第四章：语义分析

符号表管理

goto语句

1.1 语义分析部分三个核心知识点

- 检查语义错误
- 抽象地址的分配（层数、偏移）
- 符号表局部化、标识符的作用域

```
int main() {
    int a;
    float b,d;
    {
        int c;
        float a;
        {
            int d;
            float c;
            {
                float d;
                //...
                a=b+c+d;
            }
        }
        {
            char d; ... d=d+1;
        }
    }
    return 0;
}
```

此处的a,b,c,d对应的是什么

d对应的是什么

1.2 处理原则

符号表局部化处理的本质：在程序的某一个点P上，判断符号表的哪些信息是有效的

因此有以下原则：

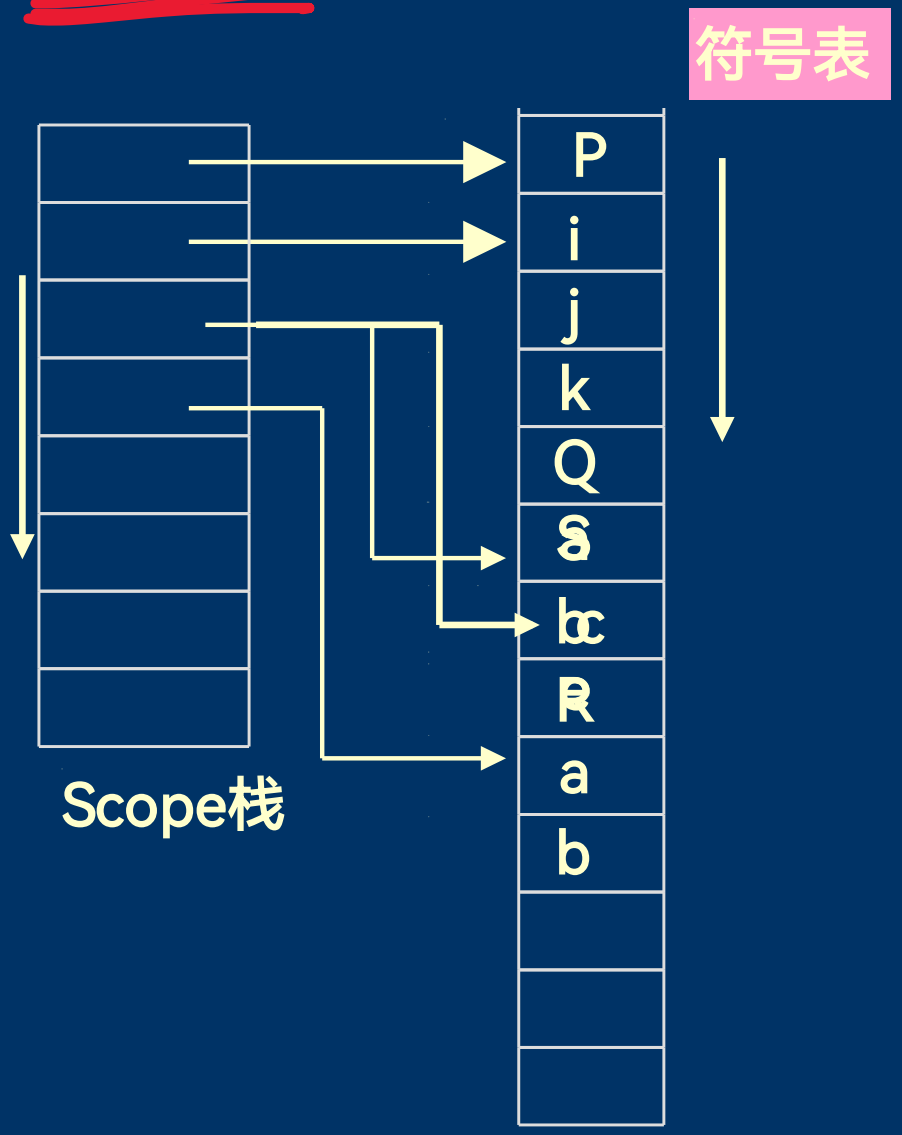
1. 每进入一个局部化区，记录本层符号表的首地址
2. 遇到声明性标识符时，构造其语义字，查本层的符号表，检查是否有重名，有则出错，否则就把其语义字填到符号表里。
3. 遇到使用性出现，查符号表，如果查到则读取其语义字，否则出现语义错误。
4. 退出一个局部化区，'作废'本层的符号表。

