



第9章 指针和数组

——指针和一维数组之间的关系



哈尔滨工业大学

苏小红

sxh@hit.edu.cn

一维数组元素的引用

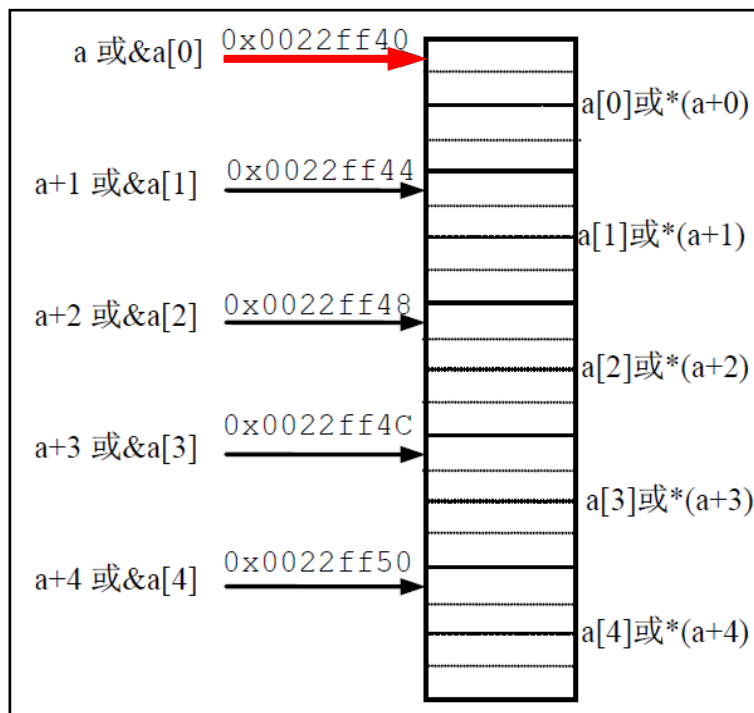
- 数组名代表数组的首地址 $\&a[0]$
 - $\&a[i] \leftrightarrow (a + i)$
 - $a+1$ 不是加上1个字节，取决于 a 的基类型
 - $a + 1 \rightarrow a + \text{sizeof}(\text{基类型})$
 - $a + i \rightarrow a + i * \text{sizeof}(\text{基类型})$

- 一维数组元素的等价引用形式

$$a[i] \leftrightarrow *(a + i)$$

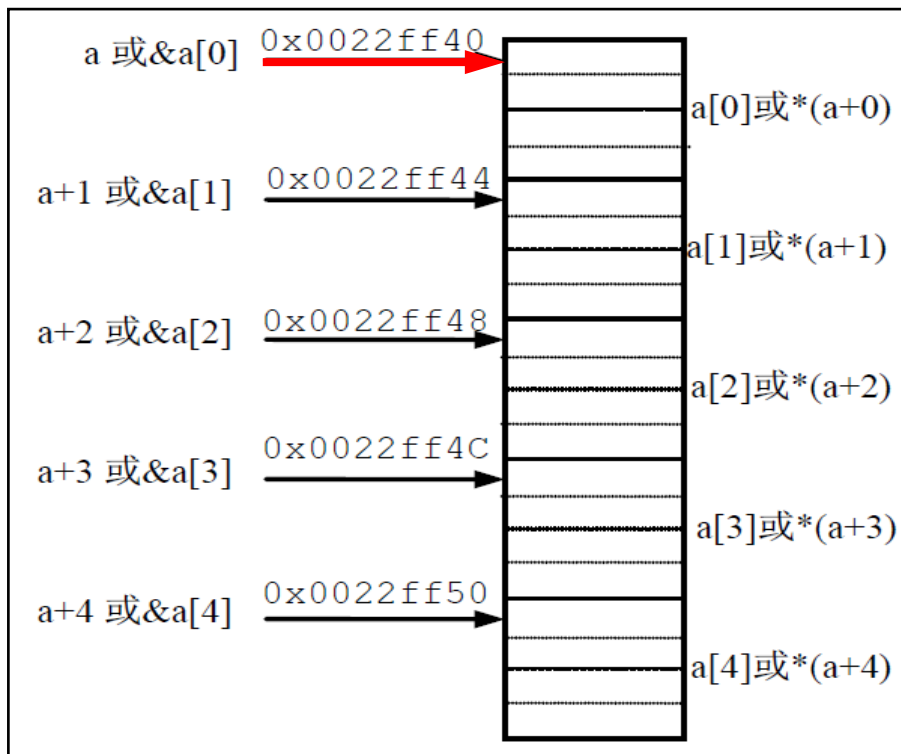
- 用下标形式访问数组元素的本质
 - 计算该元素在内存中的地址

```
int a[5];
```



