2022.5.9

先自己思考思考题和例题，答案在最后

一、循环判断条件+穷举

1.for循环

（1）组成部分：(initialize; test; update)

（2）初始化：只在循环开始时执行一次，“中介变量”

（3）判断条件：和初始化配合决定循环次数

（4）每个循环结束时更新“中介变量”的值，标志结束一次，开始下一次

思考1：判断条件是判断终止还是判断继续？

例：假如我想循环5次，我应该写：

for (int k = 0; k < 5; k++) 还是 for (int k = 0; k == 5; k++) ？

思考2:在每次循环中这三个组成部分分别在循环的什么时间执行？

练习1：编写一个要求用户输人两个整数的程序。该程序将计算并输出这两个整数之间（包括这两个整数）所有整数的和。

补充：擅用标志flag，读懂以下代码指出flag的作用意义：

// find the position of 5 in a 2-dimension array

int A[3][3] = {{1,2,3},{4,5,6},{7,8,9}};

bool flag = false; // false stands for not-found.

int position = 0; // Initialize after defining a variable is a good habit.

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

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

if (A[i][j] == 5) {

flag = true;

position = (3 \* (i + 1)) + (j + 1);

break;

}

}

if (flag)

break;

}

printf("The position of 5 is (%d, %d)\n", position / 3, position % 3);

2.while循环

（1）组成部分：while (test-condition) body;

（2）test为true才执行

（3）body循环体必须包含终止测试条件（if (…) break;）

思考3:什么时候用for什么时候用while？

3.do-while循环

（1）组成部分：do {body} while (test-condition) ;

思考4：如何选择用while循环还是do-while？（做例题）

练习2：提示用户输入一个大于零的数，如果输入小于等于零则重新显示提示语并读取，知道满足输入条件。

穷举练习3：\_ \_ \_+\_ \_ \_ = \_ \_ \_ 不重复的填入1～9使等式成立，并输出所有可能（很暴力的穷举，学会使用标记数组简化判断条件）

二、迭代法

（1）核心：不断用变量的旧值经过相同的计算得到新值的过程，适合用于重复去做一组指令的情况。其中每次执行完一次循环都会更新计算的数值，用于下一次计算。

例：用迭代法算斐波那契数列前n项，用户输入n

三、递推法

（1）核心：第n次循环的计算值取决于前n-1次的某些值，需要推出递推公式。

练习4：用1\*1和2\*2的瓷砖铺满N\*3的地板共有多少种方式？提示：用草稿纸画一画找找关系。（N为用户输入）

四、递归

（1）自己调用自己，一定要搞明白传递参数是什么。

（2）适合倒序输出问题。

简单理解：

送归：你打开面前这扇门，看到屋里面还有一扇门。你走过去，发现手中的钥匙还可以打开它，你推开门，发现里面还有一扇门，你继续打开它。若干次之后，你打开面前的门后，发现只有一间屋子，没有门了。然后，你开始原路返回，每走回一间屋子

你数一次，走到入口的时候，你可以回答出你到底用这你把钥匙打开了几扇门。

循环：你打开面前这扇门，看到屋里面还有一扇门。你走过去，发现手中的钥匙还可以打开它，你推开门，发现里面还有一扇门（若前面两扇门都一样，那么这扇门和前两扇门也一样；如果第二扇门比第一扇门小，那么这扇门也比第二扇门小，你继续

打开这扇门，一直这样继续下去直到打开所有的门。但是，入口处的人始终等不到你回去告诉他答案。）

练习5：用递归函数求解斐波那契数列，掌握return技巧

练习6: 用递归函数将字符串反序，其中将函数传递参数设置为字符数组下标的序号。掌握递归开始判断传递参数的方法以及倒序输出。

练习7:（综合）用递归函数把十进制转换为二进制，并实现在递归函数内用倒序法直接将结果输出。

思考5：迭代/递推 vs 递归 各自的优势？如何选择？

六、令人头疼的指针

（1）指针是个啥：指针是一个变量。

（2）指针储存的是个啥：储存的是其他基础类型变量的 地址

<https://blog.csdn.net/liu100m/article/details/90731422?utm_source=app&app_version=5.1.1>

先看看这个，有不懂的随时问我

七、深入数组与指针的关系

（1）分清指针变量和数组的类型，例如int a，那么a是int类型，&a是int \*类型！因为指针变量存储的内容就是地址！加取地址符号可以理解为类型升级一个地址。

（2）等价性：由于在一维情况下数组（名）和指针都代表地址，所以可以把二者理解为同一个东西。

（3）加减法：将指针/数组做加减法的时候要降级一个地址，恰好与&相反，例如a是int\*类型的话，a+n相当于a+n\*sizeof（int）

思考6: int Zippo[4][2]，说出一下变量/常量的类型

Zippo：

Zippo+0:

