第四章：语义分析

抽象地址

符号表

1. 抽象地址

- 1.1 地址分配
- 1.2 抽象地址结构
- 1.3 层数定义
- 1.4 过程活动记录
- 1.5 分配原则

1.1 地址分配原则

- 静态分配

  在编译时间即为所有数据对象分配固定的地址单元，且这些地址在运行时间始终保持不变。是一种直观的方法，但不适用于动态申请空间。

- 动态分配

  程序中变量分配的地址不是具体的地址，而是一个抽象地址，当程序运行时根据抽象地址分配具体的物理地址。

1.2 抽象地址的结构

层数	偏移

- 抽象地址的形式是一个二元组，由层数和偏移组成。

层数主要是针对嵌套式语言，表示的是某个函数所处的嵌套定义层数。

偏移是针对过程活动记录的一个相对偏移量

1.3 层数的定义

- 嵌套式语言中:

主程序设为0层

主程序直接定义的函数和过程定义为1层

若某函数为L层，则该函数直接定义的函数为L+1层

- 并列式语言中:

全局变量定义为0层，函数中的变量定义为1层

1.4 过程活动记录

- 每次函数被调用时都会给函数分配一片空间，用以存储如图信息，我们把这片存储空间称作过程活动记录
- 偏移是针对过程活动记录的，通过起始位置+偏移量就可以找到对应的物理位置。
- 过程活动记录中存储的顺序实际上是处理的先后顺序。

1.5 空间分配原则

临时变量一律一个单元

局部变量按类型大小分

形参

- 地址引用型形参：分一个单元
- 值引用型形参：按类型大小分
- 过/函形参：分两个单元（入口地址，display表信息）

2. 符号表

- 2.1 符号表结构和总体组织
- 2.2 New token与符号表的关系
- 2.3 符号表查表技术
- 2.4 标识符的作用域
- 2.5 局部化区的语义错误检查
- 2.5 标识符处理原则
- 2.6 实例分析

2.1 符号表结构和总体组织

- 符号表存储的是标识符的语义信息，包含两部分：**标识符**的名字；**语义字**(种类、类型、抽象地址...)
- 不同类别的标识符所包含的信息是不同的。
- 符号表的总体组织既可以采用多表结构，也可以采用单表结构,也可以二者折中。究竟采用哪种结构并没有统一的规定，编译程序可根据实际处理语言的需要进行选择。

2.2 New token与符号表的关系

- 声明性出现：

<标识符，a> 创建标识符a所对应的符号表，将token转化为 <标识符，指针（符号表中创建的项）>

- 使用性出现：

<标识符，a> 到符号表中查找a，找到则将token转化为<标识符，指针（符号表中查找到的项）>

2.3 符号表查表技术

- 顺序查表法
- 折半查表法（二分法）
- 散列查表法（哈希表）

2.4 标识符的作用域

- 标识符的作用域：是指某标识符可以有效使用的范围，标识符的作用域是一个程序段，称之为程序的局部化区，通常是一个子程序或者分程序，局部化区是允许含有声明的最小程序单位。
  - void function【(){...}】
  - structure data【{}】
  - { 【】 }

2.5 局部化区的语义错误检查

- 变量的重复声明

  在一个程序的局部化区里，同一个标识符不能被声明两次。

- 标识符的使用有无声明

  强类型语言规定标识符必须先声明后使用，弱类型语言无要求。

2.6 标识符处理原则

- 这两个问题清楚之后，我们来看一下标识符的处理原则，既要保存有用的信息，又要杜绝语义错误。

每进入一个局部化区，记录本层符号表的首地址

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。

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。

退出一个局部化区，'作废'本层的符号表。

实例分析

```
proc p()
    type at=array[1..100] of array[1..10] of
integer
    var x:real; a:at; i:integer;
    proc p1(var a1:at;  a2:at)
        var x:integer; a:real;
        proc p2(n:integer)
            var m:1..50; x:real;
            m,n,x(使用性出现)
        end
        a1,a2,x,a,i（使用性出现）
    end
    x,a,i（使用性出现）
end
```

实例分析

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

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。
- 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，并列的可以是10~200。 还有变量运行环境，在嵌套式语言里要考虑。