

# 第四章：语义分析

语义检查的内容  
标识符的语义表示  
类型的语义表示



# 1.1 语法和语义的区别

- w 语言包括4个内容：词法、语法、语义、语用
- w 语法：关于什么样的字符串才是该语言在组成结构上合法的程序的法则。
- w 语义：关于结构上合法的程序的意义法则。

# 1.2 程序设计语言语义的分类

## 静态语义

在编译阶段(compile-time)可以检查的语义

例如：标识符未声明

## 动态语义

目标程序运行时 (run-time) 才能检查的语义

例如：除零、溢出错误。

# 1.3 语义分析的功能

- 语义分析的内容：
  - 类型分析;
  - 标识符相关信息提取;
- 语义分析的功能:
  - 检查语义错误
  - 构造标识符属性表 (符号表)
- 语义分析的实现:
  - 与语法分析相结合

# 1.4 语义错误检查

## 类型检查

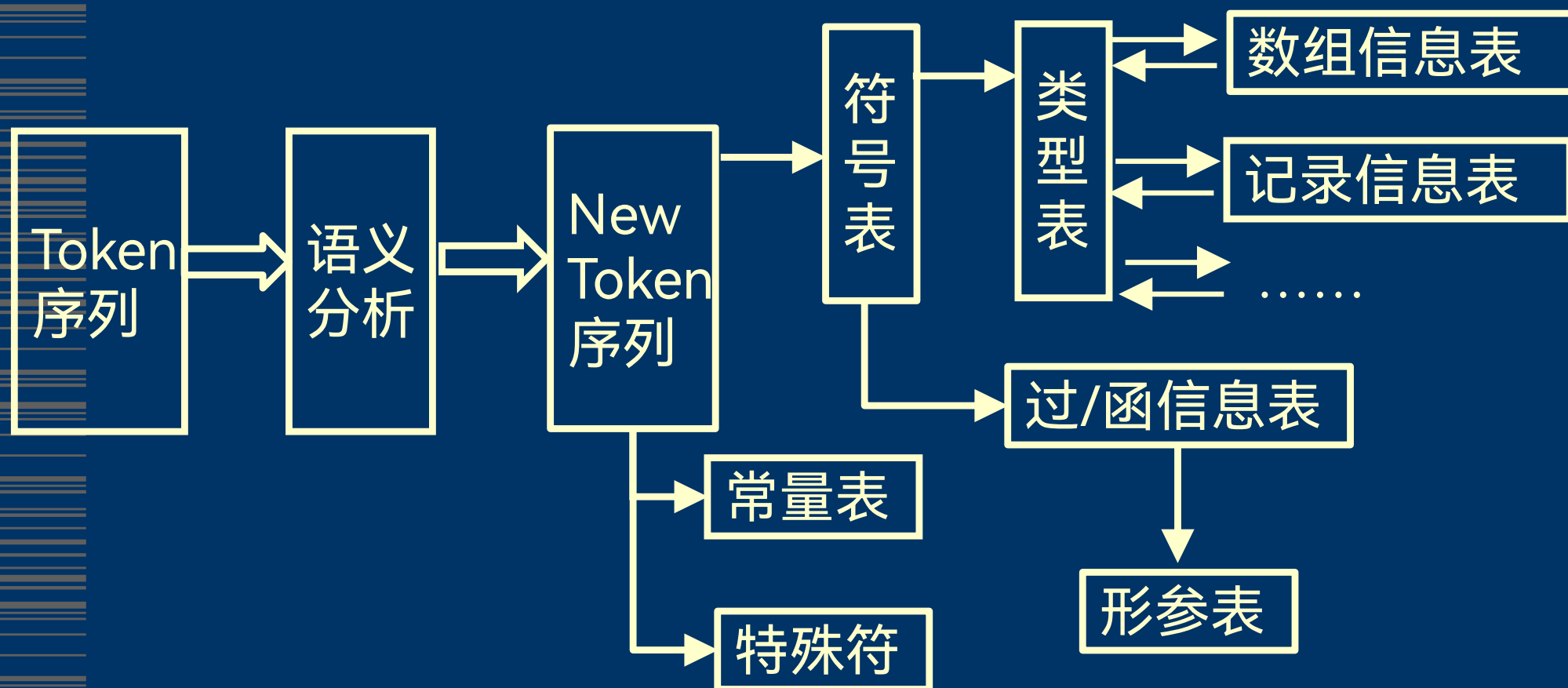
1. 各种条件表达式的类型是不是bool型?
2. 运算符的分量的类型是否相容?
3. 赋值语句的左右部的类型是否相容?
4. 形参和实参的类型是否相容?
5. 下标表达式的类型是否为所允许的类型?
6. 函数说明中的函数类型和返回值的类型是否一致?

