Decaf PA 1说明

任务描述

在本阶段中，大家的任务是编码实现Decaf语言编译器的词法分析和语法分析部分。

首先你需要运用词法分析程序自动生成工具Flex　生成Decaf语言的词法分析程序。词法分析程序是前端分析的第一部分，它的功能是从左至右扫描Decaf源语言程序，识别出标识符、保留字、整数常量、操作符等各种单词符号，并把识别结果以终结符的形式返回给语法分析程序。这一部分的实验目的是要掌握LEX工具的用法，体会正规表达式、有限自动机等理论的应用，并对词法分析程序的工作机制有比较深入的了解。

除了词法分析程序以外大家还需要运用语法分析程序自动生成工具BYACC 生成Decaf语言的语法分析程序。Decaf语法分析程序的功能是对词法分析输出的终结符串（注意不是字符串）进行自底向上的分析。这一部分的实验目的是要初步掌握YACC工具的用法，体会上下文无关文法、LALR(1)分析等理论的应用，并对语法分析程序的工作机制有比较深入的了解。

注意阶段一的两部分是密切相关的：语法分析程序并不直接对Decaf源程序进行处理，而是调用词法分析程序对Decaf源程序进行词法分析，然后对词法分析程序返回的终结符序列进行归约，也就是说，词法分析程序输出的结果才是语法分析的输入。

本阶段是整个实验的热身阶段，因此这个阶段的代码量非常少，主要的工作在于建立编程环境、熟悉代码框架、熟悉Decaf语言以及掌握Flex和BYACC的具体用法等。

注意，在BYACC把语法规则转换为语法分析程序的过程中，可能会报移进/归约冲突或者是归约/归约冲突。不消除这些冲突得到的程序也有可能正常工作，但我们要求必须消除这些冲突。

本阶段的测试要求是输出结果跟标准输出完全一致。我们保留了一些测试例子没有公开，所以请自己编写一些测试例子，测试自己的编译器是符合Decaf语言规范里的相应规定。

本阶段持续时间为2周，截止时间以网络学堂为准。请按照《Decaf实验总述》的说明打包和提交。

本阶段涉及的工具和类的说明

实验环境为JDK1.6 以上版本，词法分析器生成工具为JFlex，语法分析器生成工具为BYACC/J，开发环境建议使用eclipse。请注意BYACC本身对%nonassoc修饰的符号的处理不太正常，因此如果你需要进行语法错误检查的话，请勿涉及无结合性符号方面的错误检查（BYACC会直接进行归约而不是报错）。

在TestCases目录下，是我们从最终测试集里面抽取出来的一部分测试用例，你需要保证你的输出和我们给出的标准输出是完全一致的。

本阶段主要涉及的类和文件如下：

文件/类	含义	说明
BaseLexer	词法分析程序基础	你要在此文件中添加必要的常数处理函数
Lexer.l	LEX源程序	你要在此文件中定义正规式，并给出相应的动作。
Lexer	词法分析器，主体是yylex()	由jflex生成
BaseParser	语法分析程序基础	不需要修改，请事先阅读
Parser.y	YACC源程序	你要在其中加入Decaf的语法规则和归约动作
Parser	语法分析器，主体是yyparse()	由byacc自动生成
SemValue	文法符号的语义信息	不要修改

ParserHelper	编写YACC动作的辅助类	在这里编写yacc的动作，然后粘贴到Parser.y中
tree/*	抽象语法树的各种结点	你要在此文件中添加新的结点与相应处理函数
error/*	表示编译错误的类	不要修改
Driver	Decaf编译器入口	调试时可以修改
Option	编译器选项	不要修改
Location	文法符号的位置	不要修改
utils/*	辅助的工具类	可以增加，不要修改原来的部分
build.xml	Ant Build File	不要修改

修改好代码后，运行Ant Builder，会在result目录下产生decaf.jar文件，启动命令行输入java -jar decaf.jar就可以启动编译器。不写任何参数的会输出Usage。

