C语言——面向过程编程

单元二、指针与数组

第三次课 一维数组与指针

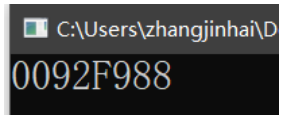
一. 一维数组的存储原理

① 数组名的真正作用：

数组名是一个不可改的地址常量，数组名代表第0个元素的地址

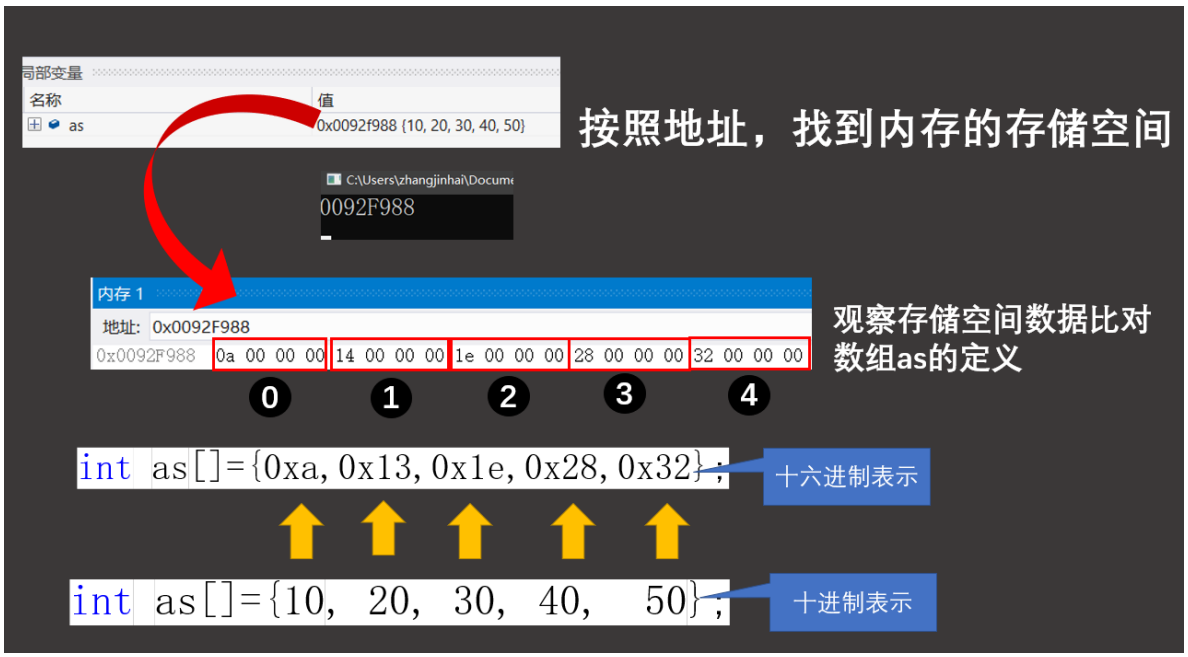
验证一下：

```
int as[]={10,20,30,40,50};
printf("%p\n",as);//打印 数组名对应的数据
```



打印结果为一个地址值：**务必理解为第0个元素的首地址** 。

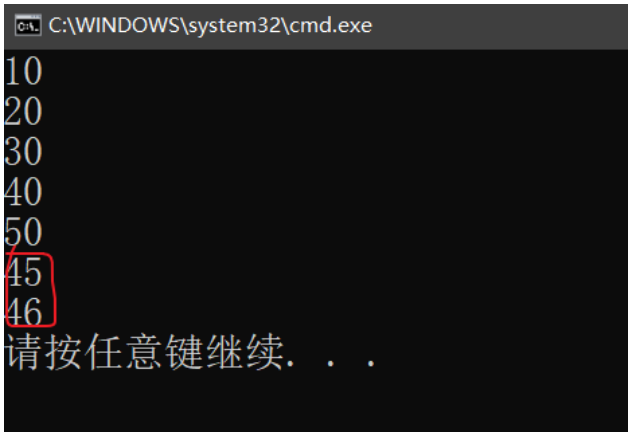
下面用调试工具验证一下：



② 数组名+偏移量操作成员空间

```
int as[]={10,20,30,40,50};
//从数组第0个元素的地址开始，依次增加偏移量产生新的地址，再用*得到地址对应的空间。
printf("%d\n",*(as+0));
printf("%d\n",*(as+1));
printf("%d\n",*(as+2));
printf("%d\n",*(as+3));
printf("%d\n",*(as+4));
//对空间数据进行自增操作。
*(as+3)+=5;

printf("%d\n",as[3]);
(*(as+3))++;
printf("%d\n",as[3]);
```

