对象的相关操作

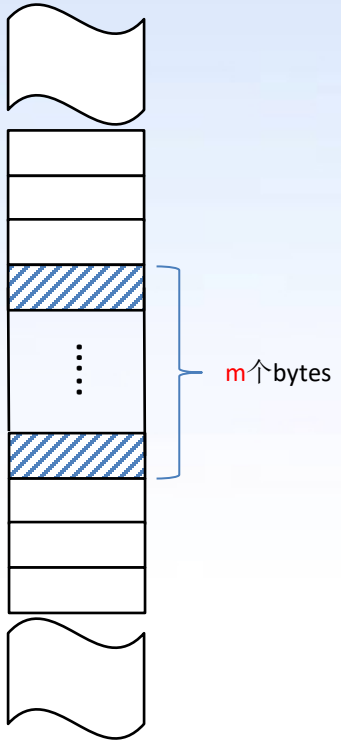
（一）分配对象

1、对象声明

2、malloc等函数

3、String Literal

4、Compound Literal



（二）定位对象

（三）读写对象的相关信息

（四）释放对象

本课程假设

32位机，且1 byte = 8 bit

1 char = 1 byte	char对齐 = 1
1 short = 2 byte	short对齐 = 2
1 int = 4 byte	int对齐 = 4
1 float = 4 byte	float对齐 = 4
1 double = 8 byte	double对齐 = 8
pointer type = 4 byte	pointer type对齐 = 4

sizeof(type-name), _Alignof(type-name)

声明对象时的内存分配

对于这样一个对象声明语句：int a;

