左值与表达式

左值（lvalue）的概念和表达式紧密相关

在C语言中，由一系列操作符和操作对象组成的一个序列被称之为表达式（Exp）

An expression (Exp) is a sequence of operators and operands

基础表达式（Primary Expression）

1、标识符（Identifier）：包括变量名或函数名

2、常量（Constant）：10, 10.5, 'a'

3、字符串（String Literal）：”hello”

4、括号（Parenthesized Expression）：(Exp)等价于Exp

5、泛型选择（generic selection）：

后缀表达式（Postfix Expression）

给定一个表达式exp，与后缀操作符结合构成的仍然是一个表达式

1、**[]**: exp1[exp2]	a[3]; a[n+1];
2、**.**: exp.identifier	struct student h; h.name
3、**->**: exp->identifier	struct strudent* p; p->name
4、**++/--**: exp++, exp--	a++;
5、**(type-name){Initializer-list}**	(int){2}, (int[3]){1,2,3}
6、**()**: exp(argument $list_{opt}$)	func(1, 3.5);

一元表达式（Unary Expression）

给定一个表达式exp，与一元操作符结构构成的仍然是一个表达式

1、**++/--:** exp/--exp	++a; --a;
3、**&**: &exp	&a;
4、*: *exp	*p;
5、**+/-**: exp, -exp	+a; -a;
6、**~/!**: exp, !exp	~a; !a;
7、**sizeof**, **_Alignof**	sizeof(int), sizeof exp, _Alignof(int)

Cast表达式（Cast Expression）

给定一个表达式exp，与cast操作符结构构成的仍然是一个表达式

1、(type-name)exp　　(float)a;

运算表达式

给定两个个表达式exp，与运算操作符结构构成的仍然是一个表达式

Multiplicative Expression

1、*、/、%：exp1 * exp2; exp1 / exp2; exp1 % exp2

Additive Expression

2、+、-：exp1 + exp2; exp1 - exp2;

Bitwise Shift表达式

给定两个表达式exp，与位运算移位操作符结构构成的仍然是一个表达式

Shift Expression

1、<<、>>： exp1 << exp2; exp1 >> exp2

条件表达式

给定两个表达式exp，与条件操作符结构构成的仍然是一个表达式

Relational Expression

1、<、>、<=、>=：exp1<exp2、exp1>exp2、exp1<=exp2、exp1>=exp2

Equality Expression

2、==、!=： exp1 == exp2、 exp1 != exp2

位运算表达式

给定两个表达式exp，与位运算操作符结构构成的仍然是一个表达式

AND/Exclusive OR/Inclusive OR Expression

3、&、^、|： exp1 & exp2; exp1 ^ exp2; exp1 | exp2

逻辑、条件运算表达式

给定两个表达式exp，与逻辑运算操作符结构构成的仍然是一个表达式

Logic AND/OR Expression

1、&&、||：exp1 && exp2; exp1 || exp2

Conditional Expression

1、exp1 ? exp2 : exp3

赋值表达式（Assignment Expression）

给定两个表达式exp，与赋值运算操作符结构构成的仍然是一个表达式

=、*=、/=、%=、+=、-=、<<=、 >>=、&=、^=、|=

exp1 = exp2;

exp1 *= exp2;

exp1 /= exp2

…

逗号运算表达式（Comma Expression）

exp1, exp2, exp3, ... exp_n

从左至右按将逗号分隔的表达式依次运算，整个表达式返回值和类型是最后一个表达式的返回值和类型

The right operand is evaluated; the result has its type and value

多个表达式可以组合成新的表达式

a+1

1、a 基础表达式 （标识符）
2、1 基础表达式 （常量）
3、a+1 运算表达式 （加法操作符引导）

*(p+1)

1、p 基础表达式 （标识符）
2、1 基础表达式 （常量）
3、p+1 运算表达式 （加法操作符引导）
4、(p+1)基础表达式 （括号）
5、*(p+1) 一元表达式 （*引导）

总是从基础表达式(Primary Expression)开始

什么是左值（lvalue）

lvalue是一个表达式，且这个表达式能定位一个对象(Object)：

C语言中Type分为Object Type和Function Type

能定位一个Object的表达式，才是lvalue

什么是左值（lvalue）

lvalue是一个表达式，且这个表达式能定位一个对象(Object)：

1、变量标识符（identifier)	1、int **a**;
2、字符串（String Literal）	2、**”hello”**
3、*exp	3、***(p+1)**
4、exp1[exp2]	4、**e[1]**
5、(type-name){initializer}	5、**(int[3]){1,2,3}**
6、exp.identifier/exp->identifier	6、struct m_struct **h, *p;** **h.name, p->name**

所有的lvalue都能放到等号左边吗？

Modifiable lvalue才能被放到等号左边，条件是lvalue定位的内存：

1、对象类型不能是数组对象类型

2、对象类型不能是不完全类型，如extern int a[]

3、不能被const修饰，如int const a;

4、如果对象类型是struct/union，成员的对象类型也不能被const修饰

exp1 = exp2

“exp1”作为一个整体必须首先是一个lvalue，否则不可以放在等号左边

exp1如果是modifable lvalue，则可以被赋值，否则编译错误

如何观察lvalue

如果一个左值表达式在赋值表达式左边

1、如果该左值是一个Modifiable lvalue，等待赋值

2、如果该左值不是一个Modifiable lvalue，编译出错

其他情况下，lvalue表达式的观察方法如下：假设lvalue定位的对象Obj

Obj: <Address, Obj_T, Name, Size, Value, Value_Type>

1、如果lvalue跟&结合，则返回值：<Address，Obj_T*>

2、如果lvalue跟sizeof结合，则返回值<Size, size_t>

3、其他情况，返回值为<Value, Value_Type>

表达式的值(Value of Expression, rvalue)

表达式的值可以形式化定义为为：**<V, V_T>**

标识符（Identifier）-- 变量名：

非数组对象类型：int a = 10;

观察a，会发生lvalue conversion，得到对应内存的表示值

<10, int>

请参考之前关于数组变量和非数组变量的内容

表达式的值(Value of Expression, rvalue)

表达式的值可以形式化定义为为：**<V, V_T>**

标识符（Identifier）-- 变量名：

数组类型：int b[10]; int c[2][3];

返回值是第一个元素的首地址，返回值类型是元素对象类型对应的指针类型

我们用一个表达式tmp来对应表达式返回的值，观察tmp的对象类型

-- 表达式b的值就是<Address, int*>，int* tmp = b;

-- 表达式c的值就是<Address, int(*)[3]>，int (*tmp)[3] = c;

表达式的值(Value of Expression)

表达式的值可以形式化定义为为：**<V, V_T>**

-- 常量

int tmp = 10; double tmp = 10.5; char tmp = 'a';

-- 字符串

char* tmp = “hello”;

