编译原理实验二报告

语义分析

1190200122 袁野

一、实现功能

程序实现了对 C—源代码的语义分析，完成了错误类型 1 到错误类型 18 的识别。

1、实验一基础上的改进

1.1 为了方便起见，将构建语法树所依赖的函数和变量新整合成了一个外部库 tree.h，方便 semantic.c 在进行语义分析的过程中调用相关函数。

1.2 此外加入了函数声明的语法生成式来表示函数的声明：ExtDef->Specifier FunDec SEMI

2、符号表

这里首先使用 Type 来表示符号名的类型，包含基本类型(int 和 double)、数组、结构体和函数，对应的，我们用一个联合体来表示各个不同的类型对应的基本信息，比如基本类型的取值、数组的类型和大小、结构体的域、函数的参数、域等。

```
struct Type_ {
    enum {BASIC,ARRAY,STRUCTURE,FUNCTION} kind;
    union{
        int basic;//基本类型 0表示int 1表示float
        struct {Type elem;int size;} array;//元素类型+数组大小
        FieldList structure;//结构体类型
        FUN function;
    }u;
};
```

对于域，我们用 FieldList 来记录定义域里的每一个符号的相关信息，并将一个域的符号挂成一个链表。

```
struct FieldList_ {
    char* name;
    Type type;
    FieldList tail;
};
```

而对于符号的映射我们用哈希表来实现，通过实验指导书中的 hash_pjw(char* name)函数将符号映射为一个小于 16383 的正整数，而对于哈希冲突的符号采取挂链的方式将他们挂在一起处理。在哈希表中我们除了记录其符号，还需要记录其类型、定义域，行号（输出错误用）。

```
struct TABLE_ {
    int is_def_struct;
    FieldList field;
    TABLE next;
    int linenumber;
};
```

3、函数实现

3.1 Program()

调用 ExtDef(),在所有递归调用完成返回后检查 error 18，是否存在函数只声明无定义。

3.2 Specifier()

由 TYPE 类型生成，则直接设置 type=int/float，由 StructSpecifier 类型生成，则调用 StructSpecifier()函数。

3.3 StructSpecifier()

3.3.1 STRUCT OptTag LC DefList RC

读取 OptTag 中的 ID 信息，生成对应的结构体名称，判断该命名是否已被使用(error 16)，调用 DefList(judge=0)函数生成域，将该 struct 加入符号表中。

3.3.2 STRUCT Tag

读取 Tag 中的 ID 信息，在符号表中查找，找到则返回该结构体对应的 Type，未找到则对应 error 17。

3.4 DefList()/DecList()/VarList()

根据产生式递归调用，并将相继产生的 FieldList 链接为链表。

3.5 ExtDecList/StmtList

根据产生式递归调用。

3.6 Dec

3.6.1 VarDec：VarDec( )

3.6.2 VarDec ASSIGNOP Exp：根据传递的 judge 值，结构体定义(judge=0)对应 error 15(定义时对域进行初始化)，否则调用 Exp()，若返回值与 type 不同，则对应 error 5(赋值号两边表达式类型不匹配)。

3.7 VarDec

3.7.1 VarDec LB INT RB：设置当前 vardec_type 为 ARRAY，并向上传递调用 VarDec(child,vardec_type,judge)，即递归生成的下一个 FieldList 的 type 类型为 vardec_type。

3.7.2 ID：创建一个 FieldList，读取 ID 的值,对于函数声明的参数，直接返回 type，不加入符号表；对于函数定义的参数或变量定义：符号表中查重(error 3)，不重复则加入符号表；对于结构体的定义：符号表中查重(error 15)，不重复则加入符号表。

3.8 FunDec

创建一个 TABLE 元素，设置对应的 type/name 等值，判断是否第一次定义(加入哈希表)，对应多次定义(error 4)

3.9 Stmt

3.9.1 Exp SEMI：直接调用 Exp()并返回。

3.9.2 CompSt：直接调用 CompSt()并返回。

3.9.3 RETURN Exp SEMI：调用 Exp()，比较其返回值与 type 的值，若不同则对应 error 8(return 语句的返回类型与函数定义的返回类型不匹配)

3.9.4 IF/WHILE 语句：调用 Exp()，若 Exp 不为 int 类型，则 error 7(操作数类型不匹配)，若匹配则调用 Stmt()

3.10 Exp

3.10.1 INT/FLOAT：直接生成对应 type 并返回。

3.10.2 ID：在哈希表中查找符号表是否存在，不存在则对应 error 1(变量在使用时未经定义)，存在则直接返回对应的 Type

3.10.3 MINUS Exp：直接调用 Exp()分析第二个符号。

3.10.4 NOT Exp：调用 Exp()后，如果其 Type 是 INT 则将该 Type 正常返回，否则对应 error 7(类型与操作不匹配)。

3.10.5 ASSIGNOP：判断左端 Exp 是否满足其为基础数据类型、结构体、结构体某

一元素、数组，不满足则对应 error 6(赋值号左边出现一个只有右值的表达式)，若左右 Exp 类型不同，则对应 error 5(赋值号两边的表达式类型不匹配)。

3.10.6 AND/OR：调用 Exp()，若不是 int 类型，则对应 error 7。

3.10.7 RELOP/PLUS/MINUS/STAR/DIV：若左右 Exp 类型不同，则对应 error 7(操作数类型不匹配)

3.10.8 Exp DOT ID：判断左端 Exp 是否为 STRUCTURE 类型，不满足则对应 error 13(对非结构体型变量使用“.”操作符)，判断右端 ID 是否为定义过的域，不满足则对应 error 14(访问结构体中未定义过的域)。

3.10.9 Exp LB Exp RB：判断左端 Exp 是否为 ARRAY 类型，不满足则对应 error 10(对非数组型变量使用“[]”操作符)，判断右端 Exp 是否为 int 类型，不满足则对应 error 12(数组访问操作符“[]”中出现非整数)

3.10.10 LP Exp RP：直接调用 Exp()并返回即可。

3.10.11 ID LP Args RP/ID LP RP：判断左端 ID 是否在 table 表，不在或者是一个为定义过的结构体名称对应 error 2(函数在调用时未经定义)。接下来判断其类型是不是 FUNCTION，不是的话在则对应 error 11(对普通变量使用“()”操作符)。最后调用 Args()并判断函数参数是否与符号表中的参数列表相同，不符合则对应 error 9(函数调用时实参与形参的数目或类型不匹配)。

3.11 Args

用 Exp()分析产生式右侧第一个符号，并递归使用 Args()分析右侧的符号，将其连接成为一个链表，然后将第一个节点返回。

二、编译过程

通过借助网上的参考资料，构建了一个 makefile 文件来编译这个程序，命令如下：

make 编译该项目并生成 parser 可执行文件

make test 测试实验指导书中的前 18 个样例

make clean 删除所有编译过程中产生的新文件