编译系统课程实验报告

实验 2：语法分析

<table>
<tr><td>姓名</td><td>杨富祥
王天一
白镇北</td><td>院系</td><td colspan="2">计算机科学与技术</td><td>学号</td><td colspan="2">1171800323
1170300220
1170301005</td></tr>
<tr><td colspan="2">任课教师</td><td colspan="2">陈鄞</td><td>指导教师</td><td colspan="3">文荟俨</td></tr>
<tr><td colspan="2">实验地点</td><td colspan="2">软件学院三楼</td><td>实验时间</td><td colspan="3">2020-04-19　13: 00 – 15: 30</td></tr>
<tr><td colspan="2" rowspan="2">实验课表现</td><td>出勤、表现得分</td><td></td><td rowspan="2">实验报告得分</td><td rowspan="2"></td><td rowspan="2">实验总分</td><td rowspan="2"></td></tr>
<tr><td>操作结果得分</td><td></td></tr>
<tr><td colspan="8">组内分工情况说明：

杨富祥：
文法设计、First 集求解、CLOSURE 函数、GOTO 函数、items 函数、LR(1)分析表填充，报告撰写。

王天一：
对 Token 序列分析输出语法动作、错误处理、LR(1)分析表填充、处理 Token 序列符合输入，测试用例，报告修改与补充。

白镇北：
语法分析树构建与输出、LR(1)分析表输出。

小组群内讨论的截图：</td></tr>
</table>

还挺长

2020/4/13 22:48:53

瞅一眼对不对

我看看

语音通话已取消

2020/4/13 23:10:01

通话时长 19:53

2020/4/16 22:20:06

转换完成了

2020/4/16 22:20:22

对了

OCT("oct"), HEX("hex"),这个全变成啥啊

num吧

嗯

是不是所有的num和real全是 num

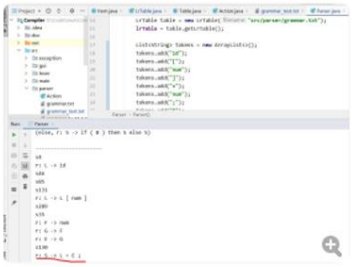
不

num是整型数，real是浮点数

那real就变成 float

你把他变成文法里面有的

2020/4/16 17:18:20



可以对这个例子做分析了，可以看出最后规约出了S

但还存在一些问题，对B->ε这类还有bug

2020/4/16 17:25:29

还差错误处理，打印语法树等

2020/4/16 17:34:10

1703010-白镇北

分析表目前能可视化吗?

不能

1703010-白镇北

那我今天做

分析表都在这里

1703010-白镇北

语法分析树我来做

```
List<String> tokens1 = new ArrayList<>();
tokens1.add("int");
tokens1.add("id");
tokens1.add("=");
tokens1.add("real");
tokens1.add(";");
```

