北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

本科生《编译原理》课程实践报告

题　　目： 语义分析实验

学　　院： 徐特立学院

专业名称： 计算机科学与技术

姓　　名： 陈照欣-1120191086

实验目的

（1） 熟悉 C 语言的语义规则，了解编译器语义分析的主要功能；
（2） 掌握语义分析模块构造的相关技术和方法，设计并实现具有一定分析 功能的 C 语言语义分析模块；
（3） 掌握编译器从前端到后端各个模块的工作原理，语义分析模块与其他 模块之间的交互过程。

实验内容

语义分析阶段的工作为基于语法分析获得的分析树构建符号表，并进行语义检查。如果存在非法的结果，请将结果报告给用户，其中语义检查的内容主要包括：
（1） 变量使用前是否进行了定义；
（2） 变量是否存在重复定义；
（3） break 语句是否在循环语句中使用；
（4） 函数调用的参数个数和类型是否匹配；
（5） 函数使用前是否进行了定义或者声明；
（6） 运算符两边的操作数的类型是否相容；
（7） 数组访问是否越界；
（8） goto 的目标是否存在；
（9） 函数是否有返回值
本次语义检查实验主要实现了前三个以及最后一个内容。

实验步骤

1. 错误类型的声明

本实验定义了三种错误类型：变量未定义、变量重复定义和在循环外使用 break。错误使用字符串链表表示。对于每一个错误都生成对应的语句并且构成链表。最后使用 output 函数将所有的错误按照顺序输出。

```
public void output() {
    if(lis.size()==0)
        return;
    System.out.println("Error:");
    for (Object li : this.lis) {
        System.out.println((String) li);
    }
}
```

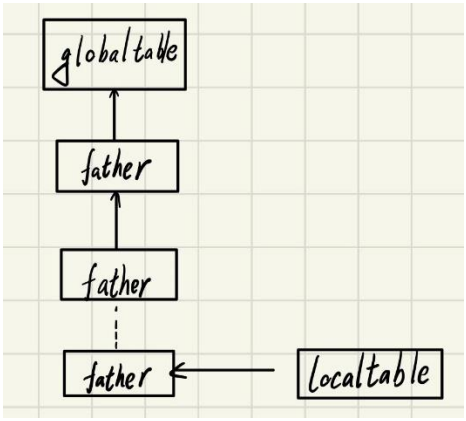
2. 符号表的构建

符号表主要由两个元素构成：father 和 items。Father 也是一个符号表，用来记录并恢复原先符号表状态；items_name 以及 items_type 是一个列表，

存放变量名称和对应的类型。利用 Linkedlist 的有序性可以方便地判断变量是否定义或者重复定义。

3. AST 语法树的遍历

TableVistor 类实现了 ASTVisitor 接口，通过 TableVisitor 对象可以实现 AST 节点的访问。维护了四个变量：localtable 局部变量表、globaltable 全局变量表、errorType 和 iteration_level 循环层数。对于每一个节点的访问，会先将 localtable 赋给节点的 scope 属性用以标志其范围，再根据 AST 节点的属性对其每一个子节点进行深度优先遍历。在进入下一个复合语句时，都会将当前 localtable 复制给 father 属性，自己再进行更新。由此构成了一个变量链表：



解决方案：

1. 对于变量未定义

遇到每一个 ASTIdentifier 时，调用 symbolTable 类中的 find_cur 函数，在全部范围内的变量表中查找对应变量。若返回结果为 false，则执行 ADDErrorS01 报错。

2. 对于变量重复定义

对于每一个 ASTDeclaration，同样需要查找变量表，但是与变量未定义的情况不同，只需要在当前局部变量表中查询变量即可。

3. 对于 break 的判断

在进入 ASTIterationStatement 之前，iteration_level 会加一，访问完成后减一。

```
this.iteration_level++;
this.visit(iterationStat.stat);
this.iteration_level--;
```

进入 breakstatement 时，根据 iteration_level 是否为 1 判断 breakstatement 是否出现在循环体内。

```
public void visit(ASTBreakStatement breakStat)throws Exception{
    if(breakStat == null)
        return;
    if(iteration_level==0){
        errorType.addES03();
    }
}
```

4. 对于 return 的判断
   维护了一个全局变量 judge_return 在进入 ASTCompoundStatement 节点之前，judge_return 设置为 false，倘若在非 void 函数中进入了 ASTReturnStatement，则将 judge_return 设置为 true。

```
public void visit(ASTReturnStatement returnStat)throws Exception{
    if(returnStat == null)
        return;
    if(returnStat.expr != null) {
        for(ASTExpression expr : returnStat.expr) {
            visit(expr);
            if(expr!=null)
                judge_return = true;
        }
    }
}
```

实验结果

0_var_not_defined.c

```
3. Parsing Finished!
Error:
ES01 > Identifier "a" is not defined
```

1_var_defined_again.c

```
3. Parsing Finished!
Error:
ES02 > Declaration "a" is defined
```

2_break_not_in_loop.c

```
3. Parsing Finished!
Error:
ES03 > BreakStatement must be in a loop statement
ES03 > BreakStatement must be in a loop statement
```

实验心得

这次实验最大的困难应该是老师这次没有给示例代码，从获取节点、节点访问到表项设计、建表等都要自己一步步完成。一开始其实踩了个比较大的坑，就是把第一类错误和第二类错误设置为了同一类 find 函数，导致不允许在局部变量表中定义一个全局变量中的同名变量。此外由于时间关系，类型判断也不全面，不能适应更多更复杂的代码。

通过这次实验，对语义分析的流程和方法有了更清楚的认识，对符号表的认识也更具像化了，了解到符号表应具有的信息以及添加表项、查找表项的流程，更加深入的理解了课本中的知识。