嵌入式C语言之-

数组的定义和初始化

讲师：叶大鹏

助力你成为优秀的电子工程师！

数组的定义

- **数组可以理解为一种容器，用来放东西的东西，语法格式为：**

  数据类型　数组名称[元素的数量]

  int32_t buffer[10]; //buffer数组最多存放10个int32_t数据

  float rawData[5];

特点为：

- 所有元素具有相同的数据类型；
- 元素数量必须是整数；
- 一旦创建，不能改变元素的数量；
- 元素在内存中是连续依次排列的。

数组的初始化

- **数组的初始化有几种常见的形式：**

- **元素全部都初始化：**

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

- **元素全部未初始化：**

```
int32_t  buffer[5];
```

如果是全局数组，默认初始化为0，如果是局部数组，数值不确定；

- **元素部分初始化：**

```
int32_t  buffer[5] = {2, 1, 4};
int32_t  buffer[5] = {[0] = 2, [2] = 4};
```

- **定义时省略元素数量初始化：**

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

数组元素数量的个数是初始化数值的个数。

数组的使用

- **数组的使用方式为：**

  数组名称[下标]

- **下标的有效范围：**

  0 ~（元素数量 - 1）

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

数组的内存分布

- **数组的内存分布是连续递增的：**

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

遍历数组，通常使用for循环，让循环变量i从0到<数组的元素个数，这样循环语句内最大的i正好是最大的有效下标

0x200003F8	1
0x200003F9	2
0x200003FA	3
0x200003FB	4
0x200003FC	5

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

数组的内存分布

- **数组的内存分布是连续递增的：**

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

0x200003EC	1
0x200003F0	2
0x200003F4	3
0x200003F8	4
0x200003FC	5

```
&buffer[0] = 200003ec.
&buffer[1] = 200003f0.
&buffer[2] = 200003f4.
&buffer[3] = 200003f8.
&buffer[4] = 200003fc.
```

数组的内存分布

- **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.
```

数组的内存分布

- **求数组的元素个数：**

```
int32_t buffer[] = {1, 2, 3, 4, 5};
int32_t num= sizeof(buffer) / sizeof(buffer[0]);
printf("num = %d.\n", num);
```

```
num = 5.
```

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)，还可以代表数组的首地址：**

```
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[])
```

数组名称的用途

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

  buffer == &buffer[0]

- 这个用途可以应用在向函数传递数组的场景，类似于指针，所以在指针课程里结合指针一起讲解：

```
float CalRawAvg(float data[], uint32_t num)
{
	...
}

float GetCelTem(void)
{
	float raw[5];
	for (uint32_t i = 0; i < 5; i++)
	{
		raw[i] = GetRawData();
	}

