编译原理Tiger编译器前端报告

陈炜栋 11300240057
董家明 11300240025

目录

一 项目要求及完成情况

1. 正确的词法/语法分析，给出对应的文法文件2分 √ (`Tiger.g4`)
2. 输出正确的抽象语法树 5分 √ (`testcases中50个测试文件,testcase-result中对应的抽象语法树`)
3. 错误处理功能
    1. 提示错误类型（词法错误、语法错误、语义错误等）、出错位置等 5分 √ （`词法错误，语法错误ANTLR默认行为,部分语义错误：变量作用域及基本类型检查`）
    2. 错误修复 3分 √ (`ANTLR默认行为 single-token insertion and single-token deletion`)
4. 发挥想象力，使编译器尽善尽美 5分 √ (`实现了变量作用域检查和基本类型检查`)
5. Project文档
    1. 使用语法（文法类型，有无改动语法、如何改动等），错误处理说明等 5分 √ （`对实现的变量作用变量作用域检查和基本类型检查的错误处理代码进行了介绍`）
    2. 对本项目语法的使用工具的体会 5分 √ (`ANTLR简述部分`)

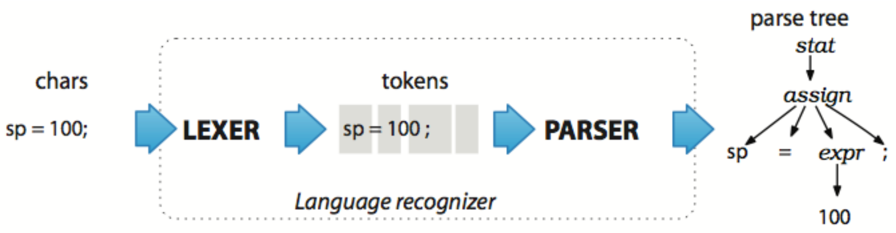
二 项目总览

使用ANTLR工具,为Tiger构造一个编译器前端，将输入的Tigger语言转化为抽象语法树，实现了变量作用域检查和简单的Type Checking。

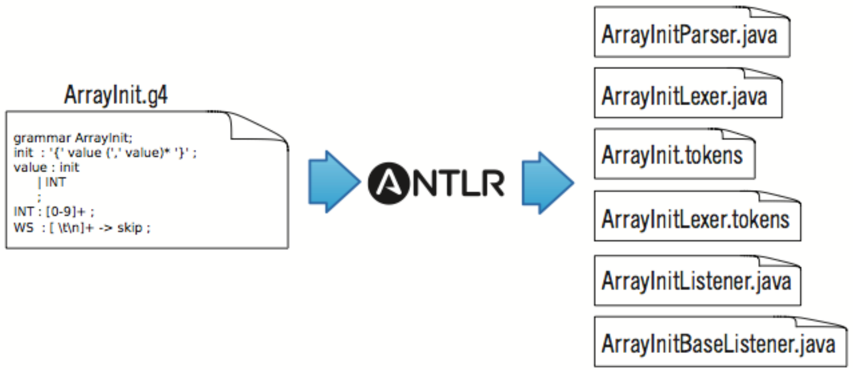
三 ANTLR简述

ANTLR(ANother Tool for language Recognition).作者Terence Parr(University of San Francisco). ANTLR是一个可以接受含有语法描述的语言描述符并且生成程序能够识别这些语言所产生的句子。作为一个翻译程序的 一部分，你可以给你的语法附上简单的操作符和行为并且告诉ANTLR如何构造AST并且如何输出它们。ANTLR知道如何使用Java，C++，C#或者Python来生成它们。 注意到 `词法错误和语法错误` ANTLR有默认的处理行为

```
Here is how ANTLR uses those ideas together in a nutshell: parsers perform single-token insertion and single-token
deletion upon mismatched token errors if possible. If not, parsers gobble up tokens until they find a token that could
reasonably follow the current rule and then return, continuing as if nothing had happened.
```



