在控制台上编写贪吃蛇游戏，需要调用window API，下面介绍一下主要的：

## 5.2 控制台程序

平常我们运⾏起来的⿊框程序其实就是控制台程序，能在控制台窗⼝执⾏的命令，也可以调⽤C语⾔函数system来执⾏。

### 5.2.1 [mode](https://learn.microsoft.com/zh-cn/windows-server/administration/windows-commands/mode)命令

我们可以用mode命令来设置控制台窗口的长和宽

1. *//设置控制台窗⼝的⻓宽：设置控制台窗⼝的⼤⼩，30⾏，100列*
2. system("mode con cols=100 lines=30");

### 5.2.2 [title](https://learn.microsoft.com/zh-cn/windows-server/administration/windows-commands/title)命令

用来设置控制台窗口的名字

1. 设置cmd窗⼝名称
2. system("title 贪吃蛇");

### 5.3.3 pause命令

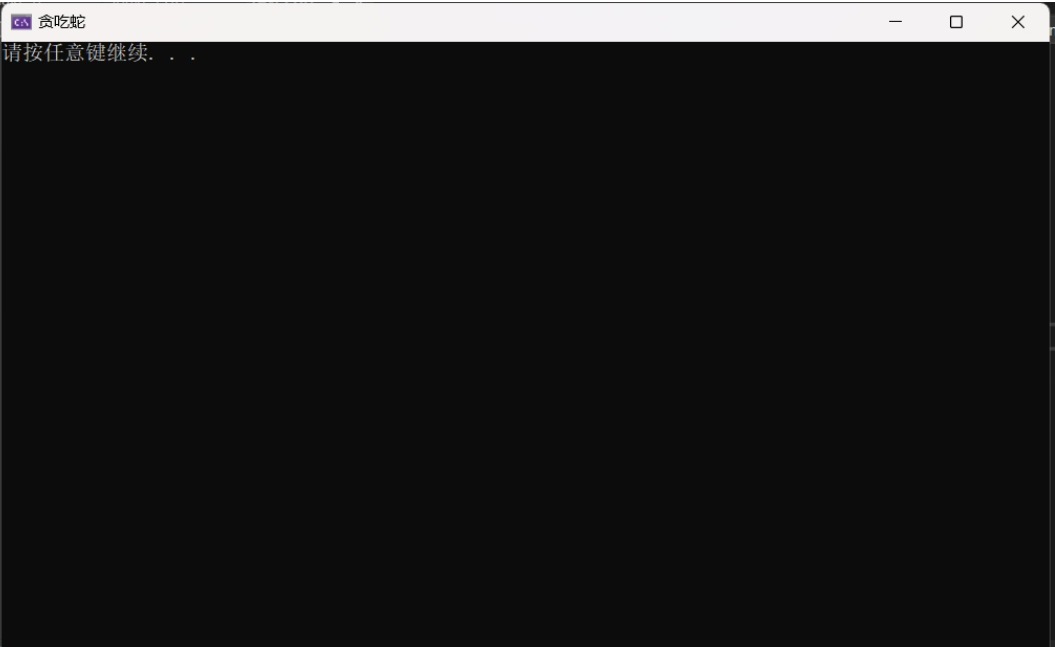
在 Windows 系统下经常使用的一个技巧，主要用于在控制台应用程序中暂停程序的执行，以便用户可以看到程序的输出，并有机会查看结果。

1. *//暂停程序的运行*
2. system("pause");

