**深 圳 大 学 实 验 报 告**

**课程名称：­ 编译原理**

**实验项目名称： 高级语言及其文法**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 张昊迪**

**报告人： 郑雨婷 学号： 2021150122 班级： 高性能**

**实验时间： 2024年 5月 22 日 -- 2024年 5 月 24 日**

**实验报告提交时间： 2024年 5月 24 日**

**教务处制**

1. 实验内容

**任务一：运行 TINY 语言的语法分析程序 TINYParser，理解 TINY语言语法分析器的实现。**

其中，TINY 语言的词法与实验二相同，TINY 语言的文法描述如下：

program -> stmt-seq

stmt-seq -> stmt-seq;stmt | stmt

stmt -> if-stmt|repeat-stmt|assign-stmt|read-stmt | write-stmt

if-stmt -> if exp then stmt-seq end | if exp then stmt-seq else stmt-seq end

repeat-stmt -> repeat stmt-seq until exp

assign-stmt -> id:= exp

read-stmt -> read id

write-stmt -> write exp

exp -> simp-exp cop simp-exp | simp-exp

cop -> < | =

simp-exp-> simp-exp addop term |term

term -> term mulop factor | factor

factor -> (exp) |num |id

addop -> + | -

mulop -> \* | /

对于如下 TINY 示例程序：

{A sample TINY program}

read x;

if 0<x then

fact:=1;

repeat

fact:=fact\*x;

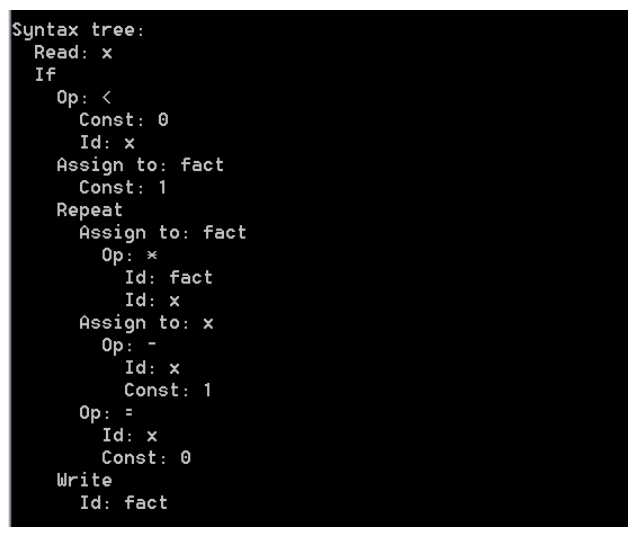
x:=x-1

until x=0;

write fact

end

TINY 语法分析器的输出语法树为：



具体的语法树结构在 TINY\_Syntax.pptx 里面描述，结合 TINYParser 代码理解语法树构造。

任务一要求：根据 TINY 语法，自己编写至少一个另外的 TINY 测试程序，运行 TINYParser 语法分析器，观察程序运行流程，得到正确的运行结果。

**任务二：基于 TinyParser 语法分析器，实现拓展语言 TINY+的语法分析器。**

其中，TINY+语言的词法与实验二相同，TINY+语言的文法描述如下（注：此处为了描述方便，对上下文无关文法的产生式表示进行了扩充，允许在产生式右部使用类似正则表达式的表示，例如第 5 条产生式右部花括号{ , identifier }代表\*闭包。其中红色部分为 TINY+文法更新的部分，其余部分为TINY 文法原有的产生式：

1 program -> declarations stmt-sequence

2 declarations -> decl ; declarations |ε

3 decl -> type-specifier varlist

4 type-specifier-> int | bool | string

5 varlist -> identifier { , identifier }

6 stmt-sequence -> statement { ; statement }

7 statement -> if-stmt | repeat-stmt | assign-stmt | read-stmt | write-stmt | while-stmt

8 while-stmt -> do stmt-sequence while bool-exp

9 if-stmt -> if exp then stmt-seq end | if exp then stmt-seq else stmt-seq end

10 repeat-stmt -> repeat stmt-sequence until exp

11 assign-stmt -> identifier:=exp

12 read-stmt -> read identifier

13 write-stmt -> write exp

14 exp -> simp-exp cop simp-exp | simp-exp

15 cop -> < | =

16 simp-exp -> simp-exp addop term |term

17 term -> term mulop factor | factor

18 factor -> (exp) |num |id

19 addop -> + | -

20 mulop -> \* | /

TINY+语言的文法主要添加了声明语句及 while 语句，对于如下 TINY+测试代码：

{this is an example}

int A,B;

bool C;

string D;

D:= 'scanner';

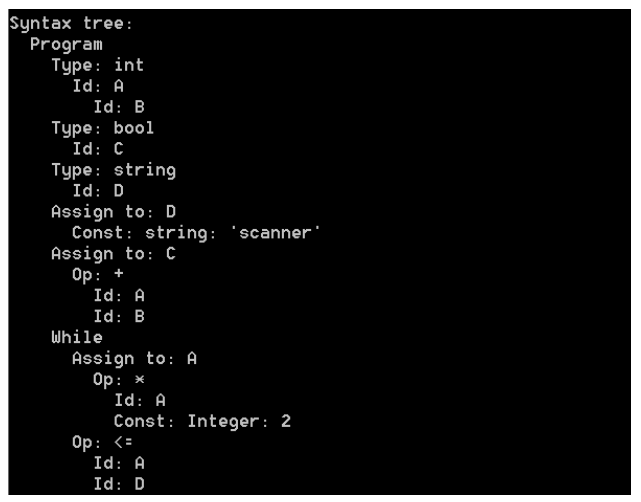
C:=A + B;

do

A:=A\*2

while A<=D

应得到以下 TINY+语法分析结果（语法树）：



任务二要求：根据 TINY+语法，修改给定的 TINY 语法分析器，实现更新的TINY+语法分析器，成功实现对上述示例程序的语法分析。并根据 TINY+文法的定义，编写至少一个另外的 TINY+测试程序，对该测试程序完成语法分析，得到正确的语法分析结果。