如上图所示，把源文件转化成AST，需要一个Lexer和Parser。Lexer把源文件读入，分成一个个token。然后Parser读入Lexer产生的token生成AST。在ANTLR提供了Lisenter和Visitor两种方式来遍历抽象语法树。 `本项目利用这些接口来实现变量声明检查。和基本的类型检查` 。



```
grammar ArrayInit;
init  : '{' value (',' value)* '}' ;
value : init
      | INT
      ;
INT : [0-9]+ ;
WS  : [ \t\n]+ -> skip ;
```

四 AST的生成

ANTLR是一个比较成熟的工具，The Definitive ANTLR 4 Reference 详细介绍了工具的使用说明。为了生成AST，需要完成grammer定义文件Tiger.g4 下面定义的语法，基本与官方语法文档指定的语法相同。 为了简化之后的Type Checking，把Ojbect相关的new class等语法去掉了。

```
java  org.antlr.v4.Tool Tiger.g4
```

然后会产生`TigerLexer.java`, `TigerParser.java`, Tiger.tokens , TiegerLexder.tokens, `TigerListener.java` `TigerBaseLisetener.java`

其中TigerLexer.java是ANTLR根据g4为我生成的Lexer，TigerParser.java是对应的Parser，TigerListener.java是前面提到的Listener方式的抽象语法树的接口，TigerBaseListener.java是这个接口的一个基本实现。我们的实现通过继承TigerBaseListener来重载中间的放来实现期望的行为。

```
java org.antlr.v4.runtime.misc.TestRig Tiger program -tree test.in  # 输出文本信息
java org.antlr.v4.runtime.misc.TestRig Tiger program -gui test.in   # 生成图形
```

TigerLexer.java节选

```
// Generated from Tiger.g4 by ANTLR 4.2.1
import org.antlr.v4.runtime.Lexer;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.Token;
import org.antlr.v4.runtime.TokenStream;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.misc.*;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
public class TigerLexer extends Lexer {
    protected static final DFA[] _decisionToDFA;
    protected static final PredictionContextCache _sharedContextCache =
        new PredictionContextCache();
    public static final int
        T__39=1, T__38=2, T__37=3, T__36=4, T__35=5, T__34=6, T__33=7, T__32=8,
        T__31=9, T__30=10, T__29=11, T__28=12, T__27=13, T__26=14, T__25=15, T__24=16,
        T__23=17, T__22=18, T__21=19, T__20=20, T__19=21, T__18=22, T__17=23,
        T__16=24, T__15=25, T__14=26, T__13=27, T__12=28, T__11=29, T__10=30,
        T__9=31, T__8=32, T__7=33, T__6=34, T__5=35, T__4=36, T__3=37, T__2=38,
        T__1=39, T__0=40, INTEGER=41, STRING=42, ID=43, COMMENT=44, LINE_COMMENT=45,
        WS=46;
    public static String[] modeNames = {
        "DEFAULT_MODE"
    };
etc ...
```

