编译原理实验报告

Lab2 语义分析

151220069 罗义力
luoyl233@foxmail.com

一、已完成功能

1.语义分析:正确判断 17 种错误类型
2.选做 2.1：函数在定义之外可以进行声明,增加错误类型 18 和错误类型 19 的判断

二、实验环境

GNU Linux Release: Ubuntu 12.04, kernel version 4.4.0-66
GCC version 5.4.0
GNU Flex version 2.6.0
GNU Bison version 3.0.4

三、使用说明

make: 编译生成分析器 parser。
make clean: 清除所有的中间文件和目标文件。
make test: 对 Makefile 文件中规定的样例测试。
可以一次传入多个文件名，输出多个文件的测试结果，如：

./parser <file1> <file2> ……

四、实验过程

1.表示各种类型的数据结构

我的数据结构参考了实验文档，Type、Array 和 Structure 类型基本和文档一致，另外自己添加的数据结构是函数类型 Func 和符号类型 Symbol。

```
struct Func_ {
    Type retType;           //返回类型
    FieldList argList;      //形参表
    int isDefined;          //是否被定义
};
```

函数类型未设置函数名，因为考虑到如下的符号类型已经提供了 name。

```
typedef enum S_TYPE {S_Type, S_Func, S_StrucDef} S_TYPE;
struct Symbol_ {
    S_TYPE kind;            //变量类型、函数或者结构体定义
    char *name;             //符号名字
    int lineno;             //符号定义行号
    union {                 //指向函数或者类型，具体由 kind 决定，结构体定义包含
        Type type;          //在 type 中
        Func func;
    };
    Symbol next;            //多个符号有相同的哈希值，指向相同的位置，tail 域依次
```

};　　//连接形成桶

2. 符号表

符号表通过散列表实现，预先定义一个大数组，通过文档提供的 hash 函数计算 hash 值，指向数组对应位置，再进行插入、删除和查找操作。所有的变量、结构体定义或者函数都是存放在同一个表中的，通过上述的 Symbol 类型实现多种类型的统一。

3. 语义分析过程

语义分析是一个对语法树遍历的过程，只有对没有任何词法和语法错误的语法树，语义分析才会进行。首先，程序会初始化符号表，调用 initTable()初始化哈希表。然后通过函数 sematicCheck(root)进入真正的语义分析过程（root 为语法树的根节点）。语义分析结束以后，需要清空语法树和符号表，依次调用 deleteTree(root)和 clearTable()。

下面着重说明的是语义分析过程。对上下文有关文法处理使用属性文法，但是在 lab1 语法分析的基础上再增加语义分析，会使整个代码非常庞大，难以理解。因此，语义分析是对已经建好的语法树结构进行遍历分析。对比之前的上下文无关文法（词法分析和语法分析），属性文法需要使用到综合属性和继承属性。综合属性在程序里面可以相当于子节点分析程序的返回值，供父节点使用，如各种类型，或者变量的定义；而继承属性，则是将父节点或者兄弟节点分析的结果作为参数传给子节点分析程序，如变量定义时，需要首先从兄弟或则父母节点获取变量的类型。

根据语法树结构，语义分析部分使用了多个函数对不同类型的节点进行处理，如下：

- High-level Definitions：Program，ExtDefList，ExtDef，ExtDecList
- Specifiers：Specifier，StructSpecifier
- Declarators：VarDec，FunDec，VarList，ParamDec
- Statements：CompSt，StmtList，Stmt
- Local Definitions：DefList，Def，DecList，Dec
- Expressions：Exp，Args

每个函数处理对应名字的节点，不同的错误会在一个或多个函数里面处理；对文法中出现的递归定义，部分过程也会进行递归处理，比如数组定义。

函数详细的定义在文件 sematic.h 和 sematic.c 中。

4. 要求 2.1（函数除了在定义之外还可以进行声明）

1) 给 ExtDef 增加新的产生式，语法分析不会对函数声明报错：

$$\text{ExtDef} \rightarrow \text{Specifier FunDec SEMI}$$

2) 在函数类型 Func_中增加 isDefined 域，说明函数是否被定义

3) 在 FunDec 中增加对函数声明和定义的处理，判断多次声明是否相互冲突（错误类型 19）

4) 在语法分析过程结束以后，扫描符号表中的函数，若 isDefined 为 0，说明函数声明但未定义（错误类型 18）。

5. 关于调试

最简单直接的调试方式还是 printf()函数，但调试结束之后要一个个删除就很麻烦。因此我利用 C 语言的条件编译，定义了一系列调试的宏，只需要修改一个值就能自动开启关闭调试输出，非常方便。

调试的宏定义在 debug.h 文件中。