编译原理课程实验 语法分析器

概述

你需要完成一个PL/0语言的编译器。这个实验分为若干部分。在这个部分中，你需要完成PL/0语言的语法分析器。

你应当已经正确完成了词法分析器。如果没有，请先完成词法分析器的实验。

词法分析器会向语法分析器提供词语流。你需要以合适的方式对词语流进行分析，并生成一颗语法树，或者宣告语法错误。

PL/0语言的语法可以用以下上下文无关文法描述：

```
<程序>→<分程序>.
<分程序>→ [<常量说明部分>][<变量说明部分>][<过程说明部分>]<语句>
<常量说明部分> → CONST<常量定义>{ ,<常量定义>};
<常量定义> → <标识符>=<无符号整数>
<无符号整数> → <数字>{<数字>}
<变量说明部分> → VAR<标识符>{ ,<标识符>};
<标识符> → <字母>{<字母>|<数字>}
<过程说明部分> → <过程首部><分程序>;{<过程说明部分>}
<过程首部> → PROCEDURE <标识符>;
<语句> → <赋值语句>|<条件语句>|<当型循环语句>|<过程调用语句>|<读语句>|<写语句>|<复合语句>|<
空语句>
<赋值语句> → <标识符>:=<表达式>
<复合语句> → BEGIN<语句>{ ;<语句>} END
<条件> → <表达式><关系运算符><表达式>|ODD<表达式>
<表达式> → [+|-]<项>{<加减运算符><项>}
<项> → <因子>{<乘除运算符><因子>}
<因子> → <标识符>|<无符号整数>|(<表达式>)
<加减运算符> → +|-
<乘除运算符> → *|/
<关系运算符> → =|#|<|<=|>|>=
<条件语句> → IF<条件>THEN<语句>
<过程调用语句> → CALL<标识符>
<当型循环语句> → WHILE<条件>DO<语句>
<读语句> → READ(<标识符>{ ,<标识符>})
<写语句> → WRITE(<标识符>{,<标识符>})
<字母> → A|B|C…X|Y|Z
<数字> → 0|1|2…7|8|9
<空语句> → epsilon
```

上表中epsilon指\epsilon$\epsilon$，即空

特别的，PL/0语言允许嵌套定义函数（即，上述文法第8行中，由过程说明部分推导出的分程序再次推导出过程说明部分），但嵌套不得超过三层。

例如

```
procedure f1;
    procedure f2;
        procedure f3;
        begin
        end;
    begin
    end;
begin
end;
begin end.
```

是允许的，而

```
procedure f1;
    procedure f2;
        procedure f3;
            procedure f4;
            begin
            end;
        begin
        end;
    begin
    end;
begin
end;
begin end.
```

则是一个语法错误的程序。

Tips:请注意和

```
procedure f1;
begin
end;
procedure f2;
begin
end;
procedure f3;
begin
end;
```

的区别。

对PL/0程序进行语法分析后，如果语法正确，应当生成以<程序>为根节点的抽象语法树。

你应该给出的抽象语法树应当对应规范推导所生成的抽象语法树。这种情况下，对应的答案是唯一的。

我们建议你使用递归下降子程序法进行语法分析。

在本程序中，请忽略语义错误，诸如重复声明的变量、使用未声明的变量等。在语法分析阶段，这些程序暂且还是正确的。

请注意你编写程序的可扩展性。之后的实验需要你对你编写的语法分析程序进行一定的修改。

评测要求

你可以选择自动评测，将这一部分的代码提交到SDU OJ上。通过自动测试后，这一部分的正确性分数将自动评为满分。

如果你不能或不愿进行自动评测，你可以要求人工验收。在这种情况下，这一部分的正确性分数将由助教给出。

如果你计划进行自动评测，你需要从标准输入读取输入的程序，用你完成的词法分析器进行词法分析，之后进行语法分析，并给出语法分析树或报告错误。

如果输入的程序包含词法错误，仅输出一行"Lexical Error"（不含引号）

如果输入的程序包含语法错误，仅输出一行"Syntax Error"（不含引号）

评测程序只会读取你的程序输出到标准输出的内容。你可以向标准错误流打印若干信息以方便调试，美观输出或是出于其他任何原因。

向标准错误流输出信息的示例如下：

C/C++

```
int a = 233;
fprintf(stderr, "Something went wrong. variable a is %d\n", a);
```

C++

```
int a = 233;
cerr << "Something went wrong. variable a is " << a << endl;
```

Java

```
int a = 233;
System.err.printf("Something went wrong. variable a is %d\n", a);
```

