记得以前经常搞不清楚 char (*p)[]跟 char *p[]，于是就写了以下的测试程序。

直观理解：

1. char（*p）[10] 是指向数组的指针。

2. char p [10]是一个数组 p。

3. char *p [10] 是指针数组，数组元素是 char *指针。

具体的不同点：

1.指向数组的指针

char（*p）[10] 中，p 指向的是一个 10 个 char 大小的数组，p+1 和 p 的差距是 sizeof（char p [10]），p+1 已指到另一个相同大小的数组了。

2.数组指针

char p[10] 中，p 也是一个指针，不过是常量指针，指向的是数组的第一个元素的地址（不是数组的首地址），见如下代码：

```
#include <stdio.h>
int main()
{
char (*p)[10] ;
char a[10];
p=a;
}
```

为什么提示

[root@localhost ~]# cc test.c

test.c: 在函数‘main’中：

test.c:6: 警告：从不兼容的指针类型赋值呢?

你把 p=a 改成 p=&a 就可以了。

这是 c 语言迷惑人的地方。很多书都告诉大家数组名就是数组的地址，但其实它是的数组第一个元素的地址，类型是 char *。而&a 表示的才是数组的地址，类型是 char (*)[10]。由于数组的地址在值上面和数组第一个元素地址的值是相同的，所以大家平时就认为&a 和 a 是一样的，产生了误会。

所以，p=a 的警告在于你把一个 char 型的地址赋给了一个 char (*)[10]的指针。当然，这样做的结果没错，不过多了个警告。

3.指针数组

char * p[10],表示的是一个数组，数组中每个元素都是指针，这里 sizeof(p)在 32 位机器上应该是 40，=10 * sizeof(char*)，这儿是数组的长度，而不是指针的长度，也说明他是个数组，跟指向指针的指针（char **p）也是不一样的

此外：

```
char *p;
char a[10];
p = a;
```

这里 p = a 的意思是 p 指向 a[0]的地址，p 指向的是一个 char。

若改成 p = &a ，则表示 p 指向数组 a，p 此时指向的是一个 a[10]数组！

p 仅仅是指向一个元素，而&a 是整个数组的地址（&a 实际上是指向数组 a 的指针），一个元素是不能指向整个数组的，因为它们类型不一致，编译器会警告！！！

虽然编译时会提醒警告：不兼容的指针类型，但还是能运行，实际上这是不对的。

正确的代码应该是：

```
/*----------------------------
看后写出如下程序，正确得到结果
----------------------------*/
#include <stdio.h>
#include <string.h>
int main()
{
char (*p)[12] ;
char a[12];
strcpy(a,"hello,world");
p=&a;
printf("p=%s\n ","the second char in p is %c\n",*p,(*p)[1]);
return 0;

}
```

char p[] = "hello,world"; 是对字符串"hello,world"做了拷贝，所以可以对拷贝字符串做任意修改

char *p = "hello,world"; 字符串常量将自己在静态存储区中的地址赋给了 p 指针

char (*p)[SIZE]：指向一维数组的指针，一维数组只能有 SIZE 个元素。代码如下：

```
#include <stdio.h>
#define TESTSIZE 20
int main(void)
{
    char szTest[][TESTSIZE] = {"hello", "world"};
    char (*p)[TESTSIZE];

    p = szTest;
    for(int i = 0; i < sizeof(szTEST)/TESTSIZE; i+
    {
        printf("%s", p + i);
    }
}
```

char *p[SIZE]：指针数组，数组有 SIZE 个元素。代码如下：

```
#include <stdio.h>
#define STRSIZE 20

int main(void)
{
     char *p[] = {"hello", "world!"};
     for(int i = 0; i < 2; i++)
         printf("%s ", p[i]);

     return 0;
}
```