



The Document of Tigger Compiler Project

复旦大学编译原理课程实践

指导老师：邱锡鹏

xpqiu@fudan.edu.cn

项目人员

11300240049 & 11300240068

单张卿 & 牙河清

2014 年 6 月

目录

项目简介.....	3
软件使用手册.....	4
网页版.....	4
桌面版.....	7
需求分析与背景介绍.....	10
开发描述与设计架构.....	11
词法分析.....	12
语法分析.....	13
语义分析细节.....	14
抽象语法树.....	20
符号表与变量作用域.....	21
错误处理.....	22
体会与总结.....	29
ANTLR 的使用	29
开发过程.....	29

项目简介

本项目主要实现了如下功能：

- 网页版 IDE，网址：<http://tigger.namron.org/>
 - 基于 highlightjs 的语法高亮
 - 基于 d3 的抽象语法树和符号表树生成（可使用鼠标拖拽和滚轮放大）
 - B/S 架构的远程编译并返回编译结果
- 桌面版 IDE
 - 语法高亮、代码折叠
 - 抽象语法树、符号表的文件夹树形显示
 - 配色调整
- 编译器
 - 通过了绝大部分（90%+）的测试源码文件的测试
 - 正确的词法、语法分析，参见对应的文法文件
 - 正确的抽象语法树生成（可以在网页版里方便地预览）
 - 错误处理功能
 - ◆ 详细的提示错误类型（包括词法错误、语法错误、语义错误等），以及出错位置的行号
 - ◆ 错误修复，即可以跳过已经发现的错误继续编译过程，找到下文可能的错误
 - 发挥想象力的功能：
 - ◆ 字符串引号自动匹配
 - ◆ 注释嵌套的层次判断与错误修正
 - ◆ 参见网页版 IDE 和桌面版 IDE

软件使用手册

网页版

网页版的 IDE 提供了一个 B/S 模式的提交和编译方法，具有极好的跨平台特性，而且可以很方便地进行更新。同时编译过程在服务器端完成，对于客户端的配置要求很低。开发过程中我们发现 ANTLR 对内存的要求非常高，所以网页端能够很好地让服务器承担这一部分开销。同时，网页端能够更方便地开发出美观的界面。

网页版的测试网址：<http://tigger.namron.org/>

TIGER COMPILER

Authors: 单张卿 | 牙河清

浏览... 未选择文件。 **UPLOAD**

SUBMIT SOURCECODE VIEW AST TREE VIEW SYMBOL TREE

Source Code:

```
1. /* an array type and an array variable */
2. let
3.   type arrtype = array of int
4.   var arr1:arrtype := arrtype [10] of 0
5. in
6.   arr1
7. end
8.
```

Compile Messages

Compile Succeeded.

在右上角的文件选择框中选择相应的文件上传，即可将源代码发送到服务器进行编译，编译完成后，上传的源代码会在相应的编辑框中高亮显示，同时底部会有相应的编译信息，如果有编译错误也会有相应的行号提示。

TIGER COMPILER

Authors: 单张卿 | 牙河清

浏览...

未选择文件。

UPLOAD

SUBMIT SOURCECODE

VIEW AST TREE

VIEW SYMBOL TREE

Source Code:

```
1. /* error : compare rec with array */
2.
3. let
4.
5.   type arrtype = array of int
6.   type rectype = {name:string, id: int}
7.
8.   var rec := rectype {name="aname", id=0}
9.   var arr := arrtype [3] of 0
10.
11. in
12.   if rec <> arr then 3 else 4
13. end
14.
```

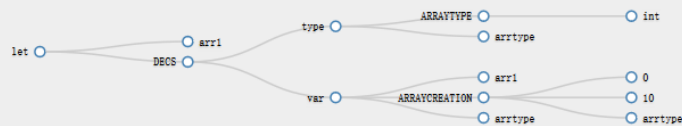
Compile Messages

Semantic Error on line 12: The two sides of operator do not balance!

选择导航栏中的 VIEW AST TREE 可以看到最近一次成功生成的抽象语法树的结构，可以使用鼠标拖拽，以及滚轮放大缩小。如果编译失败，那么不会生成新的抽象语法树，如果抽象语法树在成功编译后没有更新，请使用 CTRL+F5 强制刷新浏览器缓存。

Symbol Tree

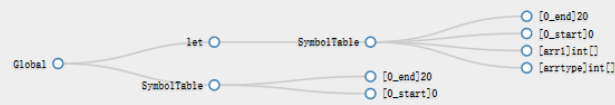
This Page displays the AST tree of the last successfully compiled sourcefile.
you can use mouse wheel to zoom in / zoom out
Please use **ctrl + F5** to clear the **cache**.



点击导航栏中的 VIEW SYMBOL TREE 可以看到最近一次成功生成的符号表（以树形结构显示）。

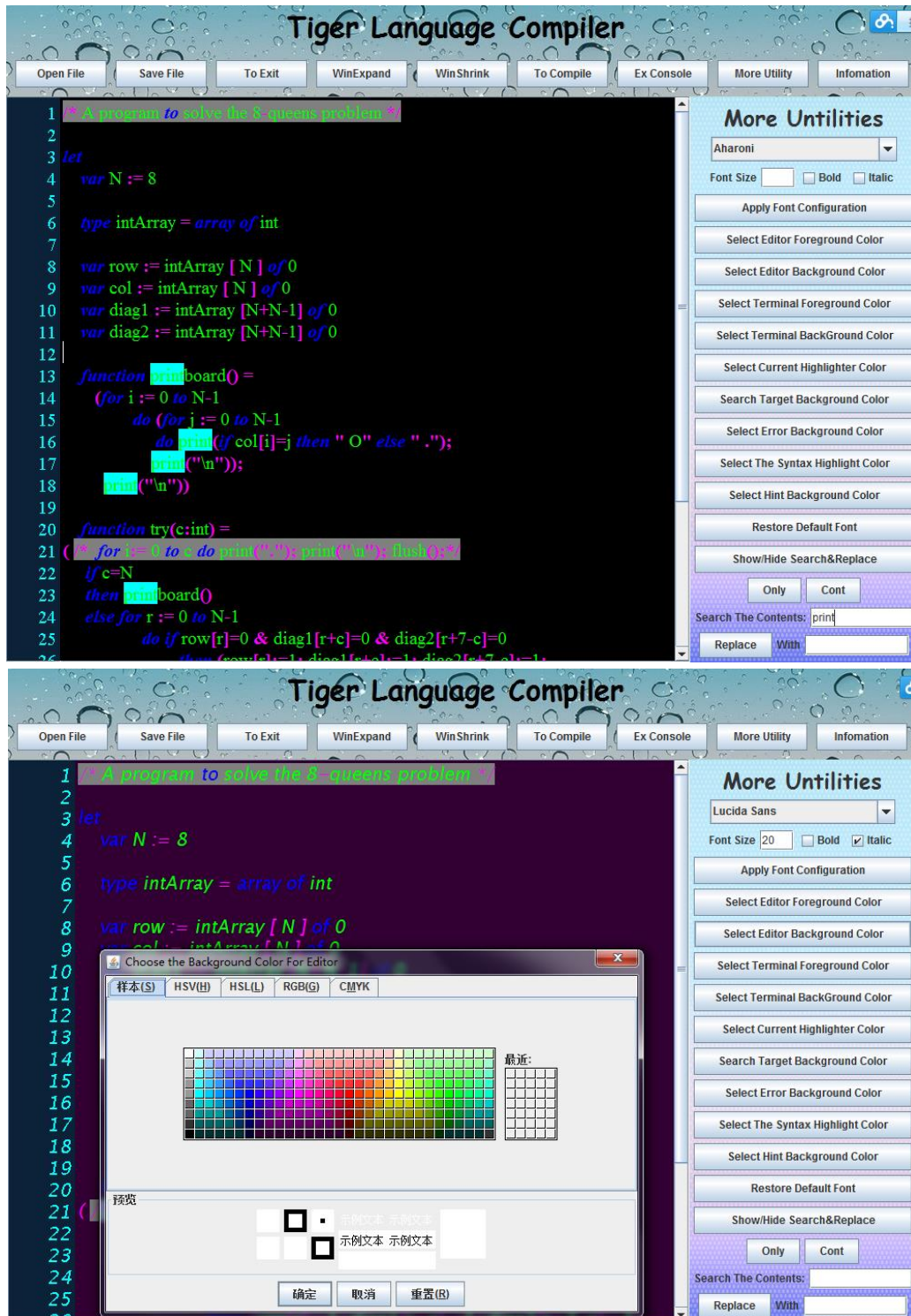
Symbol Tree

This Page displays the symbol tree of the last successfully compiled sourcefile.
you can use mouse wheel to zoom in / zoom out
Please use **ctrl + F5** to clear the **cache**.



桌面版

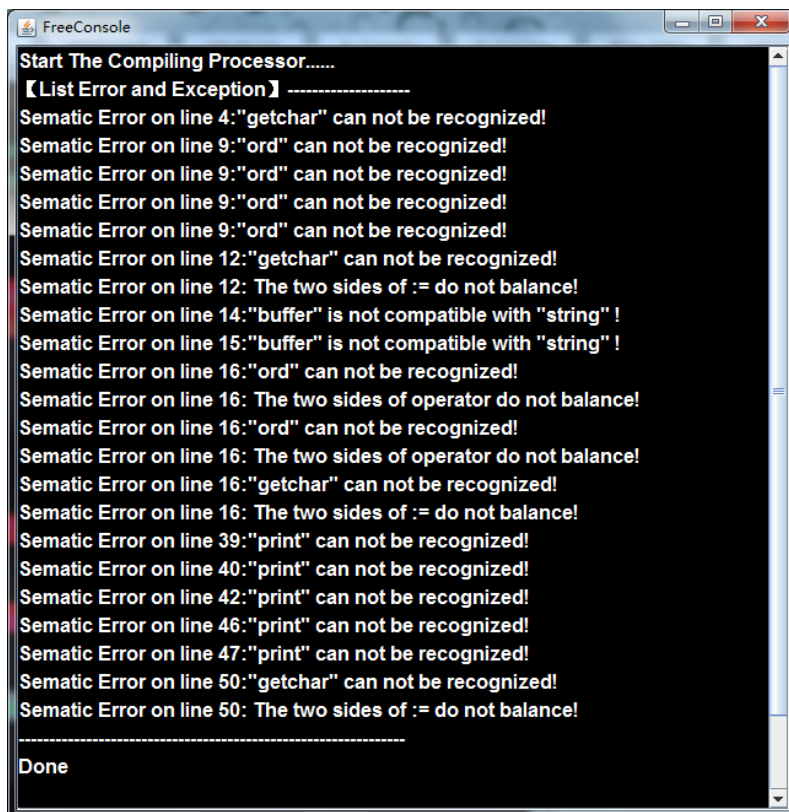
Desktop IDE 简单易用, 所有功能键都在上方, 一目了然。可以点击 More Utility 按钮, 其中有 Search&Replace 功能, 可以帮助用户更好地编辑代码 (print 作为搜索关键词被高亮)。



