

编译原理

田玲 教授、博导

lingtian@uestc.edu.cn



数据类型(Data Type)实质上是对存储器中所存储的数据进行的抽象。它包含了一组**值的集合**和一组**操作**。

1. 数据类型的作用

- 实现了数据抽象
- 使程序员从机器的具体特征中解脱出来
- 提高了编程效率

2. 数据类型的分类

- 内部类型 (built-in) : 语言定义的
- 自定义类型(user-defined): 用户定义的

第二节 内部类型

抽象表示“25”被映射成“00011001”，整数加法被映射成机器的定点加。

一. 内部类型的特点

- **内部类型**反映基本硬件特性
- 在语言级,**内部类型**标识共用某些操作的数据对象的抽象表示

整型表示能实现+、-、*、/等定点操作的数据对象的集合

二. 内部类型的优越性

1. 基本表示的不可见性

- 基本位串对程序员是不可见的。
- 优点：
 - ✓ 导致不同的程序设计风格
 - ✓ 可写性
 - ✓ 可读性
 - ✓ 可修改性

例：25+9=34

基本表示

00011001+00001001

结果00100010

二. 内部类型的优越性

2. 编译时能检查变量使用的正确性

进行静态类型检查,如非法运算,形实参类型匹配

3. 编译时可以确定无二义的操作

- **超载**(多态)的概念:运算符的意义依赖于操作数的类型。如 “+” 可以表示整数加或实数加
- 编译时,可拒绝混合运算,或提供类型转换指令
- 合理地使用超载,可以提高语言的可读性和可用性

4. 精度控制

可以通过数据类型显式定义数据的精度

第三节 用户定义类型

许多语言除了定义内部类型外，还允许程序员定义**新的数据类型**，规定基本数据对象的聚合，乃至聚合的聚合。

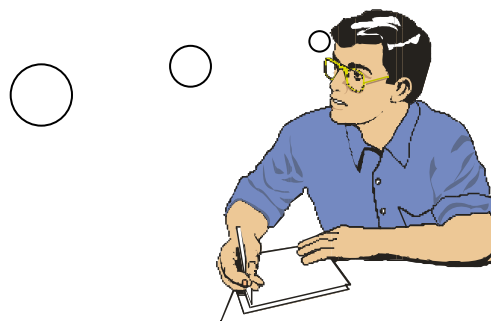


How to define?

1. 笛卡尔积

定义： N 个集合 A_1, A_2, \dots, A_n 的**笛卡尔积**表示为 $A_1 \times A_2 \times \dots \times A_n$ ，它是一个集合，其元素为 (a_1, a_2, \dots, a_n) ，其中 $a_i \in A_i$

在语言中
对应什么
构造？



1. 笛卡尔积

例：对于如前定义的多边形，
有两个域no-of-edges和edge-
size，若t1是一个边长为7.53的
等边三角形，则可以写为：

```
t1.no-of-edges = 3;  
t1.edge-size = 3.75;
```

在COBOL和PASCAL中
称为**记录**；在ALGOL中
称为**结构**。

为

integer × real

每边边长

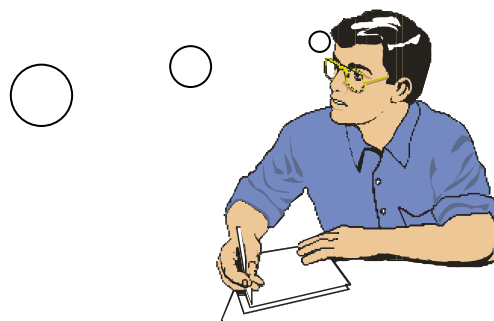


注意：语言把笛卡尔积数据对象看成由若干个域组成，
每个域有一个唯一的名字；通常用域名来选取域，对它进行
修改；

2. 有限映像（射）

定义：从定义域类型DT（domain type）的值的有限集合，到值域类型RT（range type）的值的有限集合的函数称为**有限映像（射）**。

在语言中
对应什么
构造？



有限映像（射）的一些特点

□在高级语言中通常体现为**数组构造**；

□值域对象通过下标选取。

□下标越界会出错，动态检

□下标可用来选取值域的多

□SNOBOL4的ARRAY构造是同一类型的

□DT到相应值的特定子集的绑定策略：

- ✓ 编译时绑定（静态数组）
- ✓ 对象建立时绑定（执行到分程序时，动态数组）
- ✓ 对象处理时绑定（对APL，子集范围可变）

例如：PASCAL中的数组说明：
var a: array[1..50] of char
可看成是从1到50的整数到字符集的有限映像

3.序列

定义：序列由任意多个数据项组成,这些数据项称为该序列的成分,且类型相同（记为CT）。

例：串是从所周知的序列，其成分类型为字符；顺序文件的思路也来自序列的概念。

串的一般操作有4种：

- 连接
- 首项选取
- 尾项选取
- 子串

4.递归

定义：若数据类型T包含属于同一类型T的成份，那么类型T称为**递归类型**。

递归类型的特点：

- 1) 允许在类型定义中使用被定义类型的名字
- 2) 指针是建立递归数据对象的重要手段

单链表 $f(f())$ 、二叉树 $f(fl(), fr())$ 、树 $f(f(), \dots, f())$

5.判定或 (discriminated union)

定义：判定或是一个选择对象结构的构造机制，规定在两个不同选择对象之间作出适当的选择；每一选择对象结构称为变体(Variant)。

例如：PASCAL和ADA中的变体记录；C和ALGOL68中的联合。

COBOL中的一个记录说明：

01 EMPLOYEE_RECORD

05 NAME

05 SALARY

05 HOUR_RATE REDEFINES SALARY

6.幂集

定义： 类型T的元素所有子集的集合，称为**幂集**，记为 $\text{Powerset}(T)$, T称为基类型。

幂集类型的操作：

□ 由于具有该类型的变量的值是一个子集，因此它们的基本操作是集合的操作，比如，**联合**，**与**，以及测试某个元素是否在一个集合中等。

7.小结

- ❑ 程序语言允许程序员以上六种机制来定义复杂的数据对象（新的类型）；
- ❑ 新的类型可以通过非显式的方式说明；
- ❑ 也可通过显式的方式说明
- ❑ 显示定义有如下特点：

- ✓ 可
- ✓ 可
- ✓ 可
- ✓ 一致

例如：

```
var a: real;
```

```
char
```

例如：

```
type complex = record radius:
```

```
real;
```

```
angle: real;
```

```
end
```

```
var c1, c2, c3: complex;
```


六种数据类型聚合方式

笛卡尔积

记录、结构

有限映射

数组

序 列

字符串、顺序文件

递 归

树

判定或

联合、变体记录

幂 集

集合

第四节 PASCAL语言数据类型结构

1. 非值类型

内部类型

integer, real, boolean

有序类型

每一元素都有顺序

如: 整型, 布尔型

定义新的有序类型

✓ 枚举型 其元素有限

✓ 子界型 动态范围

例:

• 引入新的数据类型 `work_day` ;

`type`

`day = (`

`Sunday, monday, tuesday, wednesday, thursday, friday, saturday); // 枚举`

枚举

`type`

`work_day`

`var`

`class`

`day`

后继

- 引入新的数据类型 `day` ;
- 定义了 `day` 由 `Sunday` 等 7 个元素组成;
- 定义了元素之间的顺序 `<` ;
- 隐含对这个新类型变量可进行赋值和比较操作;

• 求 `class_day` 的后继 ;

