简单回顾一下指针表达式的相关操作

假设对表达式exp进行evaluate， rvalue为<Value, Value_Type>（任何表达式都有rvalue）

如果Value_Type是一个对象指针类型，则可以用*exp的方式来定位一个对象M

1）该对象的Address为Value

2）该对象的Object Type为Value_Type对应的Referenced Type

3）对象其他属性随之确定

Referenced-Pointer的对应关系是指针关系的核心概念

int**的referenced type是int*

指针表达式的+操作

假设对表达式exp1进行evaluate， rvalue为<Value, Value_Type>（任何表达式都有rvalue）

假设对表达式exp2进行evaluate， rvalue的value_type为一个整数类型

则： exp1+exp2/exp2+exp1这个表达是的rvalue结果为：

<Value+exp2*sizeof(*exp1)， Value_Type>

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

p+n的值为<表达式p的值+n*sizeof(*p), int*>

指针表达式的+操作（续）

由于exp1+exp2或exp2+exp1这个表达是的rvalue的类型依然是一个对象指针类型

(exp1+exp2)或者(exp2+exp1)依然可以用之前*exp的规则来定位一个对象

将exp1+exp2或者exp2+exp2视做exp即可

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

*(p+5)依然是一个合法的lvalue表达式

指针表达式的+操作（续）

*(exp1+exp2)等价于exp1[exp2]或exp2[exp1]

C语言标准没有规定exp1或exp2哪一个必须在[]里面

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

p 等价于(p+0)/*(0+p)，即p[0]或0[p]

…

(p+n)/(n+p)，即p[n]或n[p]

指针表达式的–操作

假设对表达式exp1进行evaluate，rvalue为<Value, Value_Type>（任何表达式都有rvalue）

假设对表达式exp2进行evaluate，rvalue的value_type为一个整数类型

则：只能exp1-exp2，或者视为exp1+(-exp2)或(-exp2)+exp1

int* p;

*(p-1)等价于p[-1]或(-1)[p]

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>

注意两个指针相减的表达式的rvalue类型为ptrdiff_t

思考题

1、int(*p)[2][3]， p+2的rvalue相对于p的rvalue的offset是多少？

2、int (*r)[30], int (*t)[30], r-t=?
假设t的值是<0x0035AB10, int(*)[30]>, r的值是<0x0035AC00, int(*)[30]>

答案
1、Offset是48， 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;

int a; int* p=&a; *(p+1) 有什么问题

*(p+1)

对象p

A: 0x0068FE10
Obj_T: int*
N: p
S: 4
V: 0x0061FE10
V_T: int*
Align: 4

***p对象**

A: 0x0061FE10
Obj_T: int
N: N/A
S: 4
V: 1
V_T: int
Align: 4

***(p+1)对象**

A: 0x0061FE14
Obj_T: int
N: N/A
S: 4
V: Undefined
V_T: int
Align: 4

*定位

*定位

<0x0061FE10, int*> p+1 <0x0067FE14, int*>

读*(p+1)是一个未定义行为

左值示例：*(p+2)=30，同样的问题

*(p+2)=30

定位p

表达式取值

<30, int>

类型匹配 赋值

P对象

A: 0x0068FE10
Obj_T: int*
N: p
S: 4
V: 0x0061FE10
V_T: int*
Align: 4

p取值

*(p+2)对象

A: 0x0061FE18
Obj_T: int
N: N/A
S: 4
V: Undefined / V: 30
V_T: int
Align: 4

对*(p+2)赋值是一个未定义行为

*定位

<0x0061FE10, int*> —p+2→ <0x0061FE18, int*>

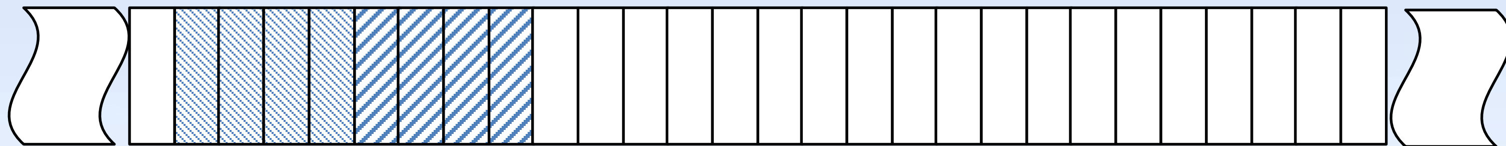
对象指针总是蕴含着对数组的访问

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),…

任何一个返回值为对象指针T*的表达式，蕴含着
指向的那块内存为一个数组，数组元素的对象类型为T
（即指针对象类型对应的Referenced Type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

再来观察int e[2]



A: 0x0082FB10
Obj_T: int[2]
N: e
S: 8
V: 0x0082FB10
V_T: int*
Align: 4

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：**Value_type为什么是int***
思考5：**Alignment为什么是4**

e[0]到底是什么?

0x0082FB10

A: 0x0082FB10
Obj_T: int[2]
N: e
S: 8
V: 0x0082FB10
V_T: int*
Align: 4

e[0]⇔*(e+0)

e+0: e的value+sizeof(*e)*0

type保持不变

取值

<0x0082FB10, int*>

*定位

e+0 < 0x0082FB10, int*>

*定位

A: 0x0082FB10
Obj_T: int
N: N/A
S: 4
V: ?
V_T: int
Align: 4

e[1]到底是什么？

0x0082FB10

A: 0x0082FB10
Obj_T: int[2]
N: e
S: 8
V: 0x0082FB10
V_T: int*
Align: 4

e[1]⇔*(e+1)
e+0: e的value+sizeof(*e)*1
type保持不变

取值

<0x0082FB10, int*>

e+1 < 0x0082FB14, int*>

*定位

A: 0x0082FB14
Obj_T: int
N: N/A
S: 4
V: ?
V_T: int
Align: 4

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)) ===
```

如果一个lvalue，定位的对象类型为数组对象，这个lvalue被称为Unmodifiable lvalue，不能对其整体赋值

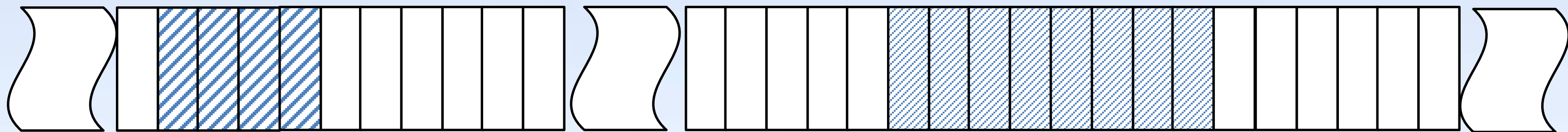
e、e[0]、e[1]的异同

Unmodifiable lvalue的含义是不能整体赋值

e对应的对象类型为int[2]，数组类型，无法对8个字节整体赋值

e[0]和e[1]对应的对象类型为int，非数组类型，可以赋值

int* p=e; p++为什么可以？



0x0068FE10

0x0082FB10

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

    p = p + 1;

    return 0;
}
```

