2020.10.2

学习内容：多维数组，函数的定义与调用，参数的传递，函数的声明和嵌套调用，函数的递归调用，变量的存储方式和生存期，指向变量的指针，指向数组的指针和指针数组，指向函数的指针和返回值是指针的函数，指向指针的指针，动态内存管理。

技术分享：

1. 多维数组；

多维数组的书写形式：例：a[b][c]……；

以二维数组为例即开辟一个b\*c\*sizeof（int）个字节的内存空间，在我看来最大的用处即a[b][c]可输入b个字符数不超过c的词、句或者长串数。至于高于三维的数组由于至今我还没进行实际应用便不多赘述。

此外还学习了二维数组的乘法和加法。

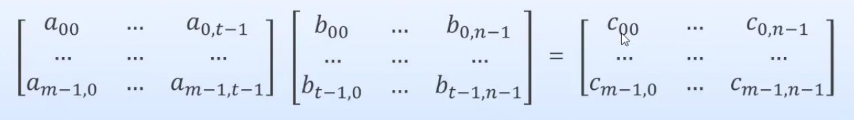
二维数组的加法即两个矩阵的相同位置的数相加

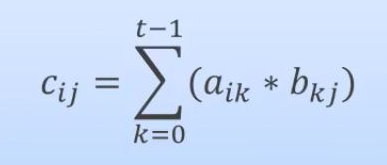
例a[0][4]={1,2,3,5}（以a代表），a[1][4]={5,3,6,4}（以b代表）而相加后的数组以c代表则c=a+b={6,5,9,9}

以打印出的形式表示即 1 2 5 3 6 5

3 5+ 6 4= 9 9

而乘法则是行的数乘以列的数再相加即一个m\*t（m是指多少行，t指多少列，后同）的矩阵a与一个t\*n的矩阵b相乘得一个m\*n的矩阵c

即有



举例以打印出的表现形式 1 2 5 3 17 11

3 5+ 6 4= 45 29（行乘以列）

1. 函数部分

**函数的声明**即声明一个函数，例如int max（int x,int y）

而**函数的定义**即在函数声明后加大括号定义函数，例如

Int max(int x,int y){  
 if(x>y){

return x;

}else{

return y;

}

}

大括号中的部分即是对函数的定义

**函数的调用**是通过在主函数中输入函数名加（）来实现的

例如

int main（）{

int c,a=5,b=3;

c=**max(a,b)**;

printf（“%d”,c）;

return 0;

}  
例子中加粗部分即是对上一例函数的调用，而括号中的数据类型则与函数的数据类型相同。

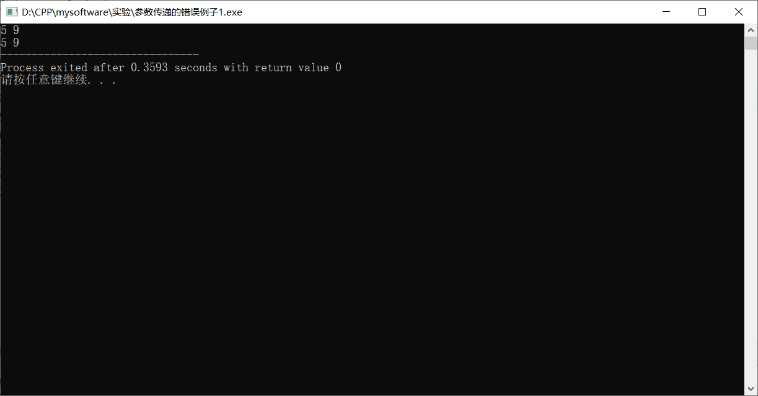
**参数的传递**

注意函数传递的错误形式

#include<stdio.h>

void exchange(int x,int y){

int temp=x;

 x=y;

y=temp;

}

int main(){

printf("%d %d\n",a,b);

