C语言——面向过程编程

单元二、指针与数组

第四次课 二维数组与指针

一. 敏锐识别二维数组

二维数组的特性：【长度固定】、【同类型】、【有编号（先行后列）】

二. 二维数组的基本用法

① 二维数组的定义：数组名 [行数量] [列数量]

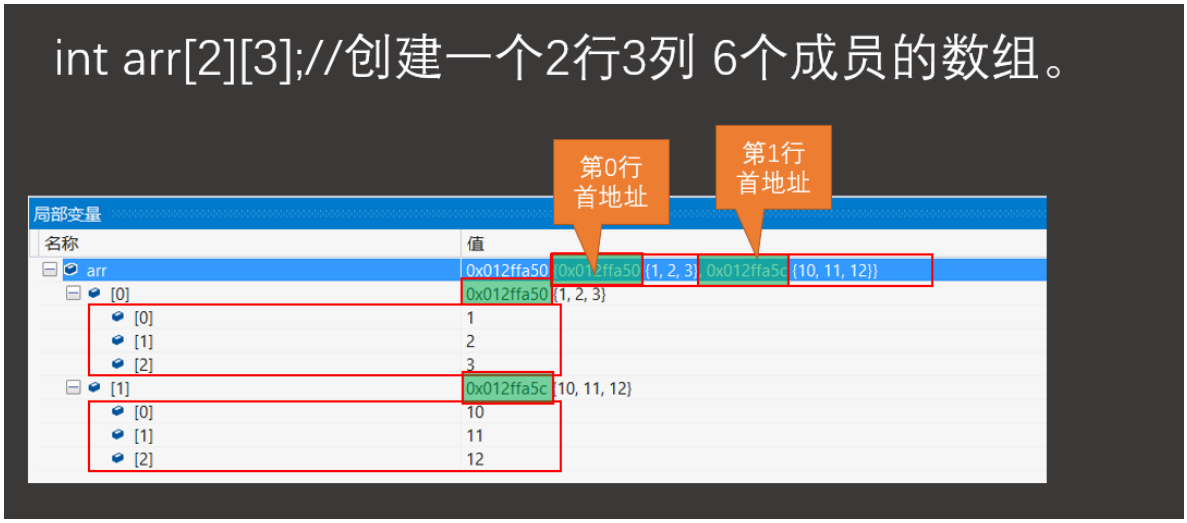
```
int arr[2][3];//创建一个2行3列 6个成员的数组。
```

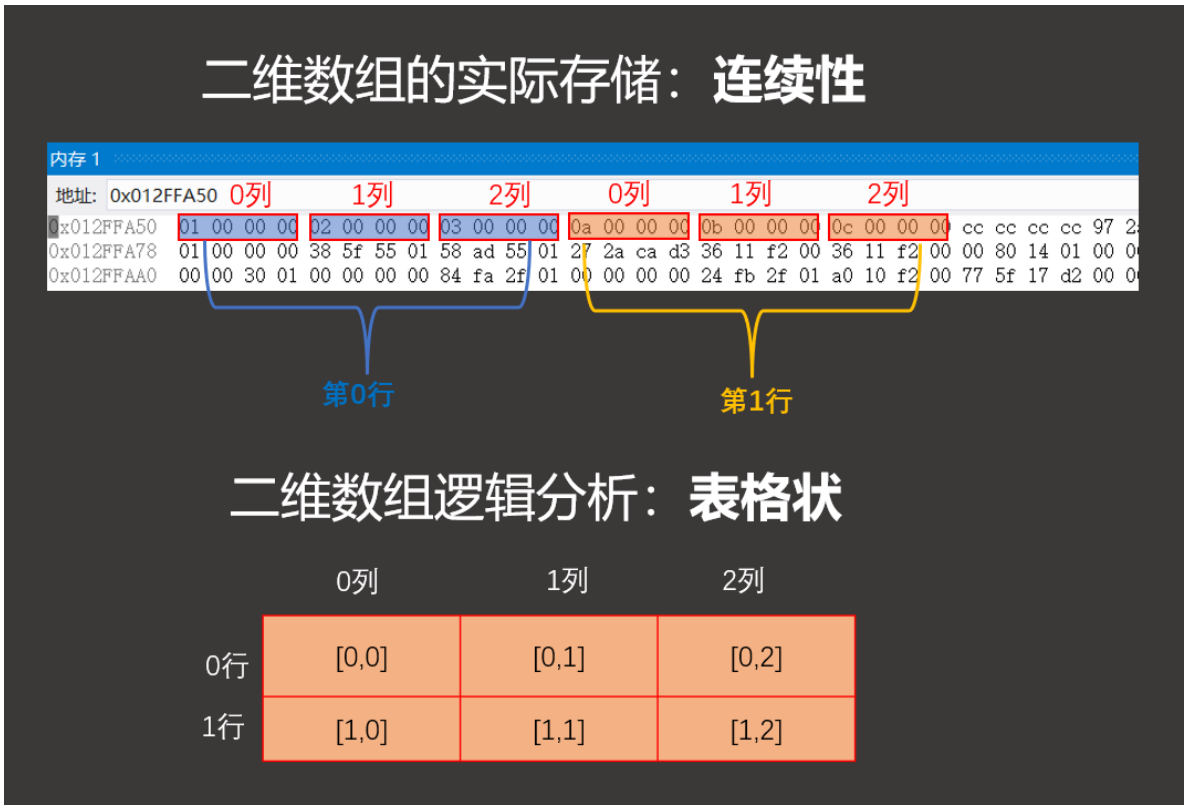
②数组成员的使用：数组名 [行号] [列号]

```
arr[0][0] = 1;
arr[0][1] = 2;
arr[0][2] = 3;

arr[1][0] = 0xa;
arr[1][1] = 0xb;
arr[1][2] = 0xc;
```

