C语言内存管理和指针交流研讨

荣文戈

w.rong@buaa.edu.cn

北京航空航天大学计算机学院

2021-03-19

交流内容

- C语言内存基本概念回顾
- 内存管理和指针关系研讨
  - 基本类型内存表达
  - 一维/多维数组内存表达
  - 参数/变量/动态内存变化

思考

1、int a;
a的值和&a的值一样吗？

2、int a[10];
a的值和&a的值一样吗？

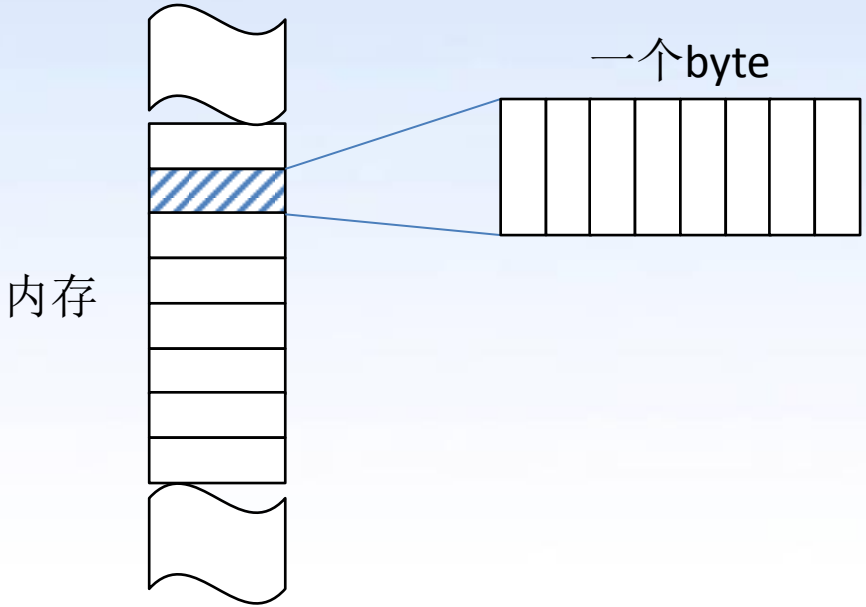
3、int a[10][20];
函数调用fun(a)
void fun(int** a)对吗？
形参为啥是int [][20]

内存基本概念简单回顾

bit（比特）
存储0/1

8个bit拼在一起，组成一个byte（字节）
存储00000000~11111111
也就是0~255

内存长什么样？



内存就是一大堆byte排列在一起组成的存储结构

内存有多少个这样的byte呢？

我们说8/16/32/64位机到底是什么意思？

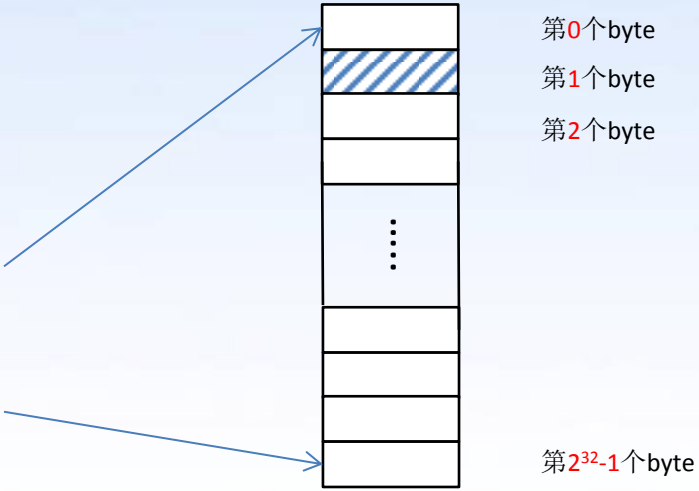
32位bit来表示byte的编号

00000000000000000000000000000000

……

11111111111111111111111111111111

32位



内存有多少个这样的byte呢？

我们说8/16/32/64位机到底是什么意思？

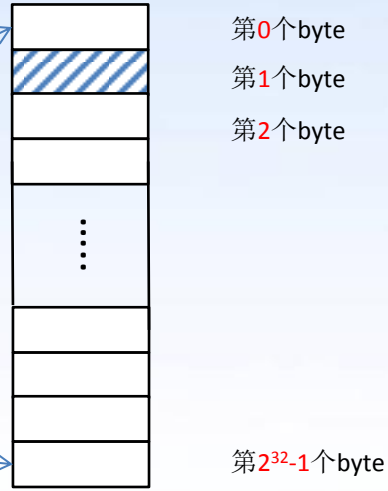
32位bit来表示byte的编号

用16进制来表示

0x00000000

……

0xFFFFFFFF



因此byte的个数一共可以有2^{32}个，$2^{32}=2^2*2^{10}*2^{10}*2^{10}$

4G内存，其中1K=1024，1M=1024K, 1G=1024M

基本概念：变量申明

在C语言中，申明一个变量具有两个基本要素：
1、变量类型
2、变量名称

变量类型约定了申请多大的内存
变量名称用于识别访问到这块内存

具体的申明方式要根据C语言的语法形式

变量类型分为1）非数组类型和2）数组类型

基本概念：非数组变量类型

int a;	变量类型为int, 变量名称为a
float b;	变量类型为float, 变量名称为b
double c;	变量类型为double, 变量名称为c

sizeof(int), sizeof(float), sizeof(double)的含义

基本概念：数组变量类型

int a[2];	变量类型为int[2], 变量名称为a
float b[2][3];	变量类型为float[2][3], 变量名称为b
double c[2][3][4];	变量类型为double[2][3][4], 变量名称为c

数组变量类型是特殊的一维构造类型，包含两要素：1、元素个数，2、元素类型是什么

int[2]包括2个元素，每个元素数据类型是int
float[2][3]包括2个元素，每个元素数据类型是float[3]
double[2][3][4]包括2个元素，每个元素数据类型是double[3][4]

sizeof(int[2]), sizeof(float[2][3]), sizeof(double[2][3][4])的含义

C语言数组变量类型都是一维的、一维的、一维的，重要的事情说三遍

基本概念：指针变量类型

任何一种变量类型，都有指向其的一个指针类型，反之亦然

int、float、double ⟺ int*、float*、double*

int[2]、float[2][3]、double[2][3][4] ⟺ int(*)[2]、float(*)[2][3]、double(*)[2][3][4]

指针变量类型本身也是一个非数组变量类型，也有其对应的指针类型

int*、float*、double* ⟺ int**、float**、double**

int(*)[2]、float(*)[2][3]、double(*)[2][3][4] ⟺ int(**)[2]、float(**)[2][3]、double(**)[2][3][4]

基本概念：指针变量申明

int *p vs. int* p

int* a	变量类型是int*，变量名是a
float* b	变量类型是float*，变量名是b
double* c	变量类型是double*，变量名是c
int (*d)[2]	变量类型是int(*)[2]，变量名是d
float (*e)[2][3]	变量类型是float(*)[2][3]，变量名是e
double (*f)[2][3][4]	变量类型是double(*)[2][3][4]，变量名是f

提问：1、sizeof(int*), sizeof(float*), sizeof(int(*)[2]) 大小都为 4，为什么？

2、int* g[2]是什么？

3、数组指针/指针数组的提法是不准确的，为什么？（数组2要素）

基本概念：数据类型总结

Variable Category	Variable Type	Variable Name	Variable Declaration	Element Type	Corresponding Pointer Type	Size
Non-Array Variables	int	a	int a;	N/A	int*	4
	char	b	char b;	N/A	char*	1
	float	c	float c;	N/A	float*	4
	int*	d	int* d;	N/A	int**	4
	char*	e	char* e;	N/A	char**	4
	float*	f	float* f;	N/A	float**	4
Array Variables	int[2]	g	int g[2];	int	int(*)[2]	8
	char[2][3]	h	char h[2][3];	char[3]	char(*)[2][3]	6
	float[2][3][4]	i	float i[2][3][4];	float[3][4]	float(*)[2][3][4]	96
	int*[2]	j	int* j[2];	int*	int*(*)[2]	8
	char*[2][3]	k	char* k[2][3];	char*[3]	char*(*)[2][3]	24
	float*[2][3][4]	l	float* l[2][3][4];	float*[3][4]	float*(*)[2][3][4]	96

简单示例：int a=10;

32位机上这样一个语句在内存中的操作：

1、申请4个byte（为什么是4？）

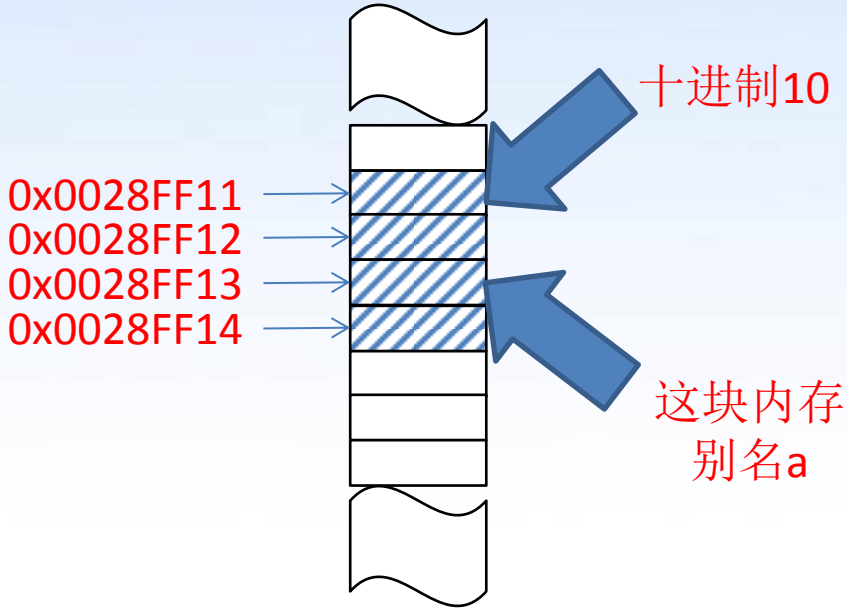
2、将10写入这4个byte

3、这4个byte组成的块的名字设为a

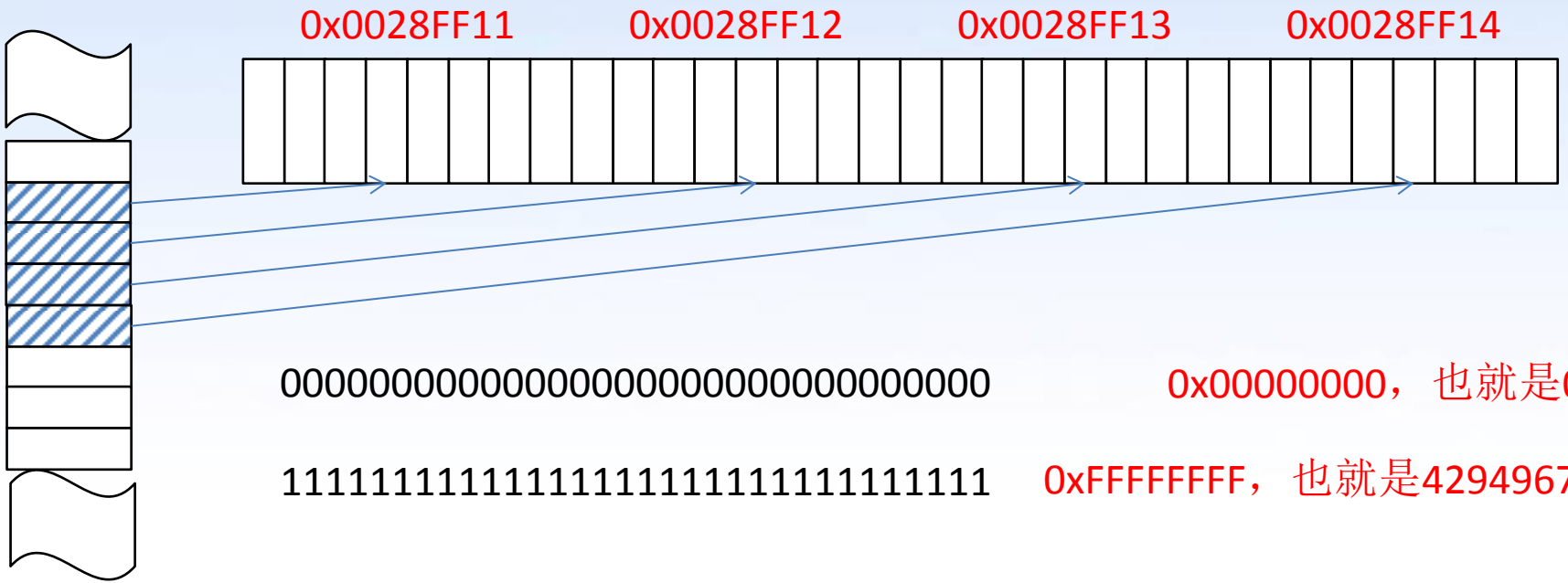
提问：这个十进制10是谁的视角？

这块内存的变量类型是什么？

这块内存的别名是什么？



int的取值范围



00000000000000000000000000000000　　0x00000000，也就是0

11111111111111111111111111111111　　0xFFFFFFFF，也就是4294967295

提问：等等，这是int还是unsigned int?