# 1.4 语义错误检查

## 一般性的语义检查

1.  $V[E]$ 中的 $V$ 是不是变量,而且是数组类型?
2.  $V.id$ 中的 $V$ 是不是变量,而且是记录类型?  $id$ 是不是该记录类型中的域名?
3.  $y+f(\dots)$ 中的 $f$ 是不是函数名?形参个数和实参个数是否一致?
4.  $p(\dots)$ 语句中的 $p$ 是不是过程名?形参个数和实参个数是否一致?
5.  $*V$ 中的 $V$ 是不是指针或文件变量?
6. 是否有变量的声明、标识符有没有重复声明?

## 2 语义分析处理后的结果



## 3.1 标识符在程序中的出现

### □ 声明性出现

如： `int x, int a[10];`

Pascal语言中出现在程序头、函数头部分

### □ 使用性出现

如： `x=x+1; a[0]=a[1]+a[2];`

Pascal语言中出现在程序体函数体部分



## 3.2 标识符的种类

- 常量标识符
- 类型标识符
- 变量标识符
  - 实在变量
  - 形参变量
    - 值引用型    地址引用型
- 过函标识符
  - 实在过函
  - 形式过函
- 域名标识符

## 3.3 标识符的属性

□ 标识符的语义信息要把其主要内容区分开，标识符的主要信息主要包括以下几个：

1. 名字
2. 种类信息
3. 类型信息
4. 对不同类型的独特的信息

## 3.4 标识符的语义表示

### □ 常量标识符

Name	Kind	TypePtr	Value
------	------	---------	-------

其中各个域的含义如下：

- ❖ **Name**是常量的名字；
- ❖ **Kind** = `constKind`，表明该标识符是常量标识符；
- ❖ **Type** = `TypePtr`，其中`TypePtr`是指向具体常量的类型的内部表示的指针；
- ❖ **Value** = `ValPtr`，其中`ValPtr`是指向具体常量值的内部表示的指针。

例： C语言的常量定义：

```
#define pai 3.14
```

```
#define count 100
```

常量标识符pai 和count的内部表示为：

pai	constKind	realPtr	↑ <3.14>
count	constKind	intPtr	↑ <100>

## 3.4 标识符的语义表示

类型标识符

Name	Kind	TypePtr
------	------	---------

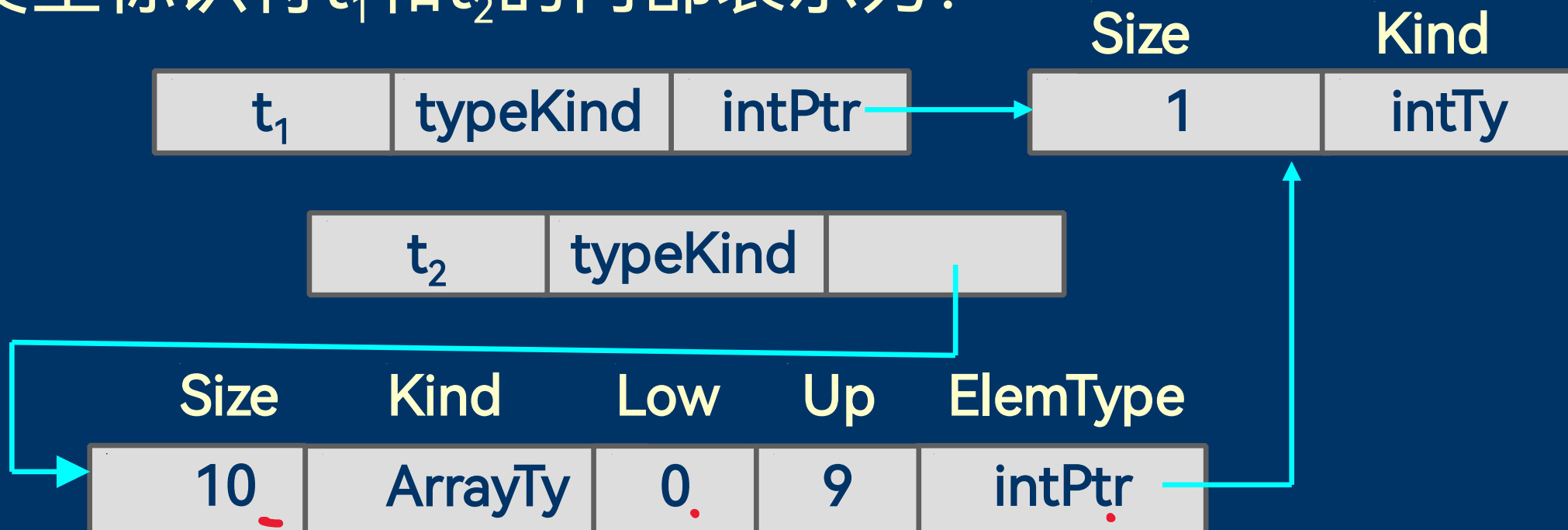
- ❖ Name是类型标识符的名字;
- ❖ Kind = typeKind, 表示标识符是类型标识符;
- ❖ Type = TypePtr, 指向类型标识符指代的类型的内部表示

