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

- 语言包括4个内容：词法、语法、语义、语用
- 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。
- 语义：关于结构上合法的程序的意义法则。

1.2 语义分析的必要性

- 一个语法正确的程序不能保证它是有意义的.
- 程序中容易出现各种语义错误：
  - 标识符未声明
  - 操作数的类型与操作符的类型不匹配
  - ……

```
1.  int x=10;
2.  Main( )
3.  { printf( “%d”,x+x );
4.    x( );
5.    f = x;
6.  }
7.  float f( )
8.  {int x=20,y;
9.   float x;
10.  printf( “%d”,x );
11. }
```

符合变量声明的语法、语义

符合函数调用的语法、不符合语义

符合赋值语句的语法、不符合语义

符合函数声明的语法、语义

符合变量声明的语法、语义

符合变量声明的语法、不符合语义

1.3 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。

1.4 语义分析的主要任务

- 根据声明部分建立符号表

符号表（symbol table）：是一种供编译器用于保存有关源程序的各种信息的数据结构。符号表的每个条目中包含与一个标识符相关的信息，这些信息全面地反映该名字的属性及它们在编译过程中的特征。

符号表的作用：

- 存储标识符的属性；
- 便于检查语义错误；
- 代码生成阶段作为地址分配的依据。

1.4 语义分析的主要任务

- 在整个程序范围内检查常见语义错误
  - 声明和使用相关的错误
  - 类型相关的语义错误

1.5 语义错误检查

- 声明和使用相关的语义错误
  - 常见的语义错误：
  - 每个使用性标识符是否都有声明?在同层内有无标识符被声明多次?
  - 标号是否有声明?有无重复声明和重复定位错误?有无非法转入错误?
  - 如何检查?
  - 每当遇到新声明的标识符,查符号表：
    - 如果当前有效的所有标识符中有相同名字的,则是重复声明错误;
    - 否则生成它的属性信息,保存到符号表中;
  - 每当遇到标识符的使用,查符号表
    - 如果没有找到,说明该标识符没有声明;
    - 如果找到,则可获取该标识符的属性,进行进一步分析;

1.5 语义错误检查

- 类型相关的语义错误
  - 各种条件表达式的类型是不是boolean型?
  - 运算符的分量的类型是否相容?
  - 赋值语句的左右部的类型是否相容?
  - 形参和实参的类型是否相容?
  - 下标表达式的类型是否为所允许的类型?

1.5 语义错误检查

- 函数说明中的函数类型和返回值的类型是否一致?
- V[E]中的V是不是变量,而且是不是数组类型?
- V.id中的V是不是变量,而且是不是结构体类型? id是不是该记录类型中的成员?
- V↑(*V)中的V是不是指针或文件变量?
- y+f(....)中的f是不是函数名?形参个数和实参个数是否一致?
- p(....)语句中的p是不是过程名?形参个数和实参个数是否一致?
- 子界类型中的下界和上界类型是否相容?下界是否小于等于上界?