1.3 Pascal语言的符号表的管理

- Pascal语言的最大特点就是允许嵌套的过程声明。规定Pascal主程序的层数为0，在主程序中声明的标识符(包括过/函标识符)的层数为1，在第i层过/函中声明的标识符(包括形参和局部标识符)的层数为i+1(i≥1)。
- 用一个Scope栈来实现标识符的嵌套作用域。
- Scope栈的每项指向当前仍有效的某层符号表的首项，具体说Scope(0) 指向0层符号表的首项，Scope(i) 指向i层符号表的首项。

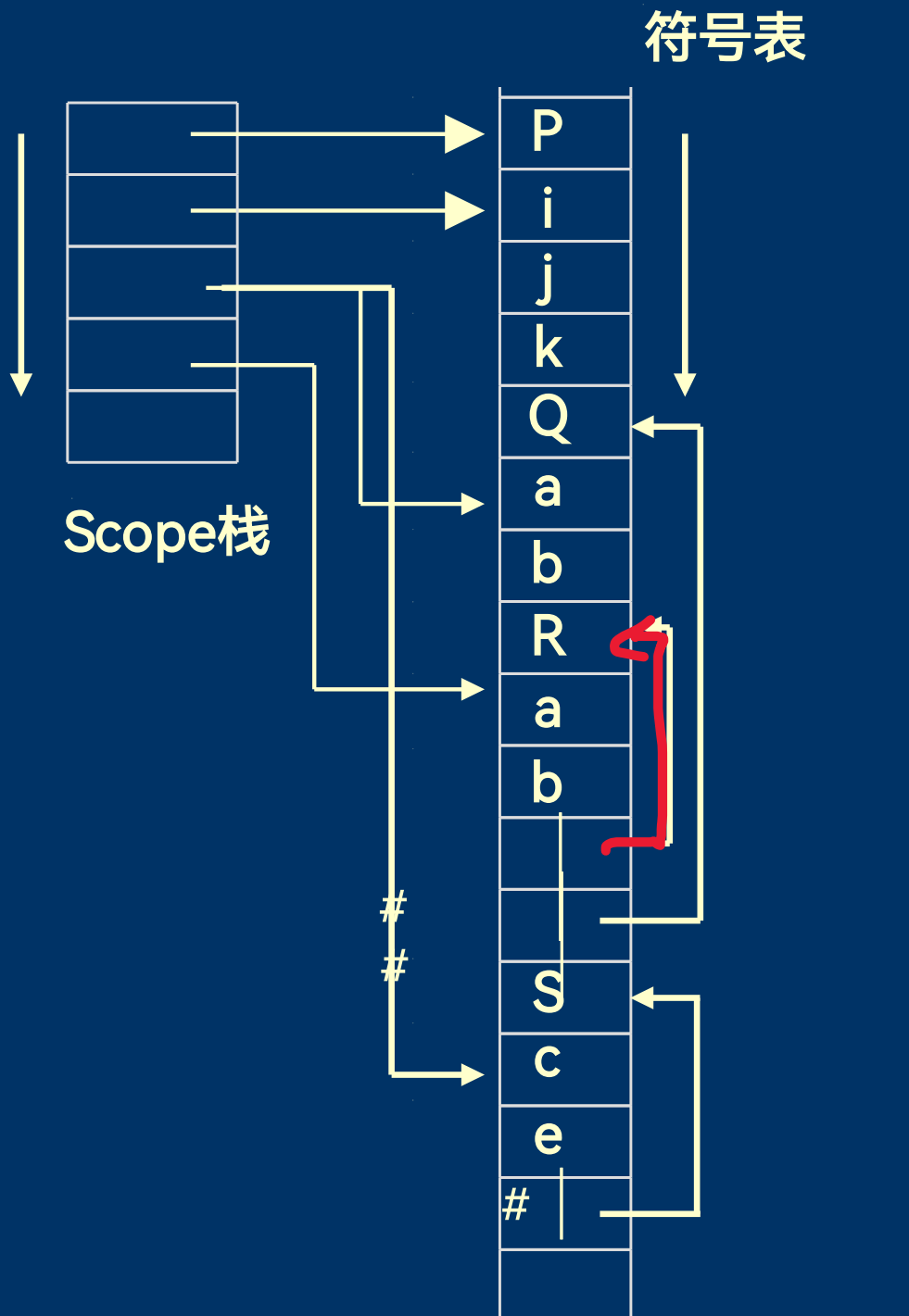
Pascal语言的符号表的实例(删除法)

```
PROGRAM example;
1.    procedure P1( )
2.        var i,j,k:integer; {t1}
3.        procedure Q2( )
4.            var a, b: real;
5.            procedure R3( )   {t2}
              var a,b:boolean; {t3}
1.            begin
2.            R的过程体（a,b,i)
3.            end
4.        begin
5.            Q的过程体（a,b,i)
6.        end;
7.        procedure S2( )
8.         var c,e:char;    {t4}
9.         begin
10.         S的过程体（a,b,i,Q)
11.        end ;    {t5}
      begin
          P的过程体    (a,b,i,P,Q,S)
20.   end;
21  begin..P( );..end.
```