③ 二维数组的存储





也可以把二维数组理解为：**一维数组的一维数组**

④ 注意：不要下标越界访问数组。

⑤ 数组成员遍历

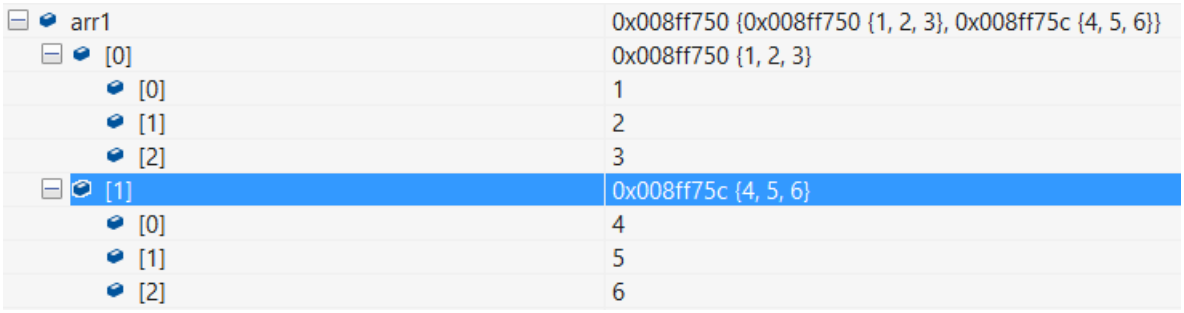
```
int arr[2][3];
int h,l;
int hLen = sizeof(arr)/sizeof(arr[0]);//计算二维数组有几行
int lLen =sizeof(arr[0])/sizeof(arr[0][0]) ;//计算二维数组有几列
//遍历数组每个成员
for( h=0 ; h<hLen ; h++ )
{
    for(l=0;l<lLen;l++)
    {
        printf("%d  ",arr[h][l]);//不换行打印某行的3个成员
    }
    printf("\n");//一行结束，打印一个换行
}
```

三. 二维数组初始化

①：完全初始化

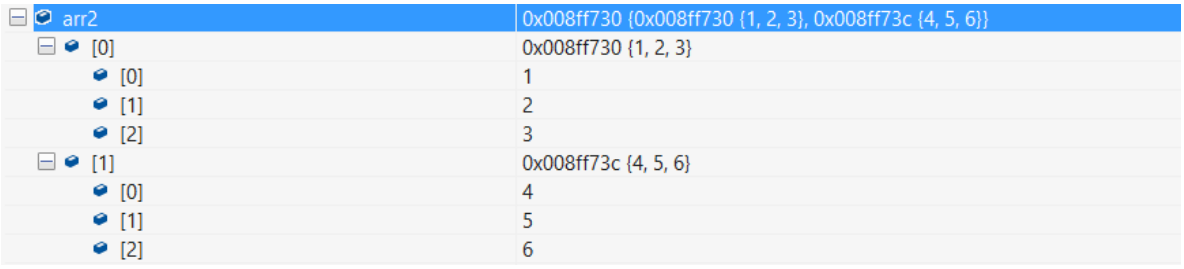
形式一：如同一维数组形式进行初始化赋值

```
int arr1[2][3] = { 1,2,3,4,5,6 };//会根据行列数量存入对应的空间位置
```