1. 实验要求

* 完成任务一及任务二的要求；
* 使用实验所提供的模板撰写实验报告，要求内容详实，有具体的设计描述、关键的代码片段、及实验结果屏幕截图；
* 在截止日期前将代码、实验报告、测试文件（如有）等所有实验相关文件压缩到一个压缩包姓名\_学号\_实验三.zip 上传至 Blackboard。

三、实验步骤

**任务一：运行 TINY 语言的语法分析程序 TINYParser，理解 TINY语言语法分析器的实现。**

**1.程序语法分析部分**

TINYParser是一个自顶向下的语法分析程序，更具体的，使用递归下降分析法，其主要思想是：对应每个变量设置一个子程序。当遇到终结符时，进行匹配；当遇到变量X时，调用X对应的子程序。

其中的匹配是指查看当前的token与预期的token类型是否相同，若相同，读取下一个token；若不同，输出错误信息。其中getToken是实验二实现过的读取下一个token的函数。

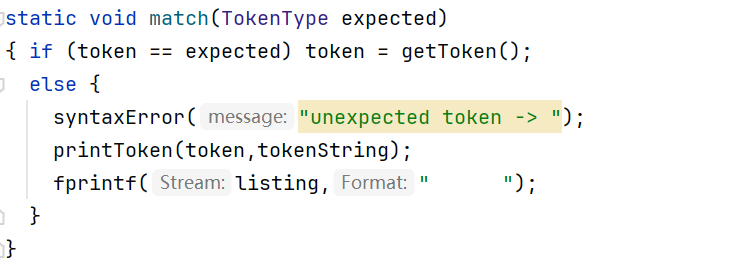


图1 match函数代码

程序从parse()函数开始，对应的是从开始字符开始的program -> stmt-seq这条产生式。产生式右部没有终结符，只有stmt-seq这一个变量，所以只用调用stmt-seq对应的子函数stmt\_sequence()。



图2 parse函数代码

接下来看函数stmt\_sequence()，程序对应的是stmt-seq -> stmt-seq;stmt | stmt这条产生式。从这条产生式中可以分析出，一个stmt\_seq可以推导出至少一个stmt，因此在函数stmt\_sequence()中，先调用statement函数一次，然后再while循环中调用statement函数，一直到遇见非法的token才结束循环,非法的条件有四个：①文件结束，遇到ENDFILE。②遇到END，代表if结构的body部分结束了。③遇到ELSE代表if结构的第一个stmt-sequence结束了。④遇到UNTIL代表repeat结构中的stmt-sequence结束了。

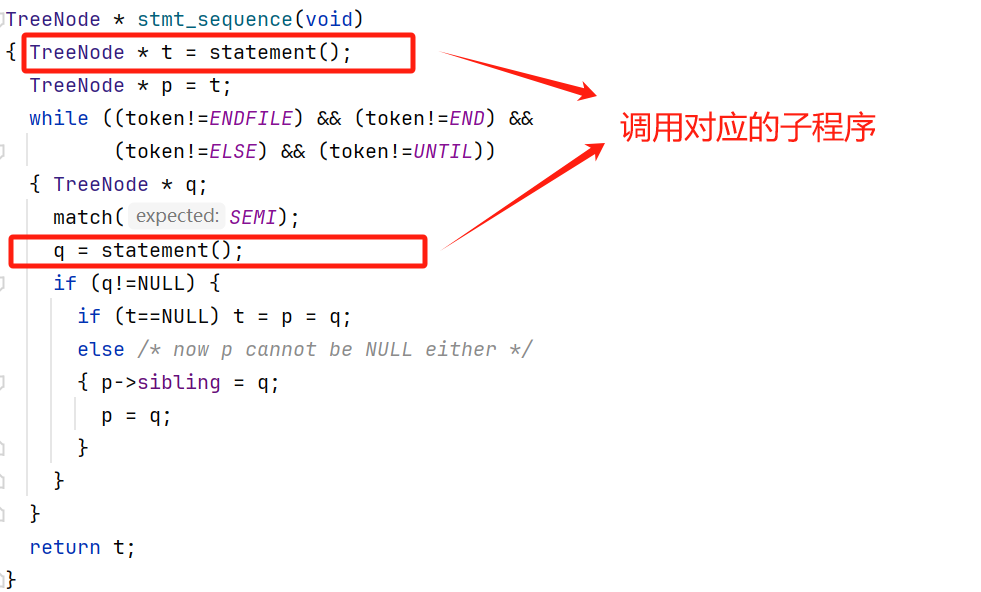


图3 stmt\_sequence函数代码

接下来看函数repeat\_stmt()，程序对应的是repeat-stmt -> repeat stmt-seq until exp这条产生式。从这条产生式的右侧既包含变量又包含终结符，因此在函数体中，先匹配REPEAT，然后调用stmt\_sequence函数，然后匹配UNTIL函数，最后再调用exp函数。

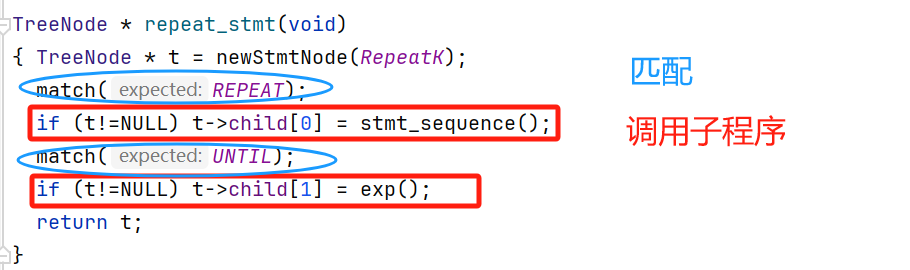


图4 repeat\_stmt函数代码

各个变量对应的子程序的实现方式几乎相同，都是按照产生式右侧的顺序进行匹配或调用函数，因此不一一列举剩余的函数。

**2.程序构建语法树部分**

整个程序在扫描输入是否合法时，也顺便构建了语法树，语法树的节点是由以下几个部分组成的：指向子节点的指针数组child、指向兄弟节点的指针sibling、节点的类型、具体的类型、节点的内容等等。

