天津大学

任务二测试用例设计文档

学院 智能与计算学部

专业 计算机科学与技术

成员 胡轶然 许振东 沈昊 李研 胡书豪

学号 3019244355 3019244356
3019234165 3019205150 3019205412

目录

1 任务概述 3
2 CSV 文件的监听器 3
2.1CSV 语法的分支标记版 3
2.2CSV 语法监听器的实现 4
2.3 构建与测试 6
2.3.1 CSV 的第一个测试用例 6
2.3.2 CSV 的第二个测试用例 6
2.3.3 CSV 的第三个测试用例 6
3 将 JSON 文本翻译成 XML 文件 7
3.1 JSON 语法的分支标记版 7
3.2 翻译器的实现 8
3.3 构建与测试 11
3.3.1 JSON 的第一个测试用例 11
3.3.2 JSON 的第二个测试用例 11
3.3.3 JSON 的第三个测试用例 12
4Cymbol 调用图生成器 12
4.1 Cymbol.g4 语法的分支标记版 13
4.2 调用图生成器程序 14
4.3 构建与测试 16
4.3.1Cymbol 的第一个测试用例 16
4.3.2Cymbol 的第二个测试用例 17
5Cymbol 变量与函数验证器 18
5.1 搭建验证器 19
5.2 对给定用例进行测试 21
5.2.1 第一个测试用例 21
5.2.2 第二测试用例 22

1 任务概述

基于任务一中完成的 CSV、JSON 和 Cymbol 语法，由浅入深地构造四个监听器。
任务分工：

胡轶然:任务二会议纪要的整理和书写;学习课本第七章 listener 和 visitor 的知识;完成第八章 Building Some Real Language Applications 中 Validating Program Symbol Usage 的任务，并提供相应的设计文档内容；24%

许振东：任务二设计文档的的整理和书写；学习课本第七章 listener 和 visitor 的知识；完成第八章 Building Some Real Language Applications 中 Loading CSV Data 的任务；19%

沈昊:学习课本第七章 listener 和 visitor 的知识;提供任务二会议纪要部分内容;完成第八章 Building Some Real Language Applications 中 Translating JSON to XML 的任务；并提供相应的设计文档内容；19%

李研：任务二会议纪要第七章基础知识的整理与讲解；提供任务二会议纪要部分内容；19%

胡书豪：提供任务二会议纪要部分内容；完成第八章 Building Some Real Language Applications 中 Generating a Call Graph 的任务；并提供相应的设计文档内容；19%

2 CSV 文件的监听器

2.1CSV 语法的分支标记版

为获取精确的监听器方法，需要对任务一中完成的 CSV 语法的备选分支进行标记
修改后的 CSV.g4文件如下：

```
grammar CSV;

file : hdr row+ ;
hdr  : row ;

row  : field (',' field)* '\r'? '\n' ;
field
    : TEXT       # text
    | STRING     # string
    |            # empty
    ;

TEXT : ~[,\n\r]+;
STRING : '"'('""'|~'"')* '"' ;   // quote-quote is an escaped quote
```