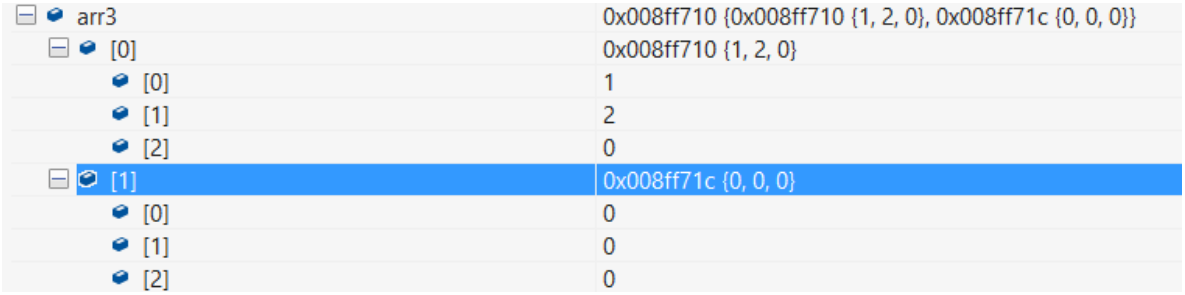
形式二：用{} 区分不同的行

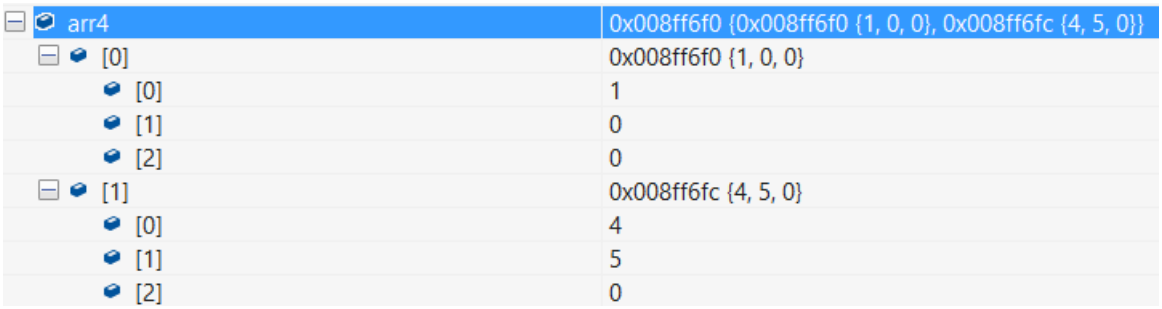
```
int arr2[2][3] = { {1,2,3},{4,5,6} };//每个内部的一对{} 圈起来一行
```



② 不完全初始化、其它成员默认 0

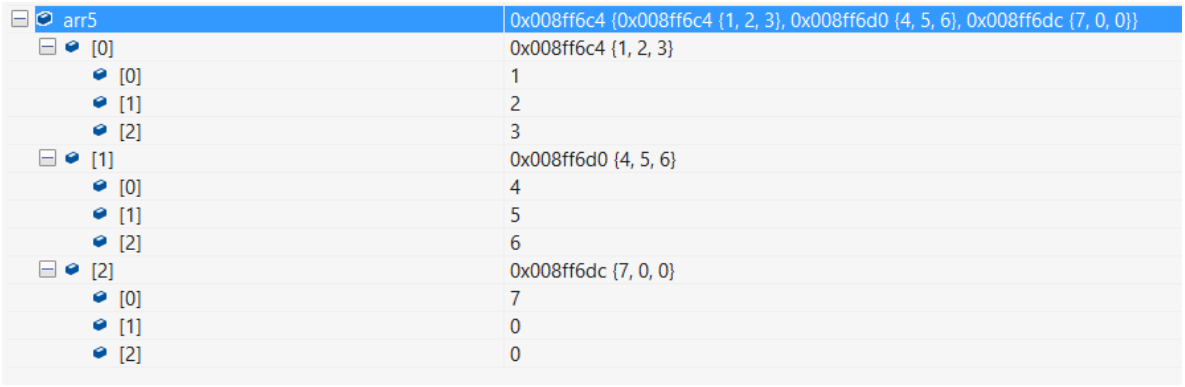
```
int arr3[2][3] = { 1,2  };
int arr4[2][3] = { {1 },{4,5 } };
```