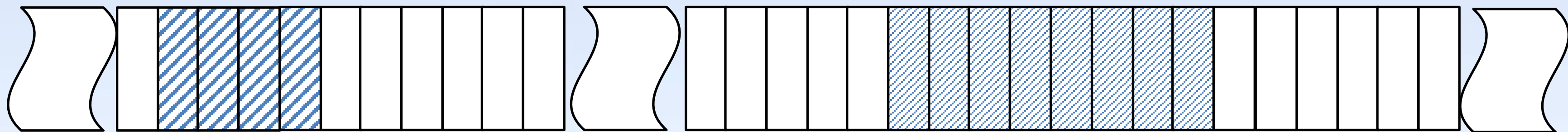
A: 0x0068FE10
Obj_T: int*
N: p
S: 4
V: Undefined 0x0082FB10
V_T: int*
Align: 4

p=e;

<0x0082FB10, int*>

A: 0x0082FB10
Obj_T: int[2]
N: e
S: 8
V: 0x0082FB10
V_T: int*
Align: 4

int* p=e；p++为什么可以？

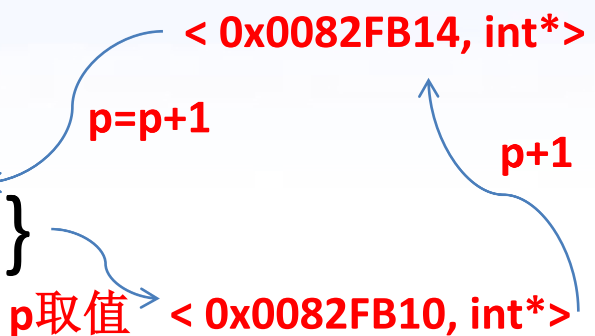


0x0068FE10

0x0082FB10

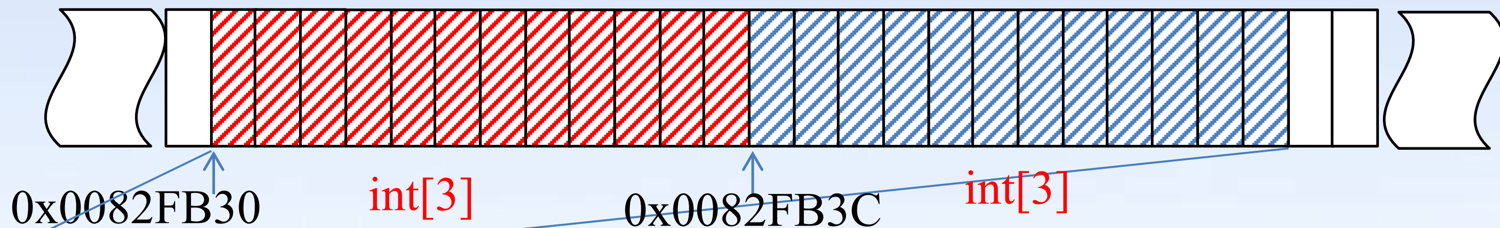
```
int main()
{
    int e[2];
    int* p = e;

    p = p + 1;

    return 0;
}
```

A: 0x0068FE10
Obj_T: int*
N: p
S: 4
V: 0x0082FB14
V_T: int*
Align: 4



修改的是对象p
没有修改对象e

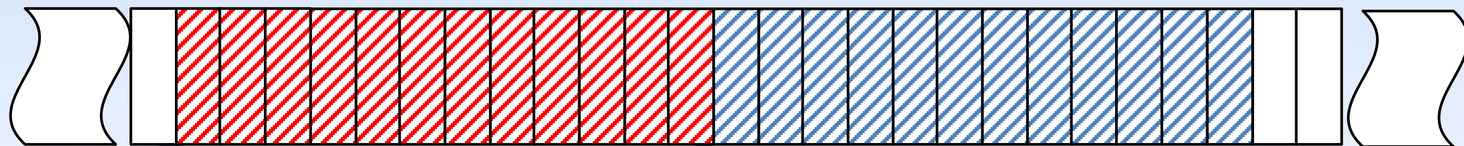
再来观察：int g[2][3]



A: 0x0082FB30
Obj_T: int[2][3]
N: g
S: 24
V: 0x0082FB30
V_T: int(*)[3]
Align: 4

思考1：size为什么是24？
思考2：Object_Type是int[2][3]
思考3：Value为什么是和Address一样？
思考4：Value Type为什么是int(*)[3]
思考5：Alignment为什么是4

g[0]到底是什么？



A: 0x0082FB30
Obj_T: int[2][3]
N: g
S: 24
V: 0x0082FB30
V_T: int(*)[3]
Align: 4

g[0]⇔*(g+0)

g+0: g的value+sizeof(*g)*0

type保持不变

取值

*定位

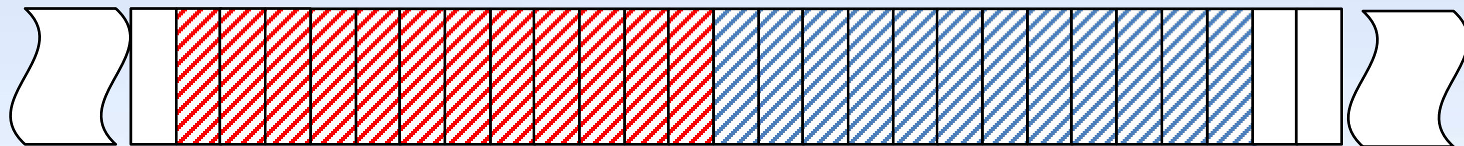
<0x0082FB30, int(*)[3]>

*定位

g+0: <0x0082FB30, int(*)[3]>

A: 0x0082FB30
Obj_T: int[3]
N: N/A
S: 12
V: 0x0082FB30
V_T: int*
Align: 4

g[1]到底是什么？



int[3]　　int[3]