TigerParser.java节选

```
// Generated from Tiger.g4 by ANTLR 4.2.1
import org.antlr.v4.runtime.atn.*;
import org.antlr.v4.runtime.dfa.DFA;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.misc.*;
import org.antlr.v4.runtime.tree.*;
import java.util.List;
import java.util.Iterator;
import java.util.ArrayList;

@SuppressWarnings({"all", "warnings", "unchecked", "unused", "cast"})
public class TigerParser extends Parser {
    protected static final DFA[] _decisionToDFA;
    protected static final PredictionContextCache _sharedContextCache =
        new PredictionContextCache();
    public static final int
        T__39=1, T__38=2, T__37=3, T__36=4, T__35=5, T__34=6, T__33=7, T__32=8,
        T__31=9, T__30=10, T__29=11, T__28=12, T__27=13, T__26=14, T__25=15, T__24=16,
        T__23=17, T__22=18, T__21=19, T__20=20, T__19=21, T__18=22, T__17=23,
        T__16=24, T__15=25, T__14=26, T__13=27, T__12=28, T__11=29, T__10=30,
        T__9=31, T__8=32, T__7=33, T__6=34, T__5=35, T__4=36, T__3=37, T__2=38,
        T__1=39, T__0=40, INTEGER=41, STRING=42, ID=43, COMMENT=44, LINE_COMMENT=45,
        WS=46;
    public static final String[] tokenNames = {
        "", "']'", "'&'", "'in'", "'of'", "','", "'['", "'-'", "'*'",
        "'while'", "'('", "':'", "'<'", "'<='", "'var'", "'array'", "'nil'", "'to'",
        "'{'", "'break'", "'let'", "'else'", "'}'", "'do'", "')'", "'function'",
        "'.'", "'+'", "'for'", "'<>'", "'='", "';'", "'if '", "'>'", "'type'",
        "':='", "'then'", "'/'", "'>='", "'|'", "'end'", "INTEGER", "STRING",
        "ID", "COMMENT", "LINE_COMMENT", "WS"
    };
etc ...
```

Tiger.g4语法文件

```
/**
   May,28 2014
   author: whimsycwd
   compiler project
*/

grammar Tiger;

program :  exp
        | decs
        ;

exp : 'nil'                                                     #Nil
     | INTEGER                                                  #Integer
     | STRING                                                   #String
     | type_id '[' exp ']' 'of' exp                             #Array
     | type_id '{' ( ID '=' exp (',' ID '=' exp )* )? '}'       #Record
     //| 'new' type_id                                            #New
     | lvalue                                                   #LeftValue
     | ID '(' ( exp (',' exp)*)? ')'                                #Call
     | '-' exp                                                      #UnaryMinus
     | exp ('*' | '/') exp                                      #Mul
     | exp ('+' | '-') exp                                      #Add
     | exp ('<>' | '=' | '>=' | '<=' | '>' | '<') exp           #Cmp
     | exp ('&' | '|') exp                                      #Logical
     | '(' exps ')'                                             #ParenExprs
     | lvalue ':=' exp                                          #Assign
     | 'if ' exp 'then' exp ('else' exp)?                       #IfStmt
     | 'while' exp 'do' exp                                     #WhileStmt
     | 'for' ID ':=' exp 'to' exp 'do' exp                          #ForStmt
     | 'break'                                                      #Break
     | 'let' decs 'in' exps 'end'                                   #LET
     ;

exps : ( exp (';' exp)* )?;

decs : dec*;

dec : 'type' ID '=' ty                                              #TypeDec
    | vardec                                                        #VarDecNothing
    | 'function' ID '(' tyfields? ')' (':' type_id)? '=' exp        #FuncDec
    //| 'primitive' ID '(' tyfields? ')' (':' type_id)?
    //| 'import' STRING
    //| 'class' ID ('extends' type_id)? '{' classfields '}'
    ;

vardec : 'var' ID (':' type_id)? ':=' exp                                   #VarDecInner
        ;
//classfields : classfield*
//        ;


//classfield : vardec
//          | 'method' ID '(' tyfields ')' (':' type_id) '=' exp
//           ;

ty : type_id
   | '{' tyfields? '}'
   | 'array' 'of' type_id
//   | 'class' ('extends' type_id)? '{' classfields '}'
   ;
tyfields : ( ID ':' type_id (',' ID ':' type_id)*)
    ;


type_id : ID
    ;
lvalue  : ID                           #SimpleVar
        | lvalue '.' ID                #DotVar
        | lvalue '[' exp ']'           #BracketVar
        ;



INTEGER :   DIGIT+;

STRING : '"' (ESC | .)*? '"';
ID : LETTER (LETTER | DIGIT) *;

fragment
ESC : '\\' [btnr"\\];

fragment
LETTER : [a-zA-Z] | '_';

fragment
DIGIT : [0-9];

COMMENT : '/*' .*? '*/'  ->skip;
LINE_COMMENT : '//' .*? '\n' ->skip;

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

五 AST测试

配置测试环境 config.profile