unsigned int和int的取值范围区别

unsigned int

11111111111111111111111111111111

4294967295

int

11111111111111111111111111111111

-2147483647

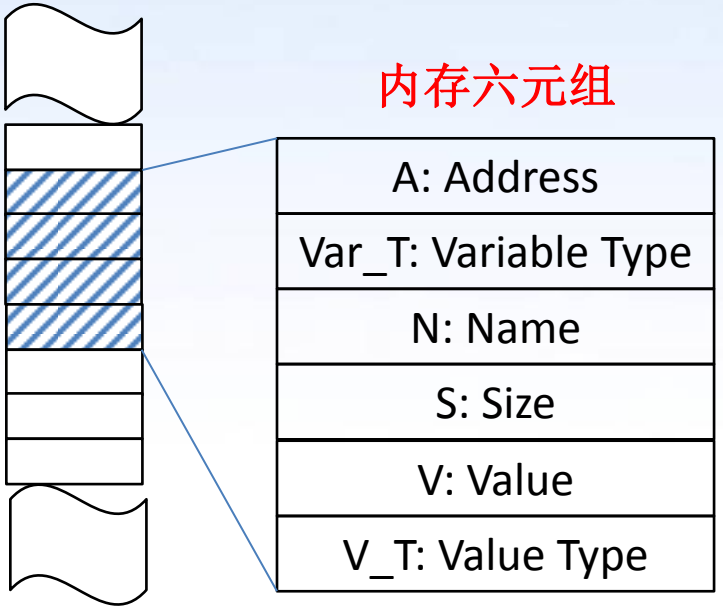
在内存中32个bit完全一样，但我们看到的值不一样，为什么？

内存六元组模型

M = {Address, Variable_Type, Name, Size, Value, Value_Type}

M：申明变量系统给分配的一段内存
Address：这一段内存第一个字节的编号
Variable_Type: 变量类型
Name：变量名称
Size：内存大小（字节数量）
Value：这段内存的表示值
Value_Type：表示值的类型

提问1：这六个属性值都必须有吗？
提问2：这六个属性值能修改吗？



内存六元组取值规则

M = {Address, Variable_Type, Name, Size, Value, Value_Type}

1、Address由系统分配，一旦确定无法修改

2、Variable_Type和Name是变量申明对应的变量类型和变量名

3、Size是这块内存的大小（字节数）

4、Value和Value_Type的取值遵循以下原则

1）Variable_Type是非数组类型

Value_Type=Variable_Type, Value是通过Value_Type去观察这段内存获得的值

2）Variable_Type是数组类型

Value_Type是指向数组里元素变量类型的指针类型，Value是数组第一个元素所处内存的第一个字节编号（等于Address)

示例

基本数据类型：int, float, double, char

指针类型：int*

一维数组：int[2], char[8]

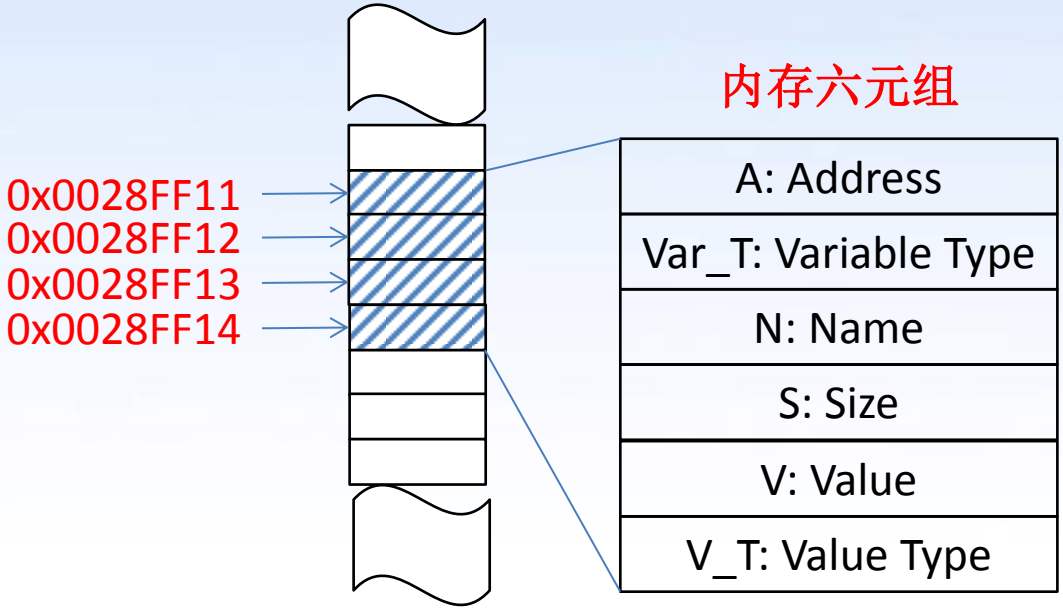
二维数组：int[2][3]

结构体/联合体：struct m_struct/union m_union

动态内存：mallco/calloc

```c
// primitive type
int a = 10;
float b = 1.0;
double c = 2.0;
char d = 'a';
// pointer
int *p;
// 1-D array
int e[2];
char f[8];
// 2-D array
int g[2][3];
// Dynamic Memory Allocation
malloc(16);
// Struct and Union
struct m_struct {
    int a;
    float b;
}h;
union m_union {
    int a;
    float b;
}i;
```

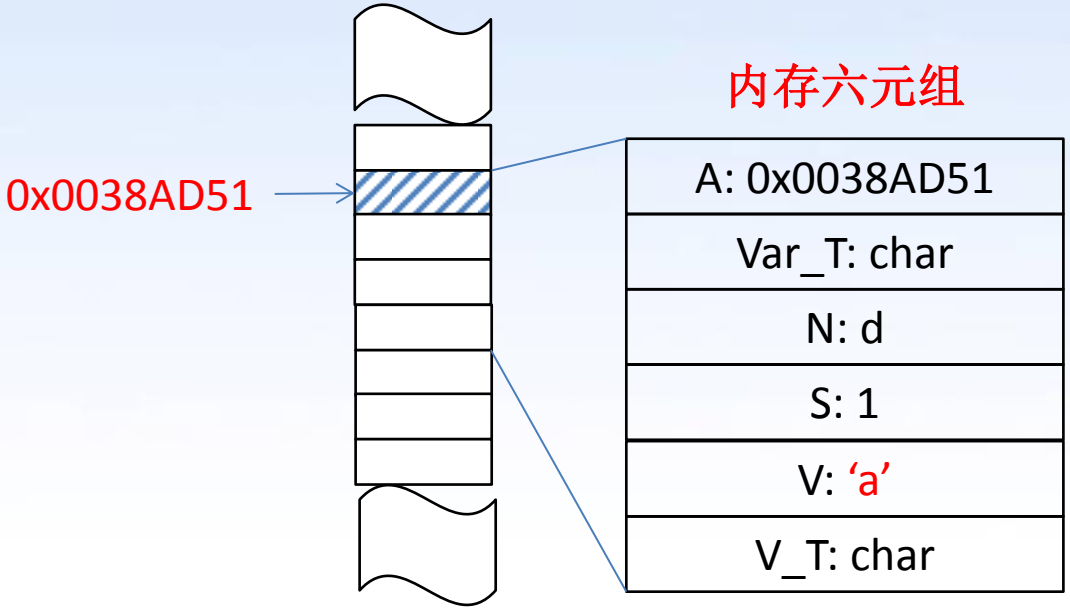
再观察int a=10



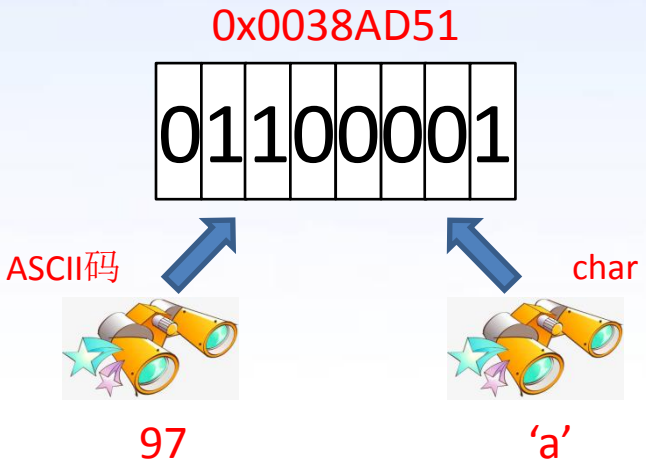
Address: 0x0028FF11
Variable Type: int
Name: a
Size: 4
Value: 10
Value Type: int

Variable_Type = Value_Type

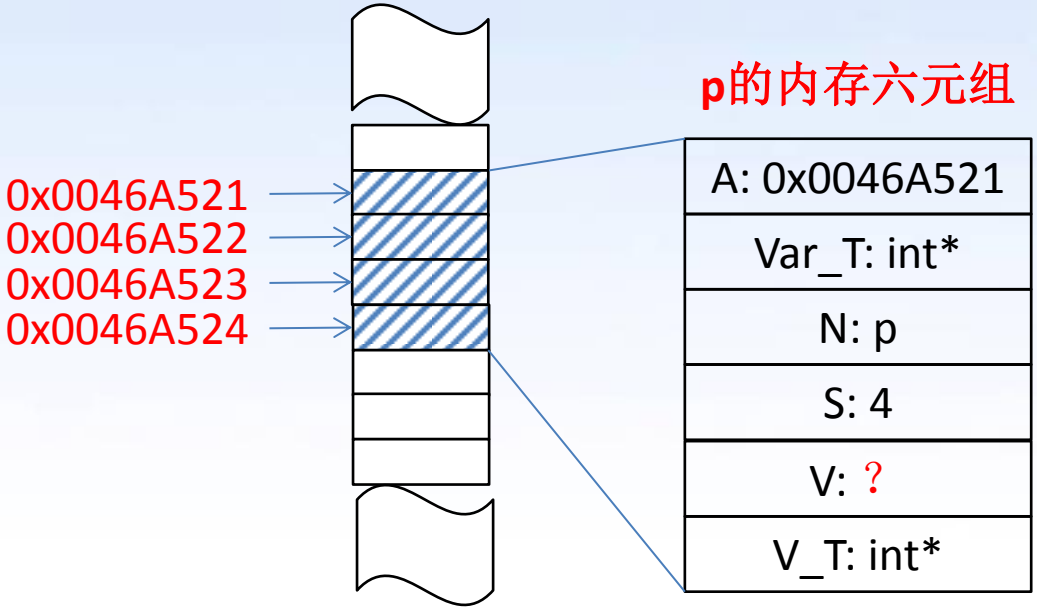
观察char d='a'



Value = 'a'，到底怎么存的？



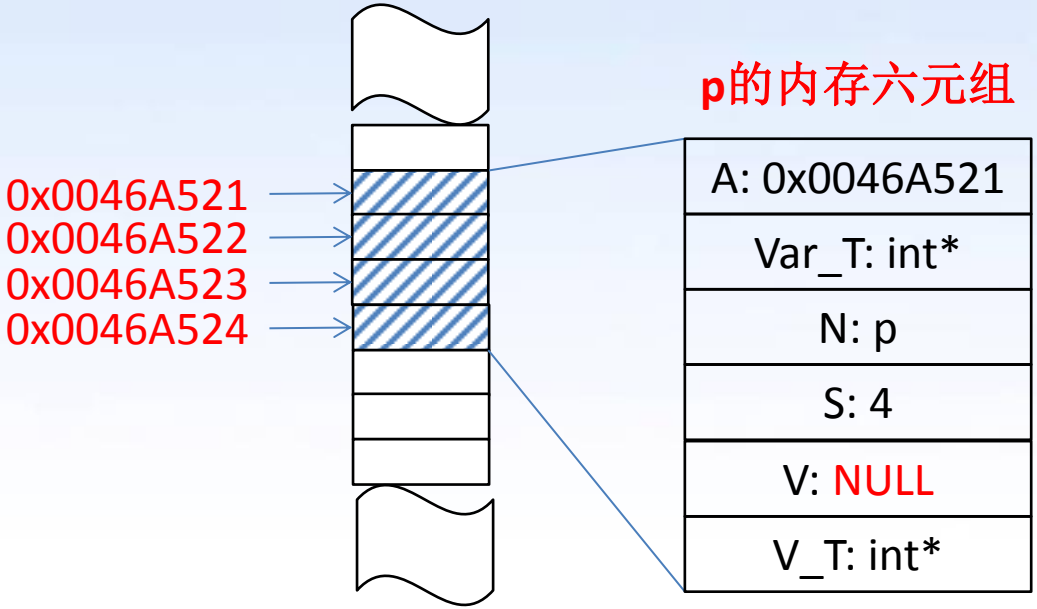
观察：int* p;