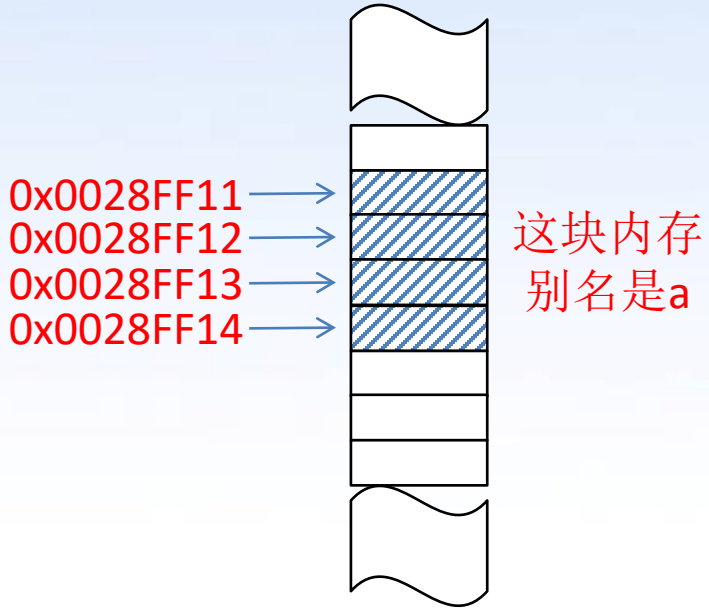
1、分配4个连续byte（为什么是4？）

2、这4个byte组成的块的别名为a

思考：0x0028FF11~0x0028FF14怎么来的？

0x0028FF11这个地址有什么问题呢？

对齐的知识点后面会说明



如何描述对象a对应的内存

int a；

1、这块连续内存第一个byte编号是0x0028FF11

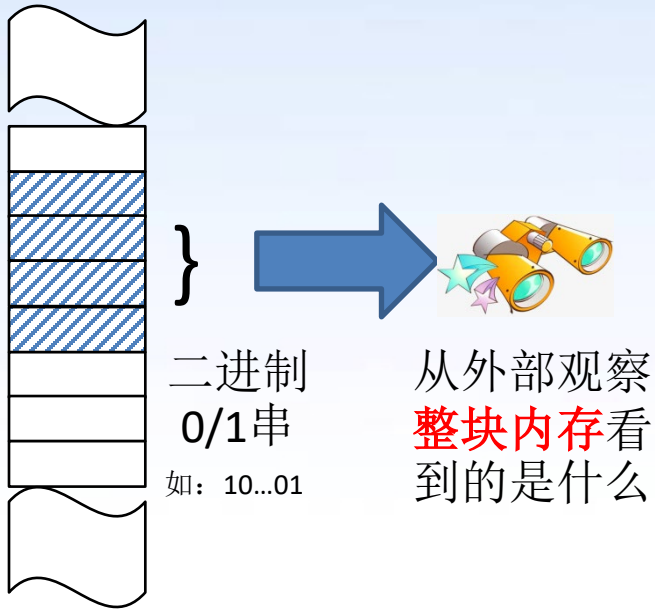
2、这块连续内存的对象类型是int

3、这块连续内存的别名是a

4、这块连续内存大小是4个字节（sizeof(int)）

5、这块连续内存的值

6、这块连续内存首地址的对齐要求是4



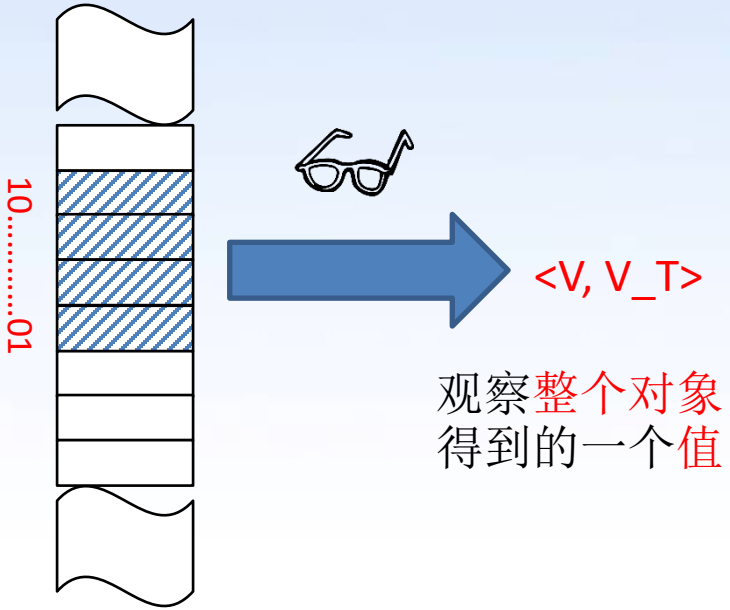
对象的值（Value）是什么？

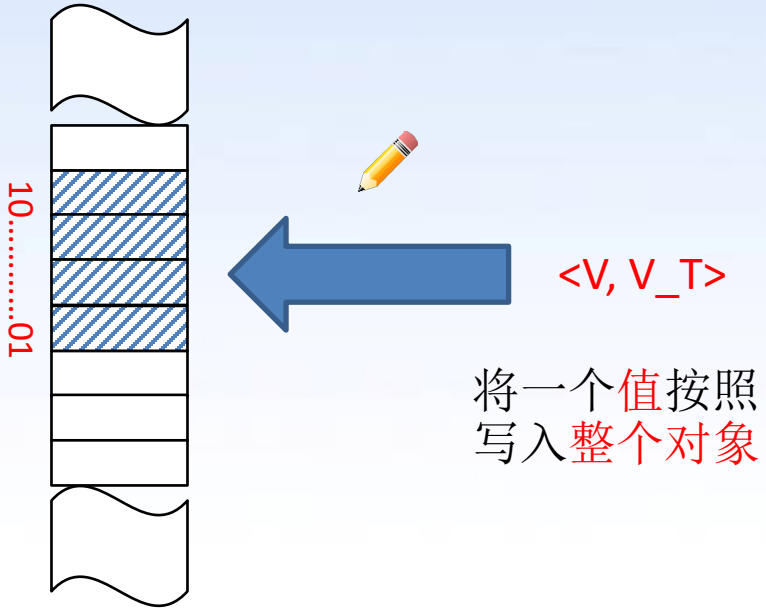
在C语言中，值包含了两个层面的语义：1）值；2）值的类型

Value (V); Value_Type (V_T)记为<V, V_T>

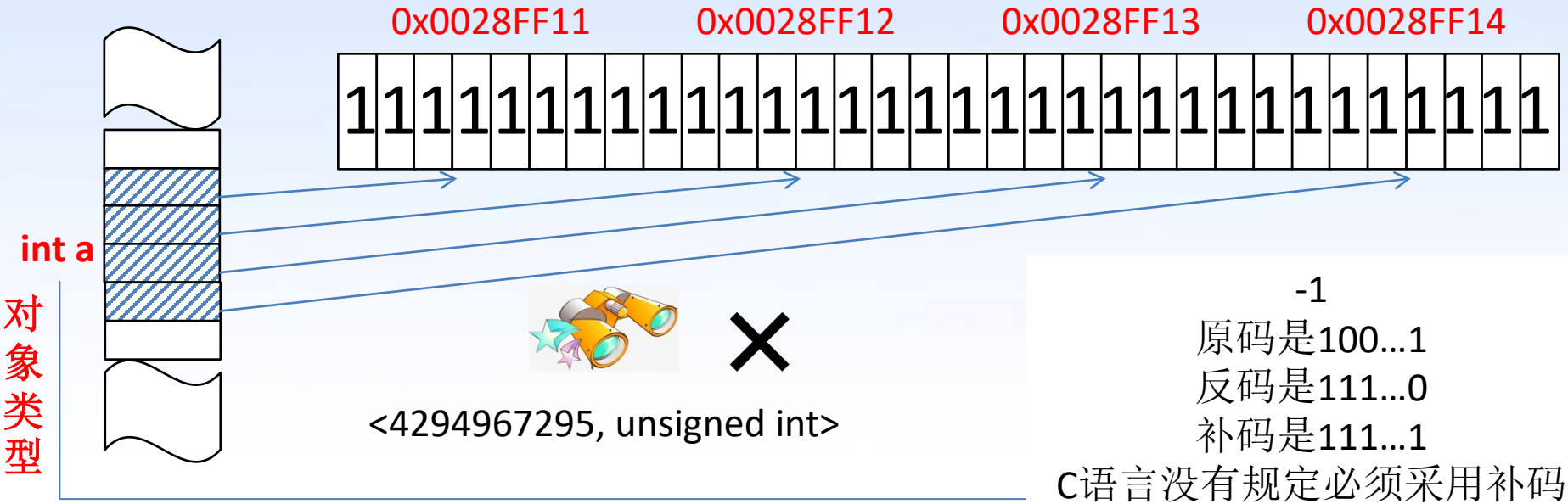
对象a所对应的那块内存也有值，也记为<Value, Value_Type>

进一步理解对象的值





进一步理解对象的表示值



-1
原码是100…1
反码是111…0
补码是111…1
C语言没有规定必须采用补码

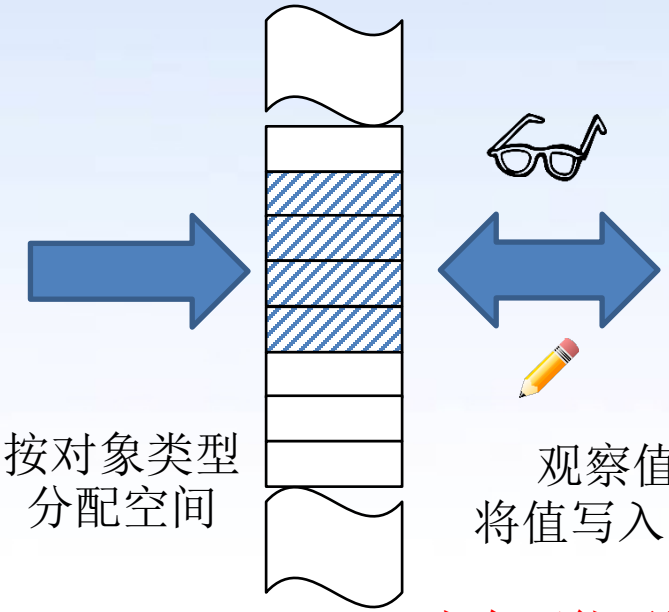
值类型

值类型和对象类型之间的逻辑关系是怎样的？

内存对应的对象类型 vs. 值类型

1、内存的对象类型是Physical View，是在内存分配时按什么类型去申请内存

2、内存的表示值类型是Logical View，是这块内存从外部观察能看到的值的类型



也有可能无法写入

复习一下：对象值类型与对象类型关系

假设对象值记为<V, V_T>，对应的对象类型记为Obj_T

1、如果Obj_T是非数组类型

V_T= Obj_T，V则是通过Obj_T去观察这段内存获得的值

例如：int a， Obj_T是int， V_T也是int

2、如果Obj_T是数组类型

V_T是该数组类型中元素对象类型对应的指针类型，V是数组第一个元素所处内存的第一个字节编号

例如：int a[10]， Obj_T是int[10]，元素类型是int， V_T是int*

思考题

1、char c; double d; float f[3][4]; int* p[3];
这四个对象对应的值类型是什么？

答案

1、char c; 对象类型Object_Type为char，非数组类型，
Value_Type=char

2、double d; 对象类型Object_Type为double，非数组类型，
Value_Type=double

3、float f[3][4]; 对象类型Object_Type为float[3][4]，数组类型，元素类型为float[4]，
Value_Type=float(*)[4]