```
BASE="/Users/whimsycwd/Repo/ANTLR/Tiger/"

export CLASSPATH=".:./lib/antlr-4.2.1-complete.jar:${BASE}lib/antlr-4.2.1-complete.jar:${BASE}classes"

alias antlr4='java  org.antlr.v4.Tool'
alias grun='java org.antlr.v4.runtime.misc.TestRig'
alias runTiger='grun Tiger program'
```

上述代码为Terminal配置了环境变量。只需改变BASE到实际的地址。在Terminal中source config.profile，就可以使用antlr4,grun,runTiger的指令的简称。

下载测试代码 fetch.sh

```
URL=http://www.computing.dcu.ie/~hamilton/teaching/CA448/testcases/

rm *.tig

for i in {1..49}
do
    curl -O  "${URL}test${i}.tig"

done

for name in 'queens.tig' 'merge.tig'
do
     curl -O "${URL}$name"
done
```

新建文件夹testcases, 在该目录下执行上述的代码可以将测试文件批量下载下来。

编译运行 run.sh

```
# clean the generated file and recompile
# May 31,2014  whimsycwd


#source config.profile

rm Tiger*.java 2>/dev/null
#rm Tiger*.class 2>/dev/null
rm Tiger*.tokens 2>/dev/null
rm ./classes/* 2>/dev/null

echo "File deleted"

echo "ANTLR Generating"
java  org.antlr.v4.Tool Tiger.g4

echo "Compile Start"

#javac  Tiger*.java -d ./classes/
javac *.java -d ./classes/
```

输出测试AST结果 runAllTest.sh

```
# RUN ALL TEST
# May 31, 2014 whimsycwd


echo "deleting testcases-result"
rm testcases-result/*

for i in {1..49}
do
    echo "test${i}.tig is processing"
    java org.antlr.v4.runtime.misc.TestRig Tiger program -tree "testcases/test${i}.tig" > "testcases-
result/test${i}.tig.res"
done

