**编译实验1-词法分析**

课程名称：编译技术

实验学时：16 学时

综合性、设计性实验：□是 🗹否

面向专业和班级：软件学院 2017级3、4班

学生人数： 126

任务编制人：彭绍武

# 实验课程安排

* 实验1: 2019-04-12 8:50-12:15
* 实验2: 2019-04-26 8:50-12:15
* 实验3: 2019-05-10 8:50-12:15
* 实验4: 2017-05-17 8:00-12:15

实验报告要求提交到教学在线（具体时间根据实际情况另行通知）

# 实验目的

本课程实验目标是参考和理解教材附录的源码，采用Java语言实现decaf语言的一个编译器，完成decaf语言的词法分析、语法分析、语义分析以及中间代码的生成。

* 1. 实验1: 实现decaf语言的词法分析程序／Flex工具
  2. 实验2: 实现decaf语言的语法分析程／Bison工具
  3. 实验3: 实现语义分析和中间代码生成
  4. 实验4: 基于前三次实验，实现完整的decaf语言编译器

# 实验环境

1. 硬件环境需求

Pentium4 2.8及以上，至少1G以上RAM

1. 软件环境需求

Linux或windowXP以后版本，JVM, Eclipse IDE, Word文档编辑器

# 实验内容

# 4.1 Experiment 1:

**Implementing a Scanner for decaf**

You are to write a lexical analyzer/scanner for the language decaf

## 4.1.1 Goals

1. The input of the scanner is a source code file and the output of the scanner is a stream of tokens.

2. Your scanner should go for longest possible match i.e. a string ‘:=’is to be identified as ‘ass-symbol’ and not as ‘:’and ‘=’.

3. Token is represented as (Kind, Value). We use the following symbols to denote different kinds of tokens

**KEY** denotes reserved words

**SYM** denotes special symbols ID denotes identifiers

**NUM** denotes numeric constants

**STR** denotes string constants

4. Check lexical errors: giving meaning error messages and the lines where errors occur. The kinds of lexical errors are:

- Illegal character, that is, scanner may recognize a character that is not in the alphabet of TINY+, such as ***@*** is an illegal character

- The right bracket of a STRING is lost, such as ‘***scanner***

- The right delimiter of a comment is lost, such as: ***{this is an example***

## 4.1.2 Example output for some decaf programs

Test1 Given the inputs:

class Main {

static void main(){

class Fibonacci f = New Fibonacci();//new a Fibonacci

Print(f.get(ReadInteger()));

}

}

/\*\*

\* Fibonacci

\*/

class Fibonacci {

int get(int i){

if(i<2){

return 1;

}

return get(i-1) + get(i-2);

}

}

The scanner should give the outputs:

(KEY, class) (ID, Main) (SYM, {) (KEY, static) (KEY, void)

(ID, main) (SYM, () (SYM, )) (SYM, {) (KEY, class)

(ID, Fibonacci) (ID, f) (SYM, =) (KEY, New) (ID, Fibonacci)

(SYM, () (SYM, )) (SYM, ;) (KEY, Print) (SYM, ()

(ID, f) (SYM, .) (ID, get) (SYM, () (KEY, ReadInteger)

(SYM, () (SYM, )) (SYM, )) (SYM, )) (SYM, ;)

(SYM, }) (SYM, }) (KEY, class) (ID, Fibonacci) (SYM, {)

(KEY, int) (ID, get) (SYM, () (KEY, int) (ID, i)

(SYM, {) (KEY, if) (SYM, () (ID, i) (SYM, <)

(NUM, 2) (SYM, )) (SYM, {) (KEY, return) (NUM, 1)

(SYM, ;) (SYM, }) (KEY, return) (ID, get) (SYM, ()

(ID, i) (SYM, -) (NUM, 1) (SYM, +) (ID, get)

(SYM, () (ID, i) (SYM, -) (NUM, 2) (SYM, ))

(SYM, ;) (SYM, )) (SYM, ))

# 4.2 Experiment 2:

Check the flex document, and run the example to generate a code in C language, and implement it;

# 备注

# 5.1 Introduction to decaf

Decaf is a strongly typed, object­oriented language with support for inheritance and encapsulation.  By design, it has many similarities with C/C++/Java, so you should find it fairly easy to pick up.  Keep in mind it is not an exact match to any of those languages.  The feature set has been trimmed down and simplified to keep the programming projects manageable.  Even so, you'll still find the language expressive enough to build all sorts of nifty object­oriented programs.

## 5.2 Lexical Conventions of decaf

1. The keywords of the language are the following:

|  |
| --- |
| **void int double bool string class true false null this extends for while if else return new NewArray Print ReadInteger ReaderLine** |

**-** All keywords are reserved.

2. Operators and punctuation characters used by the language includes:

|  |
| --- |
| **+ - \* / % < <= > >= = == != && || ! ; , . [] [ ] ( ) { }** |

3. Whitespace(i.e. spaces, tabs, and newlines)serves to separate tokens, but is otherwise ignored. Keywords and identifiers must be separated by whitespace or a token that is neither a keyword noran identifier. ifintthis is a single identifier, not three keywords. If(23this scans as four tokens.

4.An integer constant can either be specified in decimal (base 10) or hexadecimal (base 16). A decimal integer is a sequence of decimal digits. A hexadecimal integer must begin with 0X or 0x(that is a zero, not the letter oh) and is followed by a sequence of hexadecimal digits.

Hexadecimal digits include the decimal digits and the letters a through f (either upper or lowercase). For example, the following are valid integers:8, 012, 0x0, 0X12aE.

5. A double constant is a sequence of decimal digits, a period, followed by any sequence of digits (maybe none). Thus, .12 is not valid, but both 0.12and 12. are. A double can also have an optional exponent, e.g., 12.2E+2. For a double in this sort of scientific notation, the decimal point is required, the sign of the exponent is optional (if not specified, + is assumed), and the E can be lower or upper case. As above, .12E+12 is invalid, but 12.E+2 is valid. Leading zeros on the mantissa and exponent are allowed.

6. A string constant is a sequence of characters enclosed in double quotes. Strings can contain any character except a newline or double quote. A string must start and end on a single line; it cannot be split over multiple lines:

"this string is missing its close quote

this is not a part of the string above

7. An identifier is a sequence of letters, digits, and underscores, starting with a letter. Decaf is casesensitive,e.g., if is a keyword, but IF is an identifier; binky and Binky are two distinct identifiers.

8. A single-line comment is started by // and extends to the end of the line. C-style comments start with /\* and end with the first subsequent \*/. Any symbol is allowed in a comment except the sequence \*/ which ends the current comment. C-style comments do not nest.

## 5.3 Sample programs in decaf

class Main {

static void main(){

class Fibonacci f = New Fibonacci();//new a Fibonacci

Print(f.get(ReadInteger()));

}

}

/\*\*

\*Fibonacci

\*/

class Fibonacci {

int get(int i){

if(i<2){

return 1;

}

return get(i-1) + get(i-2);

}

}

/\*end\*/