在本题中，为自动评测方便，你需要输出语法分析树的括号表示形式。请注意，在最终成品中，你可以选择使用更优美的表示方法。

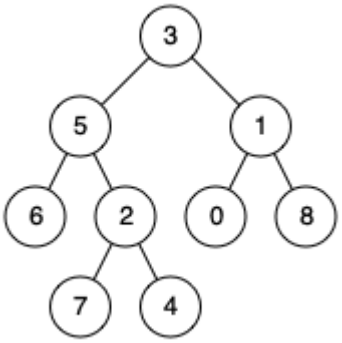
我们建议你先生成抽象语法树，之后对抽象语法树进行遍历来得到括号表示形式，不建议直接生成括号表示形式。本实验是编译器的一部分，而生成括号表示形式仅仅是为了自动评测方便考虑。

树的括号表示形式

根据下列三条规则生成树的括号表示形式

- 如果树T*T*为空，那么它的括号表示形式为空
- 如果树T*T*仅包含一个节点，那么它的括号表示形式为节点本身
- 如果树T*T*由根节点R*R*和若干子树T_1, T_2...T_n*T*1,*T*2...*T***n*组成，那么其括号表示形式为R(T_1*R*(*T*1的括号表示形式,T_2*T*2的括号表示形式, ... ,T_n*T***n*的括号表示形式))

例如



的括号表示形式为

```
3(5(6,2(7,4)),1(0,8))
```

括号表示形式中应当不含空格。

节点名称约定

为评测方便考虑，语法分析树的各节点应当遵循以下命名规则：

- 如果该节点是(，即左括号，该节点应当输出为LP
- 如果该节点是)，即右括号，该节点应当输出为RP
- 如果该节点是,，即逗号，该节点应当输出为COMMA
- 如果该节点是其他的叶子节点（对应词法分析器中的某一个词语），那么其名称为其本身
- 如果该节点不是一个叶子节点，则遵循下表进行替换输出

上下文无关文法中的节点	对应的替换词语
程序	PROGRAM
分程序	SUBPROG
常量说明部分	CONSTANTDECLARE
常量定义	CONSTANTDEFINE
无符号整数	<这是一个叶子节点，用其本身替代>
变量说明部分	VARIABLEDECLARE
标识符	<这是一个叶子节点，用其本身替代>
过程说明部分	PROCEDUREDECLARE
过程首部	PROCEDUREHEAD
语句	SENTENCE
赋值语句	ASSIGNMENT
复合语句	COMBINED
条件	CONDITION
表达式	EXPRESSION
项	ITEM
因子	FACTOR
加减运算符	<这是一个叶子节点，用其本身替代>
乘除运算符	<这是一个叶子节点，用其本身替代>
关系运算符	<这是一个叶子节点，用其本身替代>
条件语句	IFSENTENCE
过程调用语句	CALLSENTENCE
当型循环语句	WHILESENTENCE
读语句	READSENTENCE
写语句	WRITESENTENCE
空语句	EMPTY

提交程序

考虑到对于编译器这样的大型工程将代码分布在多个文件中是一个很实用的做法，我们会采用多文件编译/评测。你需要提供一个构建脚本来告知我们的评测系统如何构建你的答案可执行文件。

你需要把编译所需的代码以及一个名为build.sh的构建脚本压缩在answer.zip压缩包中（压缩包不含子文件夹），文件结构应该类似下图

```
answer.zip
|--build.sh
|--source1.xxx
|--source2.xxx
...
```

上图中的source开头的文件是你的源代码，不必遵循source开头的命名格式，可以任意命名。其中，.xxx是对应语言的后缀，如.cpp对应c++，.java对应java，.c对应C语言。

在执行build.sh后，应该构造出一个名为Main的可执行文件（对于C/C++）或可被java Main执行的Java类文件（对于Java），区分大小写。

这里提供示例build.sh。

下文中的sourcex对应你的源文件，可以任意更名/增加/删除。你也可以自行编写自己的build.sh脚本

C

```
#! /bin/bash
gcc -c source1.c -o source1.o -O2
gcc -c source2.c -o source2.o -O2

gcc source1.o source2.o -o Main -lm
```

C++

```
#! /bin/bash
g++ -c source1.cpp -o source1.o -O2
g++ -c source2.cpp -o source2.o -O2

g++ source1.o source2.o -o Main -lm
```

