嵌入式C语言之-

指针与数组

讲师：叶大鹏

助力你成为优秀的电子工程师！

数组名称的用途

- 求数组占用的内存空间：

sizeof(数组名称)

此时，数组名称代表整个数组；

```
int32_t buffer[5] = {1, 2, 3, 4, 5};
int32_t size = sizeof(buffer);
printf("sizeof(buffer) = %d.\n", size);
```

```
sizeof(buffer) = 20.
```

数组名称的用途

- 数组名称除了可以代表整个数组以外sizeof(buffer)，还可以保存数组的首地址：

```
buffer == &buffer[0]
```

- 这个用途应用在通过指针来访问数组，以及向函数传递数组的场景，在后面的指针课程里，结合指针一起重点讲解：

```
int32_t  *p = buffer;
void  Sort(int32_t buffer[])
```

数组名称是一种特殊的指针变量

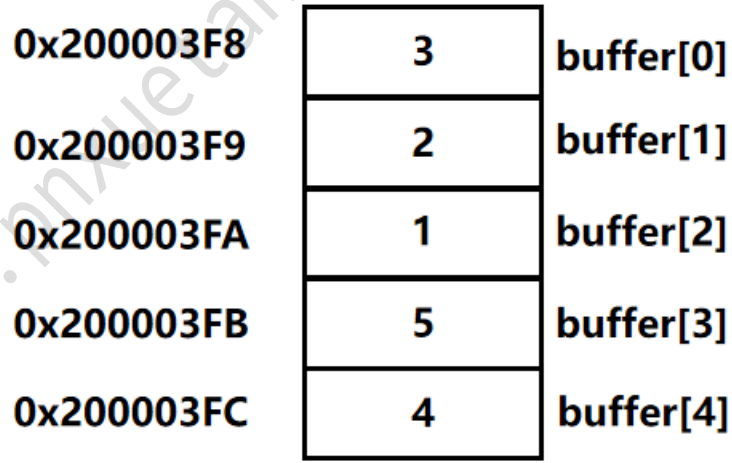
```
int8_t buffer[5] = {3, 2, 1, 5, 4};
printf("buffer = %p.\n", buffer);
for (uint8_t i = 0; i < 5; i++)
{
    printf("&buffer[%d] = %p.\n", i, &buffer[i]);
}
int8_t *ptr = buffer;
printf("ptr = %p.\n", ptr);
```

```
buffer = 200003f8.
&buffer[0] = 200003f8.
&buffer[1] = 200003f9.
&buffer[2] = 200003fa.
&buffer[3] = 200003fb.
&buffer[4] = 200003fc.
ptr = 200003f8.
```

- 数组名称buffer保存了数组的首地址，可以像指针变量一样直接读取地址值，不像结构体等其他类型变量，需要使用&；

深入思考使用下标访问数组元素

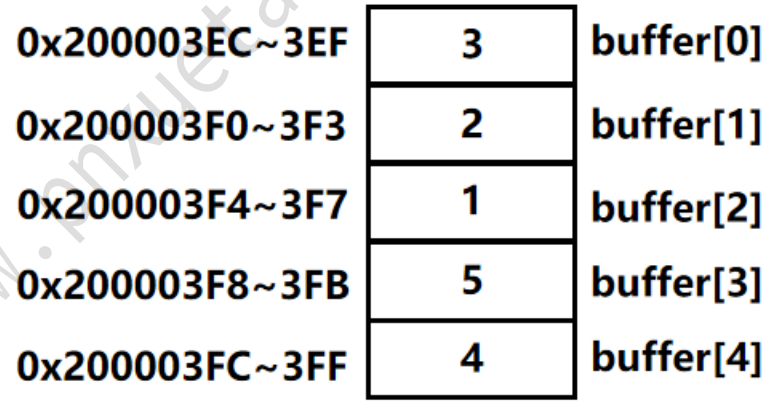
```
int8_t buffer[5] = {3, 2, 1, 5, 4};
for (uint8_t i = 0; i < 5; i++)
{
    printf("&buffer[%d] = %p.\n", i, &buffer[i]);
}
```



