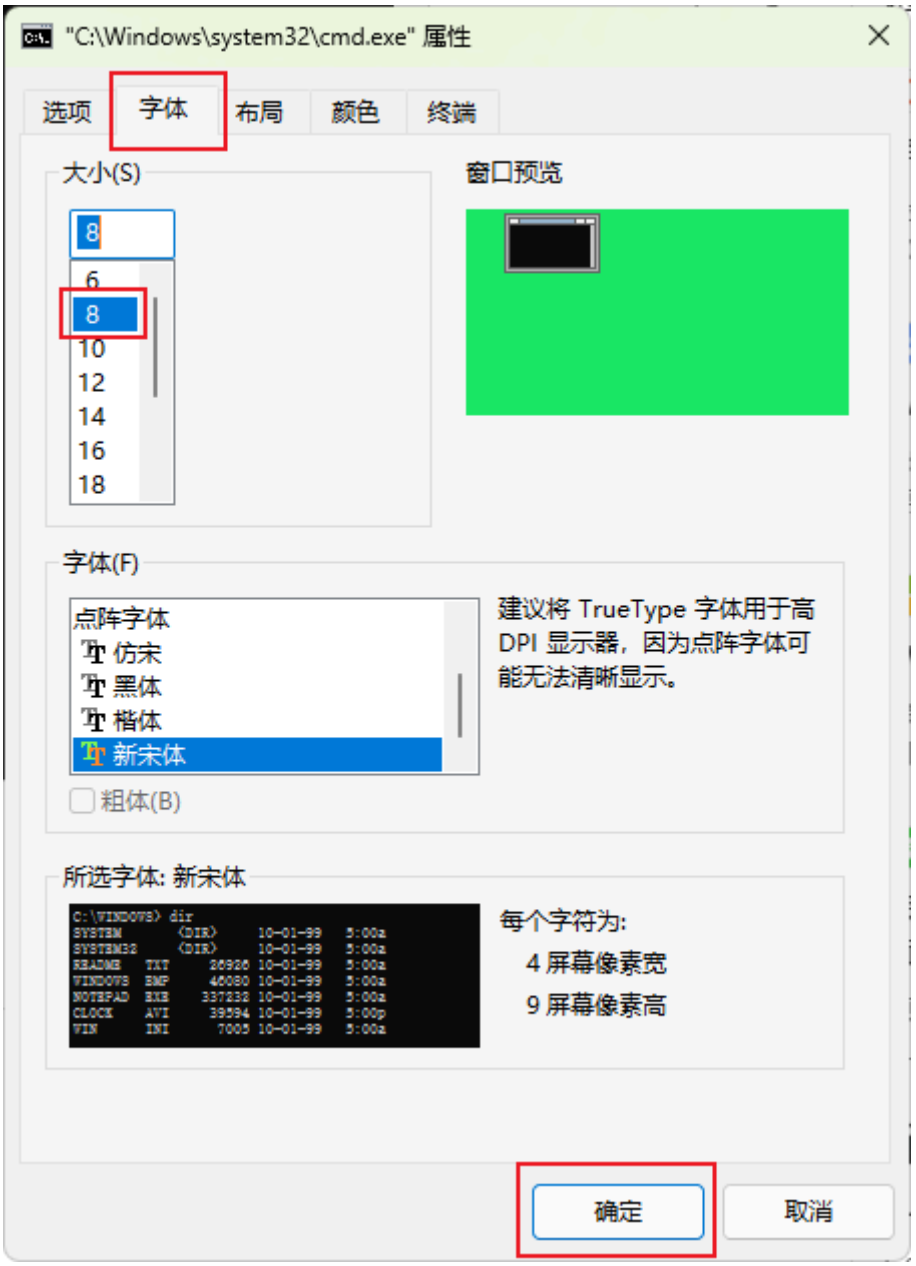
   ```
   cmd
   ```

3. 立刻同时按 `Ctrl + Shift + Enter`（**注意不是直接回车**）。
   如果系统问“是否允许更改”，点“是”。
   现在你看到的就是**纯黑色全屏 cmd**，不是 Windows Terminal。