用户也可以使用其他的代码风格配置, 右侧一栏是调整字体颜色以及语法高亮功能键,

你可以自己配置最佳风格。

点击 Ex Console 可以体验到最原始而质朴的 CMD 风格：



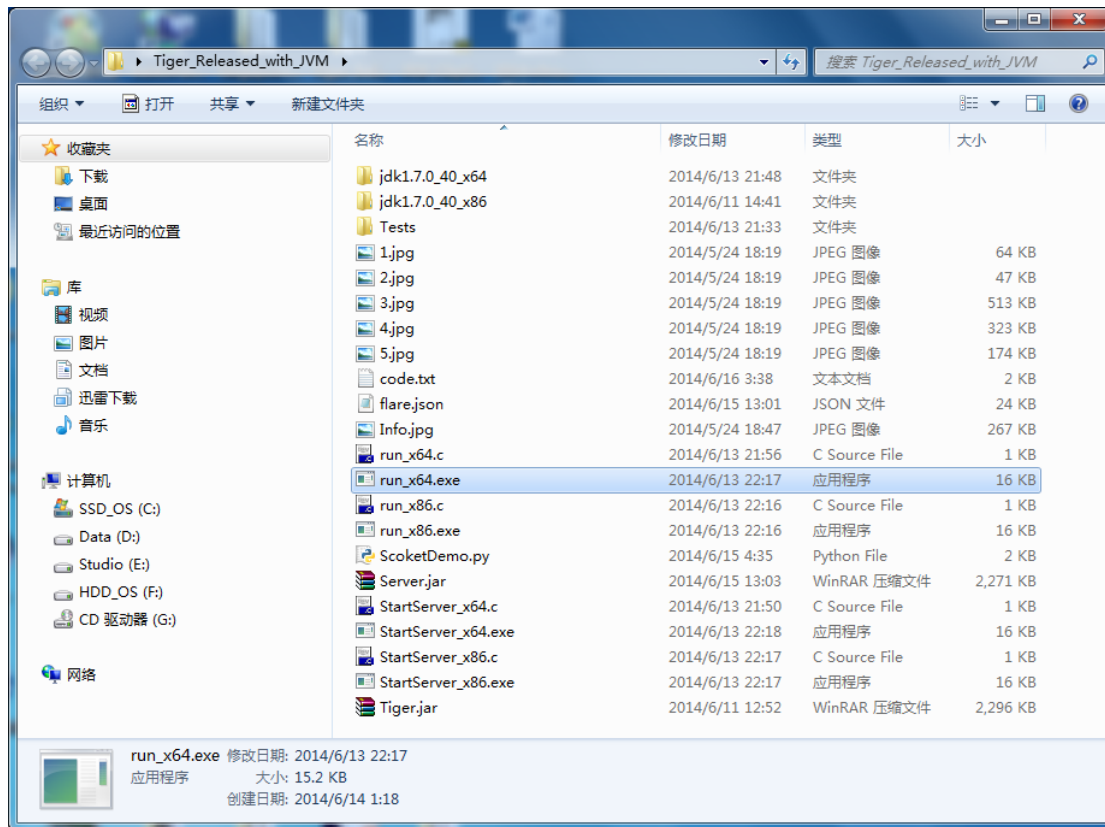
```
FreeConsole
Start The Compiling Processor.....
【List Error and Exception】-----
Sematic Error on line 4:"getchar" can not be recognized!
Sematic Error on line 9:"ord" can not be recognized!
Sematic Error on line 9:"ord" can not be recognized!
Sematic Error on line 9:"ord" can not be recognized!
Sematic Error on line 9:"ord" can not be recognized!
Sematic Error on line 12:"getchar" can not be recognized!
Sematic Error on line 12: The two sides of := do not balance!
Sematic Error on line 14:"buffer" is not compatible with "string"!
Sematic Error on line 15:"buffer" is not compatible with "string"!
Sematic Error on line 16:"ord" can not be recognized!
Sematic Error on line 16: The two sides of operator do not balance!
Sematic Error on line 16:"ord" can not be recognized!
Sematic Error on line 16: The two sides of operator do not balance!
Sematic Error on line 16:"getchar" can not be recognized!
Sematic Error on line 16: The two sides of := do not balance!
Sematic Error on line 39:"print" can not be recognized!
Sematic Error on line 40:"print" can not be recognized!
Sematic Error on line 42:"print" can not be recognized!
Sematic Error on line 46:"print" can not be recognized!
Sematic Error on line 47:"print" can not be recognized!
Sematic Error on line 50:"getchar" can not be recognized!
Sematic Error on line 50: The two sides of := do not balance!
-----
Done
```

作为 Java 编码的 IDE，其好处在于跨平台且兼容性好。但是对于不熟悉 Java 的用户而言，配置 JDK 会很头疼。我们考虑到这一点后，发布了一个自带 JVM（Java 虚拟机 7.40 版）的 IDE，使得其在任何 window 平台（x86, x64, xp, win7, win8）下都能直接打开，不需要繁琐的配置，实现“一次发布，到处执行”的超高兼容性。（缺点是文件体积大，超过 400MB）

如果用户需要在 linux 下使用，我们同样也有 jar 文件打包 IDE，也能满足用户需求（默认 linux 用户对 Java 会比较了解，能自己配置 JDK）。

只要点击 run_x64.exe 和 run_x86.exe 用户就能以不同的方式开启 IDE（建议使用 x64 模式，ANTLR 对内存要求较高）。

StartServer_x64.exe 和 StartServer_x86.exe 是我们实现的编译服务器，运行后监听端口 4968，以 TCP Socket 方式通信。通信的具体实例在 SocketDemo.py（用 Python 写的 Client 程序）中以实际可运行代码的形式给出。



需求分析与背景介绍

Tigger 语言是一种教学语言，现实中并不存在。它仅存在于编译原理（虎书）的纸页中。但这并不妨碍初学者对它的执着钻研。在好奇心的驱使下，众多学习虎书的学子们开始了实验项目。所以，我们的这个实验不仅是对自身学习水平的一个检验，更是对后续初学者的一种无私馈赠。下面的 Tigger 语言是专门为 8 皇后算法设计的，从中可以一睹其风采。

```

1  * A program to solve the 8 queens problem *
2
3  let
4    var N := 8
5
6    type intArray = array of int
7
8    var row := intArray [ N ] of 0
9    var col := intArray [ N ] of 0
10   var diag1 := intArray [N+N-1] of 0
11   var diag2 := intArray [N+N-1] of 0
12
13   function printboard() =
14     (for i := 0 to N-1
15       do (for j := 0 to N-1
16         do print(if col[i]=j then " O" else " .");
17         print("\n"));
18     print("\n"))
19
20   function try(c:int) =
21   ( * for i:= 0 to c do print("."); print("\n"); flush(); *
22     if c=N
23     then printboard()
24     else for r := 0 to N-1
25       do if row[r]=0 & diag1[r+c]=0 & diag2[r+7-c]=0

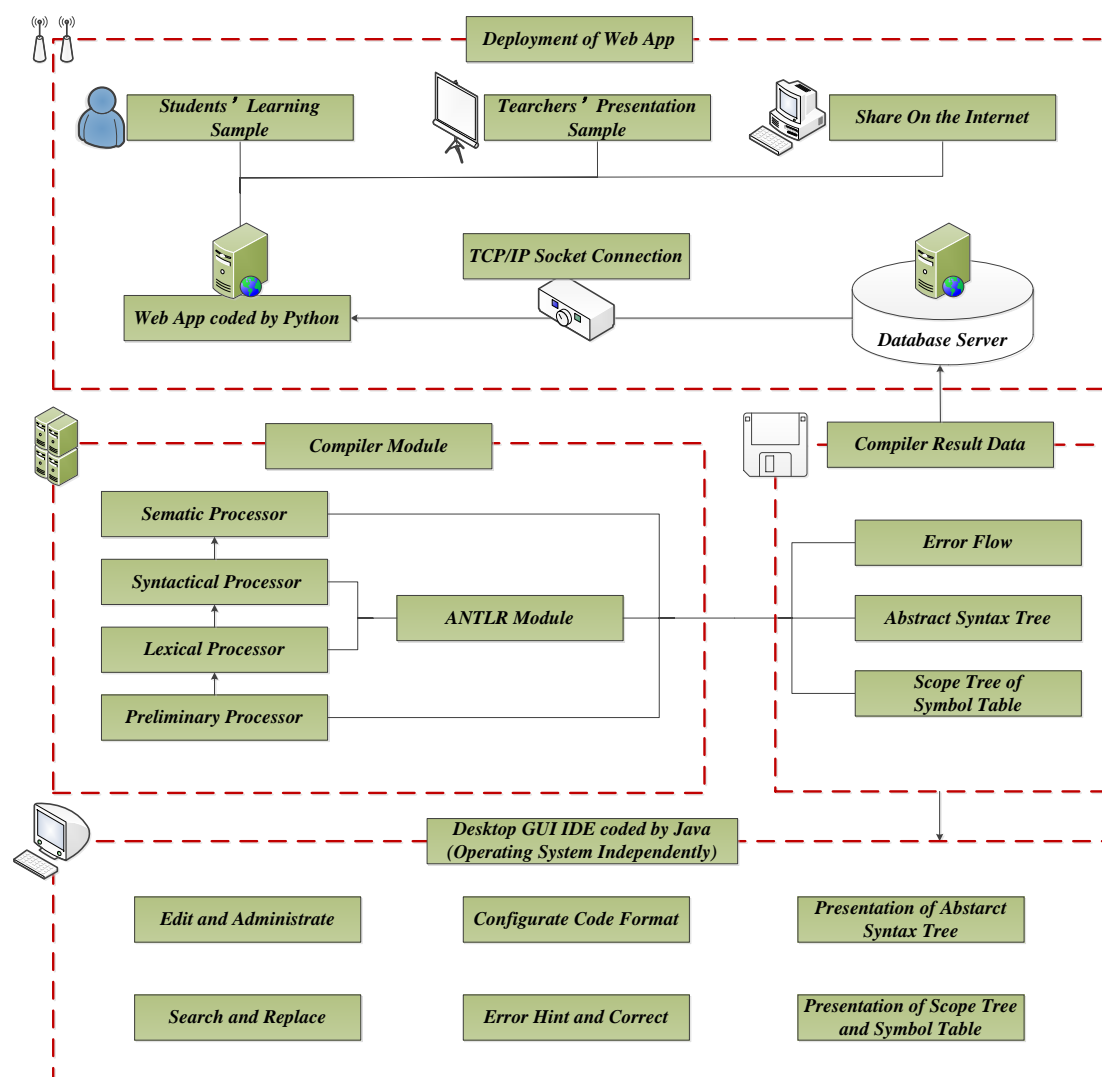
```

为了更好地理解编译原理的深层机制，为了更深入地锻炼自己的编程能力，为了更无私地为后续的学弟学妹们提供学习编译原理的直观实例，我们日以继夜地工作，最终圆满完成了这个项目。可以说，对知识的渴望和对实践的执着是这个项目最大的需求也是我们最强劲的动力。

