第6章 语义分析

- 6.1 语义分析概述
- 6.2 符号表的功能和结构
- 6.3 符号表的组织和管理
- 6.4 程序设计语言符号表的实例

6.1 语义分析概述

- 6.1.1 什么是语义?
- 6.1.2 语义分析的任务
- 6.1.3 语义分析的一般过程

6.1.1 什么是语义？

- 词法：关于单词构成的规则

  例:标识符是由字母开头后跟若干字母数字的字符串

- 语法：关于语法结构构成的规则

  例:赋值语句是按照 ID = E 构成的

- 语义：关于语法结构含义的约定

  例1. 赋值语句“x = y+1”的语义是“用表达式y+1计算得到的结果值更新变量x的值”

  例2. 变量声明“int x,y”的(不完全)语义是标识符x,y分别代表的是变量，并且该变量的类型是整型

6.1.1 什么是语义？

- 静态(static-time)语义: 指那些不需运行代码即可确定的语义
  - 由变量声明“int x”可静态确定: x 是变量标识符，它的类型是整型，x的作用域(有效范围)等。不能确定: x的值
  - 由表达式“x+y*10”可静态确定:表达式计算结果的类型，但不能确定表达式计算结果的值

6.1.1 什么是语义？

- 动态(run-time)语义: 指需要运行代码才可确定的语义
  - 数组下标变量“A[i]” 是否越界?
  - 表达式“1/y”是否除零溢出?
  - 语句“if (i>j) x = 1 else x = 0”执行的是哪段代码?
  - “ *p = 3 ”中是否引用了空指针

6.1.2 语义分析的任务

- 检查语义错误
- 一般性的语义检查
  - 下标变量V[E]中的V是不是变量,而且是数组类型?
  - 结构体变量V.id中的V是不是变量,而且是结构体类型? id是不是该结构体类型中的域名?
  - 指针变量*V中的V是不是指针或文件变量?
  - 表达式y+f(....)中的f是不是函数名?形参个数和实参个数是否一致?
  - 每个标识符是否都有声明?有无标识符被重复声明?