2.聚合构造

下标（定义域）的类型；

1)数组构造

构造符ARRAY允许程序员定义有限映像；数组构造的一般形式为：

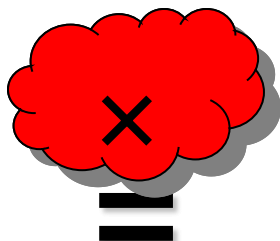
array[t1] of [t2]

元素（值域）的类型；



注意： PASCAL把下标类型不同的数组看成不同的类型

type
a1=array[1..50] of
integer;



type
a2=array[1..70] of
integer;

例:

```
procedure sort(var a:array[low..high:integer] of
ctype);
var i:integer;
    more:boolean;
begin {sort}
.....
end {sort}
```

解决办法:

引入**符合数组 (Conformant Array)**概念--维数相同, 成分类型相同的数组; 符合数组可以形、实参数匹配。



最后一点：PASCAL可以定义多维数组。

```
type row=array[-5..10] of integer;  
var my_matrix:array[3..30] of row;  
或  
var      my_matrix:array[3..30,-5..10]      of  
      integer;
```

定义了一个二
维整型数组

2)记录构造

构造符**RECORD**用以定义笛卡尔积；一般形式为：

```
Record field_1:type_1;  
      field_2:type_2;  
      ...  
      field_n:type_n;  
end
```

假设t, p是前面定义的多边形,
t.no_of_edges:=3;
t.edge_size:=7.53;
p:=t;

记录可以整体访问，也可用圆点“.”作为选择符访问单个的域；

变体记录item

PASCAL的变体记录:

记录类型允许有可变部分, 支持

标识符域available, 记录的判定成分

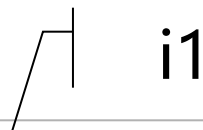
```
type  
  m  
  ite  
  .....  
  i1.price:=5.24;  
  i1.available:=true;  
  i1.amount:=29;  
  i1.where:=liquor;  
  i2.price:=324.99;  
  i2.available:=false;  
  i2.month_expect:=8;
```

例如, 如果i1和i2被说明, 则可对它们进行如下操作:



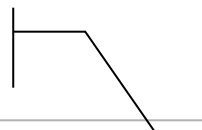
注意: PASCAL允许程序员访问记录结构的所有域, 包括标识符域。

程序的执行结果



A diagram showing a variable `i1` pointing to a table. A line from the label `i1` connects to the top of the table.

price	5.24
available	true
amount	29
where	liquor



A diagram showing a variable `i2` pointing to a table. A line from the label `i2` connects to the top of the table.

price	324.99
available	false
month_expected	8

变体记录的实现

- 改变一个变体记录的标识符，在概念上建立了一个新记录；
- 变体记录在同一块存储区上重叠存放所有变体；
- 变体记录允许程序员根据每个变体的类型，以不同的观点来解释存储在该区域中的位串；

PASCAL变体记录的

```
record price:real  
  case boolean of  
    true:(amount:integer;  
      where:dept);  
    false:(month_expected:month)  
  end
```

使用变体记录不安全

- 同一存储区对应不同的名字和类型，容易出错
- 编制程序依赖于实现
- 标识符域的标识符可缺省，不安全

3)集合构造
PASCAL语言
形式, 基类
表或集合类

下列语句是合法的:

```
leftover:=.....;  
my_salad:=[carrot..onion];  
if not bean in leftover  
    then  
my_salad:=my_salad+leftover;
```

例:

```
type vegetable = (bean,cabbage,carrot,celery,  
lettuce,onion,mushroom,zucchini);  
var my_salad,leftover:set of vegetable;
```

4) 文件构造

- ❑ PASCAL文件是任意类型的诸元素的序列;
- ❑ PASCAL文件仅能顺序处理;
- ❑ 只能进行PUT和GET操作;

例:

```
type pattern=record ...end;  
    tape=file of pattern;  
var t1,t2:tape;
```

Get操作把下一个元素读到缓冲区

3.指针

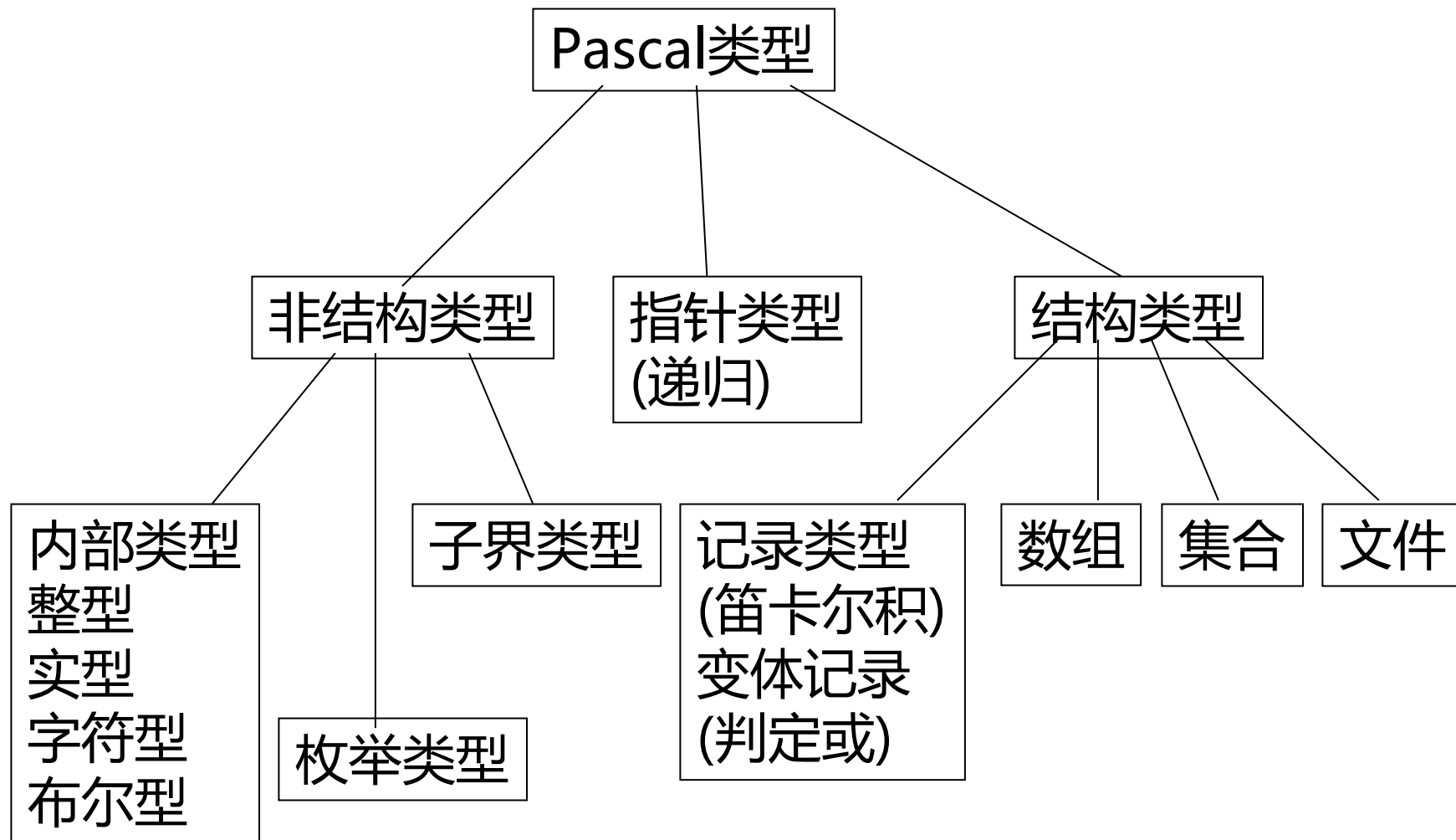
指针
可用

指针的例子:

```
type tree_ref = ↑binary tree node;  
binary_tree_node = record info:char;  
                        left,right:tree_ref  
end;  
  
var my_tree:tree_ref;  
my_tree := nil;  
new(my_tree);  
my_tree↑.info:=symbol;  
my_tree↑.left:=nil;  
my_tree↑.right:=nil;
```