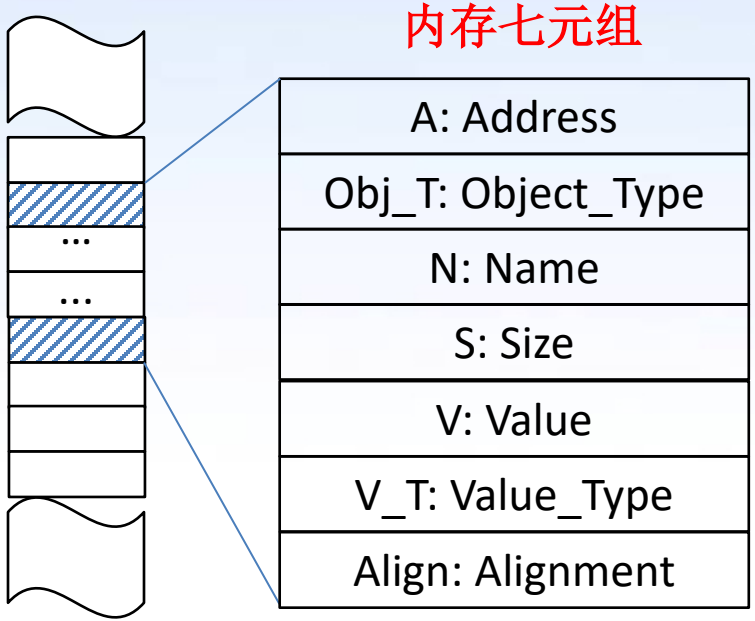
4、int* p[3]; 对象类型Object_Type为int*[3]，数组类型，元素类型为int*，
Value_Type=int**

对象七元组模型

Object = {Address, Object_Type, Name, Size, Value, Value_Type, Alignment}

Object：声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
- Object_Type: 对象类型
- Name：对象名称
- Size：内存大小（字节数量）
- Value：对象的值
- Value_Type：值的类型
- Alignment：对齐要求



内存七元组取值规则

M = {Address, Object_Type, Name, Size, Value, Value_Type, Alignment}

1、Address由系统分配，一旦确定无法修改

2、Object_Type和Name是对象类型和对象名称（对象可能匿名）

3、Size是这块内存的大小（字节数）

4、Value和Value_Type的取值根据Object_Type来确定

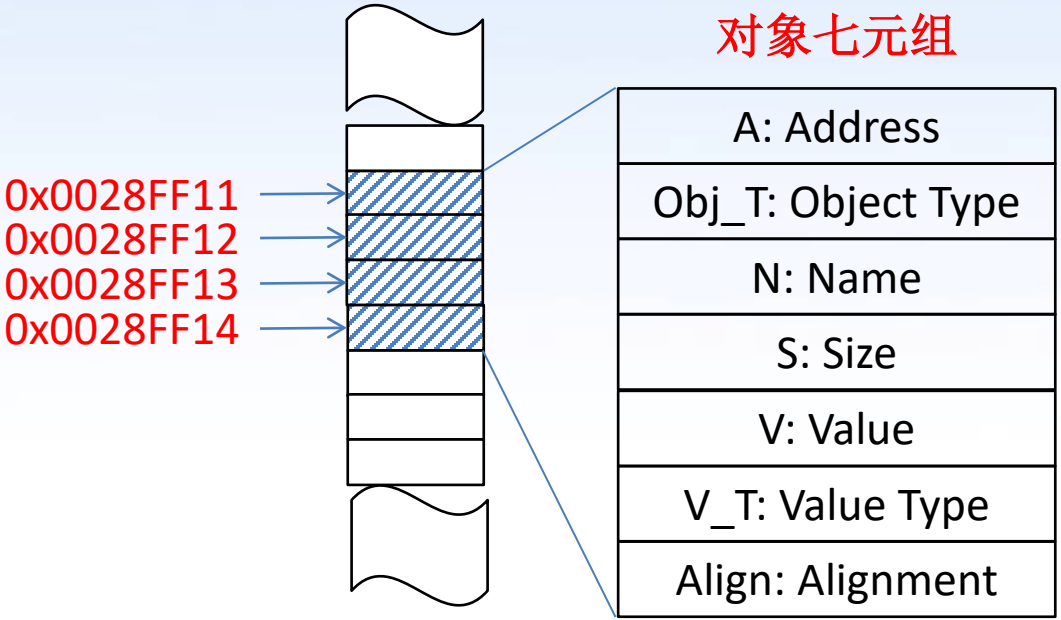
Object_Type是非数组对象类型 vs. 数组对象类型

5、Alignment由对象类型决定，也可以由程序员更改

内存七元组是为了让大家更好的了解分配的内存而提出的一个逻辑模型

通过int a声明出来的对象如何描述

int a;



Address: 0x0028FF11
Object Type: int
Name: a
Size: 4
Value: ？ (Undefined)
Value Type: int
Alignment: 4

Object_Type是非数组对象类型
Value_Type = Object_Type

思考：这段内存有Value吗？

其他对象类型声明和赋值示例

```
float b = 1.0;
double c = 2.0;
char d = 'a';

int* p;

int e[2];
char f[8];

int g[2][3];

struct m_struct {
    int a;
    float b;
}h;

union m_union {
    int a;
    float b;
}i;
```