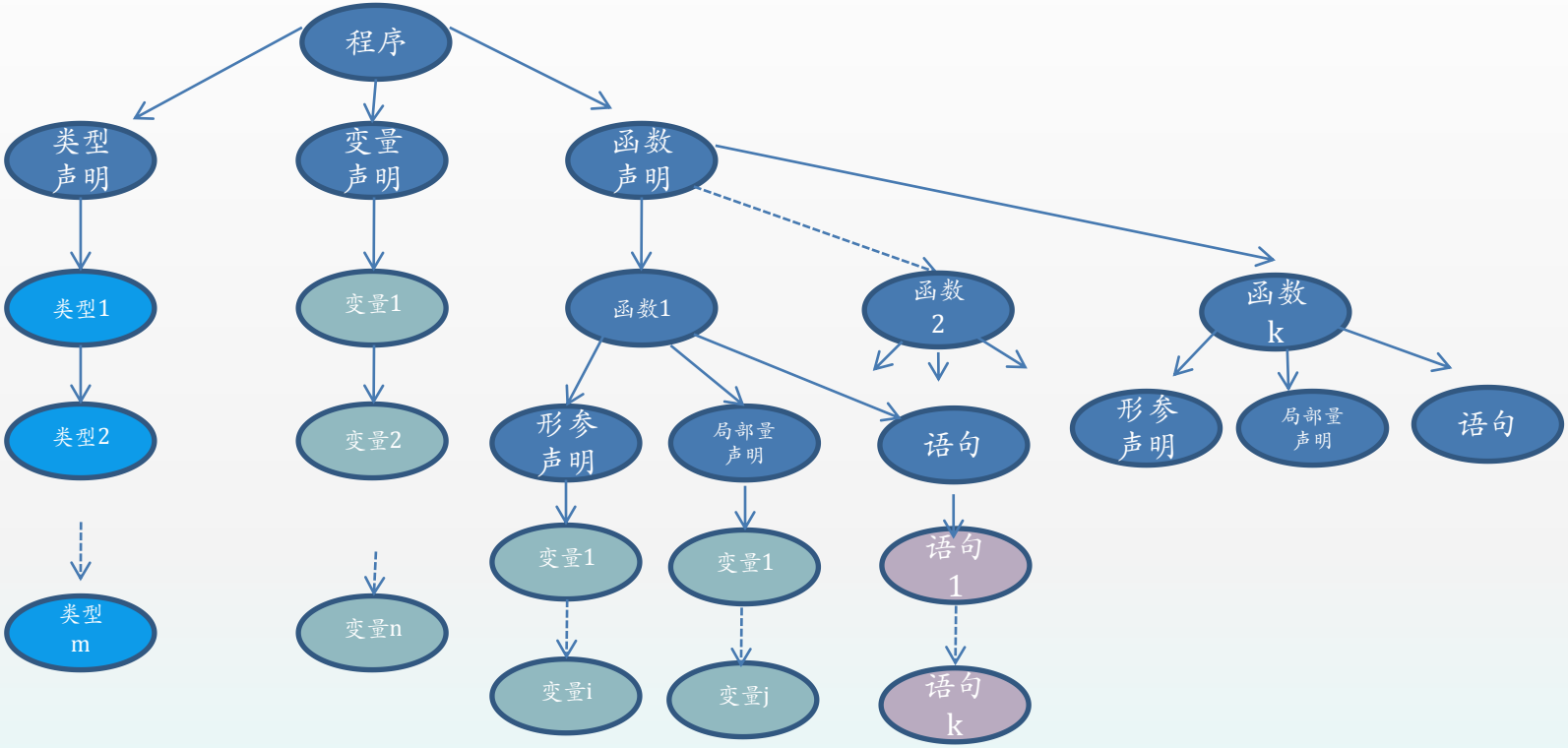
6.1.2 语义分析的任务

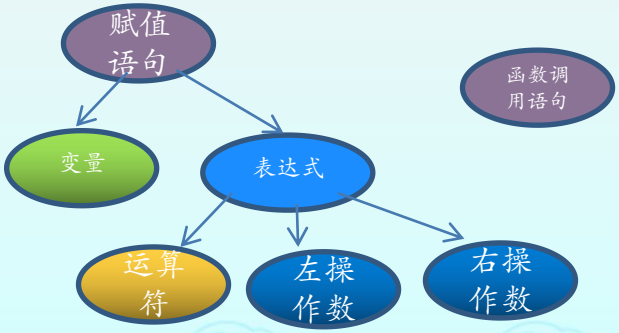
- 类型检查
  - 各种条件表达式的类型是不是boolean型?
  - 运算符分量的类型是否相容?
  - 赋值语句左右部的类型是否相容?
  - 形参和实参的类型是否相容?
  - 下标表达式的类型是否为所允许的类型?
  - 函数返回值的类型与函数声明中的返回值类型是否一致?

6.1.2 语义分析的任务

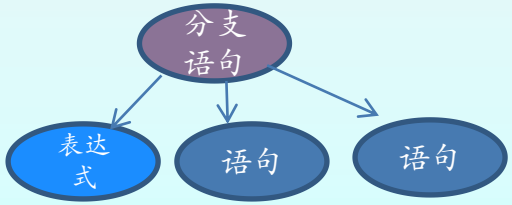
- 唯一性检查
  - 同一switch语句的分支常量不能有重复
  - 枚举类型的枚举常量不能重复
  - 结构类型的域名不能重复
  - 同一函数参数的名字不能重复

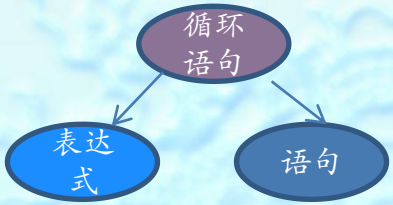
6.1.3 语义分析的一般过程





函数调用语句





6.2 符号表的功能和结构

- 符号表：记录标识符语义属性的数据结构
- 符号表的功能:
  - 收集符号的属性
  - 上下文语义的合法性检查的依据
  - 作为目标代码生成阶段地址分配的依据
- 语义属性:
  - 名字、类型、存储类别、符号的作用域及可视性、符号变量的存储分配信息、其他(结构体型的成员信息、函数及过程的形参)等