。 1 150

4.小结



第六节 C语言数据类型结构

1.非结构类型：

□分为**内部类型**和**用户自定义类型**

□非结构内数类型有整型、实型和字符型

各类整型数据
类型的特性

类型	类型标志符	数值范围	占用字节数
基本型	int	-32768~32767	2
短整型	Short	-32768~32767	2
长整型	Long	$-2^{31} \sim (2^{31}-1)$	4
无符号整型	Unsigned	0~65535	2
无符号短整型	Unsigned short	0~65535	2
无符号长整型	Unsigned long	$0 \sim (2^{32}-1)$	4

□实型又称浮点型，其值是实数的一个子集，分为单精度和双精度两种类型

浮点型数据类型的特性

类型	类型标志符	点用字节数	能表示数值的有效位	数值范围	阶的范围
单精度型	Float	4	7	$-10^{38} \sim 10^{38}$	-38~38
双精度型	Doubl	8	15~16	$-10^{308} \sim 10^{308}$	-308~308

e

□字符型数据的值是一个有限字符集的元素；在C语言中，int类型与char类型在存储中没有本质区别；



注意：C语言中没有布尔（bool）类型；0表示false，非0表示true。

用户自定义的非结构类型

□ 用户自定义的非结构类型在C语言中称为枚举类型 (enum)

`enum bool {false, true};`

或

`typedef enum {false, true} bool;`

(1) 定义了一个新类型bool

(2) bool数据类型的取值为false和true

(3) 定义了一个顺序:
false < true

(4) 可对这个类型的变量进行赋值和比较等操作

```
enum bool {false, true};  
enum bool b;  
b = true;  
if(b == true) {.....}
```


2. 聚合构造

(1) 数组

- 实现有限映像
- 说明的格式

<类型说明符> <

例如:

```
int intarr[5];  
char chararr[255];  
bool boolarr[3];
```

命名为boolarr的数组包含3个元素，类型为布尔型



注意： C语言中数组的下标总是从0开始。

2. 聚合构造

□可以定义多维数组

□说明的格式

<类型说明符> <数组名> <维数>

数组a[3][4]的存放次序为:

a[0][0] a[0][1] a[0][2] a[0][3]
a[1][0] a[1][1] a[1][2] a[1][3]
a[2][0] a[2][1] a[2][2] a[2][3]

例如:

float farr[3][4];

char c[2][2][2];

定义了一个字符
型的3维数组

□C语言的数组按行存放

□对数组名的处理相当于指针



合法

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

```
pa = &a[0]
```

2. 聚合构造

(2) 结构

□ C语言中构造符struct支持笛卡尔积

□ 说明的格式

struct <结构体名> {成员表列};

□ 成员表列由若干个成员类型说明组成:

<类型标识符> <成员名>;

例如: **struct student {
 int num;
 char name[2];
 char sex;
 int age;
 float score;
 char addr[30];
}**

2. 聚合构造



注意：C语言中结构不能整体赋值和输出，只能对其中的各个成员分别进行操作。

例如：**struct** student **me, you**;
 strcpy(**me.name**, "john");
 me.sex = 'M' ;
 me.age = 21;
 me.score = 80;
 strcpy(**me.addr**, "UESTC");

- 在内存中，结构的各成员依次存放；
- 结构体可以嵌套；

例如：**struct** student {
 int num;
 char name[2];
 char sex;
 struct date birthday;
 ...
}

2. 聚合构造

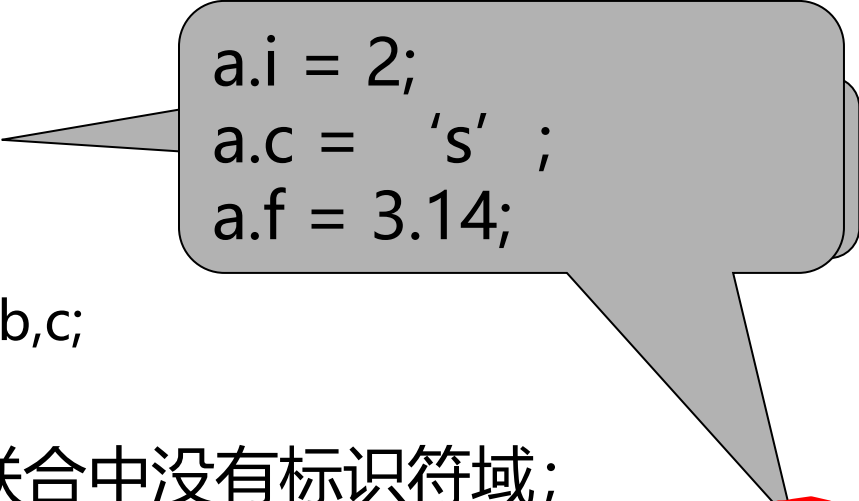
(3) 联合

□C语言中构造符union（联合）支持判定或


联合结构举例：

```
union data {  
    int i;  
    char c;  
    float f;  
}
```

```
union data a,b,c;
```



```
a.i = 2;  
a.c = 's' ;  
a.f = 3.14;
```

- 
- 注意：** (1) C语言中联合中没有标识符域；
(2) 单元中的值的类型，取决于程序员的使用；
(3) 程序的编制依赖于实现；

2. 聚合构造

(4) 文件

- C语言中文件是一个字符序列;
- 分为ASCII码文件和二进制文件;

C语言中文件的预定义格式如下:

```
typedef struct {
```

```
    int _fd;
```

```
    int _cleft;
```

```
    int _mode;
```

```
    char * _next;
```

```
    char * _buf;
```

```
} FILE;
```

文件名

缓冲区中剩下的字符数

文件操作模式

下一个字符地址

缓冲区指针



注意: C语言中的文件操作比较丰富, 打开、关闭、定位、读、写等; 除了顺序读写以外, 还可以随机读写;

3. 指针

- C语言中的**指针**是第三种数据类型，是非结构类型；
- 可用来构造结构类型；
- 支持递归；

利用指针定义递归结构的例子：

```
struct tree {  
    char day;  
    struct tree *lchild;  
    struct tree *rchild;  
};  
struct tree *my_tree;
```



注意： C语言中没有空指针，对指针赋0值来表示空；

4. 空类型

- C语言中有一种特殊的数据类型void，称为**空类型**；
- 是一种非结构类型；
- 有两个主要用途：
 - ✓ 用来表示一个无返回值的函数
 - ✓ 用来表示不确定类型的指针