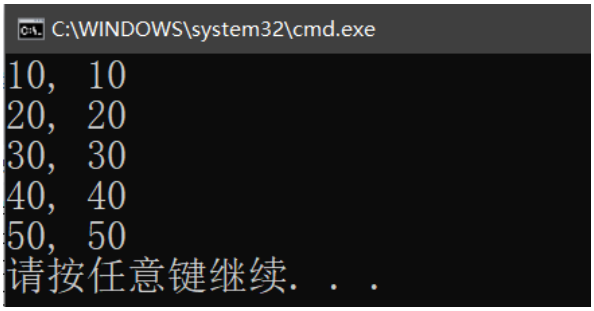


③ 深度理解[] 作为运算符的作用

数组名[下标] 等价于 ***(数组名+偏移量)**

```
int as[]={10,20,30,40,50};
int i;
for(i=0;i<5;i++)
{
    printf("%d, %d\n",as[i], *(as+i));
}
```

打印结果相同：

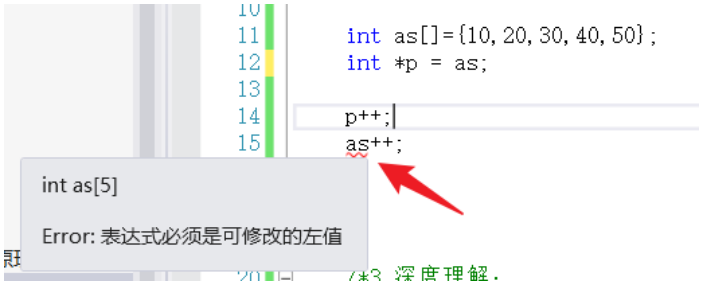


二. 指针与一维数组某元素

①：int型指针 指向int型数组 的意义

```
int as[]={10,20,30,40,50};
int *p;
p=as;//将数组名赋值给p，  就是让p 指向了数组的第0个元素。
```

②：比较 指针+1与数组名+1的区别



上图所示： **数组名是地址常量，常量即不可被改变的量。所以数组名自增是不可以的。 但p是指针变量，可以自由自增来指向另一个地址。**

③：深度理解：
数组名+[下标] 与 *(指针+移动) 与 指针+[下标] 的应用效果

```
int as[]={10,20,30,40,50};
int *p = as;
//如下4种写法都访问数据30所在的存储空间。
printf("%d\n",as[2]);//下标法
printf("%d\n",*(as+2));//地址法
printf("%d\n",p[2]);//下标法
printf("%d\n",*(p+2));//地址法
```

④：利用指针变量实现数组的便利操作

```
    int as[]={10,20,30,40,50};
    int *p = as;
    int i;
    for(i=0;i<5;i++)
    {
        printf("%d,",p[i]);//下标法
    }
    printf("\n\n");
    for(i=0;i<5;i++)
    {
        printf("%d,",*(p+i));//固定地址法
    }
    printf("\n\n");
    for(i=0;i<5;i++ )
    {
        printf("%d,",*p++);//移动地址法
    }
    printf("\n\n");
```

三. 指针类型的数组

既然指针也是一种变量，当需要很多这种变量时，就可以用数组的方式定义多个同类型的变量。

创建int型数组的代码：

```
int arr[5];
```

同理：创建int* 型的数组。 int 是基本类型，加*就是int的扩展类型即：int指针类型。

```
int* p[5];//此时，p的每个元素都是一个指针变量。
```

完整代码如下：

```
int a=1,b=2,c=3,d=4,i;
int *p[4]={&a,&b,&c,&d};//初始化指针数组成员变量的值、

for(i=0;i<4;i++)
{
    printf("%d ,",*p[i]);//p代表整个指针数组。 p[i] 找到其中一个指针变量。 *p[i]  得到指针变量指向的存储空间
}
```