在实现这个项目的过程中，我们主要参考了下列资料：

- 【1】《现代编译原理-C 语言描述》
- 【2】<http://www.credmond.net/projects/tiger-parser/>
- 【3】<http://strategox.org/Tiger/CompilerPackages>
- 【4】<http://www.lrde.epita.fr/~akim/ccmp/tiger.html>
- 【5】<http://www.computing.dcu.ie/~hamilton/teaching/CA448/testcases/index.html>

开发描述与设计架构



The Structure of Tigger Compiler IDE Application

本次项目中，我们主要采用两种编程语言 Java 和 Python，其中 Java 语言负责编译器的内核代码编写和桌面版 IDE 的设计。而 Python 语言主要负责网络传输控制、网页设计和网络访问交互的工作。完成这个项目后，我们将会得到两个 IDE 应用。一个是桌面版的 IDE 应用（Desktop IDE），另一个是网页版的 IDE 应用（Web IDE）。Desktop IDE 自带 JVM 虚拟机，适用于各种版本的 windows 程序，在 32bit 和 64bit 环境下都有对应的启动程序（模仿 eclipse），而 Web IDE 主要为了实现快捷的跨平台访问，使得我们的项目能更大程度上得到推广。

在编译器的内核中，我们用到了 ANTLR3.4 的 jar 包，作为词法分析和语法分析的工具。ANTLR 全称为 Another Tool for Language Recognition，其前身是 PCCTS，它为包括 Java，C++，C# 在内的语言提供了一个通过语法描述来自动构造自定义语言的识别器（recognizer），编译器（parser）和解释器（translator）的框架。ANTLR 可以通过断言（Predicate）解决识别冲突，支持动作（Action）和返回值（Return Value）。其他具体的设计架构在上图中描述地很清楚了，这里不再繁篇累牍地介绍。

词法分析

在经过 Tigger 代码的预处理和重新编码后，代码流需要输入到词法分析器中以便取得合适的 Token 序列。下面就是我们在 ANTLR 中设定的词法规则。

```

3 // integer and string
4 INT : DIGIT+;
5 ID : PRINTABLECHAR (PRINTABLECHAR|DIGIT)*;
6
7 // rename single or double characters
8 fragment
9 DIGIT : '0'..'9';
10 fragment
11 PRINTABLECHAR : 'a'..'z'|'A'..'Z'|'_';
12
13 FORMATCHAR : (' '\t'\n'\f'|BACKSLASH BACKSLASH){skip()};
14 COMMENT : '/*' (options{greedy=false;}:~('/'|(( '/' ~('*')))*
15 (COMMENT (options{greedy=false;}:~('/'|(( '/' ~('*')))*)* '*/' {skip()});
16
17
18

```

词法规则和语法规则都在 Tiger.g 文件中。下面具体介绍词法的内容。

- 【1】 定义数字是单个数字字符组成的正闭包（第 3 行、第 9 行）
- 【2】 定义标示符只能以非数字一般字符开头（第 5 行）
- 【3】 定义空各类字符为 token 间的分界线（第 13 行）
- 【4】 定义注释由（*.....*\、//）组成，并确定其内部的嵌套关系（第 14 行、第 15 行）

保留的关键字为：

```

'function'; 'if'; 'then'; 'else'; 'while'; 'for'; 'do';
'break'; 'let'; 'in'; 'end'; 'of'; 'to'; 'type';
'var'; 'array';

```

涉及到的符号为：

```

';' '{' '}' '[' ']' '(' ')'; '=' '!=' '+' '*' '/' ':='; ';;' '""'; '>'; '<>';
'<'; '>=' '<=' '&'; '|'; '||'; '||';

```

语法分析

下面给出具体的文法表达式 (终结符用红色表示) :

<i>prog</i> →	<i>exp</i>
<i>decs</i> →	(<i>dec</i>)*
<i>dec</i> →	<i>tydec</i> / <i>vardec</i> / <i>fundec</i>
<i>tydec</i> →	type <i>ID</i> = <i>ty</i>
<i>ty</i> →	<i>ID</i> / { <i>tyfie</i> } / {} / array of <i>ID</i>
<i>tyfields</i> →	(<i>ID</i> : <i>ID</i>)+
<i>vardec</i> →	var <i>ID</i> : <i>ID</i> := <i>exp</i> / var <i>ID</i> := <i>exp</i>
<i>fundec</i> →	function <i>ID</i> (<i>tyfields</i>) = <i>exp</i> / function <i>ID</i> (<i>tyfields</i>): <i>ID</i> = <i>exp</i> / function <i>ID</i> (). <i>ID</i> = <i>exp</i> / function <i>ID</i> () = <i>exp</i>
<i>exp</i> →	(<i>exp</i> (; <i>exp</i>)+) / - <i>exp</i> / <i>ifexp</i> / while <i>exp</i> do <i>exp</i> * / for <i>ID</i> := <i>exp</i> to <i>exp</i> do <i>exp</i> * / let <i>decs</i> in (<i>exp</i> (; <i>exp</i>)*) end / <i>ID</i> () / <i>ID</i> (<i>exp</i>) of <i>exp</i> / <i>lvalue</i> := <i>exp</i> / <i>biopexp</i>
<i>ifexp</i> →	if <i>exp</i> then <i>exp</i> else <i>exp</i> / if <i>exp</i> then <i>exp</i>
<i>biopexp</i> →	<i>cmpexp</i> ((/ &) <i>cmpexp</i>)+ / <i>cmpexp</i>
<i>cmpexp</i> →	<i>addexp</i> (= / != / >= / <= / ==) <i>addexp</i> / <i>addexp</i>
<i>addexp</i> →	<i>multiexp</i> ((+ / -) <i>multiexp</i>)+ / <i>multiexp</i>
<i>multiexp</i> →	<i>term</i> ((* /) <i>term</i>)+ / <i>term</i>
<i>term</i> →	<i>STRING</i> / <i>INT</i> / <i>BREAK</i> / <i>functioncall</i> / <i>lvalue</i> / () / (<i>exp</i>)
<i>functioncall</i> →	<i>ID</i> (<i>exp</i> (, <i>exp</i>)*) / <i>ID</i> ()
<i>dotlvalue</i> →	<i>ID</i> (. <i>ID</i>)+
<i>arrayindexvalue</i> →	<i>ID</i> [<i>exp</i>]+
<i>lvalue</i> →	<i>arrayindexlavlue</i> (. <i>lvalue</i>)+ / <i>dotlvalue</i> ((<i>exp</i>)+) / <i>dotlvalue</i> / <i>arrayindexlvalue</i> / <i>ID</i>
<i>BREAK</i> →	break
<i>INT</i> →	<i>DIGIT</i> +
<i>ID</i> →	<i>PRINTABLECHAR</i> (<i>PRINTABLECHAR</i> / <i>DIGIT</i>)*
<i>STRING</i> →	“ ((/ ”) / <i>ESCAPEDSEQ</i>)* ”
<i>ESCAPEDSEQ</i> →	\n / \t / ^c / <i>DIGIT DIGIT DIGIT</i> / /
<i>DIGIT</i> →	['0'..'9']
<i>PRINTABLECHAR</i> →	['a'..'z' / 'A'..'Z' / ' _']

通过输入上述的语法到 ANTLR 模块, 我们就可以构建出一个基于 LL (k) 的语法分析器。

语义分析细节

我们按照 Tiger 语言的所有要求进行了语义分析设计，通过遍历 AST 树上的相关节点，分析出任何一个 Token 的类型信息，进行类型兼容检查、参数数量检查、初始化声明检查等等的语义分析。具体的语义分析代码在源文件 SemanticParser.java 中（约 600 行）。其中的核心代码如下：

```

1. //用DFS的方法遍历AST数，根据树的节点类型和父节点与子节点关系判断是否合乎语义
2. public String SemanticDFS(DefaultMutableTreeNode father){
3. //将AST中遍历到的节点作为父亲节点，判断父亲节点（语句）与子节点（参数）之间的关系
4.     try{
5.         DefaultMutableTreeNode temp=(DefaultMutableTreeNode)father.getFirstChild();
6.         if(father.toString().equals("RECORDCREATION")){
7.             return "nil";
8.         }else if(father.toString().equals(".")){//记录类型检查
9.             DefaultMutableTreeNode
tmp=(DefaultMutableTreeNode) father.getChildAfter(temp);
10.             ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
11.             int index=ASTNodeIndex.get(temp);
12.             String type1=env[index].get(temp.toString());
13.             if(env[index].get(type1)!=null) type1=(String)env[index].get(type1);
14.             if(type1.indexOf("{")==-1){//检查"."号之前是否是记录类型
15.                 System.err.println
16.                     ("Semantic Error on line "+Row[index]+": "+temp.toString()+" is not a
record type");
17.                 return "UnKnow";
18.             }else if((type1.indexOf(tmp.toString())==-1)//检查记录类型中是否由此项纪录
19.                 || (type1.indexOf(tmp.toString()+":")==-1)
20.                 || ((type1.indexOf("{ "+tmp.toString()+":")==-1)
21.                 && type1.indexOf(", "+tmp.toString()+":")==-1)){
22.                 System.err.println
23.                     ("Semantic Error on line "+Row[index]+": "+temp.toString()+"
24.                     " does not have name space "+tmp.toString()+" !");
25.                 return "UnKnow";
26.             }else{//否则，将记录类型中对应的纪录项的类型返回
27.                 int p=type1.indexOf(tmp.toString()+tmp.toString().length()+1);
28.                 int q=p;
29.                 while((type1.charAt(q)!=' ' && (type1.charAt(q)!='}')) q++;
30.                 return type1.substring(p,q);
31.             }
32.         }else if(father.toString().equals("ARRAYINDEX")){//检查数组类型
33.             DefaultMutableTreeNode

```

```

tmp=(DefaultMutableTreeNode) father.getChildAfter(temp);

34.                String type1=SematicDFS(temp);
35.                ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
36.                String type2=SematicDFS(tmp);
37.                int index=ASTNodeIndex.get(temp);
38.                if(env[index].get(type1)!=null) type1=(String)env[index].get(type1);
39.                if(!PrototypeContain("int",type2,env[index])){//数组类型的下标必须是整数
40.                    System.err.println
41.                        ("Sematic Error on line "+Row[index]+": The Index of Array must be
Integer!");
42.                }
43.                if(type1.indexOf("[]")==-1){//只能对数组类型取索引值
44.                    System.err.println
45.                        ("Sematic Error on line "+Row[index]+": "+temp.toString()+" is not
an array!");
46.                }
47.                return type1.substring(0,type1.indexOf("[]")); //返回数组类型取出的元素值类型
48.            }else if(father.toString().equals("for")){//检查for语句
49.                DefaultMutableTreeNode
tmp2=(DefaultMutableTreeNode) father.getChildAfter(temp);
50.                DefaultMutableTreeNode
tmp3=(DefaultMutableTreeNode) father.getChildAfter(tmp2);
51.                String type1=SematicDFS(temp);
52.                ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
53.                String type2=SematicDFS(tmp2);
54.                String type3=SematicDFS(tmp3);
55.                int index=ASTNodeIndex.get(temp);
56.                if(!PrototypeContain("int",type1,env[index])){//for语句的三个参数都是整数
57.                    !PrototypeContain("int",type2,env[index])||
58.                    !PrototypeContain("int",type3,env[index])}{
59.                    System.err.println
60.                        ("Sematic Error on line "+Row[index]+": The elements in for is not
Integer");
61.                }
62.                return "UnKnow";
63.            }else if(father.toString().equals(":=")){//检查赋值语句
64.                DefaultMutableTreeNode
tmp=(DefaultMutableTreeNode) father.getChildAfter(temp);
65.                String type1=SematicDFS(temp);
66.                ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
67.                String type2=SematicDFS(tmp);
68.                int index=ASTNodeIndex.get(temp);
69.                if(!PrototypeContain(type1,type2,env[index])){//检查赋值语句两边的类型是否兼容
70.                    System.err.println
71.                        ("Sematic Error on line "+Row[index]+": The two sides of := do not