“hello”对应的内存对象类型为char[6]，请参看之前关于字符串的内容

简单总结一下：表达式

由变量名、常量和操作符可以构造成一个复杂的表达式（exp）

变量名：a, p

常量：10, 5

更复杂的表达式：a + 10, *(p+5),

左值：是一个能定位内存的表达式

右值：是一个表达式的返回值

左值 vs. 右值

```
int a = 10;
if (a = a + 1)
```

1、等号右边的a是一个表达式(a是lvalue），取a对应内存的表示值<10, int>，右值

2、等号右边的1是一个表达式，表示值<1, int>，右值

3、等号右边的a+1是一个表达式，表示值<11, int>，右值

4、等号左边的a是一个表达式(a是lvalue)，定位一块内存并存入一个值，左值

4、a = a + 1是一个表达式，是等号左边表达式的返回值<11, int>，右值

a = a + 1也是一个表达式，这个表达式是lvalue吗？ C和C++表现不同

进一步思考内存在左值时取值的问题

```
a = 12;
```

```
a+1 = 12;
```

```
error: lvalue required as left operand of assignment
```

1、左值要求必须是一块有效内存，a=12中，根据变量名a识别出了这块内存，没有结合任何操作符，因此还是一块合法的内存

2、在a+1=12中，因为有操作符存在，因此首先取得a所在内存的表示值<10, int>，<10, int>+<1, int>=<11, int>，左值现在是一个表达式的值，无法定位一块合法的内存

左值表达式如果不是在等号左边，或是其他表达式的子表达式，就变成了取值操作

示例1

int b[10]; size_t c;

c=sizeof(b)

-- 等号左边

1、c

这个c是lvalue

-- 等号右边

1、b是lvalue

2、和sizeof结合

3、取sizeof(b)表达式的值，

返回的是b对应内存的大小

示例2

int b[10]; size_t c;

c=sizeof(b+1-1)

-- 等号左边

1、c

这个c是lvalue

-- 等号右边

1、b是lvalue

2、b取value，1取value

3、b+1取value，1取value

4、b+1-1取value，

5、sizeof(b+1-1)获得int*类型大小

b+1-1不是lvalue

理解*exp形式的lvalue

1、通过变量名定位内存：即通过变量名来定位这段内存，例如：

int a; float b; double c;

通过a，b，c就可以定位到对应的内存

2、通过*运算符来定位内存：假设一个表达式expression的取值为<Value, Value_Type>，如果Value_Type是一个指针类型，则可以用*expression的方式来定位到Value对应字节编号开头的一段内存

我们再来看通过*运算符定位内存

用“*”来间接内存定位

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

为什么*p可以间接定位内存？

*p的时候发生了什么？

1、p是一个变量名，定位p的内存

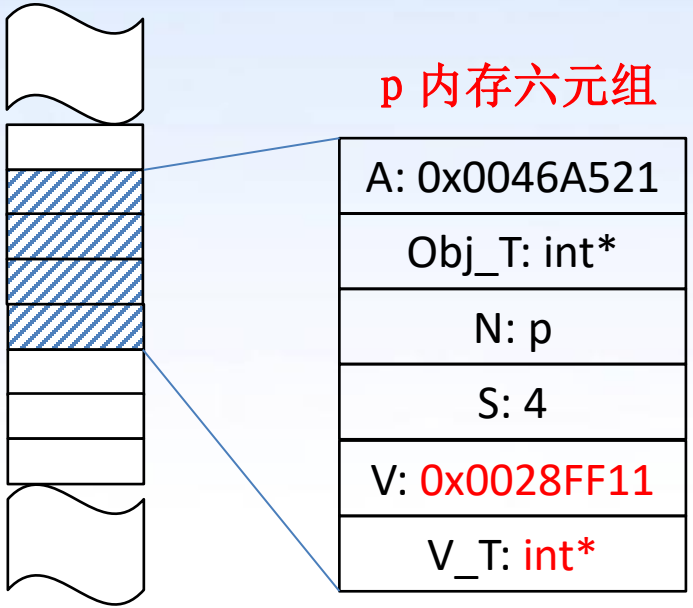
2、p前面没有&和sizeof，获得p的表示值<V, V_T>

3、根据<V, V_T>还原*p内存，规则如下：

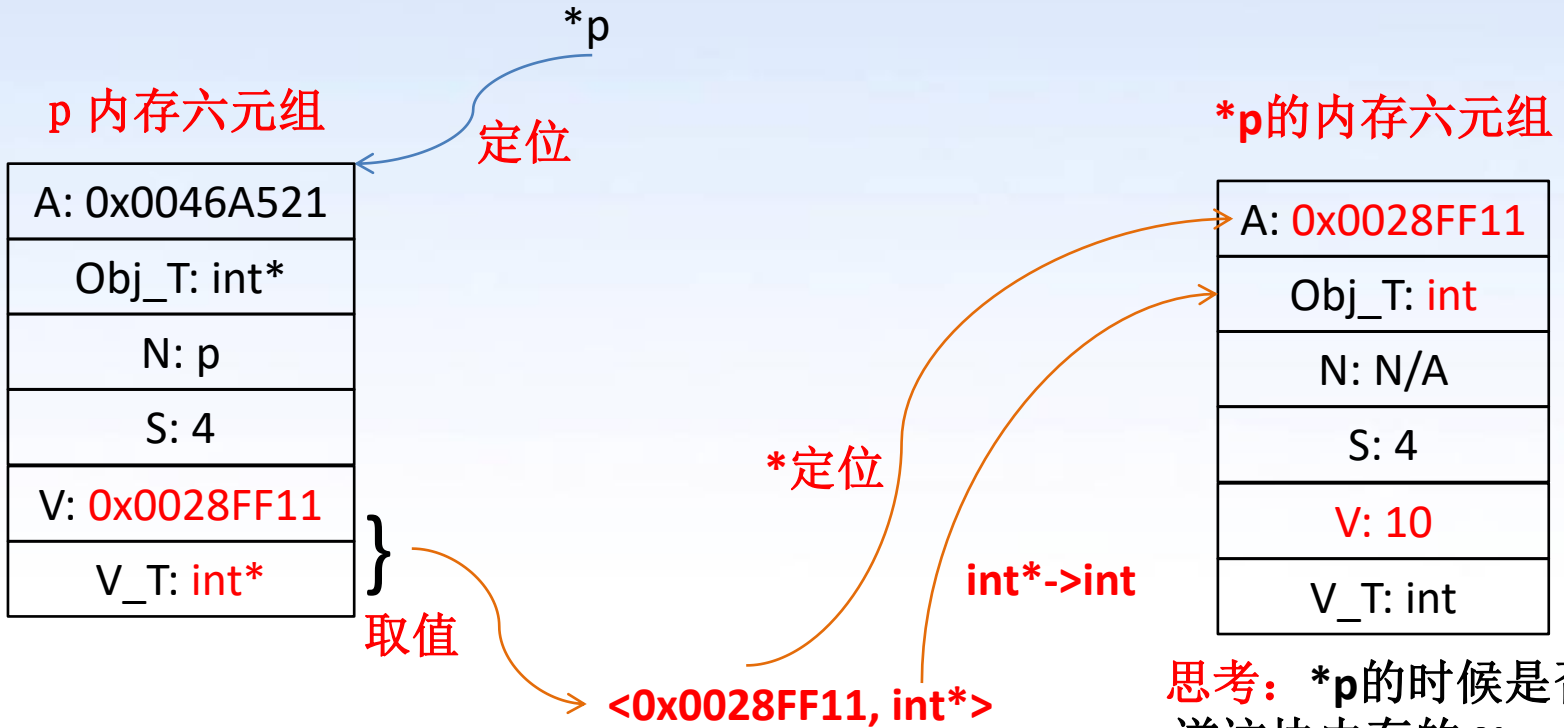
1）V的值是*p对应内存六元组的Address(A)