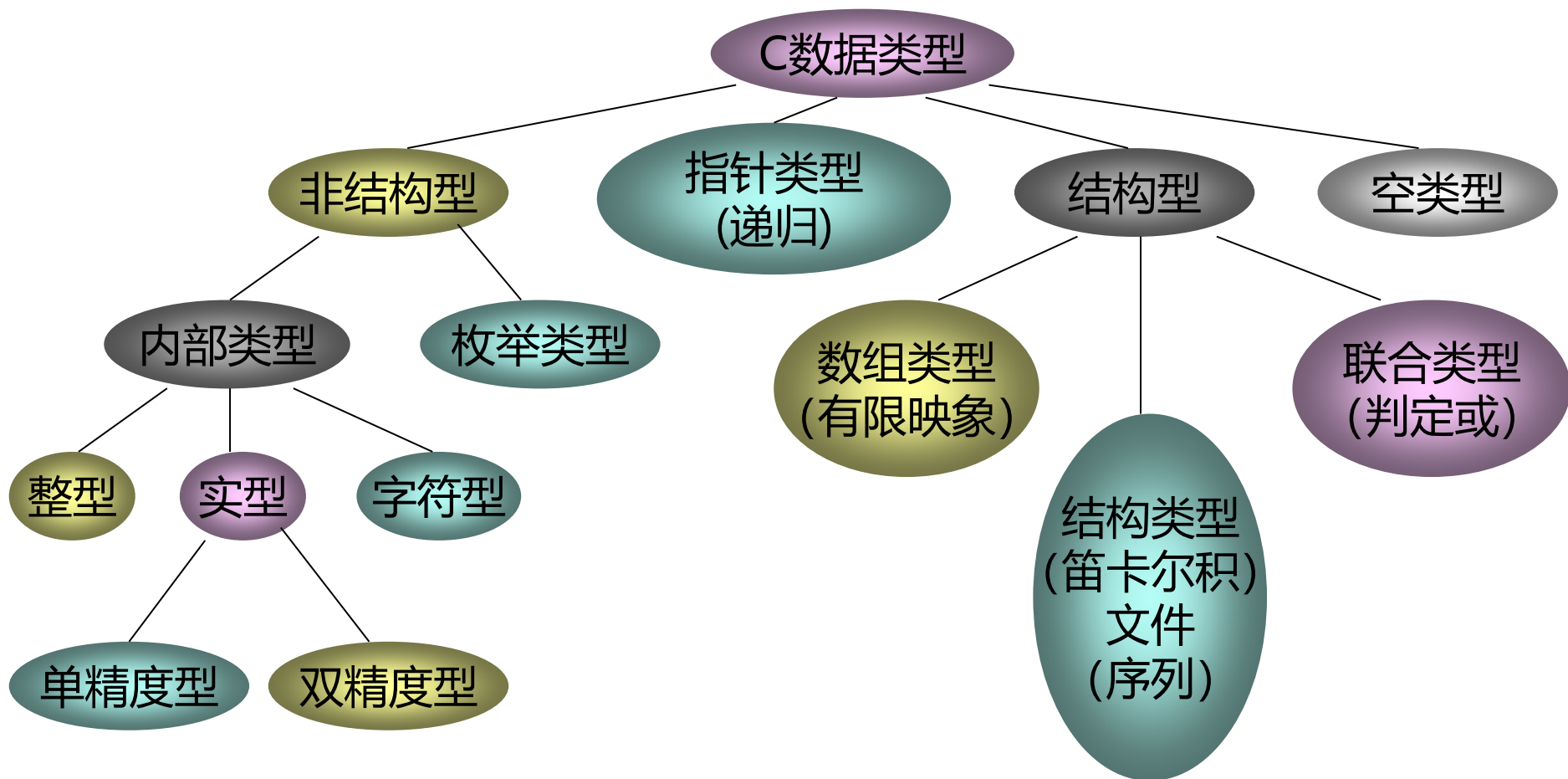
例如：

```
void main() {  
    int i;  
    i = 1;  
}
```

例如：

```
void *p;  
表示这是一个指针，它的  
值是一个地址，但不指明p  
指向的值是什么类型。
```


4. C数据类型小结



第七节 JAVA语言的数据类型

1.内部类型:

整型 int long short byte

实型 float double

字符型 char

布尔型 boolean

独立数据类型，不能直接转换成数值。

2.用户定义类型:

不支持指针、结构和联合、只支持数组

(类型) <数组名>[] | (类型) <数组名>[][]

```
int ai[ ]  
int ai[ ]=new int[10]
```

特点:

不需要说明上下界, 动态的灵活数组

用new语句显示分配实际空间

New建立实际数组, 元素个数不能随意改变

第八节 抽象数据类型

□ 用户定义类型与内部类型的异同

- ✓ 都建立某种基本表示的抽象

即程序员在定义的抽象中，应该对用到这个对象的用户隐蔽尽可能多的信息。

操作表示不能对它的成分进行右再别的抽象可以对

软件重用、模块重用可以减少许多重复工作，达到提高软件开发的效率

pygon是记录的抽象



问题

如何提供更好的抽象来达到
信息隐蔽、重用的目的？

□ 抽象数据类型的定义

满足下述特性的用户定义类型称为**抽象数据类型**:

- ① 在实现该类型的程序单元中,建立与表示有关的基本操作;
- ② 对使用该类型的程序单元来说,该类型的表示是隐蔽的。

抽象数据类型隐蔽了表示的细节,通过过程来访问抽象数据对象。对象的表示是被保护的,外界不能对它进行直接操作。对抽象数据类型的实现进行修改,只能在描述这个实现的程序单元中。

C++语言的抽象数据类型

□C++语言中的抽象数据类型称为类（class），类的实例称为对象（object）；

□C++的类满足抽象数据类型的条件（类型名标识符



C++语言类定义的一般形式

类体，其中私有段、保护段以及公有段的实现对外部不可见；公有段的数据及函数名外部可见

class <类名>

{

private:

私有段数据定义；

私有段函数定义；

protected:

保护段数据定义；

保护段函数定义；

public:

公有段数据定义；

公有段函数定义；

- ❑ 类的实例是对象，对象继承类中的数据和方法；
- ❑ 各对象的数据初始化和各个数据成员的值不同；
- ❑ C++ 支持重载和多态；
- ❑ C++ 的继承性通过派生类来实现。

JAVA抽象数据类型

- JAVA中的抽象数据类型称为类 (class),类的实例称为对象
- C++的类满足抽象数据类型的条件 (1) 和 (2) ;

- 类说明格式

class <类名>

{ <类体> }

- 超类 (Super Class)

class <类名> **extends**<父类名>

{ <类体> }

- 列出类的实现接口的类

class <类名> **extends**<父类名>**implements** call

{ <类体> }

- 抽象类 (Abstract Class)

abstract class <类名> **extends**<父类名>

{ <类体> }

- 最终类 (Final Class)

final class <类名> **extends**<父类名>

{ <类体> }

第九节 类型检查

在编译时进行的检查

□ 对数据对象的类型和使用的操作是否匹配的一致性检查称为**类型检查**;

□ 语言的类型检查分为**静态检查** (static checking) 和 **动态检查** (dynamic checking)



静态检查使程序

语言的类型检查不能全部在编译时完成, 有些要在运行时才能完成, vb, php, pascal

语言的类型检查全部在编译时完成, java、C++、python

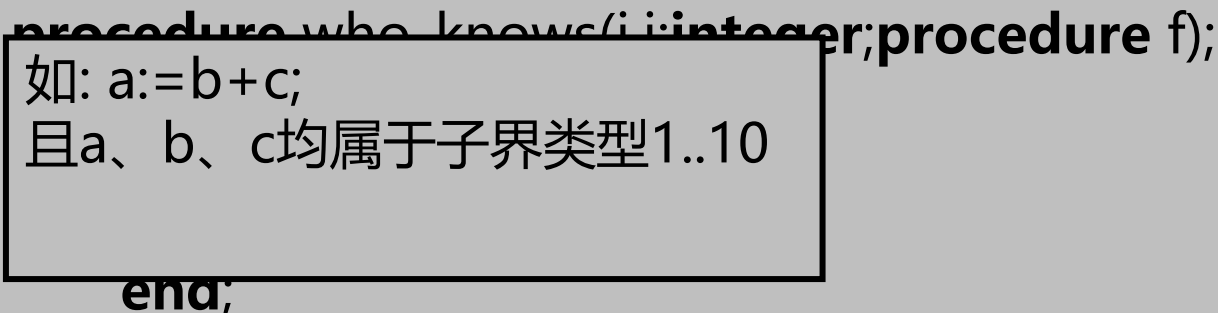
执行的效率, 但影响了可读性, 且影响了执行效率

□ 语言类型检查可分为**无类型语言**、**弱类型语言**和**强类型语言**;

