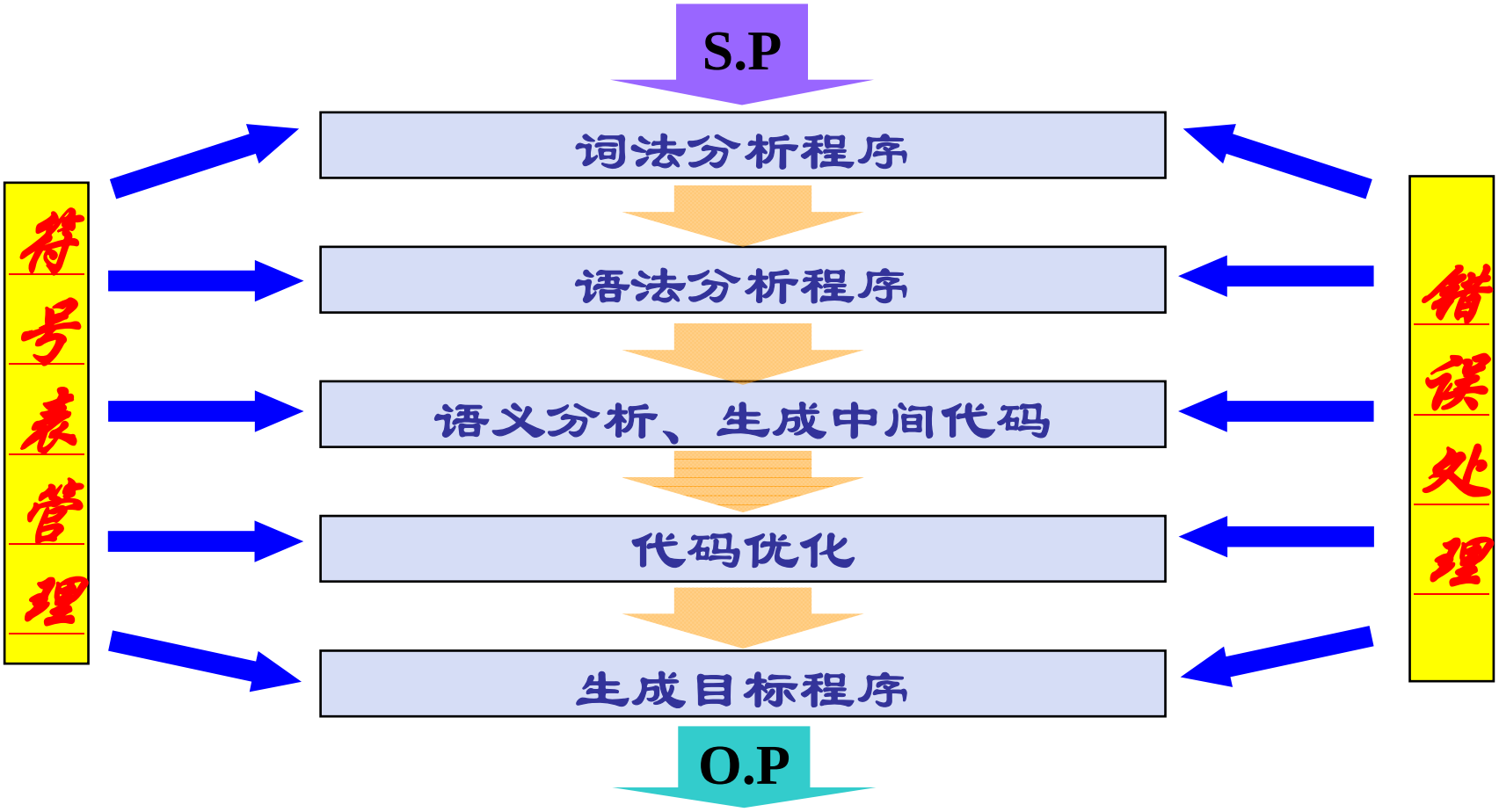

2011年5月30日

第6章　符号表管理和错误处理

教学目标

1. 明确符号表的作用、内容、组织
2. 明确错误处理的两种方法：错误校正和局部化处理

2011年5月30日

教学内容

- 6.1 符号表管理
- 6.2 错误处理

2011年5月30日

6.1 符号表管理

编译程序中使用最多的数据结构是表

源程序中的各种信息，以便查询或修改

在这些表中，尤以**符号表**最为重要

- 生存期最长
- 使用最为频繁

6.1.1 符号表的作用和内容

★ **作用：**

(1) 收集符号的各种信息

(2) 语义检查的依据

(3) 目标代码生成阶段地址分配的依据

☆ **内容：** 名字栏＋信息栏

6.1.2 符号表的组织

★ **操作：**

(1) 向表中填入一个新标识符。

(2) 对于给定一个标识符：

① 查找是否在表中；

② 访问它在表中的相关信息；

③ 在表中填写或更新它的某些信息。

(3) 更新或删除一个或一组无用的项。

6.1.2 符号表的组织

☆ **符号表的总体组织：**

(1) 多张 (同类符号归一)