代码示例：

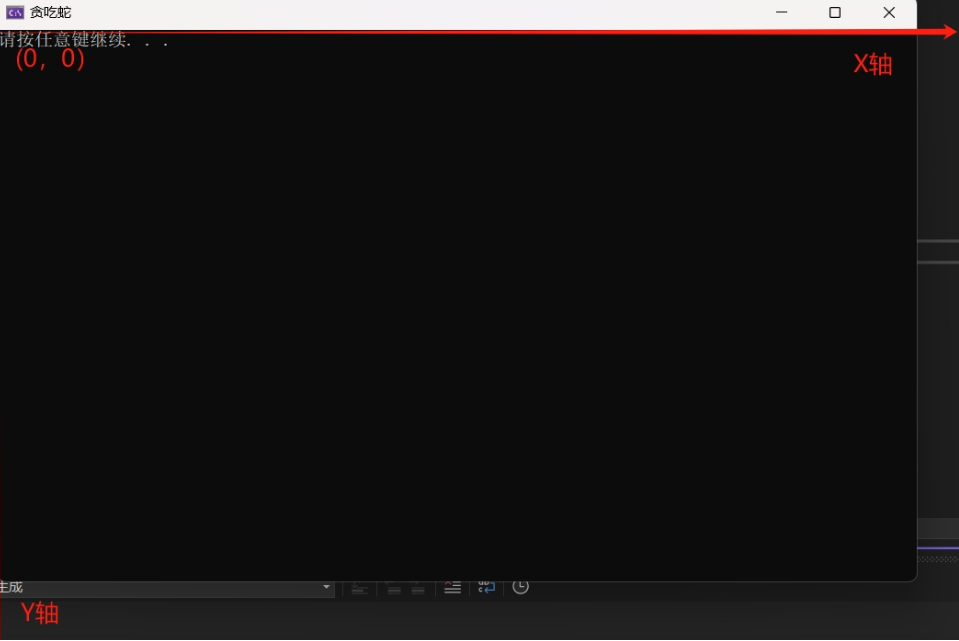
1. #include <stdio.h>
2. int main()
3. {
4. *//设置控制台窗⼝的⻓宽：设置控制台窗⼝的⼤⼩，30⾏，100列*
5. system("mode con cols=100 lines=30");
6. *//设置cmd窗⼝名称*
7. system("title 贪吃蛇");
8. *//暂停程序的运行*
9. system("pause");
10. return 0;
11. }

运行结果如下：



## 5.3 控制台屏幕上的坐标[COORD](https://learn.microsoft.com/zh-cn/windows/console/coord-str)

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幕缓冲区上的坐标，坐标系 (0，0) 的原点位于缓冲区的顶部左侧单元格。



COORD类型的声明：

1. typedef struct \_COORD {
2. SHORT X;
3. SHORT Y;
4. } COORD, \*PCOORD

 给坐标赋值

1. *//pos设置为屏幕上的一个点，其中X=10，Y=15*
2. COORD pos = { 10, 15 };