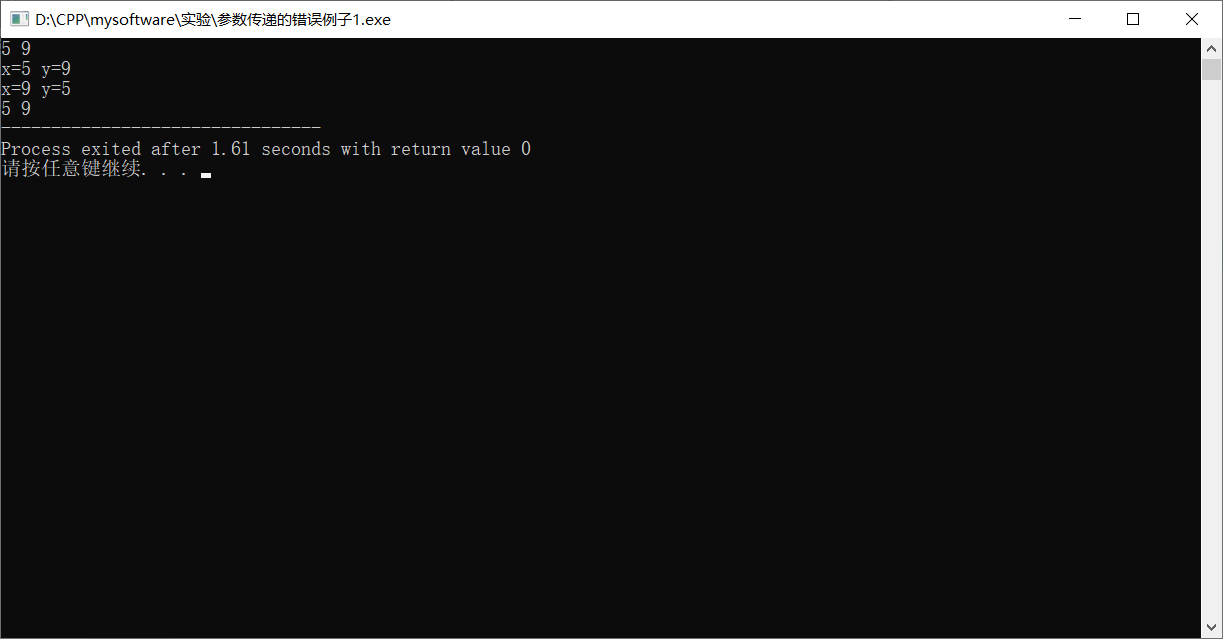
exchange(a,b);

printf("%d %d",a,b);

return 0;

}

可以看出，参数并没有有效传递，a与b的值并未发生改变，

而改变的只是x,y的值

在上述代码中x,y是形式参数而实际参数则是调用函数时往函数中传递的参数即a,b

而数组名作为参数的传递则能正确进行

例如#include<stdio.h>

void fun(int a[10]){

int i;

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

a[i]=i;

}

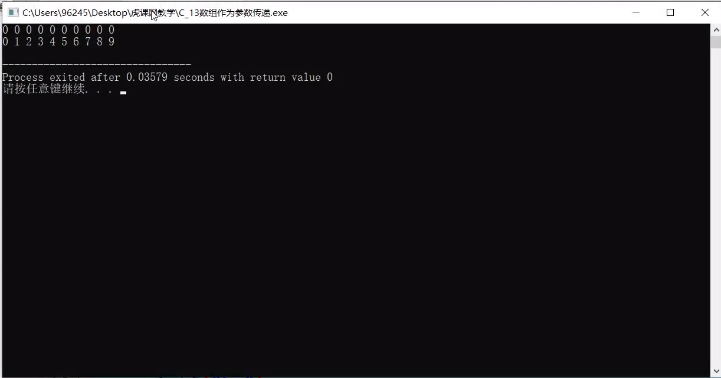
}

int main(){

int b[10]={0};

int i;

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

 printf("%d ",b[i]);

}

printf("\n");

fun(b);

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

printf("%d ",b[i]);

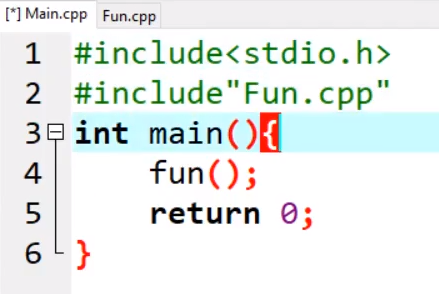
}

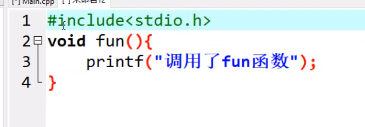
return 0;