```

```

balance!");
72.                }
73.                return type1;
74.                }else if(father.toString().equals("<>")){//对不等号两边的类型判断
75.                    DefaultMutableTreeNode
tmp=(DefaultMutableTreeNode) father.getChildAfter(temp);
76.                    String type1=SematicDFS(temp);
77.                    ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
78.                    String type2=SematicDFS(tmp);
79.                    int index=ASTNodeIndex.get(temp);
80.
            if(!PrototypeContain(type1,type2,env[index])&&!PrototypeContain(type2,type1,env[index])){
81.                System.err.println
82.                ("Sematic Error on line "+Row[index]+": The two sides of operator do
not balance!");
83.                }//不等号两边的类型必须至少有一个能兼容对方
84.                return "bool";
85.                }else if(father.toString().equals("-")){//检查负号或者减号字句
86.                    DefaultMutableTreeNode
tmp=(DefaultMutableTreeNode) father.getChildAfter(temp);
87.                    String type1=SematicDFS(temp);
88.                    ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
89.                    int index=ASTNodeIndex.get(temp);
90.                    if(tmp==null){//只有一个参数, 既为负号, 进行判断
91.                        if(!PrototypeContain("int",type1,env[index]))
92.                            System.err.println
93.                            ("Sematic Error on line "+Row[index]+": The - operator does not
have an Integer!");
94.                        return "int";
95.                    }
96.                    String type2=SematicDFS(tmp);
97.                    if(!PrototypeContain("int",type1,env[index])
|| !PrototypeContain("int",type2,env[index])){
98.                        System.err.println
99.                        ("Sematic Error on line "+Row[index]+": The two sides of operator do
not balance!");
100.                    }//判断减号两边是否是类型兼容的
101.                    return "int";
102.                }else if(OperatorSet.contains(father.toString())){//对其他双目操作符进行两边参数类型
校验
103.                    DefaultMutableTreeNode
tmp=(DefaultMutableTreeNode) father.getChildAfter(temp);
104.                    String type1=SematicDFS(temp);
105.                    ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
106.                    String type2=SematicDFS(tmp);

```



```

107.             int index=ASTNodeIndex.get(temp);
108.             if(!PrototypeContain("int",type1,env[index])
|| !PrototypeContain("int",type2,env[index])){
109.                 System.err.println
110.                 ("Sematic Error on line "+Row[index]+": The two sides of operator do
not balance!");
111.             }//双目操作符两边参数必须兼容
112.             return "int";
113.         }else if(father.toString().equals("FUNCTIONCALL")){//函数调用检查
114.             String FunctionFormat=SematicDFS(temp);
115.             if(FunctionFormat.equals("UnKnow")) return "UnKnow";//函数类型无法解析
116.             int index=ASTNodeIndex.get(temp);
117.             ASTNodeIndex.put(father,ASTNodeIndex.get(temp));
118.             String ReturnType;
119.             if(FunctionFormat.indexOf(":")==-1) ReturnType="UnKnow";//函数无返回类型
120.             else{//在函数有返回类型的情况下
121.                 ReturnType=FunctionFormat.substring(FunctionFormat.indexOf(":")+2);//获取
函数返回类型
122.                 FunctionFormat=FunctionFormat.substring(0,FunctionFormat.indexOf(":")+1);//获取函数形参表
123.             }
124.             int k=0, kk;
125.             String vartype;
126.             while((k=FunctionFormat.indexOf(":",k))!=-1){//传入参数与函数变量表逐次比较
127.                 for(kk=k+1;(FunctionFormat.charAt(kk)!=' ')&&(FunctionFormat.charAt(kk)!='\n')){
128.                     vartype=FunctionFormat.substring(k+1,kk);
129.                     temp=(DefaultMutableTreeNode)father.getChildAfter(temp);//获取函数形参表中下
一项
130.                     k=kk;
131.                     if(temp==null){//若函数形参表项数多余实际传入参数数量, 报告参数不足
132.                         System.err.println
133.                         ("Sematic Error on line "+Row[index]+": the variants are not
enough!");
134.                         break;
135.                     }
136.                     if(!PrototypeContain(vartype,SematicDFS(temp),env[index])){//若参数类型无法
兼容, 报错
137.                         System.err.println
138.                         ("Sematic Error on line "+Row[index]+": \""+
139.                         temp.toString()+"\" is not compatible with \""+vartype+"\" !");
140.                     }
141.                 }
142.                 temp=(DefaultMutableTreeNode)father.getChildAfter(temp);

```

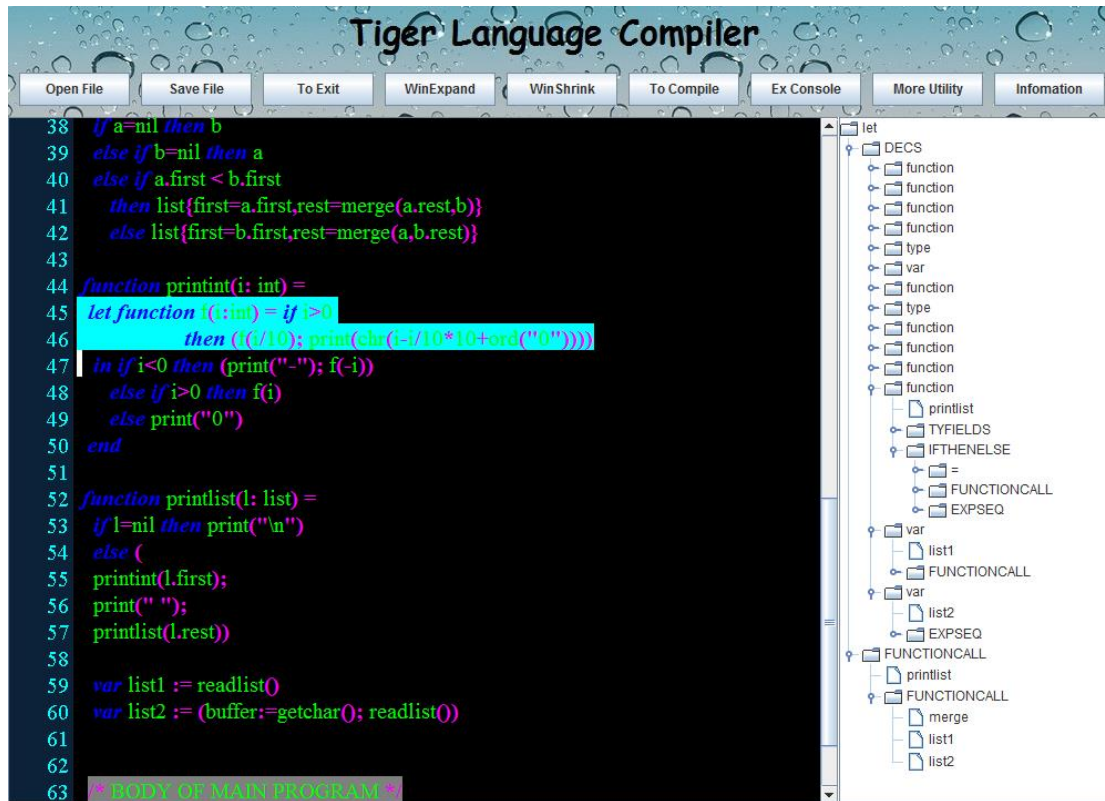
```

143.         if(temp!=null){ //若实际传入参数数量大于函数形参表项数，报告参数过多
144.             System.err.println
145.                 ("Sematic Error on line "+Row[index]+" : the variants are too more");
146.         }
147.         return ReturnType;
148.     }else{//其他语句情况下，只需要逐步分析其子节点是否合法即可，类型返回为空
149.         DefaultMutableTreeNode tmp=temp;
150.         while(temp!=null){
151.             SematicDFS(temp);
152.             temp=(DefaultMutableTreeNode)father.getChildAfter(temp);
153.         }
154.         ASTNodeIndex.put(father,ASTNodeIndex.get(tmp));
155.         return "nil";
156.     }
157. }catch(Exception ex){
158.     //ex.printStackTrace();
159.     if(!father.isLeaf()){
160.         return "UnKnow";//非叶子节点出现类型解析错误，则返回无法解析的类型UnKnow
161.     }
162.     String str=father.toString();
163.     if(Reserve.contains(str)) return "Default";//若叶子节点为保留字，返回保留字token类型
Default
164.     if(str.indexOf('\')==0){ //判断字符类型
165.         if(str.length()==3) return "char";
166.         else return "string";
167.     }
168.     if((str.charAt(0)-'0'>=0)&&((str.charAt(0)-'0')<=9)) return "int";//判断整型
169.     if((str.equals("nil")))) return "nil";
170.     int index=ASTNodeIndex.get(father);
171.     if(index<all){
172.         if(index<all-1){
173.             if(Content[index+1].equals(":")) return "nil";
174.             if(Content[index+1].equals("=")) return "nil";
175.         }
176.         //if(index>0) if(Content[index-1].equals('.') return "nil";
177.         if(env[index].containsKey(str)){//如果当前的变量作用域中包含了节点的同名token，则返回其类型
178.             str=(String)env[index].get(str);
179.             if(str.indexOf("::")!=-1)
180.                 return str.substring(str.indexOf("::")+2);
181.             else return str;
182.         }else{//解析失败的叶子节点，返回无法解析的类型UnKnow
183.             System.err.println
184.                 ("Sematic Error on line "+Row[index]+" : \""+str+"\" can not be recognized!");
185.             return "UnKnow";
186.         }

