编译原理实验 - 语义分析

161220084 刘笑今

161220084@smail.nju.edu.cn

1 实验环境

- GNU Linux Release: Ubuntu 18.04.1
- G++ version 7.3.0 (C++ 11)
- GNU Flex version 2.6.4
- GNU Bison version 3.0.4

2 编译与运行方法(make)

- make parser： 生成分析器，并拷贝到上一目录；

- make test： 对 Test 目录下的所有*.cmm 文件进行分析；

- make clean： 删除 make parser 过程中生成的所有中间文件。

3 分析器功能

可以通过遍历语法分析创建的语法分析树来对没有语法错误的C--源程序进行语义检查。语义检查的依据是实验讲义中的语义假设和所有的错误类型。

其中，支持了函数声明。即：在 C--程序中可以进行函数声明，但须满足：函数声明必须具有对应的函数定义；同一函数的多次声明必须一致。

分析器可以对不符合语义要求的语句进行报错，并支持同一程序报多处错误。

4 分析器实现简介

4.1 符号表的维护

符号表需要记录当前遍历到的语句之前的所有 ID，包括函数名、结构体名、变量名。这里通过 C++ STL 的 map 来实现。其中，为了实现选做 2.1 的内容，将函数名部分维护成一个函数声明 map，一个函数定义 map。在遍历完整个语法树后，需要检查是否有只有

声明没有定义的函数。

4.2 语法分析树的遍历

大体上，语法分析树是一个后序遍历。其理论支持是 L-属性文法。在遍历是，对需要进行操作的节点进行处理。这里的进行操作是指：需要插入符号表；需要计算综合属性/继承属性；有语义错误需要检查。

4.3 自定义类型(多维数组、结构体)

多维数组采用讲义所提到的链表方式存储。处理时需要递归处理。

```
struct {
    Type *elem;
    int size;
} arr_type;
```

要记录结构体的所有信息，只需要将结构体的名字，以及其所有的域记录下即可。这里通过 vector 来记录所有的域。

```
Struct {
    string name;
    vector<Symbol> fields;
} struct_type;
```

其中，Type 和 Symbol 是自定义的类型，分别记录一个类型和一个变量的所有语法和语义信息。