比如这个是一个赋值语句，应该把他整成这样

2020/4/16 19:54:41

刚刚我新push到github了，这个测试用例可以正常acc了

注意更新一下代码再写

2020/4/17 8:32:07

@全体成员 下午6点汇总一下代码，看一下进度，今天大致要成型，明天修改一些bug，写报告，后天就要验收了

2020/4/17 17:46:16

@1703010-白镇北 n你写的怎么样了

2020/4/17 17:49:27

1703010-白镇北

语法树已经构建好了

1703010-白镇北

加油

反正到时候得你来整了

这块

0:29:53

所以写的是

短语层次错误恢复

- 检查LR分析表中的每一个报错条目，并根据语言的使用方法来决定程序员所犯的何种错误最有可能引起这个语法错误
- 然后构造出适当的恢复过程

这个

错误处理。。

现在只能识别出一部分错误

有好多错误都是不支持的

我明天再看看吧

后天演示的话你来搞，正确的测试、错误的测试

还有那个正确的测试用例有了没

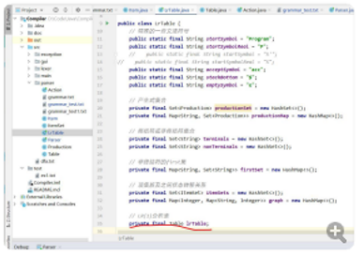
能acc吗

1703010-白镇北

ok

是这样吗

表头和表的内容



1703010-白镇北

把这棵树横过来

1703010-白镇北

就是普通的语法树吧

1703010-白镇北

我试一下行不行

这样也行吧

2020/4/17 19:11:31

这里有，它在构造函数中会构造完毕，你可以直接调用getLrTable()获得这个数据结构

1703010-白镇北

旋转好像不太行

2020/4/17 22:29:50

这样就是不太好看，如果语法树很大的话

1703010-白镇北

代码我已经push上去了

王天一

已经能识别一种错误了，c = ；

1703010-白镇北

还有一个问题，语法树是默认展开还是默认闭合用手点开好一些？

王天一

就是缺少等号右边的值的时候，能了

王天一

别的我还在调试

加油，争取明天晚上之前搞定

都行把

一、需求分析

得分

要求：采用至少一种句法分析技术（LL(1)、SLR(1)、LR(1)或 LALR(1)）对类高级语言中的基本语句进行句法分析。阐述句法分析系统所要完成的功能。

1. 能够识别声明语句、表达式及赋值语句、分支语句、循环语句和过程调用语句。
2. 可以自动计算 CLOSURE(I)和 GOTO 函数的程序，并自动生成 LR 分析表。
3. 具备语法错误处理的能力，能够准确给出错误所在位置，并采用可行的错误恢复策略。
4. 将构造好的语法分析树按照先序遍历的方式打印每一个结点的信息。

二、文法设计

得分

要求：给出如下语言成分的文法描述。

- 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

```
1. Program -> P  /*程序入口*/
2. P -> D P | S P | ε
3. D -> T id A ; | struct id { P } | proc X id ( M ) { P } /*声明语句*/
4. T -> X C
5. X -> int | float | bool | char       /*变量声明的类型*/
6. C -> [ num ] C | ε                   /*声明数组类型*/
7. A -> = F A | , id A | ε              /*声明时赋值，连续声明*/
8. M -> M , X id | X id                 /*参数类型声明*/
9. S -> L = E ; | if ( B ) then S else S | do S while ( B ) ;
   | call id ( Elist ) ; | return E ; /*赋值，分支，循环，过程调用和返回*/
10. L -> L [ num ] | id                 /*对变量或数组赋值*/
11. E -> E + G | G                      /*表达式*/
12. G -> G * F | F
13. F -> ( E ) | num | id | real | character
14. B -> B || H | H                    /*逻辑表达式*/
15. H -> H && I | I
16. I -> ! I | ( B ) | E relop E | true | false
17. relop -> < | > | <= | >= | != | ==
18. Elist -> Elist , E | E
```

三、系统设计	得分	

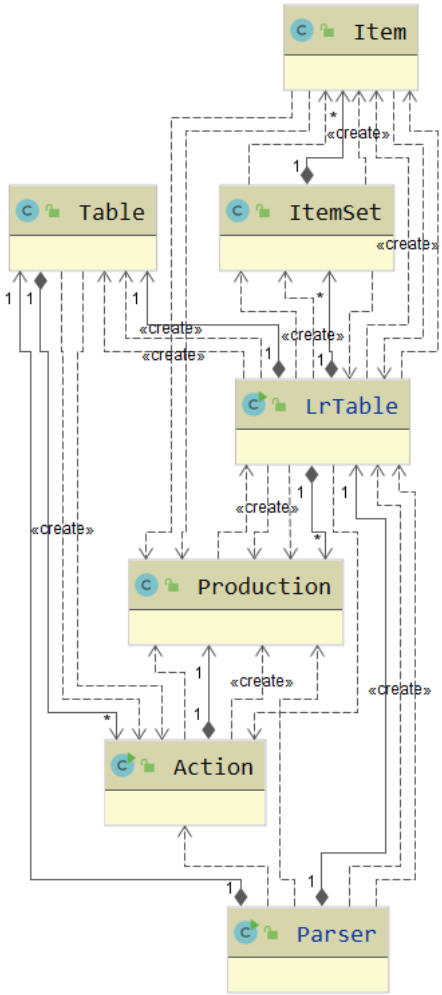
要求：分为系统概要设计和系统详细设计。

（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

（1）语法分析器的 UML 图如下所示



Action 代表移进、规约、跳转、接收和错误处理动作。

Table 为 LR(1)分析表，包含表头、动作等信息。

Item 代表 LR 分析中的项目，包括产生式、状态、展望符等。

ItemSet 为 LR 分析中的项集。
Production 为产生式，包括产生式左部和右部。
LrTable 为核心类，它在这里根据输入的文法解析出产生式，并自动计算 First 集、闭包，构建并使用 GOTO 函数自动求解项集族及其间转移关系，从而得以填充 LR(1)分析表，完成 Table 的构建。
Parser 会根据 LrTable 中构建完成的 LR(1)分析表分析 Token 序列，构建语法分析树，处理错误等。

