编译课系统程实验报告

实验 3：语义分析

姓名	卢兑珑	院系	计算机学院	学号	L170300901
任课教师		指导教师			
实验地点		实验时间			
实验课表现	出勤、表现得分		实验报告得分		实验总分	
	操作结果得分					

一、需求分析

得分

1. 巩固对语义分析的基本功能和原理的认识。
2. 能够基于语法指导翻译的知识进行语义分析。
3. 掌握类高级语言中基本语句所对应的语义动作。
4. 理解并处理语义分析中的异常和错误。

在语法分析器的基础上设计实现类高级语言的语义分析器，基本功能如下：

（1）能分析以下几类语句，并生成中间代码（三地址指令和四元式形式）：

声明语句（包括变量声明、数组声明、记录声明和过程声明）

表达式及赋值语句（包括数组元素的引用和赋值）

分支语句：if_then_else

循环语句：do_while

过程调用语句

（2）具备语义错误处理能力，包括变量或函数重复声明、变量或函数引用前未声明、运算符和运算分量之间的类型不匹配（如整型变量与数组变量相加减）等错误，能准确给出错误所在位置，并采用可行的错误恢复策略。输出的错误提示信息格式如下：

Error at Line [行号]：[说明文字]

（3）系统的输入形式：要求能够通过文件导入测试用例。测试用例要涵盖第（1）条中列出的各种类型的语句，以及第（2）条中列出的各种类型的错误。

（4）系统的输出分为两部分：一部分是打印输出符号表。另一部分是打印输出三地址指令和四元式序列，格式如下图所示（以输入语句“while a<b do if c<d then x=y+z else x=y-z”为例）：

```
1 : ( j<, a , b , 3 ) if a < b goto 3
2 : ( j , - , - , 11) goto 11
3 : ( j<, c , d , 5 ) if c < d goto 5
4 : ( j , - , - , 8 ) goto 8
5 : ( + , y , z , t1 ) t1 = y + z
6 : ( = , t1 , - , x ) x = t1
7 : ( j , - , - , 1 ) goto 1
8 : ( - , y , z , t2 ) t2 = y - z
9 : ( = , t2 , - , x ) x = t2
10 : ( j , - , - , 1 ) goto 1
```

二、文法设计

得分

要求：给出如下语言成分所对应的语义动作

- 声明语句（包括变量声明、数组声明、记录声明和过程声明）
- 表达式及赋值语句（包括数组元素的引用和赋值）
- 分支语句：if_then_else
- 循环语句：do_while
- 过程调用语句

设计的文法如下：

Program -> Function Program
Program -> $
Function -> Type ID M_E1 (Parameters) Function_Body
Function -> void ID M_E1 (Parameters) Function_Body
Parameters -> $
Parameters -> Type ID M_E8 Parameters'
Parameters' -> $
Parameters' -> , Type ID M_E8 Parameters'
Function_Body -> ;
Function_Body -> Block
Block -> { Define_Sentenses Sentenses }
Define_Sentenses -> Define_Sentense Define_Sentenses
Define_Sentenses -> $
Define_Sentense -> Type ID M_A1 Array M_A2 Indentifiers ; M_E2
Define_Sentense -> struct ID M_A3 { List_Member } ; M_E2
List_Member -> Type ID M_A1 Array ; M_A4 List_Member'
List_Member' -> Type ID M_A1 Array ; M_A5 List_Member'
List_Member' -> M_A6
Array -> M_A7
Array -> [INT] M_A8 Array M_A9_1
Indentifiers -> $
Indentifiers -> , ID M_A10 Array M_A11 Indentifiers
Sentenses -> $
Sentenses -> Sentense Sentenses
Sentense -> Expression ;
Sentense -> ;
Sentense -> return Expression M_A55 ;
Sentense -> continue ;
Sentense -> break ;
Sentense -> if (Expression) M_A12 Block M_A13 Else_Body M_A14
Sentense -> switch (Expression) { Cases_Body }
Sentense -> do Sentense while (Expression) ;
Sentense -> for (Expression ; Expression ; Expression) Sentense
Sentense -> while (M_A15 Expression) M_A12 Block M_A16
Else_Body -> else Block