- 通过buffer[i]来访问第i个元素，可以这样理解，将它分解为2个动作：
  1. 地址跳转到：首地址 + i * 步长 的地址，步长概念和讲解指针运算时步长概念一致，数据类型占用空间大小；buffer[1]也就是跳转到3F8 + 1 * 1(uint8_t步长为1) = 3F9；所以buffer[i]可以理解为在buffer保存的首地址基础上跳转i个步长；
  2. 访问buffer[i]的内存空间，空间大小为数据类型大小，也即步长大小。

深入思考使用下标访问数组元素

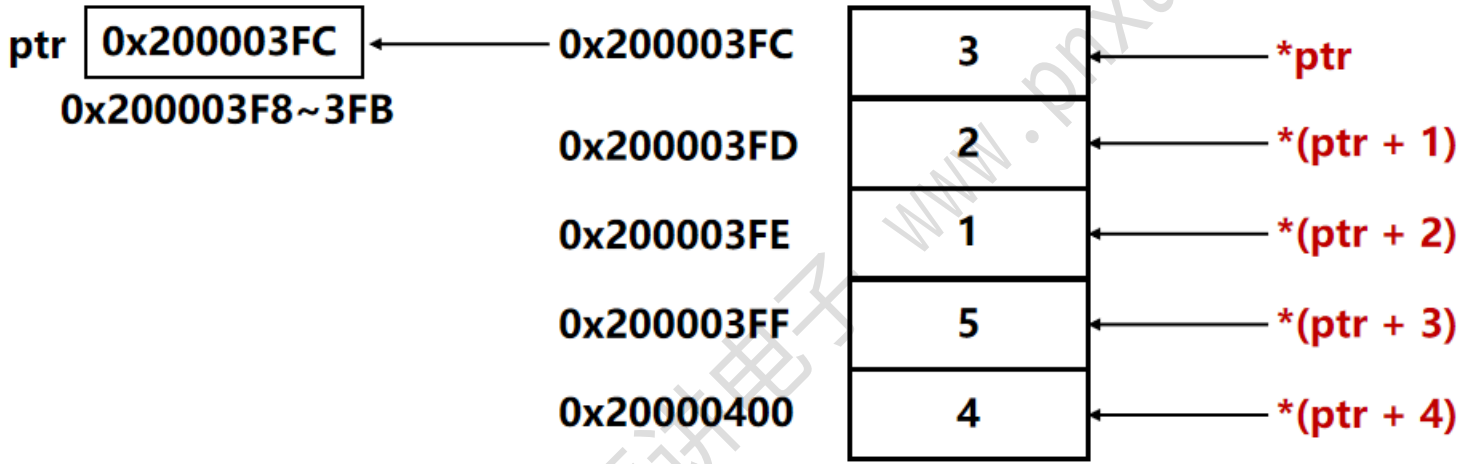
```
int32_t buffer[5] = {3, 2, 1, 5, 4};
for (uint8_t i = 0; i < 5; i++)
{
    printf("&buffer[%d] = %p.\n", i, &buffer[i]);
}
```



- 通过buffer[i]来访问第i个元素，可以这样理解，将它分解为2个动作：
  1. 地址跳转到：首地址 + i * 步长 的地址，步长概念和讲解指针运算时步长概念一致，数据类型占用空间大小；buffer[1]也就是跳转到3EC + 1 * 4(int32_t步长为4) = 3F0；所以buffer[i]可以理解为在buffer保存的基地址基础上跳转i个步长；
  2. 访问buffer[i]的内存空间，空间大小为数据类型大小，也即步长大小。