指向数组的指针

- 问题：为什么一个 `int` 型指针能指向一个 `int` 型的一维数组呢？



`a[i] ↔ *(a+i)`

`&a[i] ↔ (a+i)`

```
int a[5];
```

```
int *p = a;
```

```
int *p = &a[0];
```

`&a[0]` 是 `int` 型元素的地址

`p` 是 `int` 型指针，基类型是 `int`

`p` 的基类型与它指向的元素类型相同



一维数组元素的访问

```
#include <stdio.h>
int main()
{
    int a[5], i;
    for (i=0; i<5; i++)
    {
        scanf("%d", &a[i]);
    }
    for (i=0; i<5; i++)
    {
        printf("%4d", a[i]);
    }
    printf("\n");
    return 0;
}
```

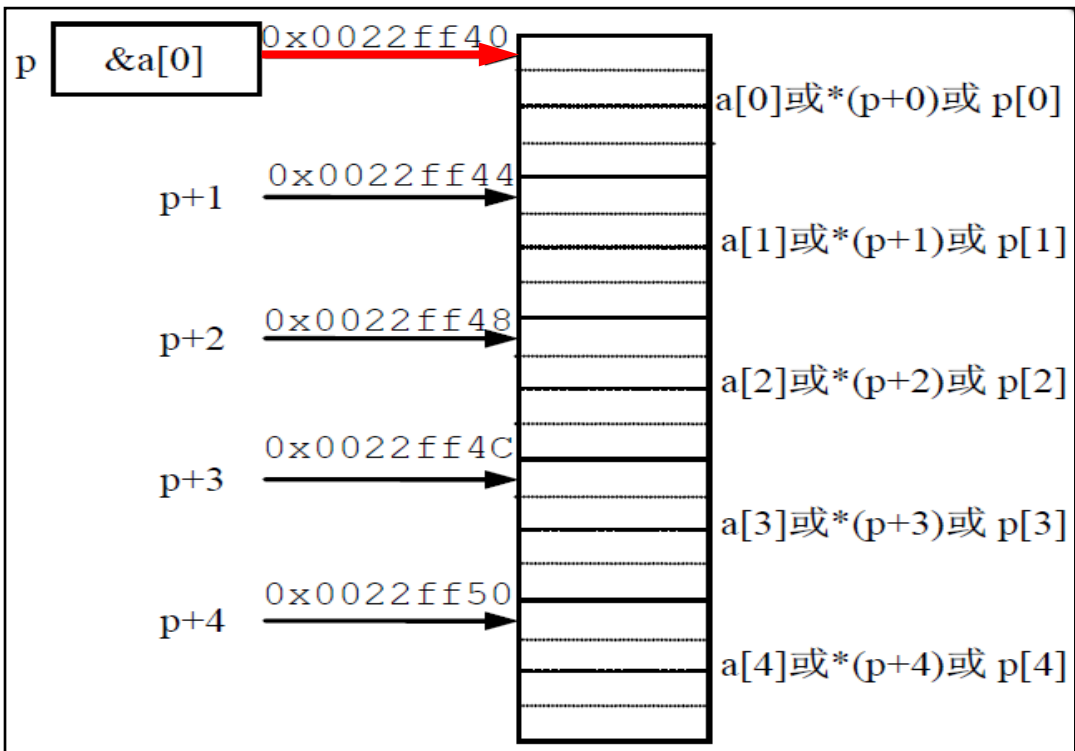
```
#include <stdio.h>
int main()
{
    int a[5], i;
    for (i=0; i<5; i++)
    {
        scanf("%d", a+i);
    }
    for (i=0; i<5; i++)
    {
        printf("%4d", *(a+i));
    }
    printf("\n");
    return 0;
}
```