Else_Body -> $
Cases_Body -> Case_Body Cases_Body
Cases_Body -> $
Case_Body -> case Constant : Sentenses
Case_Body -> default : Sentenses
Expression -> Value M_A17 Expression' M_A18
Expression' -> M_E6
Expression' -> < Value M_E7 M_A19
Expression' -> <= Value M_E7 M_A20
Expression' -> > Value M_E7 M_A21
Expression' -> >= Value M_E7 M_A22
Expression' -> == Value M_E7 M_A23
Expression' -> != Value M_E7 M_A24
Expression' -> = Value M_E7 M_A25
Expression' -> += Value M_E7 M_A26
Expression' -> -= Value M_E7 M_A27
Expression' -> *= Value M_E7 M_A28
Expression' -> /= Value M_E7 M_A29
Expression' -> %= Value M_E7 M_A30
Value -> Add_Item M_A17 Add_Items M_A18
Add_Items -> $
Add_Items -> + Add_Item M_E7 M_A31 Add_Items M_A18
Add_Items -> - Add_Item M_E7 M_A32 Add_Items M_A18
Add_Item -> Factor_Multi M_A17 Factor_Multis M_A18
Factor_Multis -> $
Factor_Multis -> * Factor_Multi M_E7 M_A33 Factor_Multis M_A18
Factor_Multis -> / Factor_Multi M_E7 M_A34 Factor_Multis M_A18
Factor_Multis -> % Factor_Multi M_E7 M_A35 Factor_Multis M_A18
Factor_Multi -> ! Factor_Multi M_A36
Factor_Multi -> ++ Factor_Multi M_A37
Factor_Multi -> -- Factor_Multi M_A38
Factor_Multi -> (Expression) M_A39
Factor_Multi -> ID M_A40 Call M_A9
Factor_Multi -> Constant M_A9
Factor_Multi -> - Factor_Multi M_A41
Call -> M_E3 M_A42 Array M_A9
Call -> M_E4 (Pass_Parameters) M_A43
Call -> . ID M_E5 M_A44
Pass_Parameters -> Expression M_E9 M_A45 Pass_Parameters' M_A46
Pass_Parameters -> M_E10 M_A47
Pass_Parameters' -> , Expression M_E9 M_A45 Pass_Parameters' M_A46
Pass_Parameters' -> M_E10 M_A47
Type -> char M_A48
Type -> int M_A49

Type -> long M_A50
Type -> short M_A51
Type -> float M_A52
Type -> double M_A53
Constant -> INT M_A54
Constant -> FLOAT M_A54
Constant -> DOUBLE M_A54
Constant -> CHAR M_A54
M_A1 -> $
M_A2 -> $
M_A3 -> $
M_A4 -> $
M_A5 -> $
M_A6 -> $
M_A7 -> $
M_A8 -> $
M_A9 -> $
M_A10 -> $
M_A11 -> $
M_A12 -> $
M_A13 -> $
M_A14 -> $
M_A15 -> $
M_A16 -> $
M_A17 -> $
M_A18 -> $
M_A19 -> $
M_A20 -> $
M_A21 -> $
M_A22 -> $
M_A23 -> $
M_A24 -> $
M_A25 -> $
M_A26 -> $
M_A27 -> $
M_A28 -> $
M_A29 -> $
M_A30 -> $
M_A31 -> $
M_A32 -> $
M_A33 -> $
M_A34 -> $
M_A35 -> $
M_A36 -> $

M_A37 -> $
M_A38 -> $
M_A39 -> $
M_A40 -> $
M_A41 -> $
M_A42 -> $
M_A43 -> $
M_A44 -> $
M_A45 -> $
M_A46 -> $
M_A47 -> $
M_A48 -> $
M_A49 -> $
M_A50 -> $
M_A51 -> $
M_A52 -> $
M_A53 -> $
M_A54 -> $
M_A55 -> $
M_E1 -> $
M_E2 -> $
M_E3 -> $
M_E4 -> $
M_E5 -> $
M_E6 -> $
M_E7 -> $
M_E8 -> $
M_E9 -> $
M_E10 -> $
M_A9_1 -> $