❑ PASCAL是弱类型语言

❑ 理由:

- ① 编译时,不能确定一个过程中的过程参数和子程序参数类型
- ② Pascal的子界类型不能静态检查
- ③ 变体记录的标识符可以在运行时改变
- ④ Pascal没有严格规定类型的一致性规则



```
procedure who_knows(i:integer;procedure f);  
  如: a:=b+c;  
  且a、b、c均属于子界类型1..10  
end;
```

❑ 将一个类型的值转换成另一个类型的值，称为**类型转换 (Type Conversion)**;

❑ 类型转换分为**拓展** (widening) 和**收缩** (narrowing);

❑ 在某些语言中，类型转换的要求和规则是隐式的，它由编译器自动生成类型转换代码。

转换之后的类型值的集合; 转换之前的类型值的集合包含转换之后的类型值的集合; 例如:
整型→实型 **实型→整型**



总结: 收缩可能导致某些信息的丢失。例如，在 PL/1 语言中，实数到整数的收缩采用截断；而 PASCAL 则使用舍入法进行从实数到整数的收缩。

一些语言规定的转换规则:

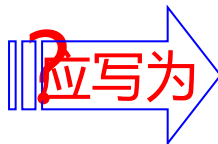
① FORTRAN语言:

- ✓ 转换规则隐式给出;
- ✓ 转换规则根据类型和类型之间的优先级来确定; 由低级类型向高级类型转换;
- ✓ FORTRAN的类型优先级为:
- ✓ COMPLEX > DOUBLE PRECISION > REAL > INTEGER

② PASCAL语言:

- ✓ 只允许整数到实数, 以及子界类型到整数的转换;
- ✓ 其他的转换必须显示处理

~~var r:real;
i:integer;
i:=r;~~



var r:real;
i:integer;
i:=round(r);

③ ALGOL 68语言:

- ✓ 完全的、形式化的隐式转换规则;
- ✓ 它一共给出6种隐式转换规则;

④ Ada语言:

- ✓ 必须显示转换;

⑤ C++:

- ✓ 隐式转换: 混合运算, 将表达式的值赋给变量, 实参向形参传值, 函数返回结果;
- ✓ 显示转换

❑ 若T1和T2是两个类型，且T1的任何值都可以赋予T2类型的变量，T1类型的实参可以对应类型T2的形参，反之亦然，则称T1和T2是兼容的。

❑ 有两种类型的相容性

- ✓ 名字等价
- ✓ 结构等价

两个变量的类型具有相同的结构。

验证法：用用户定义类型的定义来代替用户定义名，重复这一过程，直到没有用户定义类型名为止。

两个变量的类型名相同

❑ 常见语言的类型例：

- ✓ Ada采用：

```
type t=array[1..20] of integer;
```
- ✓ ALGOL 68：

```
var a,b:array[1..20] of integer;
```
- ✓ PASCAL：

```
var a:integer;
```

❑ 名字等价的实例

❑ 结构等价的实例

```
var c:t;
d:record a:integer;
      b:t;
end
```

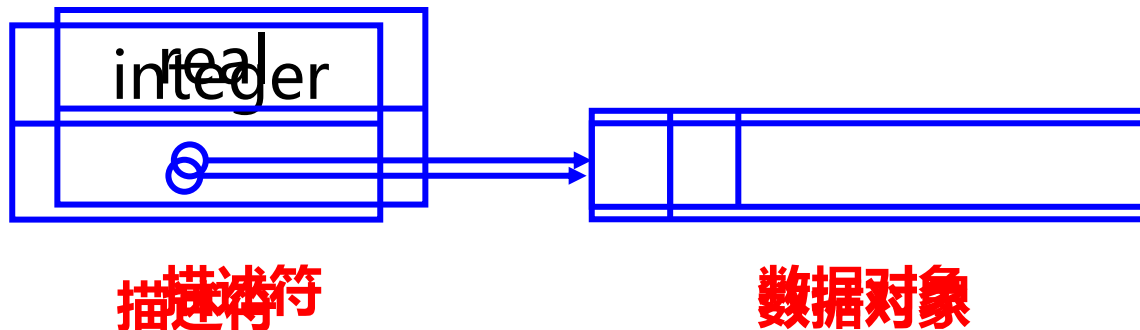
第十二节 实现模型

- ❑ 在实现模型中，数据用描述符和数据对象来表示；描述符用来描述数据对象的所有属性；
- ❑ 只考虑原理性的实现，不考虑效率；
- ❑ 以PASCAL语言为例；

