# 编译原理 实验三 报告

## 实验目标

1. 基本要求: 将C--源代码翻译成中间代码, 并能够成功在虚拟小程序上执行
2. 分组选做要求: 实现对数组变量的支持.
3. 额外选做要求: 实现对结构体变量的支持.

## 实验过程

我们在本次实验的中间代码生成是在上一次实验语法检查的语法制导翻译的框架中进行的, 语义检查的同时就生成的中间代码. 采取这种方式主要是因为我们在上次实验中实现了局部作用域, 因此符号表是会动态变化的, 无法等语法检查结束再重新开始翻译.

并且, 与实验手册上的不同, 我们中间代码的生成是增量翻译, 即对每一个语法单元的翻译时, 直接向全局的生成代码中添加代码; 而不是作为返回值返回, 再整合. 采取这种方式主要是考虑到这样做的效率比较高.

### 框架

在本次实验, 我们主要使用了两个结构体

1. **Operand**: 用来存储中间代码的各种操作数, 有变量, 临时变量, 引用, 标签, 常数这几种类型. 不同的操作数主要通过类型及其编号来区分.
2. **InterCode**: 用来表示中间代码, 实现了手册提及的所有形式. 需要提及的是: `x := *y`, `*x := y`, `x := y` 在这里被当作是不同的类型ASSIGN_FROM, ASSIGN_INTO, ASSIGN. 而 `x := &y` 是ASSIGN类型, 只不过操作数 `&y` 是引用类型.

### 属性

本次实验添加了几个属性用于中间代码生成:

1. **place**: 综合属性; 与实验手册的做法不同, 我们这里的place是综合属性, 表示表达式结果所在的变量. 有时, 也存储一个数组变量的地址.
2. **isRef**: 综合属性; 配合place使用, 用来表示place所指的变量存的是一个地址. 若作为左值, 则生成 `*place := right`; 若作为右值, 则先取出该地址上的值, `t := *place`, 在继续生成其他代码如赋值, 或者运算.

3. **label_true**, **label_false**: 继承属性; 表示逻辑表达式为真或为假时的跳转目标, 用于逻辑表达式的翻译.

## 翻译模式

表达式的翻译基本与实验手册上的一致, 只不过由于我们是增量翻译, 因此需要特别注重代码生成与翻译子节点之间的顺序.

下面重点介绍数组, 结构体的处理, 变量声明和参数传递.

### 数组

首先, 我们将数组视为一个指针. 因此无论是数组在符号表中的变量名, 或是数组的参数名, 我们都认为其存的是数组的基地址. 因此, 数组的访问过程, 其实是地址的计算过程. 这个过程写成伪代码就是:

```
// for Exp1 LB Exp2 RB
genCode "num := Exp2.place"
genCode "offset := num * #array.element_width";
genCode "place := Exp1.place + offset";
if(array.element == int) isRef = true;
array = array.element;
```

其中Exp1代表的是一个数组, 所以其place存的是其基地址. 需要注意的是, 这里的数组不一定是声明的数组变量, 可能是某个数组中的子数组. array是Exp1的综合属性, 表示Exp1所代表的数组. 把n维数组当作是n-1维数组的一维数组, 则array.element_width就是n-1维数组的大小. 用下标乘n-1维数组的大小, 这样就方便计算当前的偏移量大小.

当到达最后一层, 表达式表示的是一个数组中的整型变量时, 设isRef为真, 表示place存的是这个整型变量的地址, 而不是整型变量本身.

### 结构体

与数组类似, 我们将结构体变量看作是一个指针.
伪代码如下:

```
// for Exp DOT ID
sym = lookup(Exp.symble_table, ID)
genCode "place := Exp.place + #sym.offset"
```

其中, Exp代表的是一个结构体, 因此其place属性存的是地址. ID在符号表中维护着一个offset值, 表示从表头到该ID的偏移量.
同上, 若这个ID代表的是一个整型变量, 那么isRef会设为真, 表示place不是整型变量本身而是其地址.

**参数传递**

参数传递需要注意两点: 形参和实参的顺序问题和数组和结构体的传递问题.

对于形参, 其顺序是从上至下是第一形参, 第二形参, ... 等等. 而形参列表的产生式为:

```
ParamList -> ParamDec COMMA ParamList
    | ParamDec
```

因此无需特别处理, 在translate_ParamDec时生成代码 `PARAM ID` , 则顺序就是正确的.

但对于实参列表, 其顺序是从上至下倒数一个实参, 倒数第二个实参, ... 第一个实参. 而实参列表的产生式为:

```
Args -> Exp COMMA Args
    | Exp
```

因此, 需要在对于一个Args的翻译, 要在其调用子节点的translate_Args后才能生成代码 `ARG Exp.place` .

另外, 对于数组和结构体的传递, 我们需要传递的是指针. 因此, 无需特别处理, 因为当Exp代表一个数组时, Exp.place本身就是其基地址. 然而, 若Exp是一个数组中的整型变量, 此时我们需要先提取其值 `t := *Exp.place` , 再传递参数 `ARG t` .

# 总结

假如只考虑翻译, 这次实验相比于上一个实验较为简单, 因为需要考虑的语法单元少了, 而且翻译模式实验手册也有给出.

本来想实现优化, 但由于时间不够, 也未能如愿.