2.2CSV 语法监听器的实现

监听器 LoadCSV.java 文件如下：

```
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.LinkedHashMap;
/**
 * Terence Parr, The Definitive ANTLR4 Reference, Chapter 8
 */
public class LoadCSV {
    public static class Loader extends CSVBaseListener {
        public static final String EMPTY = "";
        /**
         *这个列表中的每个元素是一个代表一行数据的 MAP，该 mao 是从字段名到字段值的映射
         */
        List<Map<String, String>> rows = new ArrayList<Map<String, String>>();
        /**
         * 列名的列表
         */
        List<String> header;
        /**
         * 构造一个存放当前行中所有字段值的列表
         */
        List<String> currentRowFieldValues;
        /**
         * 提取合适的字符串，并将其加入 currentRow-FieldValues
         */
        public void exitString(CSVParser.StringContext ctx) {
            currentRowFieldValues.add(ctx.STRING().getText());
        }
        public void exitText(CSVParser.TextContext ctx) {
            currentRowFieldValues.add(ctx.TEXT().getText());
        }
        public void exitEmpty(CSVParser.EmptyContext ctx) {
            currentRowFieldValues.add(EMPTY);
        }
        /**
         * 在第一行 exitRow（）方法执行结束后，currentRowFieldValues 就包含了全部的列名
         */
        public void exitHdr(CSVParser.HdrContext ctx) {
            header = new ArrayList<String>();
            header.addAll(currentRowFieldValues);
        }
```

```
        /**
         * 处理行数据
         * 这个过程需要两个操作：开始和结束对一行的操作
         * 当开始对一行的处理时，我们需要创建（清楚）currentRowFieldValues，以备接受后续数据
         */
        public void enterRow(CSVParser.RowContext ctx) {
            currentRowFieldValues = new ArrayList<String>();
        }
        /**
         * 在完成对一行的处理时我们需要考虑上下文
         * 如果当前是一个数据行就创建 map 同步遍历 header 和 currentRow-FieldValues，将映射关系放入该 map 中
         */
        public void exitRow(CSVParser.RowContext ctx) {
            // 标题行什么都不做
            //if (ctx.parent instanceof CSVParser.HdrContext) return; OR:
            if (ctx.getParent().getRuleIndex() == CSVParser.RULE_hdr) return;
            // 以下为数据行
            Map<String, String> m = new LinkedHashMap<String, String>();
            int i = 0;
            for (String v : currentRowFieldValues) {
                m.put(header.get(i), v);
                i++;
            }
            rows.add(m);
        }
    }
    public static void main(String[] args) throws Exception {
        String inputFile = null;
        if (args.length > 0) inputFile = args[0];
        InputStream is = System.in;
        if (inputFile != null) is = new FileInputStream(inputFile);
        CharStream input = CharStreams.fromStream(is, StandardCharsets.UTF_8);
        CSVLexer lexer = new CSVLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        CSVParser parser = new CSVParser(tokens);
        ParseTree tree = parser.file();
        //System.out.println(tree.toStringTree(parser));
        //进行打印数据行
        ParseTreeWalker walker = new ParseTreeWalker();
        Loader loader = new Loader();
        walker.walk(loader, tree);
        System.out.println(loader.rows);
    }
}
```

2.3 构建与测试

2.3.1 CSV 的第一个测试用例

1、对于已给出的 t.csv 进行测试，t.csv 如下

```
Details,Month,Amount
Mid Bonus,June,"$2,000"
,January,"""zippo"""
Total Bonuses,"","$5,000"
```

2、在终端输入如下命令：

```
antlr4 CSV.g4
javac CSV*.java LoadCSV.java
java LoadCSV t.csv
```

3、监听显示结果如下：

```
[{Details=Mid Bonus, Month=June, Amount="$2,000"}, {Details=, Month=January, Amo
unt="""zippo"""}, {Details=Total Bonuses, Month="", Amount="$5,000"}]
```

2.3.2 CSV 的第二个测试用例

1、对任务一中自主设计的两个用例第一个进行测试

2、test1.csv 如下：

```
stuid,stuname,score
1,hyr,100
2,xzd,99
3,ly,99
4,sh,99
```

3、操作步骤与结果如下：

```
antlr4 CSV.g4
javac CSV*.java LoadCSV.java
java LoadCSV test1.csv
```

```
[{stuid=1, stuname=hyr, score=100}, {stuid=2, stuname=xzd, score=99}, {stuid=3,
stuname=ly, score=99}, {stuid=4, stuname=sh, score=99}, {stuid=}, {stuid=}]
```

2.3.3 CSV 的第三个测试用例

1、对任务一中自主设计的两个用例第二个进行测试

2、test2.csv 如下：

```
name,address,goods
hyr,"tju",soap
xzd,"tianjin""wuhu""",chiken
liyan,"daqing",coke
```

3、操作步骤与结果如下：

```
antlr4 CSV.g4
javac CSV*.java LoadCSV.java
java LoadCSV test2.csv
```

```
[{name=hyr, address="tju", goods=soap}, {name=xzd, address="tianjin""wuhu""", go
ods=chiken}, {name=liyan, address="daqing", goods=coke}, {name=}]
```