三、系统设计	得分	

要求：分为系统概要设计和系统详细设计。

（1）系统概要设计：给出必要的系统宏观层面设计图，如系统框架图、数据流图、功能模块结构图等以及相应的文字说明。

（2）系统详细设计：对如下工作进行展开描述

- ✓ 核心数据结构的设计
- ✓ 主要功能函数说明
- ✓ 程序核心部分的程序流程图

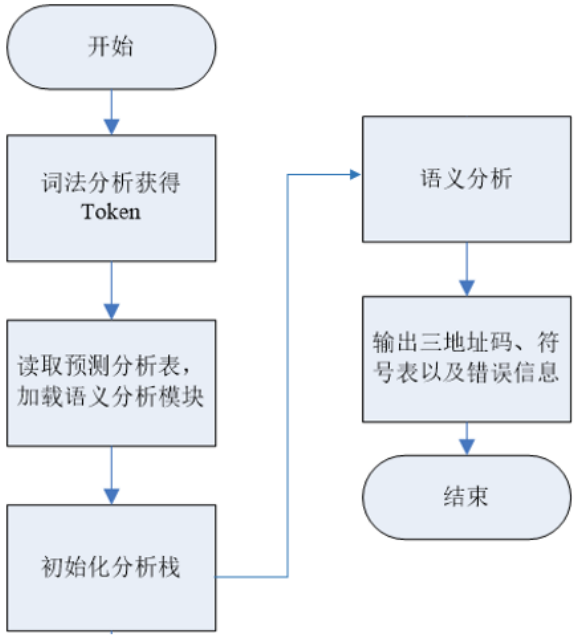
系统概要设计：

Scanning.java 是语义分析器

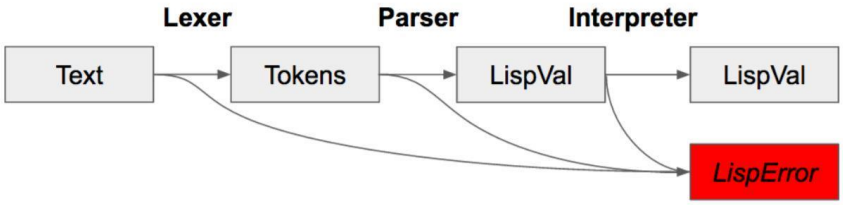
Grammar_handler.java 用于进行语法处理

Parser.java 是语法分析器

Production.java 用于语法分析过程中处理产生式
Complier.java 调用文件读取界面 ReadFileGui，词法分析界面 ScannerGui ，语法分析界面 ParserGui 和语义分析界面 SemanticGui。
SemanticAnalyser.java 是语义分析器
Function.java 用于存储语义分析得到的函数信息
Identifier.java 用于存储语义分析得到的标识符信息
Node.java 用于存储语义分析过程中，语义分析栈中的节点信息。



程序执行的流程图如上图所示



数据流图的设计如上图所示
系统详细设计：
核心数据结构：
文法非终结符含义如下：
Program：整个程序，仅由函数构成
Function：函数，包括函数声明和函数体
Parameters ：函数声明的参数列表，不可空
Parameters’函数声明的参数列表，可空
Function_Body ：函数体，可空
Block ：函数体，不可空
Define_Sentenses ：多条变量声明语句
Define_Sentense：单条变量声明语句