② 先把字体调到最小（防止太大装不下）

1. 在黑色窗口**最上面蓝色条→右键→选“属性”**。
2. 上方点“**字体**”。
3. 左边选“**8×12**”或“**点阵 8**”→点“**确定**”。

> 这里还是选18吧，太小了看不清啊。



③ 把“缓冲区”和“窗口”一起改成 100×30

1. 还在刚才的“属性”窗口→上方点“**布局**”。
2. 会看到两行数字：
   - **屏幕缓冲区大小** → 宽、高
   - **窗口大小** → 宽、高
3. 把这两行**全部改成**：
   - **宽度 100**
   - **高度 30**

点右下角“**确定**”→关闭窗口。

> 你看到的灰色，是因为 Windows 在你**手动拖过窗口边框**后，会自动把“由系统定位窗口”取消掉，并且把“窗口位置”锁死，导致**缓冲区宽度**那一栏也连带变灰。
> 只要**重新勾选**“由系统定位窗口”，所有数字会立刻变白，就能改了。

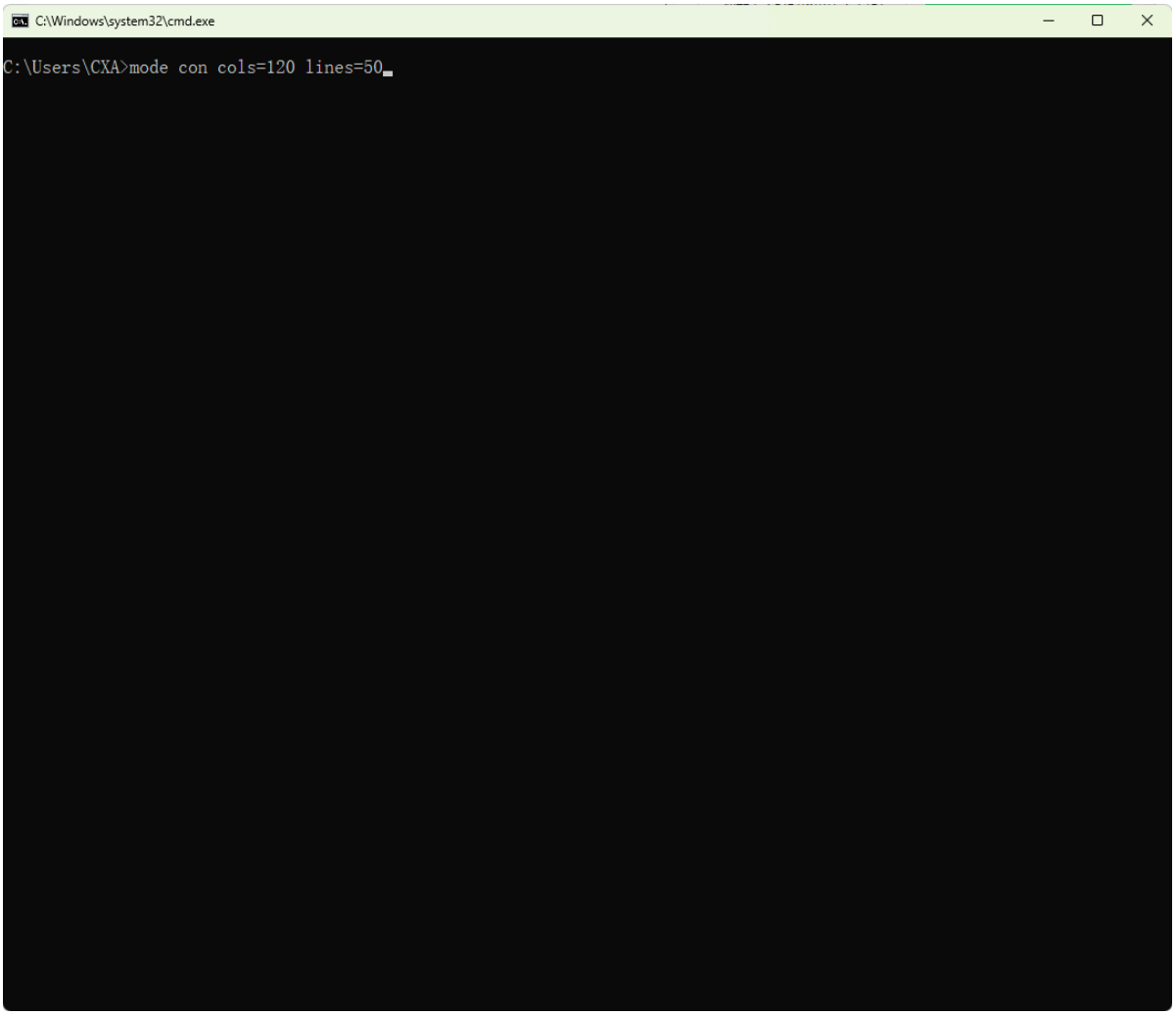
窗口位置不需要改！
只要勾了 **“由系统定位窗口”**，Windows 会自动把窗口放到屏幕左上角附近，**每次打开位置都一致**，省心又整齐。