A: 0x0082FB30
Obj_T: int[2][3]
N: g
S: 24
V: 0x0082FB30
V_T: int(*)[3]
Align: 4

g[1]⇔*(g+1)
g+1: g的value+sizeof(*g)*1
type保持不变

取值

<0x0082FB30, int(*)[3]>

g+1: <0x0082FB3C, int(*)[3]>

*定位

A: 0x0082FB3C
Obj_T: int[3]
N: N/A
S: 12
V: 0x0082FB3C
V_T: int*
Align: 4

理解这六个例子

给定一个int g[2][3] = {0};
修改g[1][2]的值

```
g[1][2] = 1;
printf("%d\n", g[1][2]);

(*(&g))[1][2] = 2;
printf("%d\n", g[1][2]);

(&(*g))[1][2] = 3;
printf("%d\n", g[1][2]);

(g+1-1)[1][2] = 4;
printf("%d\n", g[1][2]);
```

这四个赋值语句都成功的修改了g[1][2]

1、为什么？
2、有区别吗？

再来两个

```
1[g][2] = 5;
printf("%d\n", g[1][2]);
```

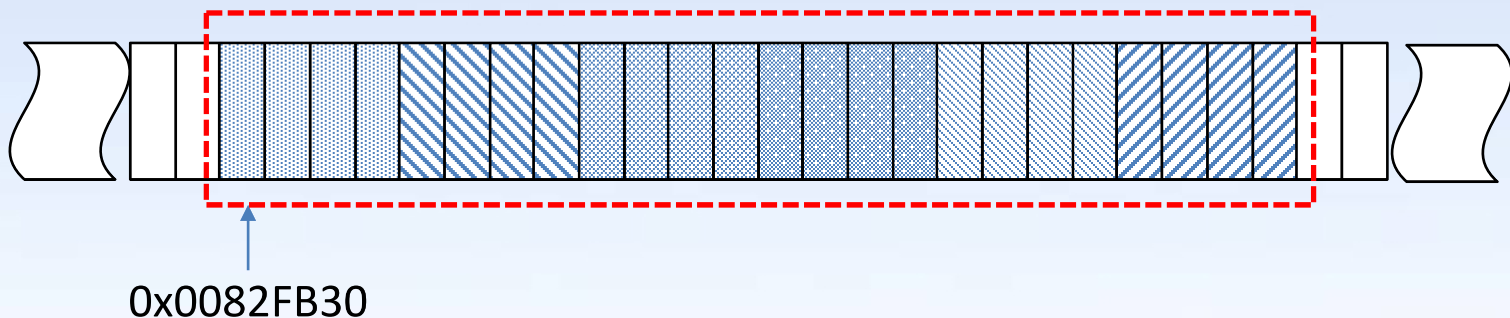
为什么也成功修改了g[1][2]

```
1[2][g] = 6;
```

为什么编译不过了？

这背后的工作机理到底是什么呢？
这几个语句并不是所谓的技巧，帮助理解数组名

int g[2][3]={0} 分配的这块内存属性

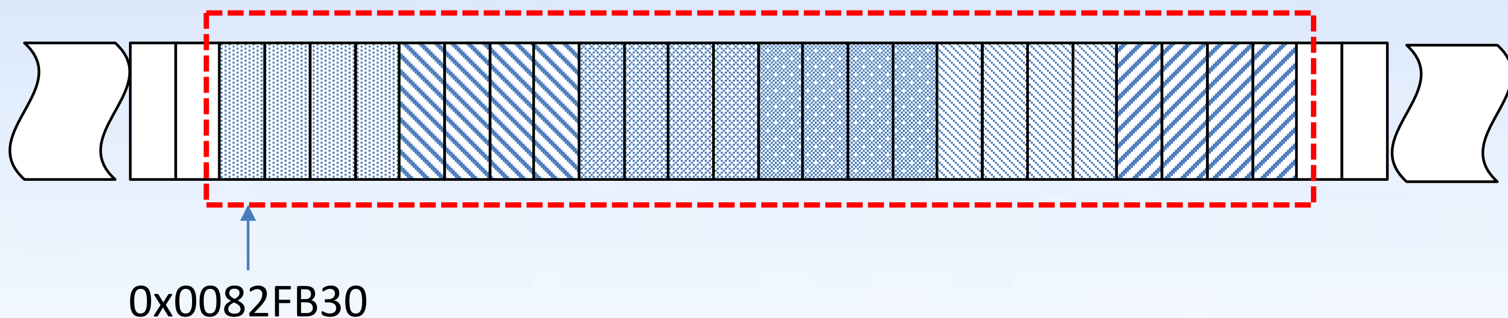


<0x0082FB30, int[2][3], g, 24, 0x0082FB30, int(*)[3], 4>

重要概念：非数组对象 vs. 数组对象 取值

现在这段内存全部都初始化为0了

给定g[1][2]这个表达式，先识别g

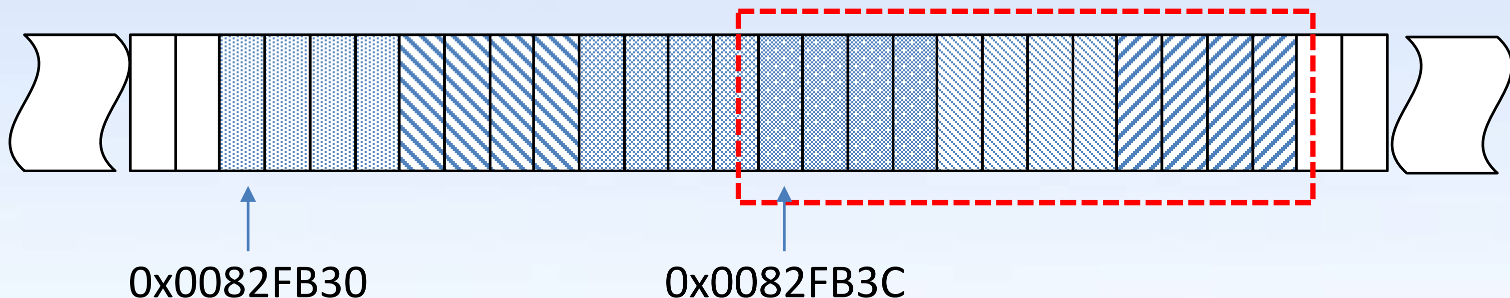


g[1][2]是一个表达式，这其中，基础表达式有g、1、2

g是一个identifier，是一个lvalue，因此可以定位到这块内存

<0x0082FB30, int[2][3], g, 24, 0x0082FB30, int(*)[3], 4>

给定g[1][2]这个表达式，识别a[1]