3 将 JSON 文本翻译成 XML 文件

读取 JSON 文本输入，并给出等价的 XML 输出。

3.1 JSON 语法的分支标记版

同样对 JSON 语法中的备选分支做一定的标记，修改后的 JSON.g4文件如下：

```
grammar JSON;

json
    : object
    | array
    ;

object
    : '{' pair (',' pair)* '}'    # AnObject
    | '{' '}'                     # EmptyObject
    ;

pair
    : STRING ':' value
    ;

array
    : '[' value (',' value)* ']'  # ArrayOfValues
    | '[' ']'                     # EmptyArray
    ;

value
    : STRING                      # String
    | NUMBER                      # Atom
    | object                      # ObjectValue
    | array                       # ArrayValue
    | 'true'                      # Atom
    | 'false'                     # Atom
    | 'null'                      # Atom
    ;


LCURLY : '{' ;
LBRACK : '[' ;
```

```
STRING
    : '"' (ESC | ~["\\])* '"' ;

fragment ESC : '\\' (["\\/bfnrt] | UNICODE) ;
fragment UNICODE : 'u' HEX HEX HEX HEX;
fragment HEX : [0-9a-fA-F] ;

NUMBER
    : '-'? INT '.' [0-9]+ EXP?    // 1.35, 1.35E-9, 0.3, -4.5
    | '-'? INT EXP                // 1e10 -2e4
    | '-'? INT                    // -3, 45
    ;

fragment INT : '0' | [1-9] [0-9]* ;   // no leading zeros
fragment EXP :  [Ee] [+\-]? INT ;

WS : [ \t\n\r]+ -> skip;
```

3.2 翻译器的实现

翻译器实现如下： JSON2XML.java

```
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
/*
{
    "description" : "An imaginary server config file",
    "logs" : {"level":"verbose", "dir":"/var/log"},
    "host" : "antlr.org",
    "admin": ["parrt", "tombu"]
    "aliases": []
}
to
<description>An imaginary server config file</description>
<logs>
    <level>verbose</level>
    <dir>/var/log</dir>
</logs>
<host>antlr.org</host>
<admin>
    <element>parrt</element> <!-- inexact -->
    <element>tombu</element>
</admin>
<aliases></aliases>
 */
public class JSON2XML {
    //标注语法分析树。将每棵树翻译完的字符串存储在子树的根节点中。
```

```
public static class XMLEmitter extends JSONBaseListener {
    ParseTreeProperty<String> xml = new ParseTreeProperty<String>();
    String getXML(ParseTree ctx) {
        return xml.get(ctx);
    }
    void setXML(ParseTree ctx, String s) {
        xml.put(ctx, s);
    }
    //使用 set 将根元素 object 或 array 生成的结果标注语法分析树的根节点。
    public void exitJson(JSONParser.JsonContext ctx) {
        setXML(ctx, getXML(ctx.getChild(0)));
    }
/*
json 的对象由键值对组成。因此对于每个 object 规则在 anobject 备选分支中的键值对，
我们要将对应的 XML 追加到语法分析树中存储的结果之后。
*/
    public void exitAnObject(JSONParser.AnObjectContext ctx) {
        StringBuilder buf = new StringBuilder();
        buf.append("\n");
        for (JSONParser.PairContext pctx : ctx.pair()) {
            buf.append(getXML(pctx));
        }
        setXML(ctx, buf.toString());
    }
    public void exitEmptyObject(JSONParser.EmptyObjectContext ctx) {
        setXML(ctx, "");
    }
//处理数组的方式与上相似，从各子节点中获取 XML 结果之后，放入<element>标签进行连接。
    public void exitArrayOfValues(JSONParser.ArrayOfValuesContext ctx) {
        StringBuilder buf = new StringBuilder();
        buf.append("\n");
        for (JSONParser.ValueContext vctx : ctx.value()) {
            buf.append("<element>");
            buf.append(getXML(vctx));
            buf.append("</element>");
            buf.append("\n");
        }
        setXML(ctx, buf.toString());
    }
    public void exitEmptyArray(JSONParser.EmptyArrayContext ctx) {
        setXML(ctx, "");
    }
/*
翻译完成 value 规则对应的所有元素后，需要处理键值对转换为标签和文本。
开始和结束标签之间的文本来源于 value 子节点。
*/
    public void exitPair(JSONParser.PairContext ctx) {
        String tag = stripQuotes(ctx.STRING().getText());
        JSONParser.ValueContext vctx = ctx.value();
        String x = String.format("<%s>%s</%s>\n", tag, getXML(vctx), tag);
        setXML(ctx, x);
```

```
        }
    /*除了 rule()方法匹配到一个对象或者数组，其可以将这些复合元素的翻译结果拷贝到
    自身的语法分析树的的节点中*/
        public void exitObjectValue(JSONParser.ObjectValueContext ctx) {
            // analogous to String value() { return object(); }
            setXML(ctx,getXML(ctx.object()));
        }
        public void exitArrayValue(JSONParser.ArrayValueContext ctx) {
            setXML(ctx, getXML(ctx.array()));  // String value() { return array(); }
        }
    //Atom 节点对应的标注值要与词法符号的文本内容相匹配。
        public void exitAtom(JSONParser.AtomContext ctx) {
            setXML(ctx, ctx.getText());
        }
    //字符串的处理基本和上述相同，除了需要额外剥离双引号
        public void exitString(JSONParser.StringContext ctx) {
            setXML(ctx, stripQuotes(ctx.getText()));
        }
        public static String stripQuotes(String s) {
            if ( s == null ||s.charAt(0) != '"') return s;
            return s.substring(1, s.length() - 1);
        }
    }
    public static void main(String[] args) throws Exception {
        String inputFile = null;
        if (args.length > 0)
            inputFile = args[0];
        InputStream is = System.in;
        if (inputFile != null) {
            is = new FileInputStream(inputFile);
        }
        CharStream input = CharStreams.fromStream(is, StandardCharsets.UTF_8);
        JSONLexer lexer = new JSONLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        JSONParser parser = new JSONParser(tokens);
        parser.setBuildParseTree(true);
        ParseTree tree = parser.json();
        // show tree in text form
        System.out.println(tree.toStringTree(parser));
        ParseTreeWalker walker = new ParseTreeWalker();
        XMLEmitter xmlEmitter = new XMLEmitter();
        walker.walk(xmlEmitter, tree);
        System.out.println(xmlEmitter.getXML(tree));
    }
}
```