（2）

• 核心数据结构

最核心数据结构是 Table 这个类，代表 LR(1)分析表，内部使用 List 存储表头信息，包括终结符和非终结符（去除增广文法开始符号和 ε），使用二维数组存放语法分析的各类动作（移入、规约、跳转、接收等）。

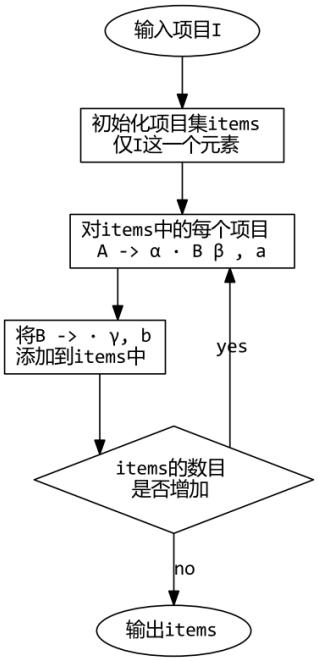
LrTable 类内部构造产生式集合 Map、终结符和非终结符集合 Set、非终结符的 First 集 Map、项集族 Set 及之间状态转移关系 Map 等来辅助 Table 的构建。

• 主要函数说明：

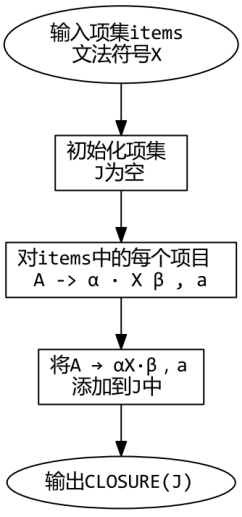
在 LrTable 类中：
getFirstSet()函数，求每一个非终结符的 First 集。
getFirstFromList()函数，在求项集闭包的时候需要计算 First(βa)，因此设置此函数。
getClosure 函数，求解 LR(1)项目集闭包。
gotoFunction()函数，计算项集之间转移关系，根据当前状态和输入符号获得下一个项集。
items()函数，为文法构造 LR(1)项集族。
constructLrTable()函数，填充 LR(1)分析表。
在 Parser 类中：
handle()函数，对 Token 序列分析，构建语法分析树。