g是g[1]这个表达式的子表达式，因此表达式g的取值如下：

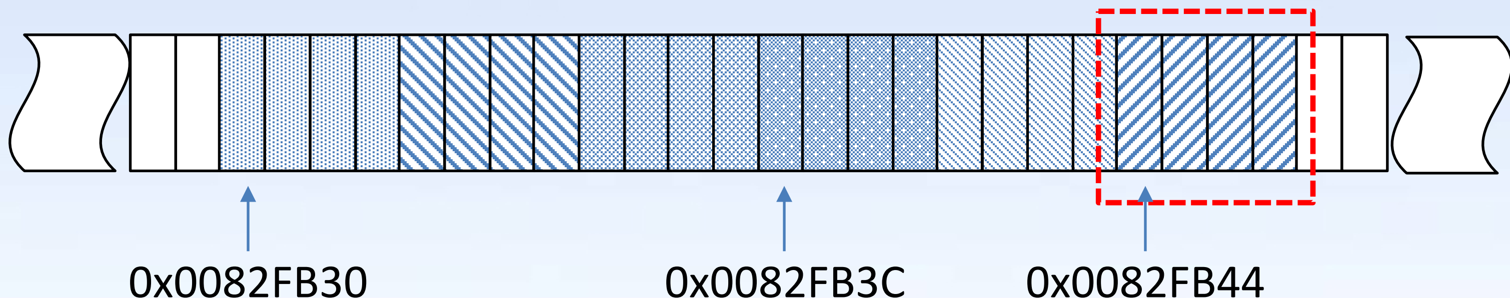
g: < 0x0082FB30, int(*)[3]>

g[1]等价于*(g+1)，下面运算g+1，获得值如下：

<0x0082FB30+1*sizeof(*a), int(*)[3]>，即<0x0082FB3C, int(*)[3]>

(g+1)定位的内存：<0x0082FB3C, int[3], 12, N/A, 0x0082FB3C, int , 4>

给定g[1][2]这个表达式，识别g[1][2]



g[1]是g[1][2]这个表达式的子表达式，因此表达式g[1]的取值如下：

g[1]: <0x0082FB3C, int*>

g[1][2]等价于*(g[1]+2)，下面运算g[1]+2，获得值如下：

<0x0082FB3C+2*sizeof(*(g[1])), int*>，即<0x0082FB44, int*>

*(g[1]+2)定位的内存：<0x0082FB44, int, 4, N/A, 0, int, 4>

如何正确地理解数组及其操作

我们来看一个C语言标准中的示例，给定int x[3][5]，如何理解x[i][j]

Here x is a 3 × 5 array of objects of type int; more precisely, x is an array of three element objects, each of which is an array of five objects of type int. In the expression x[i], which is equivalent to (*((x)+(i))), x is first converted to a pointer to the initial array of five objects of type int. Then i is adjusted according to the type of x, which conceptually entails multiplying i by the size of the object to which the pointer points, namely an array of five int objects. The results are added and indirection is applied to yield an array of five objects of type int. When used in the expression x[i][j], that array is in turn converted to a pointer to the first of the objects of type int, so x[i][j] yields an int.

强调int x[3][5]是一个一维数组，每个元素是一个int[5]

如何正确地理解数组及其操作

我们来看一个C语言标准中的示例，给定int x[3][5]，如何理解x[i][j]

Here x is a 3 × 5 array of objects of type int; more precisely, x is an array of three element objects, each of which is an array of five objects of type int. In the expression x[i], which is equivalent to (*((x)+(i))), x is first converted to a pointer to the initial array of five objects of type int. Then i is adjusted according to the type of x, which conceptually entails multiplying i by the size of the object to which the pointer points, namely an array of five int objects. The results are added and indirection is applied to yield an array of five objects of type int. When used in the expression x[i][j], that array is in turn converted to a pointer to the first of the objects of type int, so x[i][j] yields an int.

x是一个lvalue，定位到了一个对象，该对象类型是int[3][5]，
因此该对象的值是一个指向第一个int[5]元素类型的指针

如何正确地理解数组及其操作

我们来看一个C语言标准中的示例，给定int x[3][5]，如何理解x[i][j]

Here x is a 3 × 5 array of objects of type int; more precisely, x is an array of three element objects, each of which is an array of five objects of type int. In the expression x[i], which is equivalent to (*((x)+(i))), x is first converted to a pointer to the initial array of five objects of type int. Then i is adjusted according to the type of x, which conceptually entails multiplying i by the size of the object to which the pointer points, namely an array of five int objects. The results are added and indirection is applied to yield an array of five objects of type int. When used in the expression x[i][j], that array is in turn converted to a pointer to the first of the objects of type int, so x[i][j] yields an int.

先求x+i的值，也就是要跳过i个x指向的对象大小

如何正确地理解数组及其操作

我们来看一个C语言标准中的示例，给定int x[3][5]，如何理解x[i][j]

Here x is a 3 × 5 array of objects of type int; more precisely, x is an array of three element objects, each of which is an array of five objects of type int. In the expression x[i], which is equivalent to (*((x)+(i))), x is first converted to a pointer to the initial array of five objects of type int. Then i is adjusted according to the type of x, which conceptually entails multiplying i by the size of the object to which the pointer points, namely an array of five int objects. The results are added and indirection is applied to yield an array of five objects of type int. When used in the expression x[i][j], that array is in turn converted to a pointer to the first of the objects of type int, so x[i][j] yields an int.