3.3 构建与测试

3.3.1 JSON 的第一个测试用例

1、对已给出的 t.json 进行测试，t.json 如下：

```
{
    "description" : "An imaginary server confing file",
    "logs" : {"level": "verbose", "dir": "/var/log"},
    "host" : "antlr.org",
    "admin" : ["parrt", "tombu"],
    "aliases" : []
}
```

2、在终端输入如下命令：

```
antlr4 JSON.g4
javac JSON*.java
java JSON2XML t.json
```

3、结果如下：

```
(json (object { (pair "description" : (value "An imaginary server confing file")
) , (pair "logs" : (value (object { (pair "level" : (value "verbose")) , (pair "
dir" : (value "/var/log")) }))) , (pair "host" : (value "antlr.org")) , (pair "a
dmin" : (value (array [ (value "parrt") , (value "tombu") ]))) , (pair "aliases"
 : (value (array [ ]))) }))

<description>An imaginary server confing file</description>
<logs>
<level>verbose</level>
<dir>/var/log</dir>
</logs>
<host>antlr.org</host>
<admin>
<element>parrt</element>
<element>tombu</element>
</admin>
<aliases></aliases>
```

3.3.2 JSON 的第二个测试用例

1、对任务一中自主设计的两个用例的第一个进行测试

2、test0.json 如下：

```
[1,2,3]
```

3、操作步骤与结果如下：

```
antlr4 JSON.g4
javac JSON*.java
java JSON2XML test0.json
```

```
(json (array [ (value 1) , (value 2) , (value 3) ]))

<element>1</element>
<element>2</element>
<element>3</element>
```

3.3.3 JSON 的第三个测试用例

1、对任务一中自主设计的两个用例的第二个进行测试
2、test1.json 如下：

```
{
      "x1":  {
          "x2" : {
             "x3" : ["\u0001", "\u0002", "\u0003"],
             "x4" : 1.4e8,
             "x5" : -10
            },
          "x5" : false
      }
}
```