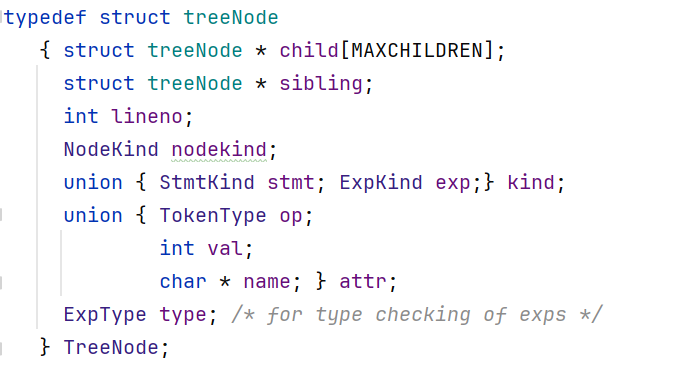
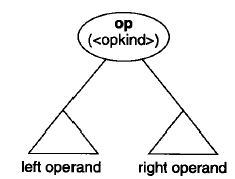
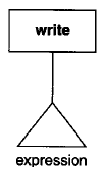
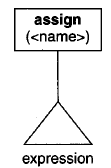
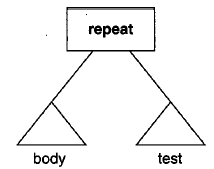
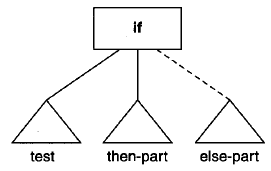


图5 TreeNode节点的定义

各个节点之间无非有两种关系，父子节点或兄弟节点。我们分析TINY语言的文法，可以得出各个statement有如下的节点关系：if-stmt可能由两个子节点或三个子节点，第一个节点可以是一个表达式节点，第二个节点可以是一个stmt-seq,若有else的部分，就有第三个子节点，第三个节点是一个stmt-seq；repeat-stmt有两个子节点，一个是重复执行的部分，是一个stmt-seq，一个是终止条件是一个表达式；assign-stmt有一个子节点，代表一个表达式；write-stmt只有一个表达式作为子节点；而有表达式左边和右部。

各个在同一个stmt-seq中的各个stmt是并列的，也就是各个stmt是兄弟节点，用sibling指针相连。



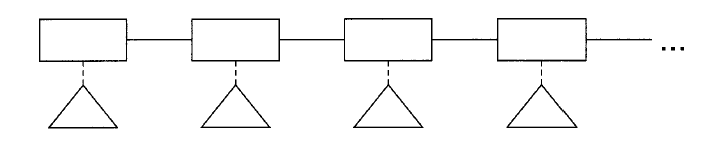


图6 语法树中节点的连接关系

例如，在stmt\_sequence中，可能有无限多个stmt，循环结束的条件是遇到ENDFILE、END、ELSE、UNTIL。遇到ENDFILE代表整个程序扫描完了，遇到END代表if-stmt中的stmt-seq扫描完成，遇到ELSE代表if-stmt中的第一个stmt-seq扫描完成，遇到UNTIL代表repeat-stmt中的stmt-seq扫描完成。同一级stmt-seq中的stmt是用节点的sibling指针相连的。还需要注意，各个stmt之间是用分号分割的，因此除了第一个statement之外，要在调用statement函数之后匹配分号。

