函数是面向过程语言的核心。

函数的优点：

可以让经常使用的代码很方便的重复使用（增强了代码复用性）

可以让代码分成更多更明晰的模块（增强了代码逻辑和可读性）

修改函数时所有调用该函数的地方都相当于一起改掉了（增强了代码可移植性和可维护性）

函数的参数：

形参：指的是形式参数，特指函数调用方传给函数的值，由于这个参数只提供值，所以只是形式上的参数。

实参：是子函数在定义变量来承接调用方传入的值。与形参必须一一对应。由于它真的定义了变量，所以是真正的参数

函数的返回值：

返回值类型是由函数注明的返回值类型决定的，与return返回的值的类型无关

函数不需要返回值时返回值类型为void（空类型）

()运算符：

单目在右，作用于函数名，表达式的值为函数的返回值，可以携带信息。

函数的调用遵循先入后出原则，是插入式执行的。即调用的函数执行完毕后，才会回到调用的地方继续向下执行。

函数的生命周期是从调用时开始，到return语句或者函数大括号自然结束为止的。

栈帧指的是函数在内存中占用的空间。调用时创建栈帧，结束时销毁栈帧。销毁时，所有栈帧内的变量都会被销毁，销毁后会留下返回值。

注意：由于销毁时函数栈帧里的内容都被销毁了，所以子函数中定义的临时变量的指针/数组是不能被返回的，会造成野指针错误。

拆分/编写函数三部曲：

1、确立函数的功能，决定要拆分/编写的模块。

2、根据自己确立的功能，确定函数需要的原料（确定参数列表）和最终加工成的成品类型（确定返回值类型）。

3、最后复制修改/编写函数体代码，让函数实现功能。

拆分原则：尽可能让拆分出的函数功能单一，从而提升可移植性。

二维数组：二维数组是数组的数组。

二维数组的初始化：

按行初始化：使用大括号嵌套的方式逐行初始化。

直接初始化：类似一维数组的方式进行初始化，一行写满了从下一行继续写。

缺省初始化：第一维中间可以留空，系统会自动判定是几行。但是第二维不能留空，因为这是数组类型的一部分。

二维数组的空间也是连续的。

二维数组通常当做表格来使用。

二维数组的四种遍历：

横向遍历：正常的先i后j遍历。

```
for (i = 0; i < X; i++) //二维数组遍历
{
    for (j = 0; j < Y; j++)
    {
        printf("%2d ", a[i][j]);
    }
    putchar('\n');
}
```

纵向遍历：在上一个的基础上，交换横纵坐标。

```
for (i = 0; i < Y; i++)
{
    for (j = 0; j < X; j++)
    {
        printf("%2d ", a[j][i]);
    }
    putchar('\n');
}
```

左上右下斜向遍历：这条斜线上的点横纵坐标的差值恒定。

```
for (i = -X + 1; i < Y; i++)
{
    j = 0;
    if (i < 0)
    {
        j = -i;
    }
    for (; j < X && i + j < Y; j++) //横纵坐标的差固定
    {
        printf("%2d ", a[j][j + i]);
    }
    putchar('\n');
}
```

右上左下斜向遍历：这条斜线上的点，横纵坐标的和恒定。

```
for (i = 0; i < X + Y - 1; i++)
{
    j = 0;
    if (i >= Y)
    {
        j = i - Y + 1;
    }
    for (; j < X && i - j >= 0; j++) //横纵坐标的和固定
    {
        printf("%2d ", a[j][i - j]);
    }
    putchar('\n');
}
```

二维数组的平移或旋转：写出平移/旋转前后的横纵坐标，寻找坐标变化规律，然后根据自己找到的规律，将数据转存到另一个二维数组中。

注意：有的题目中会需要遍历二维数组周围的一些点，一个比较简单的做法是，在定义时提前留下一行一列的空白。

数组作为参数传入函数的注意点：数组传入子函数后，会丢失掉数组的特性，变为普通指针，但多维数组除了第一维之外，其他维度是类型的一部分，所以多维数组传入后只有第一维会丢掉数组特征，其他维度仍然保持原样。