• **流程图**

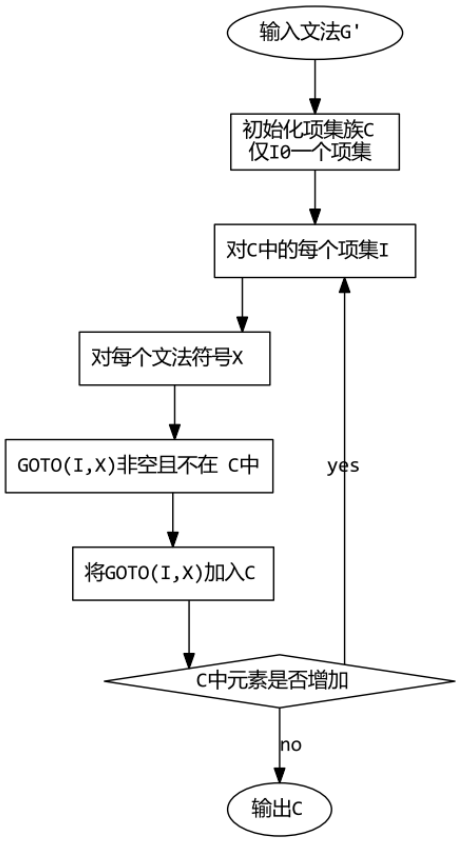
由于处理流程比较复杂，仅给出核心函数的流程图。



图：CLOSURE 函数



图：GOTO 函数



items 函数

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） 输出该句法分析器的分析表；

（3） 针对一测试程序输出其句法分析结果；

（4） 输出针对此测试程序对应的语法错误报告；

（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。

（1）系统实现中问题

①、处理流程极为复杂，项集数目非常之多，难以调试验证中间函数是否正确，故利用老师课件中小型文法来进行调试，成功之后再使用我们构建的文法和测试用例进行验证。

②、如何设计适当的数据结构来表示产生式、项目、项集、分析表等概念，比较困难。

③、词法分析中得到的结果 Token 序列，里面存储的全是种别码的形式，在我们开始语法分析之前，需要将 Token 序列进行解析。

比如 ASSIGN 是等号的种别码，那么我们在语法分析规约中需要的是“=”；再例如注释的种别码是 NOTE，在语法分析中，不需要注释的存在，所以就应该将其在 token 中删除。

④、错误处理中遇到的主要问题在错误处理、错误恢复，如果仅仅使用恐慌模式不断弹栈或读入，那么会导致在遇到下一个可归约的字符之前，可能会出现中间太多的字符被跳过，进而由于部分代码缺失，导致后面原本正确的代码段被覆盖或再次出错的情况。

因此，最终采用的是类似恐慌模式、和短语层次错误恢复 相结合的错误处理。为了提高程序容错性与效率，先对一些可能出现的常见错误进行了判断与恢复，例如：在 if 的后面应该紧跟着左括号“{”，但如果 token 中 if 后面直接就是一个 id 的话，说明缺少“{”，所以，在 token 中相应位置添加一个左括号，即可恢复，并且不会影响整个 token 序列的规约，与语法分析树的正确生成。

如果不满足常见的几种语法错误，就采用恐慌模式弹栈，寻找 GOTO 表，构造 follow 集判断是否可跟在后面等等步骤来处理错误。

（2）LR(1)分析表

LR(1)分析表

States	struct	||	&&	<=	bool	num	do	float	while	character	else	id
I0	s1				s7		s2	s3				s4
I1												s25
I2							s18					s4
I3												r: X -> float
I4												
I5												
I6	s1				s7		s2	s3				s4
I7												r: X -> bool
I8												
I9												
I10	s1				s7		s2	s3				s4
I11												s32
I12												r: C -> ε
I13					s27			s26				
I14												r: X -> int
I15												s31
I16												r: X -> char
I17						s35				s36		s37
I18							s18					s4
I19												
I20												
I21									s47			
I22												s59
I23						s35				s36		s37
I24												
I25												
I26												r: X -> float
I27												r: X -> bool
I28												s86
I29												r: X -> int
I30												r: X -> char
I31												
I32												
I33												r: T -> X C
I34						s53						
I35												
I36												
I37												
I38						s75				s76		s77
I39												
I40												
I41												
I42												
I43												
I44						s61				s62		s63