例:

C语言的类型定义:

```
typedef int t1;  
typedef int t2[10];
```

类型标识符 $t_1$ 和 $t_2$ 的内部表示为:



## 3.4 标识符的语义表示

### □ 变量标识符的内部表示

Name	Kind	TypePtr	Access	Level	Off	Value
------	------	---------	--------	-------	-----	-------

- ❖ **Kind** = varKind, 表明该标识符是变量标识符;
- ❖ **Access** = dir表示变量是直接变量, **Access** = indir表示变量是间接变量;
- ❖ **Level**表示该变量声明所在主程序/函数/过程的层数;
- ❖ **Off**表示该变量相对它所在主程序/函数/过程的内存块起始地址的偏移量;
- ❖ **Value** = ValPtr, 如果变量定义时说明了初值, 则为初值的内部表示的指针, 否则为空。

例： C语言的变量声明：

int x=10;

float y;

float\* z;

变量标识符x、y和z的内部表示为（当前层为L，  
当前偏移量为off）：

可指向

x	varKind	intPtr	dir	L	off	↑ <10>
y	varKind	realPtr	dir	L	Off+1	null
z	varKind		indir	L	Off+3	null

1

pointTy

realPtr



## 3.4 标识符的语义表示

### □ 域名标识符的内部表示

Nam	Typeptr	Kind	Off	HostType
-----	---------	------	-----	----------

- ❖ **Typeptr**表示域名所对应域名标识符类型的内部表示
- ❖ **Kind**=fieldKind, 表明该标识符是域名标识符.
- ❖ **Off** 域名相对他所在记录的偏移量
- ❖ **HostType**记录类型指针 (宿主类型)

## 3.4 标识符的语义表示

### □ 过程/函数标识符的内部表示

Nam	TypePtr	Kind	Level	off	Para	Class	Code	Size	Forwar
-----	---------	------	-------	-----	------	-------	------	------	--------

- ❖ **TypePtr**表示函数返回值类型的内部表示(过程情形是 NULL)
- ❖ ~~Kind = routKind;~~
- ❖ **Level**表示过/函的层数;
- ❖ **Off**只对形式过/函有效, 表示形式过/函在所属过/函内存块中的偏移;

## 3.4 标识符的语义表示

### □ 过程/函数标识符的内部表示

Nam	TypePtr	Kind	Level	off	Para	Class	Code	Size	Forwar
-----	---------	------	-------	-----	------	-------	------	------	--------

- ❖ **Param**表示过/函的参数表指针，参数表的结构同符号表的结构相同，参数信息可以填入符号表，也可以填入单独的参数表当中；
- ❖ **Class**= actual表示实在过/函，**Class** = formal表示形式过/函；
- ❖ **Code**只对实在过/函有效，表示过/函定义对应生成的目标代码的起始地址，当目标代码生成时回填得到，形式过/函的code为NULL；

## 3.4 标识符的语义表示

### □ 过程/函数标识符的内部表示

Nam	TypePtr	Kind	Level	off	Para	Class	Code	Size	Forwar
-----	---------	------	-------	-----	------	-------	------	------	--------

- ❖ **Size**只对实在过/函有效，表示过/函的目标代码所占内存区的大小，也要当目标代码生成以后回填得到；
- ❖ **Forward** 属性只对实在过/函有效，Forward= true表示过/函是超前声明，Forward = false表示过/函不是超前声明。

例：C语言的函数定义：

int f(int x,float\* y, int inc(int\* a)) ..... "头"