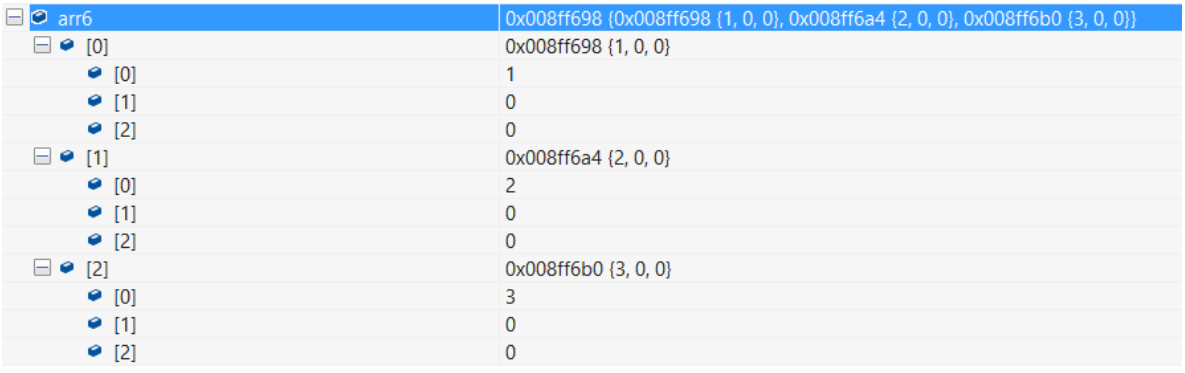




③根据成员初始情况自动计算行数：但——列数不能省略

```
int arr5[ ][3] = { 1,2,3,4,5,6 ,7  };//会根据列的数量 计算行数 即 3行
int arr6[ ][3] = { {1},{2},{3} };
```





四. 二维数组与指针

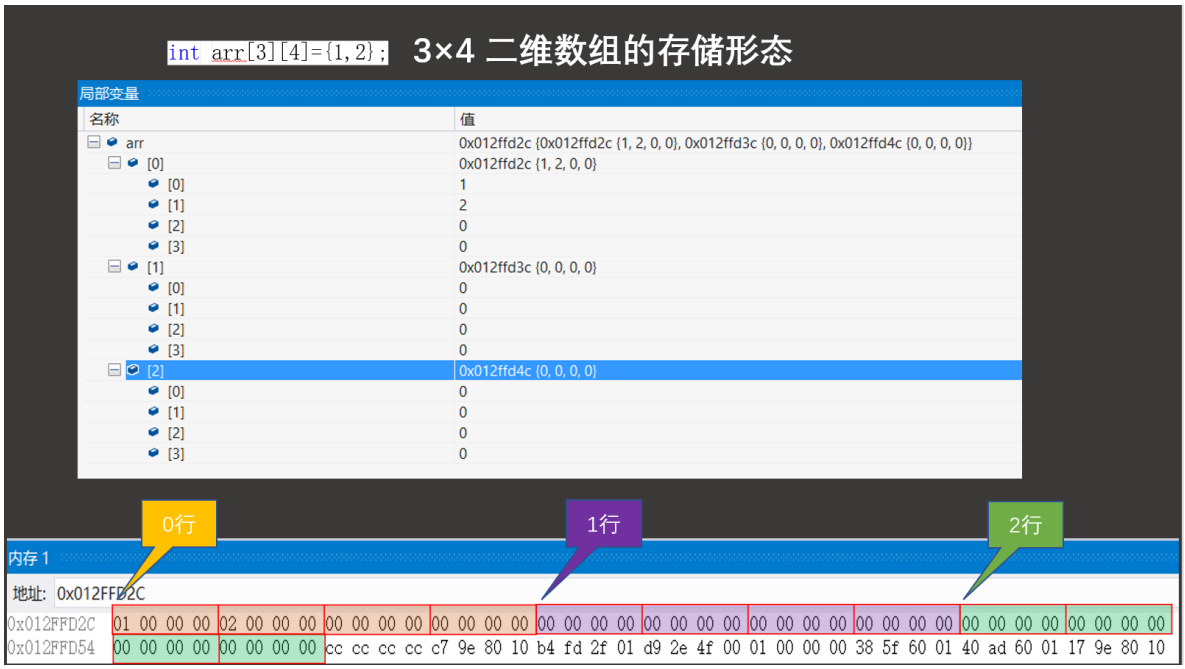
① 用int指针 遍历数组元素,证明二维数组在内存中是连续的

```c
int arr[3][4]={1,2};
int *p=(int*)&arr;//arr为二维数组名， 和int*类型不一致。但是都表示内存地址，因此可以用强制类型转换消除编译错误。
int i;
for(i=0;i<3*4;i++)//让 i循环12次，如果存储空间连续，那么p就可以遍历每一个二维数组的成员。
{
    printf("%d,",*p);
    p++;
}
```

