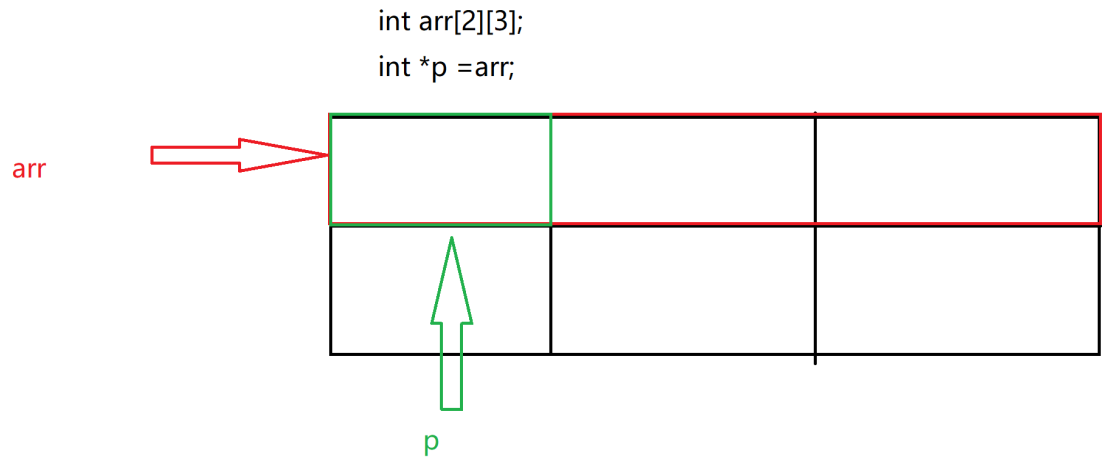
一、指针和二维数组

两个运算符：

1）&：取地址，表示取变量的地址，也可以理解为升维操作。

2）*：解引用，可以理解为降维操作。




```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int arr[2][3] = {1,2,3,4,5,6};
    int *p = arr;
    //二维数组的数组名保存的是首元素的地址，是一个行指针。
    //操作的空间是一行元素
    printf("%p\n",arr); //是行指针，指向第一行
    printf("%p\n",*arr); //是列指针，指向第一行的第一个元素
    printf("%p\n",arr+1); //是行指针，指向第二行
    printf("%p\n",*arr+1); //是列指针，指向第一行的第二个元素
    printf("%d\n",**arr); //是int类型的数据，是二维数组中第一行第一个元素
    printf("%d\n",*(*arr+0)); //是int类型的数据，是二维数组中第一行第一个元素
    //二维数组的数组名是一个行指针，操作一行元素，操作的空间超出基本数据类型了
    //所以不能用普通的指针存储二维数组
    //arr[i] <===>*(arr+i)
    /*printf("%p\n",arr[1]);
    printf("%p\n",*(arr+1));
    printf("%p\n",arr[0]);
    printf("%p\n",*(arr+0));*/
    printf("%p\n",arr+1); //偏移到下一行
    printf("%p\n",p+1); //偏移到下一个元素
    int i,j;
    for (i=0;i<2;i++)
    {
        for (j=0;j<3;j++)
        {
            printf("%d\n",*(*(arr+i)+j));
        }
    }
    return 0;
}
```

