**编译原理实验三实验报告**

计科MisakaCJQ

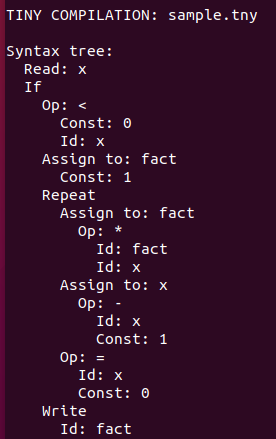
1. **实验目的**

学习已有编译器的经典语法分析源程序。

通过本次实验，加深对语法分析的理解，学会编制语法分析器

1. **实验任务**
2. 阅读已有编译器的经典语法分析源程序，并测试语法分析器的输出
3. 用C或C++语言编写一门语言的语法分析器
4. **实验内容**
5. 测试已有的TINY语言语法分析器

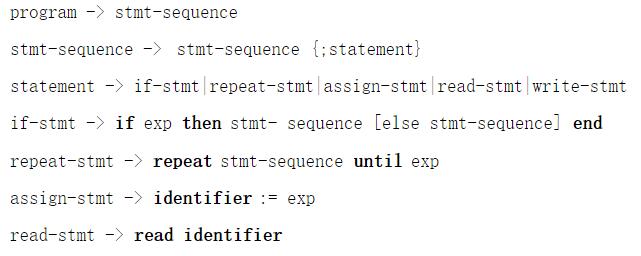
运行./bison.exe sample.tny得到语法树输出

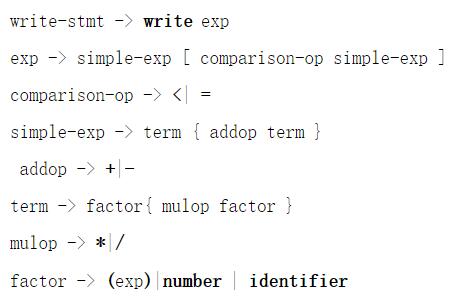


1. TINY语言的文法



消除左递归和提取左公因子后的EBNF：



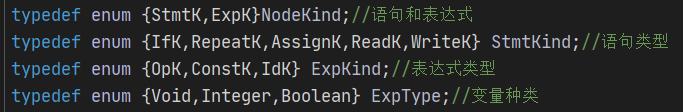


即可依据以上文法编写LL(1)语法分析器

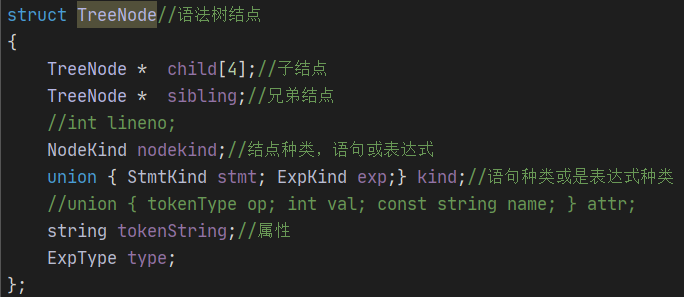
1. 代码实现

首先代码的前半部分为第一次实验中实现的TINY语言词法分析器，这里将其词法分析结果保存为一个存有词法单元的vector数组。以及四种枚举类型。

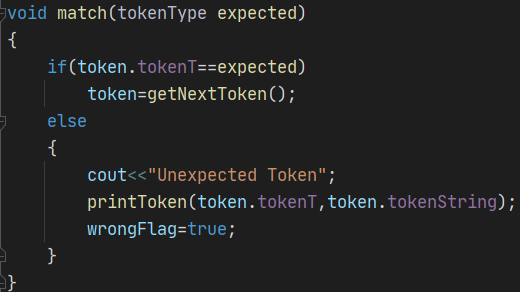




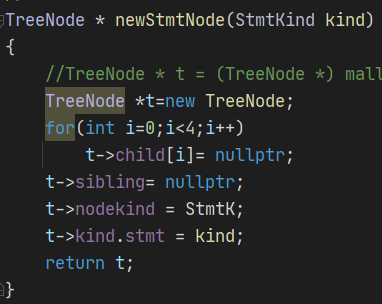
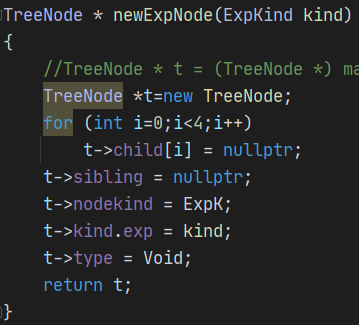
语法树结点：