## 5.4 [GetStdHandle](https://learn.microsoft.com/zh-cn/windows/console/getstdhandle)函数

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**获取⼀个句柄，来操作控制台窗口**

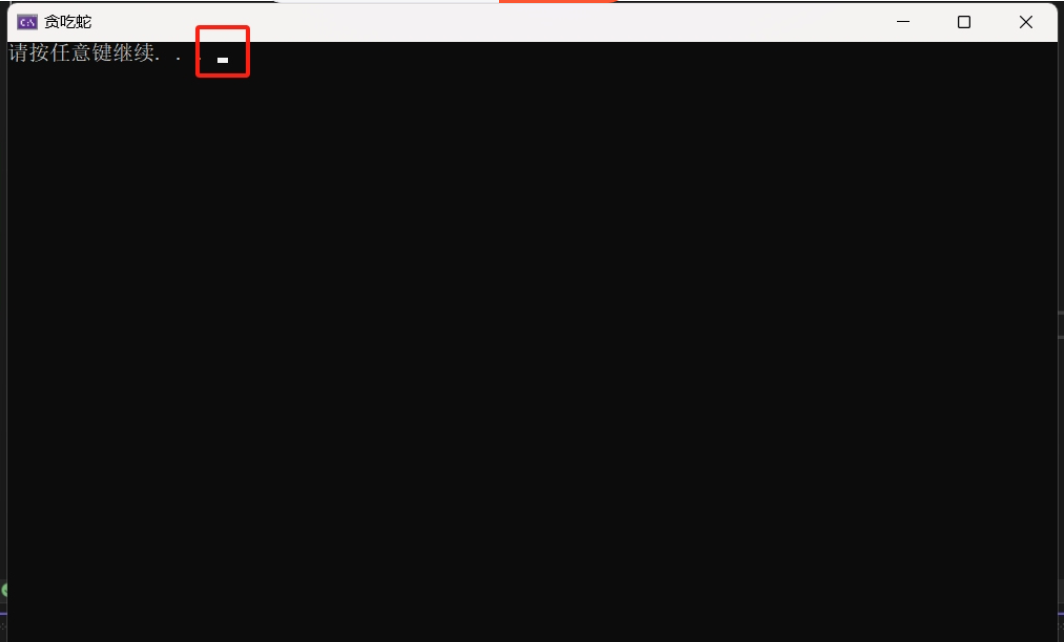
HANDLE是一个类型名，是一个**void\*类型的指针**

1. HANDLE houtput = NULL;
2. *//获取标准输出的句柄(⽤来标识不同设备的数值)*
3. houtput = GetStdHandle(STD\_OUTPUT\_HANDLE);

## 5.5 [GetConsoleCursorInfo](https://learn.microsoft.com/zh-cn/windows/console/getconsolecursorinfo)函数

GetConsoleCursorInfo函数用来**获得有关指定控制台屏幕缓冲区的光标大小和可见性的信息句柄**。

在我们运行程序的时候，控制台窗口会出现光标在闪烁：



 如果运行贪吃蛇游戏一直有一个光标在闪烁就很不美观，GetConsoleCursorInfo就是用来设置光标的大小和可见性的

1. BOOL WINAPI GetConsoleCursorInfo(
2. HANDLE hConsoleOutput,
3. PCONSOLE\_CURSOR\_INFO lpConsoleCursorInfo
4. )
5. PCONSOLE\_CURSOR\_INFO 是指向 CONSOLE\_CURSOR\_INFO 结构的指针

代码示例如下：

1. HANDLE hOutput = NULL;
2. *//获取标准输出的句柄(⽤来标识不同设备的数值)*
3. hOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);
4. CONSOLE\_CURSOR\_INFO CursorInfo;
5. GetConsoleCursorInfo(hOutput, &CursorInfo);*//获取控制台光标信息*

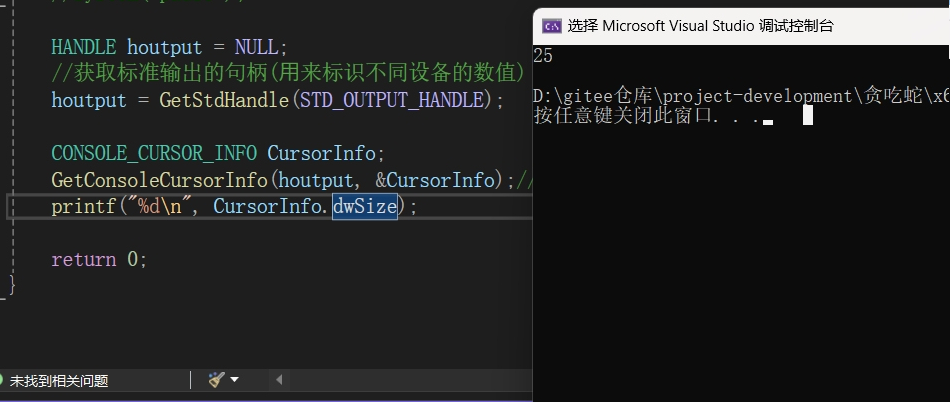
### 5.5.1 [CONSOLE\_CURSOR\_INFO](https://learn.microsoft.com/zh-cn/windows/console/console-cursor-info-str)结构体