二、数组指针

一般用来存储二维数组

定义

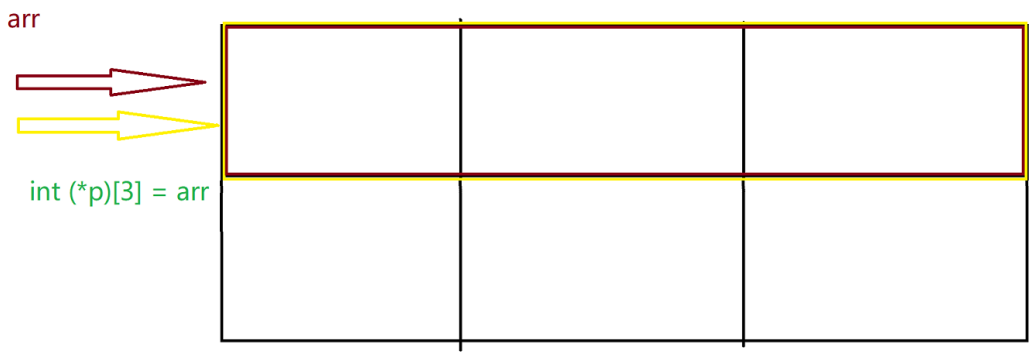
```
数据类型  (*指针名) [数组长度];
```

使用数组指针存储二维数组

```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int arr[2][3] = {1,2,3,4,5,6};
    int (*p)[3] = arr; //p+1操作的是一行元素,定义了一个数组指针保存二维数组

    int i,j;
    for (i=0;i<2;i++)
    {
        for (j=0;j<3;j++) //通过二维数组的数组名和数组指针访问二维数组中的元素
        {
            //printf("%d\n",arr[i][j]);
            //printf("%d\n",p[i][j]);
            //printf("%d\n",*(*(arr+i)+j));
            //printf("%d\n",*(*(p+i)+j));
            //printf("%d\n",*(arr[i]+j));
            printf("%d\n",*(p[i]+j));
        }
    }
    printf("arr=%p  p=%p\n",arr,p);
    printf("arr+1=%p  p+1=%p\n",arr+1,p+1);

    return 0;
}
```



```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int arr[5] ={0};
    printf("%p\n",arr); //是一个列指针
    printf("%p\n",&arr); //arr升维成行指针

    return 0;
}
```

不对数组名取地址的原因。

```c
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int arr[5] ={0};
    printf("%p\n",arr); //是一个列指针
    printf("%p\n",&arr+1); //p为arr升维后的行指针

    //此时访问arr+1就越界了，所以一般不对数组名取地址
    printf("%d\n",&(arr+1);
    return 0;
}
```

```
ubuntu@ubuntu:~/22081/day9$ ./a.out
0x7ffeb48fabb0
0x7ffeb48fabc4
ubuntu@ubuntu:~/22081/day9$ vim 7.c
ubuntu@ubuntu:~/22081/day9$ vim 5.c
ubuntu@ubuntu:~/22081/day9$ gcc 5.c
ubuntu@ubuntu:~/22081/day9$ ./a.out
0x7ffdea1ef6a0
0x7ffdea1ef6b8
```

第一二行输出分别为，输出一维数组的数组名，和输出对一维数组的数组名&后，+1的结果。

第三四行输出分别为，输出二维数组的数组名，和输出对二维数组的数组名&后，+1的结果。

三、指针数组

是一个数组，里面存的是指针。

定义

```
数据类型 指针数组名[数组长度];
int *p[3];
```

```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    char s1[][20]={"zhangsan",
                   "lisi",
                   "shiwudhasknabsakdgs",
                   "zhangsan"
                  };
    char *p1 = "zhangsan";//
    char *p2 = "lisi";
    char *p3 = "shiwudhasknabsakdgs";
    char *p4 = "zhangsan";
    char *p5 = "zhangsan";

    /*printf("%p\n",p1);
    printf("%p\n",p4); //输出p1和p4，指向的是同一块地址，因为他们指向同一个字符串常量
    printf("%p\n",p5);
    printf("%p\n",&"zhangsan");*/

    char *p[5] = {p1,p2,p3,p4};
    //printf("%s\n",p[0]);
    return 0;
}
```

指针数组用于命令行传参

```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    //argc表示的是main外部传参的个数
    //argv是一个指针数组，保存命令行传参的内容
    //./a.out a b c d
    printf("%d\n",argc); //5
    printf("%s\n",argv[0]); //./a.out
    printf("%s\n",argv[1]); // a
    printf("%s\n",argv[2]); //b
    printf("%s\n",argv[3]); //c
    return 0;
}
```

用命令行传参的方法实现建议的计算器功能

```
#include <stdio.h>
int main(int argc, const char *argv[])
{
    int a = *(argv[1]) - '0';  //求出字符对应的整形
    int b = *(argv[3]) - '0';
    if (*(argv[2]) == '+')   //如果是+
    {
        printf("%d\n",a+b);
```

```
    }
    else if(*(argv[2]) == '-')
    {
        printf("%d\n",a-b);
    }
    return 0;
}
```

四、函数

实现特定功能的代码块

定义

```
返回值类型 函数名(参数列表)
{
    实现功能的代码块;
    return; //根据返回值类型来加的，如果返回值类型为void不需要加return
}
```

返回值类型，如果不需要返回值，可以写void

char int …

```
#include <stdio.h>
int add(int,int);
void add2(int c,int d)
{
    printf("%d\n",c+d);
}
int main(int argc, const char *argv[])
{
    int x = 0;
    int y = 0;
    add(x,y);
    return 0;
}
int add(int a,int b)
{
    return a+b;
}
```

函数的调用

```
函数名(实参);
```

函数声明

作用：

告诉计算机，我定义过这个函数了，并且还要告诉计算机，我这个函数需要什么类型的参数。

```
返回值类型 函数名(参数的数据类型~~);
```