Address: 0x0046A521
Size: 4
Name: p
Content: int*
Value: ?
Type: int*

思考1：为什么size是4
思考2：这个时候value有值吗？

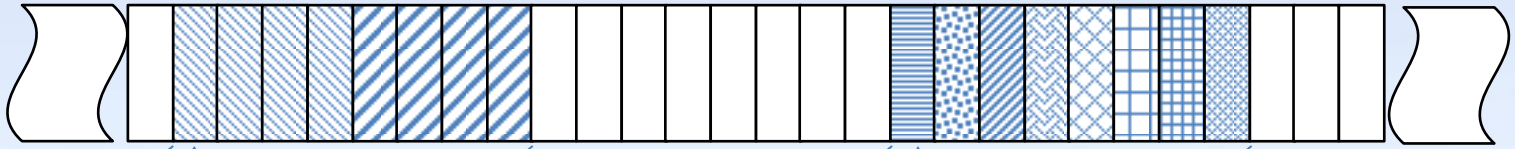
改进一下：int* p=NULL；



Address: 0x0046A521
Size: 4
Name: p
Content: int*
Value: NULL
Type: int*

思考1：初始化为NULL的好处
思考2：int*的取值范围是unsigned int还是int

观察：int e[2]和char f[8]



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

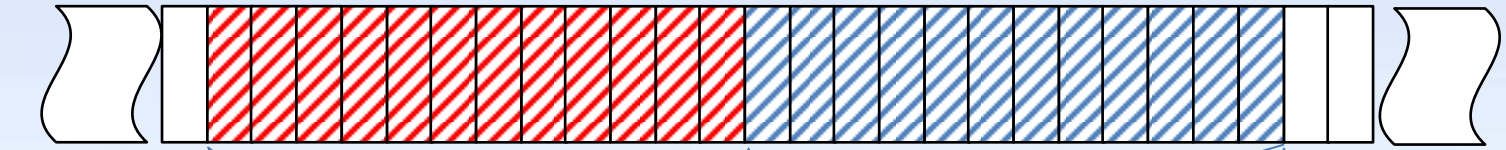
0x0076AB21

A: 0x0076AB21
Var_T: char[8]
N: f
S: 8
V: 0x0076AB21
V_T: char*

思考1：为啥大小都是8

对int e[2]变量来说
思考2：int[2]谁的视角
思考3：int*谁的视角
思考4：为啥V的值和A的值一样?

观察：int g[2][3]

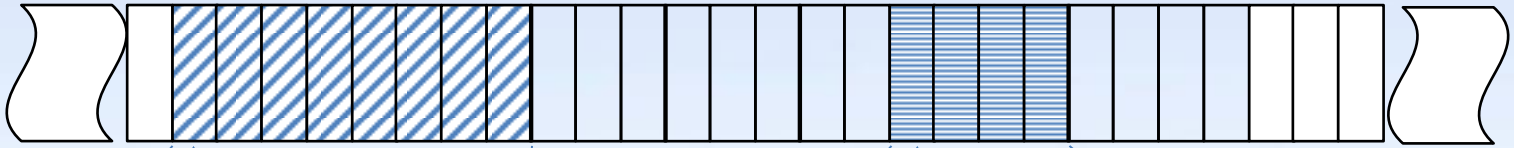


0x0098B11D int[3] 0x0098B129 int[3] 0x0098B135

A: 0x0098B11D
Var_T: int[2][3]
N: f
S: 24
V: 0x0098B11D
V_T: int(*)[3]

思考1：为啥大小是24
思考2：int[2][3]谁的视角
思考3：int(*)[3]谁的视角
思考4：为啥V的值和A的值一样?

观察：结构体变量h和联合体变量i



0x0085D611

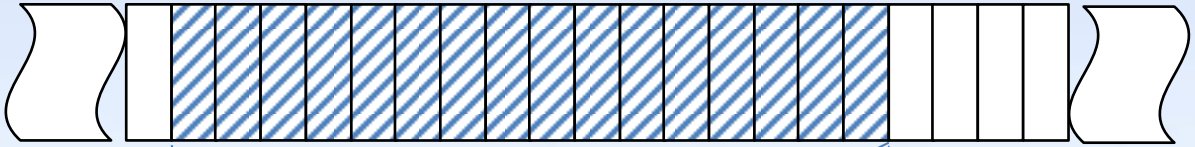
A: 0x0085D611
Var_T: struct m_struct
N: h
S: 8
V: Invisible
V_T: struct m_struct

0x0085D621

A: 0x0085D621
Var_T: union m_union
N: i
S: 4
V: Invisible
V_T: union m_union

思考1：结构体和联合体Value类型有Value吗？如果有到底是啥？

观察：malloc(16)



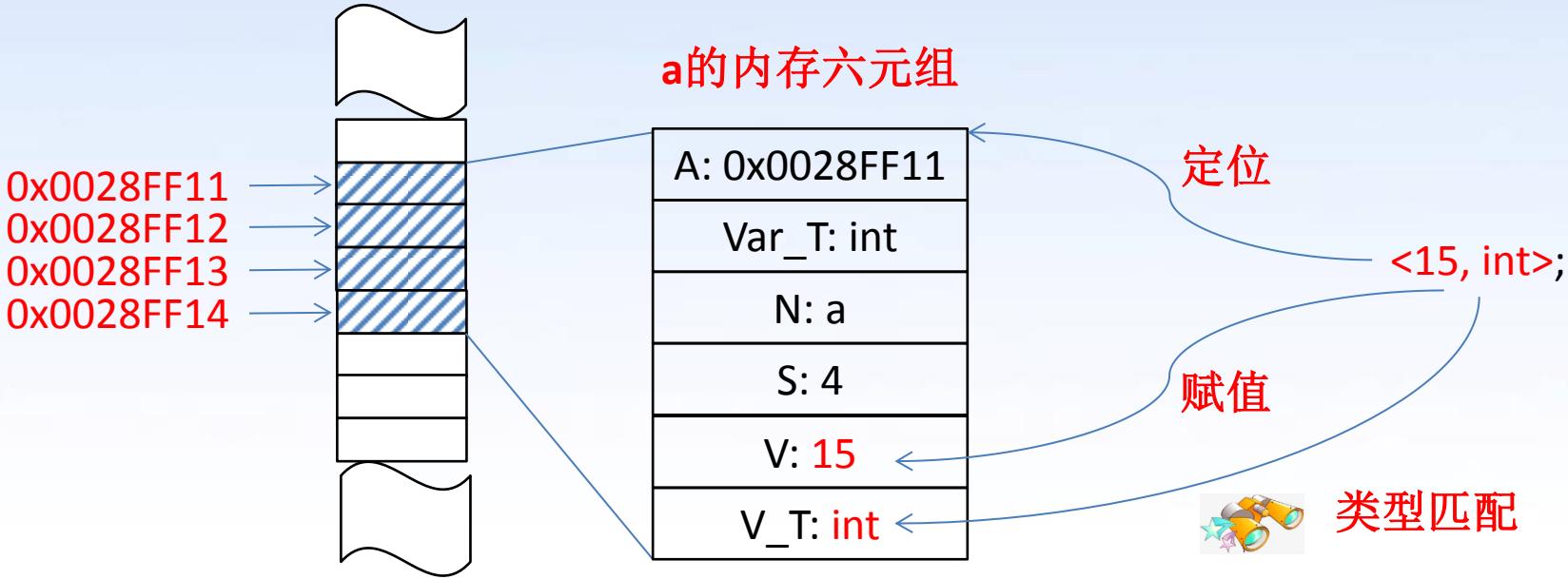
0x00351728　　0x00351738

A: 0x00351728
Var_T: N/A
N: N/A
S: 16
V: N/A
V_T: N/A

思考1：之前变量靠变量名识别内存，那么这块内存如何定位？

思考2：为什么Address和Size不可或缺？

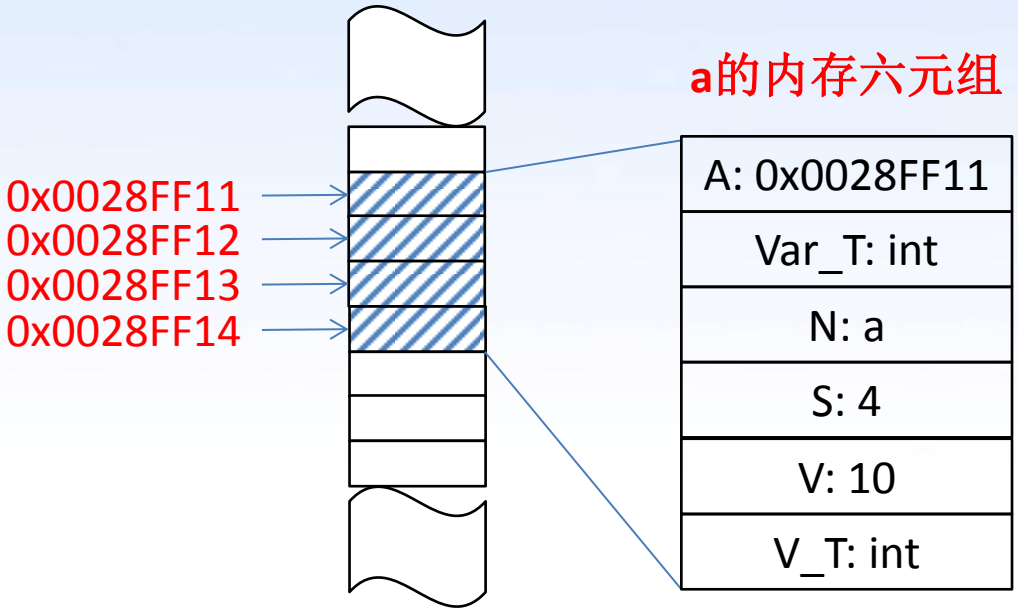
对内存赋值：int a；a=15发生了什么？



变量名被赋值，规则如下：按Value_Type类型把值放入Value中

对内存的取值操作有哪些？

int a=10;



a的内存六元组

A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int

观察下面三个语句

```
int* p=&a;
size_t c =sizeof(a);
int b = a;
```

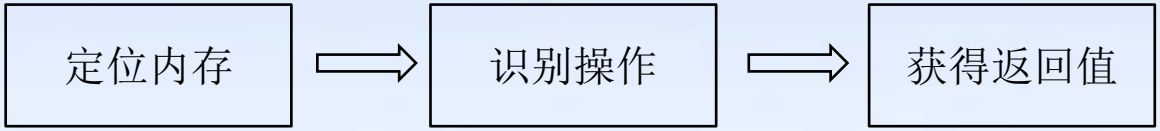
通过变量名定位a所在内存，三个操作按优先级为：

1、&a

2、sizeof(a)