基本数据类型，例如：float, double, char

指针类型，例如：int*

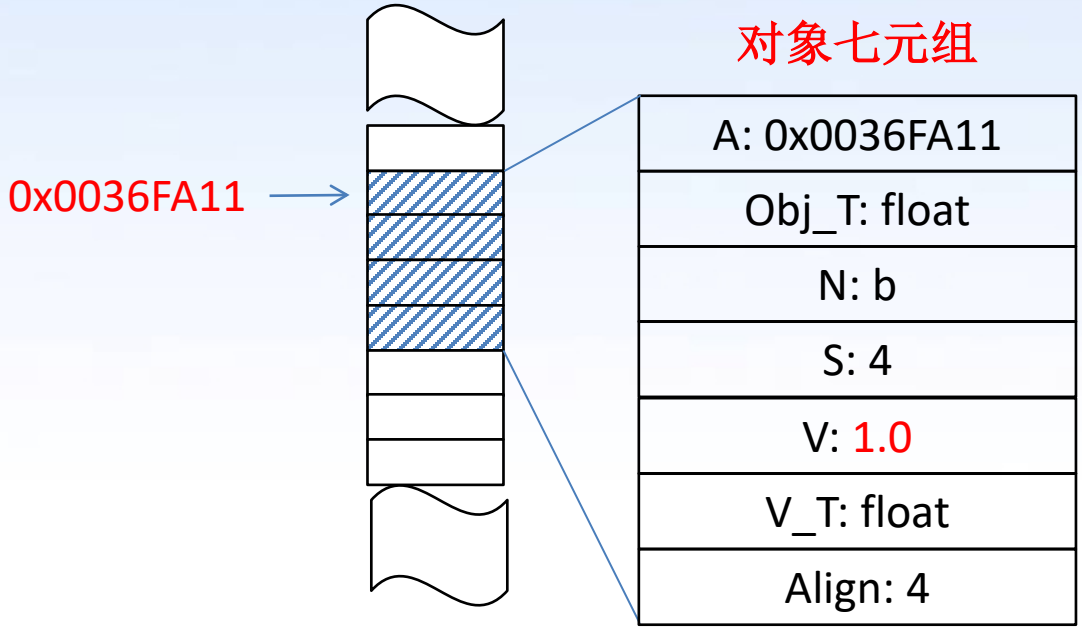
一维数组，例如：int[2], char[8]

“二维数组”，例如：int[2][3] （C语言数组都是一维的）

结构体/联合体，例如：struct m_struct/union m_union

观察：float b=1.0;

这是变量初始化，分配对象的同时对对象赋值

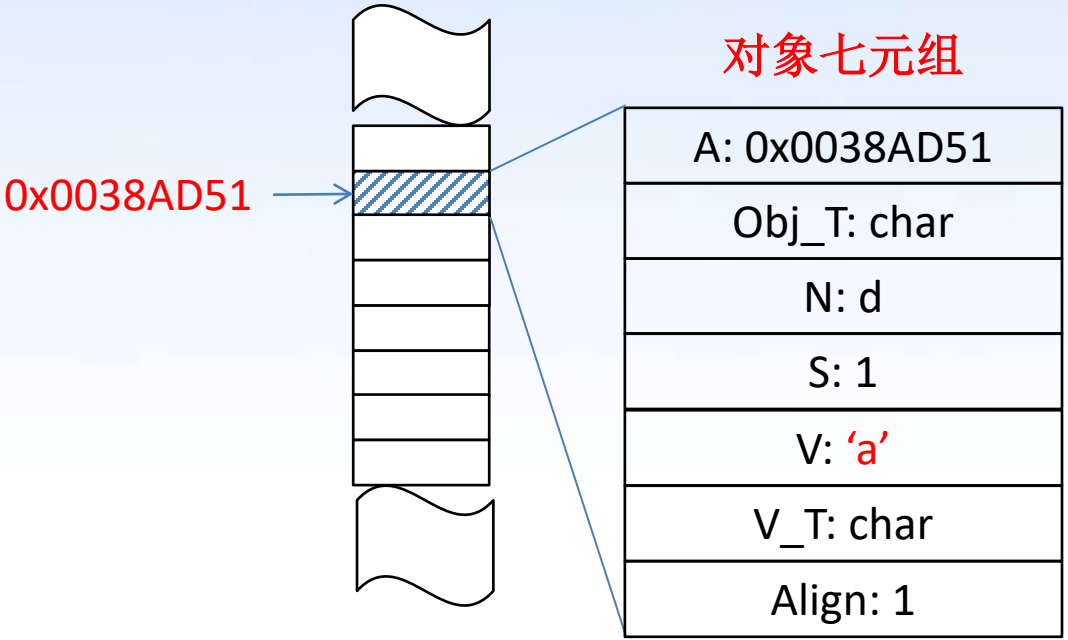


将‘1.0’按float类型映射成0/1值

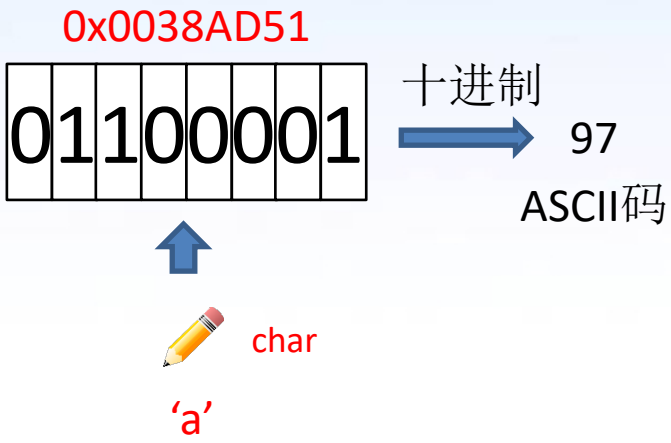
32位0/1串是实际存储的值
1.0是用float观察出来的值

观察：char d='a';