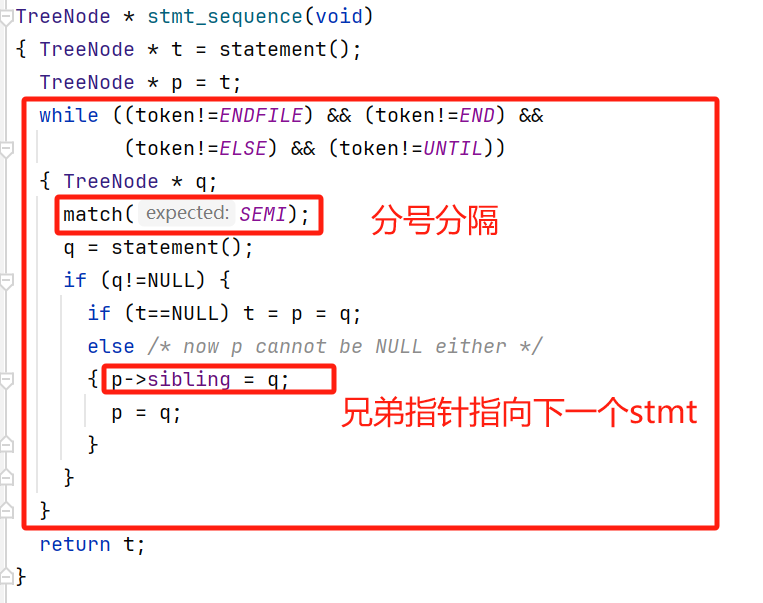


图7 stmt\_sequence函数代码

又例如，在repeat-stmt中,stmt-seq和exp分别作为本节点的两个子节点，用child[0]和child[1]分别指向这两个子节点。

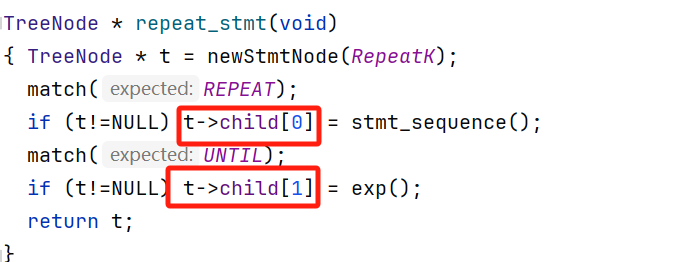


图8 repeat-stmt及其指向子节点

每个程序都按照以上的规则连接各个节点，这样，最终就能够构建出一颗完整的语法树，在输出语法树式，将语法树的根节点传入printTree函数中，即可打印出完整的语法树。具体printTree函数的实现如下，先输出当前节点，然后递归地调用printTree去打印每一个子树。当以本节点为根的树全部打印完后，再去打印其兄弟节点，当所有的兄弟节点及其子节点都打印完后，也就是当前节点为NULL时，本次printTree函数才算结束。

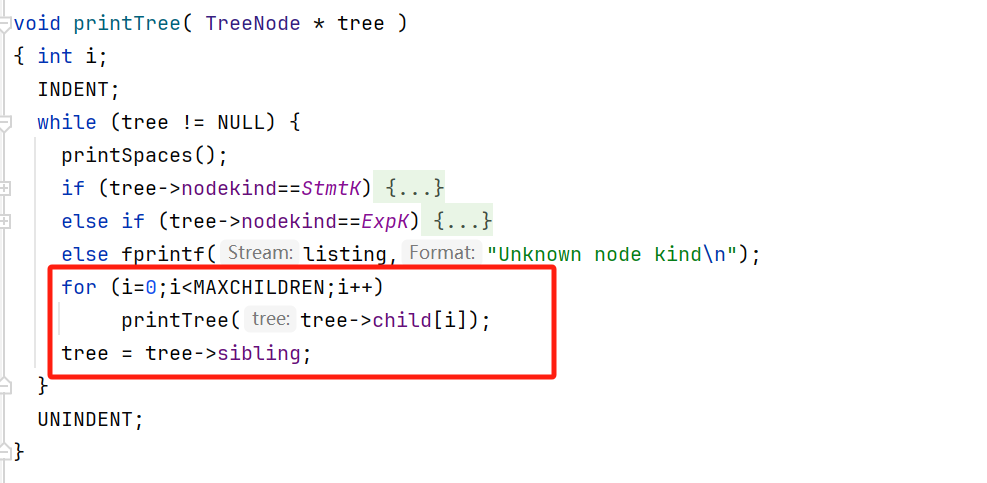


图9 printTree函数代码

每个节点在打印时会先判断是Exp节点还是Stmt节点，然后再根据具体的类型去输出需要的信息。

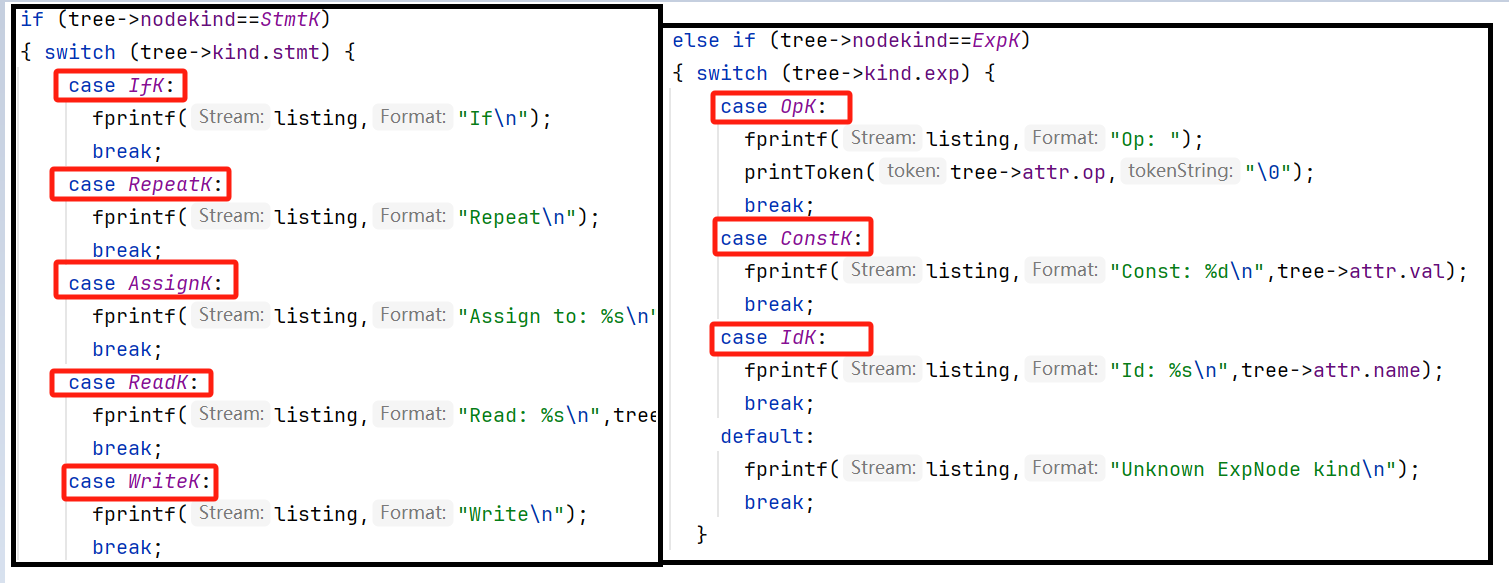
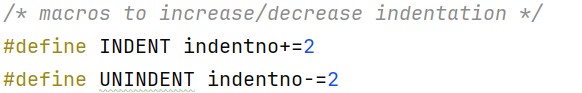


