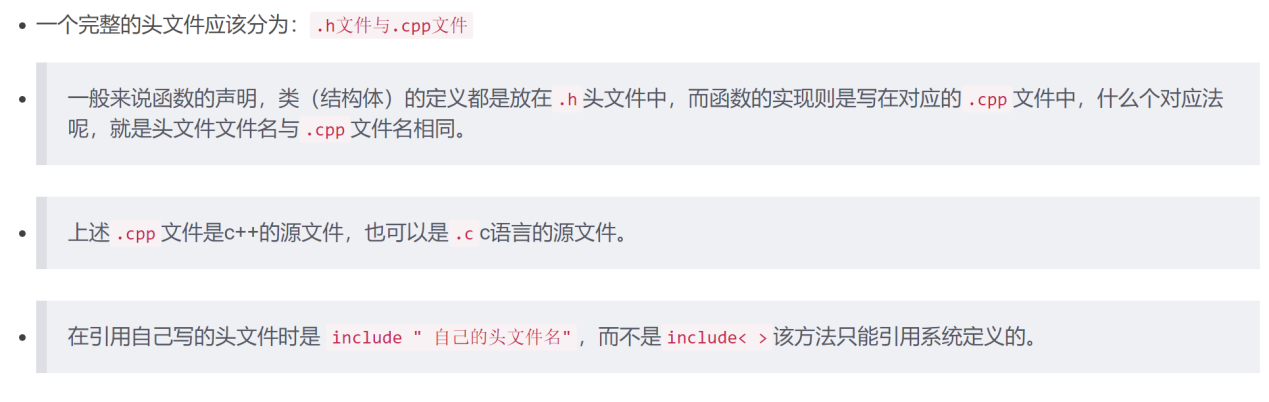
VS在源文件中如何使用自己写的头文件



头文件中按照如下格式来写入代码：

//Circle\_queue 头文件名

#ifndef CIRCLE\_QUEUE\_H //就是头文件名（全大写后加个\_H）

#define CIRCLE\_QUEUE\_H

#endif

将函数声明，类（结构体）定义写在头文件中

建立头文件对应的.cpp文件，写函数的实现（源文件名与头文件名一定要相同）



malloc是C语言中用来动态分配内存的函数，它可以根据需要分配指定大小的内存空间，并返回一个指向该空间的指针。malloc的用法如下：

首先，需要引入头文件<stdlib.h>或<malloc.h>，这样才能使用malloc函数。

然后，调用malloc函数，传入一个参数，表示要分配的内存空间的字节数。例如，如果要分配一个整数类型的空间，可以写malloc(sizeof(int))，其中sizeof(int)表示一个整数占用的字节数。

malloc函数会返回一个void\*类型的指针，这是一个通用的指针类型，可以指向任何类型的数据。为了使用这个指针，需要将它强制转换为所需的类型。例如，如果要使用一个整数类型的指针，可以写(int\*)malloc(sizeof(int))，其中(int\*)表示将void\*转换为int\*。

如果malloc函数成功分配了内存空间，就会返回一个有效的指针地址，可以通过这个指针访问或修改分配的空间中的数据。例如，如果要给分配的整数空间赋值为10，可以写\*(int\*)malloc(sizeof(int)) = 10;，其中\*表示解引用操作符，用于访问指针所指向的数据。

如果malloc函数分配失败，就会返回一个空指针NULL，表示没有可用的内存空间。这时候，需要检查返回值是否为NULL，并做相应的处理。例如，如果要判断分配是否成功，可以写if ((int\*)malloc(sizeof(int)) == NULL) { printf("内存分配失败"); } else { printf("内存分配成功"); }

使用完malloc函数分配的内存空间后，需要使用free函数释放它，以避免内存泄漏。free函数接受一个参数，就是要释放的指针地址。例如，如果要释放一个整数类型的指针p，可以写free(p);

释放完内存空间后，最好将指针置为NULL，以防止出现悬空指针或野指针。例如，如果要将一个整数类型的指针p置为NULL，可以写p = NULL;

array = (int\*)malloc(sizeof(int) \* size);

这个动态分配内存的语句，可以理解为array是一个指向一个整数数组的指针，这个数组的大小是size，也就是说，array可以访问和修改这个数组中的元素。

动态分配内存是分配在堆上，而不是栈上。

free函数是C语言中用来释放动态分配的内存空间的函数，它可以避免内存泄漏和浪费。

需要引入头文件<stdlib.h>或<malloc.h>，这样才能使用free函数。

调用free函数，传入一个参数，表示要释放的内存空间的指针。

例如，如果要释放一个整数类型的指针p，可以写free(p);

释放完内存空间后，最好将指针置为NULL，以防止出现悬空指针或野指针。

可变数组

typedef struct

{

int\* array;

int size;

}\*Array;