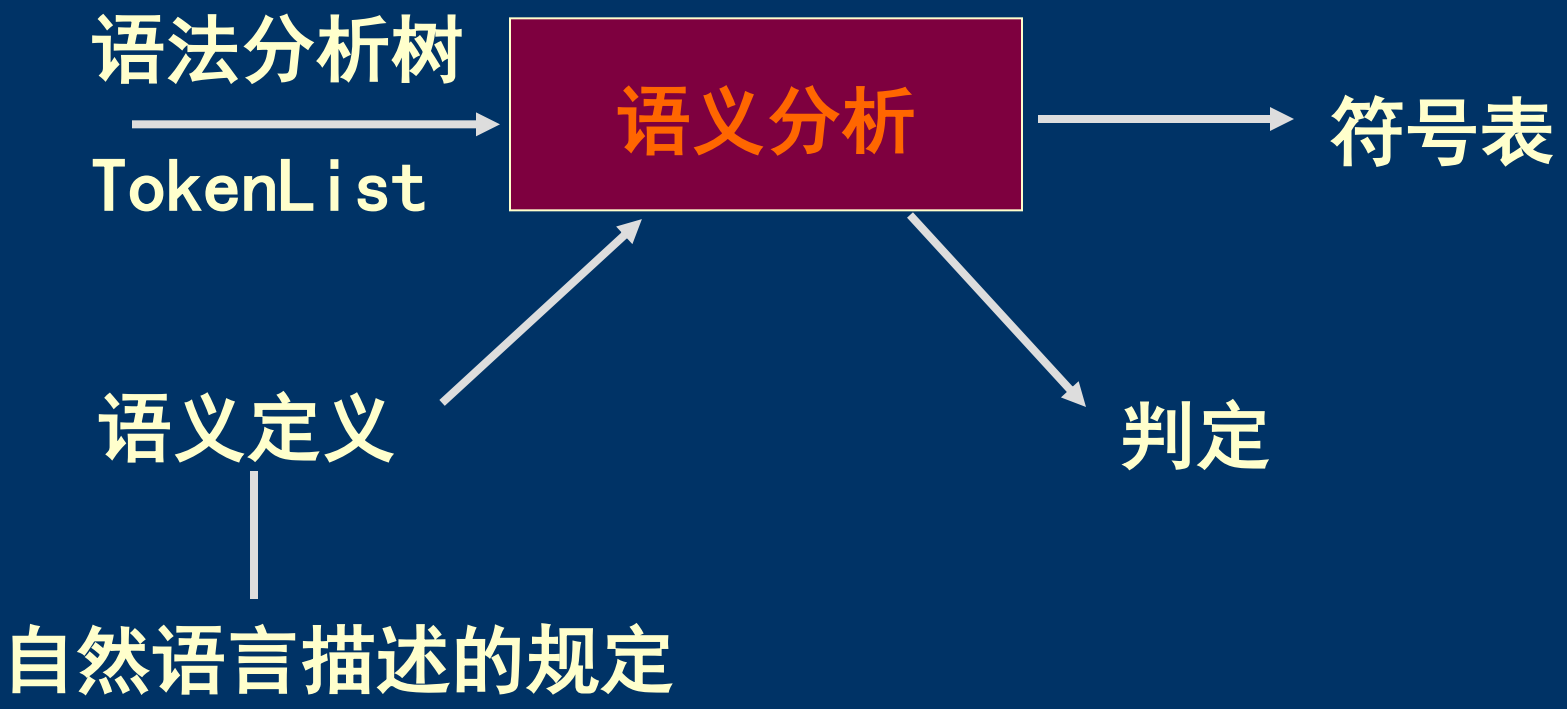
1.6 语义分析的实现

方式一：不作为独立的一遍

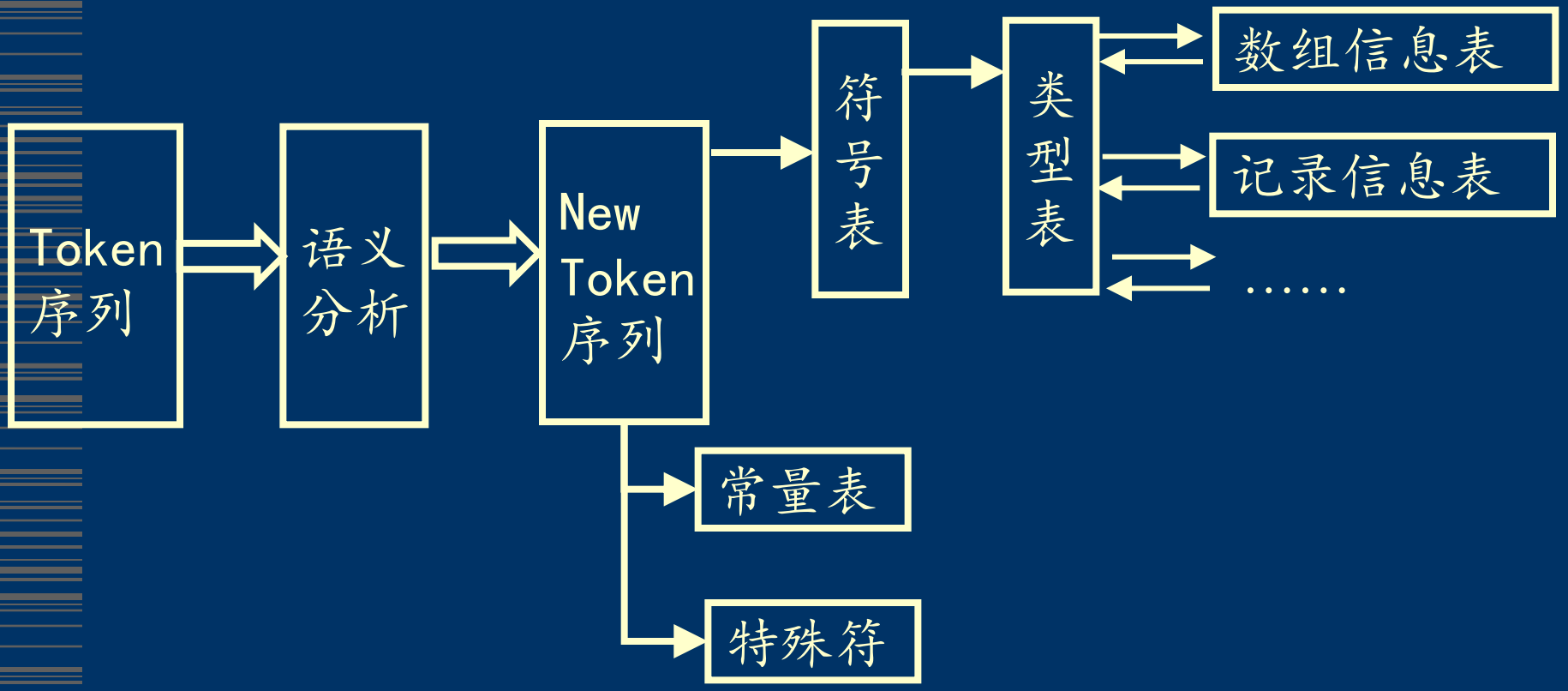
① 类型语义错误检查：可以安排在中间代码生成时进行

② 一般的语义检查：与语法分析相结合

方式二 独立一遍的语义分析的功能图示



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	Type	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	Type