$a[i] \leftrightarrow *(a+i)$

$\&a[i] \leftrightarrow (a+i)$

一维数组元素的访问

```
int a[5];      int *p = a;
```



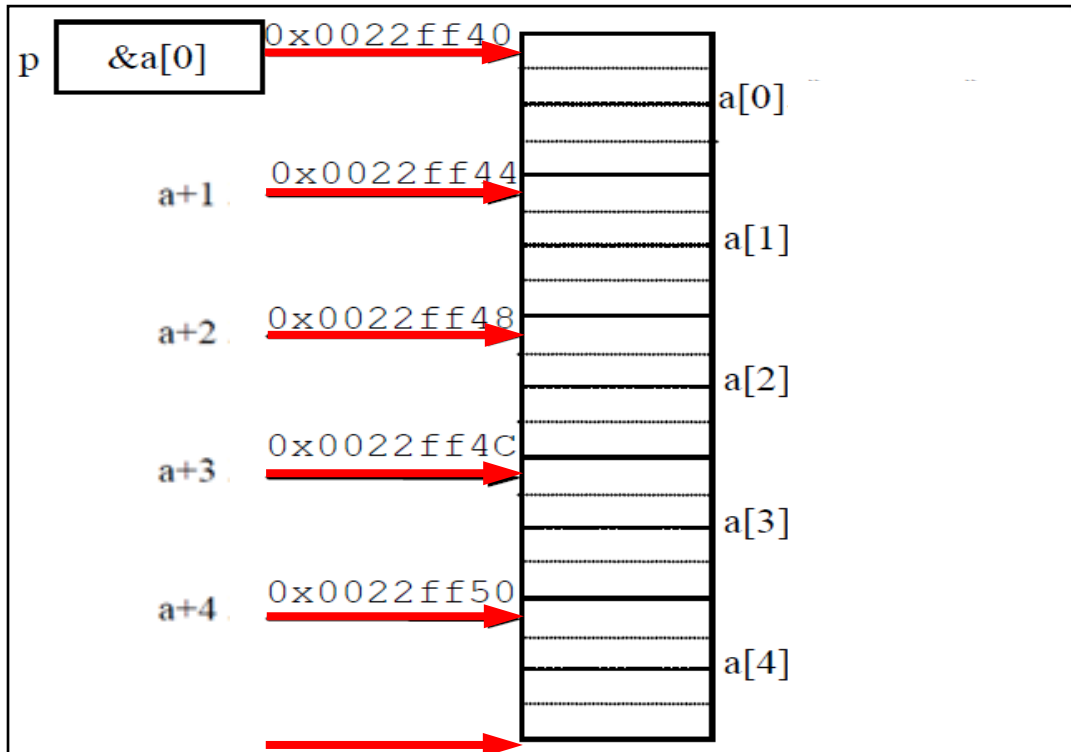
`a[i] ↔ *(a+i)`

`p[i] ↔ *(p+i)`

```
#include <stdio.h>
int main()
{
    int a[5], i, *p = NULL;
    p = a;
    for (i=0; i<5; i++)
    {
        scanf("%d", &p[i]);
    }
    p = a;
    for (i=0; i<5; i++)
    {
        printf("%4d", p[i]);
    }
    printf("\n");
    return 0;
}
```