● 关于Pascal语言的符号表的实例 (跳转法)

```
PROGRAM example;
1.    procedure P( )
2.        var i,j,k:integer; {t1}
3.        procedure Q( )
4.            var a, b: real;
5.            procedure R( )  {t2}
6.                var a,b:boolean; {t3}
7.                begin
                  R的过程体（a,b,i)
1.                end;
2.        begin
3.            Q的过程体（a,b,i)
4.        end;
5.        procedure S( )
6.         var c,e:char;    {t4}
7.         begin
8.           S的过程体（a,b,i)
9.         end;  {t5}
I     begin
I           P的过程体 (a,b,i)
    end
begin    .....    end.
```



1.4 C语言的符号表的管理

- C语言不允许函数的嵌套声明。
- 一个C程序是由若干个函数并列组成的（其中一定要有main函数），这些函数的地位都是相同的。从main函数调用开始执行，在后面声明的函数可以调用在它之前声明的函数。
- C语言的每个标识符的层次只有两个，即0层和1层。全局标识符和所有的函数标识符的层数是0，函数的形式参数和局部标识符的层数是1。

（1） 简单的C语言符号表_驻留法

```
int x,y;
void swap(int* a, int*b )
{
   int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}
void main()
{
x = 10;
y = 20;
swap(&x,&y);
}
```

x	varKind	intPtr	dir	0	0		
y	varKind	intPtr	dir	0	1		
swap	routKind	NULL	actual	0		size	false
a	varKind	atptr	indir	1	off_0		
b	varKind	btptr	indir	1	off_0+1		
tmp	varKind	intPtr	dir	1	off_0+2		
#							
main	routKind	NULL	actual	0	null	size	False
#							
#							

atptr→	1	pointerTy	intPtr
btptr→	1	pointerTy	intPtr

（2）带有分程序的C语言全局符号_驻留法

```
1.   main()
2.   {1
3.         int a;
4.         float b，d;
5.     {2
6        int c;
7.       float a;
8.       {3
9.             int d;
10.            float c;
11.          {4
12.              float d;
13.            ……
14.            a=b+c+d;
15.          } 4
16.      } 3
17.      {5
18.        char d;
19.      } 5
20.    } 2
21.  } 1
```

（ 假定off（偏移）为0）

main	routKind	NULL	actua	0	NULL	false
a	varKind	intPtr	dir	1	0	
b	varKind	realPtr	dir	1	1	
d	varKind	realPtr	dir	1	3	
c	varKind	intPtr	dir	1	5	
a	varKind	realPtr	dir	1	6	
d	varKind	intPtr	dir	1	8	
c	varKind	realPtr	dir	1	9	
d	varKind	realPtr	dir	1	11	
#						
#						
d	varKind	charPtr	dir	1	13（8）	
#						
#						
#						

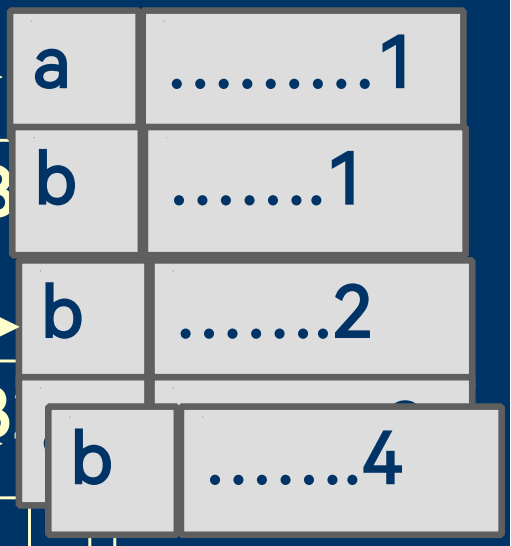
（3）带有分程序的C语言全局符号_删除法

```
void main( )
{
    int a = 1;                                   B
    int b = 1;
    {
        int b = 2;                               B
        {
            int a = 3;                           B3
            printf("a = %d, b = %d\n",a, b);
        }
        {
            int b = 4;                           B4
            printf("a = %d, b = %d\n",a, b);
        }
    }
    printf("a = %d, b = %d\n", a, b);
}
```