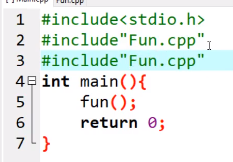
}

二者不同是因为数组名作为参数的传递方式是让a,b指向同一内存空间（也因此没必要对a指定长度）

**函数的调用**

 其中#include”Fun.cpp”即是对函数的 调用

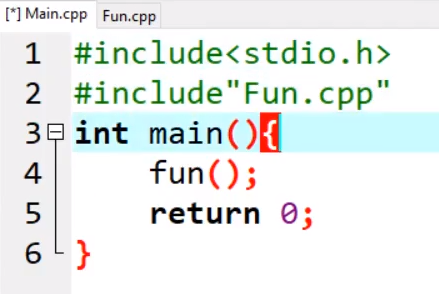


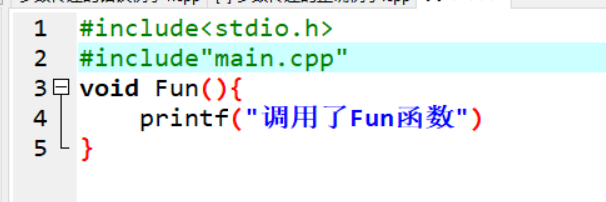


并且注意不能重复调用

这样会出现重复定义的错误

同样也不能在函数与主函数中互相调用如



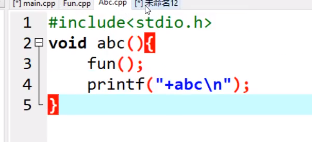
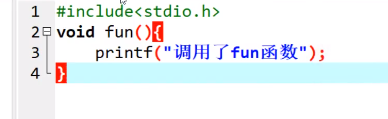
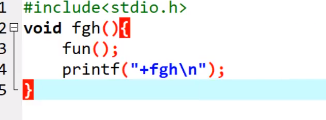
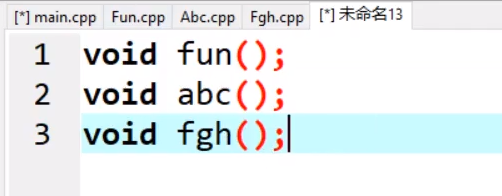


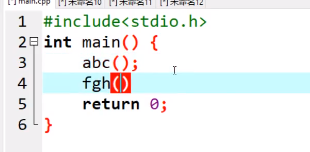
这样会无限持续下去

为了避免这繁多的错误（尤其当函数较多时），于是引入头文件与源文件

**头文件进行函数的声明（即文件名后加.h的文件）**

**源文件则是进行函数的定义（文件名后缀为.cpp）**

**如**

（第一幅图为头文件）

**函数的递归调用**

如int fbl(int n){

if(n==1||n==2){

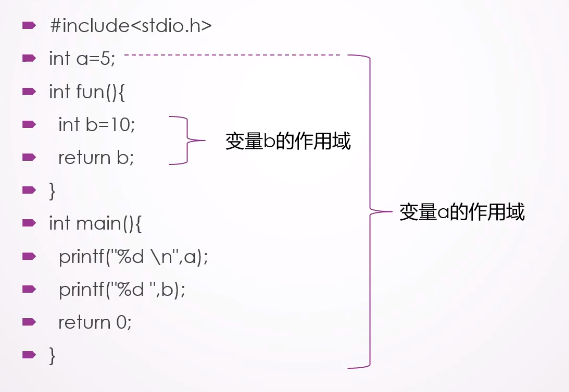
return 1;

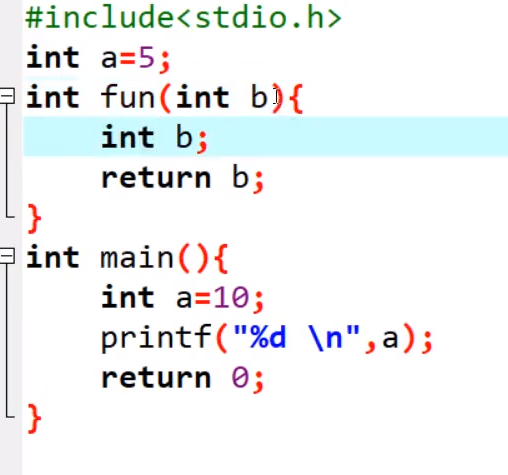
}else{

return fbl(n-1)+fbl(n-2);

}

}像这样即可进行斐波拉契数列的递归运算，在如同前述在主函数中调用即可进行递归调用。

**函数的生存期**（如图b为局部变量（在函数内部定义），a为全局变量（在函数外定义））

但是全局变量可以在函数中重新定义而局部变量不行如

图上半部分应用错误而下半部分正确。

**变量的存储方式**（变量可以分为**全局变量、静态全局变量、局部变量和静态局部变量**）

**按[作用域](https://www.baidu.com/s?wd=%E4%BD%9C%E7%94%A8%E5%9F%9F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)分**：

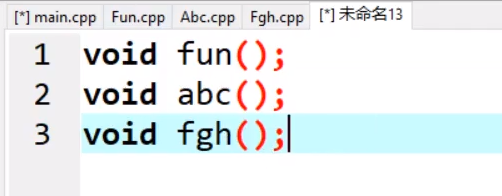
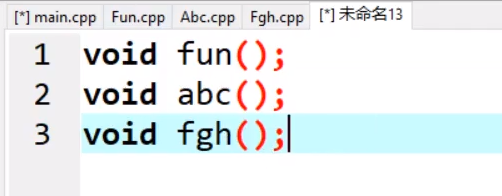
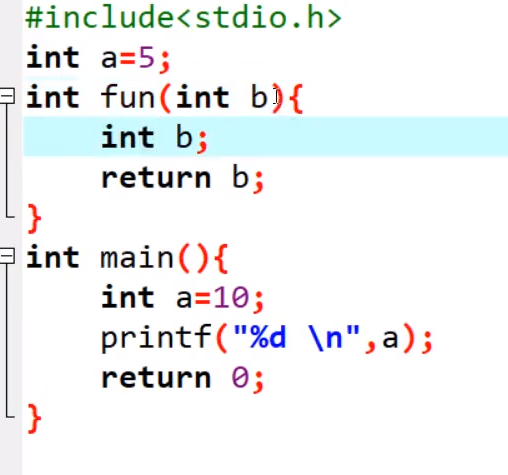
1、全局变量在整个工程文件内都有效；