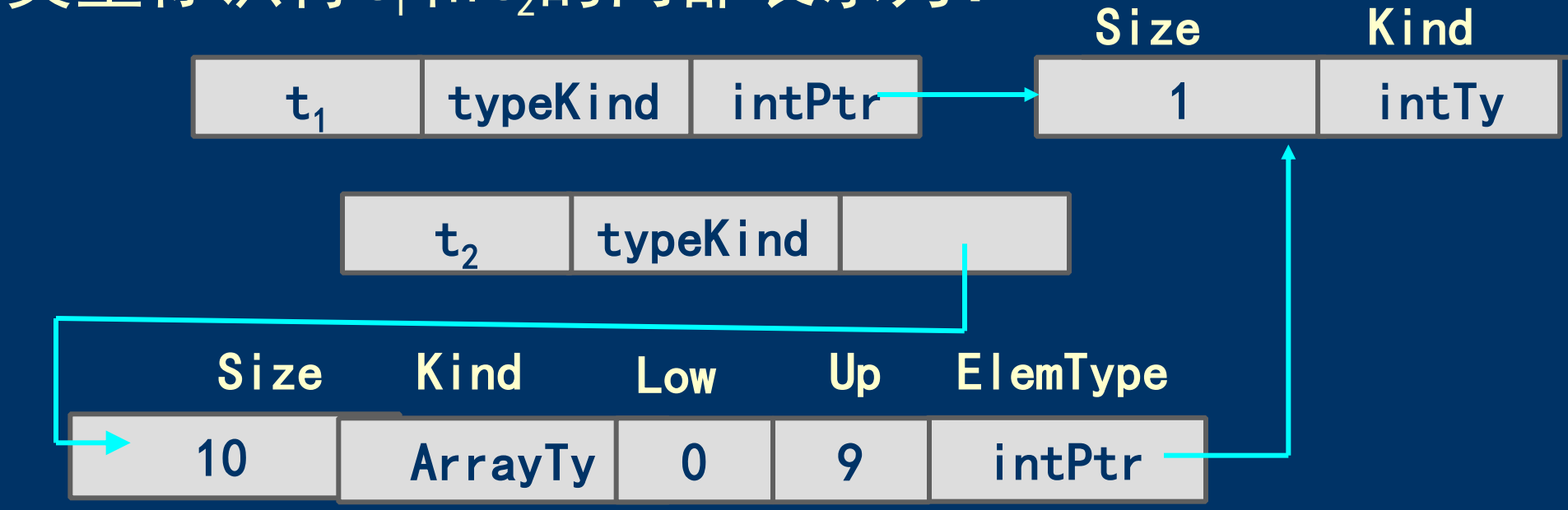
- Name是类型标识符的名字；
- Kind = typeKind, 表示标识符是类型标识符；
- Type = TypePtr, 指向类型标识符指代的类型的内部表示。

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	Type	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 ）：