{

.....f的函数体部分

}

Name	TypePtr	Kind	Level	off	Parm	Class	Code	Size	Forward
------	---------	------	-------	-----	------	-------	------	------	---------

f	intPtr	rouKind	L			actual		XXX	false
---	--------	---------	---	--	--	--------	--	-----	-------

x	varKind	intPtr			dir	L+1	off <sub>0</sub>
---	---------	--------	--	--	-----	-----	------------------

y	varKind				indir	L+1	off <sub>0</sub> +1
---	---------	--	--	--	-------	-----	---------------------

inc	intPtr	rouKind	L+1	2		formal			
-----	--------	---------	-----	---	--	--------	--	--	--

1	pointTy	realPtr
---	---------	---------

a	varKind	intPtr			indir	L+2	off <sub>0</sub>
---	---------	--------	--	--	-------	-----	------------------

## 4. 类型的语义表示

- 类型语义信息的作用
  - ❖ 类型的语义检查
  - ❖ 分配空间的大小

## 4. 类型的语义表示

W 类型可以分成下面几大类：

- ❖ 标准的类型：整形、实型、bool、字符类型，这是标准的数据类型
- ❖ 自定义的数据类型，子界类型，枚举类型
- ❖ 结构数据类型：数组，集合，记录
- ❖ 特殊的指针类型，文件类型

W 类型存储占用空间大小size属性：

为方便起见，规定RealSize取2，IntSize、BoolSize、CharSize取1。


## 4. 类型的语义表示

□ 标准类型： 大小、种类

intPtr→  
boolPtr→  
charPtr→  
realPtr→

Size	Kind
IntSize	intTy
BoolSize	boolTy
CharSize	charTy
RealSize	realTy

Size	Kind
1	intTy
1	boolTy
1	charTy
2	realTy





## 4. 类型的语义表示

### □ 子界类型

Size	Kind	Low	Up	ElemType
------	------	-----	----	----------

□ type letter='a'..'z';

letterPtr  
r

1	subrangeTy	'a'	'z'	charPtr
---	------------	-----	-----	---------

子界

## 4. 类型的语义表示

### □ 枚举类型

Size	Kind	ElemList
------	------	----------

- ❖ size表示枚举类型所占空间的大小;
- ❖ Kind = ~~enumTy~~;
- ❖ ElemList是指向枚举常量表表头的指针.

### □ 枚举常量表内部表示:

Nam	Value
-----	-------

- ❖ Name表示枚举常量的名字;
- ❖ Value表示枚举常量所代表的整数值.

例：c语言的枚举类型

```
enum color {red, yellow, blue}
```

1	enumTy	
---	--------	--

red	0
yellow	1
blue	2

例：c语言的枚举类型

```
enum color {red=10, yellow=red+2, blue}
```

1	enumTy	
---	--------	--

red	10
yellow	12
blue	13

## 4. 类型的语义表示

### □ 数组类型

Size	Kind	Low	Up	ElemType
ArraySize	ArrayTy			TypePtr

- ❖ **Size**:  $\text{Size} = (\text{Up} - \text{Low} + 1) * \text{sizeof}(\text{ElemType})$
- ❖ **Kind** = ArrayTy, 表示是数组类型;
- ❖ **Low**表示数组下标的下界; 在C语言中Low = 0;
- ❖ **Up**表示数组下标的上界; (low、up合在一起可以是子界类型)
- ❖ **ElemType** 表示数组成分类型的内部表示指针。