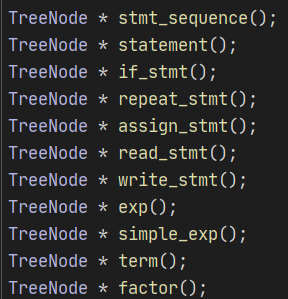


比对向前看词法单元和输入的词法单元的match函数



新建语句结点和表达式结点的函数

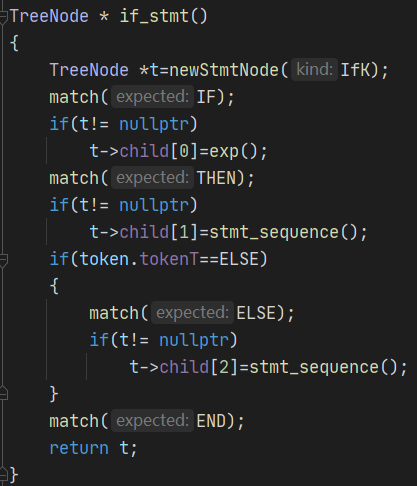
对应每一条产生式的递归处理函数：  


这里以if\_stmt为例：

这一产生式的EBNF为

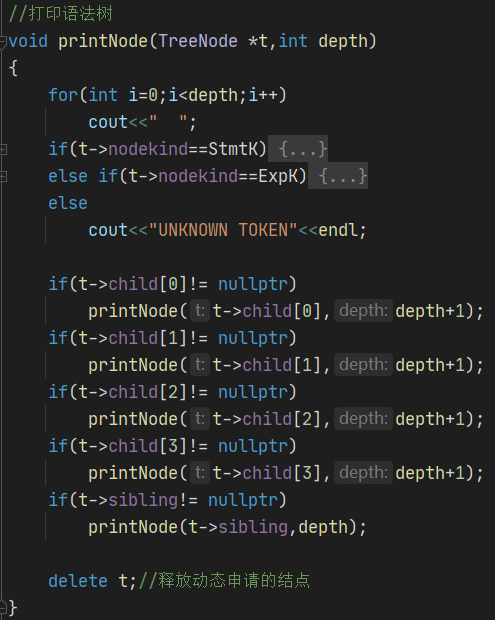
if\_stmt -> if exp then stmt-sequence [else stmt-sequence] end

对应的代码实现为：



其他产生式的实现与此一致。

语法树打印，通过前序遍历进行，同时删除语法树：



1. 样例测试与输出
2. **sample0.tny**

{ Sample program

in TINY language -

computes factorial

}

read x; { input an integer }

if 0 < x then { don't compute if x <= 0 }

fact := 1;

repeat

fact := fact \* x;

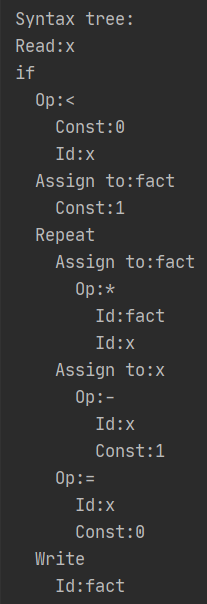
x := x - 1

until x = 0;

write fact { output factorial of x }

end

输出：

语法树正确

**2) sample1.tny**

{sample program 2

cpmpute a+b\*c

}

read a;

read b;

read c;

if 0 < a then

temp := 0;

temp := b \* c;

temp := temp + a;

write temp

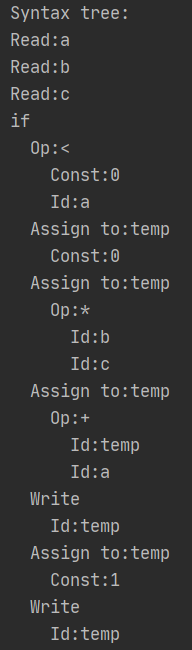
else

temp := 1;

write temp

end

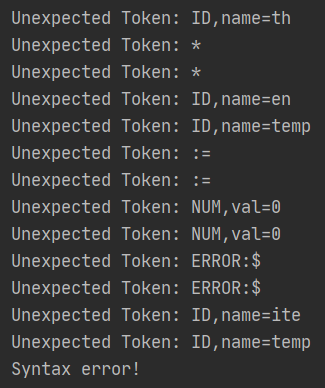
输出：

语法树正确

3)sample2.tny

在上一个样例的基础上加入一些非法字符人为构造一些语法错误。

输出：



报告了错误的词法单元，并不输出语法树。

1. **实验总结**

这一次实验的难点主要在与对TINY语言原始BNF的转化，需要将其转化为EBNF消除左递归才可以通过LL(1)文法实现语法分析器，此外还需要结合实验1实现的词法分析器才能够运行。此外就是要定义好语法树结点的数据结构以及遍历输出的方法。通过这一次实验我更加深入的了解了语法分析的原理，是我在学习编译原理这门课当中的一大收获。