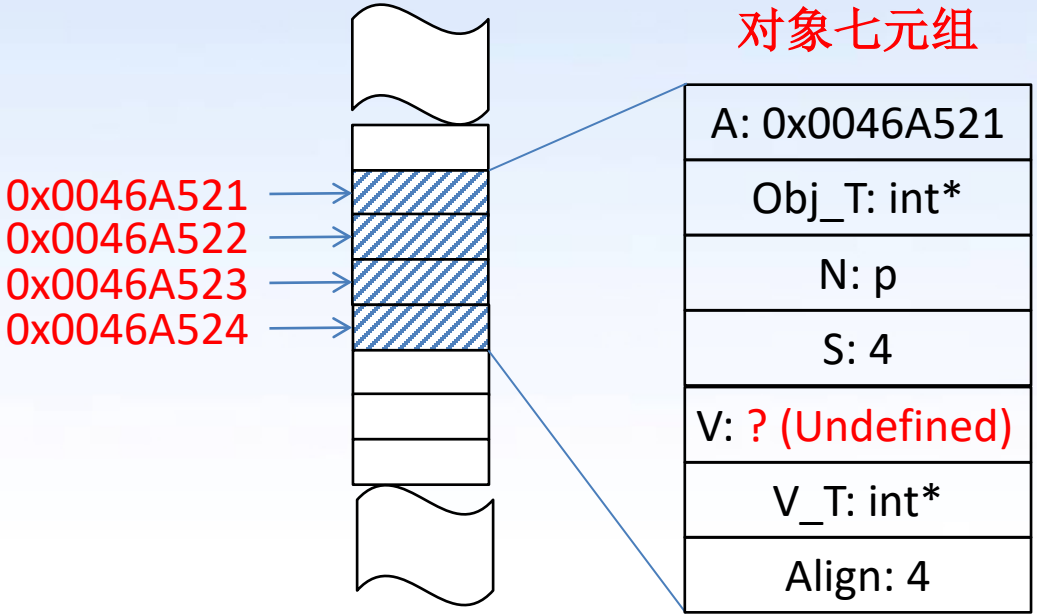
这是变量初始化，分配内存的同时对内存赋值



将'a'按char类型映射成0/1值



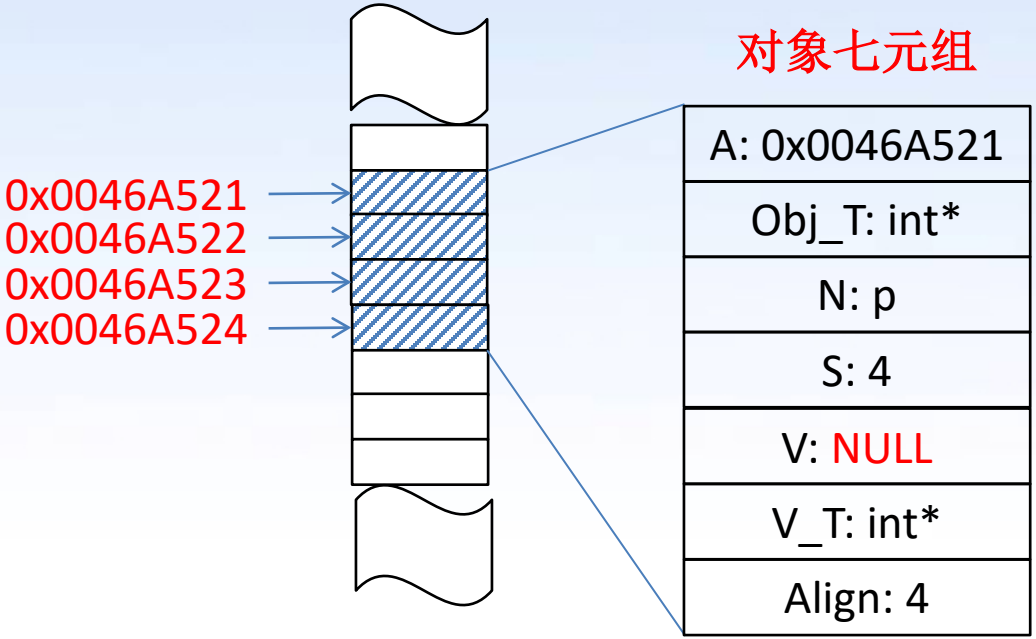
观察：int* p;



思考1：为什么size是4

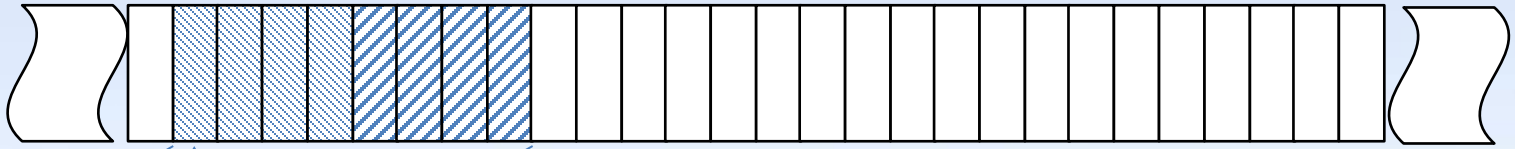
思考2：这个时候value有值吗？

改进一下：int* p=NULL;



变量合理的初始化是
良好的防错性编程习惯

观察：int e[2];



0x0076AB11

A: 0x0076AB11
Obj_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*
Align: 4

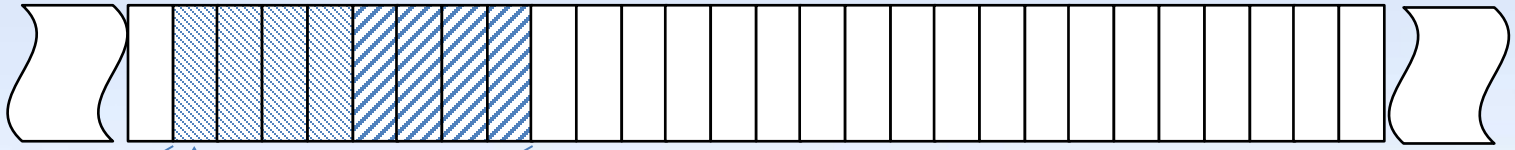
1、数组类型：Object_Type是int[2]

2、数组元素类型是int，其对应的指针类型是int*
Value_Type = int*

3、第一个元素对象类型是int，所占4个字节编号
[0x0076AB11, 0x0076AB14]

4、数组对象的对齐要求是元素类型的对齐要求

观察：int e[2]={1, 2};



0x0076AB11

A: 0x0076AB11
Obj_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*
Align: 4

<1, int> <2, int>

1、数组对应的内存整体赋值只能通过变量初始化

2、数组对象的值不会发生任何变化

提前观察一下：int e[2];e=NULL;

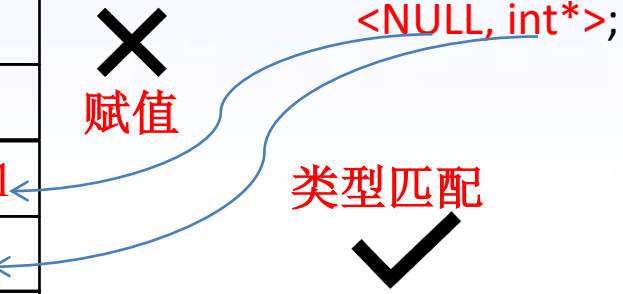
```
int e[2];

e=NULL;
```

```
=== 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)) ===
```

e的对象七元组

A: 0x0076AB11
Obj_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*
Align: 4