```

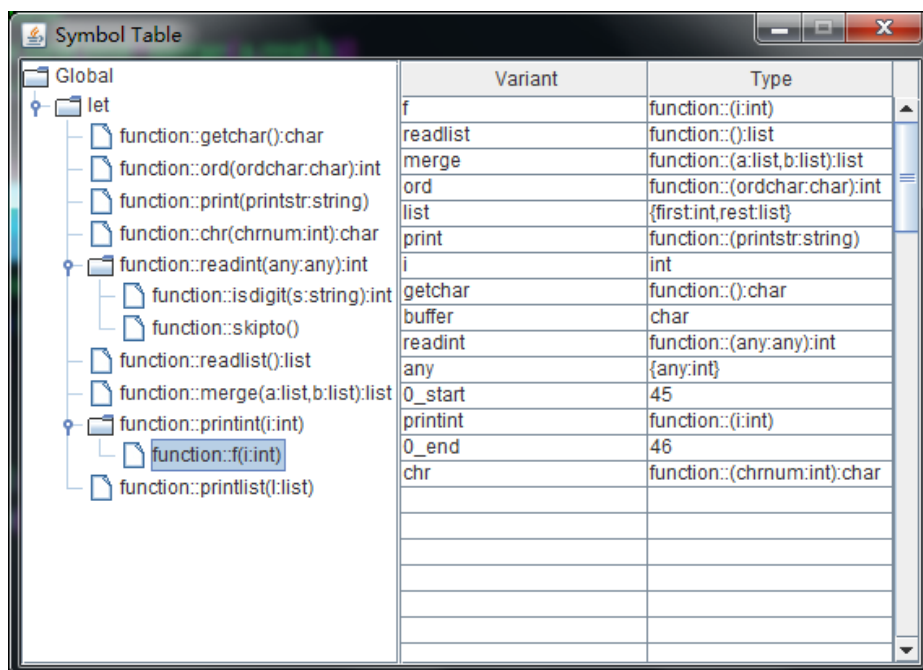
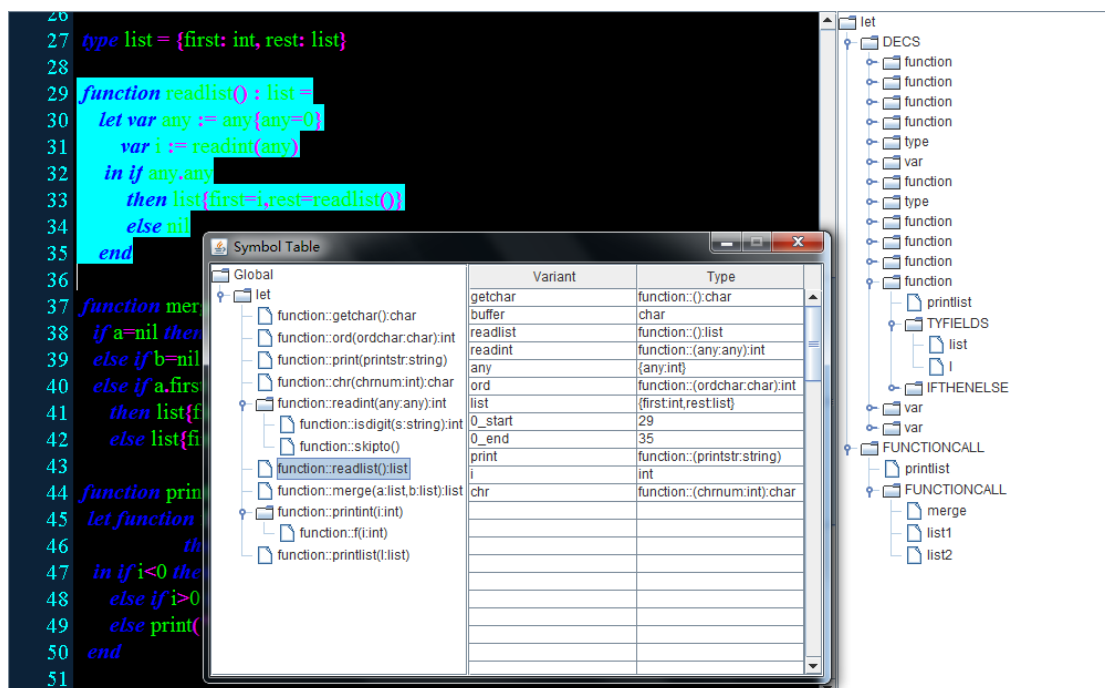
```
187.         }else{//可能出现文法错误, token无法定位, 报错
188.             System.err.println("Sematic Error :\""+str+"\" can not be located!");
189.             return "UnKnow";
190.         }
191.     }
192. }
```

抽象语法树



抽象语法树是直接由 ANTLR 模块中获得的 AST 字符串分析出来的，其给出的形式是按照树形规则嵌套的括号与 token 组成的字符串序列。我们将其重新组织成树，显示在代码的右侧，可以直接折叠展开相关节点。

符号表与变量作用域

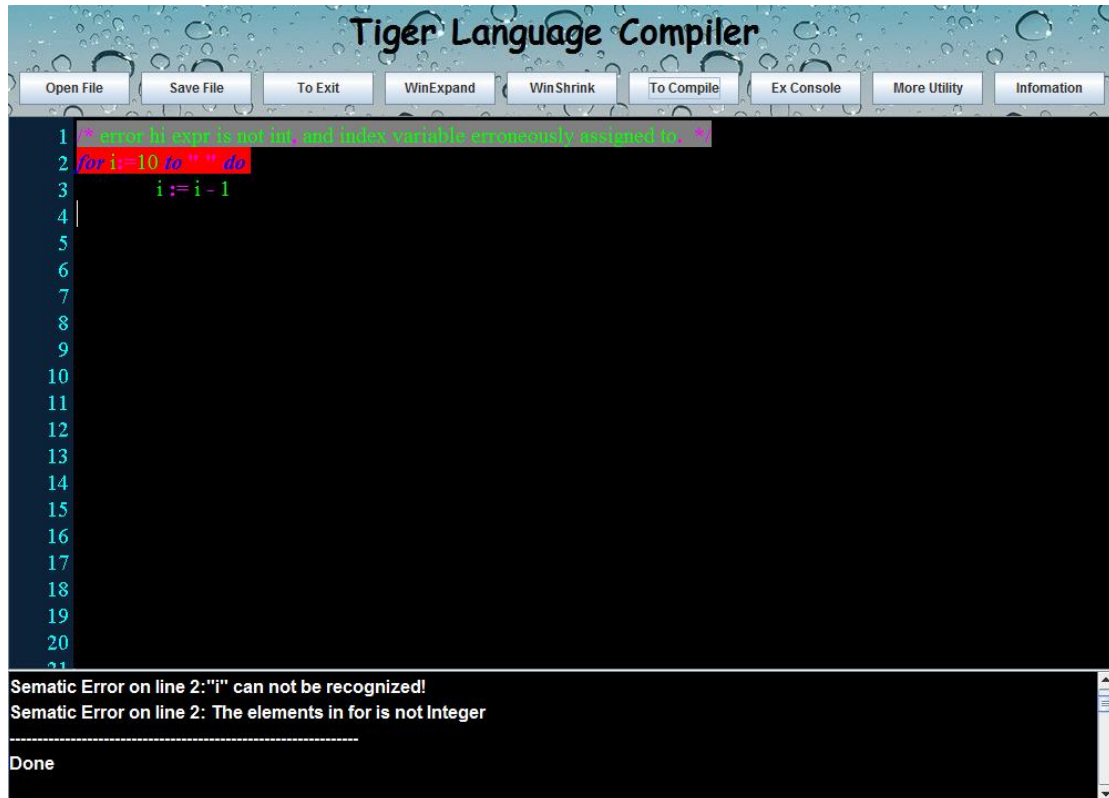


取得符号表并分析出一个变量作用域内的所有有效变量的类型是进行语义分析的最基础工作。这里我们采用栈的分析方法，将 tokens 重新入栈，用类似算数式子匹配的处理方式，分析出代码的变量作用域，并按照其隶属关系拼接成树形结构，显示在窗口左侧，每个变量作用域即对应一个节点。点击该节点，触发监听器，使得对应代码段蓝色高亮，并显示出该变量作用域中所有变量。