测试和提交方法请参照《Decaf实验总述》。

另外，Lexer与Parser类都提供了diagnose函数用于调试，其中Lexer是依次输出读到的终结符，而Parser是输出归约使用的产生式。

单词符号说明

下面先简单介绍一下Decaf语言的单词符号。Decaf语言的单词符号主要有以下5类:

1、关键字：是预先设定的一组字符串，在Decaf中关键字同时也是保留字，因此这些字符串不能用作标识符，也不能被重新定义。

2、标识符：是以字母开头，后跟若干字母、数字和下划线字符的串。例如，"int_var"是合法的标识符。需要注意的是Decaf语言区分大小写，例如，if是关键字，而IF则是标识符。

3、常量：包括整数、布尔常数、字符串、实数三种。

4、算符和界符（operators and delimiters）：包括单字符的和双字符的两种，其中单字符的算符和界符在词法分析器中直接返回其ASCII 码即可。

5. 注释：Decaf中包括单行注释（以"//"开始，至行尾结束，如果在程序结尾，则最后需要换行）和多行注释。

关于单词符号的进一步说明请参考Decaf语言规范中的词法规范一节。

实验具体任务

在本次实验框架中，对于Decaf语言规范中描述的关键字和语法规则，大部分给出了具体实现。此外，我们引入了一些新的语言特性,要求你通过学习框架中已有的词法语法分析，实现对下述新特性的语法词法分析:

1、去掉对于instanceof的处理。

2、加入repeat,until结构，扩展巴氏范式如下:

```
RepeatStmt        ::= repeat  Stmt  until  (  BoolExpr  )
```

3、引入了double类型，double类型既可以是小数形式也可以是指数形式,但必须是以数字开头,紧接着是小数点,小数点后可以为数字也可以为空。合法的实数例子有:0.12,12.,12.2E+2,12.2e+2。指数形式的E大小写均可,非法的例子有:.12,.12E+2。

4、引入了多行注释。多行注释是以/*开头,以*/结尾且不允许嵌套。注释中可以是除了*/以外的任何字符标识。如下例子:

```
/* this is a comment and it can span multiple lines
and even /* can exist /*inside  comments */
```

PS：对Decaf语言规范中的描述的关键字和语法规则只需在lexer.l和parser.y中添加，上述的新特性还需要在BaseLexer和Tree文件中进行相应修改。

语法规则说明

在Decaf语言规范中给出的参考文法是使用扩展的EBNF（扩展巴氏范式）描述的，但是BYACC并不直接接受EBNF方式描述的文法，因此需要首先把EBNF形式的参考文法改写为BYACC所接受的上下文无关文法。

在改写EBNF为上下文无关文法的时候需要注意一些习惯写法：

1、 如果需要使用递归产生式，首选左递归（原因可以从BYACC Manual中找到）。

2、 形如Stmt*这样的部分，我们将改写为：

StmtList -⟩StmtList Stmt | ε

并且用StmtList替代原文法中的所有**Stmt***的出现；形如**ClassDef**$^{+}$这样的部分，则改写为：

ClassDefList -⟩ClassDefList ClassDef | ClassDef

3、形如⟨**Expr .**⟩这样的部分，我们将改写为：

Receiver -⟩Expr . | ε

并且用Receiver替代原文法中的所有 Expr .的出现。

我们已经在parser.y的模板中为大家定义好了终结符的代表码以及其文法名字（单字符操作符的代表码是其ASCII码，文法名字就是字符本身，例如'+'的文法名字就是'+'，注意有单引号）。在词法分析器中大家只需要识别出对应单词以后返回预先定义好的代表码即可（例如==返回的是EQUAL）。

另外必须注意的是parser.y模板中示例性地给出了大部分操作符的优先级和结合性说明，大家应当根据Decaf语言规范中的说明把这部分补充完整，并体会算符优先分析以及指定算符结合性对消除冲突的作用。