6.2 符号表的功能和结构

- 6.2.1 标识符的内部表示
- 6.2.2 类型的内部表示
- 6.2.3 值的内部表示

6.2.1 标识符的内部表示

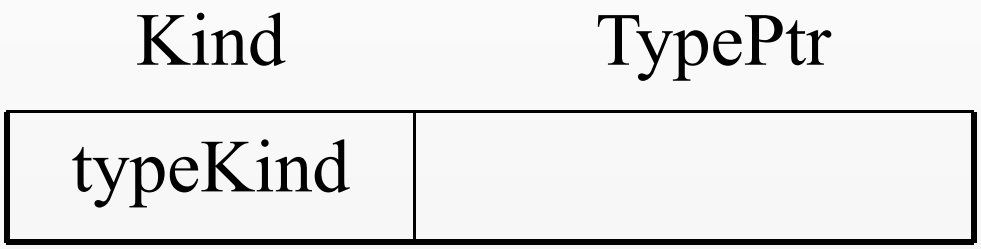
- 程序设计语言中有很多不同种类的标识符:
  - 变量标识符
  - 常量标识符
  - 类型标识符
  - 过程/函数标识符
  - 域名标识符
- 每类标识符有不同的属性,因此可以:
  - 每类标识符一张表，整个程序多张表
  - 不同种类标识符采用不同表项,整个程序一张表

常量标识符的内部表示

Kind	TypePtr	Value
constKind		

- **Kind**
  - 标识符的种类，所有常量的 **kind = constKind;**
- **TypePtr**
  - 指向常量类型内部表示的指针;
- **Value**
  - 常量值的内部表示;
- 例：**#define pai 3.14**
- **#define count 100**

类型标识符的内部表示



- **Kind**
  - 标识符的种类，所有类型标识符的 **kind = typeKind;**
- **TypePtr**
  - 指向类型内部表示的指针;
- 例： **typedef int t1;**
- **typedef int t2[10];**
- **t1 a; //t1是类型标识符，a是变量标识符**
- **t2 b; //t2是类型标识符，b是变量标识符**