- **左边(L)**、**上边(T)** 那两个数字保持灰色就行；
- 如果你把勾去掉，数字会变白，这时手动填了坐标后，窗口就**永远钉死**在那个像素点，换显示器或多屏时可能飞出可视区，反而麻烦。

所以：**让“由系统定位窗口”打着勾，窗口位置一律不动**，最省事！

现在就修改的设置就好了（也搞不太清楚）

验证



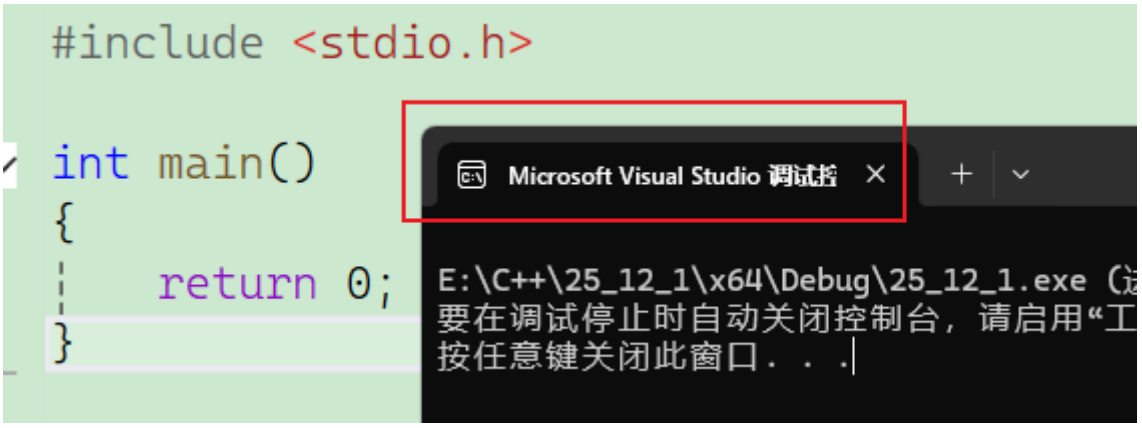
OK，现在就解决了。

Win 32 API介绍

`Windows` 这个多作业系统除了协调应用程序的执行、分配内存、管理资源之外， 它同时也是一个很大的服务中心，调用这个服务中心的各种服务（**每一种服务就是一个函数**），可以帮应用程序达到开启视窗、描绘图形、使用周边设备等目的，由于这些函数服务的对象是应用程序(`Application`)， 所以便称之为 `Application Programming Interface`，简称 `API` 函数。`WIN32 API` 也就是 `Microsoft Windows`

