ISE lab3

221900073 孙佳琪

程序所实现的功能

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，并将中间代码输出成线性结构。

如何被编译

1. make
2. ./parser 测试文件 输出文件

个性化的内容

借鉴实验指导中的单条中间代码的数据结构定义，将Operand和InterCode结构定义如下：

```
struct Operand_ {
    enum { VARIABLE, TEMP, PARAMETER, CONSTANT, LAB, FUNCT } kind;
    enum { VAL, ADDRESS} type;
    union {
        int var_no;
        long long int value;
        char* func_name;
    } u;
};

struct InterCode {
    enum { LABEL, FUNCTION, ASSIGN, ADD, SUB, MUL, DIV, GOTO, IF, RETURN, DEC,
ARG, CALL, PARAM, READ, WRITE } kind;
    enum { NORMAL, GETADDR, GETVAL, SETVAL, COPY } type;
    union {
        struct {
            Operand *left, *right;
        } assign; // 赋值
        struct {
            Operand *res, *op1, *op2;
        } binop; // 双目
        struct {
            Operand *res, *op;
        } sinop; // 单目
        struct {
            Operand* op;
        } single;
        struct {
            Operand *op1, *op2, *target;
            char relop[4];
        } cond; //条件
        struct {
            Operand* op;
            unsigned size;
        } dec; // 定义
```

```
    } u;
};


struct InterCodes {
    InterCode code;
    bool isDelete;
    InterCodes *prev, *next;
};
```

操作数（Operand）中，`kind` 枚举指定了操作数的类型，可以是变量（VARIABLE）、临时变量（TEMP）、参数（PARAMETER）、常量（CONSTANT）、标签（LAB），或函数名（FUNCT）；`type` 枚举指定了操作数是指令中使用的值（VAL）还是地址（ADDRESS）；union u 包含了根据 `kind` 和 `type` 的不同可能存储的不同数据类型的联合体。

中间代码指令InterCode中，`kind` 枚举指定了中间代码指令的类型，`type` 枚举指定了指令的操作方式，`union u` 包含了根据不同 `kind` 和 `type` 可能需要的不同数据结构的联合体。

InterCodes表示一个中间代码链表节点，方便遍历和管理中间代码序列。

在intercode.c中，实现了类似于newTemp的newConst和newLabel函数来生成一些Operand，同时也有类似于createAssign的createBinop和createSinop等生成中间代码节点的函数，方便后续具体中间代码生成过程中的调用。

具体的语义分析过程在inter.c当中，先进行一些初始化，将语义分析得到的符号表中的每一个符号都初始化一个相应的operand值。之后便是从root节点开始逐个进行分析和生成。

`setVariable()`：这个函数遍历符号表中的所有变量，为每个变量生成一个操作数（`Operand`），并分配一个编号。变量的类型可以是常规变量、数组或者结构体字段。数组和结构体的变量会被标记为“地址类型”（`ADDRESS`），其他则是值类型（`VAL`）。函数内还处理了数组变量的大小计算，并为每个变量生成相应的操作数。

`tranFunDec()`：处理函数声明。首先创建一个函数操作数，表示该函数的名称，随后根据函数是否有参数列表，调用 `tranVarList()` 来处理参数列表。

`tranCompSt()`：处理函数体，包括局部变量定义（`tranDefList()`）和语句列表（`tranStmtList()`）。

`tranVarDec()`：处理变量的声明，根据语法树中变量声明的类型（普通变量、数组变量）以及上下文（函数参数、结构体字段），生成相应的中间代码操作数，并将变量的信息（如大小、类型）存入符号表。主要的分支处理了普通标量变量和单维数组的声明，数组声明如果是多维数组，则会报错

`tranDec()`：处理具体的变量声明。如果变量为数组类型，函数会递归地处理数组元素的初始化。如果变量是普通的基础类型，函数直接创建变量并赋值。

`tranStmt()`：处理不同类型的语句，如赋值语句（`ASSIGNOP`）、控制语句（`IF`、`WHILE`）、返回语句（`RETURN`）等。每种语句都会生成不同的中间代码。其中控制流语句即是遵照实验指导中的内容来生成LABEL以及相应的GOTO跳转语句

`tranCond()`：用于处理条件判断语句，如 `IF`、`WHILE` 语句中的条件表达式。它会生成相应的跳转指令（`GOTO` 或 `LABEL`）以控制程序流程。

`tranExp()`：处理表达式的翻译。其处理方式主要可分为以下几类：(1)处理赋值表达式，对简单变量先查找变量对应的符号信息，然后计算右侧表达式的值，并将其赋给左侧的变量。对数组变量，计算数组元素的地址和大小，然后对数组元素进行赋值。对结构体变量则是通过结构体字段偏移量进行赋值；(2)逻辑运算，会先生成条件跳转（`tranCond`），根据表达式的真假值设置标签，并将结果存储在`place`中；(3)算术运算，对于加减乘除等算术运算，首先会分别计算左右操作数，然后根据运算符（`ADD`，`SUB`，`MUL`，`DIV`）生成相应的二元运算指令；(4)函数调用，通过`tranArgs`处理参数列表，然后根据函数名称生成相应的调用指令。如果是`write`，则会生成特定的调用指令；(4)数组访问，首先计算索引位置，然后根据数组的元素大小，计算最终元素的地址，并将其存储到`place`中；(5)结构体字段访问，根据字段偏移量计算相应的地址，并将字段值或地址返回；(6)变量或常量，直接将其值或地址分配给`place`。