例：C语言的数组

```
typedef int A[10];  
typedef char B [5] [10]
```

A和B的内部表示分别为：



## 4. 类型的语义表示

### □ 结构体和联合体

w RecBody的内部表示如下

Size	Kind	RecBody
------	------	---------

Name	Typeptr	Off	link
------	---------	-----	------

❖ Kind=structTy 表示结构体类型

❖ Kind=unionTy表示联合体类型

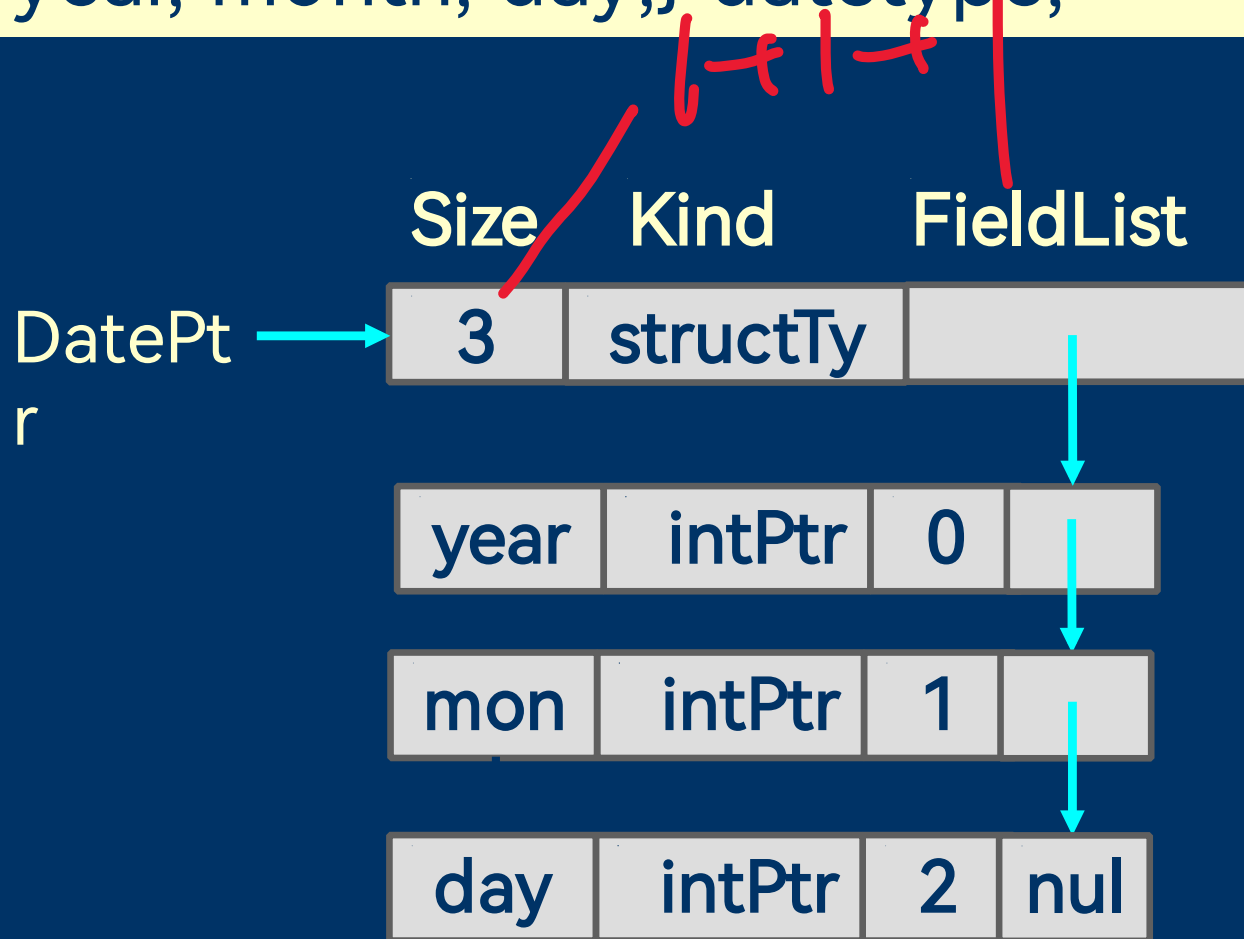
Name表示域名

Typeptr指向域的类型

Off表示纪录域相对于结构体类型分配的内存块起始地址的偏移量，对于联合类型而言，所有的域名标识符的起始偏移都是相同的，所以可以省略；

例:

```
typedef struct DateType  
{int year, month, day;} datatype;
```



对省略了.

例: c语言的联合体

typedef union {char ch; int i; float f;} datatype;

1 + 1 + 2

datatypeP  
tr:



~~~~~.



## 4. 类型的语义表示

### □ 指针类型

|      |      |          |
|------|------|----------|
| Size | Kind | BaseType |
|------|------|----------|

- ❖ size表示指针类型所占空间的大小，指地址的长度（一般一个单元）
- ❖ Kind = pointTy表示是指针类型；
- ❖ BaseType表示指针所指向空间的类型

例：c语言的指针类型

```
typedef int* T1;  
typedef float* T2;
```



指针的进阶

## 4. 类型的语义表示

### □ 集合类型

|    |    |     |
|----|----|-----|
| 大小 | 种类 | 基类型 |
|----|----|-----|

程序设计语言中设定一个集合的时候都是设定成一个有限的集合，用 $2^n$ 来确定

### □ 文件类型

|    |    |       |
|----|----|-------|
| 大小 | 种类 | 缓冲区类型 |
|----|----|-------|

大小指的是缓冲区的大小