`32位平台` 的应用程序编程接口。

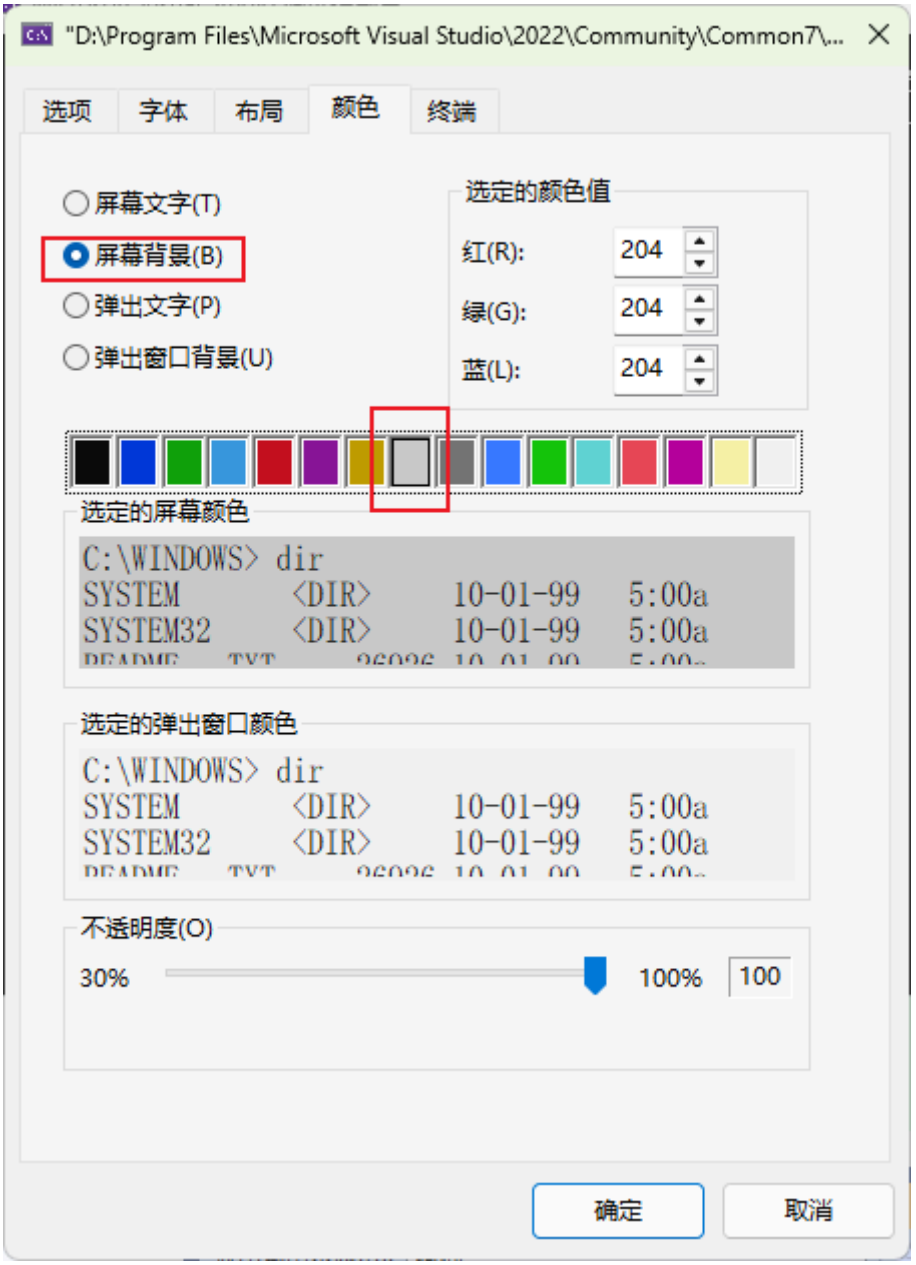
控制台设置



这里呢，如果代码运行后是这个界面，那么就代表代码运行是在终端上，这时候需要进行设置一下，

修改成让`Windows`决定。

也可以直接选控制台主机。



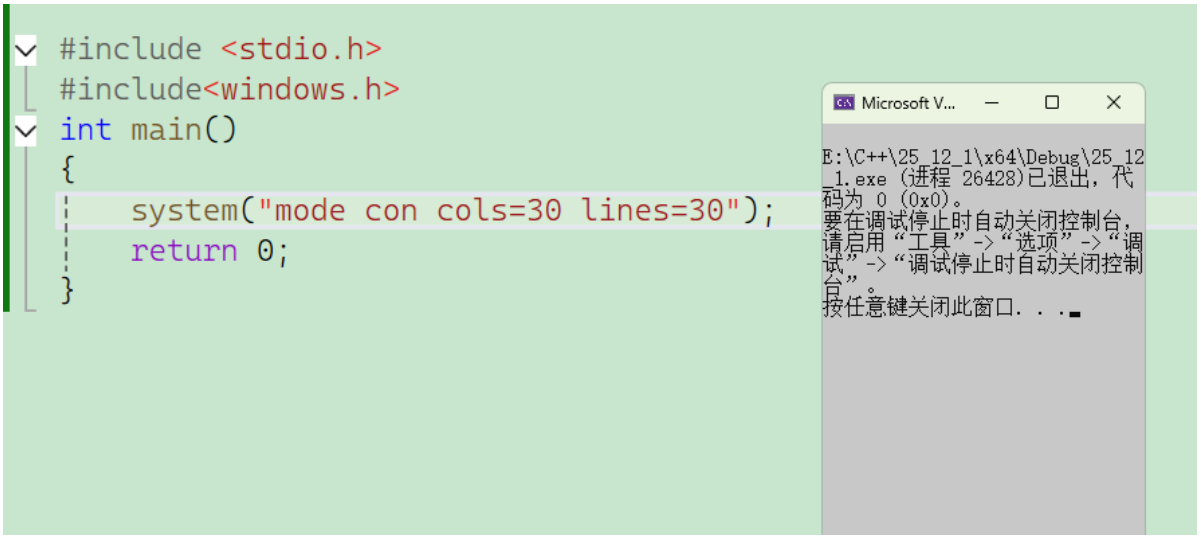
吧背景颜色修改一下，便于区分。

正文开始

程序中设置控制台

`system`函数用来执行系统命令。

```
system("mode con cols=30 lines=30");