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

3.4 标识符的语义表示

- 过程/函数标识符的内部表示

Name	Type	Kind	Level	off	Param	Class	Code	Size	Forward

- Type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

Name	Type	Kind	Level	off	Param	Class	Code	Size	Forward

- 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

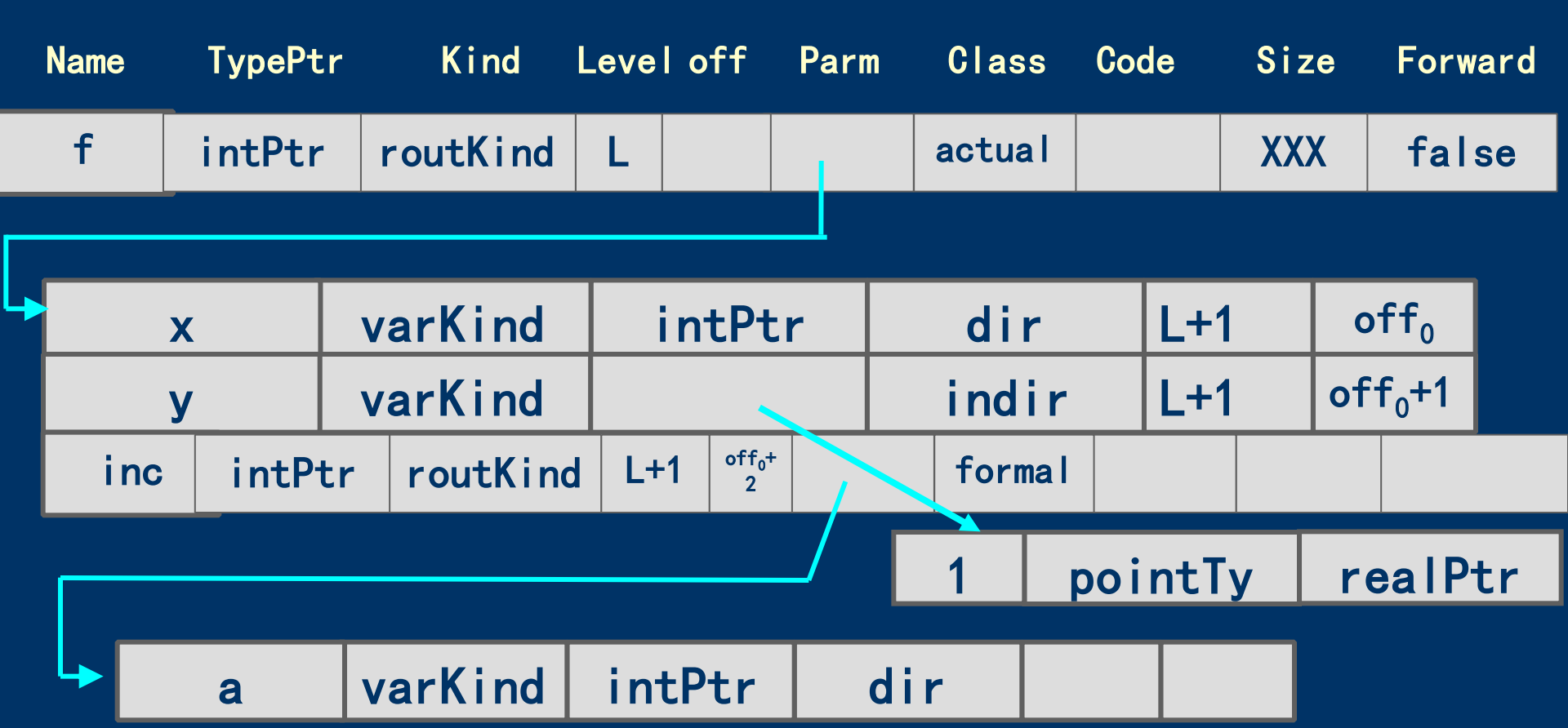
- 过程/函数标识符的内部表示

Name	Type	Kind	Level	off	Param	Class	Code	Size	Forward

- Size只对实在过/函有效，表示过/函的过程活动记录所占内存区的大小，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：Pascal语言的函标识符定义：

```
function f(x: integer; var y: real;
            function inc(a: integer): integer): integer; "头"
begin
  ......f的函数体部分
end;
```



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

- 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，记录
- ❖指针类型
- 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

	Size	Kind
intPtr→	IntSize	intTy
boolPtr→	BoolSize	boolTy
charPtr→	CharSize	charTy
realPtr→	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';

leeterPtr → | 1 | subrangeTy | 'a' | 'z' | charPtr |

4. 类型的语义表示

- 枚举类型

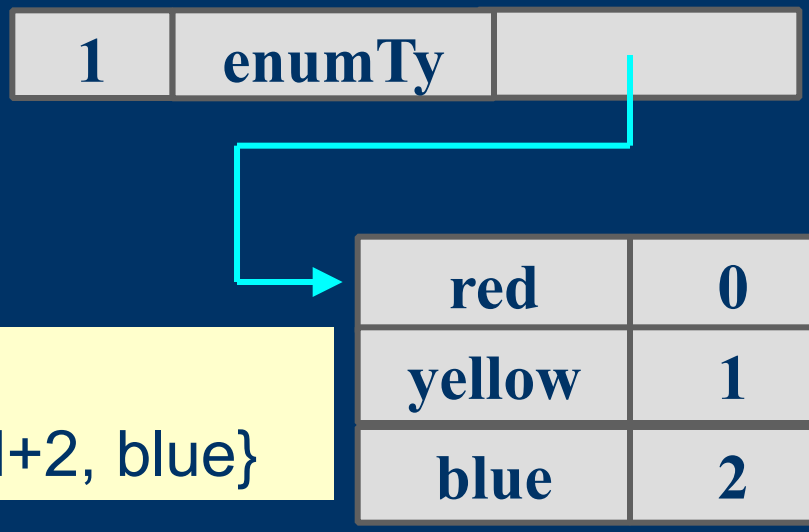
- size表示枚举类型所占空间的大小;
- Kind = enumTy;
- ElemList是指向枚举常量表表头的指针.