List_Member：结构体的成员变量
Array ： 数组
Indentifiers ： 多条标识符定义
Sentenses ： 多条程序可执行语句
Sentense ： 单条程序可执行语句
Else_Body ： 与 if 匹配的 else 程序结构，可空
Cases_Body ： 与 switch 匹配的 case 程序结构，可空
Expression：表达式，由单个 value 构成 或 由两个 value 形成 value1=value2
Value ： 由运算优先级为加法的项加减构成的值
Add_Items：由运算优先级为乘除的项构成的值，加减法的计算分量
Factor_Multis：乘除法（或同优先级）的计算分量
Call：调用过程产生的特殊值，包括函数返回值，结构体成员变量，数组引用
Pass_Parameters：调用函数的形参列表
Type：类型
Constant：常量
定义了如下语义分析变量：

```
int                             addrOffset;          // 为当前变量进行内存分配的起始地址(即相对于总内存开始位置的偏移量)
int                             tempVariableNum;     // 指令翻译过程中用到的中间变量的个数
int                             boolVariableNum;     // 指令翻译过程中用到的布尔变量的个数
int                             inputCachePointer;   // 当前输入缓冲区的输入指针
Stack<Node>         node_stack;          // 语义分析栈，用于存储节点相应属性，构建相应的注释分析树
List<String[]>      instructions;        // 语义分析的得到的指令序列，格式为<四元式序列，三地址指令>
List<Function>      functions;           // 语义分析得到的函数名列表
List<Identifier>    symbolsTable;        // 语义分析得到的符号表
List<String[]>      errorMessages;       // 语义分析产生的错误信息，格式为<行号，错误信息>
```

定义了如下错误处理变量：

```
boolean                         isExpressionError;  // 用于判断在推导表达式时是否发生错误，在表达式推导完成时恢复 false
boolean                         isFuncError;        // 用于判断函数调用时是否发生错误，在函数调用完成时恢复 false
```

存储节点信息分析树：

```
public  class  Node  {
    private  Node                           father;          //注释分析树中，本节点的父亲节点
    private  String                         name;            //注释分析树中，本节点的名称
    public    List<Node>          sons;            //注释分析树中，本节点的子节点列表（不包含语法动作和终结符，仅包含非终结符）
    public    Map<String,String> attribute;       //注释分析树中，本节点拥有的属性列表
```

标识符信息存储：

```
public  class  Identifier  {
    private  String                 name;               //符号表中标识符名称
    private  String                 type;               //符号表中标识符基本类型
    private  int                      offset;             //符号表中标识符在内存中的起始地址
    private  int                      length;             //符号表中标识符的长度
    public  List<Integer>    arr_list;           //数组标识符的各维下标
```

```
        public  List<Identifier> members;              //结构体标识符的成员变量列表
```

语义分析得到的函数信息存储：

```
public  class  Function  {
        private  String  name;
        private  String  returnType;
        public  List<String> paramTypes;
```

核心函数：

PDA：

执行语法分析的 PDA，在过程中进行语法制导翻译

其中，语法分析栈在分析过程中的可压进所有文法符号，而语义分析栈仅压入非终结符

注释分析树中只包括不是语义动作的非终结符

PDA 执行流程{

建立语法分析栈和语义分析栈，并将初始符号压入到两个栈中

while(语法制导翻译未完成){

若当前语法分析栈顶节点为非终结符，则语义分析栈弹栈，为语义动作做准备

if(当前语法分析节点与输入缓冲区输入内容匹配【说明终结符匹配成功】){

语法分析栈栈顶节点出栈，输入指针向下移动

}

else(当前语法分析节点与输入缓冲区输入内容不匹配【说明是非终结符 或 终结符匹配失败】){

查找预测分析表，选择对应的产生式对当前非终结符进行替换

if(预测分析表查找成功【当前非终结符可被替换】){

若当前非终结符为语义动作，则执行相应的语义动作

语法分析栈的栈顶符号出栈

找到的产生式的右部所有符号逆序进语法分析栈，所有非终结符逆序进语义分析栈

注释分析树中添加相应的父子节点关系，只添加不是语义动作的非终结符

【是语义动作的非终结符 不会被添加进任何一个 node 的 son 列表中，但语义动作的 node 可以具有 father】

}

else(预测分析表查找失败【当前文法符号不可被替换】){

错误处理

}

}

}

}

语义动作：

在执行 semanticAction 函数时：（以产生式 const -> INT M74_2 为例）

node 为相应的语义动作：M74_2

node.getFather()为语义动作 M74_2 对应的产生式左部非终结符 const

函数说明：

gen(s):生成指令 s

newTemp():生成新的中间变量

newBool():生成新的布尔变量

backpatch(x, s):向指令序列中 x 处回填入指令 s

removeSymbol(x):从符号表中自底向上移除 x 个符号

latesetSymbol(): 表示符号表中最后一个符号(即：最新生成的符号)

