编译系统实验三

姓名：姚舜宇　学号：1190202107　班级：1903602

实验目的

在词法分析、语法分析和语义分析的基础上将 C--源代码翻译成为中间代码。

实验环境

GNU Linux Release: Ubuntu
GCC
GNU Flex
GNU Bison

实验内容

中间代码的数据结构：双向链表。

```
struct Operand_{
    enum{FROM_ARG,VARIABLE,TEMP,CONSTANT,ADDRESS,WADDRESS,FUNCTION,LABEL,RELOP}kind;
    int u_int;//t1t2
    char* u_char;
    Type type;//用于结构体和数组变量的Offset查询
};

struct InterCode_{
    enum{ILABEL,IFUNCTION,ASSIGN,
        ADD,SUB,MUL,DIV,
        ADDRASS1,ADDRASS2,ADDRASS3,
        GOTO,IF,RETURN,DEC,ARG,
        CALL,PARAM,READ,WRITE}kind;
    union{
        struct{Operand op;}ulabel;
        struct{Operand op1,op2;}uassign;
        struct{Operand result,op1,op2;}ubinop;
        struct{Operand x,relop,y,z;}uif;
        struct{Operand op;int size;}udec;
    }u;
    InterCode before;
    InterCode next;
};
```

InterCode 表示单条中间代码，其中有属性 before 和 next 记录前后两条中间代码，即为双向链表。Operand 表示操作或变量。

每个语法单元设置相应的 translate 函数，根据产生式进行调用：

1. Exp -> Exp1 DOT ID
   Code1 = translate_Exp(Exp1, t1)
   取出结构体变量的首地址
   Code2 = [place := t1 + offset]
   通过 t1 中的 type 类型以及 ID 中的域名信息，计算出该域名在结构体中的偏移量
   记录 place -> type = ID.type，通过在符号表中查找 ID 并取出
2. Exp -> Exp1 LB Exp2 RB
   Code1 = translate_Exp(Exp1, t1)
   取出数组变量的首地址

Code2 = translate_Exp(Exp2, t2)
计算数组元素的序号，存储在 t2 -> u_int 中
Code3 = [offset := t2 * size]
Size 为该数组单个元素大小，通过 t1 -> type 可计算得到
Code4 = [place := t1 + offset]
加入偏移量得到数组元素的具体地址
记录 place -> type = Exp1.type -> u.array.elem

3. Exp -> ID LP Args RP
   判断是否为 write 函数，若是，则：
   Code1 = translate_Exp(Args -> child, t1)
   Code2 = [WRITE t1]
   若不是，则：
   Code1 = translate_Args(Args -> child, t1)
   Code2 = [place := CALL ID.name]
4. Exp -> ID LP RP
   判断是否为 read 函数，若是，则：
   Code2 = [READ place]
   若不是，则：
   Code1 = [place := CALL ID.name]
5. Translate_Args: Args -> Exp COMMA Args | Exp
   正向翻译 Exp 时，利用双向链表记录中间代码
   所有 Exp 语句翻译完成后，从链表尾部向前将中间代码加入到总的 Intercode 中
6. 数组与结构体定义
   生成语句：VarDec -> ID | VarDec1 LB INT RB (VarDec1 -> ID)或 StructSpecifier -> STRUCT Tag (Tag -> ID)
   在符号表中找到 ID 对应的 TABLE，通过其 field -> type 计算出大小 size
   Code1 = [DEC ID size]
7. 函数定义
   生成语句：FunDec -> ID LP VarList RP | ID LP RP
   Code1 = [FUNCTION ID:]
   在函数表中找到 ID 对应的 FUNCTION，通过之前记录的函数参数列表，对于每一个参数都生成一个 PARAM 的语句。
8. 其他
   根据产生式调用下层的 translate 函数，直至 translate_FunDec/Stmt/Exp/VarDec/Args/StructSpecifier。
   根据指导书以及上述自行翻译的语句生成对应的中间代码，最后返回到 Pragram 中后进行中间代码的输出。

程序执行流程：

```
yyrestart(f);
yyparse();
if(is_error==0){
    //print_tree(head);
    Program(head);
    translate_Program(head,f2);
}
return 0;
```

在 main 函数中，确保程序无词法、语法、语义错误后，使用生成的语法分析数，调用函数 translate_Program。

```
void translate_Program(struct Node* now,FILE* F){
    if(interim_is_error==1) return;
    struct Node* extdeflist=now->child;
    while(extdeflist->child!=NULL){
       struct Node* extdef=extdeflist->child;
       translate_ExtDef(extdef);
       extdeflist=extdef->brother;
    }

    if(interim_is_error==0){
        translate_print(F);
    }
}
```

使用先序遍历的方式遍历语法树，同时构建保存中间代码的双向链表。遍历结束后，调用函数 translate_print 把中间代码输出到指定文件。

编译过程：

./makefile.sh

```
#!/bin/bash
bison -d syntax.y
flex lexical.l
gcc syntax.tab.c interim.c semantic.c main.c -lfl -o parser
```

运行命令：

./parser test.cmm out.txt

将输出定位到文本文件 out.txt