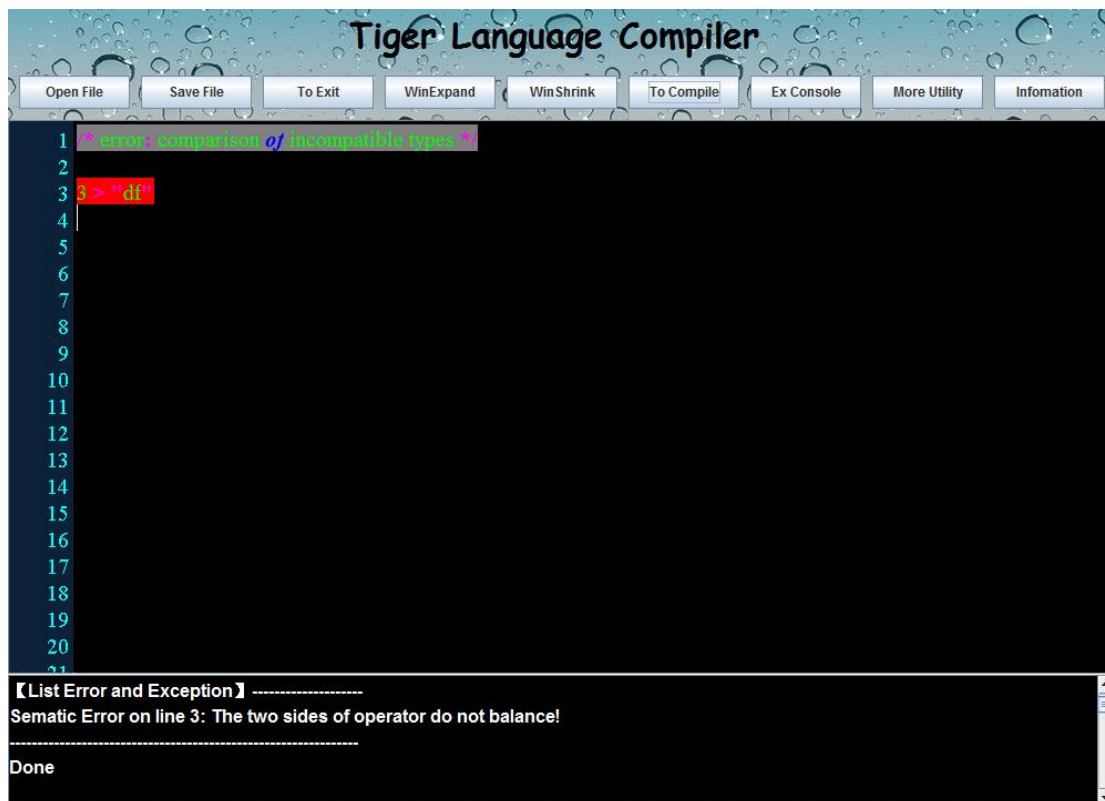
错误处理

这里，我们先给出一些测试样例的结果：

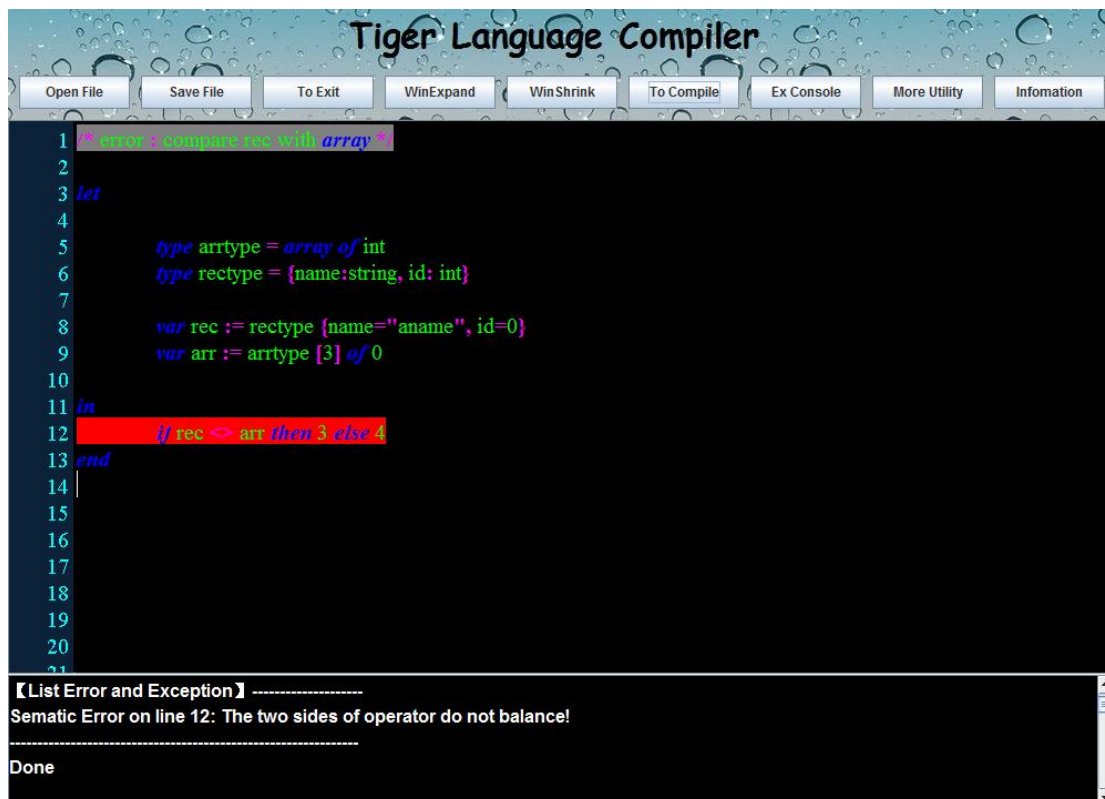
【1】【Test11.tig】【for 语句迭代范围类型错误】



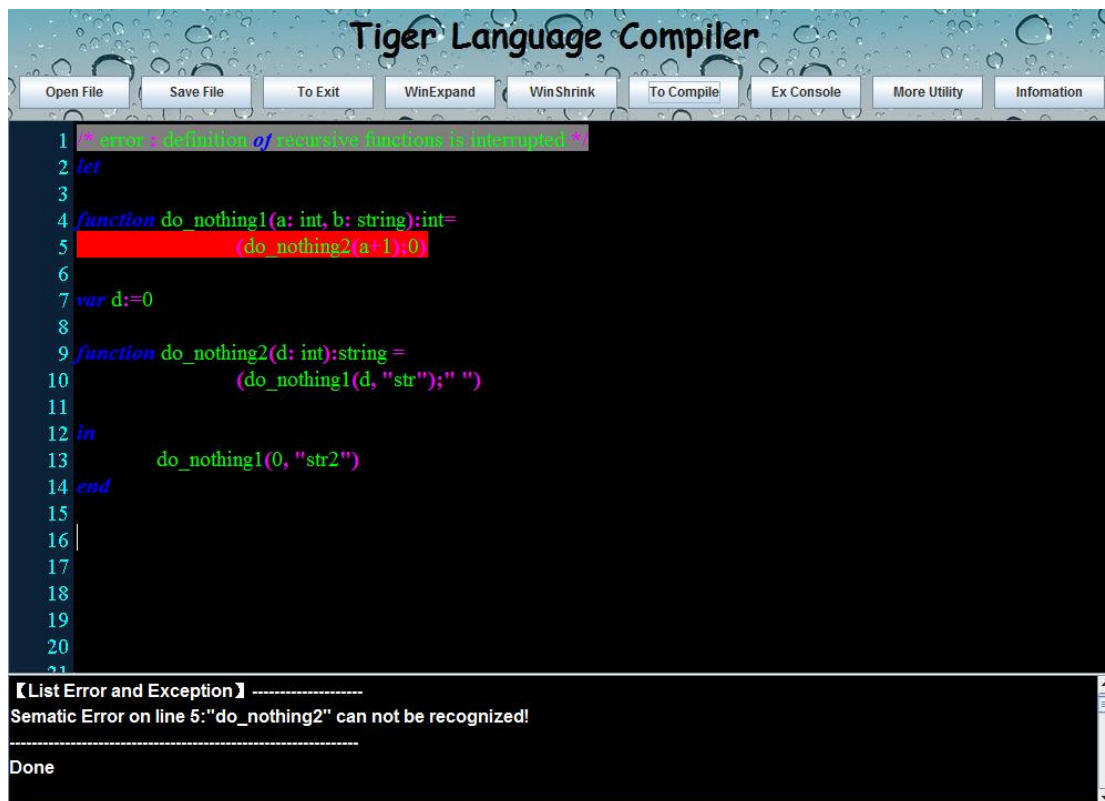
【2】【Test13. tig】【大于号左右类型不兼容】



【3】【Test14. tig】【不等号左右类型不兼容】



【4】【Test18. tig】【函数循环定义错误】



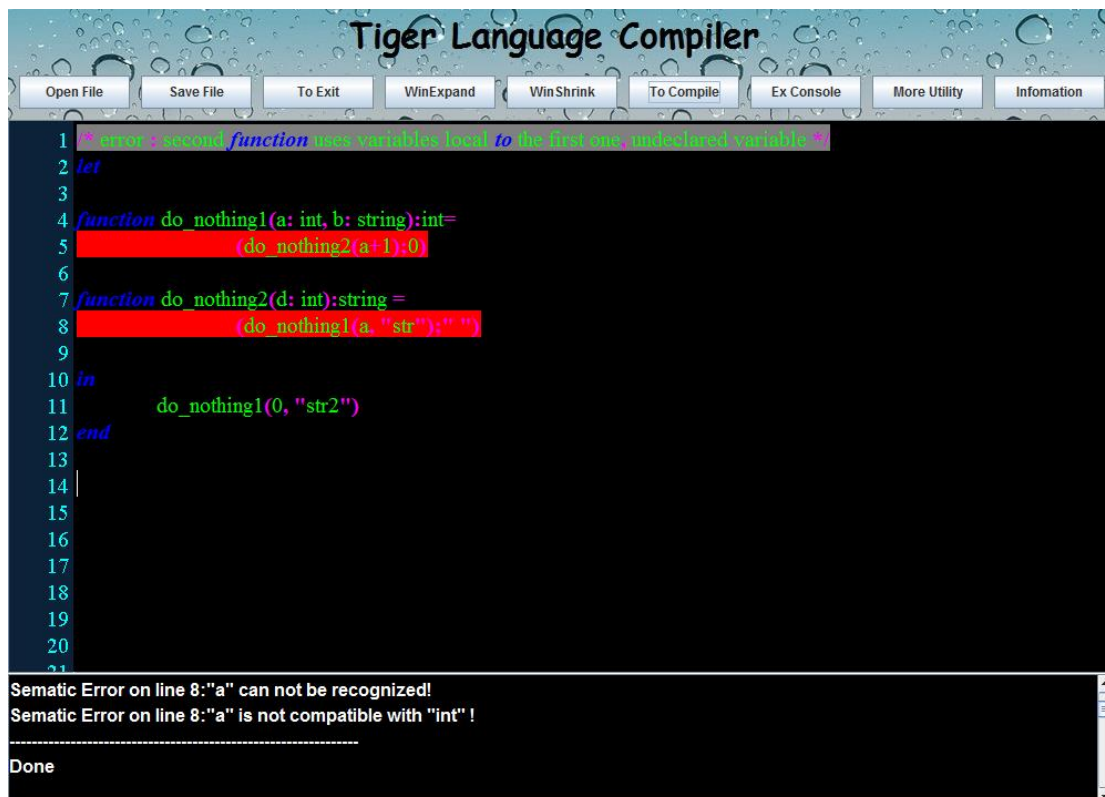
The screenshot shows the Tiger Language Compiler window. The code being compiled is as follows:

```
1  * error : definition of recursive functions is interrupted *
2  let
3
4  function do_nothing1(a: int, b: string):int=
5      (do_nothing2(a+1);0)
6
7  var d:=0
8
9  function do_nothing2(d: int):string =
10      (do_nothing1(d, "str");" ")
11
12  in
13      do_nothing1(0, "str2")
14  end
15
16
17
18
19
20
21
```

The error message displayed at the bottom is:

```
【List Error and Exception】 -----
Sematic Error on line 5:"do_nothing2" can not be recognized!
-----
Done
```

【5】【Test19. tig】【参数 a 在 do_nothing2 函数的作用域中没有声明】



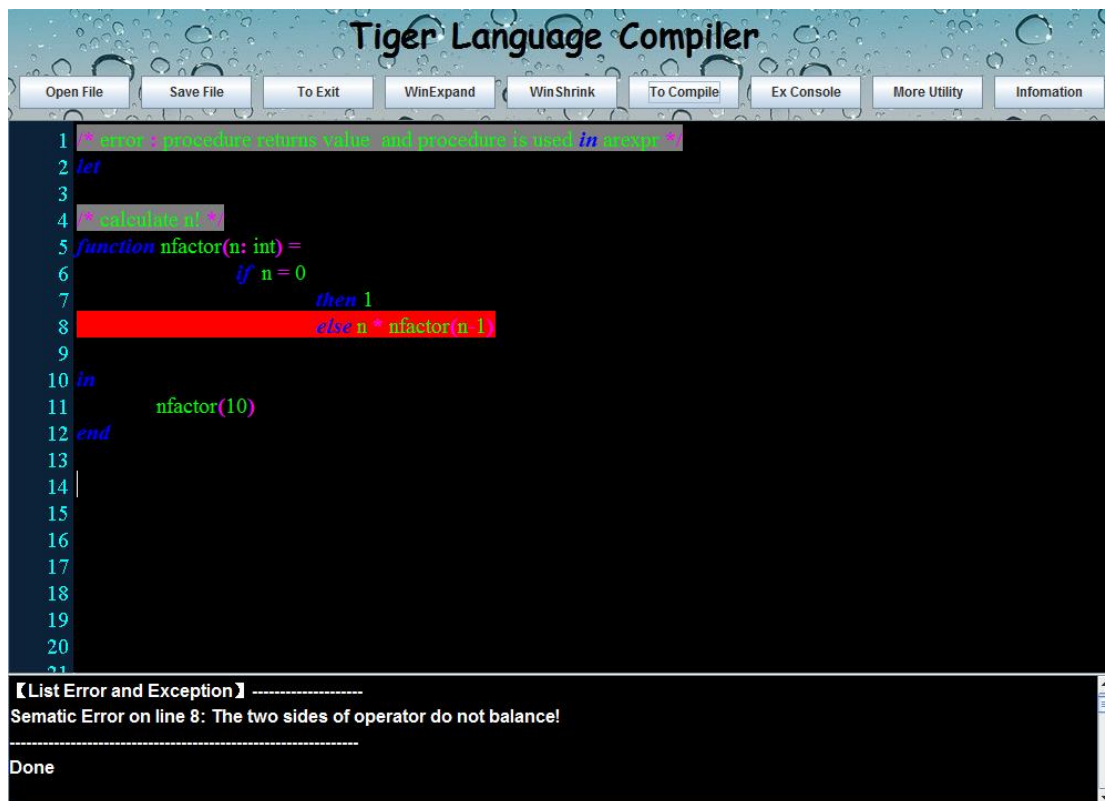
The screenshot shows the Tiger Language Compiler window. The code being compiled is as follows:

```
1  * error : second function uses variables from to the first one, undeclared variable *
2  let
3
4  function do_nothing1(a: int, b: string):int=
5      (do_nothing2(a+1);0)
6
7  function do_nothing2(d: int):string =
8      (do_nothing1(a, "str");" ")
9
10  in
11      do_nothing1(0, "str2")
12  end
13
14
15
16
17
18
19
20
21
```