enter(name,type,offset,length):向符号表中插入具有四个属性的新符号

lookup(X.lexeme)：表示在输入缓冲区中查找词法分析器得到的 Token 符号 X 的属性值

semanticAction 函数对含有不同种类文法的字符串进行分析，以及对有错误的内容进行错误处理：

```
public  void  semanticAction(String  nonTerminal,Node  node){
                // 含有 A1 的文法：
                // Define_Sentense -> Type ID M_A1 Array M_A2 Indentifiers ; M_E2
                // List_Member -> Type ID M_A1 Array ; M_A4 List_Member'
                // List_Member' -> Type ID M_A1 Array ; M_A5 List_Member'
                // A1 内容：
                // Array.name=lookup(ID.lexeme); Array.type=Type.type; Array.length=Type.length; Array.dimension=0;
```

错误处理内容如下：

检测到错误时，就停止生成指令。直到超出了错误的作用域，再继续生成指令。（比如检测到表达式分量出现错误时，直到语法分析分析完整个表达式，再开始产生指令）。对于检测到错误前生成的有影响的指令，进行删除处理。

可以检测下列错误：

函数名重复声明

变量名重复声明

使用了未定义的变量名

使用了未定义的函数名

调用结构体中不存在的成员变量

运算分量的类型不匹配

过程调用时实参与形参类型不匹配

过程调用时实参与形参数目不匹配

数组调用时调用的维度和定义的维度不匹配

函数的返回值类型与表达式要求类型不匹配

四、系统实现及结果分析

得分

要求：对如下内容展开描述。

（1） 系统实现过程中遇到的问题；

（2） 针对一测试程序输出其语义分析结果；

（3） 输出针对此测试程序经过语义分析后的符号表；

（4） 输出针对此测试程序对应的语义错误报告；

（5） 对实验结果进行分析。

注：其中的测试样例需先用已编写的词法分析程序进行处理。

遇到的问题：

1. 增加语法动作非终结符 M

做法：将语法动作 M 看作非终结符，并添加产生式“M->空”

对于语法动作而言，只要把他们看作非终结符，就不会影响 first、follow 和 select 集。在增加产生式“M->空”后，语法动作 M 就不能推导出任何其他的符号。因此不会对