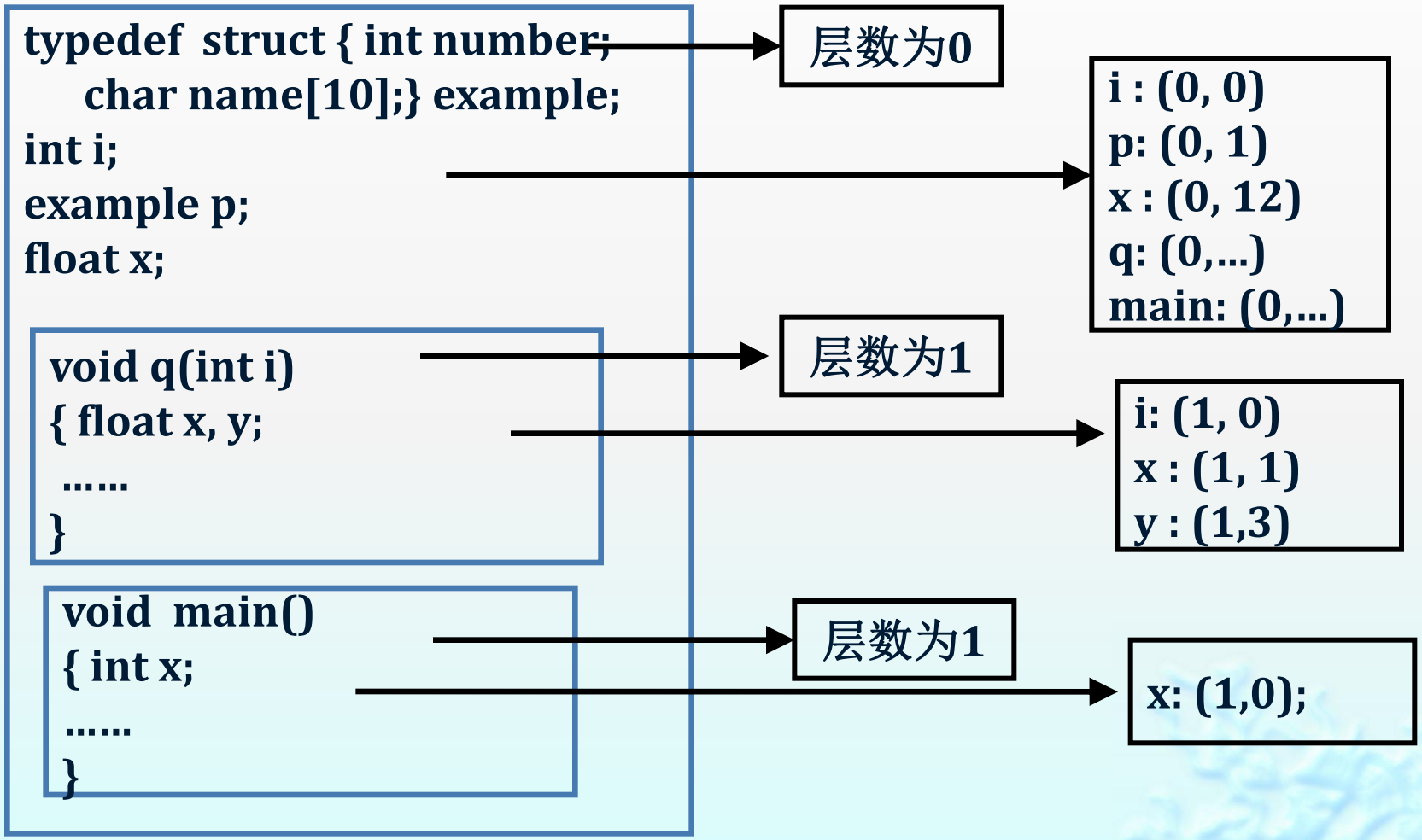
变量标识符的内部表示

Kind	TypePtr	Access	Level	Offset	Value
varKind					

- **Kind**
  - 标识符的种类, 所有变量标识符的**Kind = varKind;**
- **TypePtr**
  - 指向变量类型的内部表示;
- **Access: (dir, indir);**
- **Level:** 层数
- **Offset:** 偏移量
- **Value:** 初始化时赋给变量的值的内部表示
- 例：**int x = 10;**
- **float y;**
- **float* z;**

变量的层数和偏移

- 层数(level)
  - 某些程序语言，过程/函数的定义可以嵌套
  - 通常规定主程序的层数为0;
- 偏移量(offset)
  - 执行过程/函数的调用时，需要为其中的变量分配空间;
  - 偏移量指的是变量针对其所在过程/函数的空间的首地址的偏移量;


typedef struct { int number;
char name[10];} example;
int i;
example p;
float x;
层数为0
i : (0, 0)
p: (0, 1)
x : (0, 12)
q: (0,...)
main: (0,...)
void q(int i)
{ float x, y;
.....
}
层数为1
i: (1, 0)
x : (1, 1)
y : (1,3)
void main()
{ int x;
.....
}
层数为1
x: (1,0);

函数标识符的内部表示

Kind	TypePtr	Class	Level	Off	Param	Code	Size	Forward
routKind								

- Kind: 标识符的种类，此处为 routKind;
- TypePtr: 指向函数返回值类型的内部表示，过程情形为NULL;
- Class: 当前函数是实在函数时(有自己函数体)取actual;
  当前函数是形参函数时(作为其他函数的参数)取formal;
- Level: 函数标识符被定义的层数;
- Off: 当前函数是形参函数时，为其分配的地址偏移;
- Param: 指向当前函数的形式参数列表的指针;
- Code: 指向函数对应的目标代码的起始地址;
- Size: 目标代码的大小;
- Forward: 该声明为函数原型时取值 true，否则取值为false;
- 例： int f(int x, float*y, int inc()) { … }
  void p(int x, float*y) { …. }