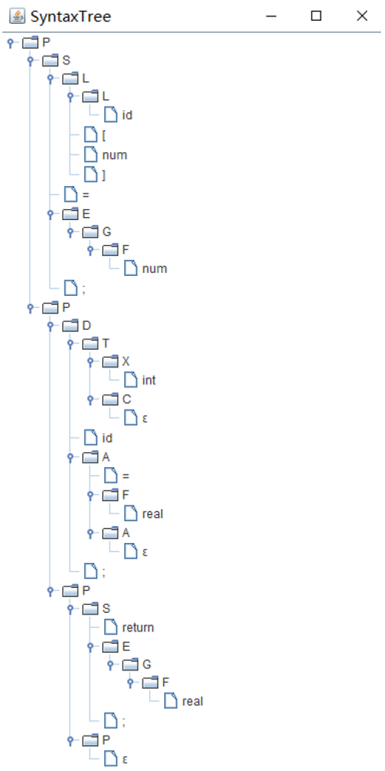
LR(1)分析表

States	struct	||	&&	<=	bool	num	do	float	while	character	else	id
I268						s35				s36		s37
I269												
I270												
I271											r: S -> retur...	
I272												
I273						s134				s135		s136
I274												
I275									r: S -> do S ...			
I276												
I277		s144										
I278												
I279		s144										
I280	r: D -> struc...				r: D -> struc...		r: D -> struc...	r: D -> struc...				r: D -> struc.
I281							s18					s4
I282												
I283												
I284											r: S -> L = E ;	
I285									r: S -> if (B ...			
I286	r: S -> call i...				r: S -> call i...		r: S -> call i...	r: S -> call i...				r: S -> call i.
I287											s295	
I288						s61				s62		s63
I289						s35				s36		s37
I290												
I291												
I292	r: D -> proc ...				r: D -> proc ...		r: D -> proc ...	r: D -> proc ...				r: D -> proc.
I293		s144										
I294	s106				s7		s107	s3				s4
I295							s107					s4
I296							s227					s4
I297												
I298	r: S -> do S ...				r: S -> do S ...		r: S -> do S ...	r: S -> do S ...				r: S -> do S.
I299												
I300											r: S -> call i...	
I301												
I302												
I303												
I304	r: S -> if (B ...				r: S -> if (B ...		r: S -> if (B ...	r: S -> if (B ...				r: S -> if (B.
I305												s194
I306						s35				s36		s37
I307											s309	
I308	r: D -> proc ...				r: D -> proc ...		r: D -> proc ...	r: D -> proc ...				r: D -> proc.
I309							s227					s4
I310											r: S -> do S ...	
I311						s35				s36		s37
I312											r: S -> if (B ...	

（3）测试用例及其语法分析树

①、首先给出一个小型程序（仅 3 条语句）及其语法树：

```
List<String> tokens = new ArrayList<>();
tokens.add("id");
tokens.add("[");
tokens.add("num");
tokens.add("]");
tokens.add("=");
tokens.add("num");
tokens.add(";");
tokens.add("int");
tokens.add("id");
tokens.add("=");
tokens.add("real");
tokens.add(";");
tokens.add("return");
tokens.add("real");
tokens.add(";");
tokens.add("$");
```



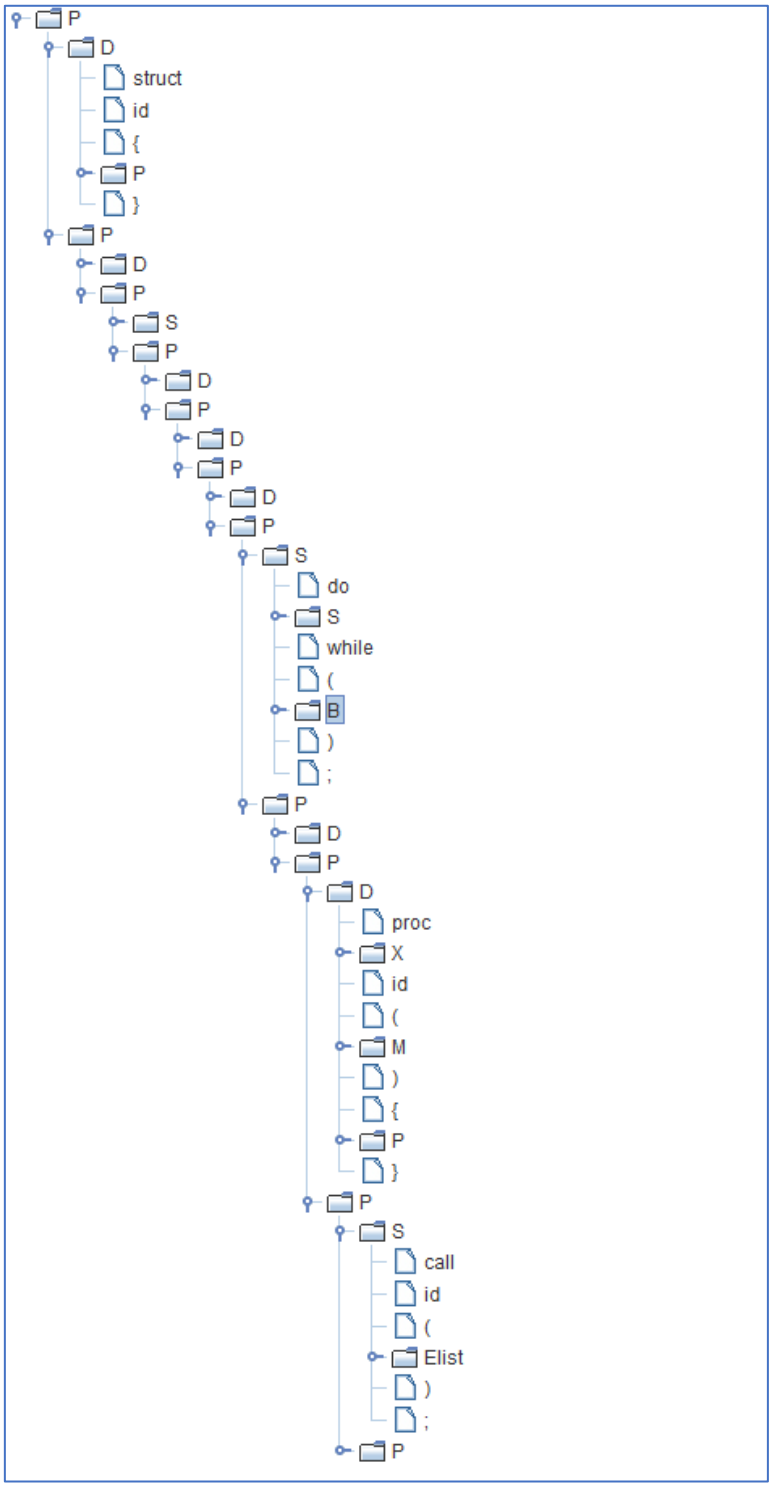
②、完整的测试用例：

```
struct student{
    int id ;
    char[10] name ;
}
    int a = 10;
    a = x + y;
    float f = 1e2;
    bool flag = 1 ;
    int d = 0;

    do
        if (a == c) then
                d = 1;
            else
                d = 2;
    while (a < 20);

    float f = 10.0;
    /* 声明 */
    proc int function(int x, int y){
         y = x + 1;
         return y;
    }
    /* 调用 */
    call function(a, d);
```