```



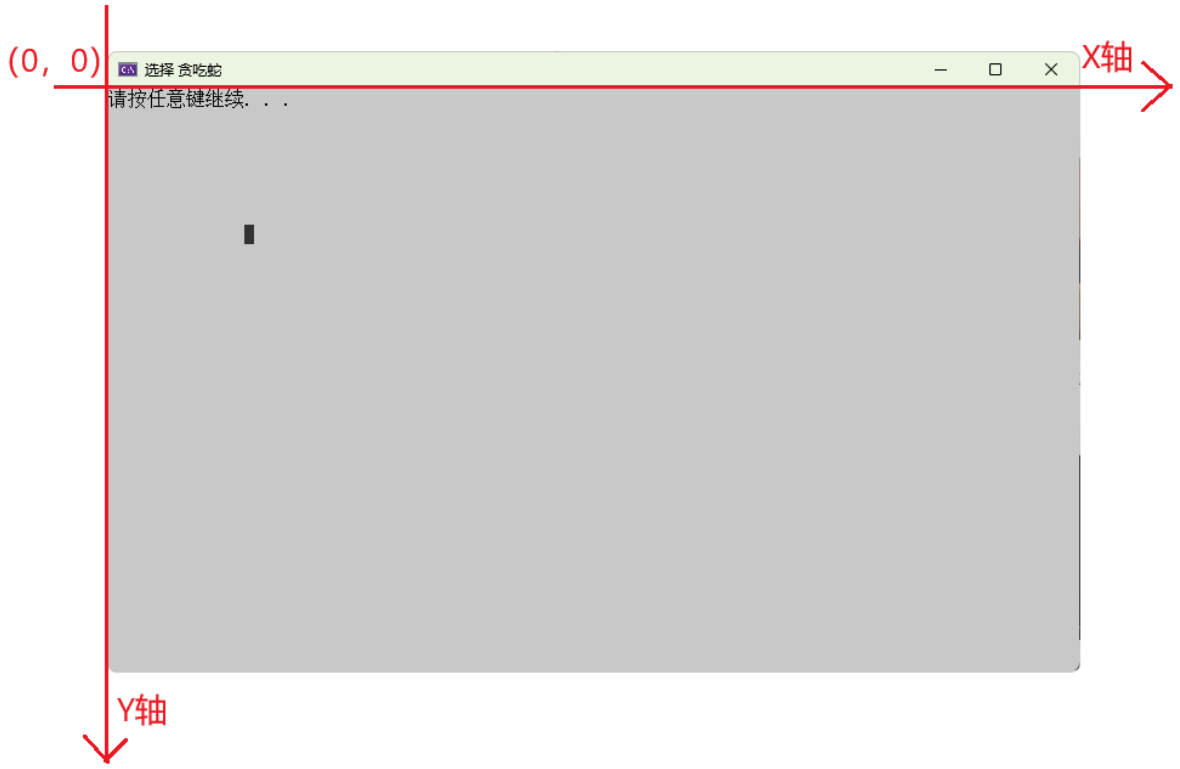

```c
#include <stdio.h>
#include<windows.h>
int main()
{
    //设置控制台窗口大小
    system("mode con cols=100 lines=30");
    //设置控制台名称
    system("title 贪吃蛇");


    //设置控制台暂停 获取一个字符后继续
    system("pause");
    return 0;
}
```