② 利用二维数组名，获得不同的行的首地址 及 不同列 对应数组成员的地址

首先 定义一个int型二维数组。

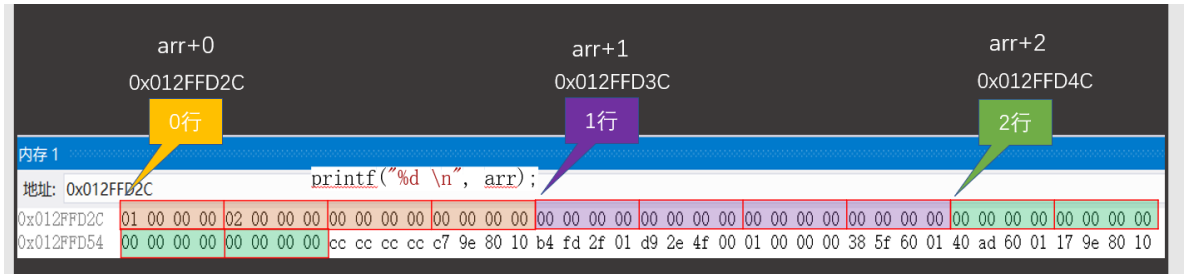
```c
int arr[3][4]={1,2};//数组的存储形态参考下图：
```



【注意】不同写法代表不同的含义

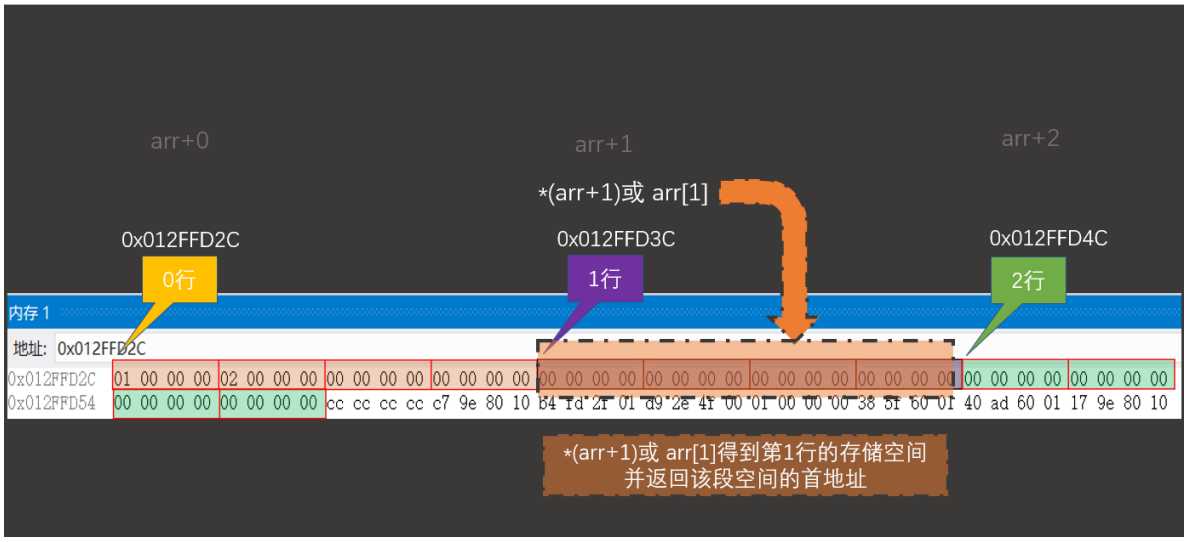
A：二维数组名+整数 得到 某行的首地址

```c
printf("%d \n", arr);//得到第0行首地址
printf("%d \n", arr+1);//得到第1行首地址
printf("%d \n", arr+2);//得到第2行首地址
```