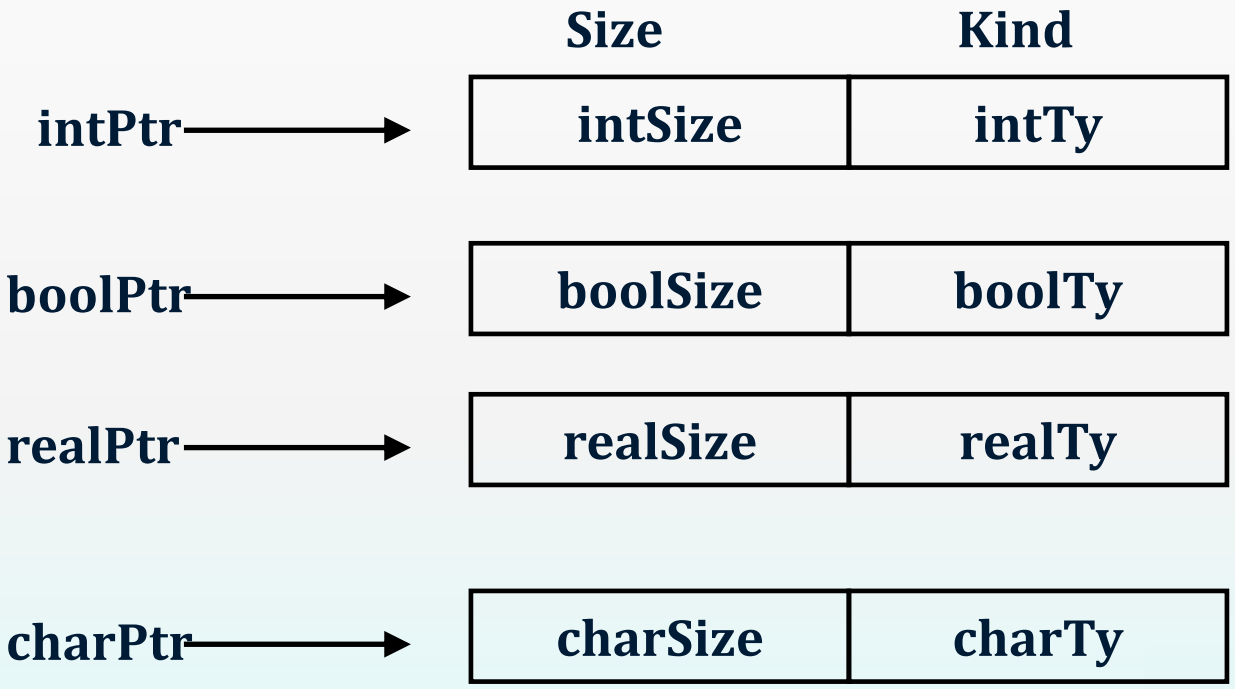
6.2.2 类型的内部表示

- 程序设计语言为完成许多事务的处理，常常要定义很多不同类型的数据，常见的类型有:
  - 整型(int, integer)
  - 实型(float, real)
  - 布尔型(bool)
  - 字符型(char)

  基本类型
  - 数组(int A[10], array A[1..10])
  - 结构体(struct, record)
  - 联合(union, 变体记录)
  - 枚举(enum)
  - 指针(int*)

  构造类型

基本类型的内部表示

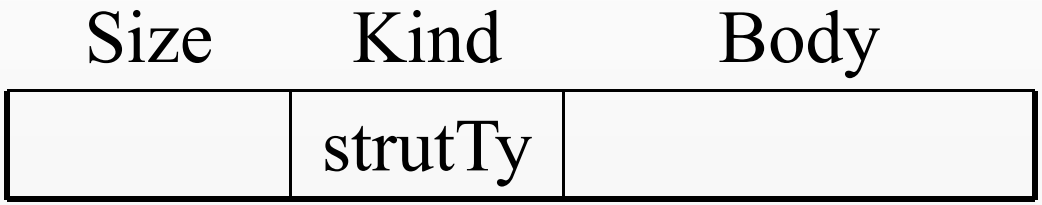

Size
Kind
intPtr
intSize
intTy
boolPtr
boolSize
boolTy
realPtr
realSize
realTy
charPtr
charSize
charTy