使用指针访问数组元素

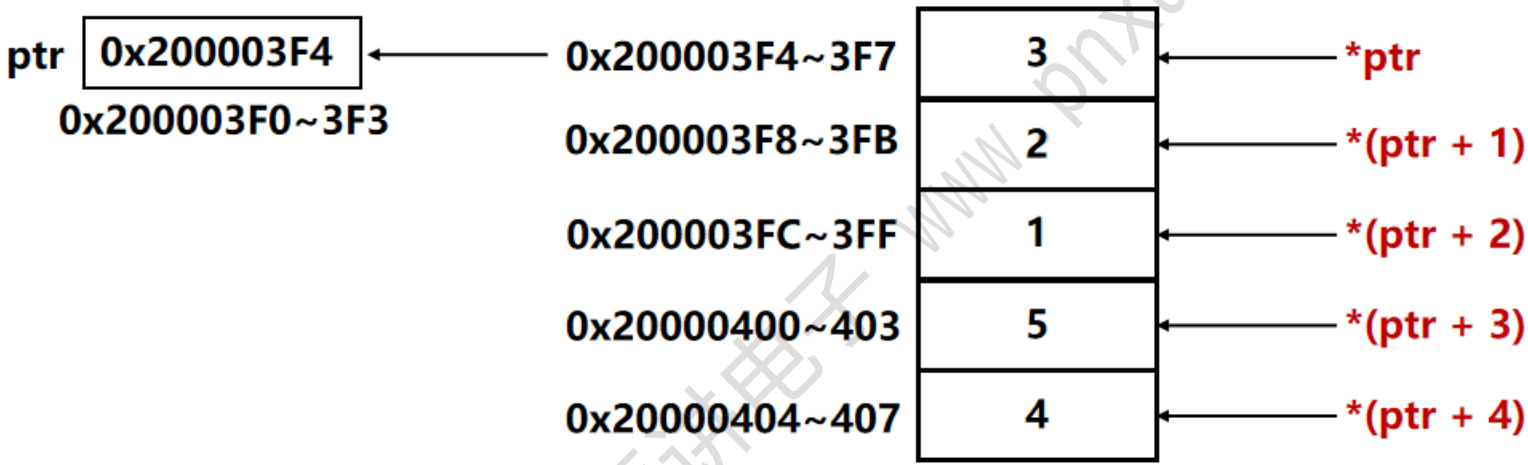
```
int8_t buffer[5] = {3, 2, 1, 5, 4};
int8_t *ptr = buffer;
```




```
for (uint8_t i = 0; i < 5; i++)
{
    printf("*(ptr + %d) = %d.\n", i, *(ptr + i));
}
```

使用指针访问数组元素

```
int32_t buffer[5] = {3, 2, 1, 5, 4};
int32_t *ptr = buffer;
```




```
for (uint8_t i = 0; i < 5; i++)
{
    printf("*(ptr + %d) = %d.\n", i, *(ptr + i));
}
```

数组名称可以作为指针使用

- 既然数组名称里保存的是数组的首地址，所以可以配合 * 来访问数组元素：

```
int32_t buffer[5] = {3, 2, 1, 5, 4};
int32_t *ptr = buffer;
```

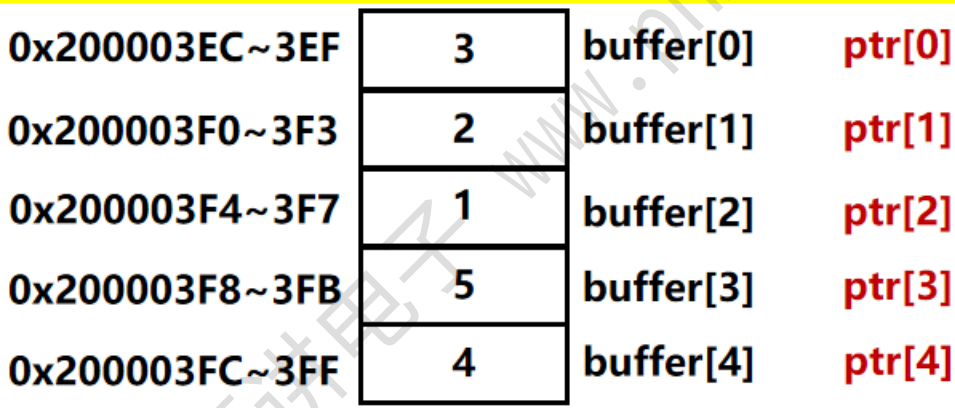
0x200003EC~3EF	3	buffer[0]	*buffer
0x200003F0~3F3	2	buffer[1]	*(buffer + 1)
0x200003F4~3F7	1	buffer[2]	*(buffer + 2)
0x200003F8~3FB	5	buffer[3]	*(buffer + 3)
0x200003FC~3FF	4	buffer[4]	*(buffer + 4)

```
for (uint8_t i = 0; i < 5; i++)
{
    printf("*(buffer + %d) = %d.\n", i, *(buffer + i));
}
```