图10 根据节点类型输出不同的信息

每次调用一次printTree函数时，都会进行INDENT,INDENT是增加缩进的，这样深度越深的节点前就会有更多的空格。而当这次printTree要结束前，也会进行UNINDENT，将空格的数量减少。具体时使用一个变量indentno来记录当前应输出多少个空格，INDENT就是使变量加2，UNINDENT就是使变量减2,再打印节点的信息前先调用printSpace输出空格。



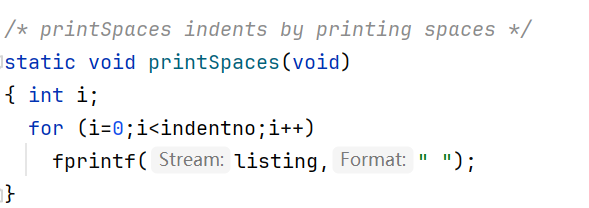


图11 缩进的实现

**任务二：基于 TinyParser 语法分析器，实现拓展语言 TINY+的语法分析器。**

1. **新增声明部分**

TINY+语言与原本的TINY语言最大的不同之处在于新增了一个声明部分。但我们还是按照自顶向下递归下降法的思路来编写程序。第一条产生式 program -> declarations stmt-sequence，可以认为语法树的根节点有两个子节点，一个是declarations,一个是stmt-sequence。因为TINY语言开始字符只推出stmt-sequence,所以省略了根节点这一步。再TINY+程序中补充根节点,在NodeKind中新增一个RootK,并且新增一个创建根节点的函数。



图12 新增根节点

declarations可能推导出零个decl或多个decl，若直接推导出零个decl也就是将NULL挂在根节点的第一个孩子上，之后输出语法树时，还是先输出以一个节点，只是为NULL什么都不输出，并不影响整个程序的正确性。declarations函数的实现与stmt- sequence的实现相似，除了declarations可能产生空，其余都相同，在代码中体现在创建的节点为NULL，如果当前的token不是INT、STRING、BOOL中的任意一个，直接返回NULL。而生成的各个decl是用兄弟指针sibling连接的。

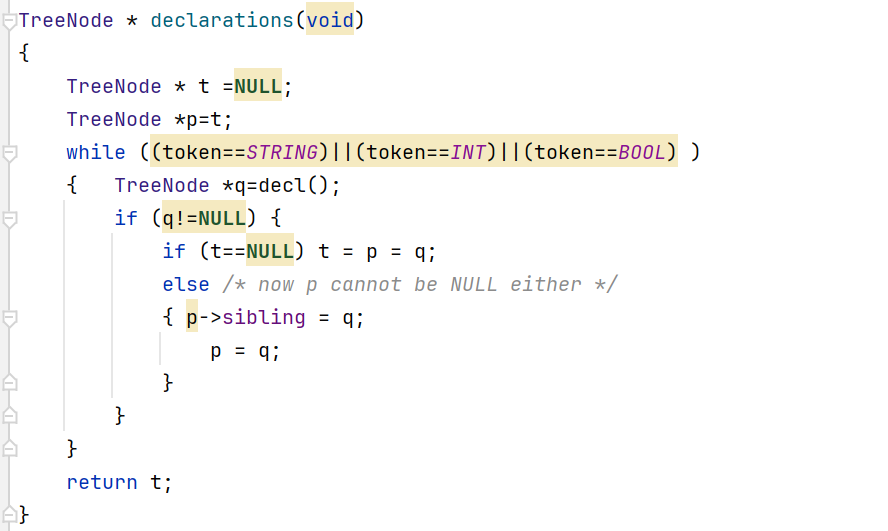


图13 declarations函数代码

下一条产生式decl -> type-specifier varlist, 并且type-specifier->int|bool|string，所以在decl函数中，先根据当前的token创建新的decl节点。decl节点是TINY+语言中新增的，因此在NodeKind中新增一个DeclK, 具体地 DeclK还细分为IntK,BoolK,StringK这三种类型。此外，在Stmt中也顺便新增了WhileK，并且新定义创建Decl节点的newDeclNode函数。

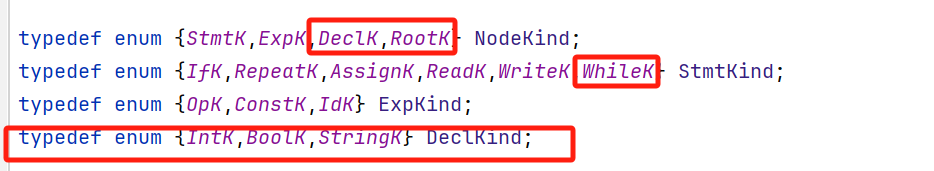


图14 新增的节点类型

通过varlist -> identifier { , identifier }这条产生可以推导出，一个varlist可以推出至少一个标识符，每个标识符之间由逗号分隔。我将每个标识符作为上一个的标识符的子节点，递归地调用varlist，直到没有逗号。

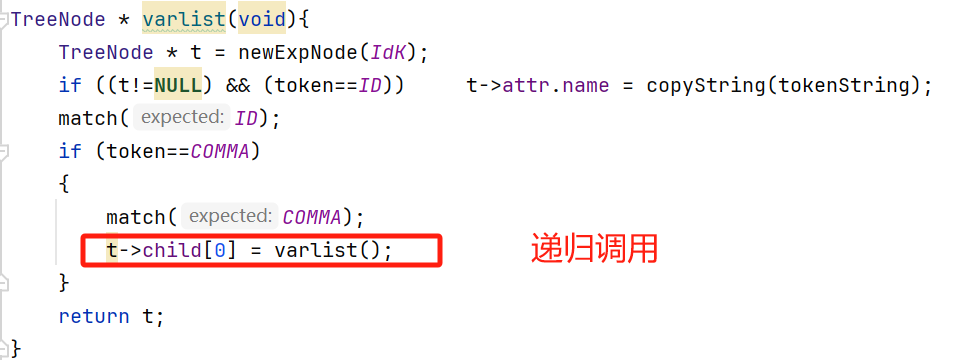


图15 varlist函数代码

**2.新增while-stmt**

很明显的看到stmt可以多产生一种形式的while-stmt，而while-stmt又和repeat-stmt的形式一模一样。所以在statement函数的switch语句中，要新增一种情况，在token为Do时，调用while\_stm（）函数。具体的while\_stmt实现如下，先遇到终结符DO，进行匹配DO，然后调用stmt\_sequence()函数，然后遇到终结符WHILE，进行匹配WHILE，最后调用boolexp()函数。