2）V_T必须是一个指针类型，其指向的对象类型就是*p内存六元组的Object_Type(Obj_T)

3）*p内存六元组其他属性根据V_T依次确定

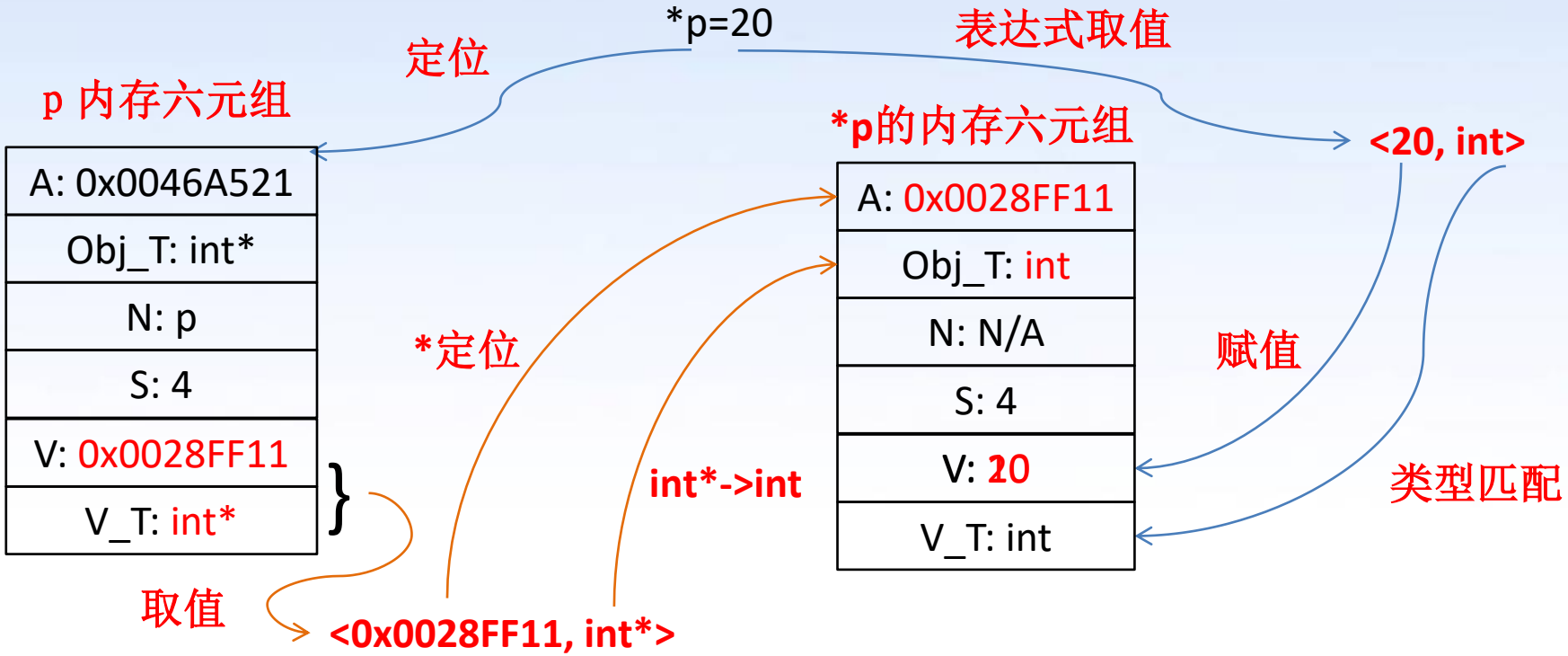


int a=10; int* p=&a; *p发生了什么？

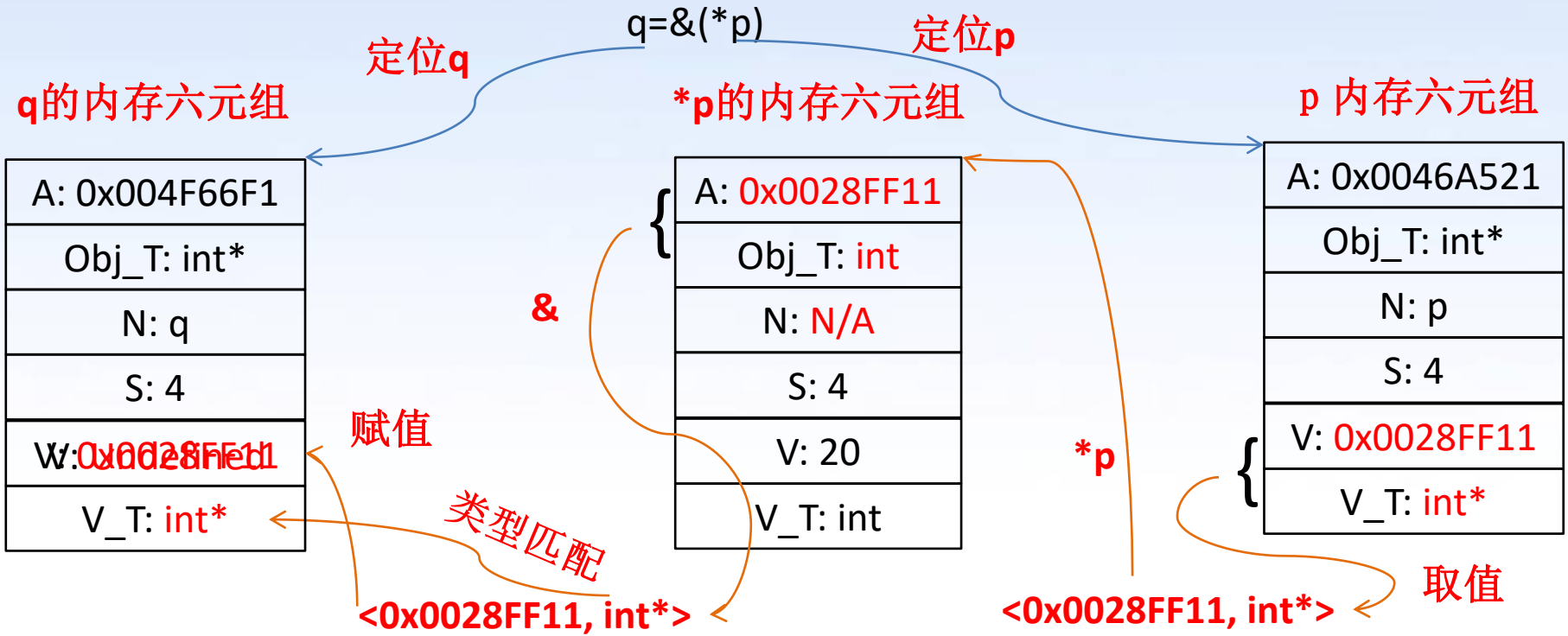


