编译原理 lab3 报告

191250045 侯为栋.

本次作业又双叒叕经历了痛苦的 debug 过程, 而且这次还不是一般的痛苦:

44 次提交, 24 次 2800 分以上, 不时还来个 0 分超时.

lab3 整整做了 4 天, 编码时间 8 小时, debug 时间至少 15 小时. 主要有很多时候都是虚空 debug, 找到一个可能不合理的地方后被告知这个地方不考.

> 当然助教哥哥真是太负责了, 无论什么奇葩问题都会回答我呜呜呜

Implement

src 文件夹内容如下:

```
src
├── ArrayT.java
├── CmmLexer.g4
├── CmmParser.g4
├── CmmSemanticVisitor.java
├── ErrorT.java
├── ErrorType.java
├── FieldList.java
├── FloatT.java
├── FunctionT.java
├── IntT.java
├── Kind.java
├── Main.java
├── ParseInfo.java
├── StructureT.java
├── Symbol.java
├── SymbolTable.java
└── Type.java
```

类型系统

类型系统通过枚举类 Kind, 抽象父类 Type 和 Type 一系列子类实现.

Kind

Kind 是类型的枚举, 包括 INT, FLOAT, ARRAY, STRUCTURE, FUNCTION, ERROR.

Type

Type 只保留一个 Kind 实例方便判断自身类型, 以及一个用于判断类型相等的抽象方法强制子类实现:

```
protected final Kind selfKind;
public abstract boolean isEquivalentType(Type t);
```

子类包括 ArrayT, ErrorT, FloatT, IntT, StructureT, FunctionT.

StructureT 和 FunctionT 引入 FieldList 类来辅助存储参数.

ErrorT

ErrorT 是错误类型, 如果子树上发生错误而且不能直接忽略的情况下, 会返回一个带有 ErrorT 的 info.

例如在定义函数时返回类型是一个错误类型, 此时不能直接忽略整个函数, 就需要存储它的返回值为 ErrorT.

FieldList

StructureT 和 FunctionT 类似, 都需要存储一系列参数, 这通过 FieldList 类实现.

FieldList 是链表节点和链表的 mix 类, 它字段是链表节点的构成, 但是方法上却又向链表看齐.

例如它的 add 方法, 一直沿着 next 寻找到末尾节点并将新节点添加在那里.

符号表

符号表又两部分组成, 符号表类 SymbolTable 和 符号类 Symbol.

SymbolTable 就是一个 hash 表, 通过一个 getIndex 函数来将 key 映射为数组下标.

Symbol 类和 FieldList 相似: 都是一个链表类, 在这里是为了解决散列冲突问题.

Symbol 只存储符号的 name 和类型, 并不关心它的作用域, 这也是本次作业的要求.

CmmSemanticVisitor

访问者类, 通过自主控制遍历方式来进行语法检查.

它的构造方式类似 CmmBaseVisitor:

```
class CmmSemanticVisitor extends AbstractParseTreeVisitor<ParseInfo>
implements CmmParserVisitor<ParseInfo> {}
```

实现了所有的 visit 方法, 主要的行为是在合适的地方进行语义检查, 以及将符号添加入符号表.

为了解决父节点向子节点的传参问题, 使用 ParseTreeProperty 类保存参数.

子节点通过 getCallParam 方法来获得参数:

```
private ParseInfo getCallParam(ParserRuleContext ctx) {
    return this.values.get(ctx.getParent());
}
```

ParseInfo

ParseInfo 是自定义信息类, 用于返回 visit 子节点后得到的信息以及承载父节点向子节点传递的信息.

其包含以下字段:

```
private FieldList f;
private Type t;
private String s;
private boolean structScope;
private boolean isRightVal;
```

不是每个子节点返回时都需要填满这些信息, 通常是需要填什么填什么.

structScope 是判断当前是不是在 struct 内, 用于判断可否进行定义时赋值.

isRightVal 是判断当前节点的返回值是否是右值, 如果是则不能出现在赋值号左边.

t 如果是 ErrorT 则表明子节点出了错误, 此时大部分情况下父节点会同样返回一个错误但不会额外报错.

其他

其实这次因为小 bug 太多反而有些记不过来了.

但是本次 debug 流程和前两次不太一样: 前两次大多是构造用例而后发现错误, 但这次真正构造用例看出来的 bug 非常少, 主要靠我对代码一行一行检查分析发现笔误或者是逻辑问题.

印象深刻的是最后一次 bug, 我构造的用例如下:

```
struct S {
struct Y {} y;
struct {} x;
};

int main() {
    struct y z;
    struct x m;
}
```

在我的实现中, 并没有很好地区分 struct 类型和 struct 实例, 所以上述用例不会报错.

这个 bug 从一开始就陪伴着我: normal test 1 一直是 75 分. 最后才把它给找出来, 累死我了.

此外, 本次的代码经历了一次大规模重构. 也不能说重构吧, 直接就是重写了.

我一开始用的是 listener, 快写完的时候发现难以实现跳过某些节点的功能, 于是中途转 visitor 了.