3、操作步骤与结果如下：

```
antlr4 JSON.g4
javac JSON*.java
java JSON2XML test1.json
```

```
(json (object { (pair "x1" : (value (object { (pair "x2" : (value (object { (pai
r "x3" : (value (array [ (value "\u0001") , (value "\u0002") , (value "\u0003")
])))) , (pair "x4" : (value 1.4e8)) , (pair "x5" : (value -10)) })))) , (pair "x5"
 : (value false)) })))) }))

<x1>
<x2>
<x3>
<element>\u0001</element>
<element>\u0002</element>
<element>\u0003</element>
</x3>
<x4>1.4e8</x4>
<x5>-10</x5>
</x2>
<x5>false</x5>
</x1>
```

4Cymbol 调用图生成器

第三部分程序：读取 Cymbol 程序，使用 DOT/graphviz 将其函数调用依赖图可视化

用 Cymbol 语法编写一个调用图生成器。

4.1 Cymbol.g4 语法的分支标记版

对任务一中 Cymbol.g4中的一些备选分支进行标记，修改后如下：

```
grammar Cymbol ;

file    : (functionDecl | varDecl)+ ;

varDecl
    : type ID ('=' expr)? ';'
    ;

type    : 'float' | 'int' | 'void' ; // user-defined type

functionDecl
    : type ID '(' formalParameters? ')' block    // "void f(int x) {...}"
    ;

formalParameters
    : formalParameter (',' formalParameter)*
    ;

formalParameter
    : type ID
    ;

block  : '{' stat* '}' ;    // possibly empty statement block

stat
    : block
    | varDecl
    | 'if' expr 'then' stat ('else' stat)?
    | 'return' expr? ';'
    | expr '=' expr ';'     // assignment
    | expr ';'              // func call
    ;

expr
    : ID '(' exprList? ')'  # Call     // func call like f(), f(x), f(1,2)
    | ID '[' expr ']'       # Index    // array index like a[i], a[i][j]
    | '-' expr              # Negate   // unary minus
    | '!' expr              # Not      // boolean not
    | expr '*' expr         # Mult     // multiply
    | expr ('+'|'-') expr   # AddSub   //
    | expr '==' expr        # Equal    // equality comparison
    | ID                    # Var
    | INT                   # Int
    | '(' expr ')'          # Parens
```

```
    ;

exprList
    : expr (',' expr)* ;    // arg list

K_FLOAT : 'float';
K_INT   : 'int';
K_VOID  : 'void';

ID              : LETTER (LETTER|DIGIT)* ;
fragment LETTER : [a-zA-Z\u0080-\u00FF] ;
INT             : '-'? DIGIT+ ;
fragment DIGIT  : [0-9] ;

WS              : [ \t\n\r]+ -> skip;

LINE_COMMENT    : '//' .*? '\r'? '\n' -> skip; // Match "//" stuff '\n'
COMMENT         : '/*' .*? '*/'       -> skip; // Match "/*" stuff "*/"
```