思考：*p的时候是否知道这块内存的 Name？

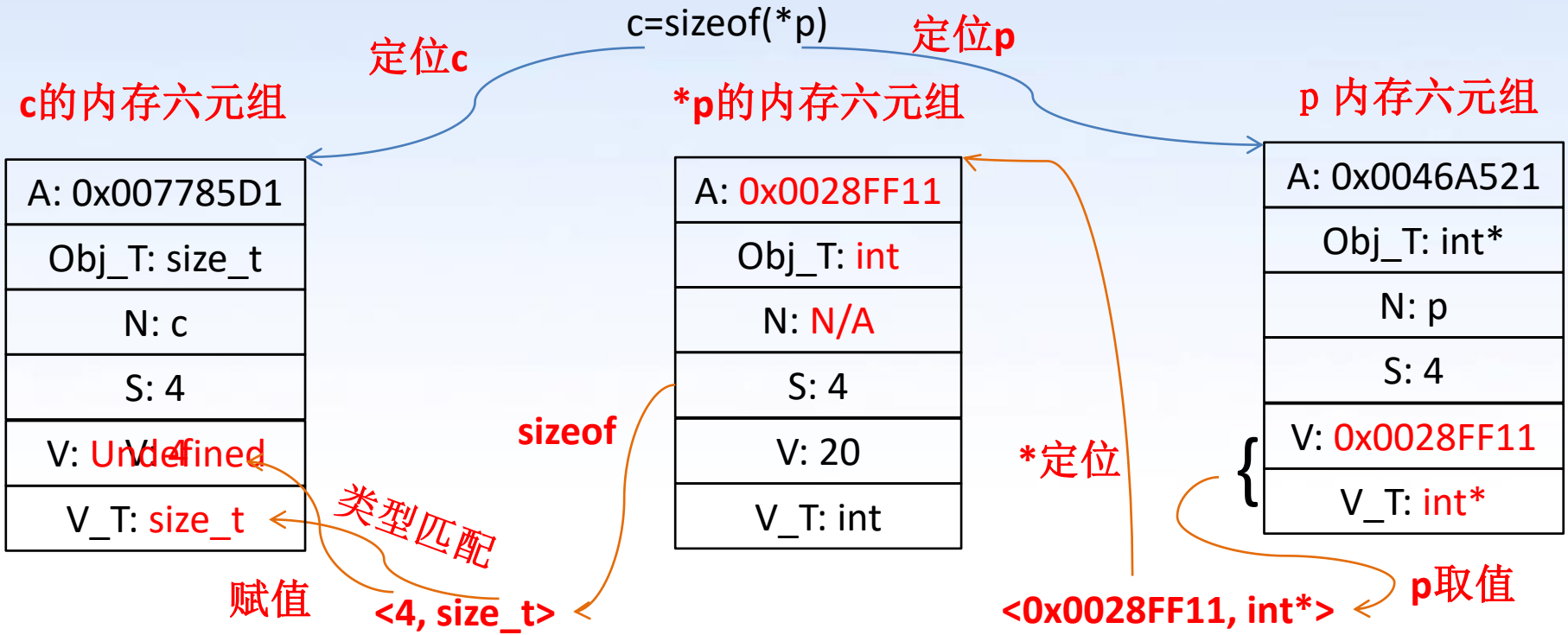
*p=20发生了什么？


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

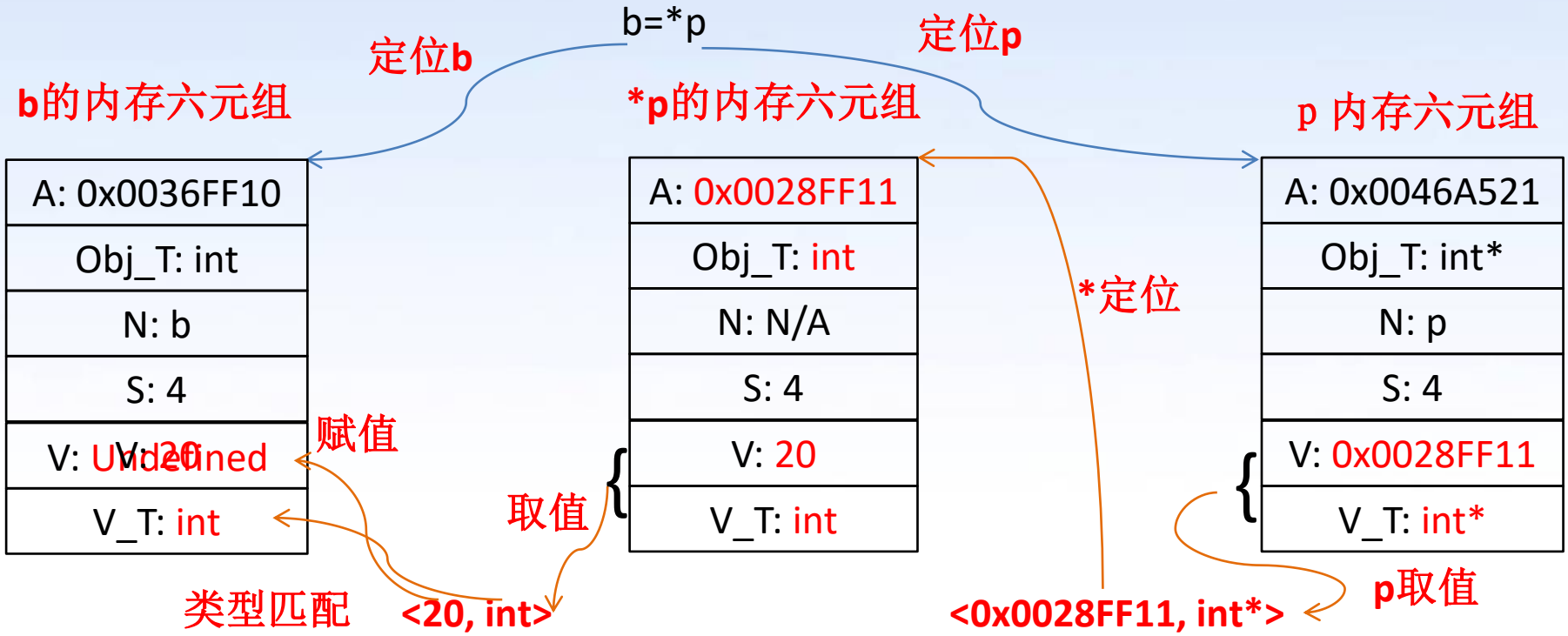
int* q=&(*p)发生了什么?