这个结构体，包含有关控制台光标的信息

1. typedef struct \_CONSOLE\_CURSOR\_INFO {
2. DWORD dwSize;
3. BOOL bVisible;
4. } CONSOLE\_CURSOR\_INFO, \*PCONSOLE\_CURSOR\_INFO;

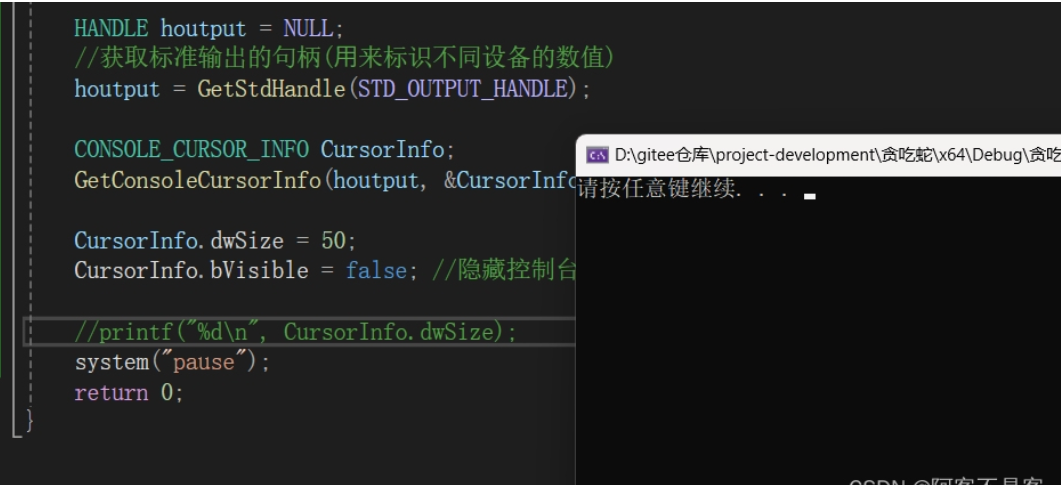
dwSize，由光标填充的字符单元格的百分⽐。 此值介于1到100之间。 光标外观会变化，范围从完全填充单元格到单元底部的⽔平线条。  
bVisible，游标的可⻅性。 如果光标可⻅，则此成员为 TRUE。

