实验三 中间代码生成

姓名：肖丹妮　　学号：201220199　　专业：计算机科学与技术系

一、实验环境与编译方法

实验环境与手册一致；利用提供的Makefile文件进行编译。

二、实验实现的功能

- 完成了必做内容，能够在词法分析、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。
- 同时完成了所有选做要求，即要求3.1、3.2。

三、实验内容与核心代码

3.1 数据结构

3.1.1 中间代码的表示

参考指导手册上的数据结构表示，进行了如下修改和完善：

- 删去了操作数中的ADDRESS类型，改为在Operand结构体中添加了字段isaddr，用于标识当前操作数存储的是值还是地址。
- 完善了中间代码InterCode结构体的值，根据类型icKind，字段u分为oneop、assign、binop、ifgoto、dec。
- 为ASSIGN和ADD类型的中间代码增加了AssignType，便于打印函数根据此条中间代码的AssignType选择不同的打印方式。AssignType可能是普通赋值、取右边的地址赋给左边、取左边的值赋给左边、令左边指向的内存地址的内容等于右边的值。

3.1.2 符号表结点

相比与实验二，增加了两个字段：val_no、isaddr，分别与Operand结构体中的val_no、isaddr对应。

3.1.3 设计的全局变量

```
ICLnodep head = NULL;                    //双向循环链表的附加头结点
int VarNum = 0;                          //计数变量，用于编号
int TempVarNum = 0;                      //计数临时变量，用于编号
int LabelNum = 0;                        //计数标号，用于编号
SymbolNodep st_im[SYMBOL_TABLE_SIZE]; //符号表，因为不存在作用域，所以只需要散列表即可，存储函数、变量、形参
ICLnodep newic = NULL;                   //指向新构造的中间代码结点
Operandp con0 = NULL;                    //常量0
Operandp con1 = NULL;                    //常量1
Operandp arg_list[100];                  //实参列表，顺序存储实参，倒序生成中间代码
int argn = 0;                            //实参列表中的实参个数
```

3.2 中间代码生成的整体实现

将关于中间代码生成的所有内容写到一个单独的文件intermediate.c中，等到语义检查全部完成并通过之后再生成中间代码。

3.2.1 构造函数

- 构造操作数——Operandp Create_Operand(int isaddr, opKind kind, ...);
- 构造单条中间代码并且插入双向循环链表——bool InterCodeList_CandI(ICLnodep first, ICLnodep toinsert, icKind kind, ...);
  根据kind及后面的参数构造中间代码，并调用InterCodeList_Insert()，将中间代码插入以first为附加头结点的双向循环链表中。

3.2.2 打印函数

- 打印操作数——void PrintOprand(FILE *fp, Operandp op);
  根据opKind的值分类打印即可。
- 打印双向循环链表中的所有中间代码——void PrintInterCodes(char *fname);
  顺序打印每一条中间代码，根据icKind的值分类打印，注意ASSIGN和ADD类型还要讨论AssignType的类型。

3.2.3 translate函数

以条件表达式的翻译函数translate_Cond为例，分析每个语法结点的思路和语法分析大致相同，只是操作的具体内容不一样。

- 利用Childi()得到当前语法树结点的子结点；
- 根据当前语法树结点的childtype将子结点分类处理；
- 具体每种子结点的翻译和手册给出的翻译模式一致，调用生成操作数的函数、翻译子结点的函数、生成中间代码的函数等对应完成。

```
bool translate_Cond(TreeNode *tn, Operandp label_true, Operandp label_false)
{
    if (tn != NULL)
    {
        bool isright = true;
        TreeNode *Child1 = Childi(tn, 1);
        TreeNode *Child2 = Childi(tn, 2);
        TreeNode *Child3 = Childi(tn, 3);
        switch (tn->childtype)
        {
        case 4: // Exp RELOP Exp
                ……
        case ……
        }
        return isright;
    }
    return true;
}
```

3.3 选做要求的实现

在必做内容的基础上增加的处理操作如下：

- 在变量定义时，若变量的类型是数组或者结构体，构造一条DEC类型的中间代码：
  - InterCodeList_CandI(head, newic, DEC, varx, Get_Width(newsn->type));
- 在翻译形参时，若参数的类型是数组或者结构体，将符号表结点的isaddr置为1；

- 在表达式翻译中，如果Exp产生了一个标识符ID，考虑数组或者结构体类型，在place变量是地址类型时应该对变量取地址，否则place变量是地址类型时，应该取变量的值存到以place值为内存地址的内存单元。

```
if (p->isaddr == 0)//右边ID代表的变量存放的值不是地址
{
    if (place->isaddr == 0)//place中存放的值不是地址
        InterCodeList_CandI(head, newic, ASSIGN, place, varx, DEFAULT);
    else
    {   //place中存放的值是地址
        if (p->type->kind == ARRAY || p->type->kind == STRUCTURE)
            InterCodeList_CandI(head, newic, ASSIGN, place, varx, GETADDR);
        else
            InterCodeList_CandI(head, newic, ASSIGN, place, varx, SETDATA);
    }
}
else ……
```

- 在表达式翻译中补充数组的翻译以及结构体的翻译
  - 数组的翻译：
    此时表达式的翻译分为两类，若Exp1产生ID，由ID查表得到数组的首地址存放在newtemp1中，再根据Exp2和Exp1数组的类型宽度相乘计算要加上的地址，存放在newtemp2中。

    ```
    Operandp width = Create_Operand(0, CONSTANT, Get_Width(tn->syn->type));
    Operandp newtemp2 = Create_Operand(0, TEMPVAR, ++TempVarNum);
    InterCodeList_CandI(head, newic, MUL, newtemp2, newtemp1, width,
    DEFAULT);
    ```

    最后讨论place以及变量（若Exp1产生ID）的存放的是值还是地址，分别生成中间代码。
  - 结构体的翻译：思路与数组大致相同，但是具体求地址时，是在Exp1的所有域中计算偏移量，直到找到ID对应的字段。

    ```
    FieldListp fieldhead = Child1->syn->type->uval.structure;
    int addr_toadd = 0;
    while (fieldhead != NULL && sameStr(fieldhead->fieldname, Child3-
    >strval) == false)
    {
        addr_toadd = addr_toadd + Get_Width(fieldhead->type);
        fieldhead = fieldhead->tail;
    }
    Operandp conx = Create_Operand(0, CONSTANT, addr_toadd);//得到偏移量存放在
    conx中
    ```

- 在表达式翻译中，若Exp产生了赋值表达式Exp1 ASSIGNOP Exp2，补充左值表达式数组元素访问或结构体特定域的访问的翻译
  - 数组访问或者结构体访问基本思路一致，递归调用translate_Exp翻译Exp1，得到左值表达式的地址放在临时变量newtemp1中，递归调用translate_Exp翻译Exp2，得到右值表达式的值或者地址放在临时变量newtemp2中。如果右值不是数组或者结构体类型，也就是newtemp2中存放的是值本身，生成中间代码：*newtemp1 := newtemp2。最后将newtemp1的结果存回place。