q=&(*p)
定位q
定位p
q的内存六元组
*p的内存六元组
p 内存六元组
A: 0x004F66F1
Obj_T: int*
N: q
S: 4
V: 0x0028FF11
V_T: int*
A: 0x0028FF11
Obj_T: int
N: N/A
S: 4
V: 20
V_T: int
A: 0x0046A521
Obj_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
&
赋值
类型匹配
<0x0028FF11, int*>
*p
<0x0028FF11, int*>
取值

size_t c=sizeof(*p)发生了什么?



int b=*p发生了什么？


b=*p
定位b
定位p
b的内存六元组
*p的内存六元组
p 内存六元组
A: 0x0036FF10
Obj_T: int
N: b
S: 4
V: Undefined
V_T: int
A: 0x0028FF11
Obj_T: int
N: N/A
S: 4
V: 20
V_T: int
A: 0x0046A521
Obj_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*
赋值
取值
*定位
类型匹配
<20, int>
<0x0028FF11, int*>
p取值

总结：使用变量名定位内存

int a;

a的内存六元组

定位a

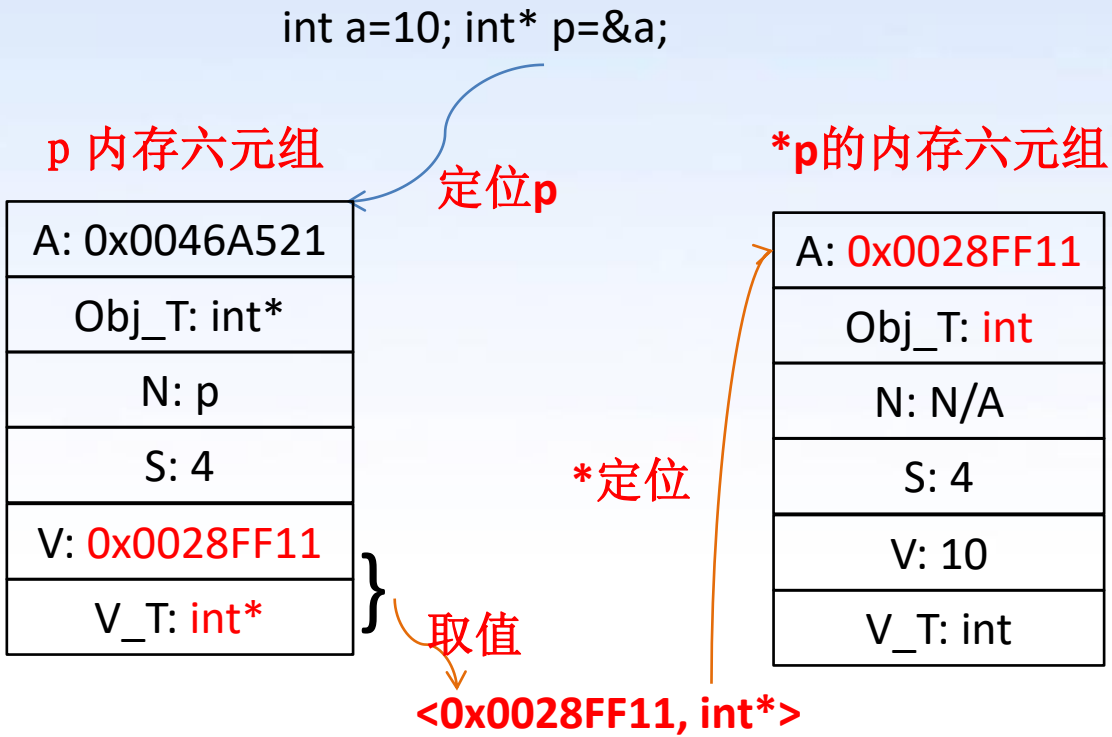
A: 0x0028FF11
Obj_T: int
N: a
S: 4
V: 10
V_T: int

1、a = 10;
等号左边定位内存

2、a+1=10; ✕
a跟+1结合，取a的表示值

3、&a;
4、sizeof(a);
5、a;

总结：使用指针对象类型定位内存

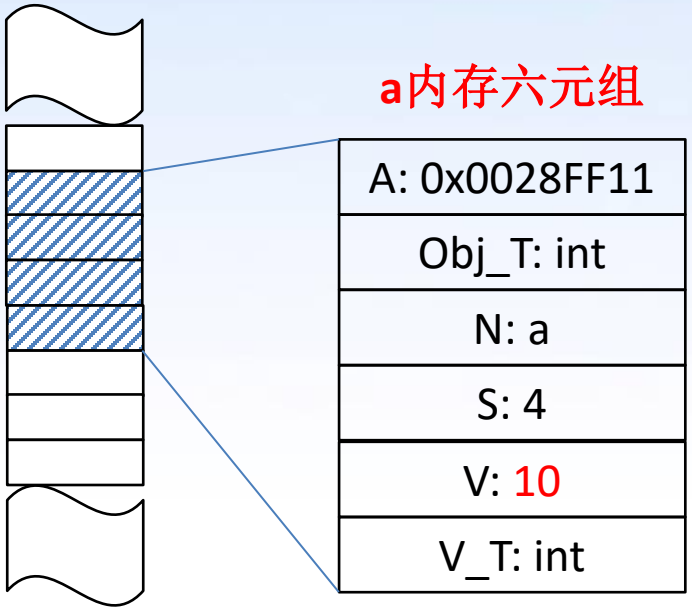


1、*p=20;
等号左边定位内存

2、*p+1=10; ✕
*p跟+1结合，取*p的表示值

3、&(*p);
4、sizeof(*p);
5、*p;

只有指针类型的值能用*进行间接定位




```
int a=10;
*a=20;
```

*a的时候发生了什么？

1、定位a的内存

2、获得a的表示值<10, int>

不是一个有效的指针类型，不能使用*操作符

```
error: invalid type argument of unary '*' (have 'int')
```

int b=*p+1 vs int b=*(p+1)

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

int b = *p+1;