Java

```
#! /bin/bash
javac *.java
```

样例

样例1

输入

```
CONST C=0;
VAR A,B;
BEGIN
    READ(B,A);
    A:=B+C;
    WRITE(A);
END.
```

输出

```
PROGRAM(SUBPROG(CONSTANTDECLARE(CONST,CONSTANTDEFINE(C,=,0),;),VARIABLEDECLARE(V
AR,A,COMMA,B,;),SENTENCE(COMBINED(BEGIN,SENTENCE(READSENTENCE(READ,LP,B,COMMA,A,
RP)),;,SENTENCE(ASSIGNMENT(A,:=,EXPRESSION(ITEM(FACTOR(B)),+,ITEM(FACTOR(C))))),
;,SENTENCE(WRITESENTENCE(WRITE,LP,A,RP)),;,SENTENCE(EMPTY),END))),.)
```

解释

一个比较适合人类阅读的，经过处理的输出序列是

```
PROGRAM(
    SUBPROG(
        CONSTANTDECLARE(
            CONST,
            CONSTANTDEFINE(
                C,
                =,
                0
            ),
            ;
        ),
        VARIABLEDECLARE(
            VAR,
            A,
            COMMA,
            B,
            ;
        ),
        SENTENCE(
            COMBINED(
                BEGIN,
                SENTENCE(
                    READSENTENCE(
                        READ,
                        LP,
                        B,
                        COMMA,
                        A,
                        RP
                    )
                ),
                ;,
                SENTENCE(
                    ASSIGNMENT(
                        A,
                        :=,
                        EXPRESSION(
                            ITEM(
                                FACTOR(
                                    B
                                )
                            ),
                            +,
                            ITEM(
                                FACTOR(
                                    C
                                )
                            )
```

```
                        )
                    )
                ),
                ;,
                SENTENCE(
                    WRITESENTENCE(
                        WRITE,
                        LP,
                        A,
                        RP
                    )
                ),
                ;,
                SENTENCE(
                    EMPTY
                ),
                END
            )
        )
    ),
    .
)
```

样例2

输入

```
1=2=3=4=5
```

输出

```
Syntax Error
```

样例3

输入

```
VAR A,B;
BEGIN
    A : = B;
END.
```

输出

```
Lexical Error
```

样例4

输入

```
VAR A,B;
BEGIN
    READ(A);
    B := A;
    WRITE(B)
END.
PROCEDURE APPENDIX
BEGIN
;
END
```

输出

```
Syntax Error
```

另有部分实例examples.zip供下载

附录

提供字符串常量供复制

```
"PROGRAM",
"SUBPROG",
"CONSTANTDECLARE",
"CONSTANTDEFINE",
"VARIABLEDECLARE",
"PROCEDUREDECLARE",
"PROCEDUREHEAD",
"SENTENCE",
"ASSIGNMENT",
"COMBINED",
"CONDITION",
"EXPRESSION",
"ITEM",
"FACTOR",
"IFSENTENCE",
"CALLSENTENCE",
"WHILESENTENCE",
"READSENTENCE",
"WRITESENTENCE",
"EMPTY",
"LP",
"RP",
"COMMA"
```

提供pretty.cpp作为工具。这个工具可以将你输出的**符合格式要求的**一行括号表达式形式的语法树转换为类似样例1解释的较为可读的语法树。你可以自行编译它来获得这个工具以方便调试。

pretty.cpp

```
#include<cstdio>
#include<iostream>
#include<string>
#include<vector>
```

```
using namespace std;
vector<string> tokens;
string ans;
void dfs(int &index, int depth)
{
again:
    ans += string(depth, '\t');
    ans += tokens[index];
    index++;
    if (index < tokens.size() && tokens[index] == "(")
    {
        ans += "(\n";
        index++;
        dfs(index, depth + 1);
        if (tokens[index] != ")")throw 233;
        ans += string(depth,'\t') +")";
        index++;
    }
    if (index < tokens.size() && tokens[index] == ",")
    {
        ans += ",\n";
        index++;
        goto again;
    }
    ans += "\n";
}
int main()
{
    string exp;
    cin >> exp;
    string tmp;
    for (int i = 0; i < exp.length(); i++)
    {
        if (exp[i] == '(' || exp[i] == ')' || exp[i] == ',')
        {
            if(!tmp.empty())tokens.push_back(tmp);
            tmp = "";
            tokens.push_back(string(1, exp[i]));
        }
        else tmp += exp[i];
    }
    if (!tmp.empty())tokens.push_back(tmp);
    try
    {
        int id = 0;
        dfs(id, 0);
    }
    catch (...)
    {
        cerr << "Error Parsing Expression" << endl;
        return -1;
    }
    cout << ans << endl;
}
```