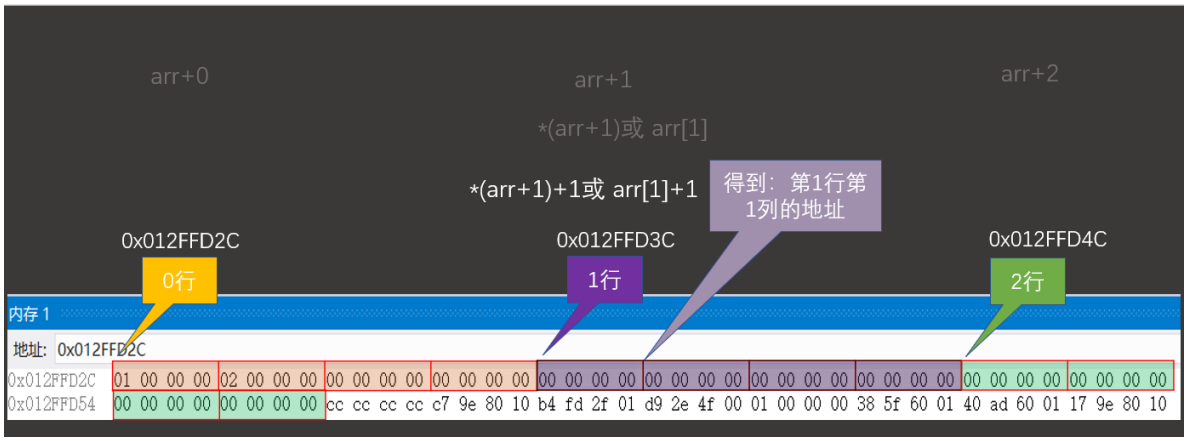
B: * (二维数组名+行号) 或 二维数组名[行号] 得到 某行的存储空间

```
printf("%d    %d\n", *(arr+1),arr[1]);//得到第1行存储空间
```



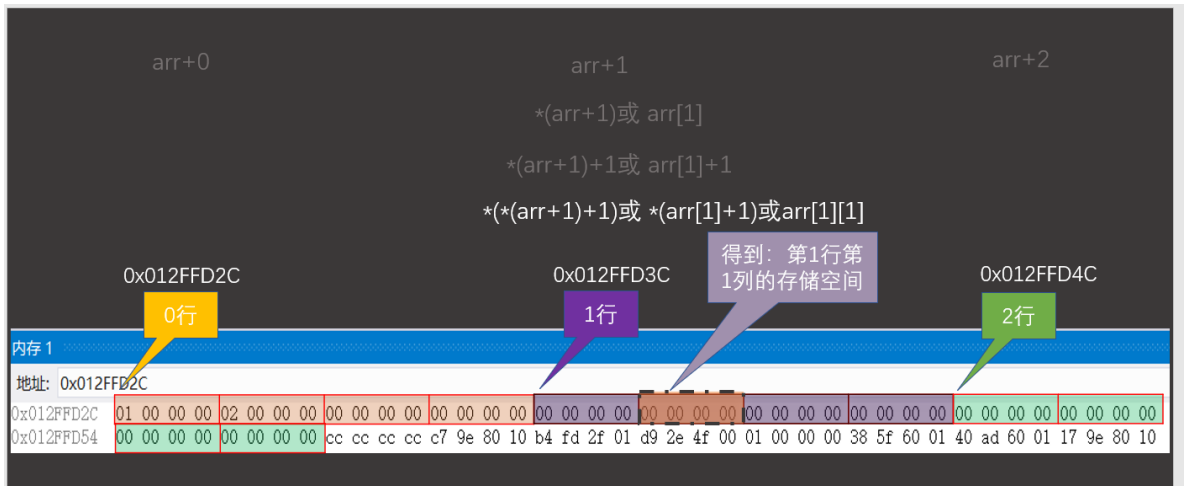
C： *(二维数组名+行号)+列号： 得到某行 内 某列的地址

```
printf("%d    %d\n", *(arr+1)+1,arr[1]+1);//在第1行的存储空间里，再移动一个int型的空间
```