	float cel = CalRawAvg(raw, 5);
	...
}
```

应用案例

- 给你一个数组 nums ， 数组中有 2n 个元素，按 [x1,x2,...,xn,y1,y2,...,yn] 的格式排列；请你将数组按 [x1,y1,x2,y2,...,xn,yn] 格式重新排列，返回重排后的数组。

- 示例 1：

输入：nums = [2,5,1,3,4,7], n = 3

输出：[2,3,5,4,1,7]

解释：由于 x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 ， 所以答案为 [2,3,5,4,1,7]

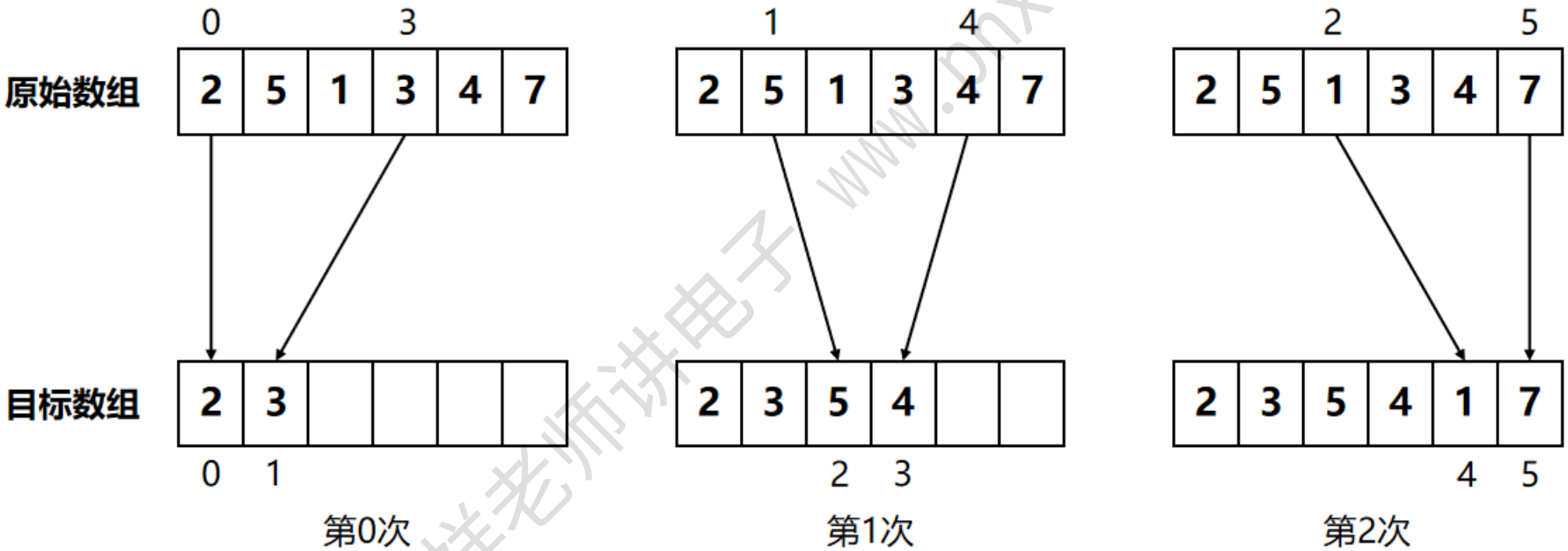
- 示例 2：

输入：nums = [1,2,3,4,4,3,2,1], n = 4

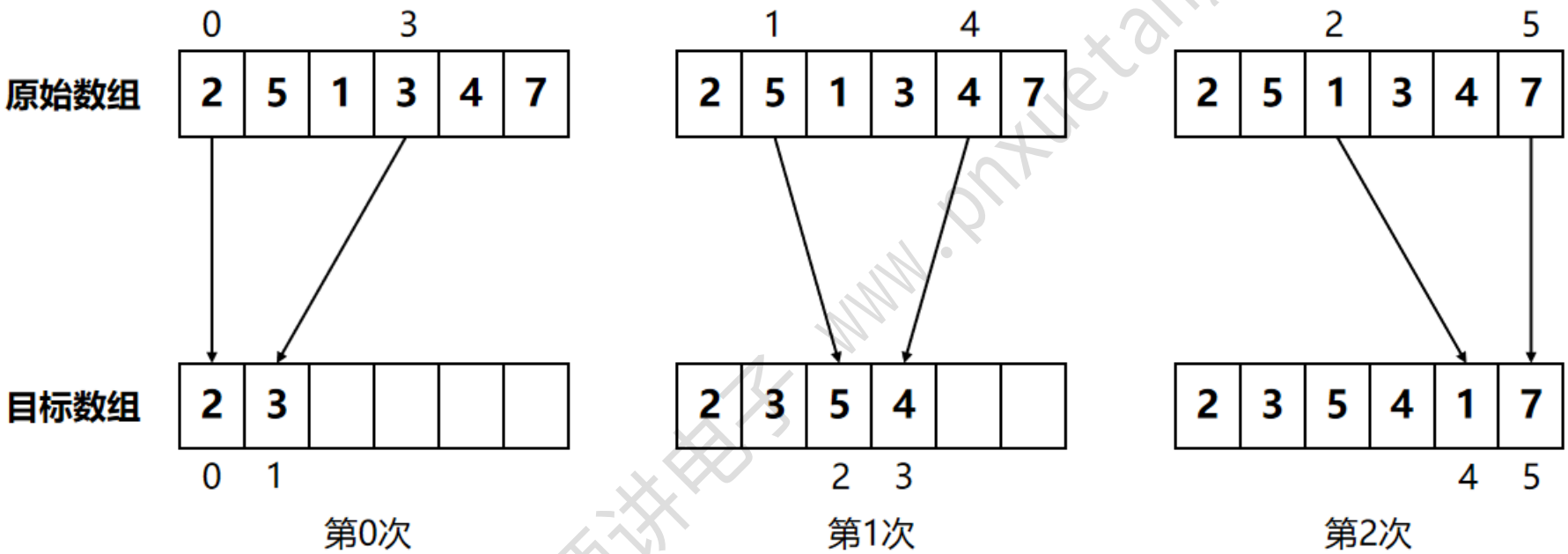
输出：[1,4,2,3,3,2,4,1]

应用案例

- 对于数组重排列这种题目，通常要使用循环来实现，解题思路实际上就是要找到循环变量i和原始数组的下标，以及目标数组下标的对应关系：



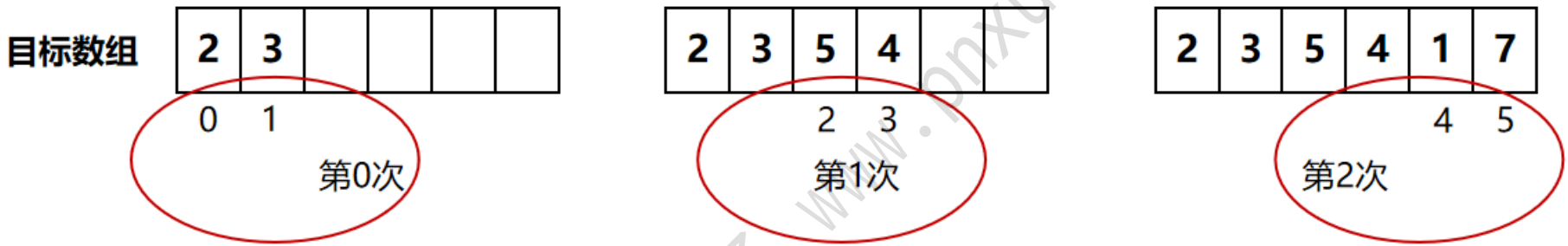
应用案例



i的初始值是0，边界是 <3(3对应的是需求里的n)

应用案例

- **求i和目标数组下标对应关系：**



每次访问的2个目标数组元素下标和i对应关系为：　（2 * i）　和　（2 * i + 1）

应用案例

- **求i和原始数组下标对应关系：**

每次访问的2个原始数组元素下标和i对应关系为：　（i）　和（i + 3(对应需求的n)）

应用案例

- **代码实现：**

```
for (uint32_t i = 0; i < n; i++)
{
        res[2 * i] = nums[i];
        res[2 * i + 1] = nums[i + n];
}
```

THANK YOU!