Zippo[0]：

Zippo[0][0]：

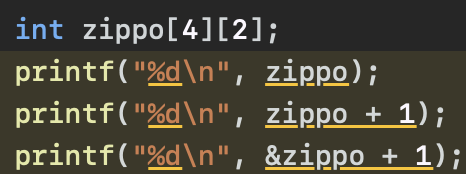
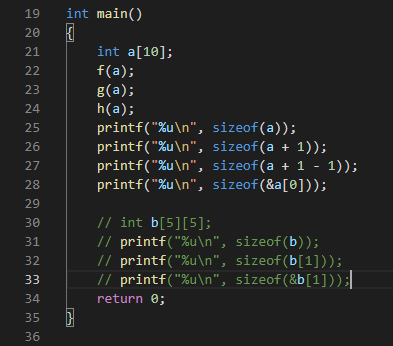
&Zippo[0][0]：

& Zippo[0]：

&Zippo：

思考7: 想一想下面的例题输出什么结果

提示：sizeof（a）的时候想清楚a的基本类型是什么就好说了



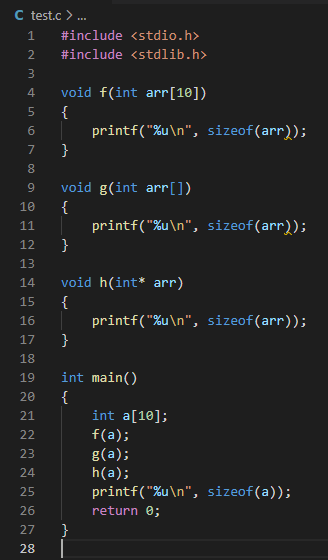
（4）数组退化：当数组作为传递参数时，会退化相应类型的指针变量。注意一定是相应类型。

思考8: 一下数组作为传递参数时将退化成什么类型的指针？

Int a[10]:

Int a[4][3]:

思考9:思考以下程序的输出结果是什么？



答案部分：

思考1：前者，千万不要把test理解为终止条件，判断条件为真，才执行每次循环。if ((bool)test == true) do…;

思考2：init只在最最开始，且仅有一次；每次循环开始时执行test；每次循环结束时update。

思考3：知道执行次数用for，否则用while。

思考4/练习2: do while确保循环至少执行一次，适用于类似提示输入并且有检查输入是否满足要求时。对比以下代码可知do while在此时相较于while循环的明显优势：

int a;

// do while version:

do {

printf("Please input a positive number:");

scanf("%d", &a);

} while (a <= 0) ;

printf("Done!\n");

// while version:

printf("Please input a positive number:");

scanf("%d", &a);