The error messages displayed at the bottom are:

```
Sematic Error on line 8:"a" can not be recognized!
Sematic Error on line 8:"a" is not compatible with "int" !
-----
Done
```


【6】【Test21. tig】【函数无返回类型，不能参与数值计算】



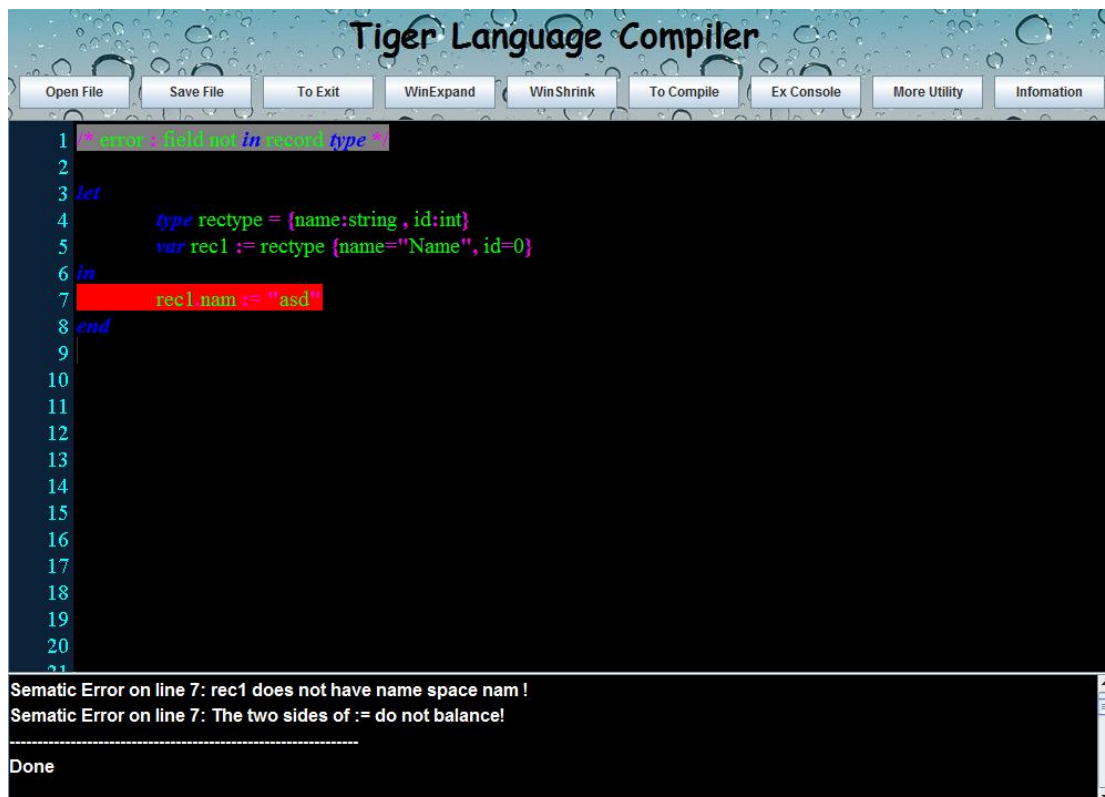
The screenshot shows the Tiger Language Compiler window. The title bar includes buttons for Open File, Save File, To Exit, WinExpand, WinShrink, To Compile, Ex Console, More Utility, and Information. The code editor displays the following code:

```
1  * error: procedure returns value and procedure is used in arxpr *  
2  let  
3  
4  * calculate n! *  
5  function nfactor(n: int) =  
6      if n = 0  
7          then 1  
8          else n * nfactor(n-1)  
9  
10 in  
11     nfactor(10)  
12 end  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

The error message at the bottom reads:

```
【List Error and Exception】-----  
Semantic Error on line 8: The two sides of operator do not balance!  
-----  
Done
```

【7】【Test22. tig】【记录类型无纪录项条目】



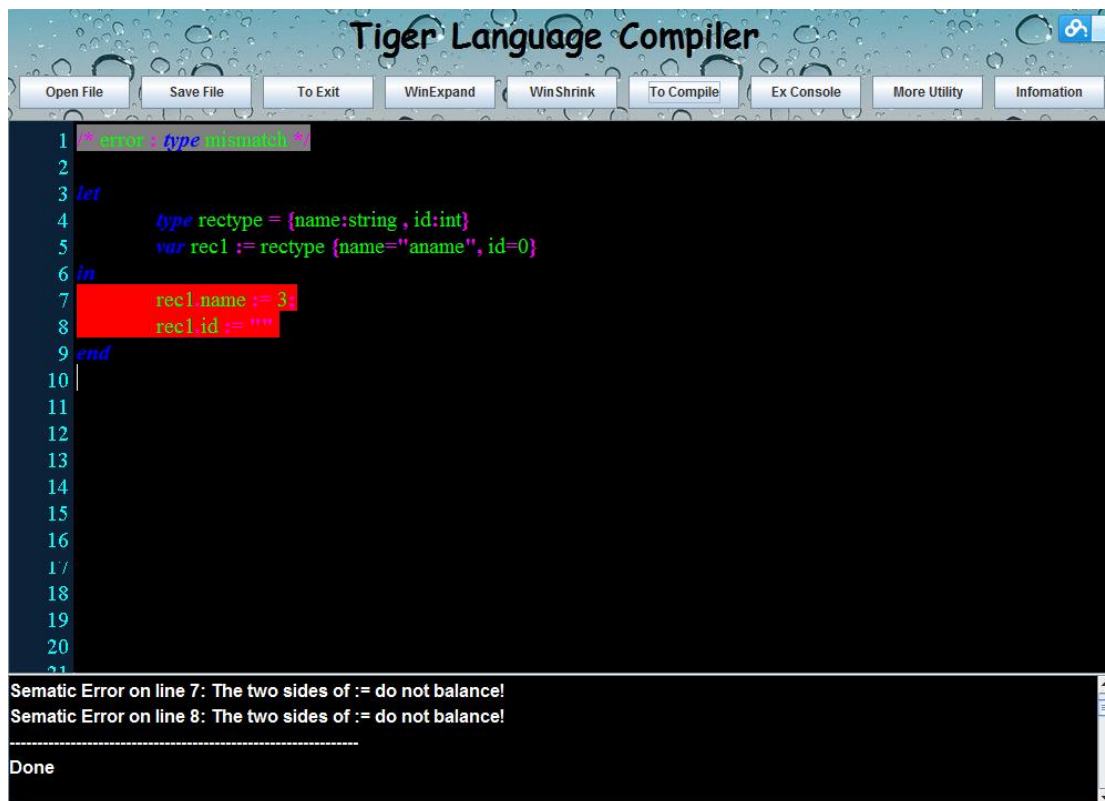
The screenshot shows the Tiger Language Compiler window. The title bar includes buttons for Open File, Save File, To Exit, WinExpand, WinShrink, To Compile, Ex Console, More Utility, and Information. The code editor displays the following code:

```
1  * error: field not in record type *  
2  
3  let  
4      type rectype = {name:string, id:int}  
5      var rec1 := rectype {name="Name", id=0}  
6  in  
7      rec1.nam := "asd"  
8  end  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21
```

The error messages at the bottom read:

```
Semantic Error on line 7: rec1 does not have name space nam !  
Semantic Error on line 7: The two sides of := do not balance!  
-----  
Done
```

【8】【Test23. tig】【记录类型与其赋值的类型不兼容】



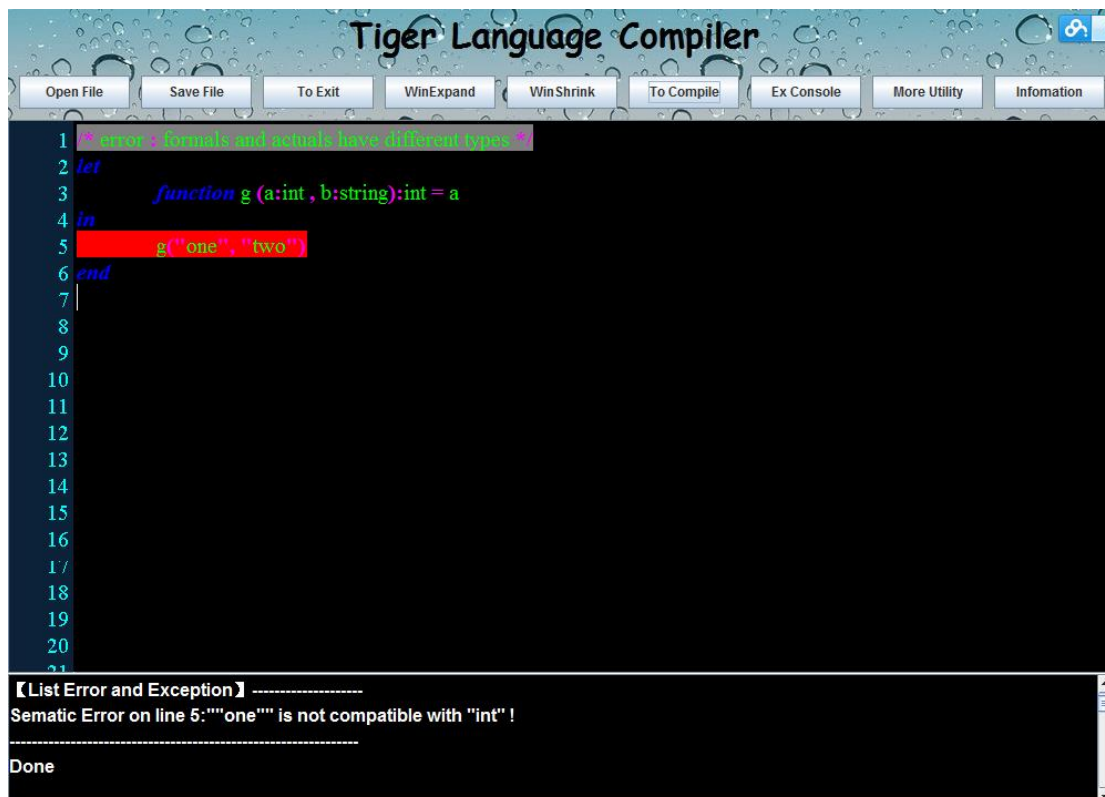
The screenshot shows the Tiger Language Compiler interface. The code in the editor is as follows:

```
1 * error: type mismatch *
2
3 let
4     type rectype = {name:string, id:int}
5     var rec1 := rectype {name="aname", id=0}
6 in
7     rec1.name := 3;
8     rec1.id := "a"
9 end
```

The compiler output at the bottom shows two semantic errors:

```
Semantic Error on line 7: The two sides of := do not balance!
Semantic Error on line 8: The two sides of := do not balance!
-----
Done
```

【9】【Test34. tig】【函数参数传入与形参类型表不兼容】



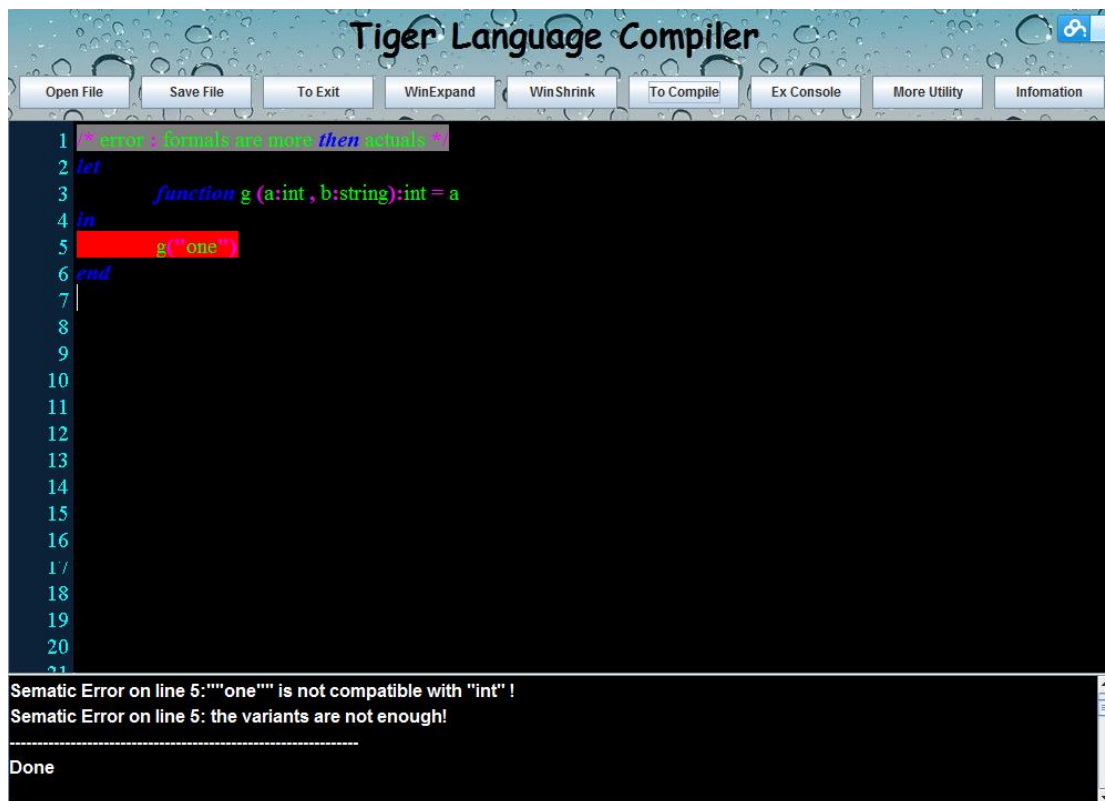
The screenshot shows the Tiger Language Compiler interface. The code in the editor is as follows:

```
1 * error: formal's and actuals have different types *
2 let
3     function g (a:int, b:string):int = a
4 in
5     g("one", "two")
6 end
```

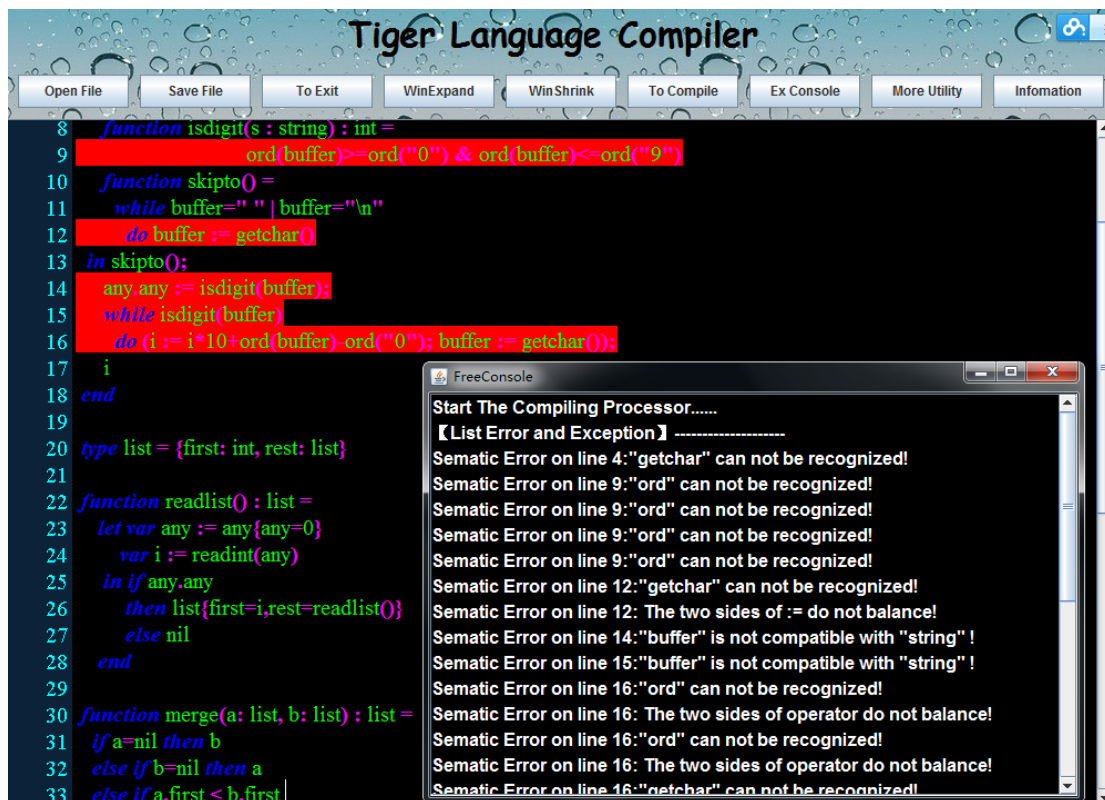
The compiler output at the bottom shows a semantic error:

```
【List Error and Exception】 -----
Semantic Error on line 5: ""one"" is not compatible with "int" !
-----
Done
```

【10】【Test32.tig】【函数传入参数少于形参表项数】



【11】【merge.tig】【很多错误，但显示了错误修复功能】



- a) IDE 能提示错误类型（词法错误、语法错误、语义错误等）、出错位置，并红色高亮相应行。
- b) 错误修复：我们不强行修改用户编辑的代码，但是我们会假设出错位置是正确的（通过 Java 的异常处理机制和 ANTLR 模块内部的逻辑处理功能），从而继续检查其他语句。这样，一次编译不会因为只遇到一个异常就终止，它会先消极地修复错误（认为出错语句实际是对的），从而尽可能多地继续检查错误。然后积极地向用户提供错误修复的建议（就是 CMD 中出错提示冒号后面的部分），从而完善用户体验。

体会与总结

ANTLR 的使用

ANTLR 的使用还是比较挑战性的，首先需要阅读大量的文档，查阅大量在线的资料，然后进行了大量讨论之后才成功配置好相应的环境运行起来。

在使用过程中我们发现 ANTLR 的内存占用非常可怕，小小的一个源代码文件在编译过程中竟然会吃掉几个 G 的内存，以至于最后不得不在寝室共用的工作站上测试和运行。后来查阅文档我们才发现原来 ANTLR 文法规则使用了 LL(K)，根据源代码来决定 K 的值，比其它的问法更消耗资源，这样造成大量的内存占用，不知道是基于什么考虑，如果用 LR 的话应该不至于这么夸张。

开发过程

累：从前期内核编写，到后期的 IDE 实现，工作量浩大。但是这毕竟是本科阶段最后一个课程实践项目了，应该认真对待。

难：ANTLR 一开始上手难，花了很久才搞清楚其机理。不过和 javacc, bison 等工具比起来，ANTLR 的语法规则简单，词法和语法的规则基本类似。同时，ANTLR 有现成的转化为抽象语法树的机制，前期的投入也可以印证“磨刀不误砍柴工”这句老话。

合作力量大：我们两位同学，日以继夜、通力合作，在艰难繁琐的设计工作中不断砥砺自己的意志，丰富自己的知识，可以说获益匪浅。在开发过程中，我们充分利用了各种工具的特性，极大地提高了开发的效率。如跨平台、服务端和通信部分使用了 python，编译器和桌面版的 IDE 使用了 Java，网页 IDE 设计使用了 bootstrap 和 d3，使得整个工程项目美观、高效而且有很好的跨平台特性。所谓人尽其才，物尽其用，我们充分利用了各自的长处，很好地完成了这次合作。

加深理解：“百鸟在林，不如一鸟在手”。虽然学习书本知识很重要，但是没有实践，我们也最多只是“笔下虽有千言，胸中实无一策”的夸夸其谈之徒。实践最大的意义就是化无形的智慧为有形的杰作，这也是我们所大为感慨的。

教训：编译器对于五十多个测试文件基本都能正确编译并给出相应信息，但是还有 4~5 个合法编译文件被检查出错误（虽然可以直接通过修改源代码来修正），这是因为在实现之前没有研究过测试样例，同时也忽略了文档中语法的一条非常琐碎的要求：这个语言的函数定义是可以在文件中的任何位置（通常在其它语言如 C 中是不行的，必须先定义再使用），而我们项目中函数作用域的搜索是用栈实现的（变量用栈实现没有什么问题，因为变量必须先定义）。这里我们犯了一个想当然的错误，最后工程做好以后再改工作量就太大了，所以这个项目只差一点点就完美，实在是个遗憾。