3、a

内存取值相关的三种操作

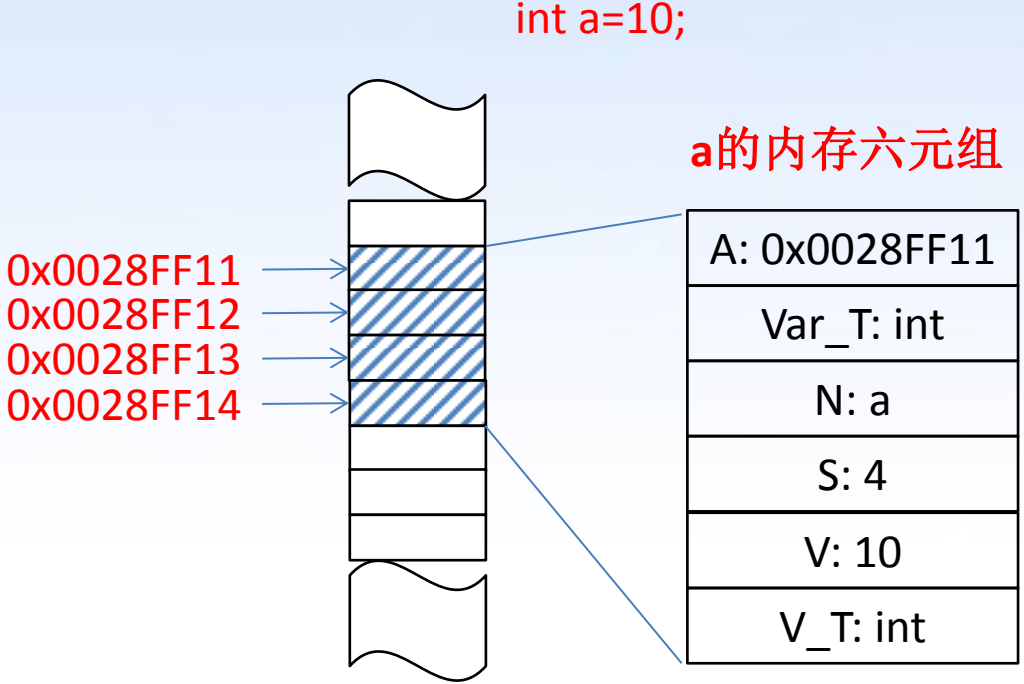


内存定位的方法：
1、变量名定位
2、指针间接定位

内存相关有三种操作类型，每一种都获得一个返回值：
1、获得内存的首地址，&
2、获得内存的大小，sizeof
3、获得内存的表示值，除1，2两种操作符以外

返回值两元素
1、返回值的类型
2、返回值的值

观察：int a=10；&a返回什么？



对于&a：

1、识别a所在内存

2、发现前面有&

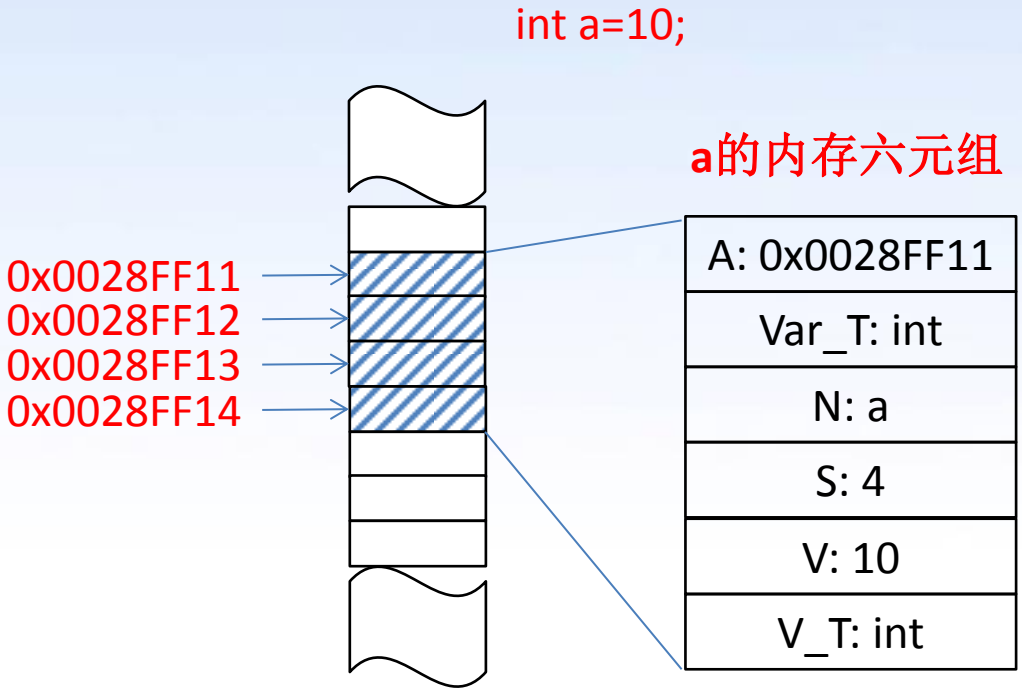
3、返回值规则如下：

返回值：

<Address, Pointer to Variable Type>

&a：<0x0028FF11, int*>

观察：int a=10; sizeof(a)返回什么？



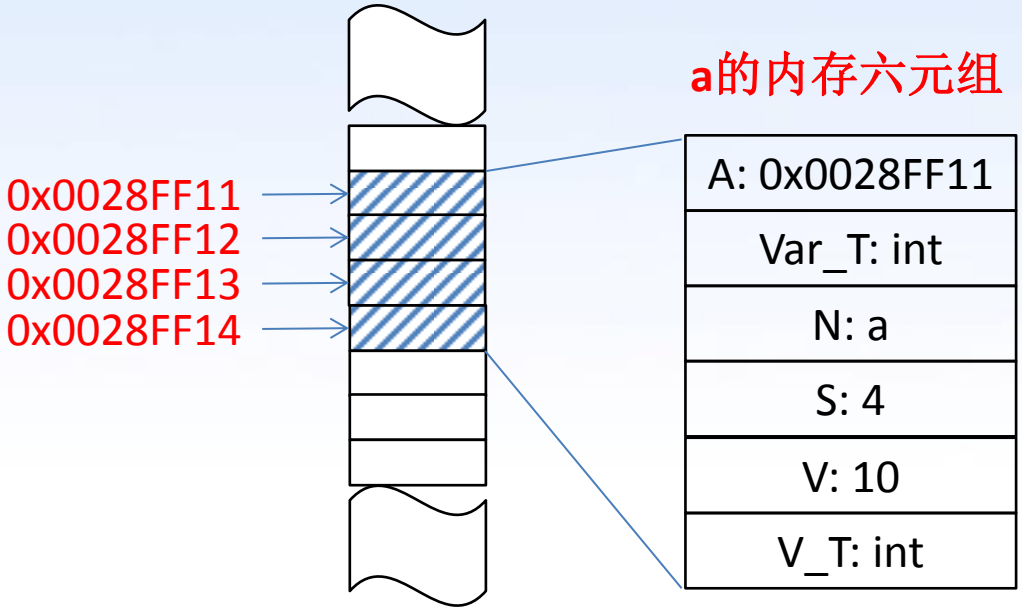
对于sizeof(a)：
1、识别a所在内存
2、发现前面没有&
3、发现和sizeof结合一起
3、返回值规则如下：

返回值：<Size, size_t>
size_t和unsigned int什么关系？

sizeof(a)：<4, size_t>

观察：int a=10；a返回什么？

int a=10;