2、[静态全局变量](https://www.baidu.com/s?wd=%E9%9D%99%E6%80%81%E5%85%A8%E5%B1%80%E5%8F%98%E9%87%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)只在定义它的文件内有效；

3、**静态局部变量只在定义它的函数内有效，且程序仅分配一次内存**，函数返回后，该变量不会消失；局部变量在定义它的函数内有效，但是函数返回后失效。  
4、**全局变量**和[静态变量](https://www.baidu.com/s?wd=%E9%9D%99%E6%80%81%E5%8F%98%E9%87%8F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)如果没有手工[初始化](https://www.baidu.com/s?wd=%E5%88%9D%E5%A7%8B%E5%8C%96&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，则由编译器[初始化](https://www.baidu.com/s?wd=%E5%88%9D%E5%A7%8B%E5%8C%96&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)为0。局部变量的值[不可知](https://www.baidu.com/s?wd=%E4%B8%8D%E5%8F%AF%E7%9F%A5&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)。

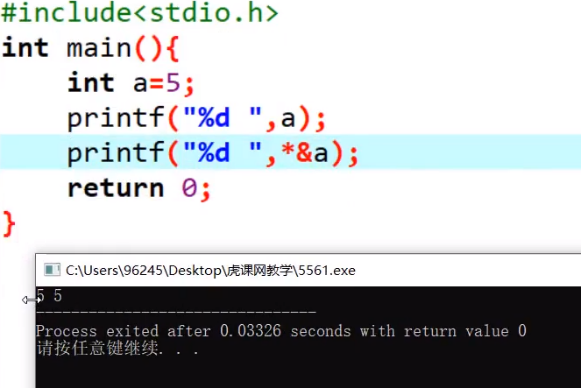
5、**静态局部变量**与**全局变量**共享全局数据区，**但静态局部变量只在定义它的函数中可见**。静态局部变量与局部变量在存储位置上不同，使得其存在的时限也不同，导致对这两者操作 的运行结果也不同。

**（该部分从https://blog.csdn.net/qq\_39736982/article/details/82685277引用）**

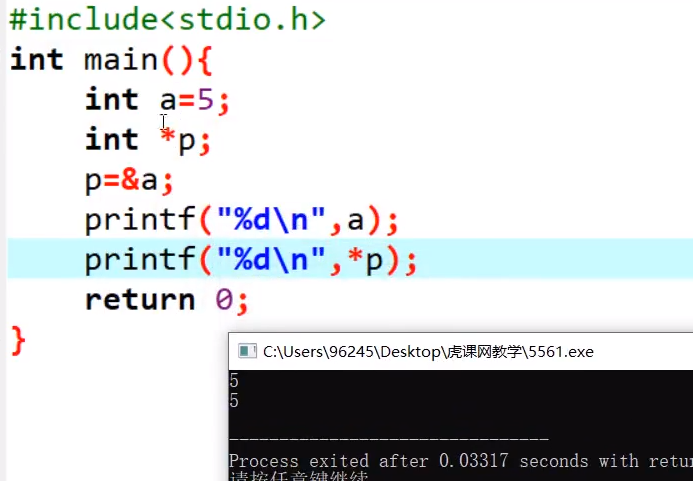
**3.指针（含义：1、指向的地方，也就是地址2、指向的数据类型，决定了长度）**