指针可以作为数组使用

- 指针变量如果保存了数组的首地址，可以配合[i]方式访问数组元素：

```
int32_t buffer[5] = {3, 2, 1, 5, 4};
int32_t *ptr = buffer;
```




```
for (uint8_t i = 0; i < 5; i++)
{
    printf("ptr[%d] = %d.\n", i, ptr[i]);
}
```

数组名称和指针变量的区别

1. 数组名称第一种作用，可以代表整个数组：

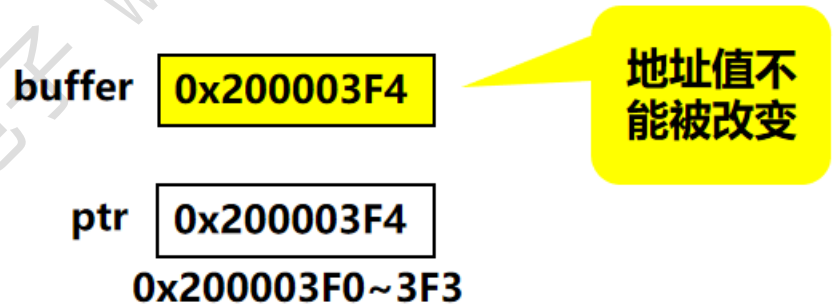
   sizeof(数组名称)

   此时，数组名称代表整个数组；

2. 数组名称的第二种作用，可以保存数组首地址，但是不需要像指针变量那样需要初始化赋值，而且保存的地址数值是不能被改变的：

```
int32_t buffer[5] = {3, 2, 1, 5, 4};

int32_t *ptr = buffer;
```



嵌入式C语言之-

数组作为函数参数

讲师：叶大鹏

数组作为函数参数

```
float CalRawAvg(float data[5], uint32_t len)
{
    float res = 0;
    for (uint32_t i = 0; i < len; i++)
    {
        res += data[i];
    }
    return res /= len;
}
int main(void)
{
    float raw[5] = {20.5, 21, 22, 23, 24};
    float cel = CalRawAvg(raw, 5);
    return 0;
}
```

- 可以定义为：

1. CalRawAvg(float data[5], uint32_t len)
2. CalRawAvg(float data[], uint32_t len)
3. CalRawAvg(float data[2], uint32_t len)

- 为什么float data[]和float data[2]这种方式也可以作为函数参数?

数组作为函数参数

```
float CalRawAvg(float data[], uint32_t len)
{
    float res = 0;
    for (uint32_t i = 0; i < len; i++)
    {
        res += data[i];
    }
    return res /= len;
}
int main(void)
{
    float raw[5] = {20.5, 21, 22, 23, 24};
    float cel = CalRawAvg(raw, 5);
    return 0;
}
```

- 实质上，数组作为参数，传递的并不是数组本身，而是数组的首地址，所以是地址传递，因为数组不能像结构体那样进行整体赋值；
- 不管是float data[]还是float data[2]，其实都等价于float *data，传参时发生的是：float *data = raw;
- 通常，定义函数时采用float *data或者float data[]；既然传递的只是地址，所以需要增加一个参数用来表示数组的元素个数。

THANK YOU!