1、定位p的内存
2、p取值<value, value_type>
3、定位*p的内存
4、取*p内存的表示值
5、表示值+1
6、将值写入变量b

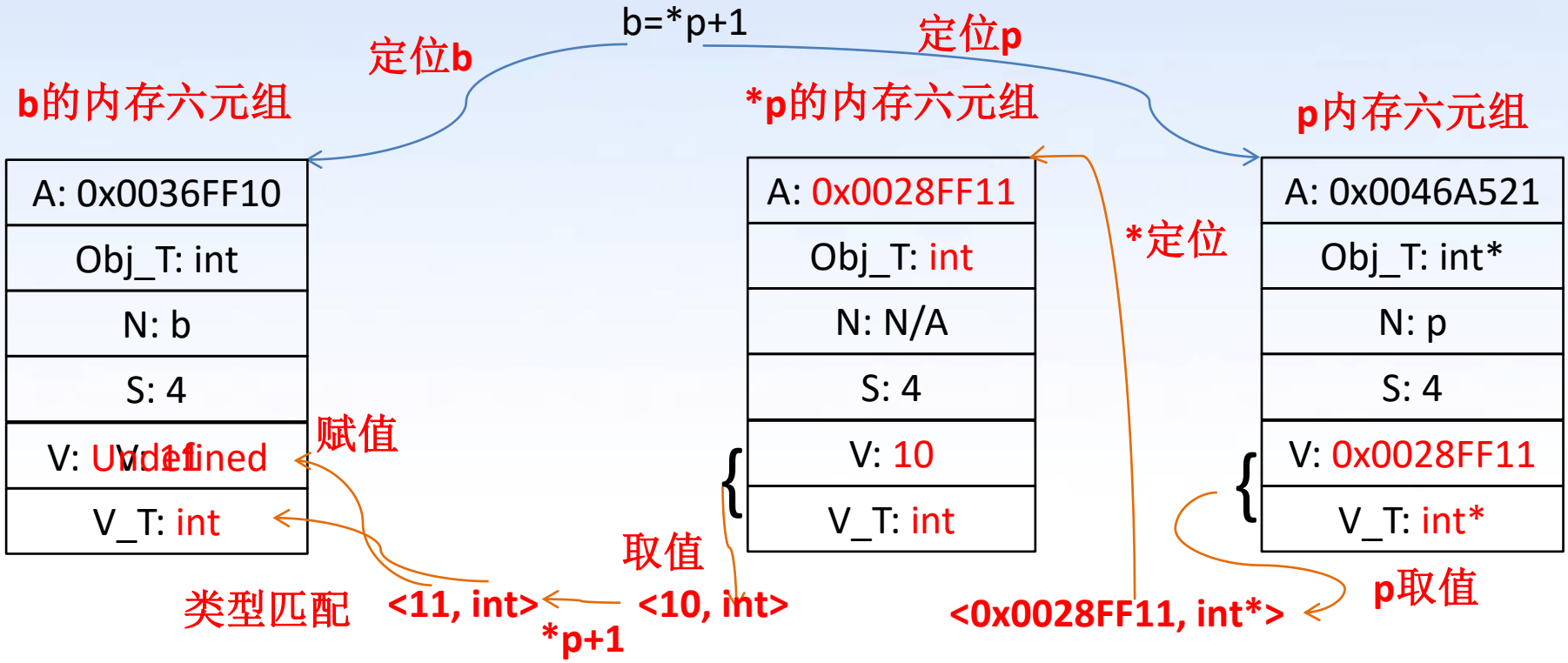
int b = *(p+1);

1、定位p的内存
2、p取值<value, value_type>
3、计算p+1的值
4、定位*(p+1)的内存
5、取*(p+1) 内存的表示值
6、将值写入变量b

p的内存六元组

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

int b=*p+1发生了什么？



int b=*(p+1)的计算过程

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

int b = *(p+1);

1、定位p的内存
2、p取值<value, value_type>
3、计算p+1的值
4、定位*(p+1)的内存
5、取*(p+1) 内存的表示值
6、将值写入变量b

p的内存六元组

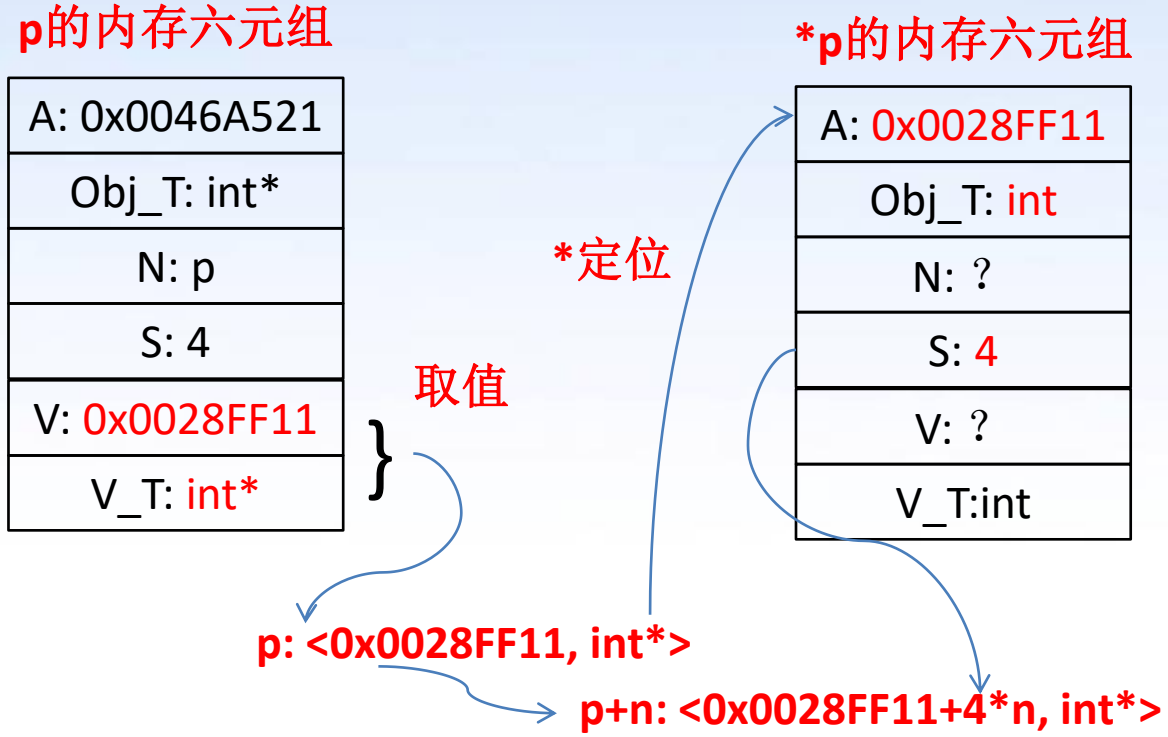
A: 0x0046A521
Obj_T: int*
N: p
S: 4
V: 0x0028FF11
V_T: int*