对于a：

1、识别a所在内存

2、发现前面没有&

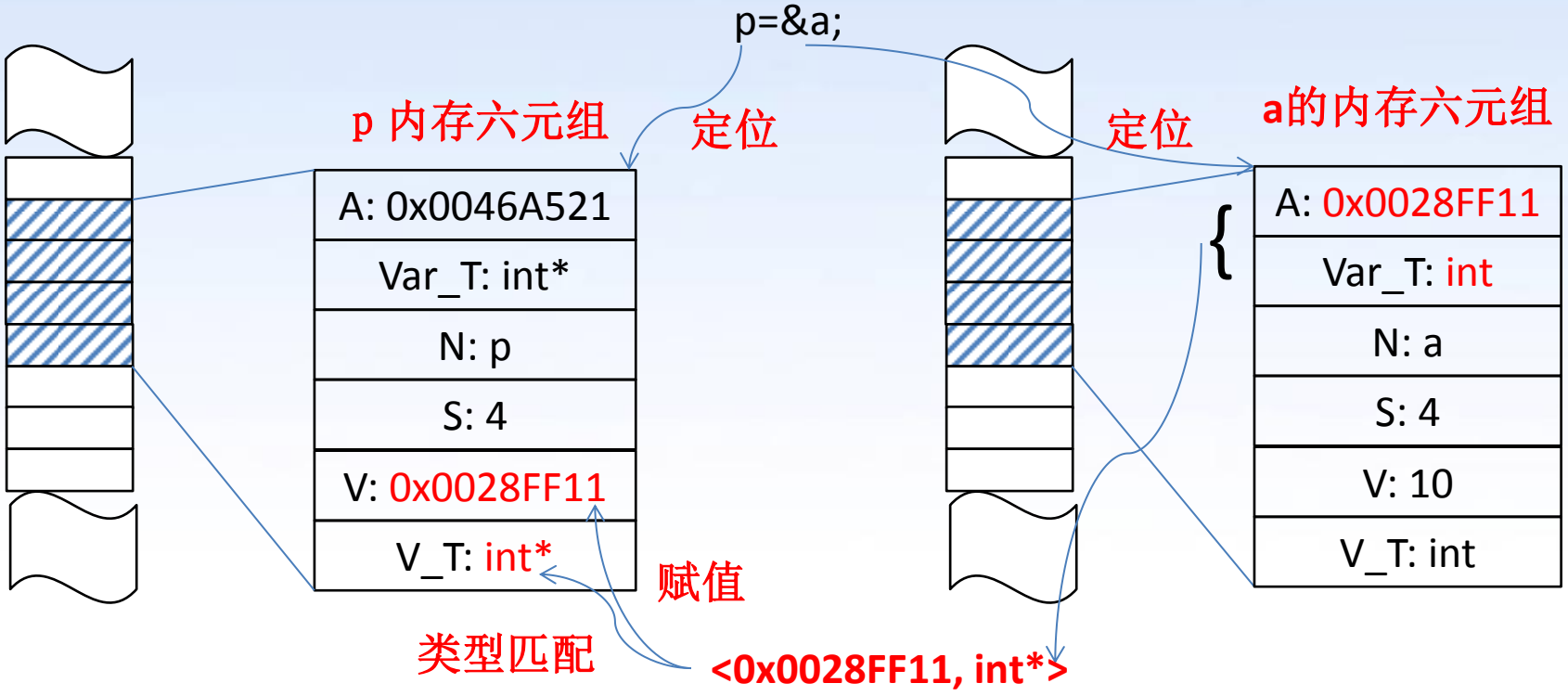
3、发现没有和sizeof结合一起

3、返回值规则如下：

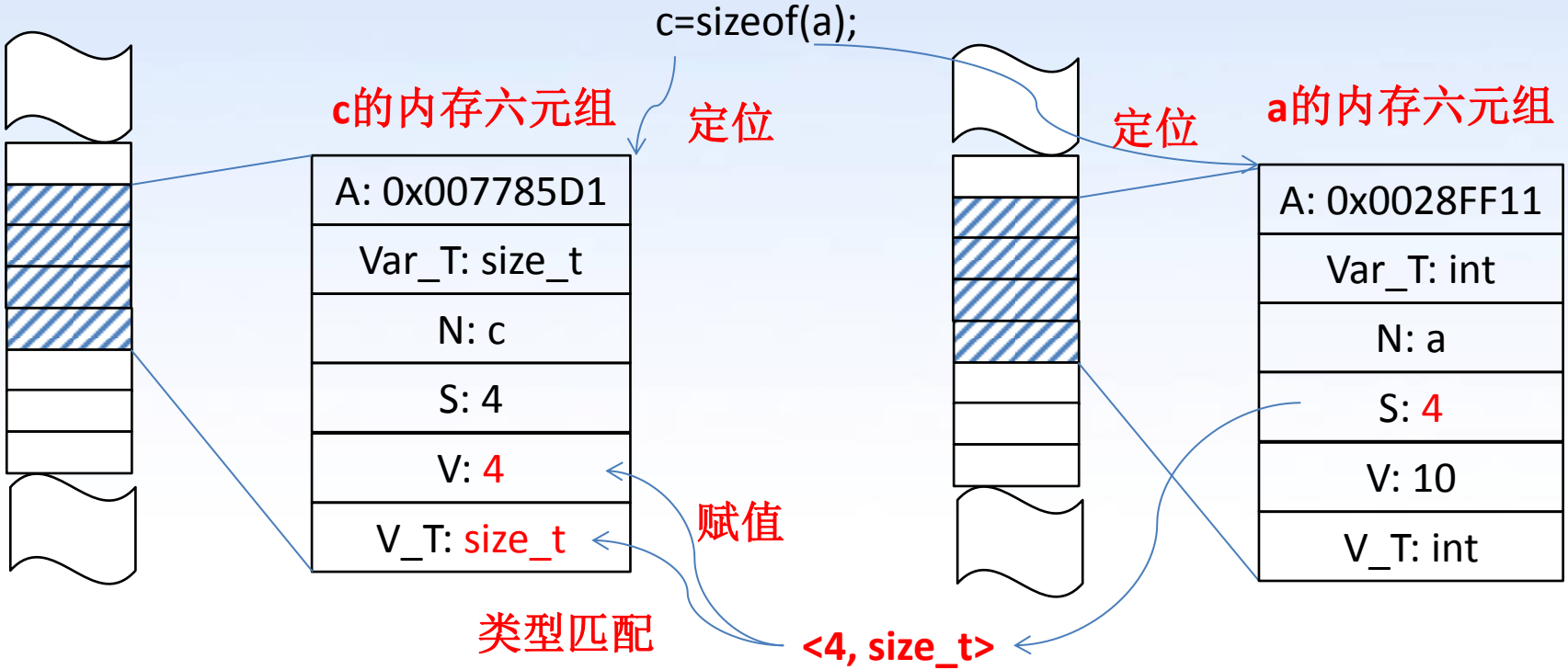
返回值：<Value, Value_Type>

a: <10, int>

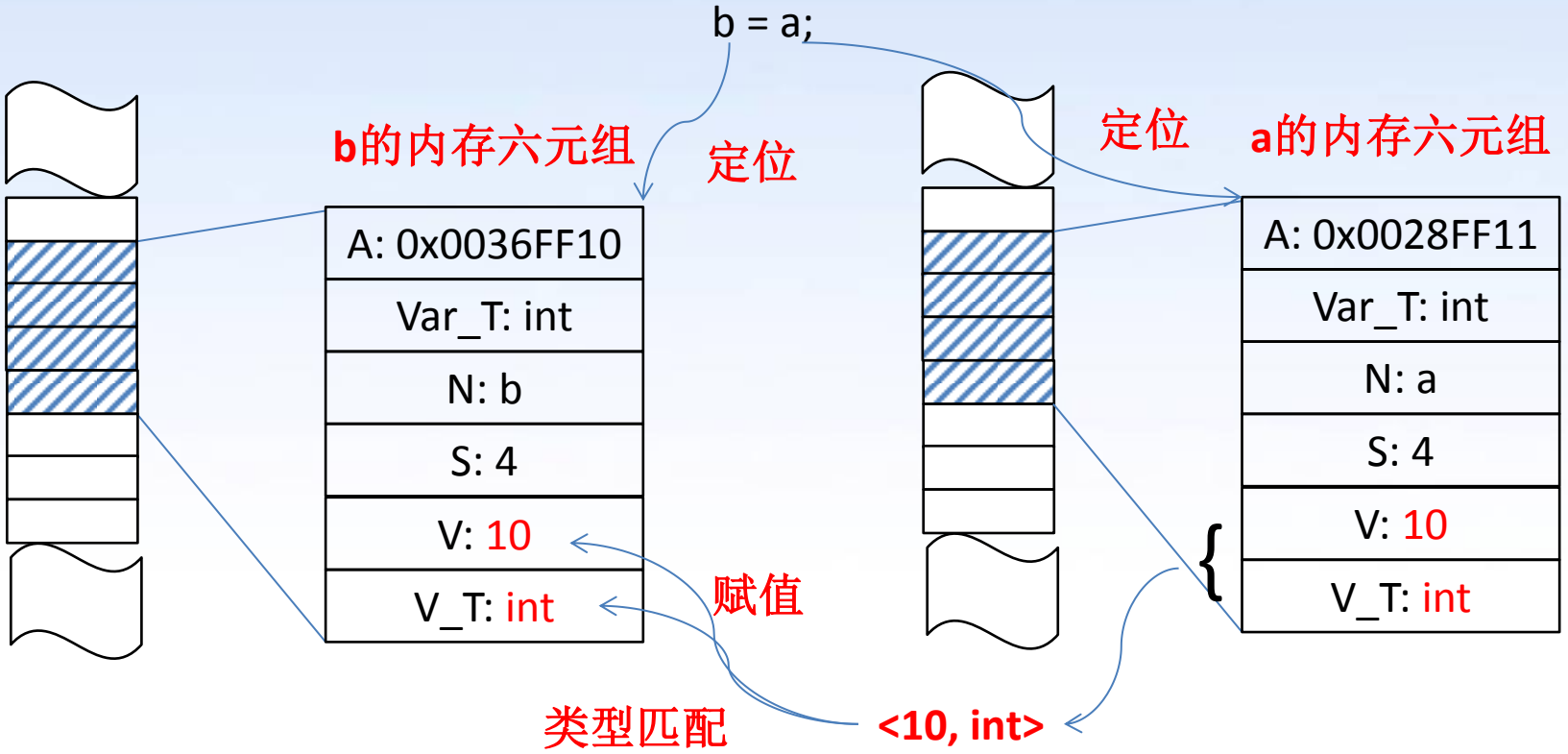
观察：int* p=&a发生了什么？



size_t c=sizeof(a)发生了什么？



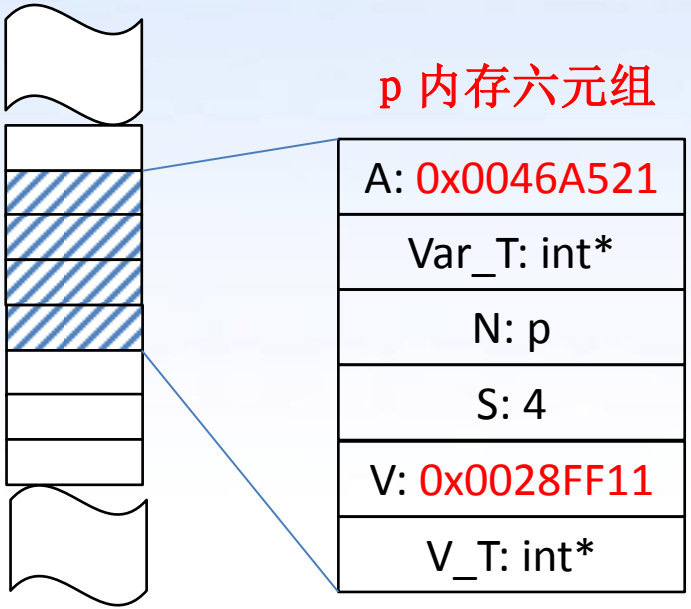
int b=a时发生了什么？



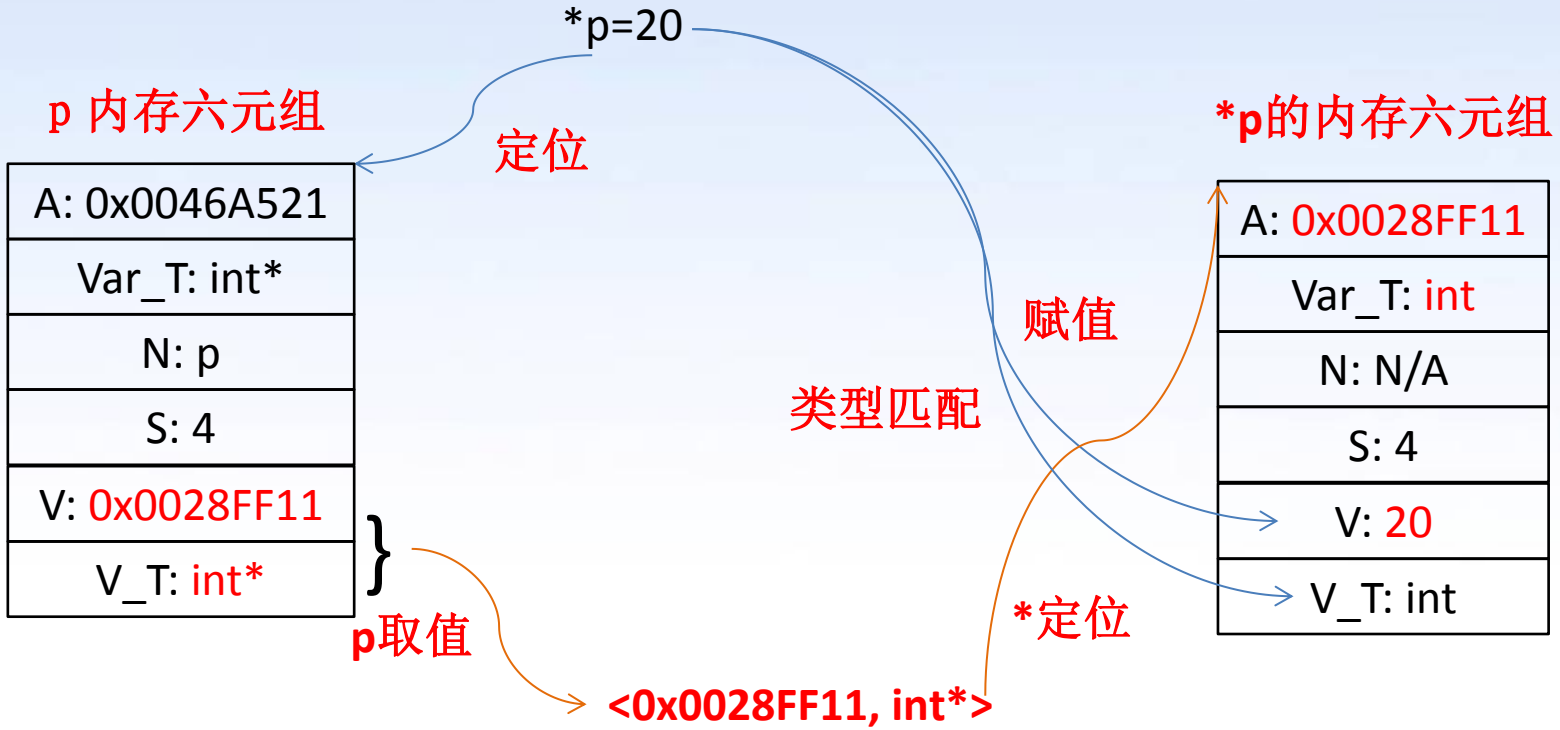
指针类型的间接内存定位：*p

之前我们定了一个int a=10;
刚才我们定义了一个int* p=&a;
利用* + <地址，指针类型>间接定位内存

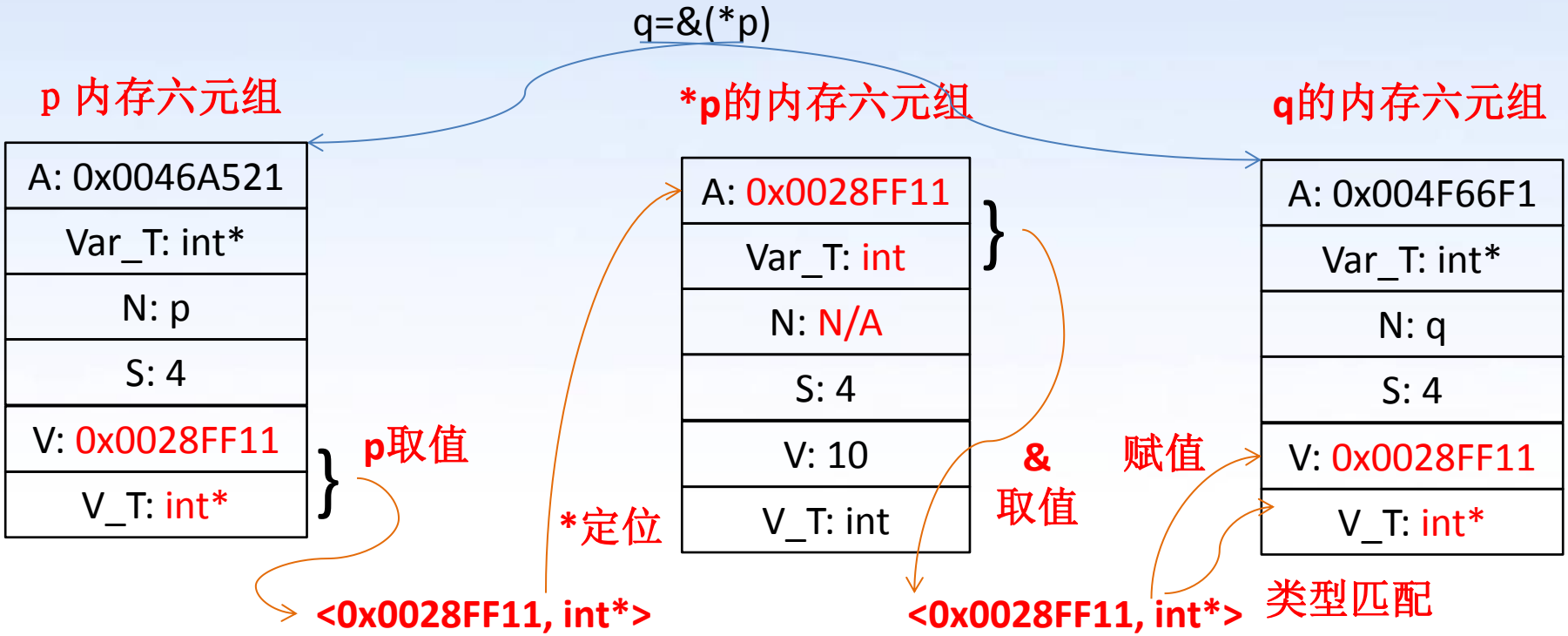
*p=20的时候发生了什么？
1、定位p的内存
2、获得p的表示值<Value, Value_Type>
3、找到内存中Value对应的字节作为起点
4、根据Value_Type还原那块内存六元组，其中那块内存的Variable_Type是Value_Type指针对应的指向的变量类型
4、将20赋值给那块内存