*(x+i)，即x[i]就是定位一个从x+i地址开始的int[5]的对象

如何正确地理解数组及其操作

我们来看一个C语言标准中的示例，给定int x[3][5]，如何理解x[i][j]

Here x is a 3 × 5 array of objects of type int; more precisely, x is an array of three element objects, each of which is an array of five objects of type int. In the expression x[i], which is equivalent to (*((x)+(i))), x is first converted to a pointer to the initial array of five objects of type int. Then i is adjusted according to the type of x, which conceptually entails multiplying i by the size of the object to which the pointer points, namely an array of five int objects. The results are added and indirection is applied to yield an array of five objects of type int. When used in the expression x[i][j], that array is in turn converted to a pointer to the first of the objects of type int, so x[i][j] yields an int.

x[i]是x[i][j]这个表达式的子表达式，因为x[i]是一个lvalue，
定位一个类型为int[5]的对象，因此x[i]就取值成一个int*的指针

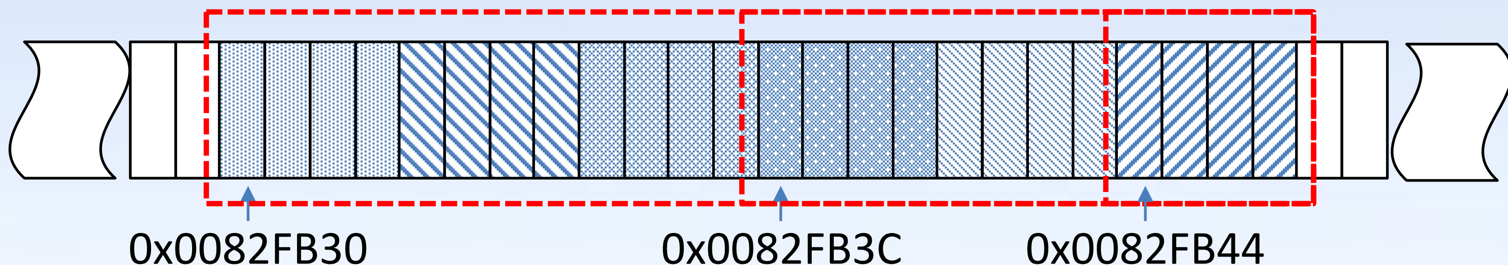
如何正确地理解数组及其操作

我们来看一个C语言标准中的示例，给定int x[3][5]，如何理解x[i][j]

Here x is a 3 × 5 array of objects of type int; more precisely, x is an array of three element objects, each of which is an array of five objects of type int. In the expression x[i], which is equivalent to (*((x)+(i))), x is first converted to a pointer to the initial array of five objects of type int. Then i is adjusted according to the type of x, which conceptually entails multiplying i by the size of the object to which the pointer points, namely an array of five int objects. The results are added and indirection is applied to yield an array of five objects of type int. When used in the expression x[i][j], that array is in turn converted to a pointer to the first of the objects of type int, so x[i][j] yields an int.

继续按照一维数组的方式计算地址偏移量然后定位对象
x[i][j]定位到了一个int类型的对象，其返回值类型就是int

g[1][2]=1的操作过程



1、识别基础表达式，包括a、1、2（等号左边）、1（等号右边）

2、g是lvalue，定位这24个字节内存（对象类型int[2][3]）

3、g是g[1]子表达式，没有跟&、sizeof、typeof结合，g取值: <0x0082FB30, int(*)[3]>

4、g[1]等价于*(g+1)，首先计算g+1: <0x0082FB3C, int(*)[3]>

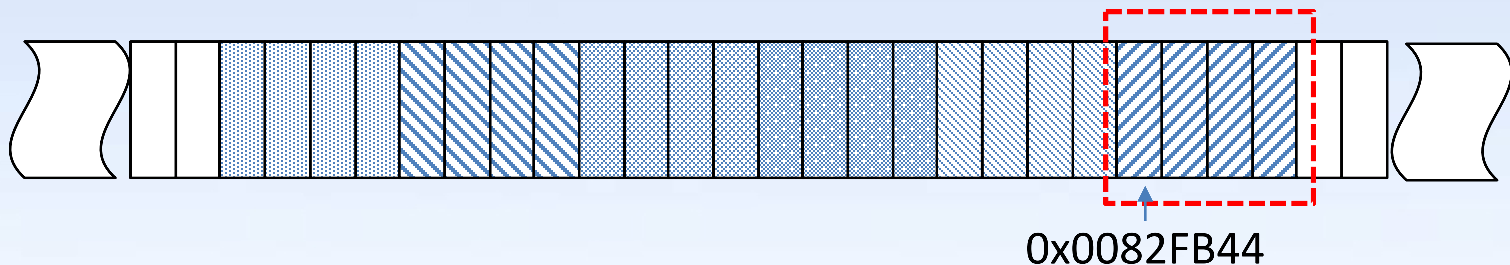
5、观察*(g+1)，这个表达式是lvalue，定位这12个字节内存（对象类型int[3]）

6、g[1]是g[1][2]子表达式，没有跟&、sizeof、typeof结合，g[1]取值: <0x0082FB3C, int*>

7、g[1][2]等价于*(g[1]+2)，首先计算g[1]+2: <0x0082FB44, int*>

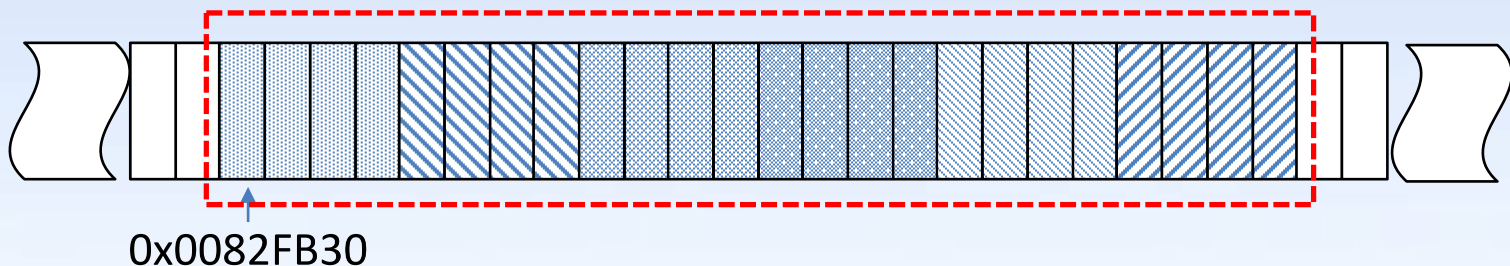
8、观察*(g[1]+2)，这个表达式是lvalue，定位这4个字节内存（对象类型int）