定义一个结构体类型，名为Array。然后用typedef给这个结构体类型起一个别名Array，表示指向Array类型的指针。这样，你就可以用Array来声明一个Array类型的指针变量，而不用每次都写struct Array \*

memcpy函数是一个C语言标准库函数，它用于将一块内存区域的内容复制到另一块内存区域。它的函数原型是：

void \*memcpy(void \*dest, const void \*src, size\_t n);

其中，dest是目标内存区域的指针，src是源内存区域的指针，n是要复制的字节数。函数返回一个指向dest的指针。

memcpy函数有以下特点：

它可以复制任何类型的数据，不仅仅是字符串。

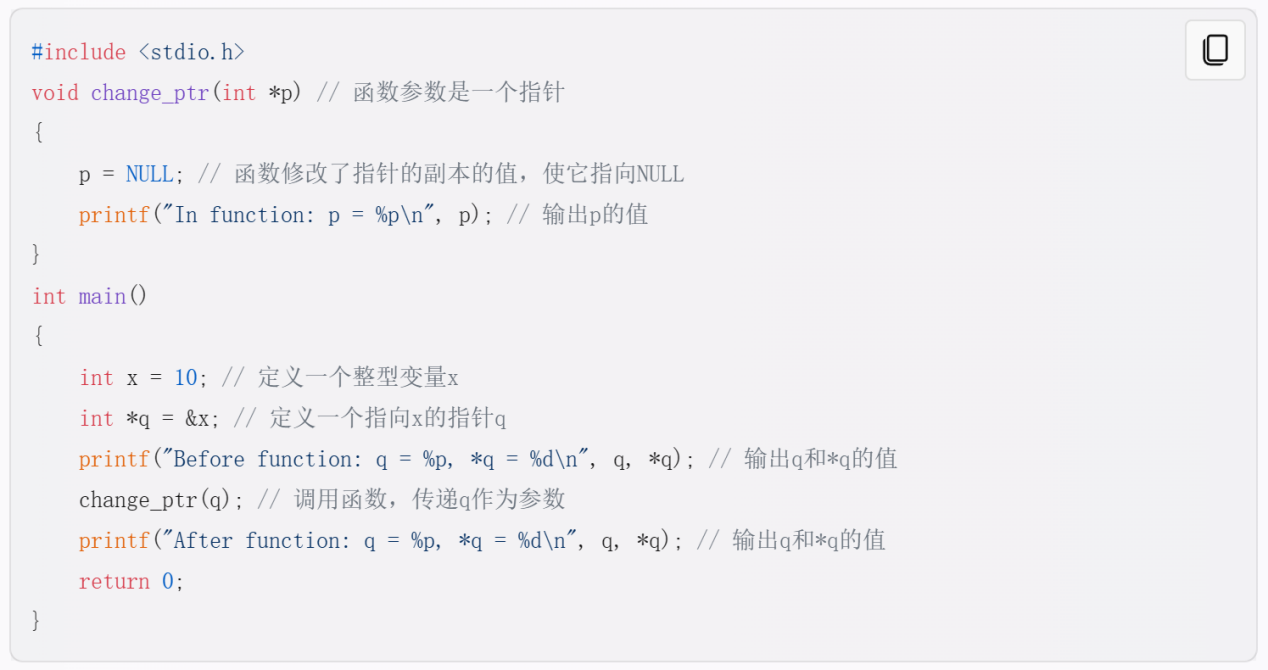
它不会检查源和目标内存区域是否重叠，如果重叠，可能会导致数据损坏。

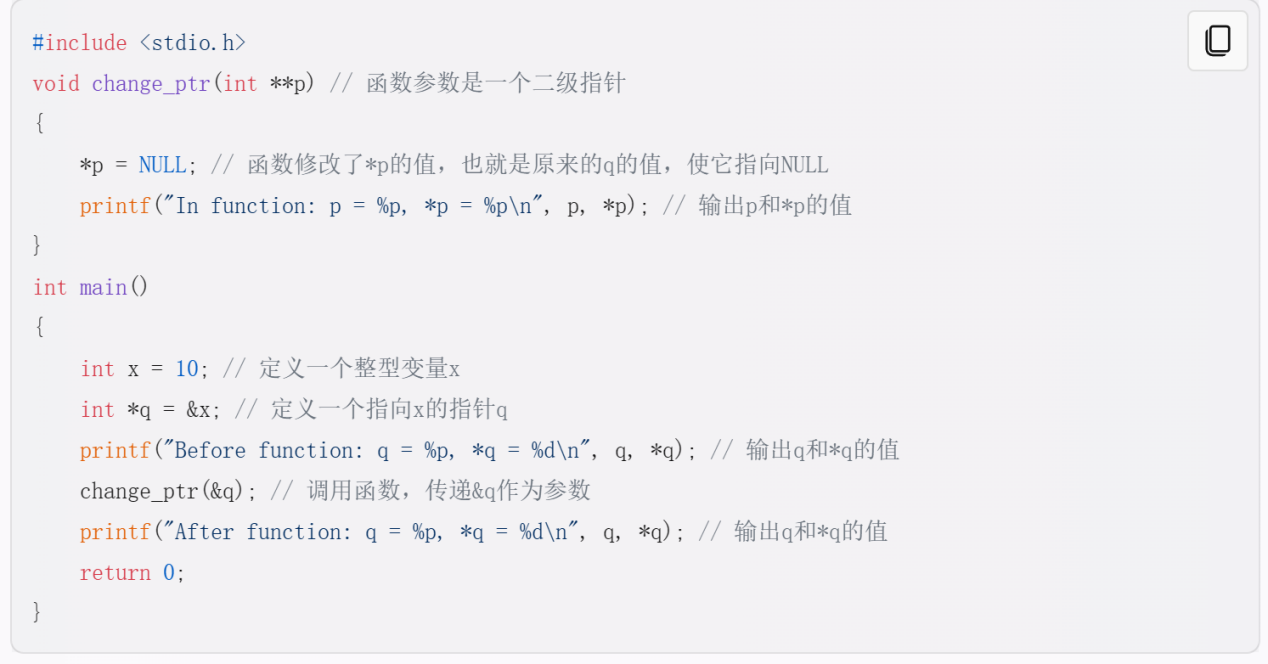
它不会在遇到’\0’字符时停止复制，而是一定会复制完n个字节。



拷贝要很多时间，内存空间并不一定够

因为函数的参数是按值传递的，也就是说，函数只能操作参数的副本，而不能改变原来的参数。如果参数是一个指针，那么函数可以改变指针的副本的值，使它指向另一个地址，但是这不会影响原来的指针的值。而如果函数想要修改指针指向的变量的值，那么就需要传递一个指向指针的指针，也就是二级指针，这样函数才能通过解引用操作符\*来访问和修改原来的指针和它所指向的变量。

从输出结果可以看出，函数change\_ptr只能修改p的值，而不能修改q和q的值。



realloc函数是一种用于重新分配内存空间的函数，它可以根据需要增加或减少已经分配的内存空间的大小。

realloc函数的用法如下：

- realloc函数的原型是void \*realloc(void \*ptr, size\_t size)，其中ptr是指向已经分配的内存空间的指针，size是新的内存空间的大小，以字节为单位。

- realloc函数的返回值是一个指向新的内存空间的指针，如果重新分配成功，这个指针可能和原来的指针相同，也可能不同。如果重新分配失败，返回NULL，并且原来的内存空间不变。