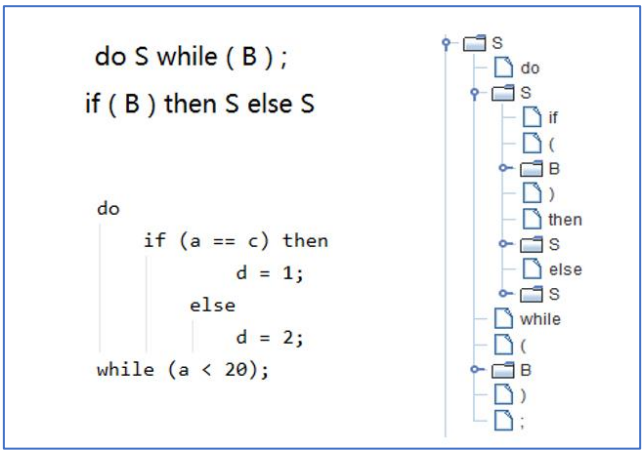
```
struct student
    int id ;
    char[10] name ;
}
    int int  a = 10 ;
    a = x + y;
    float f = 1e2;
    bool flag = ;
    int d = 0 ;

    do
        if  (a == c then
                d = 1;
            else
                d = 2
    while (a < 20) ;

    float f = 10.0;
    /* 声明 */
    proc int function(int x, int y){
         y = x + 1;
         return y;
    }
    /* 调用 */
    call function(a, d);
```