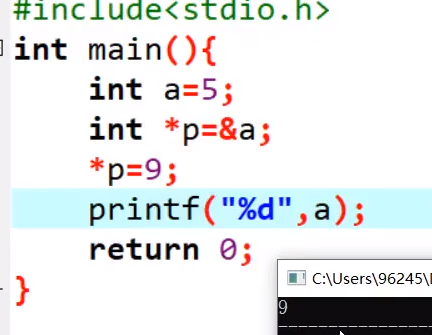
**指向变量的指针**

**指针变量的定义**：例int \*p（变量p是一个指针变量，这个指针指向一个整型变量）

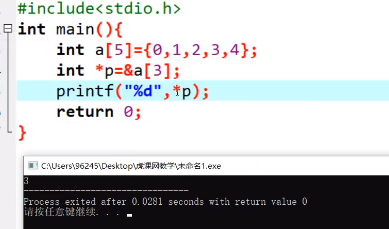
指针变量的指针运算符是\*（\*变量的地址==变量如\*&a==a）如

指针变量的间接访问则是先定义一个指针变量

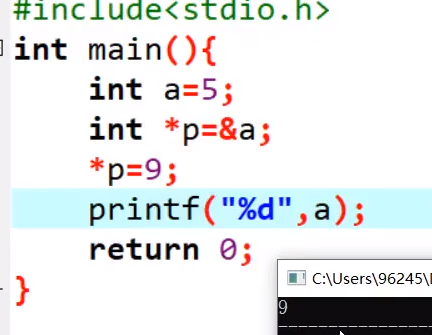
在指定该变量与何相等如

同样我们也能通过指针对变量进行修改如

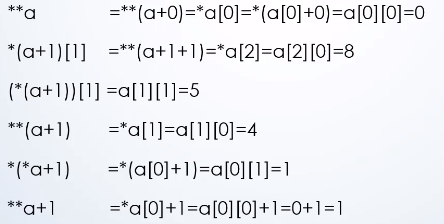
**指向数组的指针和指针数组**

可以通过指针指定某一数组如

同时一维数组的数组名代表数组中的第一个元素的地址

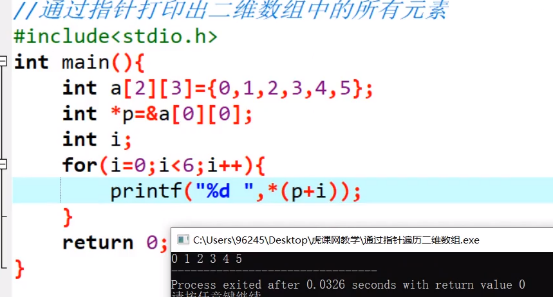
如 int \*p=&a[0]=a;

故因此\*(p+1)=&a[1]即\*(p+i）=&a[i]

而根据运算符的优先级有以下计算式

总体而言即是有两式概括\*(a+i)=a[i],\*(a[i]+j)=a[i][j]

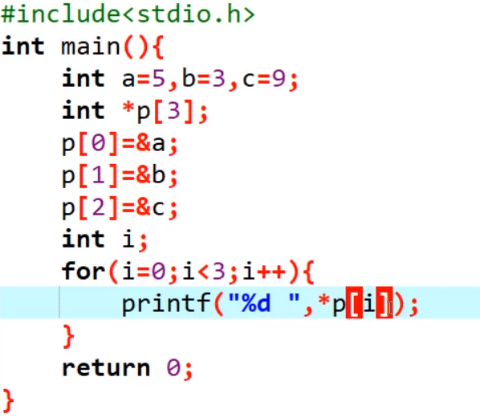
因此以指针指向多维数组即可以如int \*p=&a[0][0]||int \*p=a[0]表示

故可以用指针打印出二维数组中元素如

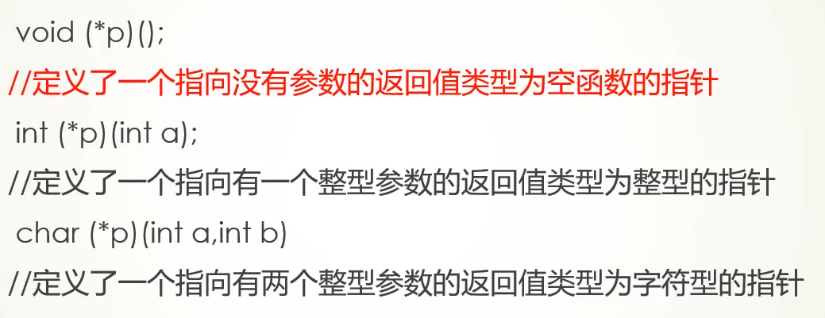
**指针数组**

指针数组的表现形式即为如int \*p[3]

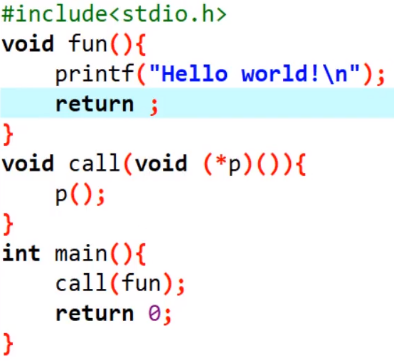
则指针数组的赋值可以为

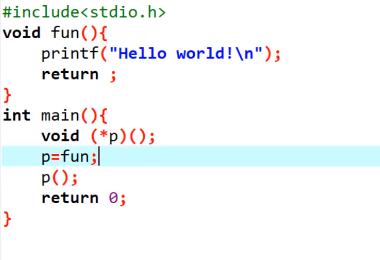
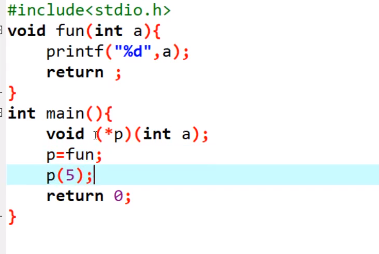
如图