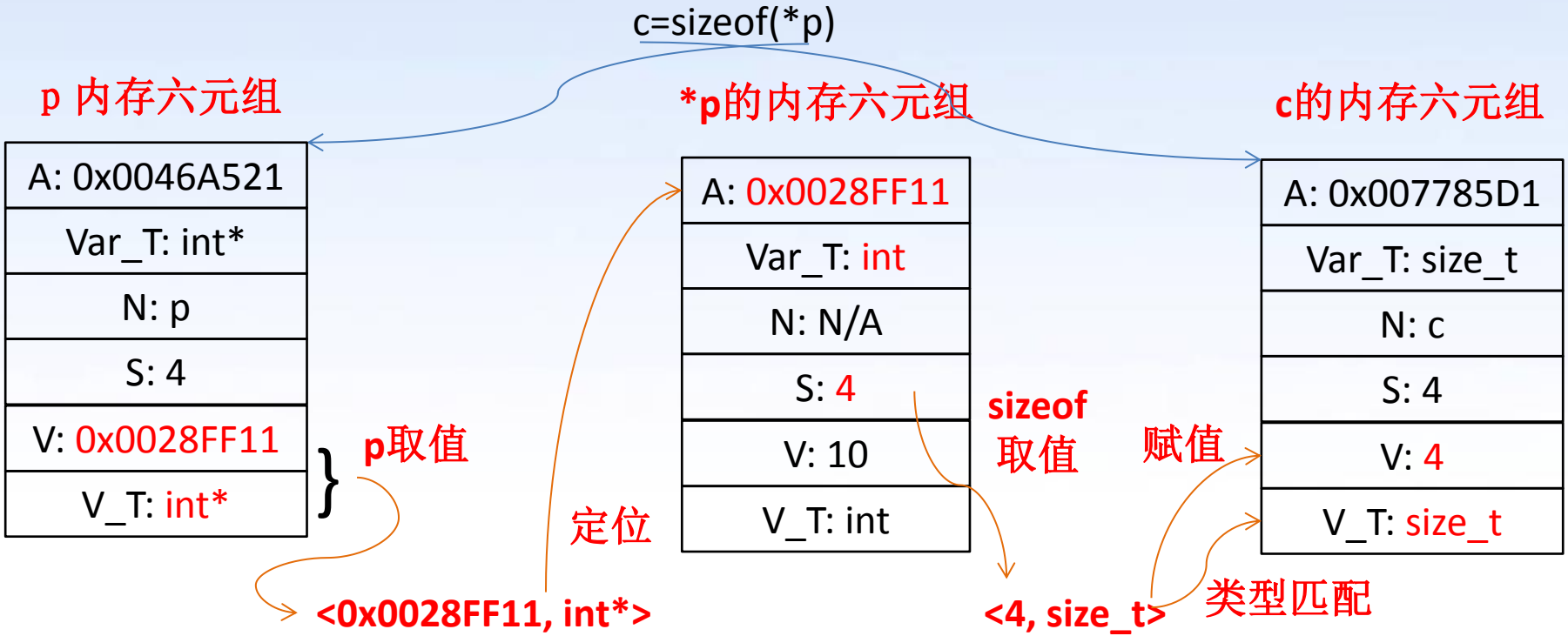
*p=20发生了什么？


*p=20
p 内存六元组
定位
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
p取值
<0x0028FF11, int*>
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 20
V_T: int
赋值
类型匹配
*定位

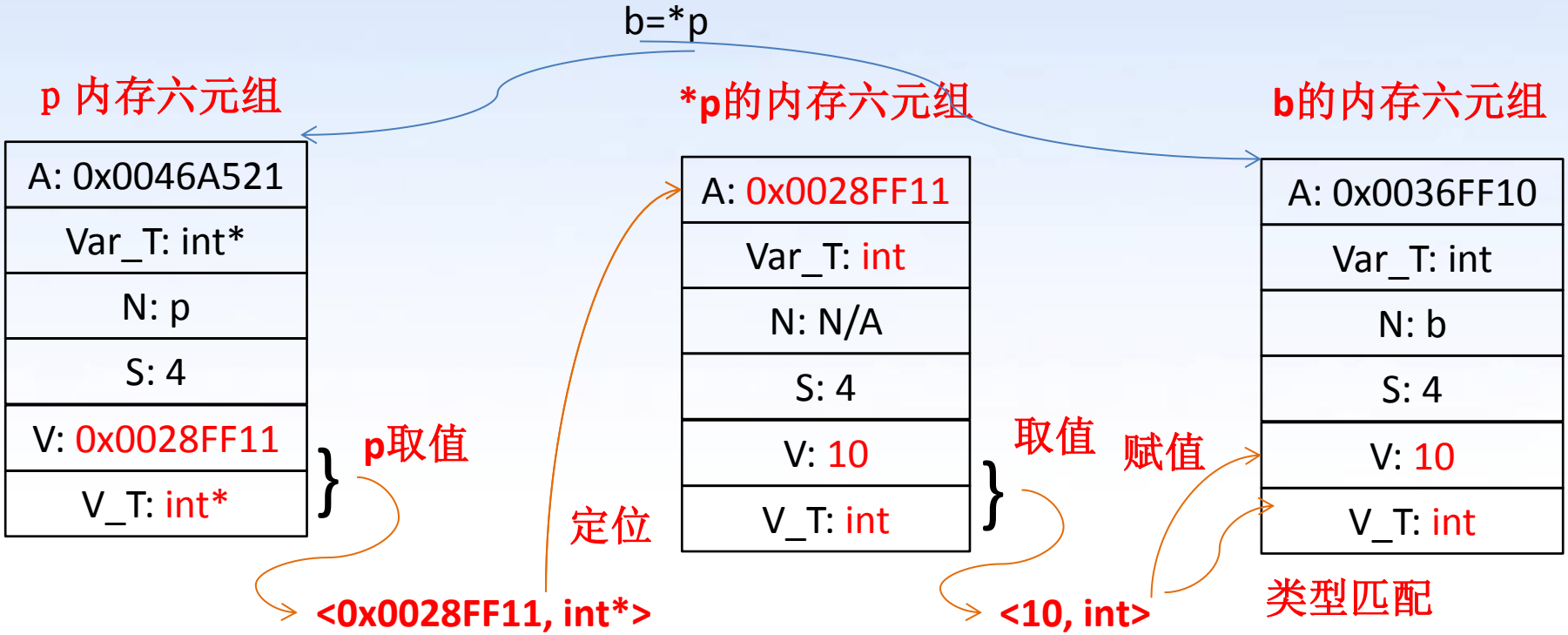
int* q=&(*p)发生了什么?


q=&(*p)
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
p取值
<0x0028FF11, int*>
*定位
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
&
取值
<0x0028FF11, int*>
q的内存六元组
A: 0x004F66F1
Var_T: int*
N: q
S: 4
V: 0x0028FF11
V_T: int*
赋值
类型匹配

size_t c=sizeof(*p)发生了什么?



int b=*p发生了什么？


b=*p
p 内存六元组
A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
p取值
<0x0028FF11, int*>
定位
*p的内存六元组
A: 0x0028FF11
Var_T: int
N: N/A
S: 4
V: 10
V_T: int
取值
<10, int>
赋值
b的内存六元组
A: 0x0036FF10
Var_T: int
N: b
S: 4
V: 10
V_T: int
类型匹配

总结：使用变量名定位内存

a;

定位a

a的内存六元组

A: 0x0028FF11
Var_T: int
N: a
S: 4
V: 10
V_T: int

int a = 10;

&a

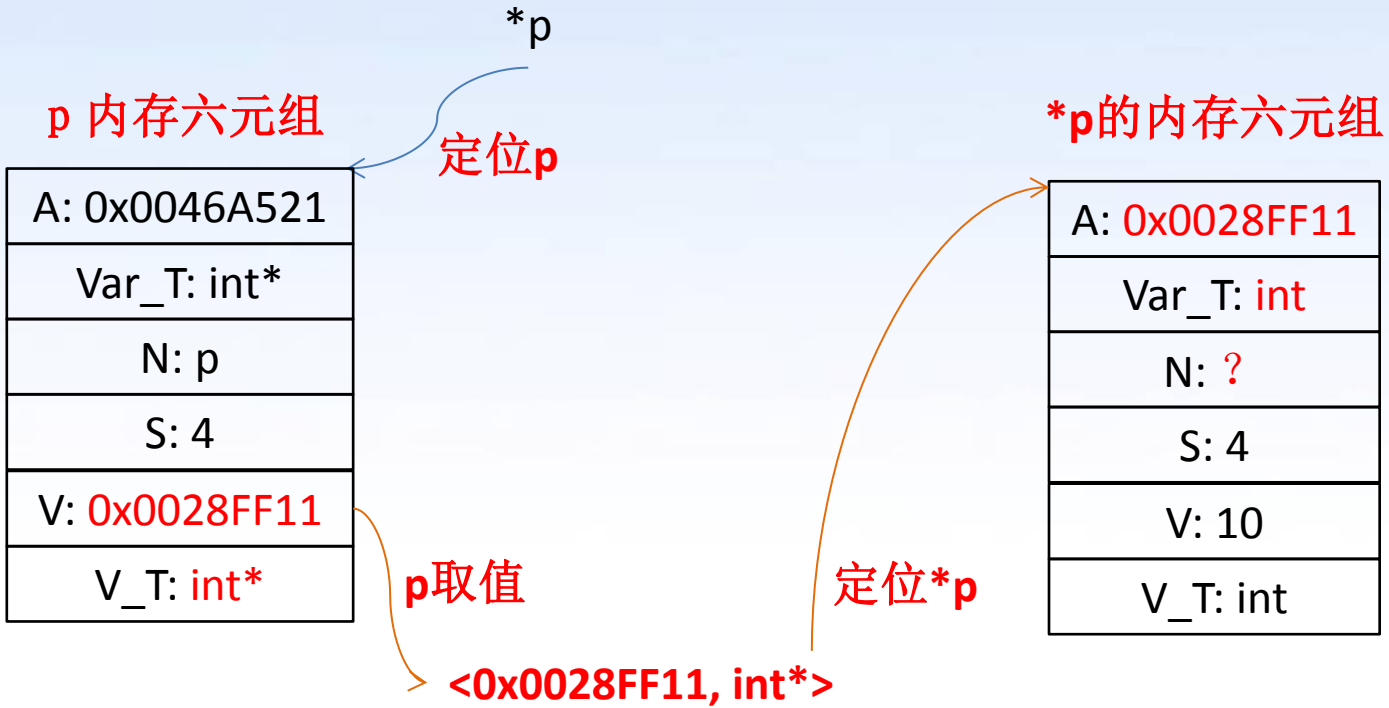
sizeof(a)