一维数组元素的访问

```
int a[5];      int *p = a;
```

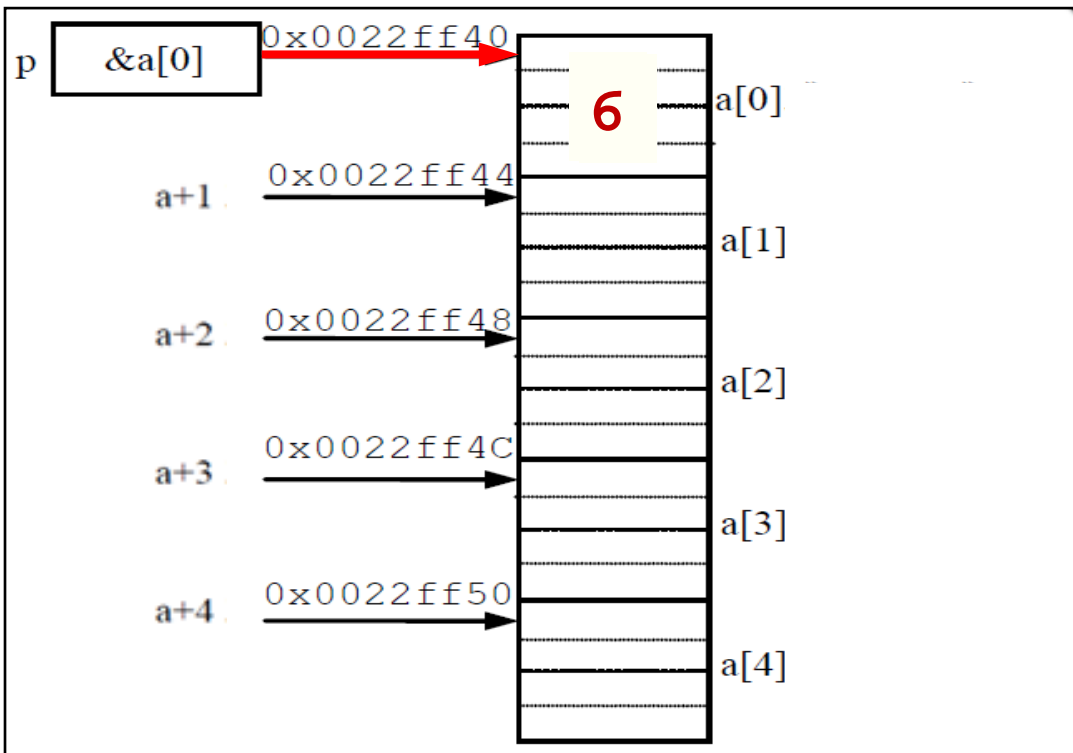


`p++`不是增加1字节，取决于`p`的基类型

```
#include <stdio.h>
int main()
{
    int a[5], *p = NULL;
    for (p=a; p<a+5; p++)
    {
        scanf("%d", p);
    }
    for (p=a; p<a+5; p++)
    {
        printf("%4d", *p);
    }
    printf("\n");
    return 0;
}
```

指向数组的指针

```
int a[5];      int *p = a;
```



p++不是增加1字节，取决于p的基类型

问题：*p++与(*p)++有何区别？

```
printf("%d\n", ++(*p));
```

6

```
printf("%d\n", (*p)++);
```

5

(*p)++ ，对*p加1，不改变p的指向

```
printf("%d\n", *p++);
```

```
printf("%d\n", *(p++));
```

```
printf("%d\n", *p);
```

5

```
p++;
```

当指针指向数组时，p++才有意义

指针和一维数组做函数参数

```
void InputArray(int a[], int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
}
```

被调函数的形参声明为数组类型，
用下标法访问数组元素

```
void OutputArray(int a[], int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        printf("%4d", a[i]);
    }
    printf("\n");
}
```


指针和一维数组做函数参数

```
void InputArray(int *p, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        scanf("%d", p++);
    }
}
```

被调函数的形参声明为**指针类型**,
用**指针算术运算**访问数组元素

问题：编写处理数组的循环时，
使用数组下标和指针算术运算，
哪种更好呢？

```
void OutputArray(int *p, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        printf("%4d", *p++);
    }
    printf("\n");
}
```

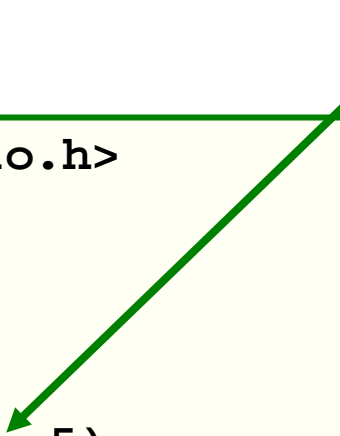
指针和一维数组做函数参数

```
#include <stdio.h>
int main()
{
    int a[5];
    InputArray(a, 5);
    OutputArray(a, 5);
    return 0;
}
```

在主函数中用数组名做函数实参

在主函数中这样做舍近求远
没必要

```
#include <stdio.h>
int main()
{
    int a[5];
    int *p = a;
    InputArray(p, 5);
    OutputArray(p, 5);
    return 0;
}
```



小结

- 指针与一维数组间的关系的关键
 - 牢记 $a[i] \leftrightarrow *(a+i)$
- 一维数组和指针做函数形参是等同的
- 数组和指针并非在所有情况下都是等同的
 - `sizeof(数组名)` 和 `sizeof(指针变量名)`，不可互换