for name in 'queens.tig' 'merge.tig'
do
    echo "${name} is processing"
    java org.antlr.v4.runtime.misc.TestRig Tiger program -tree "testcases/${name}" > "testcases-result/${name}.res"
done
```

新建testcases-result 利用上述代码能够批量输出测试代码到testcases-result/

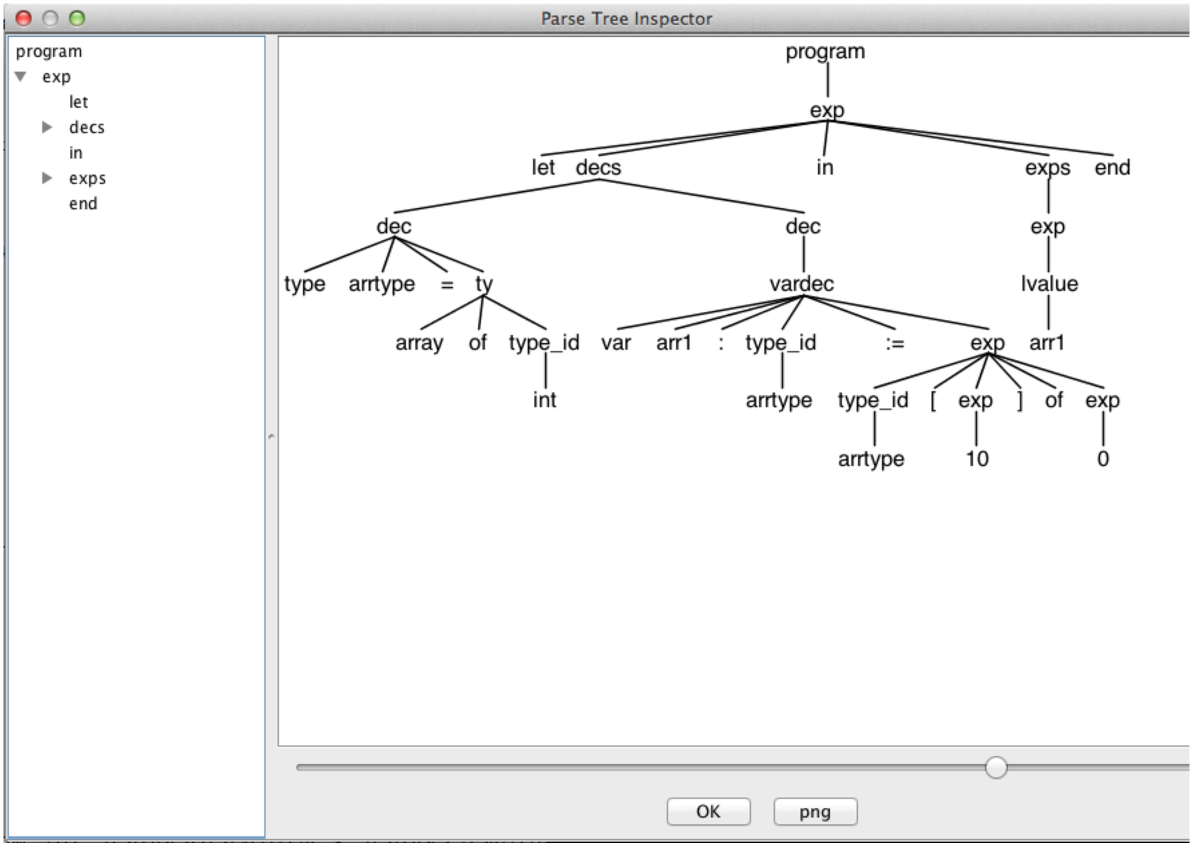
结果示例 1 testcases/test1.tig

```
/* an array type and an array variable */
let
    type  arrtype = array of int
    var arr1:arrtype := arrtype [10] of 0
in
    arr1
end
```

代码输出了AST结果文件： testcases-result/test.tig.res

```
(program (exp let (decs (dec type arrtype = (ty array of (type_id int))) (dec (vardec var arr1 : (type_id arrtype) := (exp
(type_id arrtype) [ (exp 10) ] of (exp 0))))) in (exps (exp (lvalue arr1))) end))
```

对应的图形化的结果为：


Parse Tree Inspector
program
exp
let
decs
in
exps
end
program
exp
let decs in exps end
dec dec exp
type arrtype = ty vardec lvalue
array of type_id var arr1 : type_id := exp arr1
int arrtype type_id [exp] of exp
arrtype 10 0
OK
png

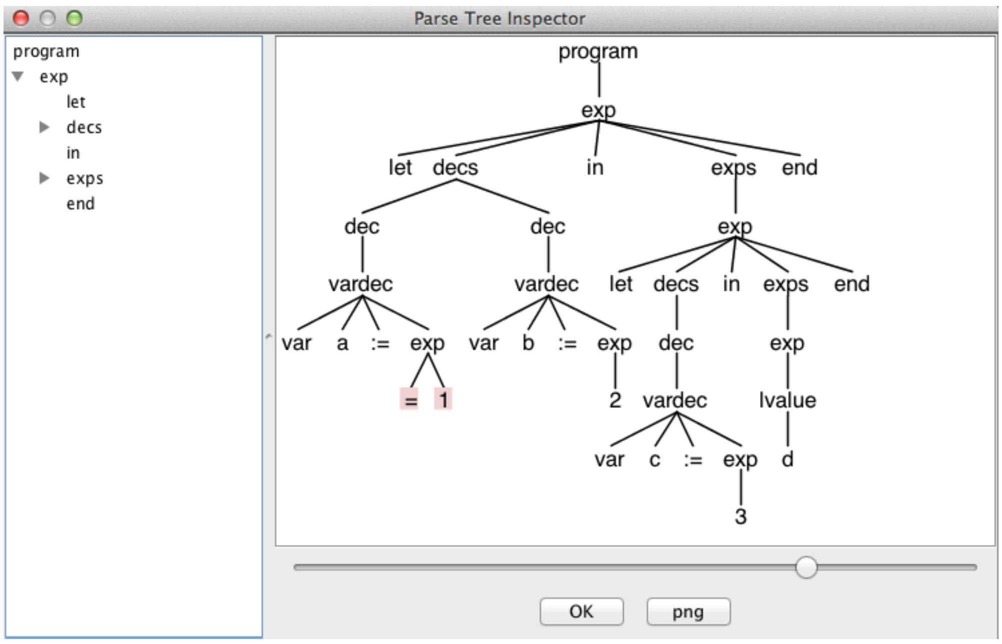
结果示例 2 mytest/error.in 错误提示和行号