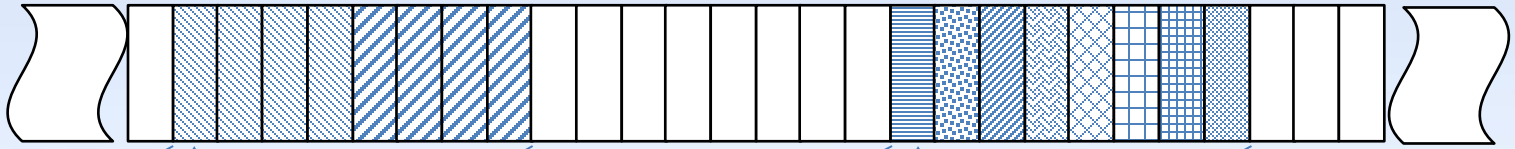
数组对象的Value一定是指向数组第一个元素的地址编号

V的值必须和A相等

e并不是一个常量，它的值不能更改是语法限制的

真实原因：e不是一个modifiable lvalue（后续讲解）

观察：int e[2];和char f[8];



0x0076AB11

A: 0x0076AB11
Obj_T: int[2]
N: e
S: 8
V: 0x0076AB11
V_T: int*
Align: 4

0x0076AB21

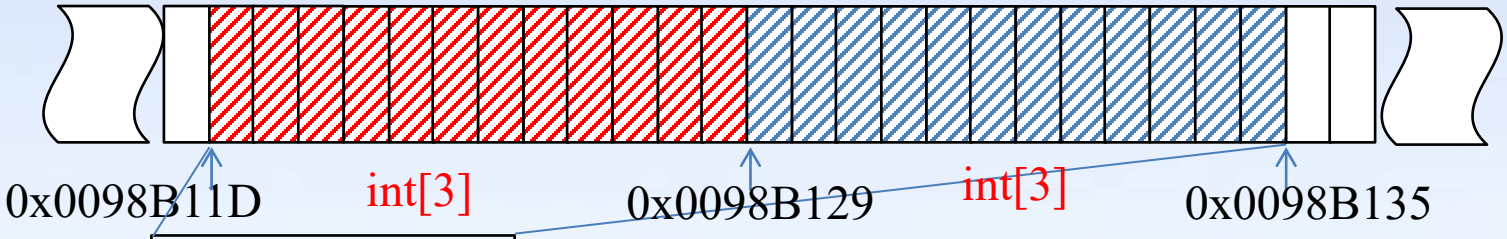
A: 0x0076AB21
Obj_T: char[8]
N: f
S: 8
V: 0x0076AB21
V_T: char*
Align: 1

思考：为什么大小都是8

sizeof(int[2]) = 8
2*sizeof(int)

sizeof(char[8]) = 8
8*sizeof(char)

观察：int g[2][3]；



A: 0x0098B11D
Obj_T: int[2][3]
N: f
S: 24
V: 0x0098B11D
V_T: int(*)[3]
Align: 4

1、对象类型：Object_Type是int[2][3]

2、元素对象类型是int[3]，其对应的指针类型是int(*)[3]
Value_Type = int(*)[3]

3、第一个元素对象类型是int[3]，所占12个字节编号
[0x0098B11D, 0x0098B128]

4、数组对象的对齐要求是元素对象的对齐要求（递归）

思考题

1、声明一个对象int m[2][3][4]，对象m对应这个内存的Object_Type=?

2、这个对象的元素对象类型是什么？

3、对象m的值类型Value_Type是什么？

A: 0x0037DC11
Obj_T: int[2][3][4]
N: m
S: 96
V: 0x0037DC11
V_T: int(*)[3][4]
Align: 4

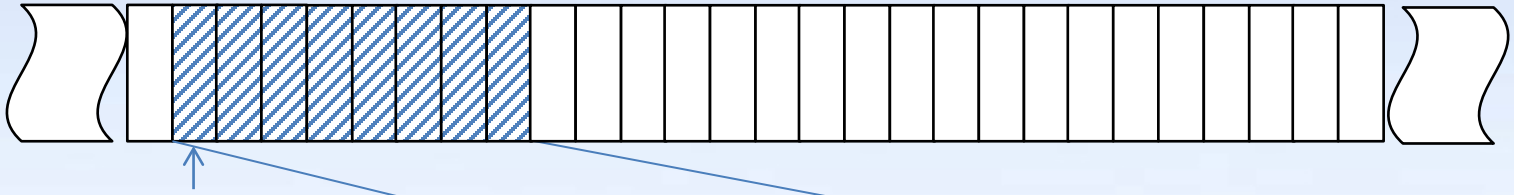
答案：

1、Object_Type=int[2][3][4]

2、该数组的元素对象类型为int[3][4]

3、Value_Type=int(*)[3][4]

观察：结构体变量h



0x0085D611

```
struct m_struct {
    int a;
    float b;
}h;
```

A: 0x0085D611
Obj_T: struct m_struct
N: h
S: 8
V: Undefined
V_T: struct m_struct
Align: 4

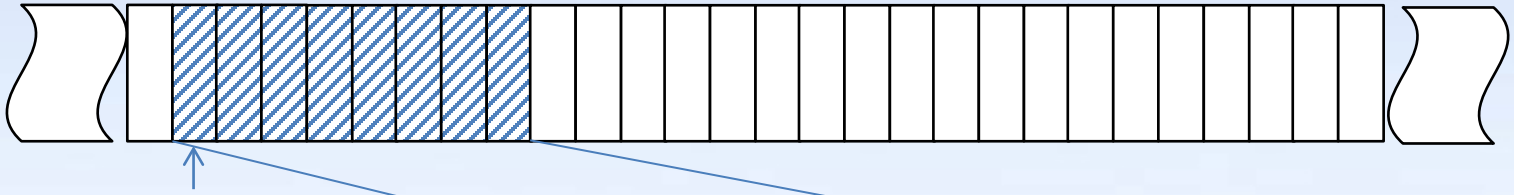
struct m_struct是非数组对象类型

使用sizeof获得结构体变量大小
sizeof(struct m_struct)