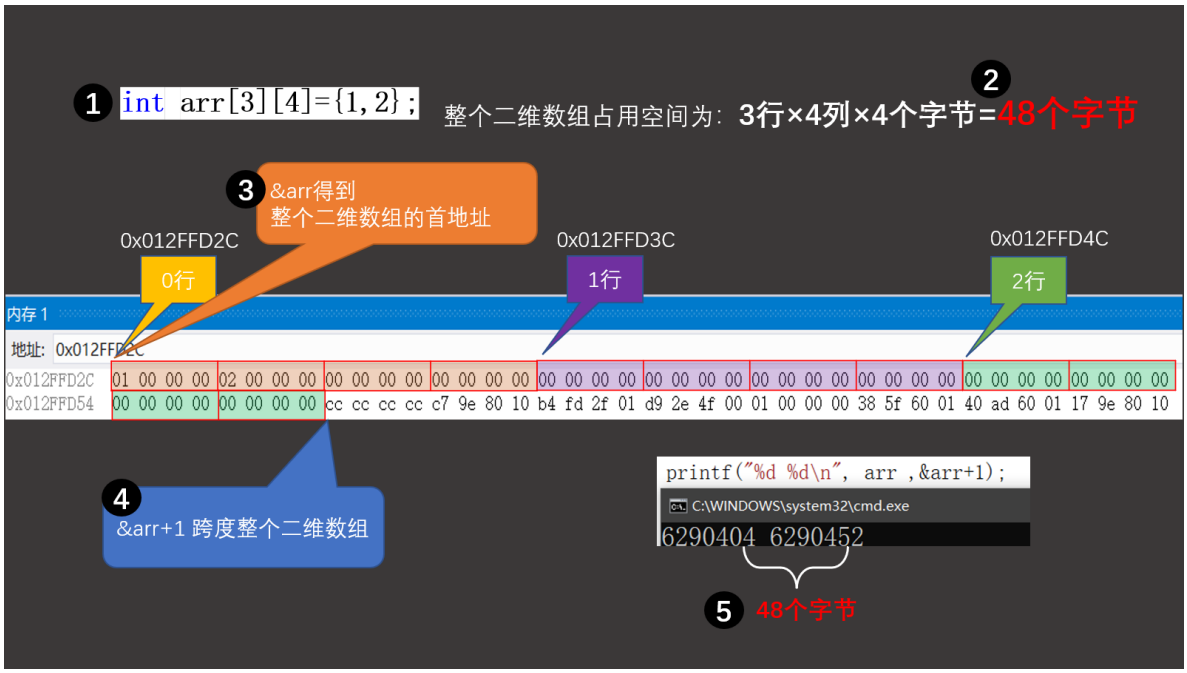
D： *（*(二维数组名+行号)+列号）： 得到某行 某列的存储空间。

```
printf("%d, %d,%d\n", *(*(arr+1)+1),*(arr[1]+1), arr[1][1]);//得到某行某列的存储空间
```



E： &二维数组+1 将跨度整个二维数组的字节数

```
int arr[3][4]={1,2};
printf("%d %d\n", arr ,&arr+1); //&arr得到整个二维数组的首地址。 &arr+1将跨度整个二维数组的存储空间
```



③定义一个*一维数组指针*遍历每一行

```
int arr[3][4]={1,2};
int (*p)[4] = NULL;//定义一个一维数组指针，其可以指向二维数组的某行。
p=(int (*)[4])&arr ;//让p指向二维数组的第0行。
int i,j;
for(i=0;i<3;i++)//循环3次，即二维数组有3行
{
    printf("%d\n",p);//打印p指向行的首地址，以此观察p是否为行的跨度移动。
    p++;    //指针指向下一行。
}
```

④.使用一维数组指针 遍历每一行 每一个成员

一维数组指针定义： **int (*指针变量名)[一维数组长度]**

```
int arr[3][4]={1,2};
int (*p)[4] = NULL;//定义一个一维数组指针，其可以指向二维数组的某行。
p=(int (*)[4])&arr;//让p指向二维数组的第0行。
for(i=0;i<3;i++)
{
    for(j=0;j<4;j++)
    {
        printf("%d，",   (*p)[j]);//下标法 :*p得到整个行的空间 (*p)[j]在整行空间里再得到某个下标j的空间
        printf("%d | ",  *( *p +j));//地址法：*p+j在整个行空间内得到第j个地址  *( *p +j))得到本行第j个指针指向的空间
    }
    printf("\n");
    p++;//指针指向下一行。
}
```

