lab3实验报告

151242060 银加

实验环境及编译方法

我使用的是MacOS系统，我在实验的根目录下创建了Makefile文件。在测试时可以在根目录下执行 `make all` 即可完成编译并且生成parser语法分析程序。需要注意的是，与实验指导稍有区别的是在mac上 `-lfl` 需要替换成 `-ll` 才能编译成功，所以在测试时可能需要根据相应操作系统对Makefile进行修改。

进行测试时，可以执行 `make test1` ，... `make test4` 对测试样例分别进行测试。输出结果 `out.ir` 文件均显示在test文件夹中。所有样例均来自实验指导。如果需要添加新的测试样例，可以在根目录下test文件夹中添加新的样例，然后在Makefile中添加相应语句或直接执行即可。

实现功能说明

我的语义分析器实现的功能包括所有的必做内容，加上选做(3.1)

文件结构

在lab2的基础上新增了以下两个文件：

- `intercode.h`，`intercode.c` 主要实现了与中间代码生成相关的数据结构。包括 `Operand` 、 `Intercode` ，分别实现了操作数和中间代码的结构体类型。除此之外，文件中还包括关于代码优化相关的函数。

在lab2的基础上主要修改了以下一个文件：

- `semantic.c` 在遍历结点进行语义分析的同时进行中间代码分析。

主要功能实现方法

- **线性IR**
  线性IR的实现我采用的是双向链表，其中定义了 `pre` 和 `next` 两个分别指向前后的指针，数据结构基本采用了实验指导中的结构，我枚举了中间代码的可能类型，见 `Intercode_` 中enum枚举的类型。

- **数组与结构体**
  我解决了一维数组和结构体变量的地址计算，主要实现了一个 `typeSize` 函数，以 `type` 作为传入参数计算地址，如遇到高维数组，则会报错无法translate。具体实现参见 `symboltable.c`

代码优化方法

与代码优化相关的函数主要在 `intecode.c` 中，当然在 `semantic.c` 中进行分析时也有小的优化。下面介绍 `intecode.c` 中的优化函数：

- **优化Goto代码**
  对应函数： `optGotoCode`
  这里主要消除不必要的跳转，例如：

```
IF v1 RELOP v2 GOTO LABEL 1
GOTO LABEL 2
LABEL 1

//转化为

IF v1 ~RELOP v2 GOTO LABEL2
LABEL 1
```

也会消除如下形式的 `Goto` 冗余：

```
GOTO LABEL 1
LABEL 1

//转化为

LABEL 1
```

- **消除冗余的Label**
  对应函数： `deleteLabel`
  将没有GOTO语句或者IF_GOTO语句指向的Label删去，并且把连续的label合并，例如实验指导中样例1中的

```
LABEL   label6  :
LABEL   label3  :

//转化为

LABEL   label3  :
```

- **计算常数运算并替换常数**
  对应函数： `figure_const`
  将常数之间的运算计算出来，如果存储结果的是一个临时变量 则将该临时变量直接替换为该结果常数，例如：

```
t3 := #1
WRITE t3

//转化为

WRITE #1
```

- **消除死代码**
  对应函数：`remove_dead_code`
  基本思路是先将中间代码划分为基本块，然后对基本块活跃变量分析，得到可以消除的代码

- **寻找局部公共子表达式**
  对应函数：`same_subexp`
  对于局部内的两个赋值操作，如果赋值内容相同，且被赋值对象为临时变量，则可以对两个赋值操作进行合并，对于“局部”的具体定义，详见代码。

实验总结

本次实验主要内容为中间代码生成，和中间代码优化。其中中间代码优化是很有意思的环节，因为有各种方法可以尝试，我使用的也只是其中的几种最容易想到，也比较容易实现的方法，如果有更多时间，我可能能完成更多的优化，总的来说这次实验加深了我对于概念的理解和运用，收获颇丰。