图：正确示例与错误示例

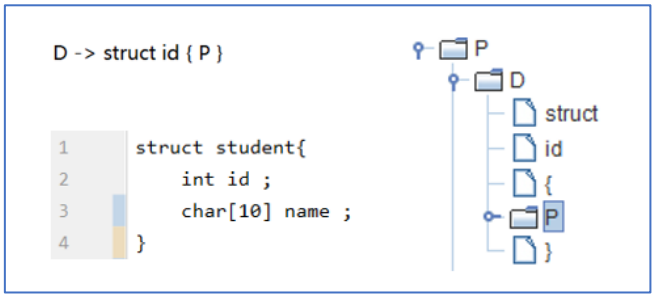
③、语法树：



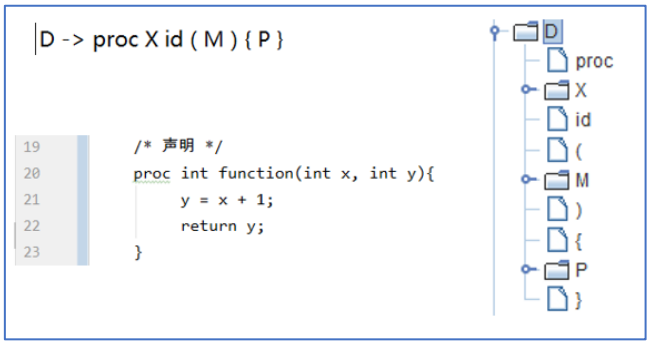
图：正确示例完整语法树（篇幅原因，略收起）



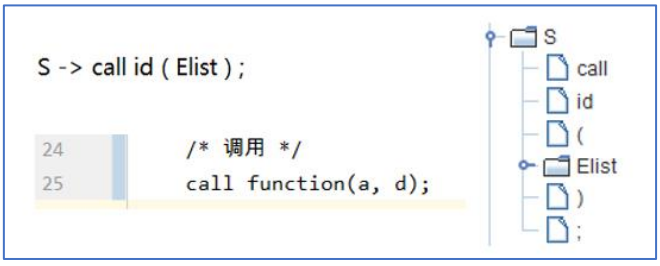
图：在循环中嵌套 if-else 的文法、用例、及语法树



图：结构体定义的文法、用例、及语法树



图：函数声明 文法、用例、及语法树



图：函数调用 文法、用例、及语法树

（4）错误报告

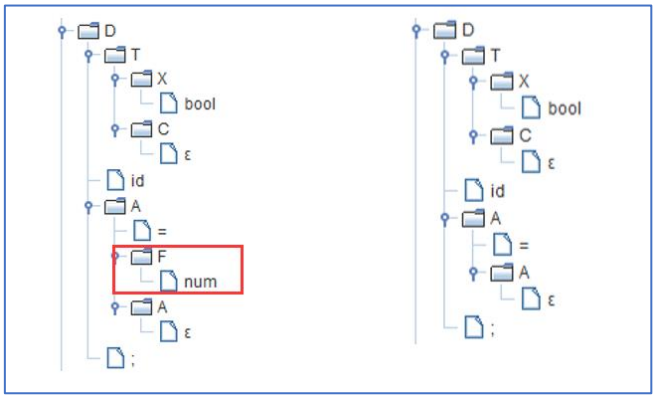
```
acc
Error at Line [1~2]:    int
Error at Line [5]:  'int'
Error at Line [8]:  ';'
Error at Line [12]:  'then'
Error at Line [15~16]:    while
```

图：错误报告

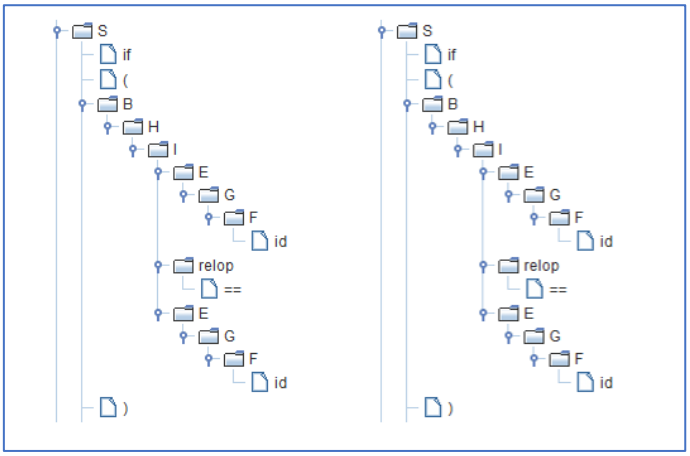
上图为错误报告，第一条对应的是 Struct 结构体缺少左括号的情况，错误提示信息表示在某一行读到哪一个字符时会出现错误。

第一条错误表示：在第一行结束，第二行开始之间，读到 int 会出现错误。

并且程序具备一定的自动纠错能力，并且不影响语法分析树的构建



图：bool flag =1; 和 bool flag = ; 的语法分析树



图：if(a==c) 和 if a==c) 的语法分析树,完全相同，自动纠错

（5）结果分析

LR 的自底向上分析可以在最后规约出 P，打印 acc，证明分析成功。

词法分析获得的 Token 序列要先进行处理，确保语法分析正常进行。

指导教师评语：

日期：