- 枚举常量表内部表示:

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

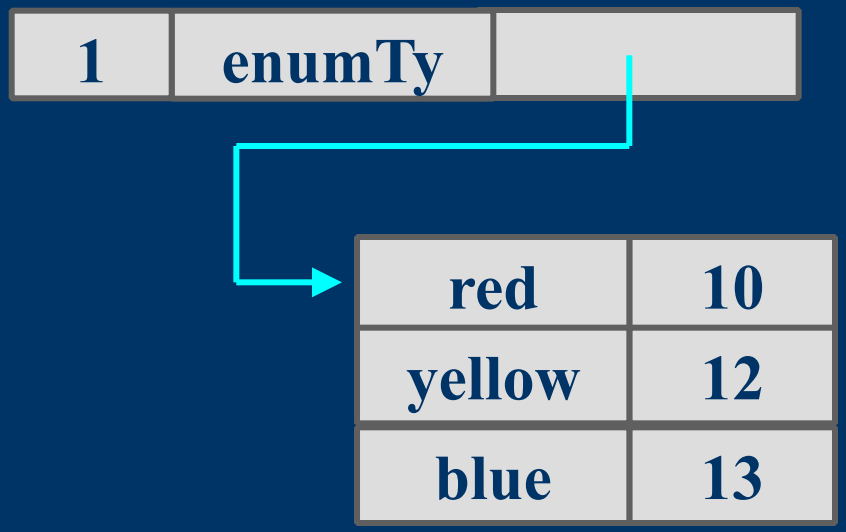
例：c语言的枚举类型

enum color {red, yellow, blue}



例：c语言的枚举类型

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



4. 类型的语义表示

- 数组类型

Size	Kind	Low	Up	ElemType
ArraySize	ArrayTy			TypePtr

- Size: Size = (Up-Low+1)*sizeof(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;
- 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

Size	Kind	RecBody

  - Kind=structTy 表示结构体类型
  - Kind=unionTy表示联合体类型

- RecBody的内部表示如下

Name	Type	Off	link

Name表示域名

Type指向域的类型

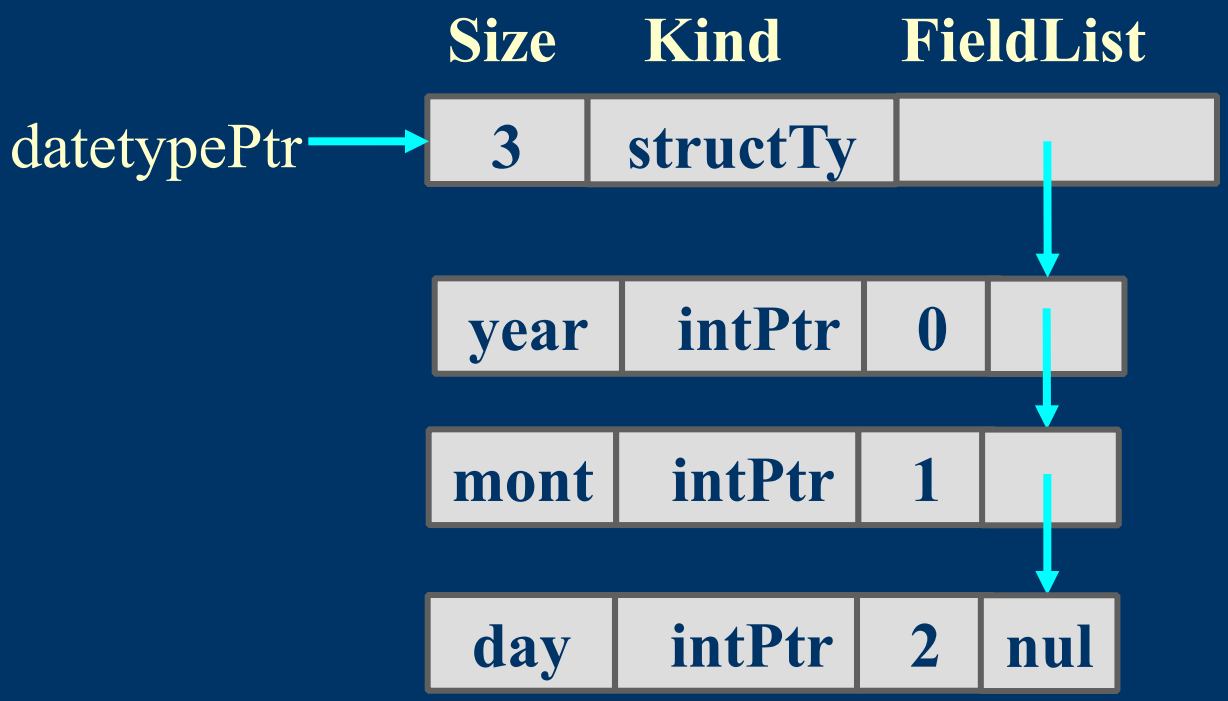
Off表示纪录域相对于结构体类型分配的内存块起始地址的偏移量，对于联合类型而言，所有的域名标识符的起始偏移都是相同的，所以可以省略；