**指向函数的指针和返回值是指针的函数**

在使用时应知道函数在内存中的入口地址就是函数的指针

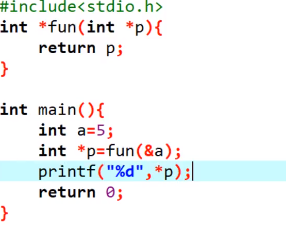
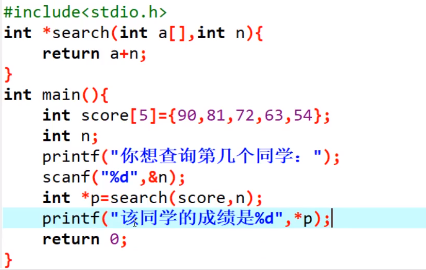
如



那么指向函数的指针变量的使用方式即可为

如图

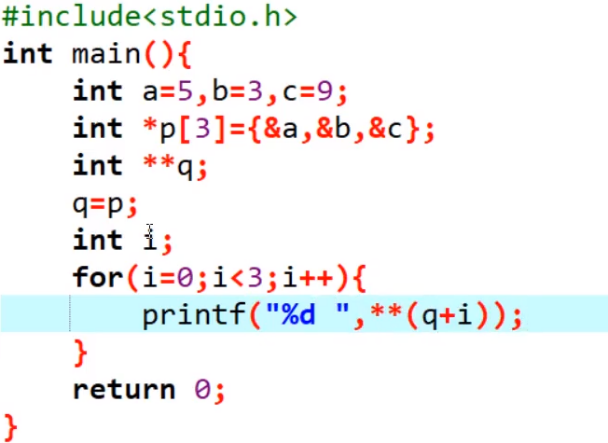
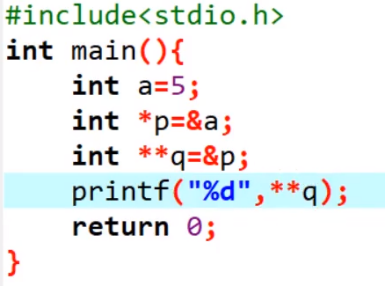
而返回值是指针的函数表现形式如下

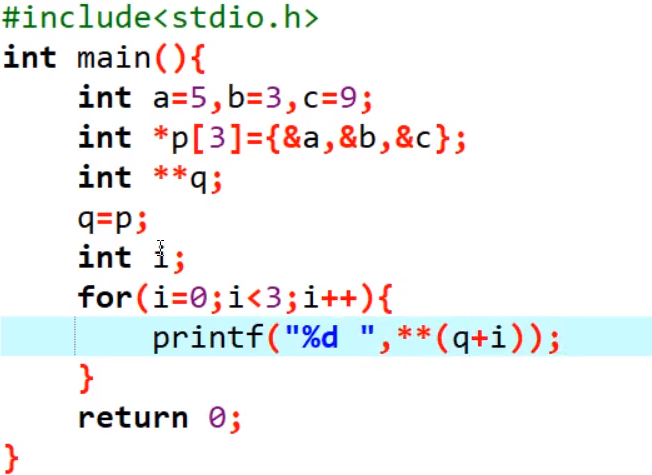
Int \*fun(int \*p){  
 return p;

}

则使用方式如图

**指向指针的指针**（表现形式如\*\*p）

使用方式如图



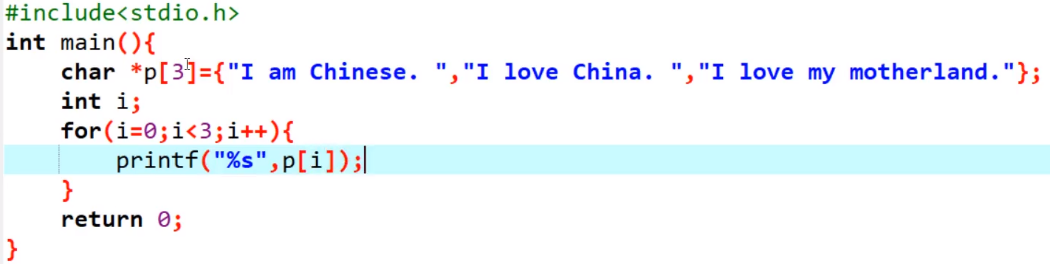
同时注意右图中\*\*(q+i)==\*q[i]