g[1][2]=1的操作过程



g[1][2]的对象类型是int，是modifiable lvalue，可以放到等号左边赋值

(*(&g))[1][2] = 2的操作过程



1、识别基础表达式，包括g、1、2（等号左边）、2（等号右边）

2、g是lvalue，定位这24个字节内存（对象类型int[2][3]）

3、g和&结合，&g返回值: <0x0082FB30, int(*)[2][3]>

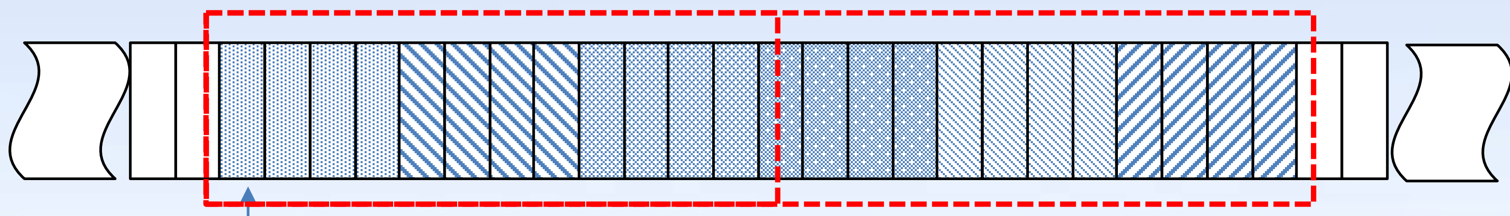
4、观察*(&g)，这个表达式是lvalue，定位这24个字节内存（对象类型int[2][3]）

6、*(&g)是(*(&g))[1]子表达式，没有跟&、sizeof、typeof结合， *(&g)取值: < 0x0082FB30, int(*)[3]>

后续过程就跟之前g[1][2]一样了

这个表达式有几个lvalue？　　g、*(&g)、(*(&g))[1]、(*(&g))[1][2]

(&(*g))[1][2] = 3的操作过程



0x0082FB30

1、识别基础表达式，包括g、1、2 、3

2、g是lvalue，定位这24个字节内存（对象类型int[2][3]）

3、g是*g的子表达式，没有跟&、sizeof、typeof结合，g取值: <0x0082FB30, int(*)[3]>

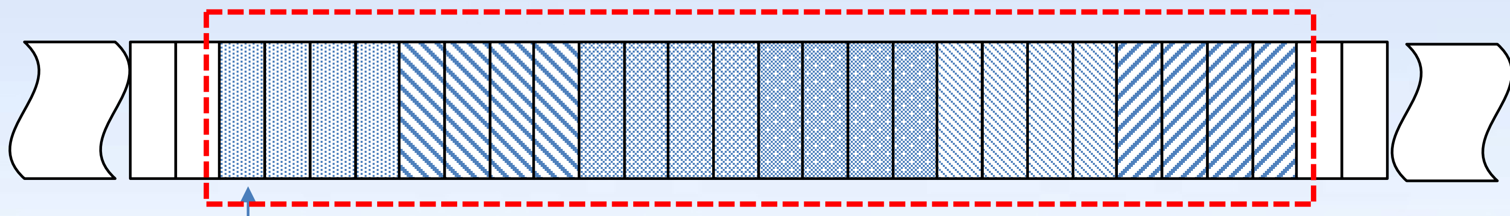
4、*g这个表达式是lvalue，定位这12个字节内存（对象类型int[3]）

6、*g是&(*g)子表达式，&(*g)取值: < 0x0082FB30, int(*)[3]>

后续过程就跟之前g[1][2]一样了

这个表达式有几个lvalue？ g、*g、 (&(*g))[1]、 (&(*g))[1][2]都是lvalue

(g+1-1)[1][2] = 4的操作过程



0x0082FB30

1、识别基础表达式，包括g、1、1（g+1-1中的两个1）、1、2、4

2、g是lvalue，定位这24个字节内存（对象类型int[2][3]）

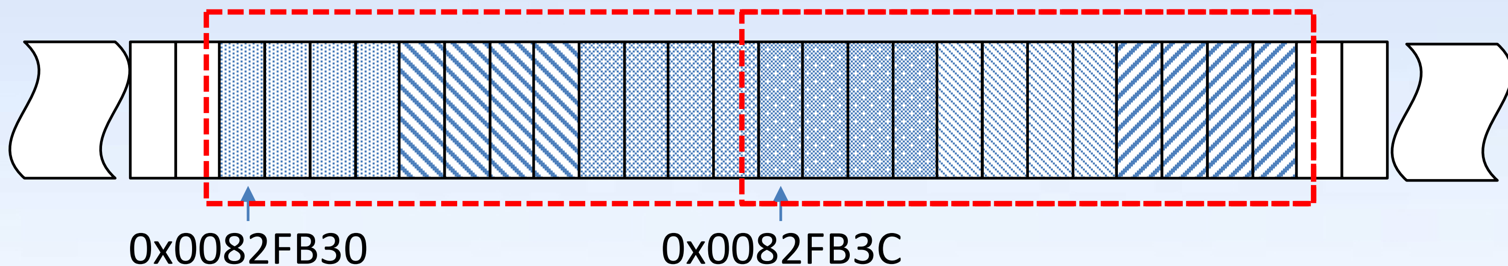
3、g是g+1-1的子表达式，没有跟&、sizeof、typeof结合，g取值: <0x0082FB30, int(*)[3]>

4、计算a+1-1表达式的值，结果是<0x0082FB30, int(*)[3]>

后续过程就跟之前g[1][2]一样了

这个表达式有几个lvalue？　　g、(g+1-1)[1]、 (g+1-1)[1][2]都是lvalue

1[g][2] = 5的操作过程



1、识别基础表达式，包括1、g、2、5

2、1是常量表达式，取值<1, int>

3、1[g]等价于*(1+g)，g是lvalue，定位这24个字节内存（对象类型int[2][3]）

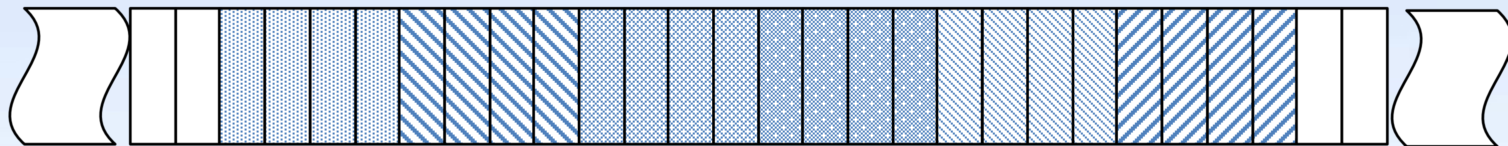
4、计算1+g，返回值：<0x0082FB3C, int(*)[3]>

5、观察*(1+g)，这个表达式是lvalue，定位这12个字节内存（对象类型int[3]）

后续过程就跟之前g[1][2]一样了

这个表达式有几个lvalue？ g、1[g]、1[g][2]都是lvalue

1[2][g] = 6的操作过程



0x0082FB30

1、识别基础表达式，包括1、2、g、6

2、1、2是常量表达式，取值<1, int>，<2, int>

3、1[2]等价于*(1+2)