(2) 一张 (不同类符号归一)

(3) 前两种的折中 (种类相似原则)

★ **符号表项的组织：**

(1) 线性组织 (按扫描的先后顺序排序)

(2) 排序组织 (按字符代码值的大小排序)

(3) 散列组织：效率高，为多数编译程序采用

一个符号在散列表中的位置：取决于“杂凑函数（HASH）”得到的函数值来决定（对函数值求整、相对于表求余）

Hash表的基本思想是：

- 为符号表设置一个足够大的空间M
- 为符号构造一个散列函数Hash(Ki)，使得0≤Hash(Ki)≤M-1,i=1，2，…，n
- 这样查找Ki时，Hash(Ki)就决定了Ki在符号表中的位置

构造Hash函数的方法：

- 将标识符中的每个字符转换为一个非负整数
- 将得到的各个整数组合成一个整数（可以将第一个、中间的和最后一个字符值加在一起，也可以将所有字符的值加起来）
- 将结果数调整到0~M-1范围内，可以利用取模的方法，Ki%M（M为素数）

解决地址冲突的方法：

由于用户定义标识符的随机性，Hash函数值在0~M-1范围内不一定唯一

若两个标识符具有相同的函数值，则可用开放地址法或链地址法解决冲突，有关内容可以参考《数据结构》的教材。

6.2 错误处理

- 词法错误
- 语法错误
- 语义错误
- 违反了语言的环境限制
  - 数组维数太大
  - 循环嵌套层数太多

★ **词法错误：不合法单词**

例：
```
mian( ){
    int  3sum;
    …
```

词法错误、语法错误和语义错误

★ **语法错误：源程序在语法上不符合文法**

例：`A[x , y  =B+*C`

☆ **语义错误主要包括**：程序不符合语义规则或
超越具体计算机系统的限制

语义规则

1. 标识符先说明后引用
2. 标识符引用要符合作用域规定
3. 过程调用时实参与形参类型一致
4. 参与运算的操作数类型一致
5. 下标变量的下标不能越界

超越系统限制：(计算机系统和编译系统)

1. 数据溢出错误，常数太大，计算结果溢出。
2. 符号表、静态存储分配数据区溢出。
3. 动态存储分配数据区溢出。

错误处理方法有两种：

- **错误校正法：**

  根据文法进行错误改正

- **错误局部化法：**

  把错误的影响限制在一个局部的范围，避免错误扩散和影响程序其他部分的分析

错误局部化法

词法分析：发现不合法字符，显示错误，并跳过该标识符(单词)继续往下分析。

语法语义分析：跳过所在的语法成分(短语或语句)，一般是跳到语句右界符，然后从新语句继续往下分析。

错误局部化处理的实现（递归下降分析法）

err: 全局变量，存放错误信息。

•用递归下降分析时，如果发现错误，便将有关错误信息（字符串或者编号）送err，然后转错误处理程序；

• 出错程序先打印或显示出错位置以及出错信息，然后跳出一段源程序,直到跳到语句的右界符或正在分析的语法成分的合法后继符号为止，然后再往下分析。

if<C> then <statement>[else< statement >];

```
if_ statement( )
 {
    getsym( ); /*读下个单词符号*/
    C( ) ;      /*表达式处理程序*/
    if  not  sym=“then”
        {err :=“缺then” ;
          error( ); /*出错处理程序*/
        }
     else
        {getsym( );
          statement( );
         }
    if sym=“else”
       {getsym( );
         statement(  );
       }
}
```

```
error( )
 {printf(linecnt, err);
   do
     getsym( );
   while(sym!=“;” or sym!=“end” )
 }
```

- 发现错误立即跳到语句结尾处(语句右界符 ；或end),这样处理较粗糙,将跳过太多；
- 上例中,缺then,就将跳过整个条件语句,使得then后的语句都被跳过而不分析,其中有错误就发现不了

(3) 提高错误局部化程度的方法

设 S_1：合法后继符号集 (某语法成分的后继符号)

S_2：停止符号集　　(跳读必须停止的符号集)

```
error(S1,S2 )
 {printf(linecnt, err);
  do
    getsym( );
  while(sym not in  S1 or not in  S2 )
 }
```

if<C> then <statement>[else< statement >];

若<C>有错,则可跳到then

若statement有错,则可跳到else

小结

✓编译程序在其工作过程中使用最多的数据结构是表，在这些表中，符号表最为重要，它的生存期最长、使用最频繁。

✓掌握符号表的作用、内容、组织（多采用散列法）

✓明确错误处理的两种方法：错误校正和局部化处理

✓了解局部化处理方法的实现

2011年5月30日