四. 指向一维数组的指针

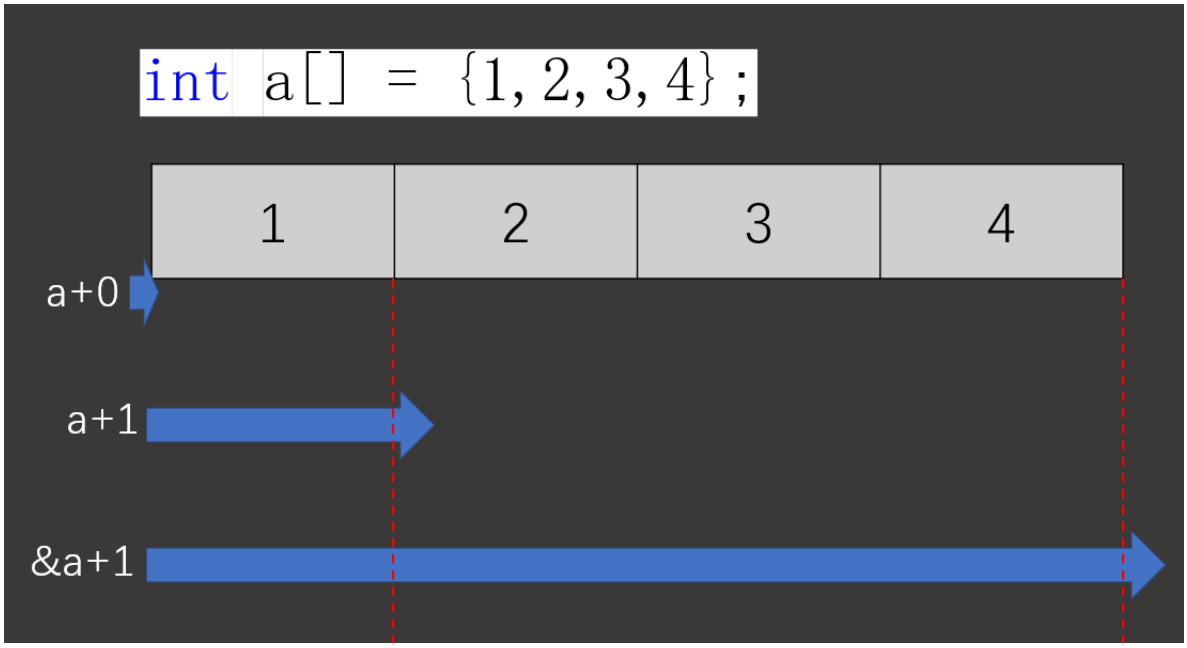
前面我们接触的指针 都是指向基本数据类型（char short int long long float double...）的。指针+1 会根据其类型会产生不同的字节跨度。如果把数组也看作是一种数据类型，那么

```
int arr[5];//所占空间为4×5 等于 20个字节空间。
```

有这样的一种指针，它可以指向一个一维数组。这样的指针+1 所产生的地址跨度为 ： 基本数据类型字节数×数组长度。

①：理解：&数组名+1的作用

```
int a[] = {1,2,3,4};
printf("%d  %d  %d\n",a,a+1, &a+1);
//a:代表数组第0个成员的地址相当于a+0
//a+1:代表第1个成员的地址
//&a: 计算整个数组存储空间后的首地址
//&a+1:代表在数组首地址基础上位移一个数组长度的距离得到一个新的地址。
```



如果需要定义这种指针变量，定义方式如下：

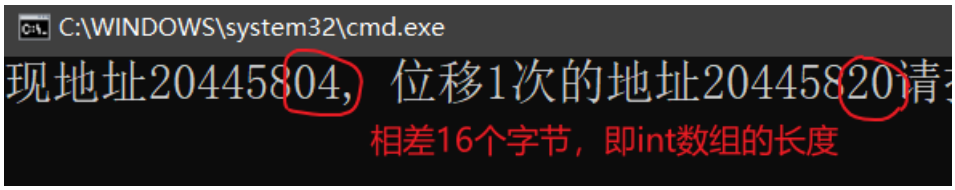
```
int (*pa)[4];//可以指向4个连续int型存储空间的地址。
```

注意：（）不能省略，如果写成：int *pa[4]的话，称为指针数组。

这种指针变量可以指向任何长度为4的int型数组。

完整代码：

```
int a[] = {1,2,3,4};
int (*pa)[4];
pa=&a;
printf("现地址%d, 位移1次的地址%d",pa,pa+1);
```



②：理解指向整个数组的指针 与 指向数组某个元素指针的区别

```
int a[] = {1,2,3,4};
int *p;
int (*pa)[4];//可以指向4个连续int型存储空间的地址。
p=a;//可以理解为 p=a+0 即指向第0个元素
pa=&a;//pa指向整个数组
printf("%d  %d\n%d %d",p,p+1, pa,pa+1);//打印不同的地址跨度
```