****1.半角符号****：

* ****定义****：半角符号是ASCII码表中的字符，这些字符通常占用**一个字节**的存储空间。
* ****特点****：它们主要用于英文字符和一些基本的标点符号，如英文字母（A-Z，a-z）、数字（0-9）以及基本的符号（如 !, @, #, $, % 等）。
* ****显示****：在显示和打印时，半角符号通常占用较窄的空间。

****2.全角符号****：

****定义****：全角符号是Unicode字符集中的字符，这些字符通常占用**两个字节**的存储空间。

****特点****：它们主要用于表示中日韩等语言的字符以及一些特殊符号。此外，全角符号还包括全角的英文字符和数字，这些字符在视觉上与半角字符相似，但占用的空间更大。

****显示****：在显示和打印时，全角符号通常占用较宽的空间。

****UTF-8**：**使用1到4个字节来表示一个字符，是互联网上最常用的Unicode编码方式。UTF-8编码方式具有兼容ASCII码、节省存储空间、避免字节顺序问题等优点。



//ps:不要用cin读取，因为cin会天然跳过空格和回车的读取

//\_getch--无回显 的回车是\r；而getchar--有回显 是\n

getchar()函数在读取字符时会**自动忽略掉换行符**（即回车键输入的'\n'）。因此，如果用户只是按下了回车键而没有输入其他字符，getchar()函数会返回一个特殊的值EOF（End Of File），表示已经读取到了输入流的末尾。



cin读取的时候会天然地不读 空格、回车等，那么**如何处理含有空格的字符输入**呢？

#include<iostream>

using namespace std; //ps:不要用cin读取，因为cin会天然跳过空格和回车的读取

//\_getche--有回显 的回车是\r；而getchar--有回显 是\n

int main()

{

char str[3][128] = { 0};//str0代表第一行输入的文字

for (int i = 0; i < 3; ++i) {

cout << "请输入第" << i+1 << "行" << endl;

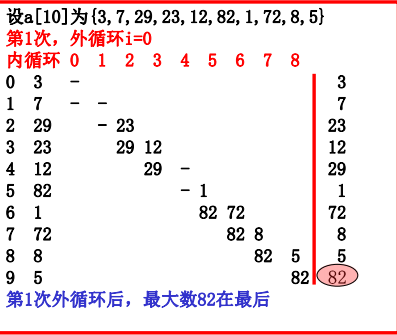
**cin.getline(str[i], 128);//cin.getline(char\* s, streamsize n)，其中 s 是字符数组的指针，n 是要读取的最大字符数（包括空字符）**

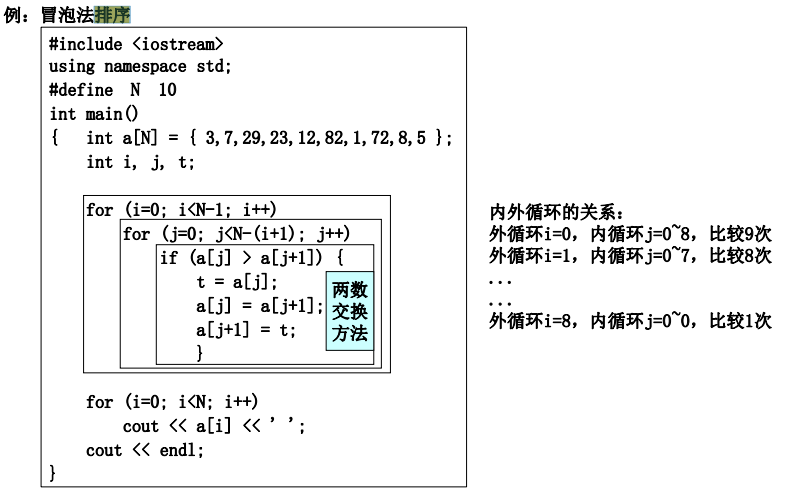
}

return 0;

}

回顾：冒泡排序和选择排序





冒泡排序：

基本思想

冒泡排序是一种简单的排序算法，它重复地遍历要排序的列表，比较相邻的元素，如果它们的顺序错误就交换它们的位置。这个过程像是一个气泡一样，把较大的元素逐渐“浮”到列表的末尾。

工作原理

从列表的第一个元素开始，比较相邻的两个元素。

如果第一个元素比第二个元素大（假设是升序排序），则交换它们的位置。

对每一对相邻元素做同样的工作，从列表的开始到结尾。

在每一次遍历列表后，最大的元素会“浮”到列表的末尾，因为它被交换到了那里。

重复上述步骤，直到没有需要交换的元素为止，这意味着列表已经排序完成。

示例

假设有一个列表 [64, 34, 25, 12, 22, 11, 90]，进行冒泡排序的过程如下：

第一轮: [34, 25, 12, 22, 11, 64, 90]

第二轮: [25, 12, 22, 11, 34, 64, 90]

第三轮: [12, 22, 11, 25, 34, 64, 90]

第四轮: [12, 11, 22, 25, 34, 64, 90]

第五轮: [11, 12, 22, 25, 34, 64, 90]

复杂度

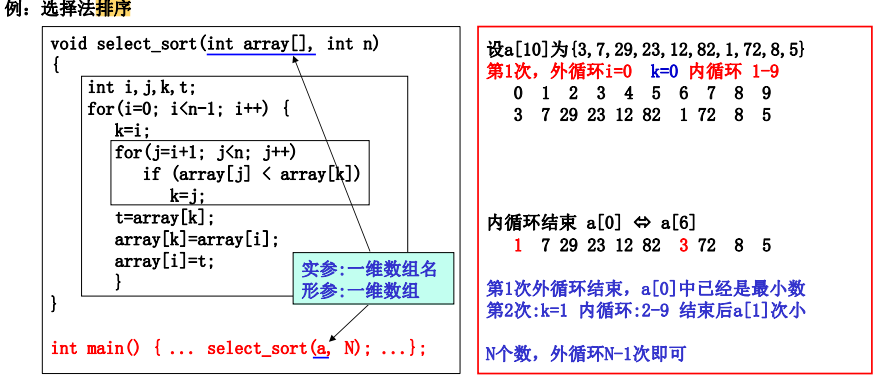
时间复杂度:

最优情况: O(n) （列表已经有序）

最坏情况: O(n^2) （列表逆序）

平均情况: O(n^2)

空间复杂度: O(1) （原地排序）



（key--就是不断比较，得到最小的，然后放在前面，**一开始k=i**，只是为了得到一个比较的基准）

选择排序

基本思想

选择排序是一种简单直观的排序算法。它的工作原理是每次从未排序的部分中选出最小（或最大）的元素，放到已排序部分的末尾。

工作原理

从未排序的列表中找出最小（或最大）的元素，存放到排序序列的起始位置。

接着，从剩余未排序元素中继续寻找最小（或最大）的元素，然后放到已排序序列的末尾。

重复上述步骤，直到所有元素均排序完毕。

示例

假设有一个列表 [64, 25, 12, 22, 11]，进行选择排序的过程如下：

第一轮: 找到最小元素11，放到第一个位置 [11, 25, 12, 22, 64]

第二轮: 找到剩余部分最小元素12，放到第二个位置 [11, 12, 25, 22, 64]

第三轮: 找到剩余部分最小元素22，放到第三个位置 [11, 12, 22, 25, 64]

第四轮: 剩余部分只有一个元素64，放到最后一个位置 [11, 12, 22, 25, 64]

复杂度

时间复杂度:

最优情况: O(n^2)

最坏情况: O(n^2)

平均情况: O(n^2)

空间复杂度: O(1) （原地排序）