思考题：1、Value现在是什么？
2、对齐是多少？

结构体/联合体的对齐后续介绍

观察：结构体变量h



0x0085D611

```
struct m_struct {
    int a;
    float b;
} h = {10, 1.0};
```

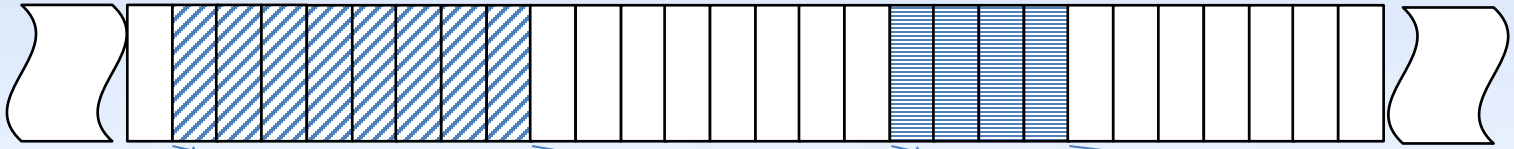
A: 0x0085D611
Obj_T: struct m_struct
N: h
S: 8
V: Defined
V_T: struct m_struct
Align: 4

printf("h= ? \n", h);

对象的值是观察整个内存

对象h有值，我们只是看不懂

观察：对比结构体变量h和联合体变量i



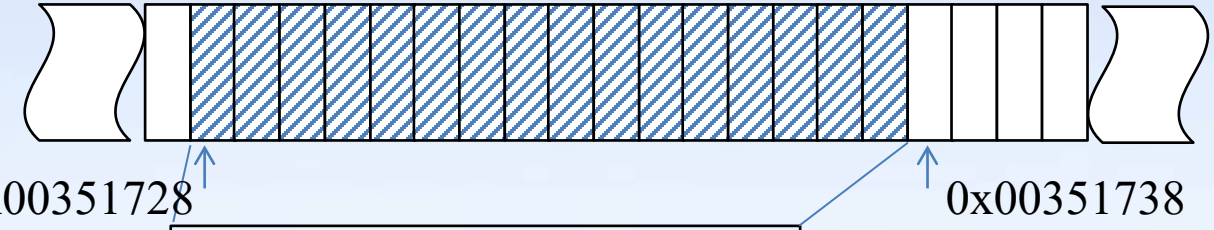

```
struct m_struct {
    int a;
    float b;
}h;

union m_union {
    int a;
    float b;
}i;
```

A: 0x0085D611
Obj_T: struct m_struct
N: h
S: 8
V: Undefined
V_T: struct m_struct
Align: 4

A: 0x0085D621
Obj_T: union m_union
N: i
S: 4
V: Undefined
V_T: union m_union
Align: 4

内存管理函数分配对象：malloc(16)




```
malloc(16);
```

A: 0x00351728
Obj_T: N/A
N: N/A
S: 16
V: N/A
V_T: N/A
Align: fundamental alignment

思考1： 为什么有Address和Size

思考2： 这块内存为什么没有Object_Type和Name

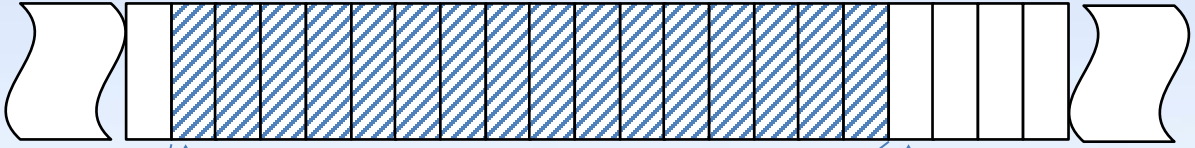
思考3： 为什么这块内存没有Value和Value_Type

思考4： 什么是Fundamental Alignment

alignof(max_align_t)

#include <stddef.h>

aligned_alloc(4, 16)



0x00351728 0x00351738

A: 0x00351728
Obj_T: N/A
N: N/A
S: 16
V: N/A
V_T: N/A
Align: 4

aligned_alloc可以指定对象对齐要求
#include <stdlib.>

风险1：超过Fundamental Alignment系统可能会返回NULL
风险2：小于Fundamental Alignment可能会引起未来指针强制转换问题

后续会详细讲解对齐要求

String Literal vs. String

A character string literal is a sequence of zero or more multibyte characters enclosed in double-quotes

例如：“hello\0world”

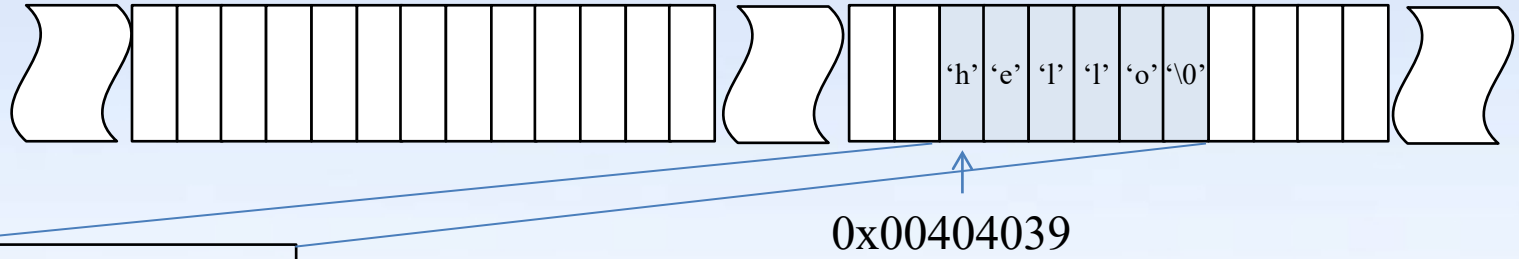
A string is a contiguous sequence of characters terminated by and including the first null character

例如：“hello world”

最后都有一个隐藏的’\0’，双引号内也嵌入’\0’的称之为String Literal

不失一般性，可以统称为字符串

“hello”分配对象



A: 0x00404039
Obj_T: char[6]
N: N/A
S: 6
V: 0x00404039
V_T: char*
Align: 1

”hello”的含义：

在read-only memory找合适的空间，存放'h', 'e', 'l', 'l', 'o', '\0’

该对象为匿名对象

“hello”分配的对象特点

```
int main(int argc,char* argv[])
{
    "hello";
    "hello";
    return 0;
}
```

这两个”hello”分配出来的对象有什么特点？

这两个对象可能复用，也可能不复用

String literals need not designate distinct objects

Compound Literal

(type-name){Initializer-list}

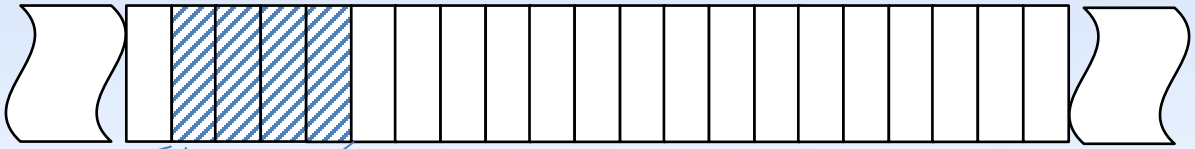
声明定义一个匿名(unnamed)的对象

该对象的对象类型是type-name

该对象由Initializer-list进行初始化

Provides an unnamed object whose value is given by the initializer list.