a = 3, b = 2

a = 1, b = 4

a = 1, b = 1



1.5 嵌套式语言并列式语言的比较

- 特殊情形，从程序设计语言的角度来说有嵌套式语言和并列式语言
- 从处理难度角度来说，可能嵌套式语言复杂，并列式简单。 嵌套式语言里还要考虑变量运行环境。
- 并列式语言的局部化区比较固定，层数分成两层，全局量定义成0层，局部量定义成1层。分配地址特殊情形就是分程序结构，分程序可以是嵌套的，他的层数都是看成同一层，查局部化区每个分程序看成一个独立的局部化区。
- 分配抽象地址的时候为了节省空间，并列的分程序可以是并行的分配，嵌套的分程序必须要连续的往下分，前一个分程序从10~100，并列的可以还是10~100。

2. goto语句和标号处理

- 2.1 goto语句及"goto之争"
- 2.2 标号的出现形式
- 2.3 可能出现的错误
- 2.4 实现方式
- 2.5 实现算法

2.1 goto语句及"goto之争"

goto 标号

标号:

例如:

```
r1:x1=f(x0);
if (abs(x1-x0)<0.0001)  goto r2;
x0=x1; goto r1;
r2: ;
```

- 在上世纪70年代，曾经发生过关于goto语句的一场大辩论，主题就是要不要在高级程序设计语言中把goto语句保留下来

2.2 标号的出现形式

- 标号在程序中出现共有三种形式

1. 声明性出现 在程序的声明部分给出一个声明，声明哪些标识符是标号。例如：label R1,R2
2. 定位性出现 起到定位的作用，例如： R1：s, s是一个语句，R1是一个语句标号，表示R1这个语句标号定位在s语句前面
3. 转移性出现 例如goto R1, 即把程序的控制结构转到标号为R1的位置上开始去执行

2.3 可能出现的错误

1.标号的重复声明

2.标号的重复定位

3.使用的标号未声明（语言决定的）

4.有转移但是无定位

2.4 实现方式

- 一般有两种实现方式:

goto R → Jump R

效率较高但实现困难，难以定位跳转地址

goto R → Jump *R

为标号分配一个存储单元，跳转指令采用间接寻址方式，从对应存储单元中读取该标号定位的地址。

通常采用第二种实现方式

2.4 实现方式

- 要做好两部分工作:

1.分配存储单元

2.检查语义错误

- 因此需要构建两个表

标号表:

标号名	状态标志	地址

状态有定位填1

无定位填0

未找到定位标号表:

标号名
标号名
标号名

2.5 实现算法

- **遇到声明性标号时查标号表**

1.如果表中有该标号，说明该标号被重复声明

2.如果没有，则将标号填到符号表里，状态填0

- **遇到定位性标号时查标号表**

1. 没查到，报错（标号未声明）

2.查到，状态位为0，则将其变为1，生成目标代码时回填地址。状态位为1，报错（标号重复定位）

- **遇到转移性标号时查标号表**

查到，则建立指针；查不到，将该标号填入'未找到定位标号表'

- 程序单位结束时，复查'未找到定位标号表'，确定是否存在错误。

例子:

lable r1,r2;【1】

r1【2】:x1=f(x0);

if (abs(x1-x0)<0.0001) goto r2【3】;

x0=x1; goto r1【4】;

r2【5】: ;

【2】

r1	1	地址
r2	1	地址

r2

使用驻留法中的跳转方法给出下面类C程序（嵌套式）处理结束时的全局符号表。（采用顺序表结构） 表项为层数和偏移

```
void P (){
      int  i , j , k;
      void P1 (){
         real  m; int n;
              void  P2 (){
                    bool a , b;
                    P2的过程体
                    }
              P1 的过程体
          }
      P的过程体
}
void  Q(){
      real a, b;
      void  Q1( ){
            int  i  , j;
            Q1的过程体
            }
      Q的过程体
}
```