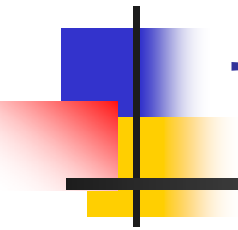


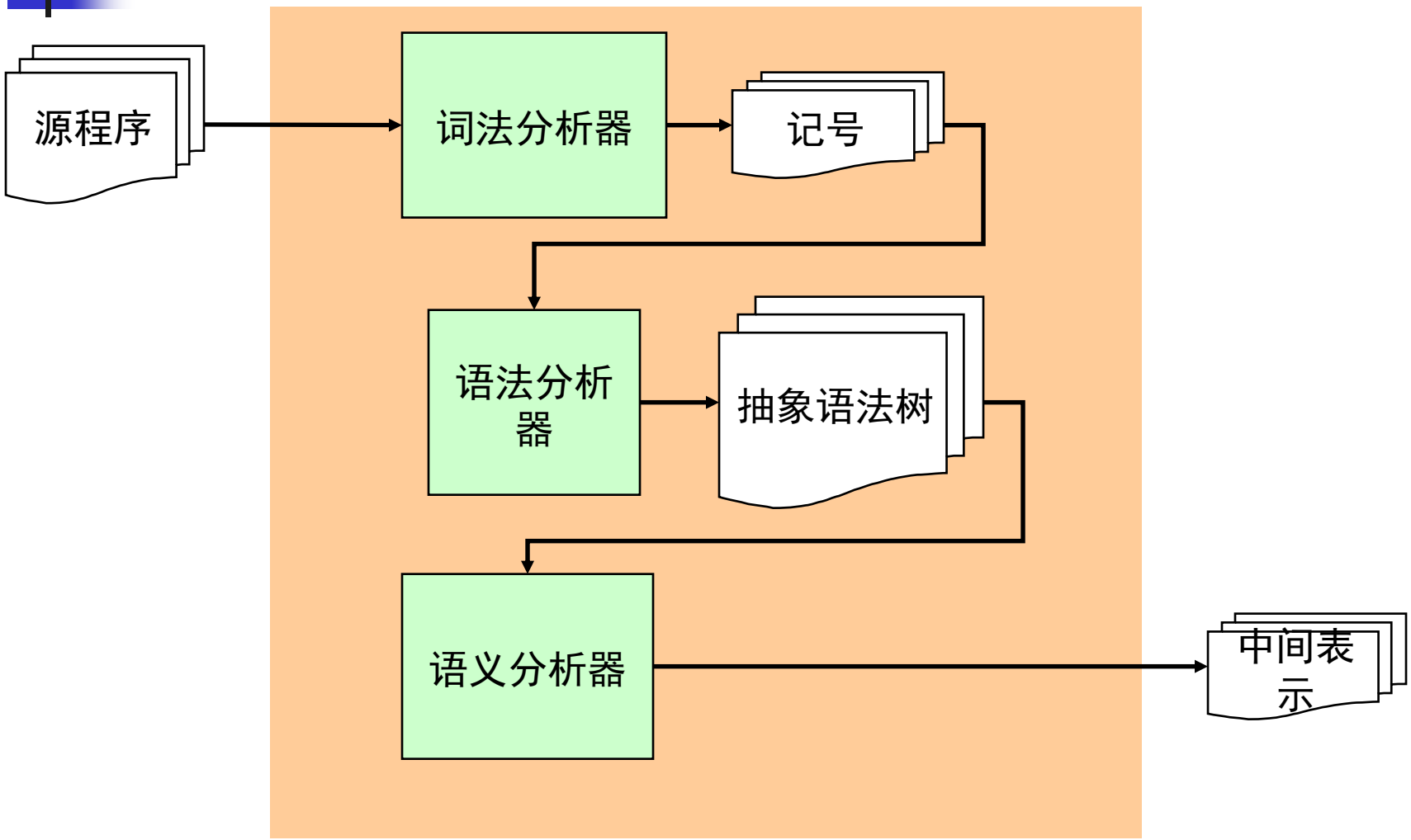
语义分析：符号表

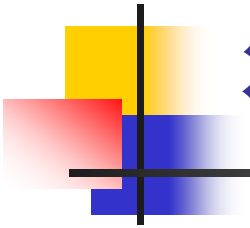
编译原理

华保健

bjhua@ustc.edu.cn

前端





符号表

- 用来存储程序中的变量相关信息
  - 类型
  - 作用域
  - 访问控制信息
  - 。 。 。
- 必须非常高效
  - 程序中的变量规模会很大

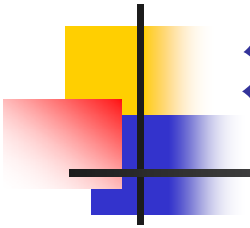
符号表的接口

```
#ifndef TABLE_H
#define TABLE_H

typedef … Table_t; // 数据结构

// 新建一个符号表
Table_t Table_new ();
// 符号表的插入
void Table_enter (Table_t, Key_t, Value_t);
// 符号表的查找
Value_t Table_lookup (Table_t, Key_t);

#endif
```

符号表的典型数据结构