指针变量p+n到底是什么意思？

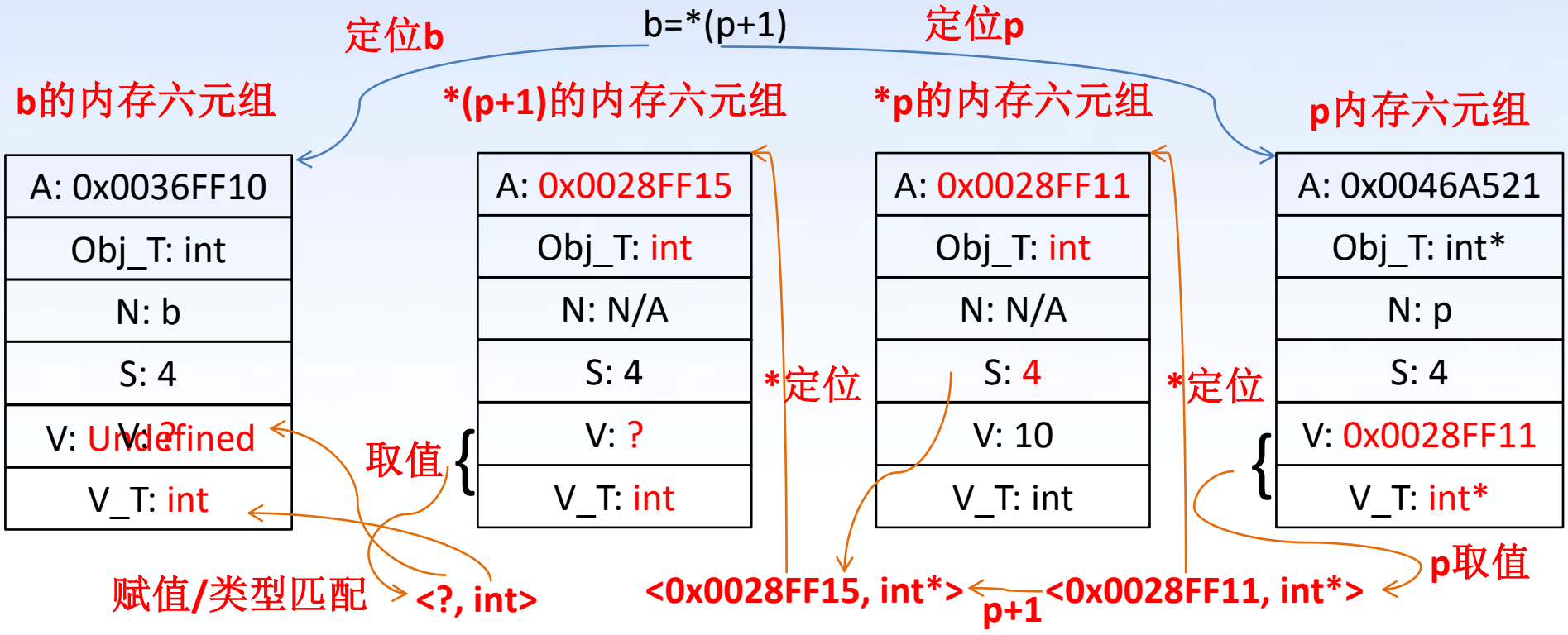
```
int a = 10;
int* p=&a;
```

p+n=？

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



int b=*(p+1)发生了什么?



这里有什么问题？溢出的危害

b=*(p+1)

b的内存六元组

A: 0x0036FF10
Obj_T: int
N: b
S: 4
V: Undefined
V_T: int

***(p+1)的内存六元组**

A: 0x0028FF15
Obj_T: int
N: N/A
S: 4
V: ?
V_T: int

取值

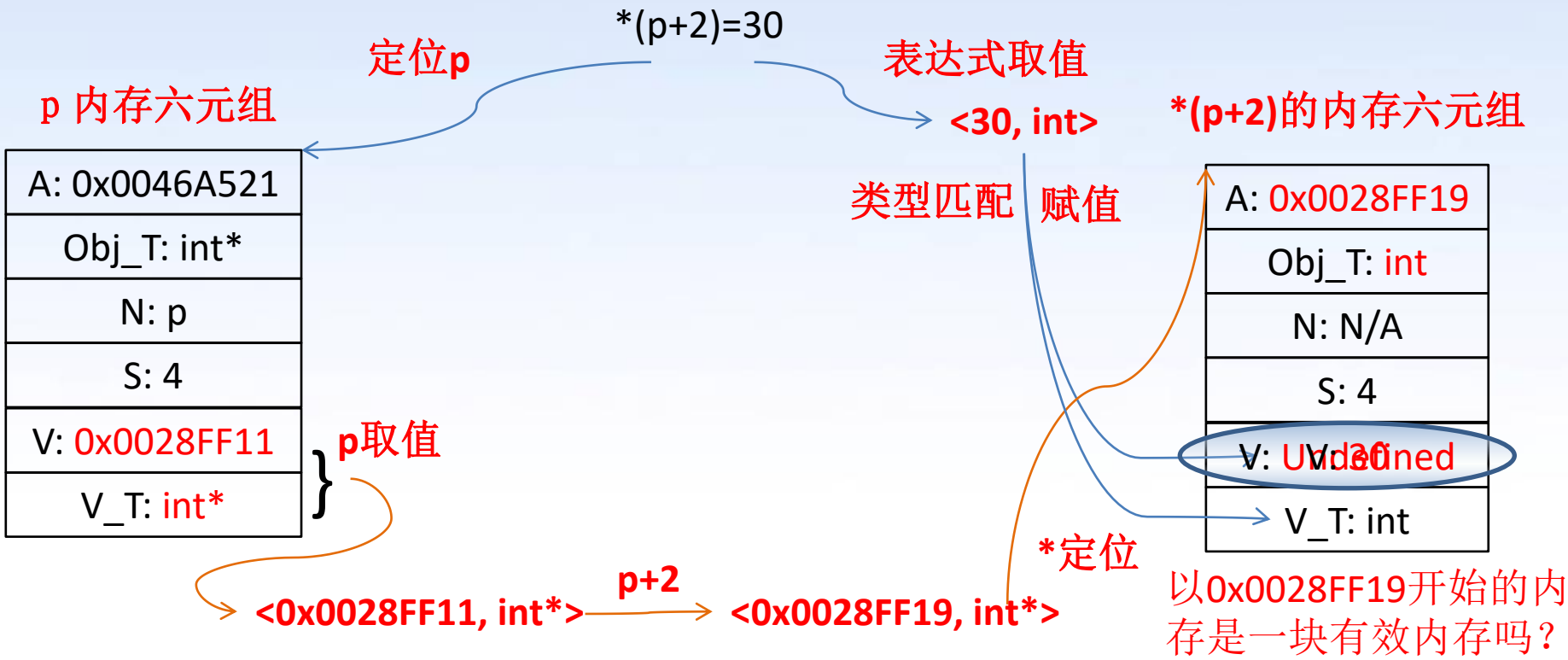
赋值/类型匹配 <?, int>

0x0028FF11是变量a所在内存的首地址

*(p+1)内存的起始位置是0x0028FF15

以0x0028FF15开始的内存是一块有效内存吗？

左值示例：*(p+2)=30，同样的问题



再来回顾一下指针到底是什么？

int* p;