4.2 调用图生成器程序

调用图生成器程序如下：

```
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.misc.*;
import org.antlr.v4.runtime.tree.*;

import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.util.Set;

public class CallGraph {

    public static class Graph {
        //这里使用的是 org.antlr.v4.runtime.misc: OrderedHashSet, MultiMap
        Set<String>nodes = new OrderedHashSet<String>();    //函数的列表
        MultiMap<String, String> edges =                    //调用者 -> 被调用者
            new MultiMap<String, String>();

        public void edge(String source, String target) {
            edges.map(source, target);
        } //边

        //toDOT() 完整地获取对应的 DOT 代码
        public String toDOT() {
            StringBuffer buf = new StringBuffer();
            buf.append("digraph G {\n");
            buf.append("    ranksep=.25\n");
```

```
            buf.append("    edge [arrowsize=.5]\n");
            buf.append("    node [shape=circle, fontname=\"ArialNarrow\",\n");
            buf.append("          fontsize=12, fixedsize=true, height=.45];\n");
            buf.append("    ");
            for (String node : nodes) { // 首先打印所有节点
                buf.append(node);
                buf.append("; ");
            }
            buf.append("\n");
            for (String src : edges.keySet()) {
                for (String trg : edges.get(src)) {
                    buf.append("    ");
                    buf.append(src);
                    buf.append(" -> ");
                    buf.append(trg);
                    buf.append(";\n");
                }
            }
            buf.append("}\n");
            return buf.toString();
        }
    }

    //使用监听器填充这些数据结构
    static class FunctionListener extends CymbolBaseListener {
        Graph graph = new Graph();
        String currentFunctionName = null;
        //监听器需要两个用于记录的字段

        public void enterFunctionDecl(CymbolParser.FunctionDeclContext ctx) {
            currentFunctionName = ctx.ID().getText();
            graph.nodes.add(currentFunctionName);
        }//当语法分析器遇到函数定义时的方法，令其记录当前函数名

        public void exitCall(CymbolParser.CallContext ctx) {
            String funcName = ctx.ID().getText();
            // 将当前函数映射到被调用函数上
            graph.edge(currentFunctionName, funcName);
        }//当语法分析器发现函数调用时，程序就会记录一条从当前函数到被调用的函数的边
    }


    public static void main(String[] args) throws Exception {
        String inputFile = null;
        if (args.length > 0) inputFile = args[0];
        InputStream is = System.in;
        if (inputFile != null) is = new FileInputStream(inputFile);
        CharStream input = CharStreams.fromStream(is, StandardCharsets.UTF_8);
        CymbolLexer lexer = new CymbolLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        CymbolParser parser = new CymbolParser(tokens);
```

```
        ParseTree tree = parser.file();
        //System.out.println(tree.toStringTree(parser));


        //在遍历中使用自定义的监听器，并产生期望的输出
        ParseTreeWalker walker = new ParseTreeWalker();
        FunctionListener collector = new FunctionListener();
        walker.walk(collector, tree);
        System.out.println(collector.graph.toString());
        System.out.println(collector.graph.toDOT());
    }
}
```

4.3 构建与测试

4.3.1Cymbol 的第一个测试用例

1、对已给出的 t.cymbol 进行测试
2、测试用例如下：

```
int main() {
    fact();
    a();
}
float fact(int n) {
      print(n);
      if (n == 0) then
         return 1;
      return n * fact(n - 1);
}
void a() {
     int x = b();
     if false then {
        c();
        d();
     }
}
void b() { c(); }
void c() { b(); }
void d() {}
void e() {}
```

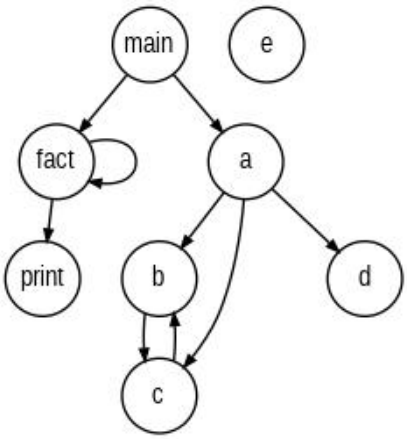
3、操作步骤以及结果如下：

```
antlr4 Cymbol.g4
javac Cymbol*.java CallGraph.java
java CallGraph t.cymbol
```

```
CallGraph$Graph@e2144e4
digraph G {
    ranksep=.25
    edge [arrowsize=.5]
    node [shape=circle, fontname="ArialNarrow",
          fontsize=12, fixedsize=true, height=.45];
    main; fact; a; b; c; d; e;
    main -> fact;
    main -> a;
    fact -> print;
    fact -> fact;
    a -> b;
    a -> c;
    a -> d;
    b -> c;
    c -> b;
```

4、在 graphviz 中预览函数调用图：

```
gedit 1.dot
dot -Tpng -o 1.png 1.dot
xdg-open 1.png
```



4.3.2Cymbol 的第二个测试用例

1、对自主设计的一个用例进行测试

2、test1.cymbol 的测试用例如下：

```
// Cymbol test1 by hyr
int gv = 6; // 全局变量声明
int a;
int plus(int a){
    return x+1; //加法
}
int fact(int x, int y) { // factorial function
if x==1 then return -x; //一元否定
if x==2 then return !x; //布尔非
if x==3 then return x*y; //乘法
if x==y then return (x+y)*2; //括号表达式
return x * plus(y);
}
```