```
 1 let
 2     var a:== 1
 3     var b:= 2
 4 in
 5     let
 6         var c := 3
 7     in
 8         d
 9     end
10 end
```

代码输出结果

```
whimsyMini:Tiger whimsycwd$ runTiger -tree error.in
line 2:11 no viable alternative at input '='
(program (exp let (decs (dec (vardec var a := (exp = 1))) (dec (vardec var b := (exp 2)))) in (exps (exp let (decs (dec (vardec
var c := (exp 3)))) in (exps (exp (lvalue d))) end)) end))
```

对应的图形化的结果,追到错误的地方，被高亮出来了，并且跳过继续解析。



六 变量作用域检查

为了实现变量声明的检查，两次遍历抽象语法树，第一次遍历抽象语法树，简历变量表，注意在遍历的过程中，还要考虑作用域，进入一个作用域的时候需要创建新的作用域。作用域构成一个链表，resolve一个变量的时候需要跟往上层去查找。

CheckSymbols.java

```
...
    public void process(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);
        }
        ANTLRInputStream input = new ANTLRInputStream(is);
        TigerLexer lexer = new TigerLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        TigerParser parser = new TigerParser(tokens);
        parser.setBuildParseTree(true);
        ParseTree tree = parser.program();
        // show tree in text form
//          System.out.println(tree.toStringTree(parser));

        ParseTreeWalker walker = new ParseTreeWalker();
        DefPhase def = new DefPhase();
        walker.walk(def, tree);
        // create next phase and feed symbol table info from def to ref phase
        RefPhase ref = new RefPhase(def.globals, def.scopes);
        walker.walk(ref, tree);
        //TypePhase type = new TypePhase();
        //walker.walk(type, tree);
    }
...
```

具体的代码 `CheckSymbols.java` `DefPhase.java` `Scope.java` `BaseScope.java` `GlobalScope.java` `LocalScope.java` `RefPhase.java`

第一阶段DefPhase,如下所示，比如在进入Function的时候`currentScope = new LocalScope(currentScope);` 新建一个作用域。在退出Function的时候 `currentScope = currentScope.getEnclosingScope();` 往链表上层去找。

DefPhase.java

```
...
    public void enterFuncDec(TigerParser.FuncDecContext ctx){
        // formal parameter
        //if (debug)
        //    System.out.println("Entering FuncDec "+ currentScope);
        currentScope = new LocalScope(currentScope);

        TigerParser.TyfieldsContext tyfields = ctx.tyfields();
        if (tyfields != null){
            // if there is formal parameters.
            List list = tyfields.ID();
            for (TerminalNode e : list){
                currentScope.define(new VariableSymbol( e.getSymbol().getText() ) );  // 把变量添加到当前作用域
            }
        }
        saveScope(ctx, currentScope);
    }
    public void exitFuncDec(TigerParser.FuncDecContext ctx){

        //if (debug)
        //System.out.println("Exiting FuncDec "+ currentScope);
        //System.out.println(currentScope);
        currentScope = currentScope.getEnclosingScope();   // 退出作用域
    }
...
```

检查变量varName是否在存在。解析变量代码如下。

BaseScope.java