4、1+2的返回值是<3, int>，不是一个有效指针类型，无法跟*结合

报错

思考一下：数组名是什么

数组名是一个Identifier，是一个对象标识符，是一个lvalue表达式

数组名不是一个指针，也没有特殊性，背后的机制就是表达式的evaluate

int g[2][3]; g++出错是因为g是无法修改的lvalue，而不是因为g是常量

数组名是lvalue，但不可以放到等号左边

思考题

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;

假设对象**n**对应的内存首地址为Addr
返回值表示为**<Value, Value_Type>**形式

<u>表达式的值形式为**Addr+Offset**的形式</u>

&n: <Addr, int(*)[2][3][4][5]>
&n+1: <Addr+480, int(*)[2][3][4][5]>

n:<Addr, int(*)[3][4][5]>
n+1: <Addr+240, int(*)[3][4][5]>

n[0]: <Addr, int(*)[4][5]>
n[0]+1: <Addr+80, int(*)[4][5]>

n[0][0]: <Addr, int(*)[5]>
n[0][0]+1: <Addr+20, int(*)[5]>

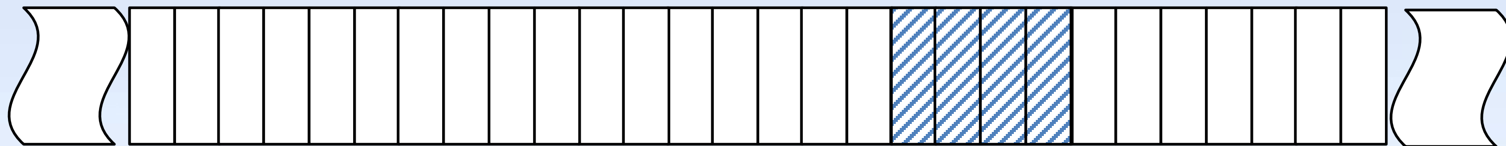
n[0][0][0]: <Addr, int*>
n[0][0][0]: <Addr+4, int*>

n[0][0][0][0]: <0, int>
n[0][0][0][0]: <1, int>

了解malloc

1. 如何确定malloc分配内存后返回的指针类型
2. malloc(sizeof(int)) vs. malloc(sizeof(int)*1)
3. malloc(sizeof(char)*8) vs. malloc(sizeof(char[4])*2)
4. 了解利用实际应用中malloc分配高维数组的方法

访问malloc分配的内存



malloc(4);

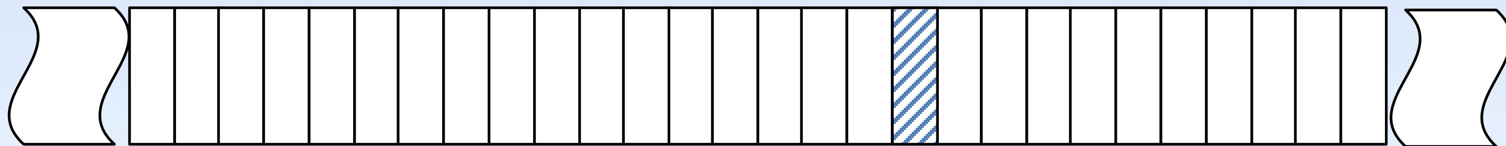
这块对象没有名称

无法通过对象名进行定位使用

本课程假设基础对齐值是16

A: 0x00A231F0
Obj_T: N/A
N: N/A
S: 4
V: N/A
V_T: N/A
Align: fundamental alignment

访问malloc分配的内存



0x00A231F0

void* p = malloc(4);

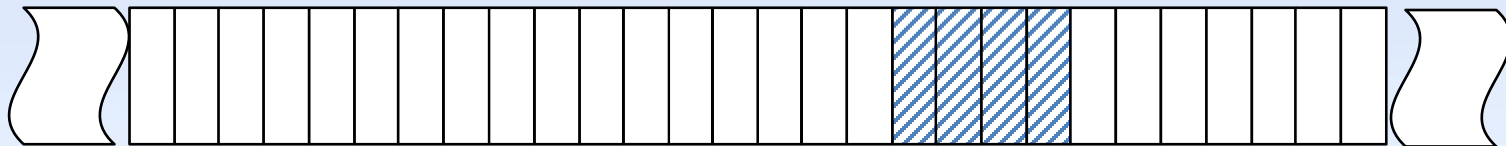
p: <0x00A231F0, void*>

*p定位

A: 0x00A231F0
Obj_T: ?
N: N/A
S: ?
V: ?
V_T: ?
Align: 16

malloc返回的值需要强制转换成一个有意义的对象类型

访问malloc分配的内存



0x00A231F0

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

p: <0x00A231F0, int*>

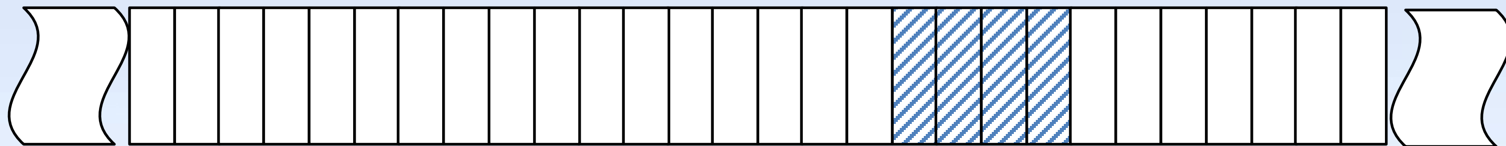
*p定位

*p = 10;