已有的语法分析产生任何影响。

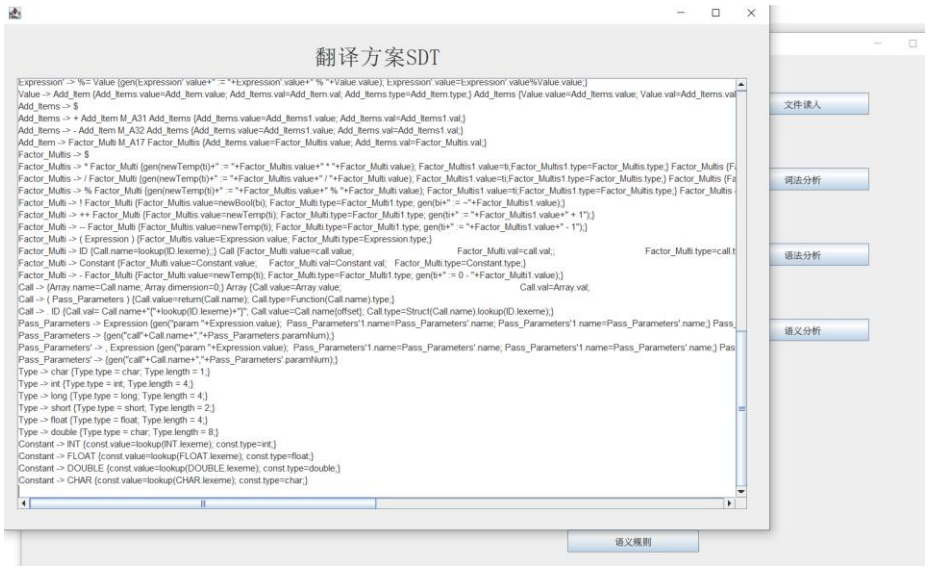
2. 语法动作执行时间

只需要在 PDA 出栈的时候执行相应的程序语句就可以了，按照 PDA 出栈的顺序就是按照程序分析的顺序，这样可以保证语法动作在执行时，需要的继承属性已全部分析完成。

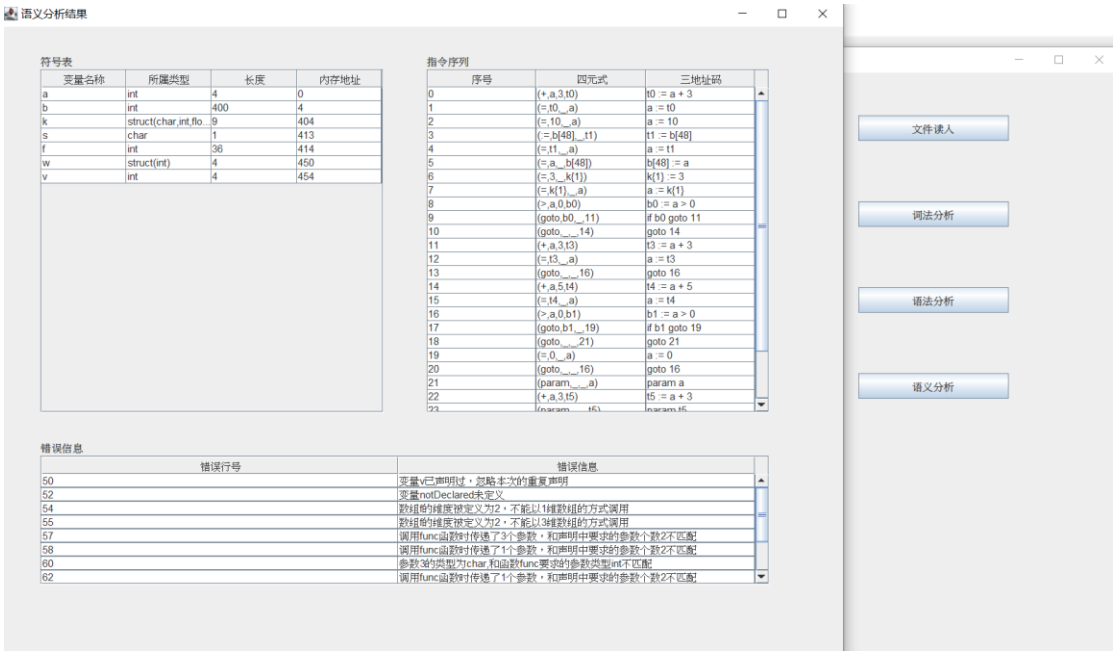
3. 如何存储节点的属性：

建立语义分析栈，进出栈时机与 PDA 栈相同，但语义分析栈中仅保留非终结符（终结符仅有词法分析器提供的综合属性，因此不必保存）。语义分析栈的进出栈元素为 node，在 node 中使用 Map 存储相应的属性名及属性值。并建立 list 来存储语义节点的父子关系，构建虚拟的注释分析树。