COORD控制台屏幕坐标

COORD 是Windows API中定义的一个结构体，表示一个字符在控制台屏幕上的坐标

```c
typedef struct _COORD {
  SHORT X;
  SHORT Y;
} COORD, *PCOORD;
```



屏幕上任何一个位置就是一个坐标。

怎么定义一个坐标呢?

```
COORD pos = { 0,0 };
COORD pos = { 10,10 };
```

GetStdHandle 函数

要控制控制台窗口，首先要获得这个窗口。

`GetStdHandle` 是一个**Windows API函数。**它用于从一个特定的标准设备（标准输入、标准输出或标准错误）中取得一个**句柄**（用来标识不同设备的数值），使用这个句柄可以操作设备（也就是屏幕）。

> 句柄 其实是一个数字，只是叫句柄而已，简单理解为抓手。

函数定义（飞机票）

```
HANDLE WINAPI GetStdHandle(
  _In_ DWORD nStdHandle
);
```

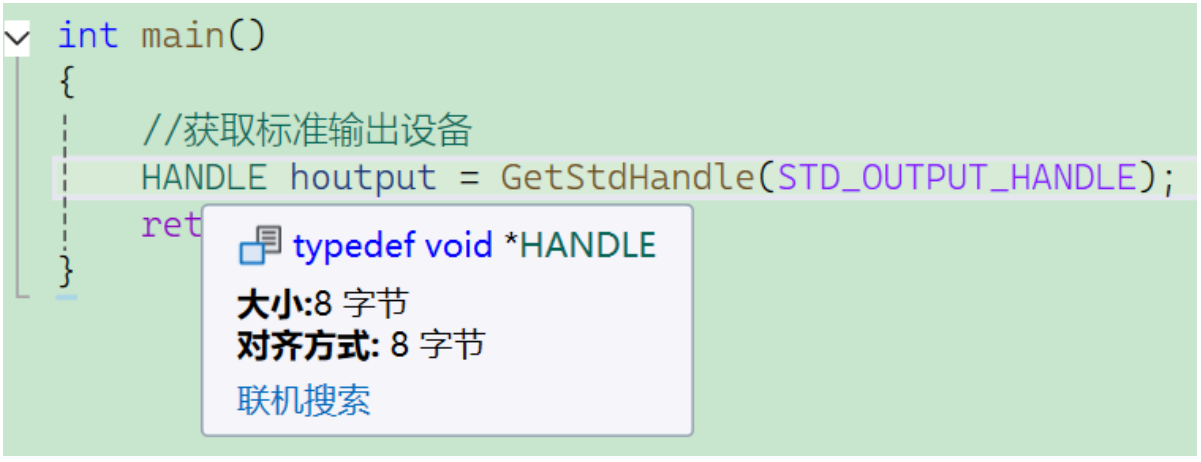
展开表

值	含义
STD_INPUT_HANDLE `((DWORD)-10)`	标准输入设备。 最初，这是输入缓冲区 `CONIN$` 的控制台。
STD_OUTPUT_HANDLE `((DWORD)-11)`	标准输出设备。 最初，这是活动控制台屏幕缓冲区 `CONOUT$`。
STD_ERROR_HANDLE `((DWORD)-12)`	标准错误设备。 最初，这是活动控制台屏幕缓冲区 `CONOUT$`。

