Lab.6 语义分析实验报告

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：

- 变量或函数使用前是否进行了定义；
- 变量或函数是否存在重复定义；
- break语句是否在循环块中使用。

实验过程

实验思路

通过对AST树进行深度优先遍历，建立作用域和函数表，并设立inLoop标志，用以检查三种错误。

*AST*树

语义分析的处理对象是语法分析生成的AST树。本次实验使用的是BITMiniCC框架自动生成的AST树。

作用域

创建VarTable类，用以标识变量。类定义如下：

```
public class VarTable {
    public String name;
    public ASTExpression value;
}
```

创建对象`private List<VarTable> Scope = new LinkedList<>()`用以保存当前作用域内已声明的变量；

创建对象`private List<VarTable> GlobalVarTable = new LinkedList<>()`用以保存全局变量；

创建栈`private Stack<List<VarTable>> Var = new Stack<>()`用以保存所有作用域。

由于对AST树进行深度优先遍历，所以作用域可以用栈保存。当一个作用域遍历完时，就将栈顶作用域弹出，并更新Scope。

函数表

创建FuncTable类，用以保存函数的先关信息。类定义如下：

```
public class FuncTable {
    public List<VarTable> varTables = new LinkedList<>();
    public String funcName;
}
```

创建对象`private List<FuncTable> ProcTable = new LinkedList<>();`用以保存已定义的函数。

*inLoop*标志

创建初值为`false`的布尔型变量`inLoop`，用以标识当前是否在循环中。

部分代码

MySemanticAnalyzer.java

```
package bit.minisys.minicc.semantic;

import bit.minisys.minicc.parser.ast.*;

import java.util.List;
import java.util.*;

public class MySemanticAnalyzer {
    private List<VarTable> GlobalVarTable = new LinkedList<>();
    private List<FuncTable> ProcTable = new LinkedList<>();
    private Stack<List<VarTable>> Var = new Stack<>();
    private StringBuilder errorInfo = new StringBuilder();
    private boolean inLoop = false;

    public void GetfuncDeclaration(FuncTable funcTable, ASTFunctionDeclarator
declarator){}

    public void GetVarTable(VarTable varTable, ASTInitList node){}

    public void visit(ASTCompilationUnit astCompilationUnit){}

    public void visit(ASTFunctionDefine astFunctionDefine){}

    public void visit(ASTExpression expression){}

    public void visit(ASTUnaryExpression astUnaryExpression){}

    public void visit(ASTPostfixExpression astPostfixExpression){}

    public void visit(ASTBinaryExpression astBinaryExpression){}

    public void visit(ASTIdentifier astIdentifier){}

    public void visit(ASTFunctionCall astFunctionCall){}
```

```
    public void visit(ASTArrayAccess astArrayAccess){}

    public void visit(ASTStatement statement){}

    public void visit(ASTSelectionStatement astSelectionStatement){}

    public void visit(ASTCompoundStatement astCompoundStatement){}

    public void visit(ASTIterationDeclaredStatement
astIterationDeclaredStatement){}

    public void visit(ASTIterationStatement astIterationStatement){}

    public void visit(ASTExpressionStatement astExpressionStatement){}

    public void visit(ASTBreakStatement astBreakStatement){}

    public void visit(ASTReturnStatement astReturnStatement){}

    public void visit(ASTLabeledStatement astLabeledStatement){}

    public String getErrorInfo(){}
}
```

MySemantic.java

```
package bit.minisys.minicc.semantic;

import bit.minisys.minicc.parser.ast.*;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.*;

public class MySemantic implements IMiniCCSemantic{
    public String run(String iFile) throws Exception{
        ObjectMapper mapper = new ObjectMapper();
        ASTCompilationUnit program = mapper.readValue(new File(iFile),
ASTCompilationUnit.class);

        MySemanticAnalyzer semanticAnalyzer = new MySemanticAnalyzer();
        semanticAnalyzer.visit(program);

        System.out.println(semanticAnalyzer.getErrorInfo());

        System.out.println("4. Semantic finished!");

        return iFile;
```

```
    }
}
```

运行截图

测试代码：

```
int func(int t) {
    return u+t;
}

int main() {
    int sum = 0;

    for (int i=0; i<10; i+=2) {
        if (sum < 10) {
            sum += i;
        }
    }

    int sum = push();
    push();
    x = sum;

    break;

    return 0;
}
```

运行截图：

```
Start to compile ...
2. LexAnalyse finished!
3. Parse finished!
ES01: Identifier "u" is undefined.
ES02: Variable "sum" is already defined in this scope.
ES01: Identifier "push" is undefined.
ES01: Identifier "push" is undefined.
ES01: Identifier "x" is undefined.
ES03: a break statement may only be used within a loop or switch.

4. Semantic finished!
```

实验心得

虽然思路非常简单，无非就是遍历AST树并维护两个表和一个标志，但是在实际代码实现过程中，出现了各种各样的逻辑漏洞，再加上代码量也不小，花费的时间远多于预期。