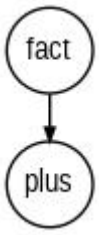
3、操作步骤以及结果如下：

```
antlr4 Cymbol.g4
javac Cymbol*.java CallGraph.java
java CallGraph test1.cymbol
```

```
CallGraph$Graph@497470ed
digraph G {
    ranksep=.25
    edge [arrowsize=.5]
    node [shape=circle, fontname="ArialNarrow",
          fontsize=12, fixedsize=true, height=.45];
    plus; fact;
    fact -> plus;
}
```

4、在 graphviz 中预览函数调用图：

```
gedit 2.dot
dot -Tpng -o 2.png 2.dot
xdg-open 2.png
```



5Cymbol 变量与函数验证器

1、为 Cymbol 构造符号表，用于检测未定义的变量和函数，确保变量和函数被正确运用。
编写的 Cymbol 验证器需要作出以下校验：
引用的变量必须有可以见的（在定义域中）定义；
引用的函数必须有定义（函数必须以任何顺序出现）；
变量不可用作函数
函数不可用作变量
2、示例中的 vars.cymbol 与 vars2.cymbol 中部分标识符无效或存在冲突

```
vars.cymobl

int f(int x, float y) {
  g();           // forward reference is ok
  i = 3;         // no declaration for i (error)
  g = 4;         // g is not variable (error)
  return x + y;  // x, y are defined, so no problem
}

void g() {
  int x = 0;
  float y;
  y = 9;         // y is defined
  f();           // backward reference is ok
  z();           // no such definition (error)
  y();           // y is not function (error)
  x = f;         // f is not a variable (error)
}
```

```
vars2.cymbol

int x;
int y;
void a()
{
  int x;
  x = 1;    // x resolves to current scope, not x in globalscope
  y = 2;    // y is not found in current scope, but resolves in global
  { int y = x; }
}
void b(int z)
{}
```

5.1 搭建验证器

1、首先定义验证器的基本结构 DefPhase.java。文件的实现如下。

```
import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.tree.ParseTreeProperty;

public class DefPhase extends CymbolBaseListener {

    ParseTreeProperty<Scope> scopes = new ParseTreeProperty<Scope>();
    GlobalScope globals;
    Scope currentScope;    // 当前符号的作用域

    public void enterFile(CymbolParser.FileContext ctx) {
        globals = new GlobalScope(null);
        currentScope = globals;
    }

    public void exitFile(CymbolParser.FileContext ctx) {
        System.out.println(globals);
    }

    public void enterFunctionDecl(CymbolParser.FunctionDeclContext ctx) {
        String name = ctx.ID().getText();
        int typeTokenType = ctx.type().start.getType();
        Symbol.Type type = CheckSymbols.getType(typeTokenType);
        // 新建一个指向外围作用域的作用域，这样就完成了入栈操作
        FunctionSymbol function = new FunctionSymbol(name, type, currentScope);
        currentScope.define(function);  // 在当前作用域中定义函数
        saveScope(ctx, function);       // 入栈：设置函数作用域的父作用域为当前作用域
        currentScope = function;        // 现在当前作用域就是函数作用域了
    }

    void saveScope(ParserRuleContext ctx, Scope s) {
        scopes.put(ctx, s);
```

```
    }

    public void exitFunctionDecl(CymbolParser.FunctionDeclContext ctx) {
        System.out.println(currentScope);
        currentScope = currentScope.getEnclosingScope();  // 作用域出栈
    }

    public void enterBlock(CymbolParser.BlockContext ctx) {
        // push new local scope
        currentScope = new LocalScope(currentScope);
        saveScope(ctx, currentScope);
    }

    public void exitBlock(CymbolParser.BlockContext ctx) {
        System.out.println(currentScope);
        currentScope = currentScope.getEnclosingScope(); // pop scope
    }

    public void exitFormalParameter(CymbolParser.FormalParameterContext ctx) {
        defineVar(ctx.type(), ctx.ID().getSymbol());
    }

    public void exitVarDecl(CymbolParser.VarDeclContext ctx) {
        defineVar(ctx.type(), ctx.ID().getSymbol());
    }

    void defineVar(CymbolParser.TypeContext typeCtx, Token nameToken) {
        int typeTokenType = typeCtx.start.getType();
        Symbol.Type type = CheckSymbols.getType(typeTokenType);
        VariableSymbol var = new VariableSymbol(nameToken.getText(), type);
        currentScope.define(var);   // 在当前作用域中定义符号
    }
}
```