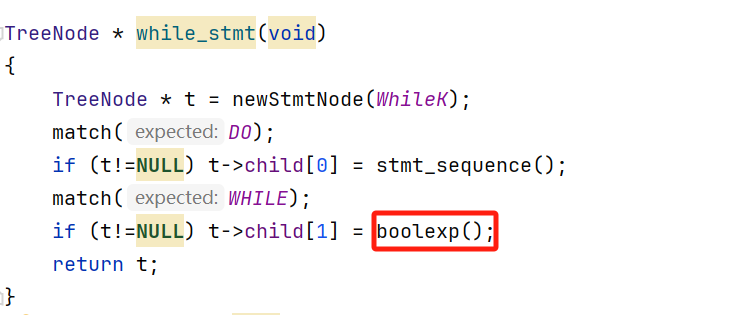


图16 新增while\_stmt函数

while-stmt函数中的boolexp是一个TINY+语言新增的部分，需要自己实现。Boolexp表达的含义就是多个表达式之间用and或or连接，构成一个boolexp。与exp表达式中的加号与减号的地位相似，and 和or 是连接各个exp表达式的节点。一个boolexp至少包含一个exp表达式，因此在while循环外调用一次exp()函数，之后再循环地调用exp，创建节点，直到当前token不为Ｔ＿TRUE或Ｔ＿FALSE。



图17 新增boolexp函数

新增的while\_Stmt中会调用stmt\_sequence，那么stmt\_sequence中while循环的结束条件就应该再加一个，遇到WHILE代表while-stmt中的stmt-seq扫描完成。

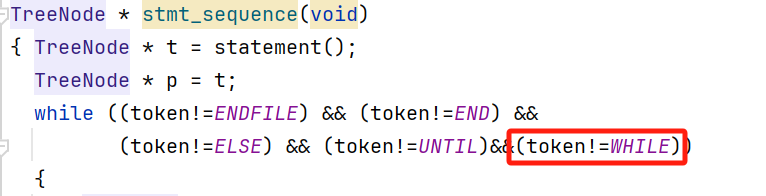


图18 stmt\_sequence中新增WHILE的结束条件

**3.终结符的添加**

虽然给出的TINY+语言产生式中cop 只有小于和等于，但是在样例中存在大于等于。因此要在exp函数中修改添加上<= >= >这三个终结符。

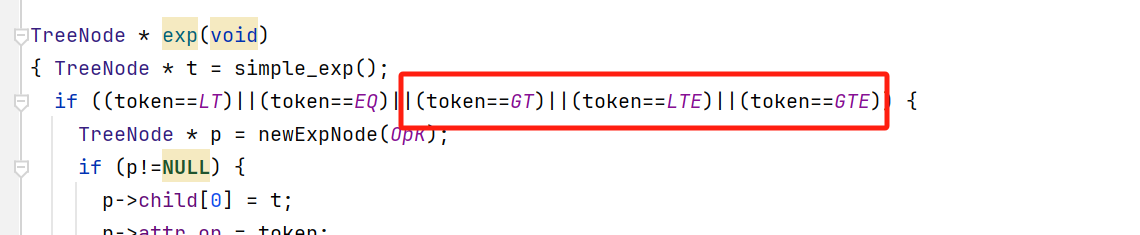


图19 修改exp函数，添加上<= >= >

**4.赋值语句的改变**

虽然给出的产生式赋值语句没有改变，但是在样例中出现了赋字符串的值，需要在assign\_stmt函数中修改。相同的，也可能赋值为true或false。进行分类讨论，如果当前token是STR将节点的类型设为Sstring;如果当前token是T\_TRUE或T\_FALSE,将当前类型设为Boolean.