这个Alignment为什么是4？

A: 0x00A231F0
Obj_T: int
N: N/A
S: 4
V: Undefined V: 10
V_T: int
Align: 4

访问malloc分配的内存



0x00A231F0

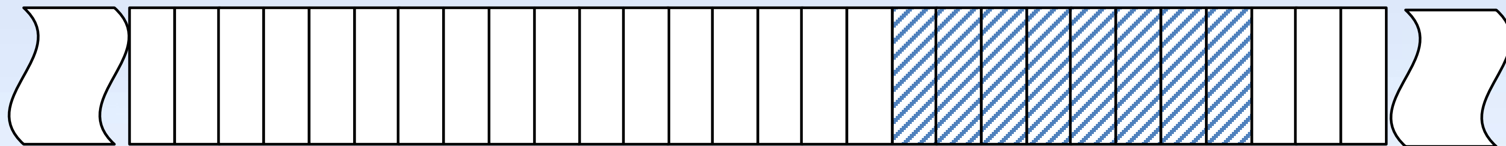
double* p = (double*)malloc(4);

p: < 0x00A231F0, double*>

*p定位

A: 0x00A231F0
Obj_T: double
N: N/A
S: 8
V: ?
V_T: double
Align: 8

访问malloc分配的内存



0x00A231F0

double* p = (double*)malloc(sizeof(double));

p: < 0x00A231F0, double*>

*p定位

A: 0x00A231F0
Obj_T: double
N: N/A
S: 8
V: ?
V_T: double
Align: 8

利用sizeof计算待分配空间大小

理解malloc(sizeof(Obj_T)*N)

malloc(sizeof(int)*10)如何理解

```
int a[10];
int* p = a;
```

对象a对应10个连续int组成的内存块，a这个表达式rvalue类型是int*

malloc(sizeof(int)*10)申请10个连续int空间

语义可视为申请一个int[10]的对象，int[10]类型->返回值类型为int*

```
int* p = (int*)malloc(sizeof(int)*10);
```

malloc的形式化定义

假设一个对象类型Obj_T，malloc申请N个Obj_T大小的内存可形式化定义为

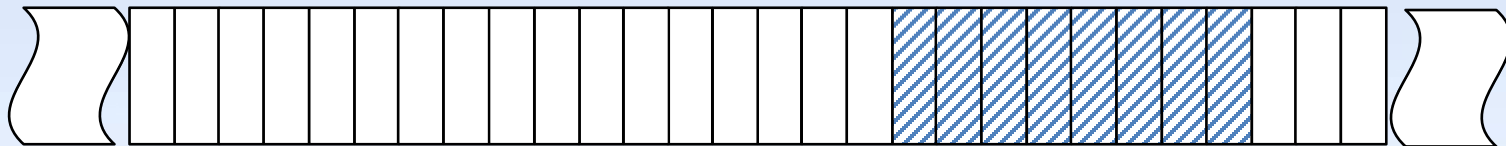
```
Obj_T* p = (Obj_T*)malloc(sizeof(Obj_T)*N)
```

这是最常见的分配一维数组的方法

可视为分配了一个Obj_T[N]对象类型空间

```
int* p = (int*)malloc(sizeof(int)*10)
```

访问malloc分配的内存（续）



0x00A231F0

char* p = (char*)malloc(sizeof(char)*8);

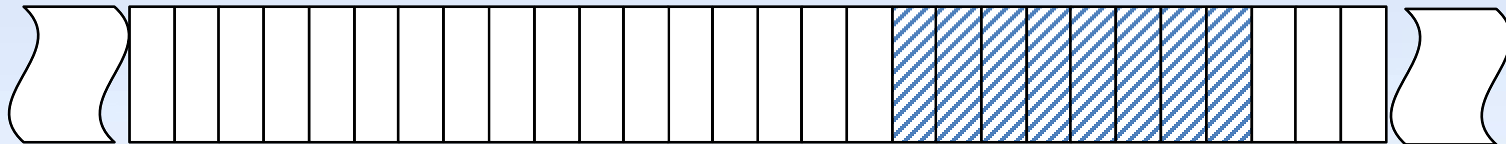
p: < 0x00A231F0, char*>

*定位

A: 0x00A231F0
Obj_T: char
N: N/A
S: 1
V: ?
V_T: char
Align: 1

可视为分配一个char[8]类型对象的内存

访问malloc分配的内存（续）



0x00A231F0

char (*p)[4] = (char(*)[4])malloc(sizeof(char[4])*2);

p: < 0x00A231F0, char(*)[4]>

*定位

A: 0x00A231F0
Obj_T: char[4]
N: N/A
S: 4
V: 0x00A231F0
V_T: char*
Align: 1

可视为分配一个char[2][4]类型的空间

同样申请8个字节，区分指针类型的差异

char* vs. char(*)[4]

利用malloc直接分配高维数组的缺点

```
int (*p)[3][4] =(int(*)[3][4])malloc(sieof(int[3][4])*2);
```

指针类型为int(*)[3][4]，数字3和4需要写死在程序中，工程扩展性较差

malloc更常用于分配一维数组

更常见malloc分配二维数组

```
int** pp = (int**)malloc(sizeof(int*)*2);

for(int i=0; i<2; i++) {
    pp[i] = (int*)malloc(sizeof(int)*3);
}
```

pp[i][j]保持二维数组形式

sizeof(int) vs. sizeof(int)*1

思考：

```
int* p = (int*)malloc(sizeof(int))
```

vs.

```
int* p = (int*)malloc(sizeof(int)*1)
```

malloc(sizeof(int))隐含的语义是分配了一个int[1]类型的空间

malloc分配的内存需要释放

1、通过对象声明分配出来的内存不需要释放

2、通过malloc分配出来的内存需要用free进行释放

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

执行了多少次malloc，就需要执行多少次free。Memory Leak是一个恒久的挑战！！！

free的时候为什么不需要指定大小？

思考题

一、利用malloc分配240个字节，如何定义指针对象p，让该240字节类型视为

1、char[240]

2、int[6][10]

3、int[3][4][5]

二、void* p = malloc(32);

1、int* q = (int*)p;

2、char* r = (char*)p;

3、int (*s)[4] = (int(*)[4])p;

4、char (*t)[2][4] = (char(*)[2][4])p;

请给出q+1, r+1, s+1, t+1的值，假设P的值是Add，形如Add+Offset

思考题

答案

一、

1、char* p = (char*)malloc(sizeof(char)*240);

2、int (*p)[10] = (int(*)[10])malloc(sizeof(int[10])*6);

3、int (*p)[4][5] = (int(*)[4][5])malloc(sizeof(int[4][5])*3);

二、

1、Add+4；2、Add+1；3、Add+16；4、Add+8