Lab3-中间代码生成

程序功能说明

任务是在词法分析、语法分析和语义分析程序的基础上，将`C--`源代码翻译为中间代码。本次实验使用三地址码作为中间代码。

程序接收`C--`语言源码作为输入，输出三地址码形式的中间代码，并且可以在虚拟机上正确运行。

本实验完成了所有选做，可维护性和健壮性良好。

实现方式

在完成词法分析、语法分析、语义检查的基础上生成中间代码，总体逻辑是再语法树上执行先序遍历，并且需要使用语义检查时生成的符号表。

定义的相关结构如下：

- 操作数`Operand`：操作数类型，字符串值（用于函数名，变量名，不等号），整数值（用于临时变量，标签，立即数，空间大小）
- 单元中间代码`Incode`：代码类型，操作数1,2,3，前指针，后指针
- 中间代码段`Snippet`：头指针，尾指针
- `ExpCode`：前置代码段，返回操作数，返回类型

本次实验中，我选用双向链表来组织和表示中间代码，`ExpCode`作为Exp语法单元的处理结果，当Exp被处理时，它应当返回一段前置代码段用于计算Exp的值，结果最终存放在一个操作数，并把这个操作数和它的类型返回给父节点。

编译使用说明

源文件：`main.c, lexical.l, syntax.y, tree.h, semantic.h, semantic.c, intercode.h, intercode.c`

编译时直接执行make即可，make的默认目标包含了clean，不需要手动`make clean`。

编译产生可执行程序`pcc`（全称：`pikaball C-- Complier`），当不指定outputfile时，将输出到stdout。

```
./pcc filename [-o outputfile]
```

（附注：gcc版本是`gcc (Ubuntu 12.3.0-1ubuntu1~23.04) 12.3.0`）

细节说明

对语义检查的修改

在上一个实验中，对于多重作用域的问题，我在每次结束CompSt语法单元时会从符号表删除当前深度的符号链；本实验中，我将中间代码生成作为一个单独的模块，在语义检查结束后才执行，所以不能直接删符号，采取的方案是为符号表项结构体添加一个`int removed`，用于在语义检查过程中标识当前符号是否已被移除，但不会从符号表中删除。

修改结构后，符号表的查找函数也做相应更改，加入`removed`检查，另外，也需要单独为中间代码生成写一个单独的查找函数（因为此时不需要检查`removed`）。

第二个修改是给符号表项添加一个`int isParam`，用来标识当前符号是否是函数的一个参数。这个修改主要是为了数组和结构体服务的，因为在这个实验规定的中间代码语法中，数组和结构体在作为变量和作为参数时有不同的表现形式。

具体来说：如果源代码是：

```
int main()
{
int a[10];
a[2]=1;
}
```

这里a[2]作为一个exp单元被翻译时，先生成偏移量，然后对数组符号取地址（这里va实际直接对应a[0]），计算目标地址：

```
...
t0 := #2 * #4
t1 := &va
t1 := t1 + t0
*t1 := #1
...
```

而当这里的a作为参数出现时，va这个符号保存的是a[0]的地址而不是a[0]本身，不需要再加&符号取地址。

逻辑表达式的处理

对于逻辑表达式，它应先计算一个实际返回值，而不应当直接揉进IF里：

`if [Exp] ...`，我会先计算`Exp`的值，然后保存到临时变量，再判断该变量是否为0，比如`if (a>b) ...`翻译成:

```
t0 := va - vb
IF t0 > #0 GOTO label0
t1 := 0
GOTO label1
label0 :
t1 := 1
label1 :
IF t1 != 0 GOTO ...
```

其中t1即是`a>b`这个表达式的值。而不会写成：`IF va > vb ...`，因为这样必须在IF节点直接进入Exp内部解析（而不是交给Exp的翻译函数），会破坏代码结构的一致性，而且也无法处理`if (a)`，`if ([Exp] and [Exp])`的形式。也就是说，所有在源码中的if语句，都要一致地采用`!= #0`的翻译形式。

多维数组的处理

当遇到多维数组形如`a[2][3][4]`时，其语法结构是`Exp(a[2][3]) LB 4 RB`，因为必须计算偏移量，这个偏移量要逐层算出，但符号表只存了a，而不将`a[]`和`a[][]`这样的子结构作为单独一项，所以不能递归地进入前面的Exp翻译，故而在遇到这样的结构时原地解析掉，取出每一层的索引值存进一个数组，再逐层算偏移量。具体可见代码。