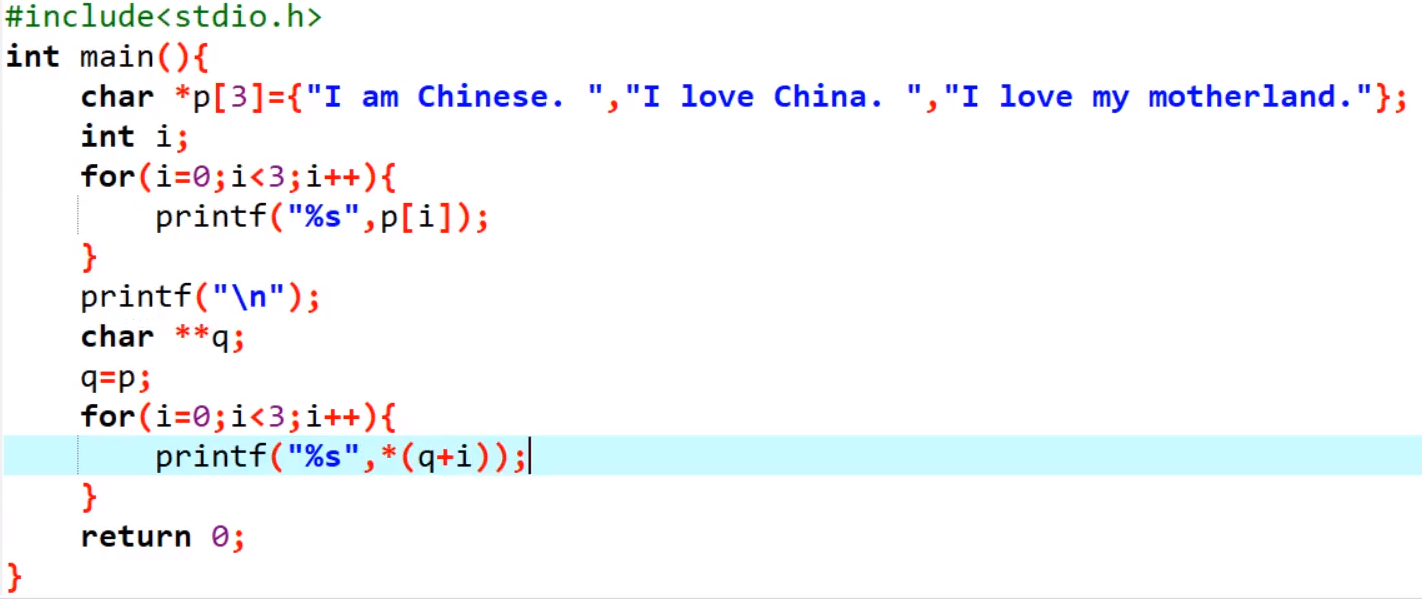
故有一下运算式：

\*\*a=\*a[0]=a[0][0]

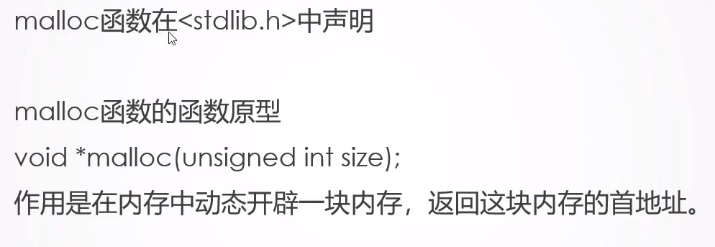
a[0]=&a[0][0]

a=&a[0]