数组类型的内部表示

Size	Kind	Low	Up	ElemTy
	arrayTy			

- ElemTy: 指向数组成分类型的指针;
- size = sizeof(ElemTy)× (Up-Low+1)
- 例：
- int a[7]; (7, arrayTy, 0, 6, intptr)
- float a[10][5]; (100, arrayTy, 0,9, aptr)
- aptr:(10, arrayTy, 0,4, realptr)

结构类型的内部表示



Size: 所有域的类型的**size的总和**;
Body:是域名标识符的内部表示链;
例：

```
typedef struct { int  i;   char name[10]; float  x; }example;
example  wang;
example  factory_a[100];
```

联合类型的内部表示

Body: 指向联合体中域定义链表
Size: 所有域的类型的size中最大值;

例：

```
typedef union{ int  i;   char name[10]; float  x; } test;
test  lee;
test  factory_b[100];
```

枚举类型的内部表示

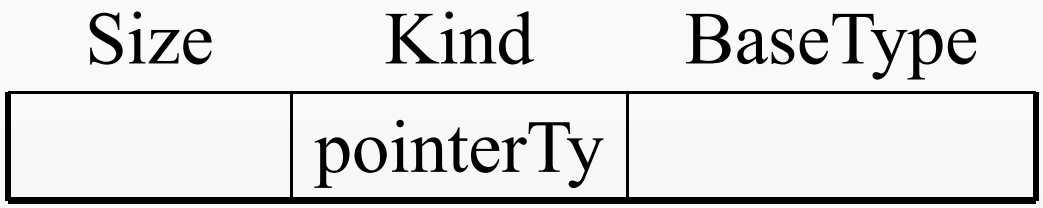
Size	Kind	EList
	enumTy	

EList: 枚举常量链表;
Size:枚举类型值的空间大小(通常为1个单元);

例：

```
enum color {red, yellow, blue};
enum day {car , bus = 0, taxi, ship = 5, plane};
```

指针类型的内部表示



BaseType: 指向指针的基类型;
Size:指针的空间大小(通常为1个单元);

例:

```
int* p1;
float* p2;
int *p3[10];   //指针数组
int (*p4)[10];   //数组指针
int* (p5[10]);   //指针数组
int**   p6;         //指针的指针
```

6.2.3 值的内部表示

- 可表示的值
  - int
  - float
  - false, true(通常对应0和1)
  - char: ASCII码值
  - 枚举类型: 每个枚举常量对应一个整数
- 结构值
  - 数组
  - 结构
  - 联合
  - 指针

6.3 符号表的组织和管理

- 一个编译程序，从词法分析、语义分析到代码生成的整个过程中，符号表是连贯上下文进行语义检查、语义处理、代码生成和存储分配的主要依据，因此符号表的组织直接关系到这些功能的时空效率.
- 表的总体组织
  - 多表：节省空间,总体管理复杂度高
  - 单表：便于统一管理，但可能有空间的浪费
- 表项的组织
  - 线性表、二分法、散列表

6.3 符号表的组织和管理

- 标识符的作用域：程序中的每个标识符都有自己的作用域.在其作用域范围内,标识符是可见的或有效的.
- 符号表的局部化处理：

  在构建和查找符号表时体现出标识符的作用域.
- 局部化区：

  每个允许有声明的程序段称为一个局部化区.