a

总结：使用指针变量类型定位内存

```
int a = 10;
int* p=&a;

&(*p)
sizeof(*p)
*p
```

思考：*p的时候 是否 知道 这块内存的 Name？

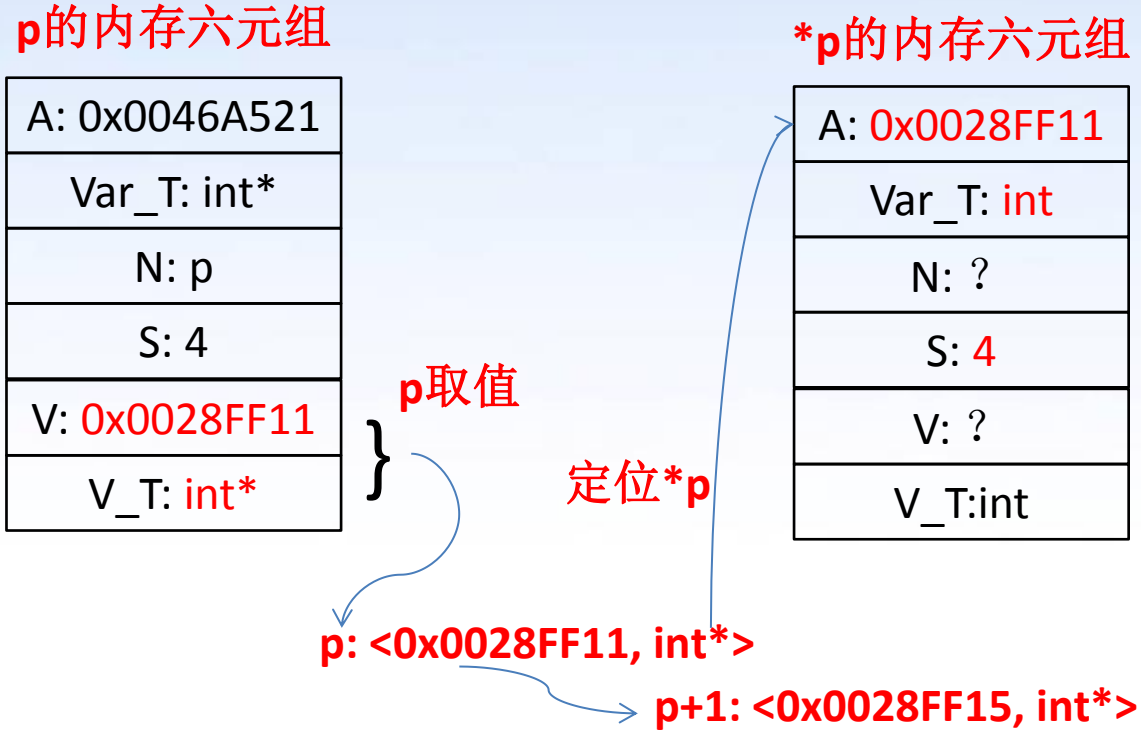


指针变量p+n到底是什么意思？

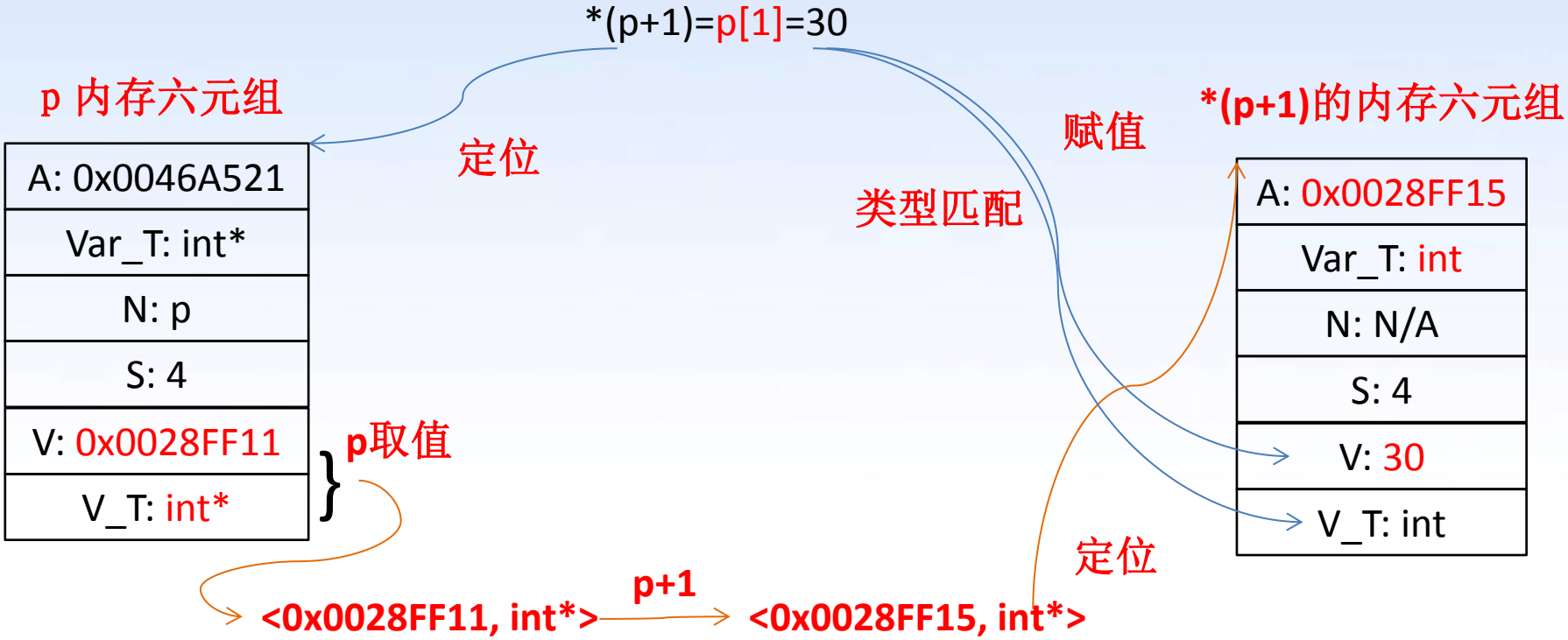
```
int a = 10;
int* p=&a;
```

p+n=？

1、定位p的内存
2、p取值<value, value_type>
3、p+n的value=
p的value+sizeof(*p)*n
4、p+n的Value_Type不变



*(p+1)=30发生了什么？有什么问题？



再来回顾一下指针到底是什么？

int* p;

1、我们首先看到int*，知道只是一个指针类型，其指向的变量类型为int
2、我们看到p，这是存储指针的4字节空间名称
3、这个p的value开始的空间是一个int
4、对p的Value指向的那个内存定位方法有*p或p[0]
对任何指针类型，*(p+n) = p[n]

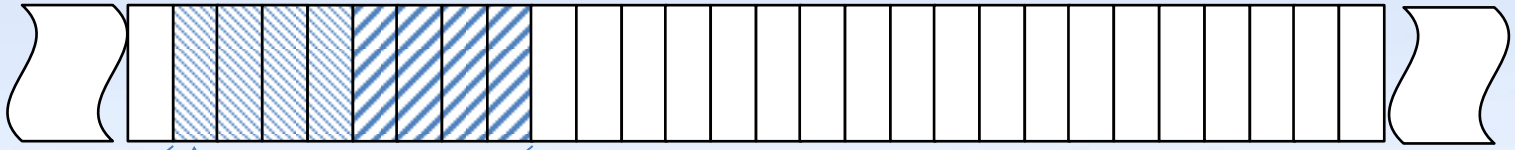
p 内存六元组

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

思考：为什么有p的Value和Value_Type能重构出一个int的内存描述

Tips：书上一般些int *p; 我们可以写成int* p，把int*写成一个整体，看成一个类型

再来观察int e[2]

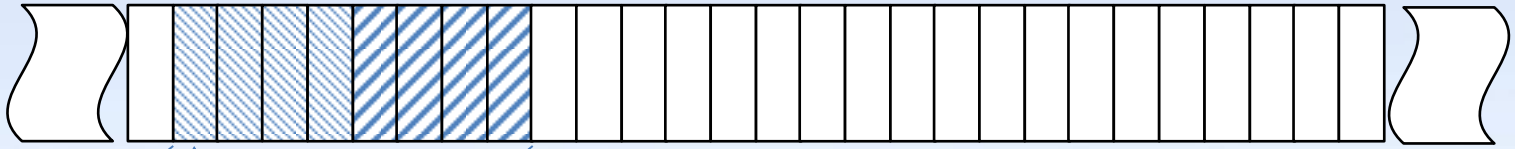


0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

思考1：size为什么是8？
思考2：Variable_Type是int[2]
思考3：value为什么是和Address怎么一样？
思考4：Type为什么是int*

&e的返回值是多少？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

} &e

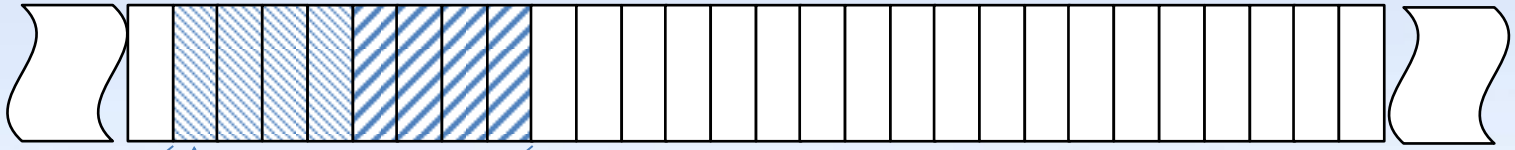
<0x0076AB11, int(*)[2]>

如何定义一个变量x=&e

x的类型必须是int(*)[2]

int (*x)[2] = &e

sizeof(e)的返回值是多少？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

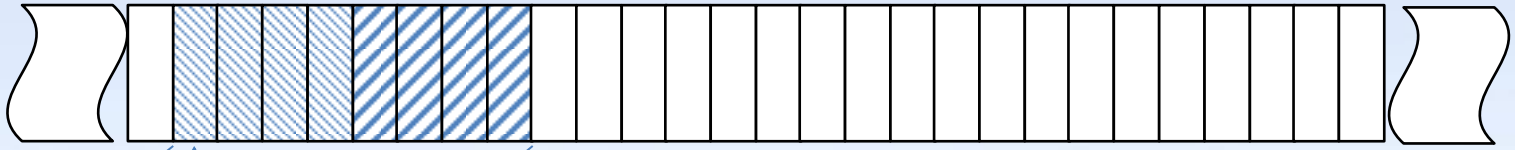
sizeof → <8, size_t>

如何定义一个变量x=sizeof(e)

x的类型必须是size_t

size_t x = sizeof(e)

e的返回值是多少？



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

取值

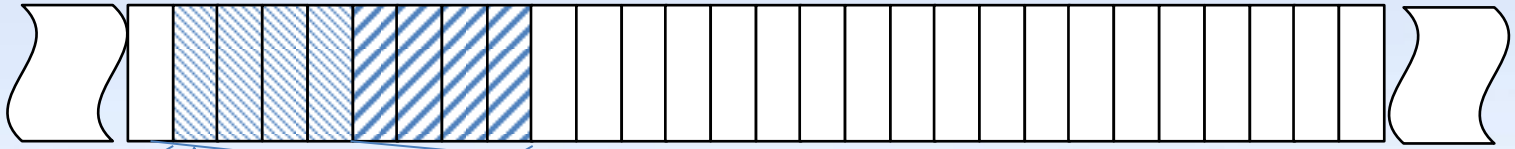
<0x0076AB11, int*>

如何定义一个变量x=e

x的类型必须是int*

int* x = e

e[0]到底是什么？&、sizeof、e[0]



0x0076AB11

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

取值

<0x0076AB11, int*>

e+0 <0x0076AB11, int*>

e[0]=*(e+0)
e的value+sizeof(*e)*0
e的type保持不变

定位

定位

A: 0x0076AB11
Var_T: int
N: N/A
S: 4
V: ?
V_T: int

e/(e+0)的内存六元组

e=e+1和e++为什么会报错

```
int main()
{
    int e[2];

    e = e + 1;

    return 0;
}
```

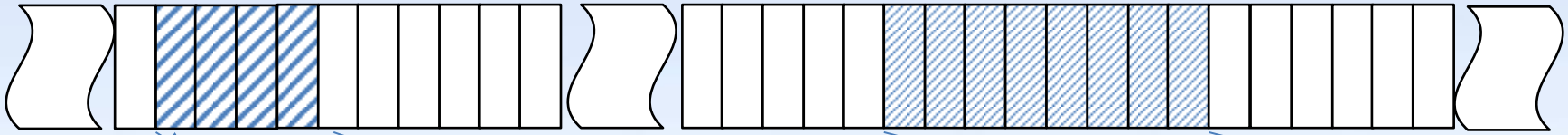
```
=== Build file: "no target" in "no project" (compiler: unknown) ===
In function 'main':
error: assignment to expression with array type
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

数组的Value一定是指向数组第一个元素的地址编号
对于数组e来说，V的值必须和A相等

因此，e并不是一个常量，它的值不能更改是语法限制的
对常量修改的错误应该是：assignment of read-only variable 'e'

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

int* p=e；p++为什么可以？



0x0046A521

0x0076AB11