1.内部类型和用户定义的非结构类型的实现模型

- ❑描述符一般由“类型”和一个指针组成
- ❑子界的描述符必须包括子界的界值
- ❑布尔型和字符型可以压缩存储

整数变量的表示

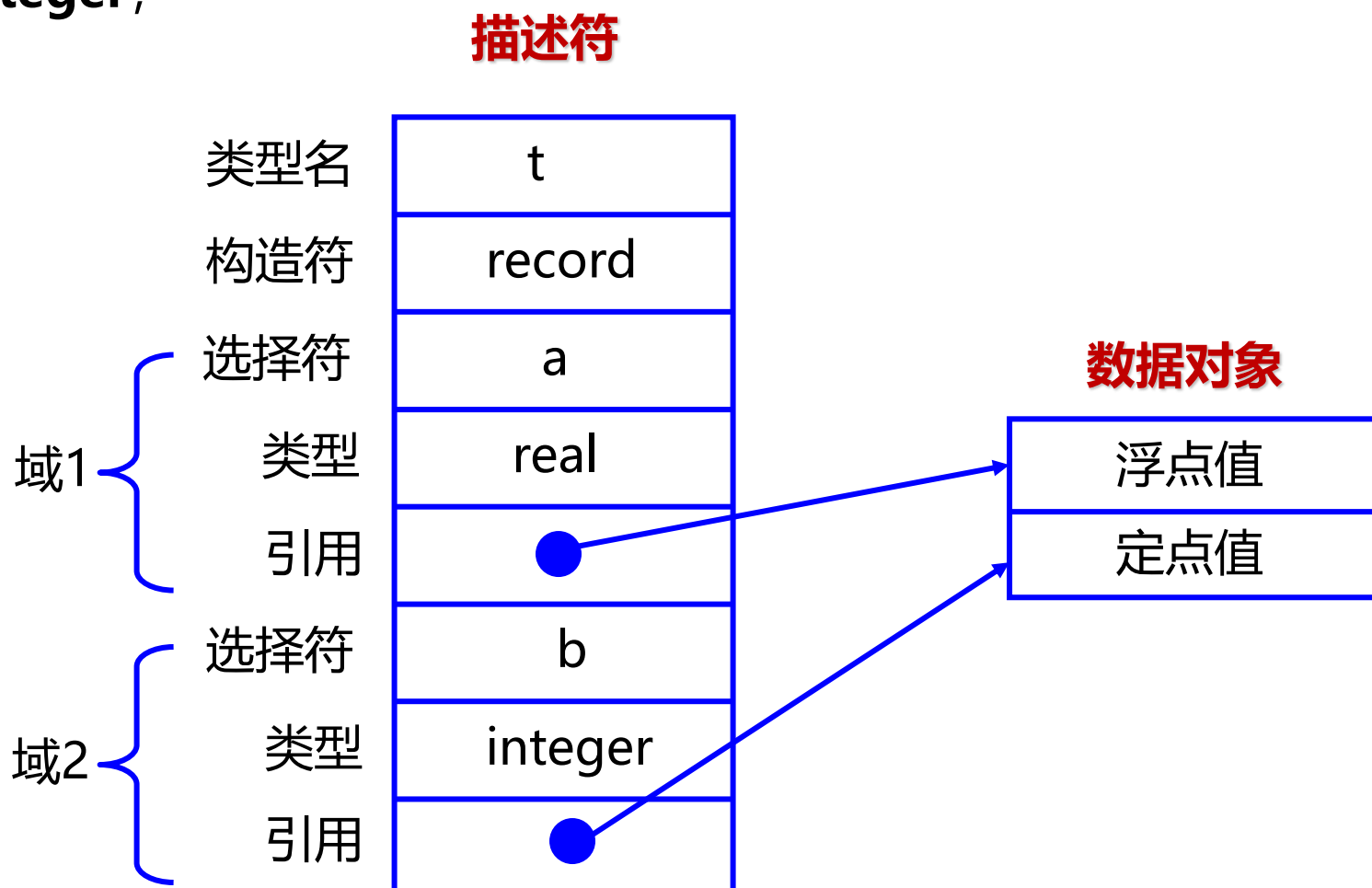


2.结构类型的实现模型

□笛卡尔积

- ✓ 各成分顺序排列（数据）
- ✓ 描述符包含：**类型名**、**构造符**、若干三元式。每个域对应一个三元式(**选择符名,域类型,指针**)
- ✓ 每个成分占整数个可编址的存储单元(字编址或字节编址)
- ✓ 可以用packed显式说明压缩存储

例：
type t=**record** a:**real**;
 b:**integer**;
end;



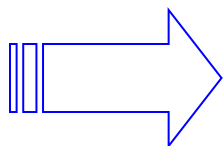
□有限映像

- ✓ 为每一成分分配整数个可编址的存储单元
- ✓ 描述符包括: **类型名、构造符、定义域的基类型、下界、上界、成分类型、单元个数、首地址**
- ✓ $A[i]$ 地址公式的计算

$$b + k(i - m) = b - km + ki = b' + ki$$

其中 b 为首地址, k 为每个元素所占存储单元个数, m 为下界

动态数组描述符中的某些属性需要到运行时才能确定, 描述符如何处理?



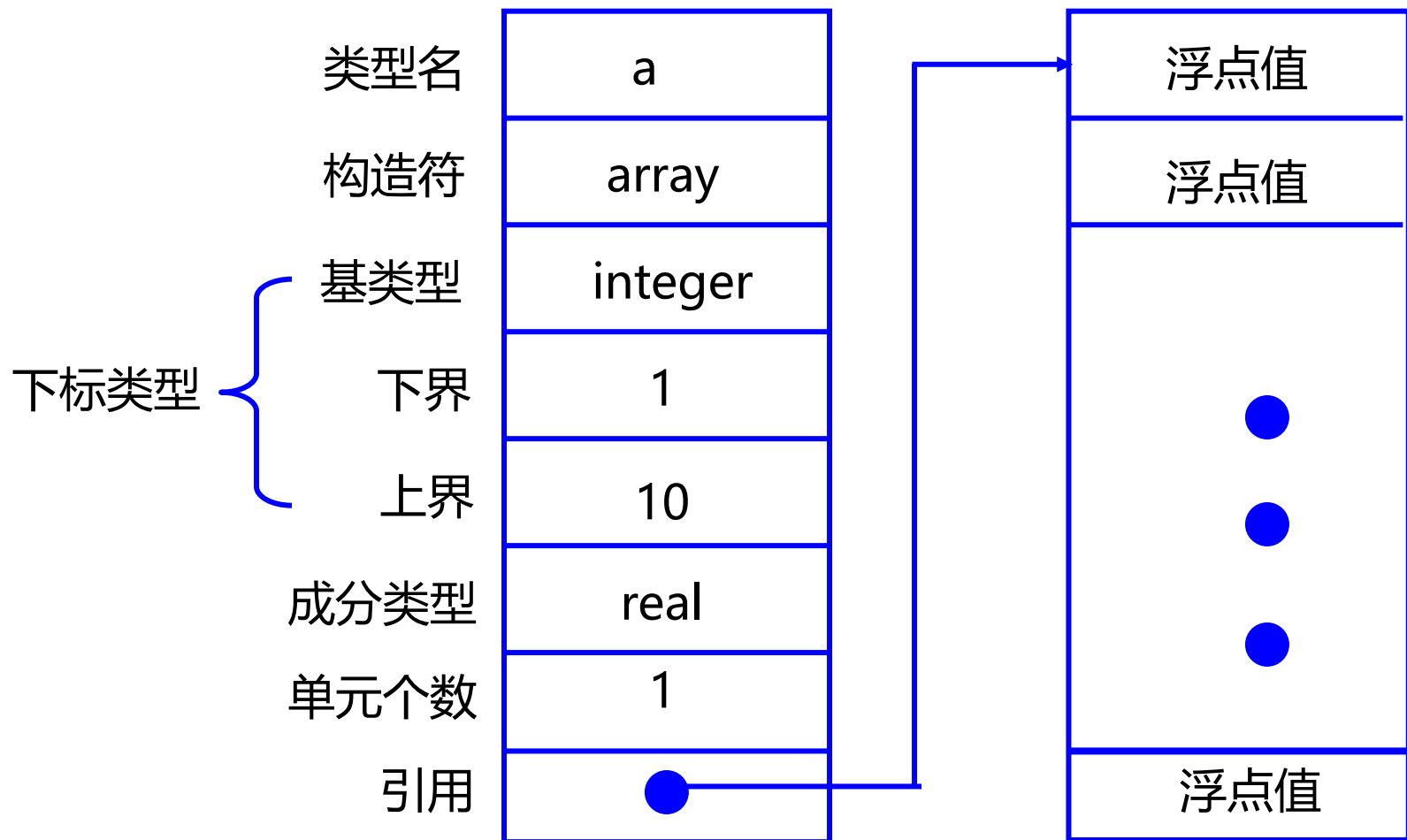
静态部分: 只包含编译时所需信息 (成分类型、对动态部分的引用等)

动态部分: 包含对数据对象对象的引用, 运行时分配到单元活动记录中

例：
type a=array[1..10] of real;