 如下图所示：



光标的高度为正常一个字符高度的25%

我们获得了光标的句柄信息，我们想要修改该怎么办呢？



我们发现光标高度并没有改变也没有隐藏，那该怎么修改呢，我们需要用到SetConsoleCursorInfo函数了。

## 5.6[SetConsoleCursorInfo](https://learn.microsoft.com/zh-cn/windows/console/setconsolecursorinfo)函数

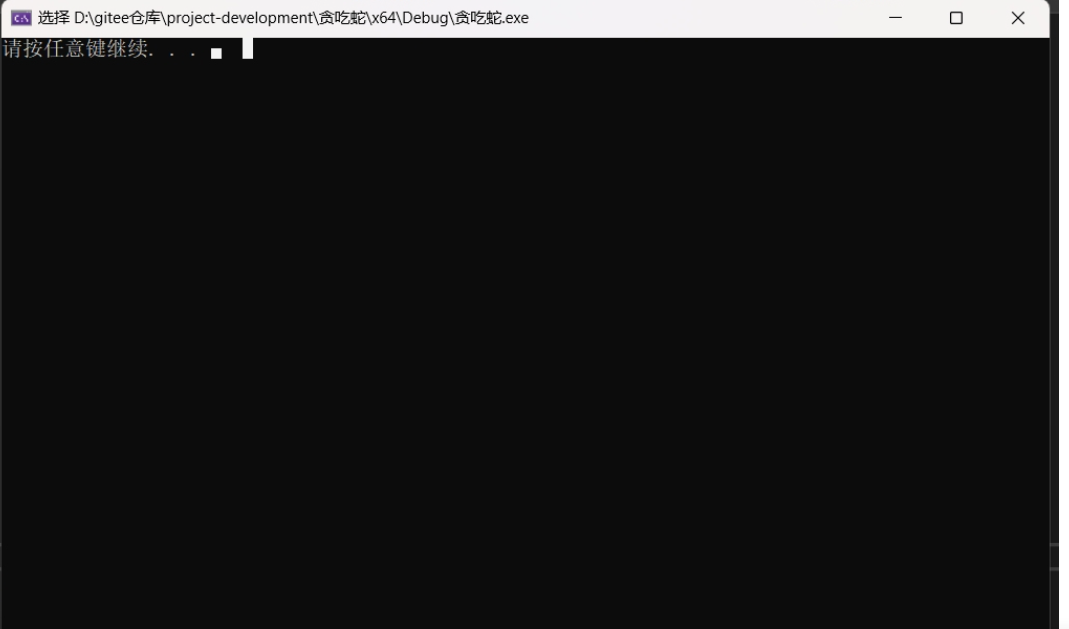
该函数用来设置指定控制台屏幕缓冲区的**光标的大小和可见性**。

1. BOOL WINAPI SetConsoleCursorInfo(
2. HANDLE hConsoleOutput,
3. const CONSOLE\_CURSOR\_INFO \*lpConsoleCursorInfo
4. );

代码示例如下：

1. HANDLE houtput = NULL;
2. *//获取标准输出的句柄(⽤来标识不同设备的数值)*
3. houtput = GetStdHandle(STD\_OUTPUT\_HANDLE);
5. *//定义一个光标信息*
6. CONSOLE\_CURSOR\_INFO CursorInfo = { 0 };
7. GetConsoleCursorInfo(houtput, &CursorInfo);*//获取控制台光标信息*
9. CursorInfo.dwSize = 50;
10. *//CursorInfo.bVisible = false; //隐藏控制台光标*
11. SetConsoleCursorInfo(houtput, &CursorInfo);

效果如下所示：



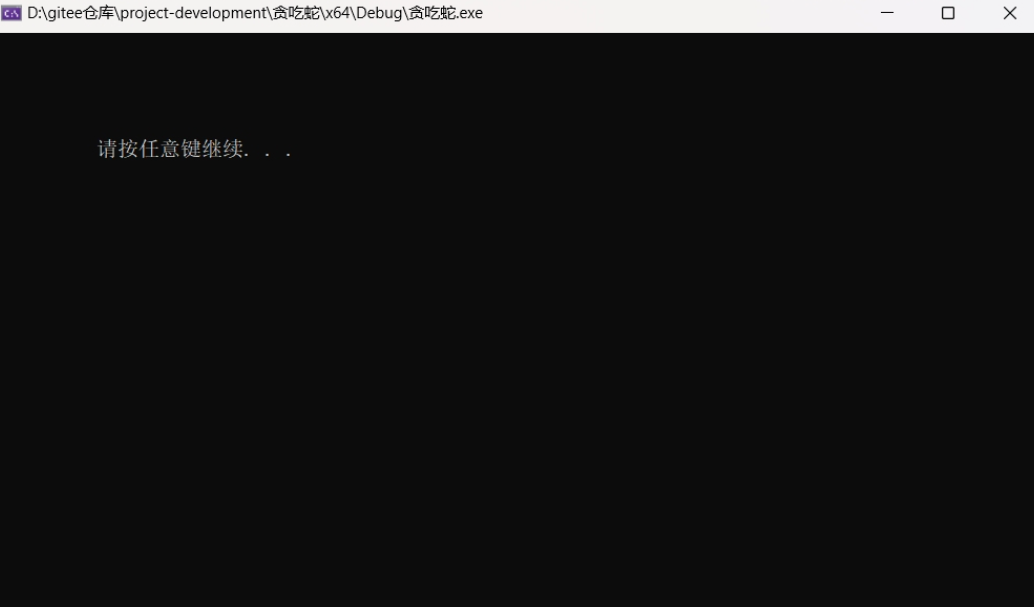
## 5.7 [SetConsoleCursorPosition](https://learn.microsoft.com/zh-cn/windows/console/setconsolecursorposition)函数

