ISE lab2

221900073 孙佳琪

程序所实现的功能

在词法分析和语法分析程序的基础上编写一个程序，对C□□源 代码进行语义分析和类型检查，并打印分析结果。与实验一不同的是，实验二不再借助已有的 工具，所有的任务都必须手写代码来完成。另外，虽然语义分析在整个编译器的实现中并不是 难度最大的任务，但却是最细致、琐碎的任务。因此需要用心地设计诸如符号表、变量类型等 数据结构的实现细节，从而正确、高效地实现语义分析的各种功能。

如何被编译

1. make
2. ./parser 测试文件

个性化的内容

在语法树的node结构体部分，对原来的结构体进行一些补充，新加入指向父节点的指针以及指向继承属性和综合属性的指针，并引入instruct变量来表示该节点是否在struct中，以方便后续文件中展开语义分析。

```
typedef struct Node_ Node;


struct Node_ {
    char* name;
    MyType type;
    int line;
    int no;
    union {
        unsigned type_int;
        float type_float;
        char type_str[40];
    } val;
    FieldList* syn;
    FieldList* inh;
    Node* parent;
    Node* child;
    Node* next;
    uint8_t instruct;
};
```

同时，使用如下代码来建立类型，符号，以及符号表：

```
struct Type_ {
    enum { INT, FLOAT, ARRAY, STRUCTURE, STRUCTVAR, FUNC, WRONGFUNC } kind;
```

```
    //基本类型
    int basic;
    //数组：元素类型，数组大小
    struct
    {
        Type* elem;
        int size;
    } array;
    //结构体：链表
    FieldList* structure;
    FieldList* structvar;
    //函数：参数链表，返回类型
    struct
    {
        Type* ret;
        FieldList* args;
    } function;
    ;
};

struct FieldList_ {
    char* name; //域的名字
    Type* type; //域的类型
    FieldList* tail; //下一个域
};

struct TableList_ {
    char* name;
    Type* type;
    TableList* next;
};
```

其中kind表示该结构体的类型，而对于稍微复杂一些的结构体（数组 链表 函数），在Type中也有相应的指针或是结构体来表示其的特有属性，如上面代码的注释所示，在此就不多赘述。

具体的语义分析过程在semantic.c当中，对每一个产生式都进行处理，其中涉及到类型或者参数传递的地方就需要用到node中新加入的继承属性或综合属性来进行传递。以下举例说明一些函数中的大概处理流程：

ExtDefList的三种情况中，对于变量声明，需要传递 `inh` 属性到 `Specifier` 和 `ExtDecList` 中，以支持类型推导；对于结构体声明，只需要调用 `Specifier` 进行类型处理；对于函数定义，此时需要初始化 `Function` 类型的 `Type` 并设置返回类型。然后调用 `FunDec` 来解析函数的参数列表，并调用 `CompSt` 处理函数体的复合语句。

Specifier的两种情况中，对于TYPE，直接设置Type结构体中的kind属性；对于StructSpecifier，初始化 `STRUCTURE` 类型，并将 `inh` 传递给 `StructSpecifier` 子节点，完成结构体的处理。

Def函数中，首先检查 `node->inh` 是否为NULL，确保这个节点有前驱属性，如果 `node->instruct` 为 1，说明当前是一个结构体成员定义，则将 `instruct` 属性传递给孩子节点，确保这些孩子节点也被标记为结构体成员，传递继承属性 `inh` 给第一个子节点并调用 `Specifier` 函数进行处理，之后传递 `syn` 属性给第二个子节点 `DecList`，最终 `DecList` 的属性赋给 `node->syn`，以便后续使用。

DecList函数中，先检查 `node->inh` 是否存在，并继承 `instruct` 属性（如果有的话）传递给第一个子节点，再检查继承属性 `inh` 是否是结构体类型，且未定义具体结构体，此时会抛出错误17，如果有不止一个声明，需要递归处理子节点 `DecList`，并将当前的 `syn` 列表尾部指向下一个 `DecList` 的 `syn`。

FunDec函数中，首先查询函数名是否已存在于符号表中以检查重定义，若重定义则报错并设置错误类型。接着解析参数列表 `VarList` 并更新参数类型链表。

代码中在各函数内置了大量的错误检查，通过对类型不匹配、变量重定义、未定义结构体等情况报错。例如，在 `VarDec` 中，如果变量已经在当前作用域定义过，则输出错误提示“变量重定义”（错误类型3）；在 `ParamDec` 中如果参数声明不合法则标记为错误函数。