注意

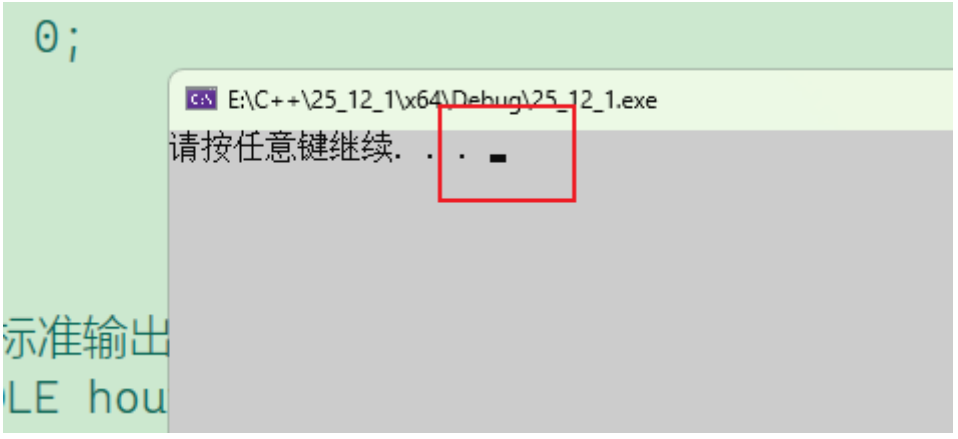
这些常量的值都是无符号数，但是在头文件中被定义为从有符号数的强制转换，并利用 C 编译器将它们滚动到刚好低于最大 32 位值。 在以不分析标头并重新定义常量的语言与这些句柄交互时，请注意此约束。 例如，`((DWORD)-10)` 实际上是无符号数 `4294967286`。

HANDLE是一种空指针变量。



GetConsoleCursorInfo 函数（飞机票）

这里有光标一直在闪，如果贪吃蛇一直在跑的话，光标一直闪非常不方便，我们可以把光标隐藏起来。



检索有关指定控制台屏幕缓冲区的光标大小和可见性的信息。

```
BOOL WINAPI GetConsoleCursorInfo(
  _In_  HANDLE               hConsoleOutput,
  _Out_ PCONSOLE_CURSOR_INFO lpConsoleCursorInfo
);
```

它的第一个参数就是刚才获取到的句柄。

第二个参数是指向 CONSOLE_CURSOR_INFO 结构的指针，该结构接收有关控制台游标的信息。

CONSOLE_CURSOR_INFO 结构

```
typedef struct _CONSOLE_CURSOR_INFO {
  DWORD dwSize;
  BOOL  bVisible;
} CONSOLE_CURSOR_INFO, *PCONSOLE_CURSOR_INFO;
```

- dwSize，由光标填充的字符单元格的百分比。 此值介于1到100之间。 光标外观会变化，范围从完全填充单元格到单元底部的水平线条。
- bVisible，游标的可见性。 如果光标可见，则此成员为 TRUE。

示例代码：

```
int main()
{
    //获取标准输出设备
    HANDLE houtput = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cursor_info = { 0 };

    GetConsoleCursorInfo(houtput, &cursor_info);

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

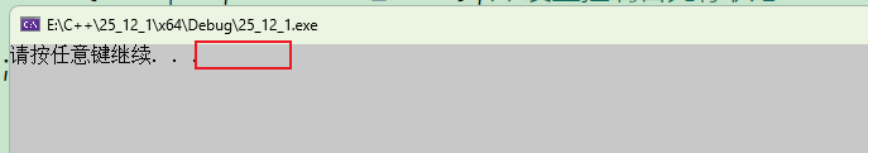
这里的25表示看见的光标，只占到一个字符的25%。也可以直接隐藏，那就需要设置结构体另一个成员。

> 这里需要注意的是，不能只修改，不然将会不成功，而是设置 ---> 修改 ---> 再设置。

```
int main()
{
    //获取标准输出设备
    HANDLE houtput = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cursor_info = { 0 };