一般为一个函数或一个分程序语句

```
int i, j;
void test ( int j )
{  real x;
    ... j ...
  { int x; ....}
  ......
}
void main()
{ char i;
   ......
  {int i; ......}
   ......
}
```

6.3 符号表的组织和管理

- **局部化处理的思想：**
- 进入局部化区，作标记；
- 遇见标识符声明
  - 查找符号表,检查标识符是否已经被声明过；
  - 如果是，则重复声明错；
  - 如果不是，则建立标识符的内部表示，将其放入符号表；
- 遇见标识符使用
  - 查找符号表,检查标识符是否有声明；
  - 如果是，则取出标识符的属性进行语义分析；
  - 如果不是，则未声明错；
- 退出局部化单位,删除该局部化单位里声明的所有标识符；

6.4 程序设计语言符号表的实例

- **局部化符号表:**
- 每个局部化单位一张表
- 由于局部化单位可能嵌套，所以用一个Scope栈来管理各个局部表
- 每当进入新一层局部化单位，将其符号表的首地址压入Scope栈
- 每当退出一层局部化区时，从Scope栈栈顶将该层符号表首地址弹出
- 访问时，从Scope栈栈顶所记录的符号表的开始查找，若未找到，则再依次从次栈顶所记录的符号表开始查找，直到找到或所有符号表中都找不到为止

```
int i, j;
void test ( int j )
{  real x;
     ... j ...
  { int x; ....}
   ......
}
void main()
{ char i;
   ......
  { int i; ...... }
   ......
}
```

6.4 程序设计语言符号表的实例

- 全局化符号表:
- 每个局部化单位对应一个唯一的局部化编号num，符号表的表项为(num, id, attributes);
- 初始化: CurrentNum = 0;
- 每当进入一个新的局部化单位时, CurrentNum ++;
- 遇到定义性标识符(声明),检查所有对应CurrentNum的表项,判定是否有重复定义,如果没有则将其属性及其num登记到符号表中;
- 遇到使用性标识符(表达式或语句),查符号表,(查所有num<=CurrentNum的表项，且按照num, CurrentNum-1, CurrentNum-2,…, 0的顺序查找)如果都没有,则为标识识符未声明错;否则,最先查到的表项为标识符对应的属性;
- 结束一个局部化单位时,“删除”所有CurrentNum对应的表项，CurrentNum --;

全局化符号表—驻留法

```
int i, j;
void test ( int j )
{  real x;
    … j …
  { int x; …x…}
    …x…
  { bool x; …x…}
    …x…
}
void main()
{ char i;
    ……
  { int i; …… }
    ……
}
```

(0, i, 0, 0, intptr, ……)
(0, j, 0, 1, intptr, ……)
(0, test, 0, , voidptr, ……)
(1, j, 1, 0, intptr, ……)
(1, x, 1, 1, realptr, ……)
(2, x, 1, 3, intptr, ……)
(#, ……)
(2, x, 1, 3, boolptr, ……)
(#, ……)
(#, ……)
(0, main, 0, , voidptr, …)
(1, i, 1, 0, charptr, ……)
(2, i, 1, 1, intptr, ……)
(#, ……)
(#, ……)
(#, ……NULL)

全局化符号表—外拉链散列表

```
int i, j;
void test ( int j )
{  real x;
    … j …
  { int x; …x…}
    …x…
  { bool x; …x…}
    …x…
}
void main()
{ char i;
   ……
  { int i; …… }
   ……
}
```

- 以标识符名字为主键
- 将事先设计好的散列函数作用到标识符名字上，计算得到标识符的符号表地址
- 当出现重名标识符时，采用外拉链的方式化解冲突
- 由于标识符的最新定义总是出现在链表的头部，所以总会被最先查到
- 当分程序无效，分程序中定义的标识符也会从表中直接删除掉

作业

1. 写出下列类型的内部表示.

typedef struct {char name[10]; int age; }person;

typedef person List[10];

typedef union { int data; real length; char name[4];}

作业

2. 写出当分析下列程序时，创建符号表的过程. (局部符号表和全局符号表(删除法和驻留法均可))

```
typedef struct {char name[30]; int age;} Student;
Student Lee;
int i ;
int GetAge(Student S)   { return S.age;}
void SetAge( int i)  {Lee.age=i; }
void main() { student  Lee; SetAge(10);
                {int i = 20;  Lee.age = i;}
              }
```