描述符

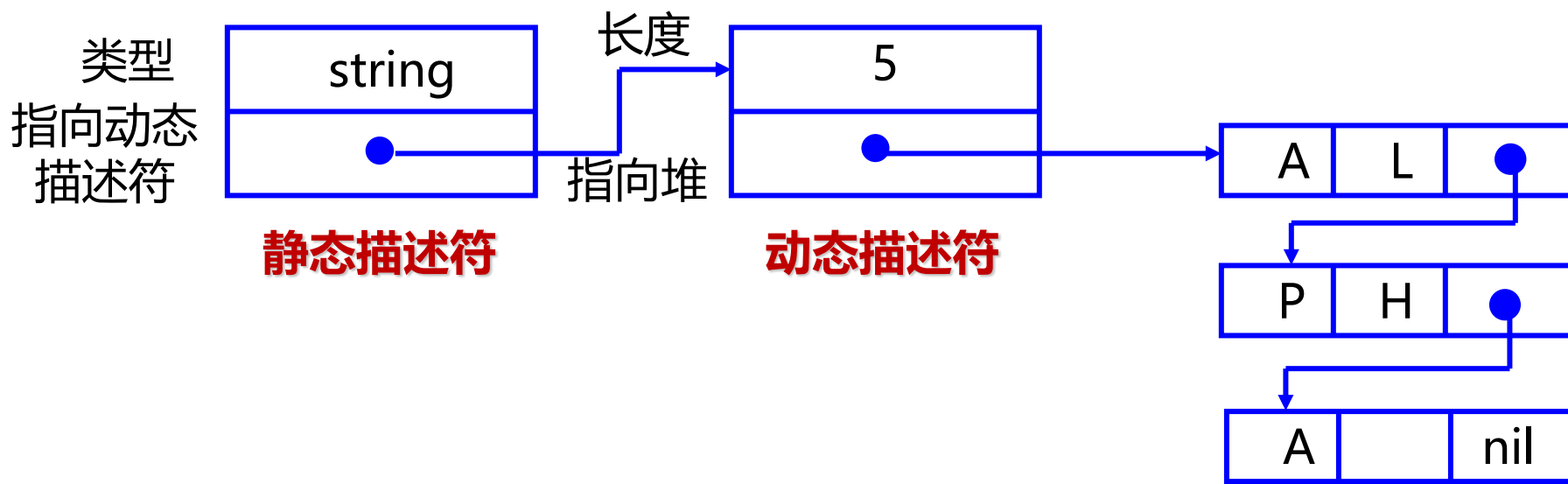
数据对象



□ 序列

- ✓ PASCAL中的串长度静态可定；静态分配；
- ✓ 其他语言中（如SNOBOL 4和ALGOL 68中），可变长串表示：

静态描述符 + 动态描述符 + 堆



□判定或

- ✓ 对**判定或**类型的变量所分配的空间应足以容纳需要最大空间的变体的值
- ✓ Pascal的变体记录的表示包括:**描述符、数据对象、case表、若干变体描述符**

例:

```
type v=record a:integer;  
              case b:boolean of  
                true:(c:integer);  
                false:(d:integer;  
                       e:real)  
              end;
```



描述符

v
variant record
a
integer
•
b
boolean
n •
•

数据对象

定点值
布尔值

case表

true	•
false	•

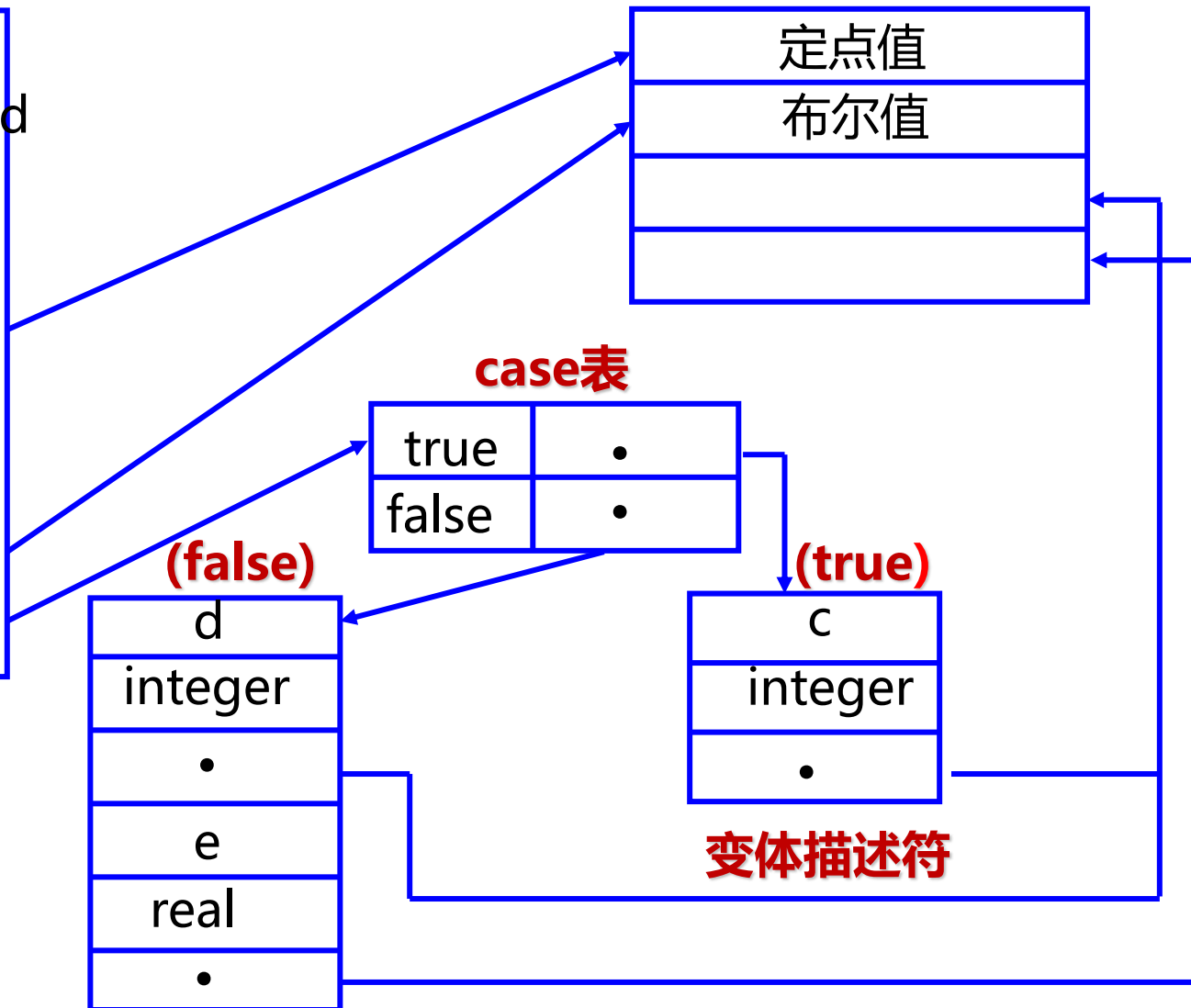
(false)

d
integer
•
e
real
•

(true)