```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

A: 0x0046A521
Var_T: int*
N: p
S: 4
V: 0x0076AB11
V_T: int*

p的内存六元组

<0x0076AB11, int*>

A: 0x0076AB11
Var_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*

e的内存六元组

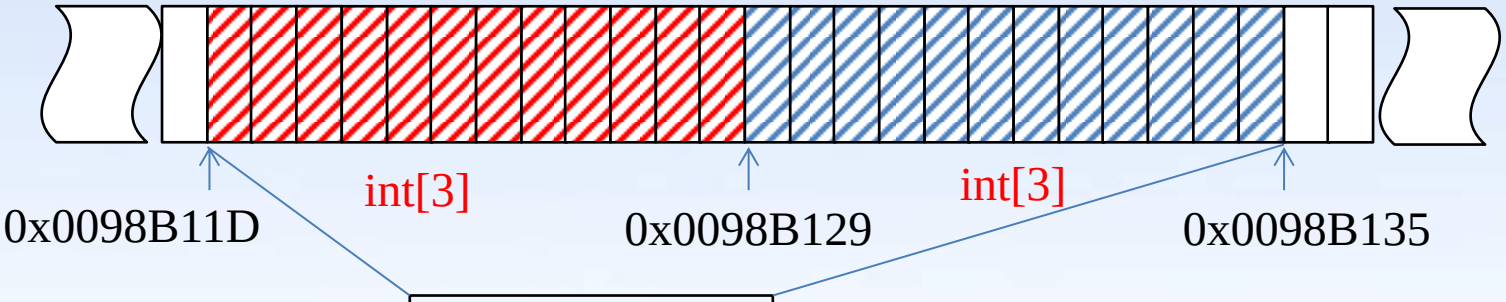
特殊的数组：字符串

```
int main()
{
    char str1[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
    char str2[6] = "hello";
    char* ptr = "hello";

    strcpy(str1, "hello world");
    strcpy(str2, "hello kitty");
}
```

思考1：str1是标准的数组初始化

思考2： str2初始化里面的”hello”取值和ptr初始化的里面”hello”取值有区别吗？

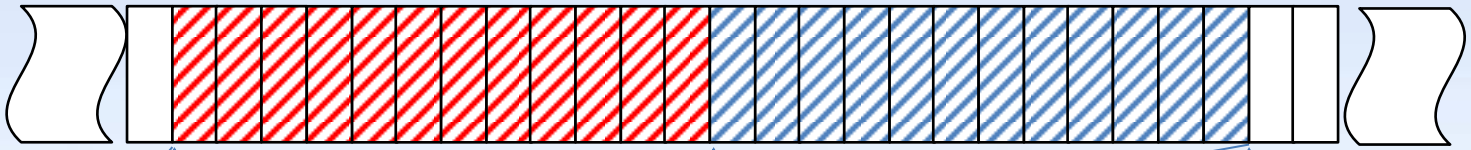
再来观察：int g[2][3]



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

思考1：size为什么是24？
思考2：Variable_Type是int[2][3]
思考3：value为什么是和Address怎么一样？
思考4：Type为什么是int(*)[3]

&g的返回值是多少？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

&g

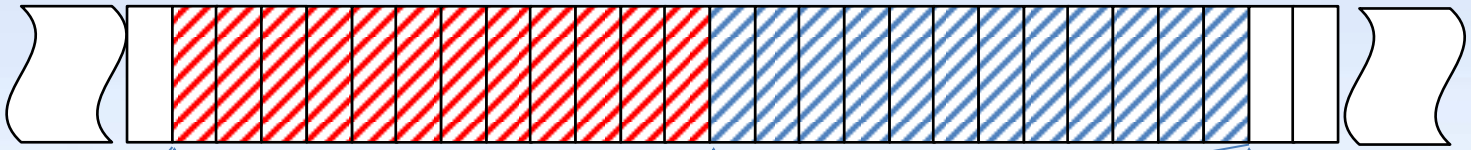
<0x0098B11D, int(*)[2][3]>

如何定义一个变量y=&g

y的类型必须是int(*)[2][3]

int (*y)[2][3] = &g

sizeof(g)的返回值是多少？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

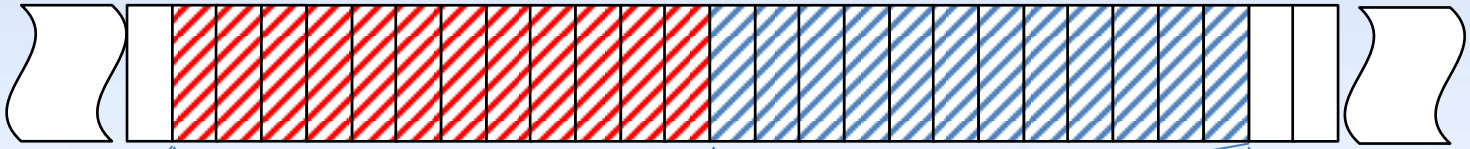
sizeof → <24, size_t>

如何定义一个变量y=sizeof(g)

y的类型必须是size_t

size_t y = sizeof(g)

g的返回值是多少？



A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

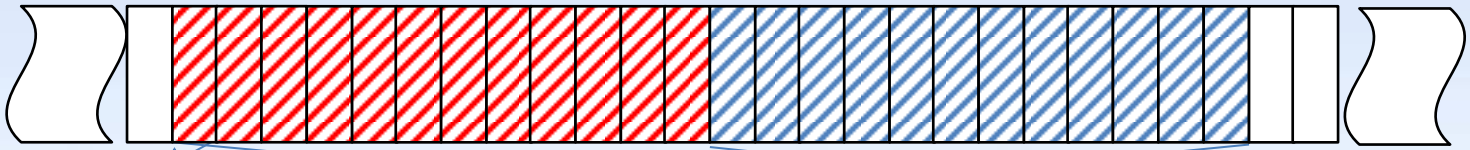
取值 <0x0098B11D, int(*)[3]>

如何定义一个变量y=g

y的类型必须是int(*)[3]

int (*y)[3] = g

g[1]到底是什么？&、sizeof、g[1]



0x0098B11D

int[3]

int[3]

A: 0x0098B11D
Var_T: int[2][3]
N: g
S: 24
V: 0x0098B11D
V_T: int(*)[3]

g的内存六元组

g[1]=*(g+1)

g的value+sizeof(*g)*1

g+1的type保持不变

取值

定位

<0x0098B11D, int(*)[3]>

g+1<0x0098B12F, int(*)[3]>

定位

A: 0x0098B11D
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B11D
V_T: int*

*g的内存六元组

A: 0x0098B12F
Var_T: int[3]
N: N/A
S: 12
V: 0x0098B12F
V_T: int*

*(g+1)的内存六元组

数组变量作为函数参数

```
void func1(int* pe)
{
    //do something
}

void func2(int (*pg)[3])
{
    //do something
}

int main()
{
    int e[2];
    int g[2][3];

    func1(e);
    func2(g);

    return 0;
}
```

main函数里面：

e: <0x0076AB11, int*> ⟶ pe: <0x0076AB11, int*>

func1和func2函数里面：

g: <0x0098B11D, int(*)[3]> ⟶ pg: <0x0098B11D, int(*)[3]>

C语言参数传递的机制是：**Pass By Value**

形参中：**int*=int[]，int(*)[3]=int[][3]**

数组中第一维信息的丢失是C语言

所有数组类型变量内存的取值机制造成的

int** pg当形参能行吗？

```
void func(int** pg)
{
    // do nothing
}

int main()
{
    int g[2][3] = {0};

    func((int**)g);
    return 0;
}
```

pg的内存六元组

A: 0x0036AA31
Var_T: int**
N: g
S: 24
V: 0x0098B11D
V_T: int**

<0x0098B11D, int**>

*pg的内存六元组

A: 0x0098B11D
Var_T: int*
N: g
S: 24
V: 0/NULL
V_T: int*

**pg

Value=0怎么来的？
会导致后续什么问题？

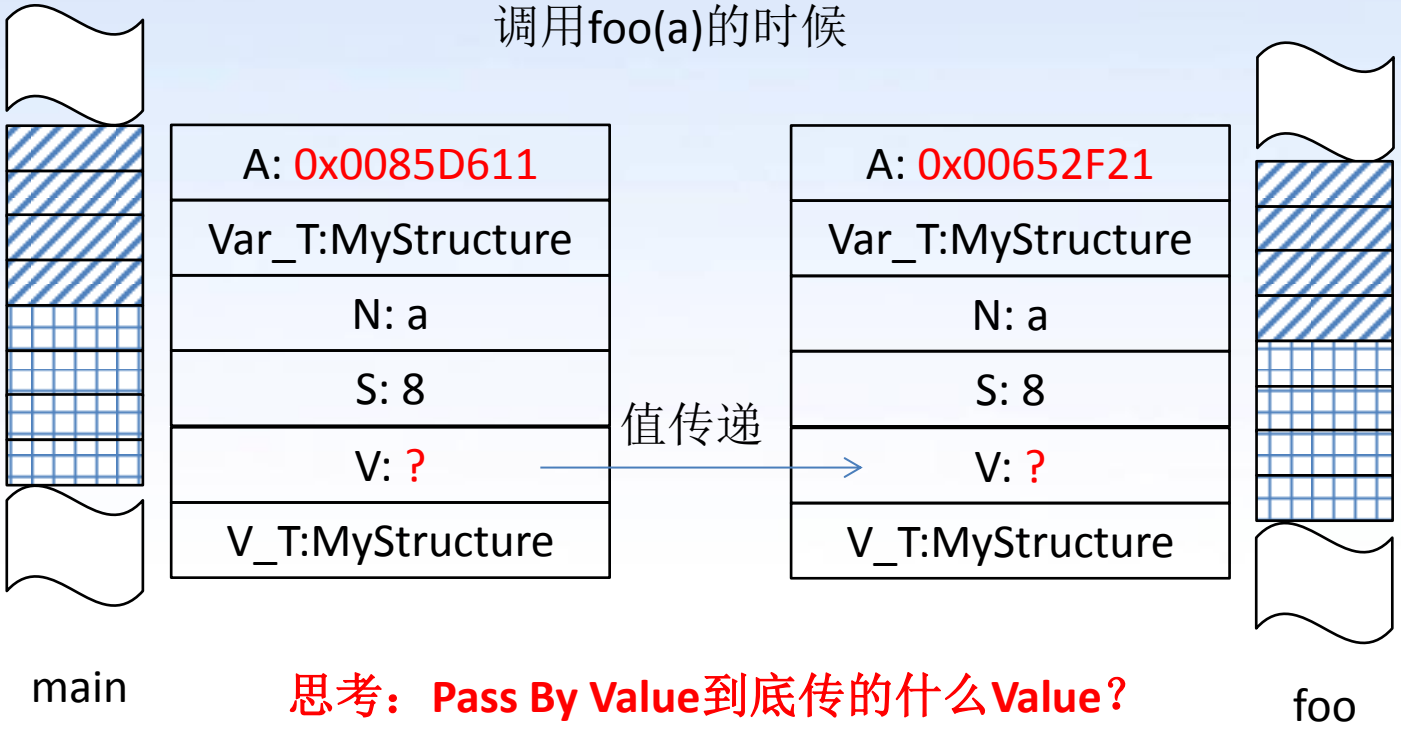
注意g[2][3]={0}

func((int*)g, 6)
func((int *)g, 2, 3)

void func(int* pg, int size)
void func(int* pg, int row, int column)

结构体作为函数参数

```c
typedef struct _MyStructure{
    int a;
    int b;
} MyStructure;

int foo(MyStructure a)
{
    a.a++;
}

int main()
{
    MyStructure a;

    a.a = a.b = 0;

    foo(a);

    printf("a.a=%d\n", a.a);

    return 0;
}
```



思考：Pass By Value到底传的什么Value？

malloc的问题

int* p = (int *)malloc(sizeof(int)*4);

p的内存六元组

A: 0x00652F21
Var_T: int*
N: p
S: 4
V: 0x00351728
V_T: int*

malloc的返回值
<0x00351728, void*>

(int *)强制转换

A: 0x00351728
Var_T: N/A
N: N/A
S: 16
V: N/A
V_T: N/A

提问：
int* vs. sizeof(int)*4

int* vs. sizeof(16)

int (*p)[4] vs. sizeof(16)

思考：malloc的返回值是void*，为什么必须强制转换成一种类型指针

回顾：int[2]，int[2][3]真的是类型吗？

Source: int[2], Target: VINT

typedef int VINT[2]
VINT是不是变量类型？

VINT e;
VINT* p = &e;
int (*q)[2] = &e;

Source: int[2][3], Target: VVINT

typedef int VVINT[2][3]

或

typedef int VINT[3]
typedef VINT VVINT[2]

再看sizeof()

int e[2]，以下表达式返回值是什么？

sizeof(e) vs. sizeof(e+1)

假设变量e的内存首地址为0x0076AB11

sizeof(内容有三类)

sizeof(int[2])：获得一个变量类型的大小
sizeof(e)：获得一个变量内存的大小
sizeof(e+1)：获得一个表达式返回值变量类型的大小

测试一下效果

int n[2][3][4][5]={0}，以下表达式返回值是什么？

&n, &n+1
n, n+1
n[0], n[0]+1
n[0][0], n[0][0]+1
n[0][0][0], n[0][0][0]+1
n[0][0][0][0], n[0][0][0][0]+1

假设变量n的内存首地址为0x00FF3811

知之、好之、乐之

谢谢！