- realloc函数会尽量保留原来的内存空间中的数据，如果新的内存空间比原来的大，那么多出来的部分会被初始化为0；如果新的内存空间比原来的小，那么多余的数据会被丢弃。

- realloc函数会根据当前的内存状况来决定是否需要移动数据。如果原来的内存空间后面有足够的连续空闲空间，那么realloc函数会直接扩展或缩小原来的内存空间，并返回原来的指针；如果原来的内存空间后面没有足够的连续空闲空间，那么realloc函数会在堆中寻找一个合适的新位置，将原来的数据复制到新位置，并释放原来的内存空间，然后返回新位置的指针。

- realloc函数可以用于动态地调整数组或结构体等数据结构的大小，以适应不同的需求。但是使用realloc函数时要注意以下几点：

- 调用realloc函数时要检查返回值是否为NULL，如果为NULL，说明重新分配失败，此时要处理好原来的内存空间，避免内存泄漏或无效访问。

- 调用realloc函数时要保留原来的指针，以防重新分配失败导致指针丢失。如果重新分配成功，可以用新返回的指针覆盖原来的指针。

- 调用realloc函数时要注意新旧内存空间大小之间的关系，如果新大小比旧大小小，要注意数据是否会被截断或丢失；如果新大小比旧大小大，要注意是否需要初始化多出来的部分。

- 调用realloc函数时要注意传入的指针是否有效，如果传入NULL，那么相当于调用malloc函数；如果传入非NULL但不是由malloc、calloc或realloc分配的指针，那么可能会导致未定义行为。

exit函数是一种用于终止程序运行的函数，它可以在任何地方调用，不管是在主函数还是在子函数中。exit函数的用法如下：

exit函数的原型是void exit(int status)，其中status是一个整数，表示程序的退出状态。一般来说，status为0表示正常退出，非0表示异常退出。这个状态值会被传递给操作系统或者调用程序，以供其它程序使用。

exit函数的功能是关闭所有文件，释放所有动态分配的内存，终止当前进程，并返回给操作系统或者调用程序。exit函数不会返回任何值。

exit函数和return语句的区别是，return语句只能用在函数中，表示从函数返回；exit函数可以用在任何地方，表示从程序退出。return语句只会影响当前函数的执行，exit函数会影响整个程序的执行。