c
integer
•

变体描述符



□ 幂集

- ✓ 集合关联若干个机器字,通过每一位的取值可知该集合中有基类型的哪些元素
- ✓ 位操作

□ 指针

- ✓ 指针变量的表示类似于内部类型, 只是其值为一地址, 并且它指向的数据对象分配在堆上

□ 层次数据结构对象的表示

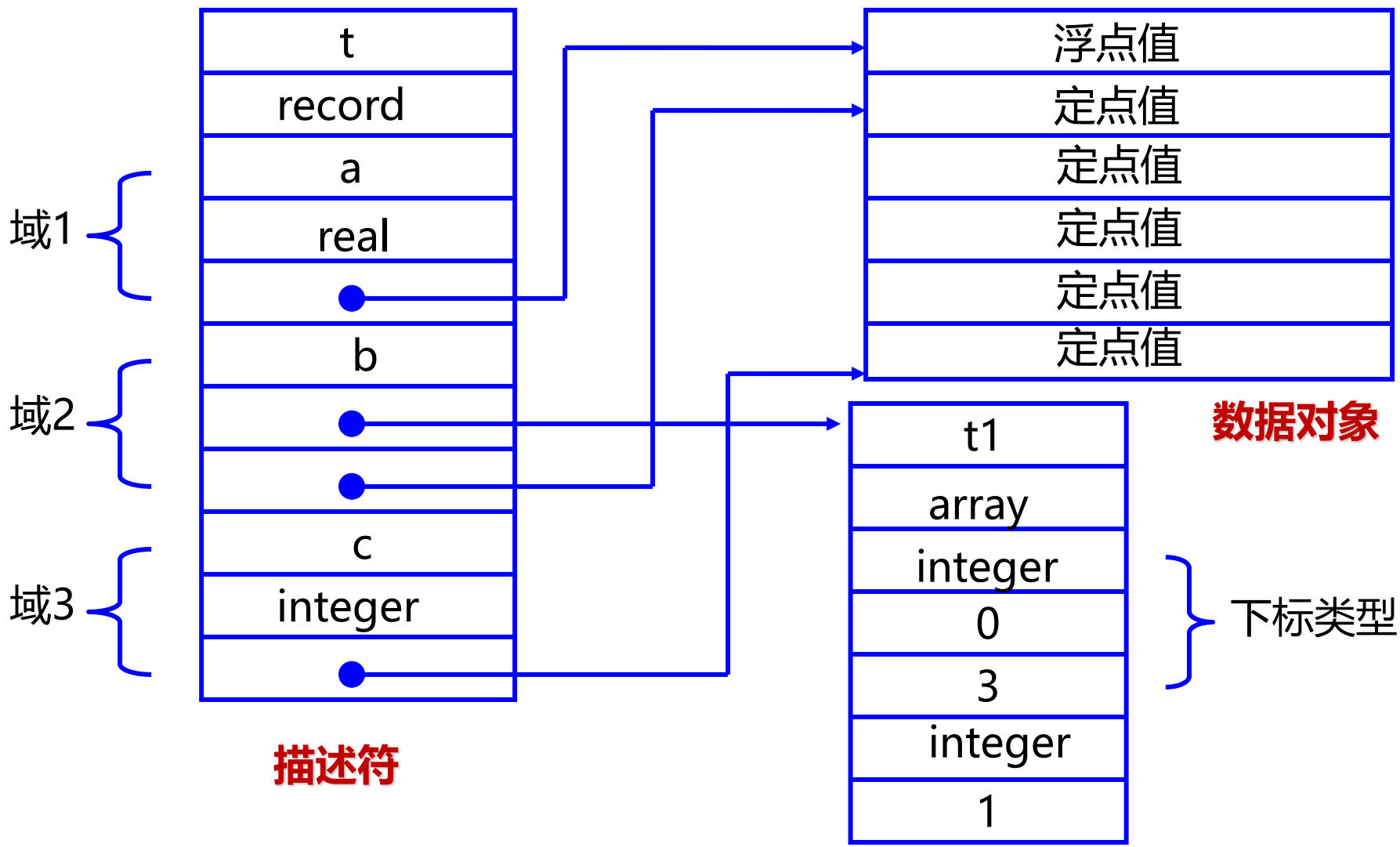
- ✓ 描述符中的类型可以指向另外的描述符

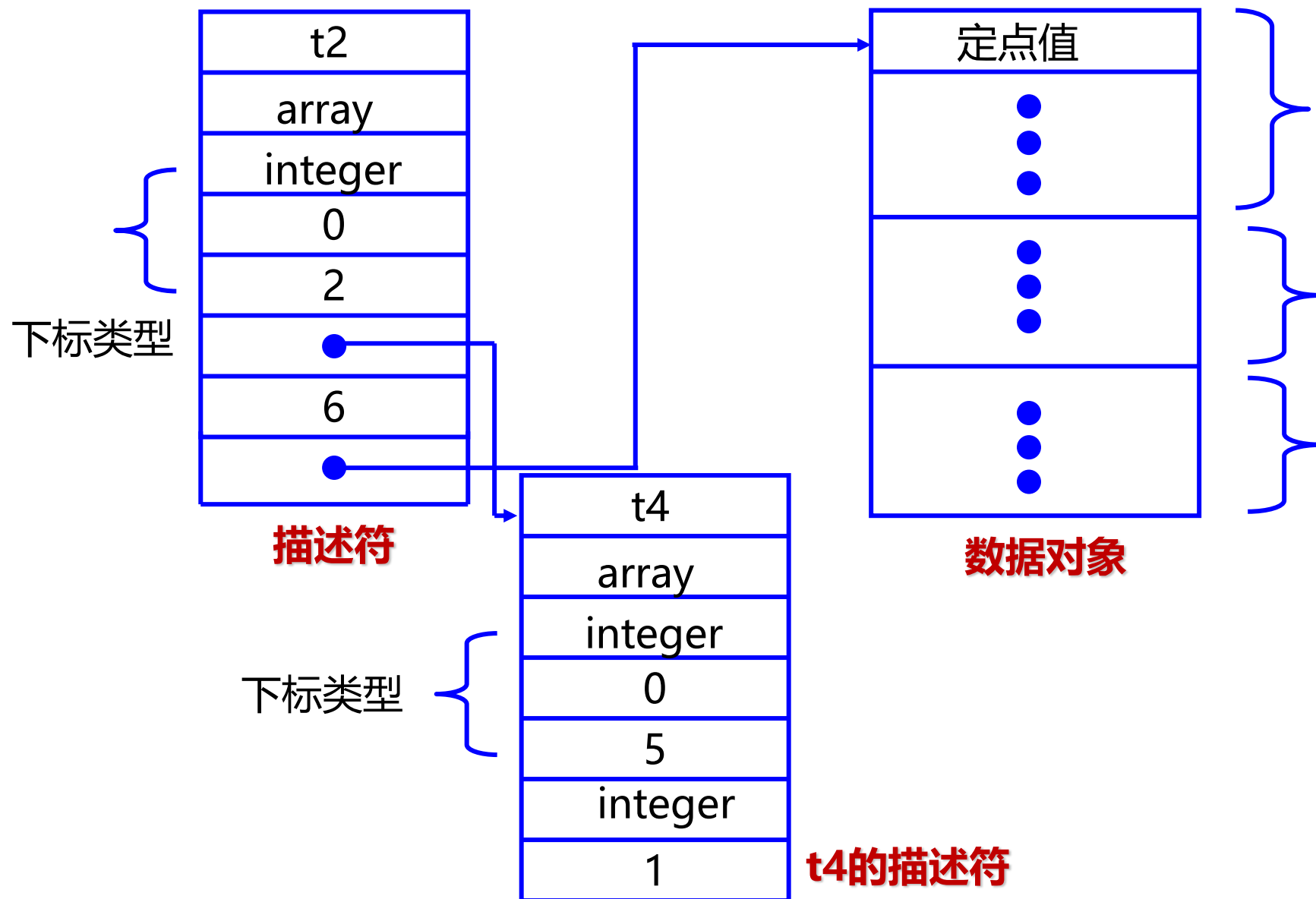
例1：

```
type t1=array[0..3] of integer;  
  t=record a:real;  
    b:t1;  
    c:integer;  
end;
```

例2：

```
type t4=array[0..5] of integer;  
  t2=array[0..2] of t4;
```





第二章习题

1. 必做题:

2-2、2-8

2. 思考题:

2-3、2-13、2-14