图20 修改assign\_stmt函数，新增赋值为字符串、布尔值的情况

**5.语法树输出的改动**

新增了两种类型的节点，根节点和声明节点。相应的在printTree函数中修改，是根节点就在控制台直接输出"Program: \n"。

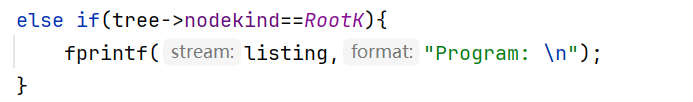


图21 输出根节点

对于声明节点，还需要根据声明的具体类型，输出不同的信息。

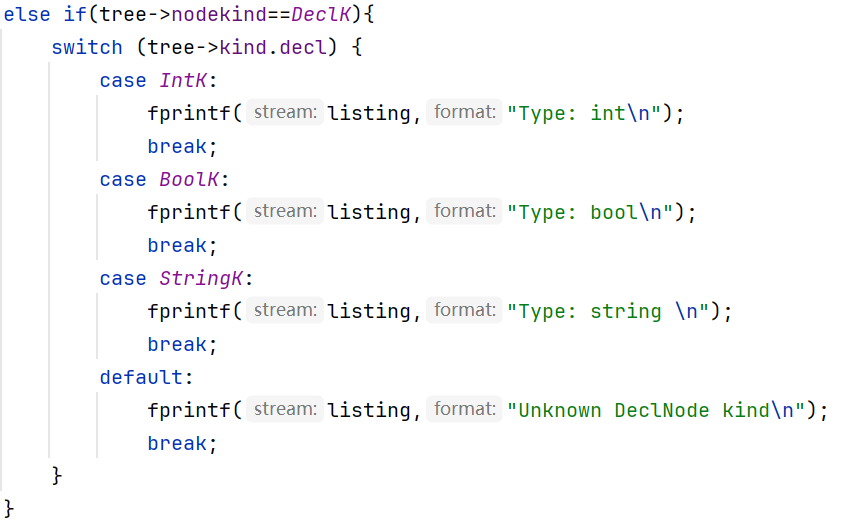
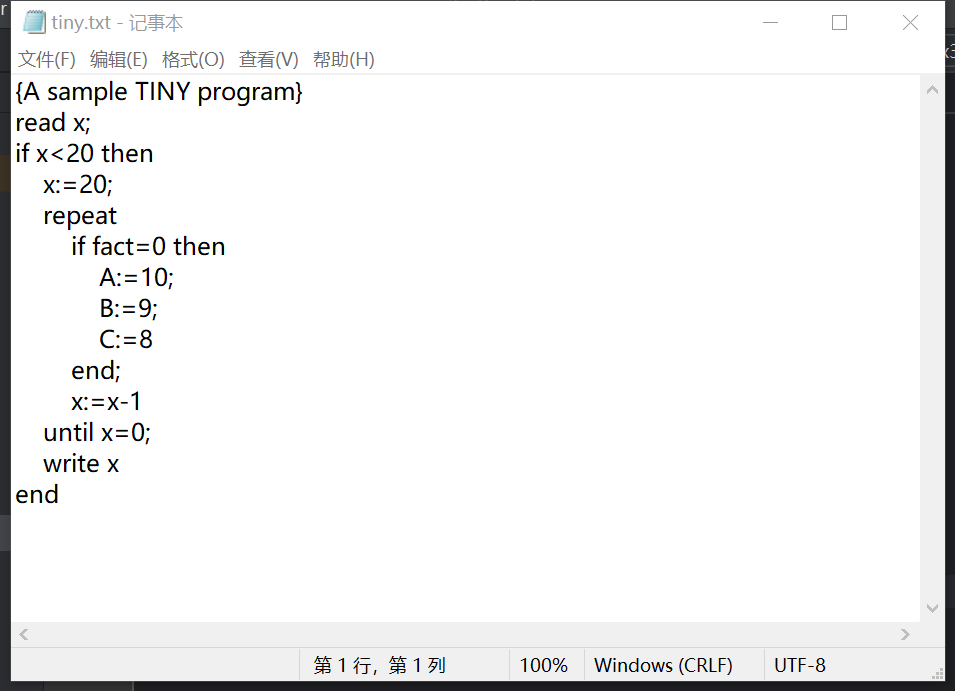


图22 输出声明节点

按照输出样例，还有一些输出格式的小改动，这里不作详细说明。

1. 实验结果

任务一：除了PDF给出的TINY语言样例以外，编写如下图TINY语言，得到的运行结果如下图23（Repeat部分展示有所重叠）：



分析该样例的语法结构，先是一个read结构；然后是一个if结构，在if结构中嵌套一个repeat结构，repeat结构又嵌套一个if语句。以下输出结果正确。

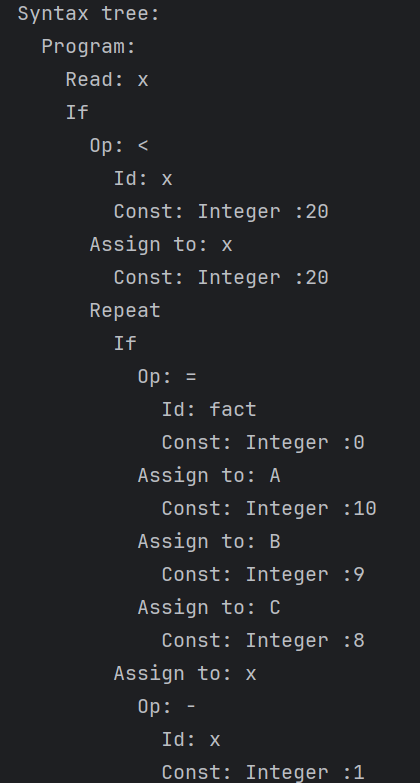
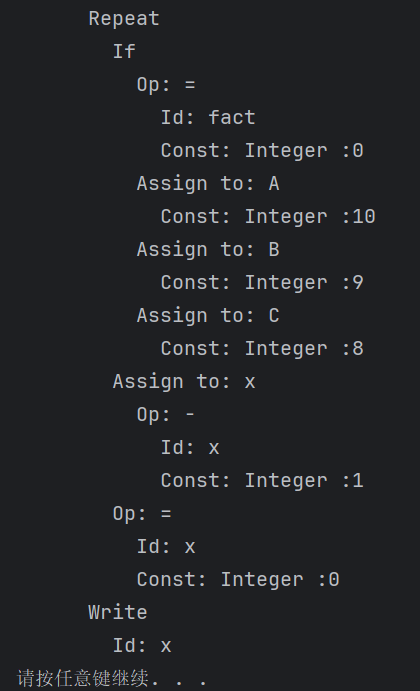
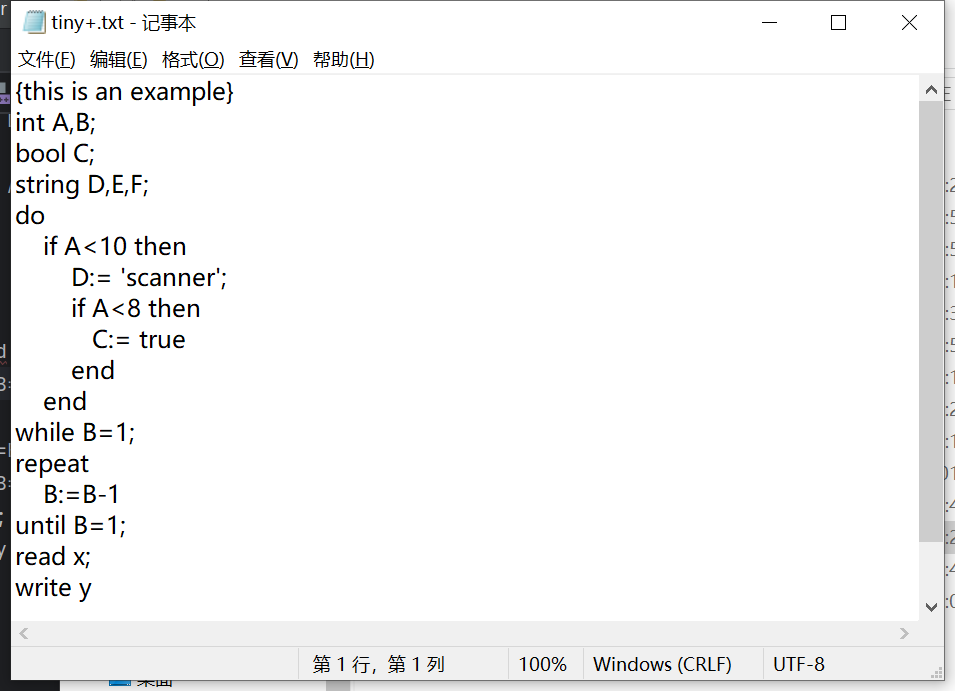
 