    GetConsoleCursorInfo(houtput, &cursor_info);
    //printf("%d\n",cursor_info.dwSize);
    //cursor_info.dwSize = 50;
    cursor_info.bVisible = false; //隐藏控制台光标
    SetConsoleCursorInfo(houtput, &cursor_info);//设置控制台光标状态

    system("pause");
```



不是我没截，是真的没了。

SetConsoleCursorPosition 函数

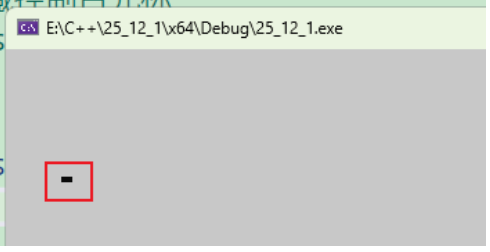
```
BOOL WINAPI SetConsoleCursorPosition(
  _In_ HANDLE hConsoleOutput,
  _In_ COORD  dwCursorPosition
);
```

设置指定控制台屏幕缓冲区中的光标位置，将想要设置的坐标信息放在COORD类型的pos中，用SetConsoleCursorPosition函数将光标位置设置到指定的位置。

指定新光标位置（以字符为单位）的 COORD 结构。 坐标是屏幕缓冲区字符单元的列和行。 **坐标必须位于控制台屏幕缓冲区的边界以内。**

```
////printf("%d\n",cursor_info.dwSize);
////cursor_info.dwSize = 50;
//cursor_info.bVisible = false; //隐藏控制台光标
//SetConsoleCursorInfo(houtput, &curs

COORD pos = { 5,5 };
SetConsoleCursorPosition(houtput, pos
```




```
COORD pos = { 5, 5};
HANDLE hOutput = NULL;
//获取标准输出的句柄(用来标识不同设备的数值)
hOutput = GetStdHandle(STD_OUTPUT_HANDLE);
//设置标准输出上光标的位置为pos
SetConsoleCursorPosition(hOutput, pos);
```

光标被定位到（5，5）。

设置光标位置函数set_pos

```
void set_pos(short x, short y)
{
    //拿到标准输出设备句柄
    HANDLE houtput = GetStdHandle(STD_OUTPUT_HANDLE);
    //定义光标位置
    COORD pos = { x,y };
    //设置光标位置
    SetConsoleCursorPosition(houtput, pos);
}
```

GetAsyncKeyState 函数

获取按键情况，GetAsyncKeyState的函数原型如下：

```
SHORT GetAsyncKeyState( int vKey );
```

将键盘上每个键的虚拟键值传递给函数，函数通过返回值来分辨按键的状态。

6	0x36	6 key
7	0x37	7 key
8	0x38	8 key
9	0x39	9 key
	0x3A-40	Undefined
A	0x41	A key
B	0x42	B key
C	0x43	C key
D	0x44	D key
E	0x45	E key
F	0x46	F key
G	0x47	G key
H	0x48	H key
I	0x49	I key
J	0x4A	J key
K	0x4B	K key

`GetAsyncKeyState` 的返回值是short类型，在上一次调用 `GetAsyncKeyState` 函数后，如果返回的16位的short数据中，最高位是1，说明按键的状态是按下，如果最高是0，说明按键的状态是抬起；如果最低位被置为1则说明，该按键被按过，否则为0。

如果我们要判断一个键是否被按过，可以检测`GetAsyncKeyState`返回值的最低值是否为1.

```
#define KEY_PRESS(VK) ( (GetAsyncKeyState(VK) & 0x1) ? 1 : 0 )
```

```
int main()
{
    while (1)
    {
        if (KEY_PRESS(0x30))
            printf("0\n");
        else if(KEY_PRESS(0x31))
            printf("1\n");
        else if (KEY_PRESS(0x32))
```

```
            printf("2\n");
        else if (KEY_PRESS(0x33))
            printf("3\n");
        else if (KEY_PRESS(0x34))
            printf("4\n");
        else if (KEY_PRESS(0x35))
            printf("5\n");
        else if (KEY_PRESS(0x36))
            printf("6\n");
        else if (KEY_PRESS(0x37))
            printf("7\n");
        else if (KEY_PRESS(0x38))
            printf("8\n");
        else if (KEY_PRESS(0x39))
            printf("9\n");
        else if (KEY_PRESS(0x26))
            printf("上\n");
        else if (KEY_PRESS(0x25))
            printf("左\n");
        else if (KEY_PRESS(0x27))
            printf("右\n");
        else if (KEY_PRESS(0x28))
            printf("下\n");
    }
    return 0;
}
```

按键检测。