while (a <= 0) {

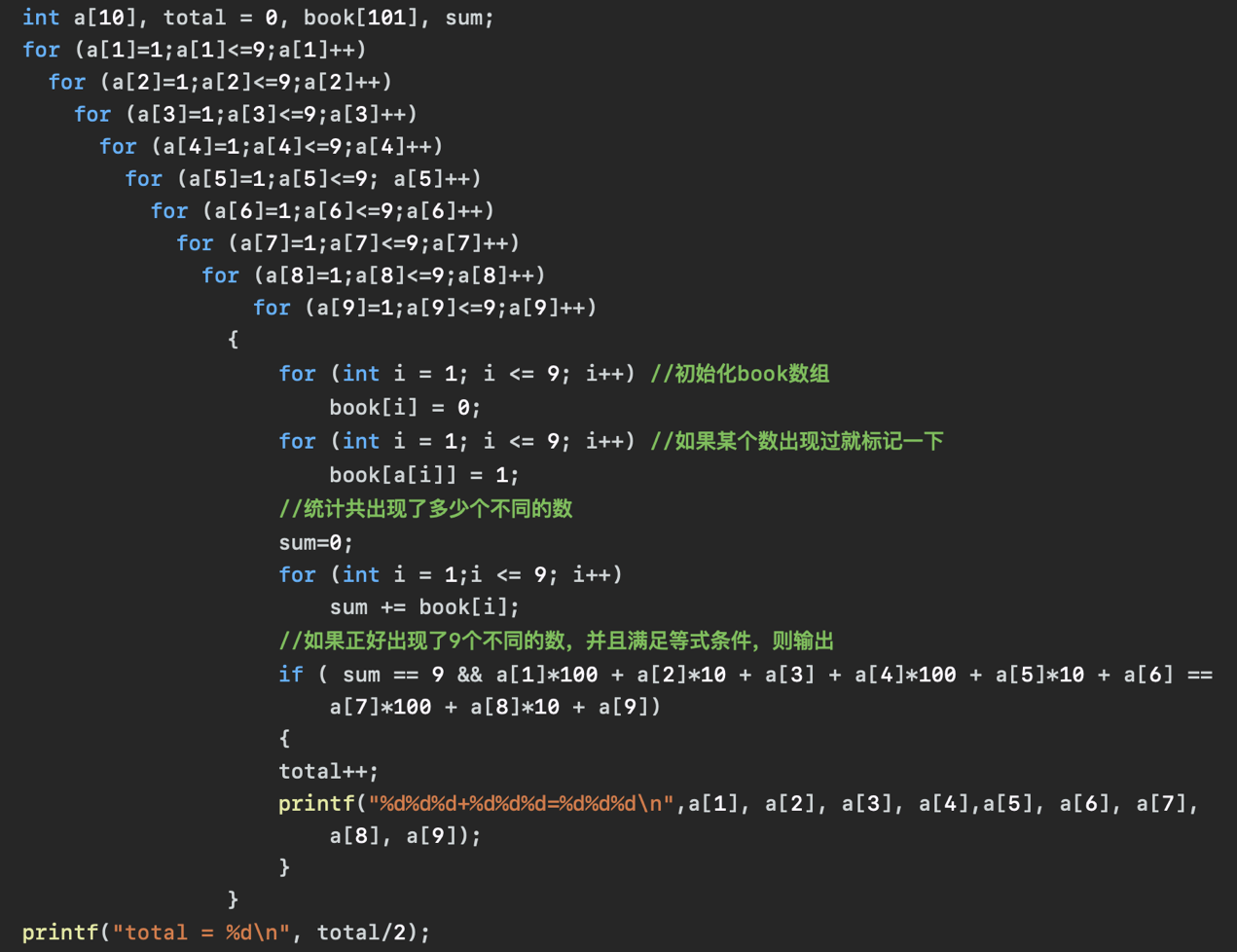
printf("Please input a positive number:");

scanf("%d", &a);

}

printf("Done!\n");

穷举练习答案：



是不是很慢？咋办？等你以后学了深度优先搜索的

思考5: 递推/迭代代码较复杂时间复杂度高（循环和判断条件多），但空间复杂度低，适合数据量很大时使用；递归代码简单，思路清晰，但理解略有困难，空间复杂度大，适合小数据。有句比较经典的话叫人理解递推迭代，神创造递归哈哈哈。

思考6:

Zippo：数组名，怎么定义就是什么 int [4][2]

Zippo+0: 完全等价于&Zippo[0], int (\*)[2]，相当于指向有两个元素的一位数组的指针

Zippo[0]：int[2]

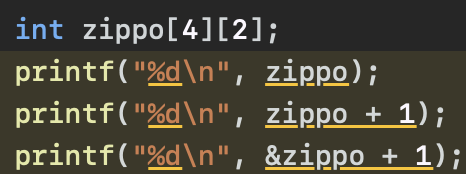
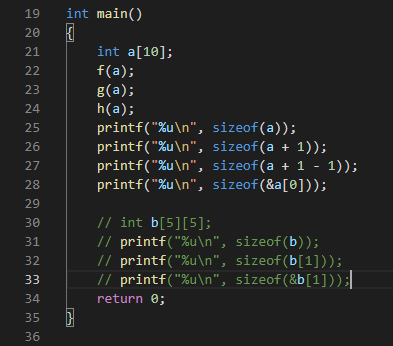
Zippo[0][0]：int

&Zippo[0][0]：int\*

& Zippo[0]：

&Zippo：int (\*)[4][2]，相当于指向Zippo的指针

思考7:



左侧：40；8；8；8（8还是4取决于编译器，你的codeblocks是4）

右侧：一个随机地址的十进制数字n；n+4\*2；n+4\*2\*4

思考8:

Int a[10]: int \* a

Int a[4][3]: int (\*)[3]，指向一个有三个int元素的一维数组的指针，这也是为什么二维数组作为传递参数时后边中括号里的数字必须要写，因为程序得知道对a做加减法时+-的应该是sizeof（\*a），其中\*a的类型是int[3]，如果不说明三的话程序不知道sizeof（\*a）是多少，此处写了三的话就是3\*4=12，微观相当于偏移12个地址，如果宏观一点当作二维数组来看的话就相当于跳到了数组的下一行。

所以，一般当数组作为传递参数的时候，都会传递一个int类型的变量，记录一位数组元素的个数，二维数组的行数以防数组越界问题。

思考9:

8；8；8；40（codeblocks是4）