编译原理实验报告二

161220070 李鑫烨

完成的功能点

在属性文法的理论基础上，借助于变量类型以及符号表等数据结构，在语法分析过程中对符号表进行插入、查询等操作，完成对 C–源码的语义分析和类型检查。在完成对变量未定义、类型不匹配等内容同时，完成了以下内容

- 在语法分析和语义分析层面上支持对函数的声明。
- 修改符号表结构，支持嵌套作用域。
- 修改类型等价机制为结构等价机制。

实现思路

实验二的实现思路并不复杂，但错误类型检查等细节处理较为琐碎。具体实现思路在于

1. 设计存储类型、变量及函数等信息的数据结构，并封装判断函数、类型是否等价等操作。
2. 设计符号表数据结构，存储已定义变量、函数以及已声明类型等信息，封装插入与查询等操作，同时为了支持嵌套作用域，封装进入作用域与离开作用域操作。
3. 对语法分析树进行后根遍历，借助符号表对程序进行语义分析和类型检查。这本质上是 L 属性文法中每个节点的继承属性与综合属性计算的过程。

数据结构表示

由于采用 C++ 完成实验二，在类型、变量与函数对应的数据结构设计中采用基于对象的设计方法，在符号表的设计过程中较多地运用了 STL 中提供的容器。

Type 类用于存储和类型有关的信息，并封装判断类型等价操作。Type 类设计难度在于实验中类型除了内置数据类型之外存在数组类型和结构体类型。若采用 C++ 中继承的思路，将 Type 与具体的数据类型设计为基类与派生类的关系，那么在 Type 中则需要设计过多冗余的接口，并且虚函数动态绑定的机制决定之后用到 Type 之处均需定义为指针类型，内存管理繁琐。解决方案为在 Type 类内部添加指示变量，以指示 Type 当前状态。在此基础上封装判断类型等价操作，若指示变量则显然不等价，否则进行进一步判断。

进一步深究 Type 为不同具体类型时的数据结构。Type 为内置数据类型时，只需设置枚举变量以区分 int 与 float，相对简单。Type 为数组类型时，需要记录数组的长度和维度以及元素类型，采用了近似于递归的方法，构造了数组类型的链表，链表长度即为数组类型维度。

```
    struct {
        Type *elem;
        size_t size;
    } array;
```

结构体类型中，将其中域同样视为变量，实现了一定程度的代码复用。因此结构体类型只需存储名字与域的集合，通过 Type 与 Symbol 的复用来消解部分复杂性。

```
    struct {
        string name;
        vector<Symbol> fields;
    } structure;
```

Symbol 类用于存储变量相关信息，相对 Type 类设计较简单。Symbol 类只存储变量名、行号以及变量类型。为了判断结构体等价以及之后判断函数参数列表等价方便，封装了变量等价操作，两变量只需类型相等则判断为等价。

Function 类用于存储函数相关信息，包括函数名，返回类型与参数列表。为了支持函数声明，添加布尔变量区分声明与定义。返回类型用 Type 类进行定义，参数列表同样采用类似于结构体类型处理域的方法，定义为 Symbol 的集合。

```
    class Function {
        string name;
        Type ret;
        vector<Symbol> args;
        bool def;
  };
```

符号表数据结构的设计需要较多地考虑效率。考虑到函数定义和结构体类型的声明均为全局，因此不考虑作用域，运用 STL 中 map 容器，以函数名或结构体类型名为索引，取到其对应的数据结构。而存储变量时考虑嵌套作用域，因此采用栈结构管理作用域，具体实现如下

```
    class Scope {
        map<string, Symbol> symbols;
    };
    class SymbolTable() {
        vector<Scope> scopes;
        map<string, Type> decTypes;
        map<string, Function> decFunc;
    };
```

编译运行方法

在 lab2/Code 中输入以下指令，

- make parser: 生成可执行文件 parser 并替换上级目录中 parser。
- make run: 已生成可执行文件 parser 条件下，测试 lab2/Test 中所有用例。
- make clean: 删除可执行文件及中间文件。

实验总结

实验二相对一而言，需要处理的细节较多，错误类型处理也较为琐碎，因此工作量较大。但背后思路实际上非常清晰，是一个基于 L 属性文法的语法分析树后根遍历的过程。有些处理上，比如判断表达式是否为左值，显式地定义了左值这一综合属性。而在处理某些产生式过程中，比如产生式

$$Def \ := \ Specifier \ DecList \ \text{SEMI}$$

我们需要将 Specifier 所对应的 Type 传递给 DecList 乃至于由 DecList 推导出的 Dec，这实际上是一个隐式地计算继承属性并向下传递的过程。从这个角度来观察能找到语义分析过程背后理论支撑。