图23 TINY语言样例及运行结果

任务二：除了PDF给出的TINY+语言样例以外，编写如下图TINY+语言，得到的运行结果如下图24：



分析该样例的语法结构，前三条声明语句是并列的，三条声明语句是兄弟节点。每条声明语句声明的变量一个是一个的子节点。在do-while结构中嵌套一个if语句，if语句又嵌套一个if语句。之后repeat结构。之后read结构；之后write结构。do-while结构、repeat结构、read结构、write结构之间是并列的，是相邻的兄弟节点。以下输出结果正确。

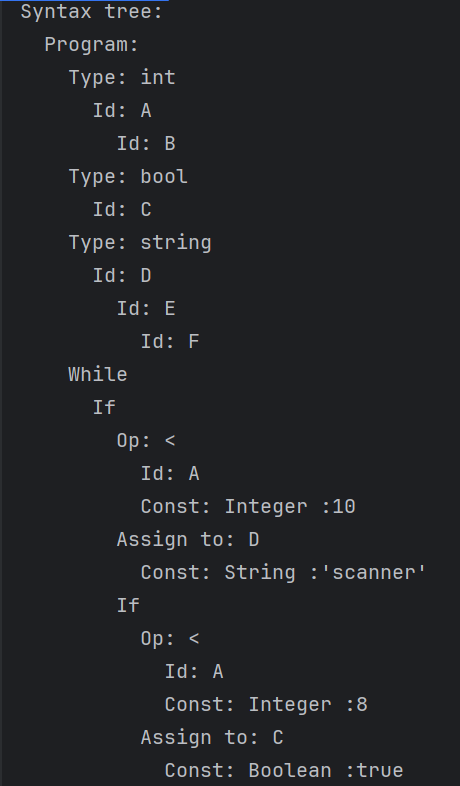
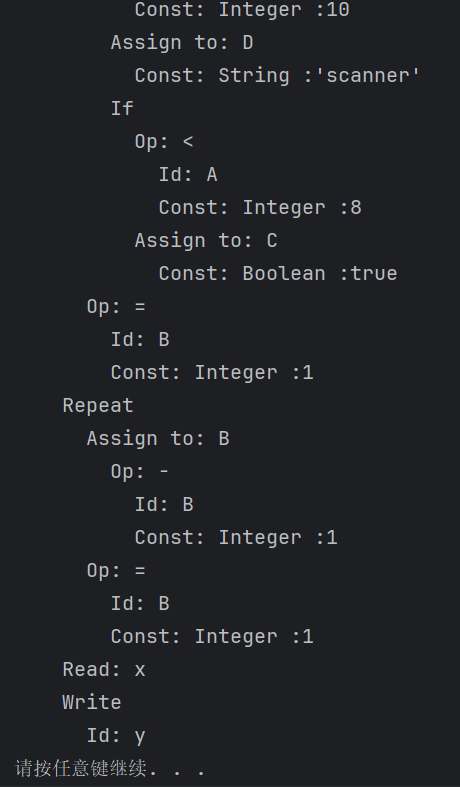
 

图24 TINY+语言样例及运行结果

五、实验总结

该实验为自顶向下的语法分析程序设计，我们需要运行 TINY 语言的语法分析程序TINYParser，理解 TINY 语言语法分析器的实现。在分析给定程序的过程中，才真正学习到了自顶向下语法分析如何编程实现。之后根据 TINY+语法，修改给定的 TINY 语法分析器，实现更新的 TINY+语法分析器，更是自己动手完成了自顶向下语法分析程序的编写，让我对自顶向下的语法分析掌握更加熟练。

本次实验的难点在于分析原有的TINY语言语法分析程序，TINYPraser是一个自顶向下的语法分析去，最主要的是搞清楚构建语法树的规则，搞清楚什么时候要将节点作为孩子节点，什么时候要将节点作为兄弟节点。在理解了自顶向下的思想，搞清楚语法树的结构以及构建语法树的规则之后，只需要根据TINY＋的产生式编写变量对应的程序，构建语法树；并且修改printTree函数的输出规则即可。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  **指导教师签字：**  年 月 日 |
| 备注： |