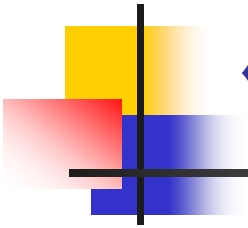
```
// 符号表是典型的字典结构：
   symbolTable: key ->
   value
// 一种简单的数据结构的定义（概
   念上的）：
typedef char *key;
typedef struct value{
  Type_t type;
  Scope_t scope;
  … // 必要的其他字段
} value;
```

变量\映射	type	scope	…
x	INT	0	…
y	BOOL	1	…
…	…	…	…



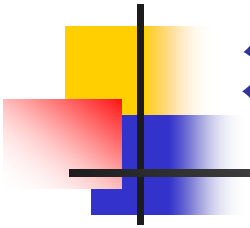
符号表的高效实现

- 为了高效，可以使用哈希表等数据结构来实现符号表
  - 查找是O(1)时间
- 为了节约空间，也可以使用红黑树等平衡树
  - 查找是O(lg N)时间



作用域

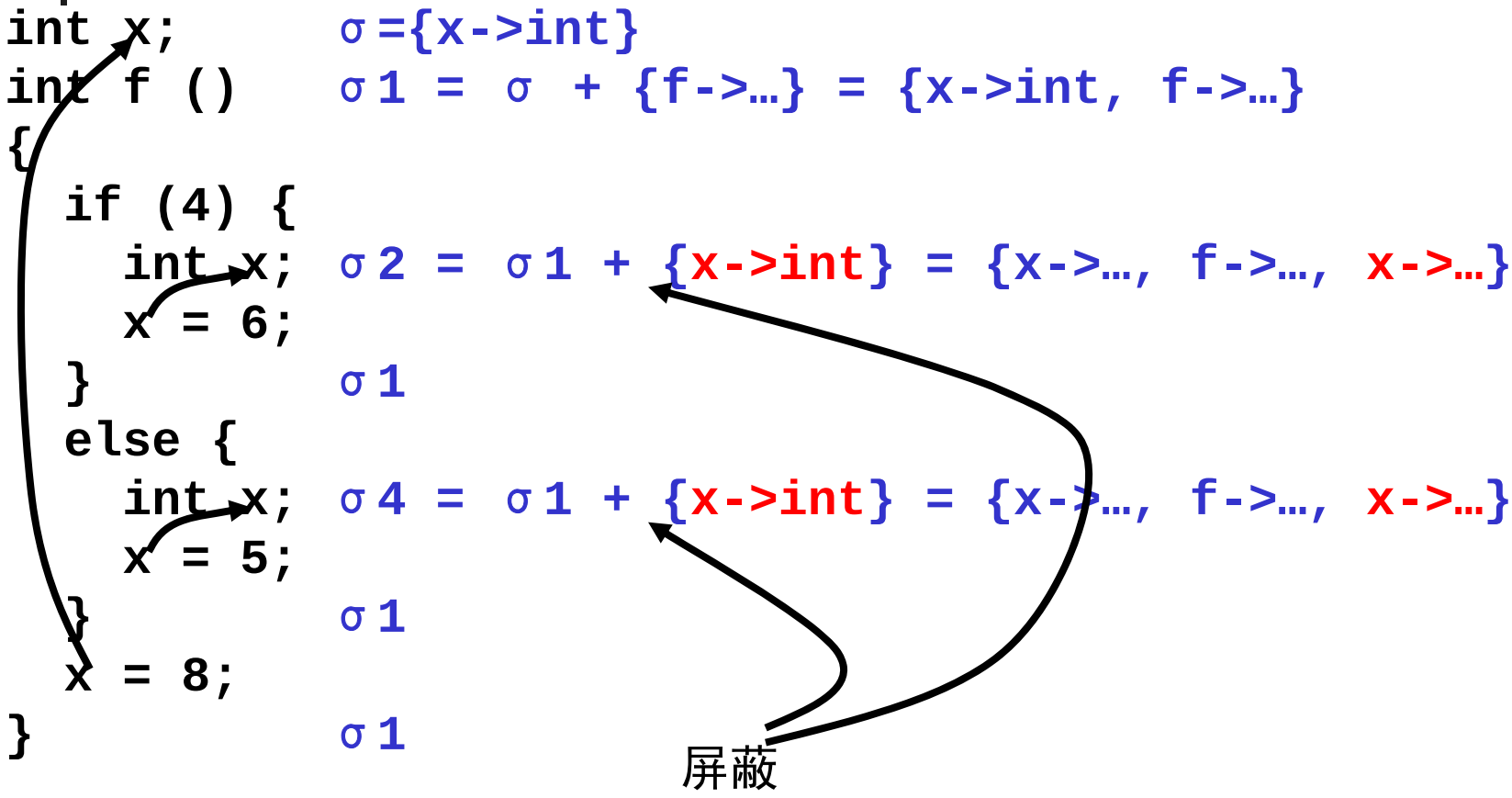
```
int x;
int f ()
{
  if (4) {
    int x;
    x = 6;
  }
  else {
    int x;
    x = 5;
  }
  x = 8;
}
```