(int){1}



0x0061FE0C

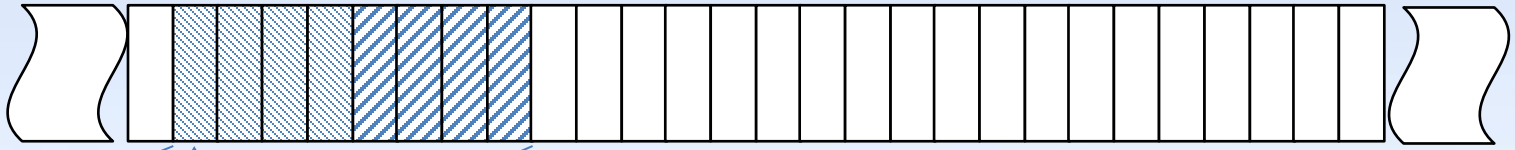
A: 0x0061FE0C
Obj_T: int
N: N/A
S: 4
V: 1
V_T: int
Align: 4

1、对象类型是int

2、初始化的值为1

3、匿名对象

(int[2]){1, 2}



0x0061FE18

A: 0x0061FE18
Obj_T: int[2]
N: N/A
S: 8
V: 0x0061FE18
V_T: int*
Align: 4

<1, int> <2, int>

1、对象类型是int[2]

2、2个int元素的值分别是1，2

(int[]){1,2}

数组如果不指定长度

就是初始化列表元素的个数

对象的相关操作

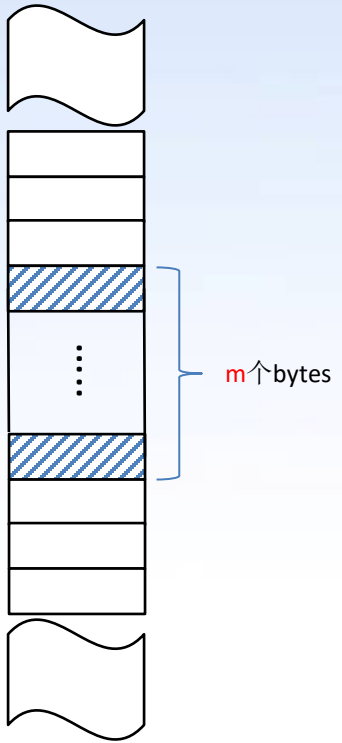
（一）分配对象

1、对象声明

2、malloc等函数

3、String Literal

4、Compound Literal



（二）定位对象

（三）读写对象的相关信息

（四）释放对象

通过lvalue（左值）定位（designate）对象

C语言中，如果要定位一个对象，只能通过lvalue

什么是lvalue呢？

左值（ lvalue）的概念和表达式紧密相关

什么是表达式

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）：包括对象标识符和函数标识符

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

多个表达式可以组合成新的表达式

int a;

a+1

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

int* p;

*(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

表达式优先级顺序

优先级依次降低


```c
int main(int argc, char* argv[])
{
    char a = 0b0001;
    char b = 0b1010;
    char c = 0b1011;
    char d = 0b0111;

    char e;
    char f;
    char g;

    e = a | b ^ c & d;
    f = a | (b ^ c) & d;
    g = (a | b) ^ c & d;

    printf("%d %d %d\n", e, f, g);

    return 0;
}
```

```
9 1 8

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

思考题

在上例中的 g = (a | b) ^ c & d;
一共有多少个子表达式？

1、g, a, b, c, d
2、a | b
3、(a | b)
4、c & d
5、(a | b) ^ c & d
6、g = (a | b) ^ c & d;

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、对象标识符（identifier)

2、字符串（String Literal）

3、*exp

4、exp1[exp2]

5、(type-name){initializer}

6、exp.identifier/exp->identifier

1、int **a**;

2、**”hello”**

3、给定int* p, ***p, *(p+1)**

4、给定int e[2], **e[1], 1[e]**

5、**(int[3]){1,2,3}**

6、struct m_struct **h, *p;**

h.name, p->name

对象的相关操作

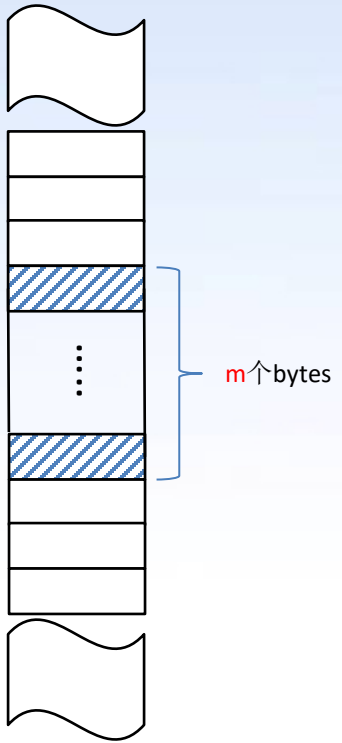
（一）分配对象

1、对象声明

2、malloc等函数

3、String Literal

4、Compound Literal



（二）定位对象

（三）Access对象

（四）释放对象

Access对象的含义

Access意味着对这个对象的读和写

首先需要了解对表达式的evaluate

Evaluation of Expression

In the abstract machine, all expressions are evaluated as specified by the semantics.

表达式需要根据语法的规定来进行evaluate

Evaluation of Expression

给定一个表达式（Expression)，Evaluation过程包括：

1、Value Computation （求值）

2、Initiation of Side Effect（确定副作用）

Value Computation: 得到这个表达式的rvalue，以及rvalue的类型
rvalue在标准中也被称为表达式的value

Side Effect: 状态的改变（changes in the state of the execution environment）