link指向下一个域名标识符内部表示的指针

例：

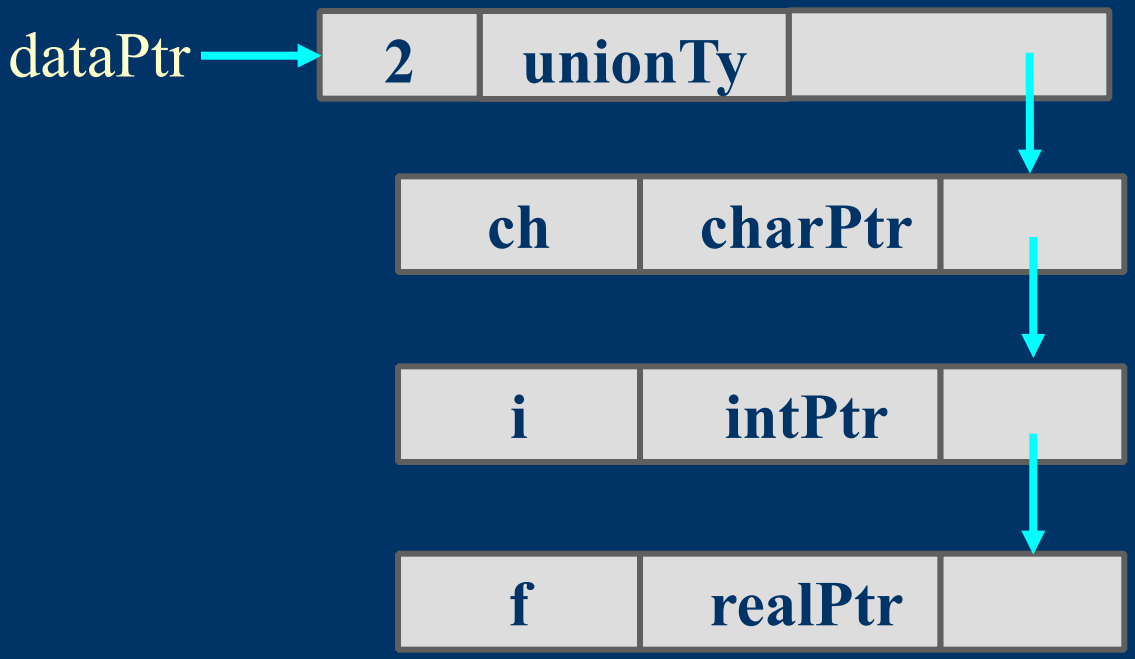
typedef struct Date

{int year, month, day;} datetype;



例：c语言的联合体

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



4. 类型的语义表示

- 指针类型

❖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语言的针举类型

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