```
    public Symbol resolve(String name) {
        Symbol s = symbols.get(name);
        if ( s!=null ) return s;
        // if not here, check any enclosing scope
        if ( enclosingScope != null ) return enclosingScope.resolve(name);
        return null; // not found
    }
```

在第二阶段RefPhase当变量解析不到该变量，输出该类型的错误。

RefPhase.java

```
...
    public void exitSimpleVar(TigerParser.SimpleVarContext ctx){

        String name = ctx.ID().getSymbol().getText();

        Symbol var = currentScope.resolve(name);
       // print(currentScope);
        if (var == null){
            CheckSymbols.error(ctx.ID().getSymbol(),"no such variable: " + name);
        }
...
```

七 变量声明测试

结果示例 mytest/test.in

```
let
    var a:= 1
    var b:= 2
in
    let
        var c := 3
    in
        d
    end
end
```

注意到这个代码中 d并找不到此变量。可得到输出结果

```
whimsyMini:Tiger whimsycwd$ java CheckSymbols mytest/test.in
line 8:8 no such variable: d
```

八 基本的类型检查

为了实现基本的类型检查，对于每一个抽象语法树上的节点都需要给他一个类型。定义了一个AST节点到Type的映射typeSys.

```
'for' ID ':=' exp 'to' exp 'do' exp
```

下面的代检查了三个exp的类型。 exp1 和 exp2必须是Type.tInt。而对于整个语句的类型定义为exp3.

RefPhase.java

```
...
        Type t1 = typeSys.get(ctx.exp(0));
        Type t2 = typeSys.get(ctx.exp(1));

        if (t1 != Type.tInt){
            CheckSymbols.error( ctx.getStart(), "loop pd expression must be tInt" );
            typeSys.put(ctx, Type.tVoid);
        } else {
            typeSys.put(ctx, t2);
        }
        print("In WhileStmt : " + typeSys.get(ctx) );
    }

    public void exitForStmt(TigerParser.ForStmtContext ctx){
        Type t1 = typeSys.get(ctx.exp(0));
        Type t2 = typeSys.get(ctx.exp(1));
        Type t3 = typeSys.get(ctx.exp(2));

        if (t1 != Type.tInt || t2 != Type.tInt){
            CheckSymbols.error( ctx.getStart(), "loop start and end must both be tInt" );
            typeSys.put(ctx, Type.tVoid);
        } else {
            typeSys.put(ctx, t3);
        }
        print("In ForStmt " + typeSys.get(ctx));
        currentScope = currentScope.getEnclosingScope();
    }

    public void exitLET(TigerParser.LETContext ctx){
        typeSys.put(ctx, typeSys.get( ctx.exps() ) );
        currentScope = currentScope.getEnclosingScope(); // exit a scope
        print("In LET: "+typeSys.get(ctx) );
    }
    public void exitExps(TigerParser.ExpsContext ctx){
        List list = ctx.exp();
        typeSys.put(ctx, typeSys.get( list.get(list.size() -1 ) ) );
        print("In exps: "+  typeSys.get(ctx) );

    }
...
```

九 基本的类型检查测试

结果示例 mytest/add.in

```
 1 let
 2     var a := nil
 3     var b := 12
 4     var c := "adfaf"
 5     var d := 133
 6 in
 7     -c;
 8     -b;
 9     b+d;
10     b-d;
11     a+c;
12     b/d;
13     b*d;
14     b*(d+b);
15     d := b+d
16
17 end
```

这个代码的输出结果，发现了其类型的错误。注意到line 7 -c,其中c是一个String错误。line 11 a + c, 只有Int有定义 加法。

```
whimsyMini:mytest whimsycwd$ java CheckSymbols add.in
line 7:4 Expected - Int, got - tString
line 11:4 Expected Int +/- Int got tVoid +/- tString
```