⑤定义二维数组指针 遍历每一行 每一个元素

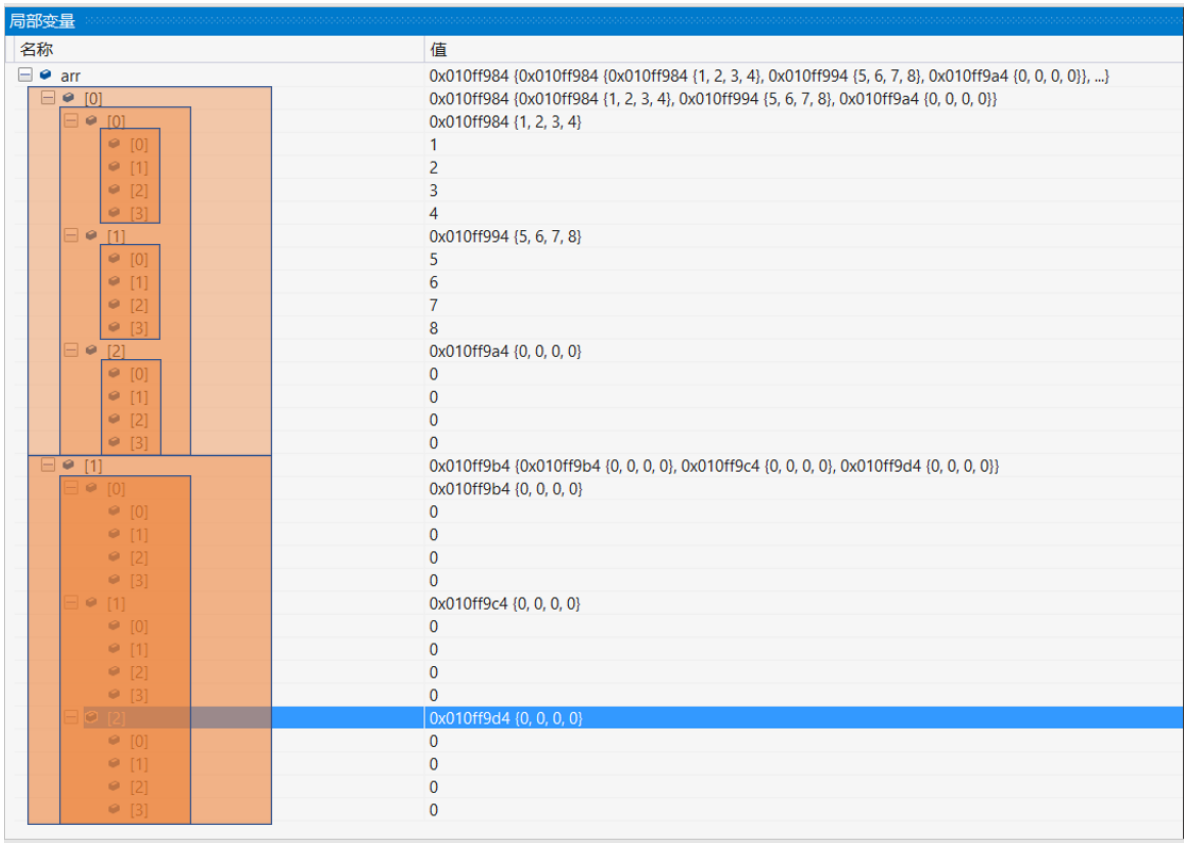
二维数组指针定义： int (*指针变量名) [行数] [列数]

```
int arr[3][4]={1,2};
int (*p)[3][4] = &arr;//定义一个能够指向3行4列的指针
int i,j;
for(i=0;i<3;i++)
{
    for(j=0;j<4;j++)
    {
        printf("%d，%d\n",(*p)[i][j],arr[i][j]);//下标法 取数据
        printf("%d，%d\n", *(*(*p+i)+j),*(*(arr+i)+j) );//地址法 取数据
    }
}
```

五. 多维数组

三维数组定义与初始化

```
int arr[2][3][4]={1,2,3,4,5,6,7,8};//三维数组的逻辑结构如下
```



各种形式的三维数组遍历

① 下标方式进行三维数组遍历

```
int arr[2][3][4]={1,2,3,4,5,6,7,8};
int i,j,k;
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        for(k=0;k<4;k++)
        {
            printf("%d,",arr[i][j][k]);
        }
    }
}
```

② 用int型的指针变量(或称 列指针) 进行三维数组遍历

```
int arr[2][3][4]={1,2,3,4,5,6,7,8};
int *p=(int*)&arr;
int i,j,k;
for(i=0;i<2*3*4;i++)
{
    printf("%d,",*p);
    p++;//指向下一个数组成员的地址
}
```

③ 用指向一维数组的指针变量（或称 行指针）进行三维数组遍历

```
int arr[2][3][4]={1,2,3,4,5,6,7,8};
int i,j,k;
int (*ph)[4] =(int (* )[4])&arr ;
for(i=0;i<2;i++)
```

```
{
    for(j=0;j<3;j++)
    {
        for(k=0;k<4;k++)
        {
            printf("%d,",   (*ph)[k]  );
        }
        ph++;//指向下一行首地址
    }
}
```

④ 用指向二维数组的指针变量（或称 平面指针）进行三维数组遍历

```
int arr[2][3][4]={1,2,3,4,5,6,7,8};
int i,j,k;
int (*pm)[3][4]=(int (* )[3][4])&arr;
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        for(k=0;k<4;k++)
        {
            printf("%d,",    (*pm)[j][k]  );
        }
    }
    pm++;//指向下一个二维平面的首地址
}
```

⑤ 用指向三维数组的指针（或称 立体指针）进行三维数组遍历

```
int arr[2][3][4]={1,2,3,4,5,6,7,8};
int i,j,k;
int (*pt)[2][3][4]= &arr;
for(i=0;i<2;i++)
{
    for(j=0;j<3;j++)
    {
        for(k=0;k<4;k++)
        {
            printf("%d,",    (*pt)[i][j][k]  );
        }
    }
}
```