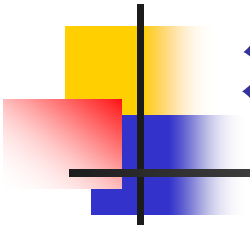


符号表处理作用域的方法

- 方法#1：一张表的方法
  - 进入作用域时，插入元素
  - 退出作用域时，删除元素

示例


int x;
int f ()
{
if (4) {
int x;
x = 6;
}
else {
int x;
x = 5;
}
x = 8;
}
σ={x->int}
σ1 = σ + {f->…} = {x->int, f->…}
σ2 = σ1 + {x->int} = {x->…, f->…, x->…}
σ1
σ4 = σ1 + {x->int} = {x->…, f->…, x->…}
σ1
σ1
屏蔽



符号表处理作用域的方法

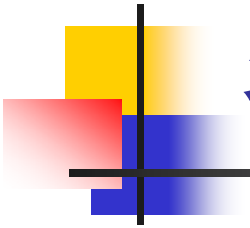
- 方法#1：一张表的方法
  - 进入作用域时，插入元素
  - 退出作用域时，删除元素
- 方法#2：采用符号表构成的栈
  - 进入作用域时，插入新的符号表
  - 退出作用域时，删除栈顶符号表

示例

```
int x;
int f ()
{
  if (4) {
    int x;
    x = 6;
  }
  else {
    int x;
    x = 5;
  }
  x = 8;
}
```

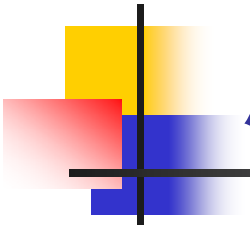
变量\映射	type	scope
x	INT	1

变量\映射	type	scope
x	INT	0
f	…	0



名字空间

```
struct list
{
  int x;
  struct list *list;
} *list;

void walk (struct list *list)
{
  list:
    printf (“%d\n”, list->x);
    if (list = list->list)
      goto list;
}
```



用符号表处理名字空间

- 每个名字空间用一个表来处理
- 以C语言为例
  - 有不同的名字空间:
    - 变量
    - 标签
    - 标号
    - 。。。
  - 可以每类定义一张符号表