# 墨烯的C语言技术栈-C语言基础-012

数组

要存储1-100的数字,怎么存储?

C语言中给数组的定义:一组相同类型元素的集合

数组定义

int arr[10] = {1,2,3,4,5,6,7,8,9,10}; // 定义一个整型数组,最多放10个元素

int main()

{

// 0-9都是整数

int arr[10] = {10,11,12,13,14,15,16,17,18,19};// arr是数组名 [10]是申请内存空间 序号从0开始也就对应10的位置上 []不写就会按后面存多少数给内存

printf("%d\n",arr[8]);

int i = 0;

while(i < 10)

{

printf("%d\n",arr[i]);

i++;

}

也可以使用for循环

for(i=0;i<10;i++)

{

printf("%d\n",arr[i]);

}

return 0;

}

数组的下标

C语言规定:数组的每个元素都有一个下标,下标从0开始的

数组可以通过下标来访问的

如arr[8] = 18;

单个访问是用下标来访问的

多个访问是用循环或者其他结构结合来访问的

如后面刷算法使用的中最简单的冒泡排序

这里初步说明一下 以后考试和面试会用到

冒泡排序是相邻元素的比较和互换大的放后面 小的放前面

#include <stdio.h>

void mp(int arr[],int len) // 设置数组arr[] 和 整型变量len

{

int i,j,temp; // 设置整型变量 i, j, temp

for(i = 0; i < len - 1; i++) //

i初始值设置为0 len为为数组长度 i自增(每次循环后i+1)

len-1的原因是每一轮比较会把较大的数排在最后

然后下一轮就不考虑最后一个数

这一个过程嘛 简单来说就是 一个外循环 而且每一轮外循环都对数组每一个元素进行至少一次比较.

这是因为每一轮外层循环后 最大的元素会被列到数组末尾

就比如 1 4 3 6 8 7

一次外循环就变成 1 4 3 6 7 (8)

所以每一次会把末尾的减去也就是len -1的原因 用来来控制轮次

{

for(j = 0; j < len - 1 - i; j++) //

j初始值设置为0

j为什么要 < len-1-i呢 -i是每轮已确定最大值元素的个数 -1避免数组越界(越界就是超过内存空间)

这个是优化方向的考虑 在外循环中已经确定了最大元素被移动到数组的末尾,

因此内循环不需要对已经确定位置的元素进行比较(也就是末尾的) 只需要相邻两数进行比较 也就是控制比较的元素边界

如 1 4 3 6 7 中

1 4进行比较 4 3进行比较 3 6进行比较 6 7 进行比较

然后会变成 1 3 4 6 7 因为这是比较简单一串数组所以一次性就排好了

一次性排不好就会是

如1 7 6 5 0 这样

1 7 比较 7 6 比较 6 5比较 5 0比较

变成 1 7 6 5 0 然后 1 6 7 5 0 然后 1 5 6 7 0 然后

0 1 5 6 7

这就是两两元素进行比较 至少一次比较

这两个嘛没啥好说的 也就是 外循环控制循环轮次排出最后一个最大数 内循环就是两两相邻元素进行比较 也就这样

{

if(arr[j] > arr[j+1]) // 这个就是冒泡的核心思想了 通过不断比较相邻的两个元素 以将较大的元素逐渐移到数组的末尾 然后排序

{

temp = arr[j]; // 将arr[j]数存入 temp

arr[j] = arr[j + 1]; // 这种情况是arr[j] >arr[j+1]的时候 所以将arr[j]的数换位到arr[j+1] 也就是后一位

arr[j + 1] = temp;// 然后将temp存入原来后一位

简单来说就是 6 3这种情况 把6存入temp 然后 6那个位置存入了arr[j + 1]也就是3这个数 再把temp = 6这个数放在arr[j + 1]这个位置

如果这样还不能理解的话 就更简单的来说

军训排队你180cm站在 你朋友175cm前面 然后教官说高的站在后面 然后你站在一边等你朋友向前去 然后你站到你朋友原来的位置上 就是这样

}

}

}

}

void mpxh(int arr[],int len)

{

int i;

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

{

printf("%d\t",arr[i]);

}

printf("\n");

// 这个没啥好说的 就是一个循环编历方法 然后下面换行

}

int main()

{

int arr[] = {53, 14, 17, 8, 2, 88, 23};

int len = (int) sizeof(arr) / szieof(\*arr);

printf("初始数列:");

mpxh(arr,len);

printf("开始对其冒泡排序...");

mp(arr,len);

printf("排序完成:");

mpxh(arr,len);

return 0;

// 这样的结果就是输出最后一个结果为

排序完成:2 8 14 17 23 53 88

冒泡排序

优点:

简单易懂 最基础的排序方法嘛

稳定排序 它们在排序后的相对顺序不会改变

空间复杂度低 只需要使用固定的额外空间来交换元素 空间复杂度为O(1)

缺点

时间复杂度高 就一个嵌套循环让时间复杂度直接增加到O(n的平方) (这里没有学过时间复杂度的可以理解为指数性的增加)

不适于大数据的集 高的时间复杂度让它在处理大数据上消耗大量的计算资源和时间

排序效果依赖初始顺序 如果初始顺序接近有序，排序的效率可能会提高；但如果初始数组完全无序，排序的效率会相对较低。

}