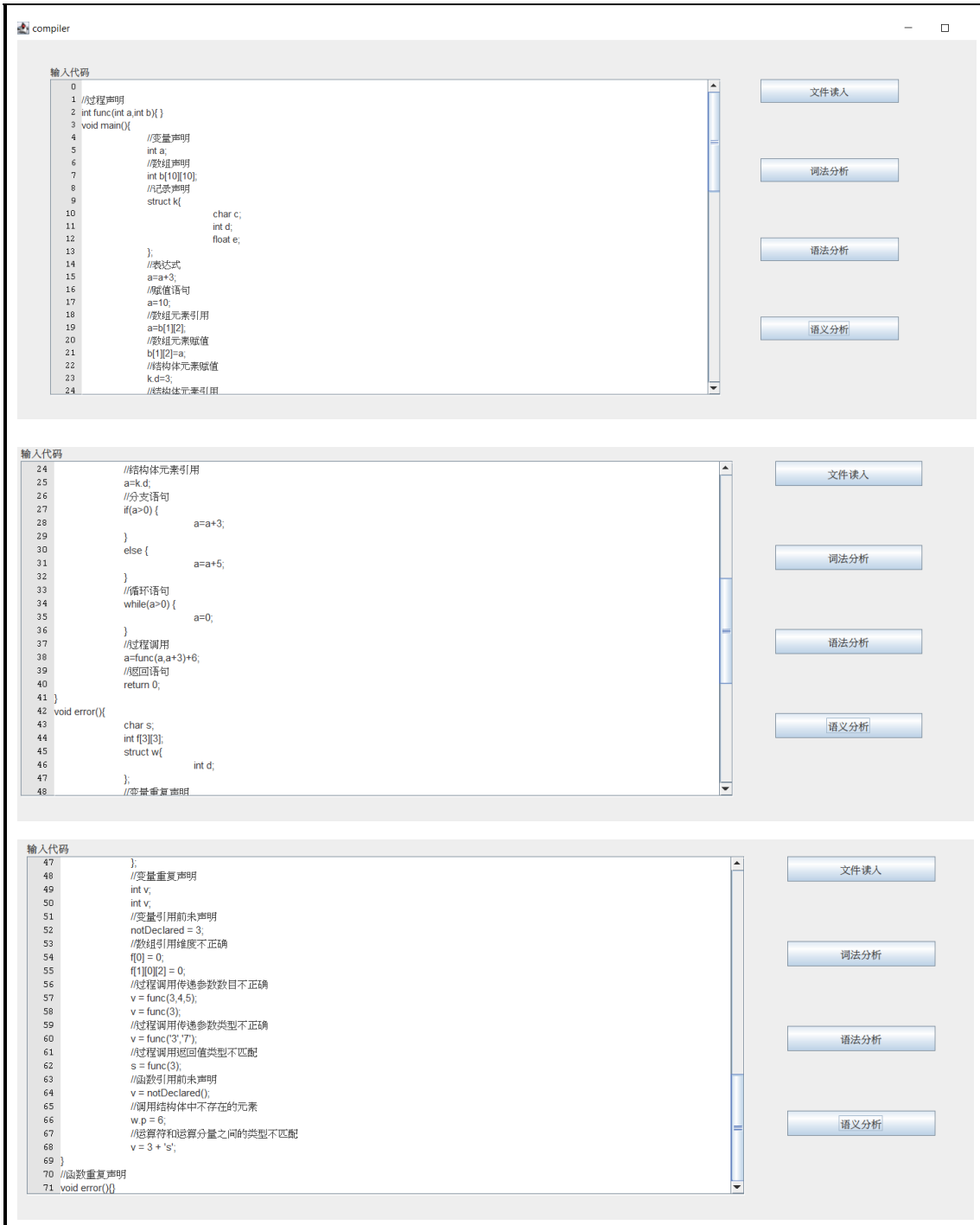
翻译方案如下：



语义分析结果和错误信息如下：



测试使用的代码如下：



实验结果分析：

源代码中包含 while 循环、if-else 分支、if 分支、函数声明以及数组元素的使用。并设 置了若干错误，变量的重复声明、函数的重复声明以及对非数组元素的下标引用等错误，以及在 int 和 float 类型之间的运算。 最终的中间代码中，也实现了相应的代码回填。符号表中实现了对嵌套的符号表的访问和打印。

指导教师评语：   日期：