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

1. BOOL WINAPI SetConsoleCursorPosition(
2. HANDLE hConsoleOutput,
3. COORD pos
4. );

代码示例如下：

1. int main()
2. {
3. HANDLE houtput = NULL;
4. *//获取标准输出的句柄(⽤来标识不同设备的数值)*
5. houtput = GetStdHandle(STD\_OUTPUT\_HANDLE);
7. *//定位光标位置*
8. COORD pos = { 10, 5 };
9. SetConsoleCursorPosition(houtput, pos);
11. *//getchar();*
12. system("pause");
13. return 0;
14. }



但是如果我们需要多次调用光标出现在不同位置会很麻烦，因此我们可以封装一个函数来实现定位光标的操作。

### 5.7.1 set\_pos：封装一个设置光标位置得函数

1. void set\_pos(short x, short y)
2. {
3. COORD pos = { x, y };
4. HANDLE hOutput = NULL;
5. *//获取标准输出的句柄(⽤来标识不同设备的数值)*
6. hOutput = GetStdHandle(STD\_OUTPUT\_HANDLE);
7. *//设置标准输出上光标的位置为pos*
8. SetConsoleCursorPosition(hOutput, pos);
9. }

## 5.8 [GetAsyncKeyState](https://learn.microsoft.com/zh-cn/windows/win32/api/winuser/nf-winuser-getasynckeystate)函数

我们用通过键盘上的按键来操控贪吃蛇，但我们需要获取我们按了什么键。

我们可以通过GetAsyncKeyState函数，将键盘上每个键的**虚拟键值（**[**虚拟键码-Win32**](https://learn.microsoft.com/zh-cn/windows/win32/inputdev/virtual-key-codes)**）**传递给函数，函数通过返回值来分辨按键的状态。

函数返回一个short类型的值，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

 那我们如何判断最低为是否为1呢？

可以通过与0x01按位与（&）来进行判断

代码示例如下：

1. int main()
2. {
3. short ret = GetAsyncKeyState(0x50);
5. if ((ret & 1) == 1)
6. printf("5被按过\n");
7. else
8. printf("没有被按过\n");
10. return 0;
11. }

在多次调用的时候为了避免重复，选择用宏定义进行封装

1. *//结果是1表示按过，结果是0表示没按过*
2. #define KEY\_PRESS(vk) (GetAsyncKeyState(vk)&1)?1:0)

检测数字键：

1. int main()
2. {
3. while (1)
4. {
5. if (KEY\_PRESS(0x30))
6. {
7. printf("0\n");
8. }
9. else if (KEY\_PRESS(0x31))
10. {
11. printf("1\n");
12. }
13. else if (KEY\_PRESS(0x32))
14. {
15. printf("2\n");
16. }
17. else if (KEY\_PRESS(0x33))
18. {
19. printf("3\n");
20. }
21. else if (KEY\_PRESS(0x34))
22. {
23. printf("4\n");
24. }
25. else if (KEY\_PRESS(0x35))
26. {
27. printf("5\n");
28. }
29. else if (KEY\_PRESS(0x36))
30. {
31. printf("6\n");
32. }
33. else if (KEY\_PRESS(0x37))
34. {
35. printf("7\n");
36. }
37. else if (KEY\_PRESS(0x38))
38. {
39. printf("8\n");
40. }
41. else if (KEY\_PRESS(0x39))
42. {
43. printf("9\n");
44. }
45. }
46. return 0;
47. }