2、总验证器 CheckSymbol.java

```
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.ParserRuleContext;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.tree.*;

import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;

public class CheckSymbols {

    public static Symbol.Type getType(int tokenType) {
        switch (tokenType) {
        case CymbolParser.K_VOID :  return Symbol.Type.tVOID;
```

```
        case CymbolParser.K_INT :   return Symbol.Type.tINT;
        case CymbolParser.K_FLOAT : return Symbol.Type.tFLOAT;
        }//数据类型

        return Symbol.Type.tINVALID;
    }

    public static void error(Token t, String msg) {
        System.err.printf("line %d:%d %s\n", t.getLine(), t.getCharPositionInLine(),
msg);
    }

    public static void main(String[] args) throws Exception {
        String inputFile = null;
        if (args.length > 0)
            inputFile = args[0];
        InputStream is = System.in;
        if (inputFile != null) {
            is = new FileInputStream(inputFile);
        }
        CharStream input = CharStreams.fromStream(is, StandardCharsets.UTF_8);
        CymbolLexer lexer = new CymbolLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        CymbolParser parser = new CymbolParser(tokens);
        parser.setBuildParseTree(true);
        ParseTree tree = parser.file();
        // 以文本形式显示树
        System.out.println(tree.toStringTree(parser));

        ParseTreeWalker walker = new ParseTreeWalker();
        DefPhase def = new DefPhase();
        walker.walk(def, tree);
        // 新建一个阶段，将 def 中的符号表信息传递给该阶段
        RefPhase ref = new RefPhase(def.globals, def.scopes);
        walker.walk(ref, tree);
    }
}
```

5.2 对给定用例进行测试

5.2.1 第一个测试用例

操作步骤以及结果如下：

```
antlr4 Cymbol.g4
javac Cymbol*.java CheckSymbols.java *Phase.java *Scope.java *Symbol.java
java CheckSymbols vars.cymbol
```

```
(file (functionDecl (type int) f ( (formalParameters (formalParameter (type int)
 x) , (formalParameter (type float) y)) ) (block { (stat (expr g ( )) ;) (stat (
expr i) = (expr 3) ;) (stat (expr g) = (expr 4) ;) (stat return (expr (expr x) +
 (expr y)) ;) })) (functionDecl (type void) g ( ) (block { (stat (varDecl (type
int) x = (expr 0) ;)) (stat (varDecl (type float) y ;)) (stat (expr y) = (expr 9
) ;) (stat (expr f ( )) ;) (stat (expr z ( )) ;) (stat (expr y ( )) ;) (stat (ex
pr x) = (expr f) ;) }))))
locals:[]
function<f:tINT>:[<x:tINT>, <y:tFLOAT>]
locals:[x, y]
function<g:tVOID>:[]
globals:[f, g]
line 3:2 no such variable: i
line 4:2 g is not a variable
line 13:2 no such function: z
line 14:2 y is not a function
line 15:6 f is not a variable
```

5.2.2 第二测试用例

操作步骤以及结果如下：

```
antlr4 Cymbol.g4
javac Cymbol*.java CheckSymbols.java *Phase.java *Scope.java *Symbol.java
java CheckSymbols vars2.cymbol
```

```
(file (varDecl (type int) x ;) (varDecl (type int) y ;) (functionDecl (type void
) a ( ) (block { (stat (varDecl (type int) x ;)) (stat (expr x) = (expr 1) ;) (s
tat (expr y) = (expr 2) ;) (stat (block { (stat (varDecl (type int) y = (expr x)
 ;)) })) })) (functionDecl (type void) b ( (formalParameters (formalParameter (t
ype int) z)) ) (block { })))
locals:[y]
locals:[x]
function<a:tVOID>:[]
locals:[]
function<b:tVOID>:[<z:tINT>]
globals:[x, y, a, b]
```