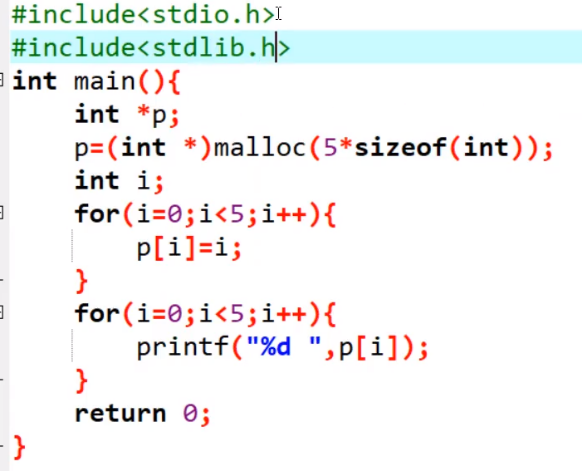
而指向字符串同样可以使用指针如图



**动态内存分配 （即为指针分配一块内存空）**

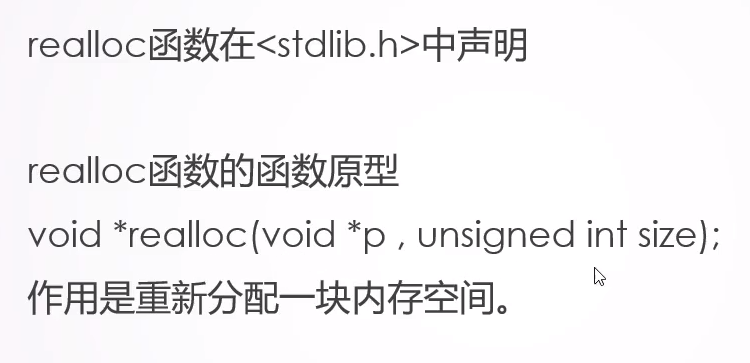
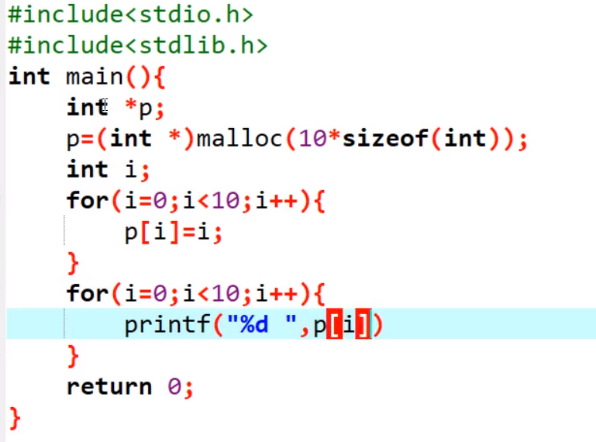
而要达成这种目的我们有三种函数（malloc，realloc，calloc，均在<stdlib.h>中声明）

Void\*类型（void\*表示指向空类型的指针，空类型的长度为0，故q=q+1没有意义）

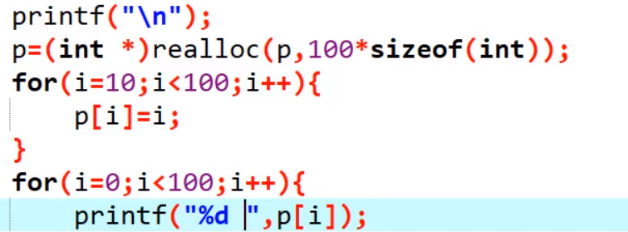
malloc函数的使用如图

realloc函数

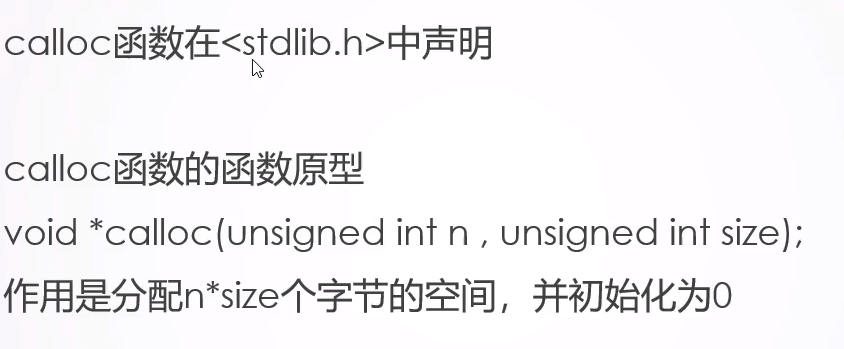
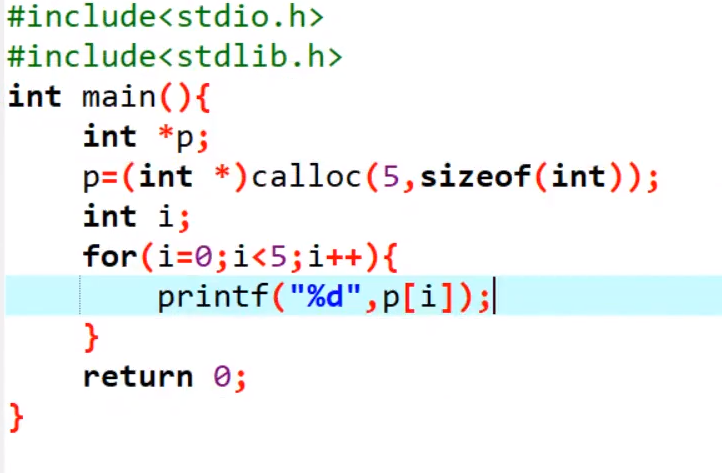
如图



使用如图



calloc函数



使用如图

由于对指针没进行过不知道指针具体用途无法说明，而且没太看懂指针，emm多包涵

o(╥﹏╥)o