关于详细的语法说明，请参考Decaf语言规范的参考文法一节。

抽象语法树（AST）

所谓的抽象语法树（Abstract Syntax Tree），是指一种只跟我们关心的内容有关的语法树表示形式。抽象语法是相对于具体语法而言的，所谓具体语法是指针对字符串形式的语法规则，而且这样的语法规则没有二义性，适合于指导语法分析过程。抽象语法树是一种非常接近源代码的中间表示，它的特点是：

1、 不含我们不关心的终结符，例如逗号等（实际上只含标识符、常量等终结符）。

2、 不具体体现语法分析的细节步骤，例如对于List -> List E|E这样的规则，按照语法分析的细节步骤来记录的话应该是一棵二叉树，但是在抽象语法树中我们只需要表示成一个链表，这样更便于后续处理。

3、 可能在结构上含有二义性，例如加法表达式在抽象语法中可能是Expr -> Expr + Expr，但是这种二义性对抽象语法树而言是无害的——因为我们已经有语法树了。

4、 体现源程序的语法结构。

使用抽象语法树表示程序的最大好处是把语法分析结果保存下来，后面可以反复利用。

在面向对象的语言中描述抽象语法树是非常简单的：我们只需要为每种非终结符创建一个类。如果存在A -> B的规则的话我们就让B是A的子类（具体实现的时候考虑后面的处理可能有所不同）。

在我们的代码框架中我们已经为你定义好decaf规范中大部分符号在AST中对应的数据结构，但需要你对新引入的符号定义结构和功能，请在动手实现之前大致了解一下decaf.ast包下各个类。

提示

1、我们建议你按照由易到难的顺序，逐步增加词法分析程序识别的单词符号的类别。每增加一类，用Lexer的diagnose函数测试一下。

2、词法分析部分的重点是掌握Flex工具的用法，尤其是要掌握正规式的写法。以下是关于写正规式的几点建议：

- 为避免与LEX操作符混淆，最好对仅含有字符串常量的表达式都用""括起来。
- 有时使用宏可以使正规式更加简洁清晰。
- 可以通过yytext()函数获得当前匹配到的字符串。

3、语法分析器部分的任务比较简单，大家只要把Decaf的EBNF语法规范改写成BYACC能够识别的语法规则并且在相应的归约动作中加入建立AST的语句就可以了。需要注意的是：Decaf语言规范中给出的表达式规则有二义性，YACC在处理时会出现冲突。有两种办法可以去掉冲突：一种是通过改写文法去掉二义性，例如PL/0语言表达式文法；另一种是利用优先级和结合性去掉二义性。在本实验中我们采用第二种方法，目的是希望大家熟悉YACC解决冲突的机制并体会这样做的好处。

4、构造语法分析器的过程中，有一个重要的问题是如何避免冲突。对此，我们有以下几点建议：

- 不要一次把所有的语法规则都添加进去，比较好的方法是逐次添加少量规则，并及时测试，这样不会导致冲突"大爆发"。
- 将左边符号相同的规则合并在一起，这样有利于避免手误。
- 各条规则的右部尽量对齐，这样比较清晰且易修改。
- 如果语法规则需要递归，尽量使用左递归，这样可以使语法分析程序尽早进行归约，不致使状态栈溢出。
- 在调试过程中，可以利用%start子句来指定某个非终结符为开始符号，这样可以将注意力集中在部分规则上。
- 如果希望看到详细的冲突情况的报告，可以查看parser.output文件，里面详细报告了冲突的情况和BYACC选用的解决冲突方法。注意可能YACC默认的处理方法就是符合Decaf语言规范的，但我们仍然要求使用优先级和结合性来明确的消除冲突。

5、要充分考虑到输入的各种可能情况，我们给的例子只能涵盖一部分合法和不合法的输入。请自己设计一些例子，对程序进行全面的测试，测试也是实验的重要内容。