*p;

1、p是一个表达式exp，该返回值类型是int*，合法指针类型
2、exp的返回值<Value, Value_Type>
3、*exp定位一个内存块M
4、M的Address的exp的Value
5、M的Obj_T是exp的Value_Type指向的类型

p是一个表达式

*(p+1);

1、p+1是一个表达式exp，该返回值类型是int*，合法指针类型
2、exp的返回值<Value, Value_Type>
3、*exp定位一个内存块M
4、M的Address的exp的Value
5、M的Obj_T是exp的Value_Type指向的类型

p+1是一个表达式

*(exp+n) <=> exp[n]

任何一个表达式exp，只要这个表达式返回值类型是一个有效地指针类型，则

*(exp+n) <=> exp[n]

int* p;

*p = *(p+0)	p[0]
*(p+1)	p[1]
*(p+2)	p[2]
...	...
*(p+n)	p[n]

思考题

1、int* p;
(p+1)[2]和p[3]的写法是否一样？

答案: *(exp+n) <=> exp[n]

1、*(p+1+2)，将p看作exp
等价于*(p+3)，即p[3]

2、*(p+1+2)，将p+1看作exp
等价于*((p+1)+2)，即(p+1)[2]

因此，(p+1)[2]和p[3]等价

指针指向的内存是数组吗？

给定一个int* p，可以通过偏移量随意进行内存访问p[n]，例如：
p[0], p[1], p[2], …., p[99], … ， 或
*p, *(p+1), *(p+2)… *(p+99),…

任何一个返回值为指针的表达式，蕴含着
指向的那块内存为一个数组，元素为指针对象类型对应的对象类型
但大小未知

C语言指针的偏移访问灵活，但危险性也很大

两个指针相减是什么意思？

```
int* p;
int* q;

p – q = ?
```

1、相减的两个指针类型必须一致

2、假设p: <Value1, int*>, q: <Value2, int*>

p – q: <(Value1-Value2)/sizeof(*p), ptrdiff_t>

思考题

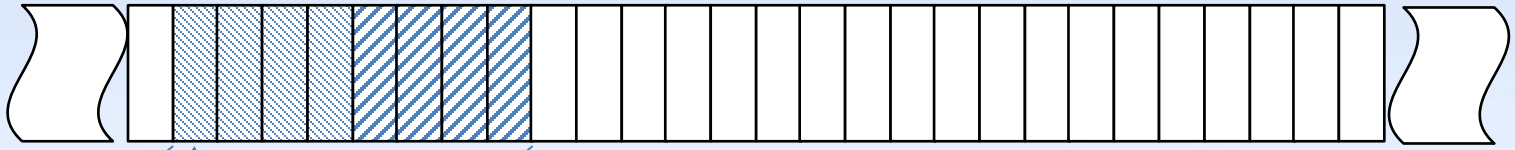
1、int(*p)[2][3]， p+2=?
假设p的值是<0x0035AB11, int(*)[2][3]>

2、int (*r)[30], int (*t)[30], r-t=?
假设t的值是<0x0035AB11, int(*)[30], r的值是<0x0035AC01, int(*)[30]>

答案
1、p+2的值是<0x0035AB41, int(*)[2][3]>，
因为sizeof(*p)的返回值是<24, size_t>

2、r-t的值: <2, ptrdiff_t>
(0x0035AC01-0x0035AB11)/sizeof(int[30]) = 2;

再来观察int e[2]

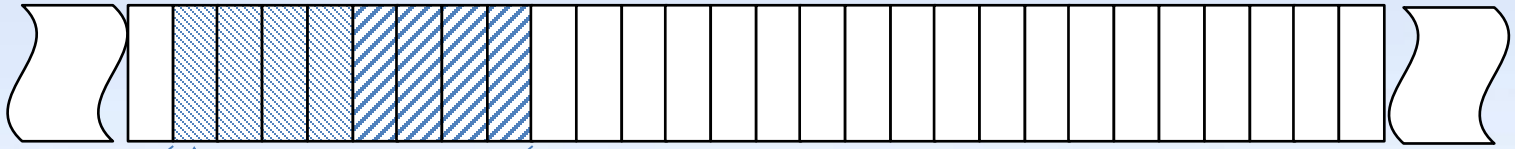


0x0076AB11

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

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

&e的返回值是多少？



0x0076AB11

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

} &e

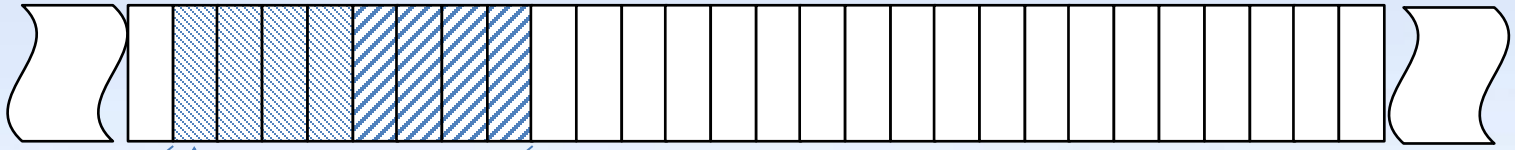
<0x0076AB11, int(*)[2]>

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

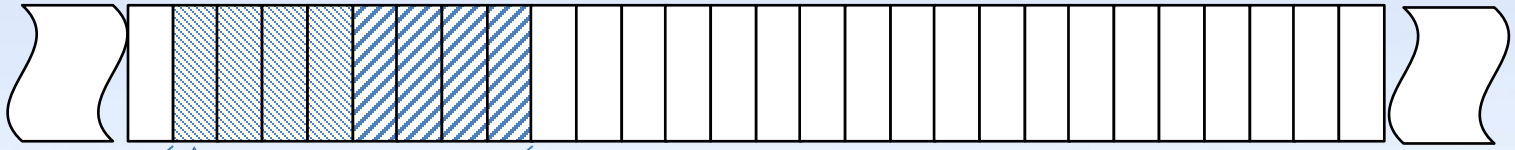
sizeof → <8, size_t>

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
Obj_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

再看sizeof()

int e[2]，以下表达式返回值是什么？

sizeof(int[2]) vs. sizeof(e) vs. sizeof(e+1-1)

假设变量e的内存首地址为0x0076AB11

sizeof(int[2])：8 获得一个对象类型的大小
sizeof(e)：8 获得一个变量内存的大小
sizeof(e+1-1)：4 获得一个表达式返回值对象类型的大小
e+1-1: <0x0076AB11, int*>

sizeof(e+1-1)并不会真的去计算一下e+1-1的值