

東南大學

编译原理课程设计 设计报告

组长: _____

成员: _____

东南大学计算机科学与工程学院

二〇〇八年四月二十八日

设计任务名称	SeuLex		
完成时间	<u>2008-4-28</u>	验收时间	
本组成员情况			
学 号	姓 名	承 担 的 任 务	成 绩
		<u>将正规式转为后缀,到NFA 的转换,合并NFA,转换为 DFA, DFA 的最小化,输出状态转移表以及驱动程序。</u>	
		<u>对 Lex 文件的解析。</u>	
		<u>对输出代码进行格式化。</u>	

注：本设计报告中各部分如果页数不够，请自行扩页。原则是一定要把报告写详细，能说明本组设计的成果和特色，能够反映小组中每个人的工作。报告中应该叙述设计中的每个模块。设计报告将是评定各人成绩的重要依据之一。

1 编译对象与编译功能

1.1 编译对象

（作为编译对象的 C 语言子集的词法、语法描述）

详见本报告目录下 Cminus 的 lex 程序文件 [Cminus.1](#)

1.2 编译功能

（所完成的项目功能及对应的程序单元）

1. Lex 输入文件的解析，对应于 LexFileReader 类
2. 正规表达式由中缀转为后缀，对应于 Infix2postfixTransformer 类
3. 正规表达式到 NFA 的转换，对应于 NFA 类
4. 多个 NFA 的合并，对应于 MergedNFA 类
5. NFA 的确定化，以及 DFA 的最小化，对应于 DFA 类
6. MergedNFA 类与 DFA 类中的相应算保证返回状态与返回内容的对应
7. 生成 DFA 状态转移表以及驱动程序的生成，对应于 CodeGen 类
8. 对生成代码进行格式化，以方便阅读，对应于 Formatter 类

<h2>2. 主要特色</h2> <ol style="list-style-type: none">1. 正规定义段只接受形如[A-Z]的定义2. 规则段中. 表示所有单个字符3. 规则段中*表示闭包运算4. 规则段中+表示一个或一个以上的重复5. 规则段中?表示空或一次6. 规则段中 表示或7. 规则段中连接运算符省略8. 规则段中[]表示或, 如果里面有形如 A-Z 的内容则表示 A B ... Z9. 规则段中{}表示花括号内为正规定义, 需要到正规定义段寻找其含义10. 规则段中" " 表示引号中的内容按照字符来看待11. 规则段中()表示优先级12. 规则段中如果想将上述符号当作字符来看待, 则需要在该字符前加上转义符 \13. 生成的 C++程序文件名为 SeuLex_Generated_Code.cpp14. 生成程序中有 seuLex() 函数以供调用, 其返回值为 int 型15. 用户可以在规则段加入 return 语句

1. 正规定义段只接受形如[A-Z]的定义
2. 规则段中.表示所有单个字符
3. 规则段中*表示闭包运算
4. 规则段中+表示一个或一个以上的重复
5. 规则段中?表示空或一次
6. 规则段中|表示或
7. 规则段中连接运算符省略
8. 规则段中[]表示或，如果里面有形如A-Z的内容则表示A|B|...|Z
9. 规则段中{}表示花括号内为正定义，需要到正定义段寻找其含义
10. 规则段中” ”表示引号中的内容按照字符来看待
11. 规则段中()表示优先级
12. 规则段中如果想将上述符号当作字符来看待，则需要在该字符前加上转义符\
\\
13. 生成的C++程序文件名为SeuLex_Generated_Code.cpp
14. 生成程序中有seuLex()函数以供调用，其返回值为int型
15. 用户可以在规则段加入return语句

3 概要设计与详细设计

（由总到分地介绍 SeuLex 的设计，包括模块间的关系，具体的算法等。采用面向对象方法的，同时介绍类（或对象）之间的关系。在文字说明的同时，尽可能多采用规范的图示方法。）

3.1 概要设计

（以描述模块间关系为主）

1. Class SeuLex 入口函数 main()在此类中
2. Class LexFileReader 负责将*.l 文件分割为 C 语言定义段，正规定义段，规则段（返回正规式的数组与 C 语句的数组，下标对应），用户子程序段
3. Class Infix2postfixTransformer 负责将正规式转为后缀形式
4. Class Table 封装了状态转移表的数据结构
5. Class NFA 负责将后缀形式的正规式转为 NFA
6. Class MergedNFA 负责将一个 NFA 的集合合并为一个
7. Class DFA 负责将 NFA 转换为等价的 DFA，以及 DFA 的最小化
8. Class CodeGen 负责生成 DFA 的状态转移表及其驱动程序
9. Class Formatter 负责将生成的代码格式化，以符合阅读习惯
10. Class RegExpr 与 Class Debug 存放了一些常量

3.2 详细设计

（以描述数据结构及算法实现为主）

1. Infix2postfixTransformer 类中的 public String postfix(String regExpr)函数中，postFixExpr 是待返回的后缀字符串，从头到尾扫描字符串 regExpr，如果遇到转义符\，就将转义符后的字符直接加到 postFixExpr 尾部；如果遇到普通字符，就直接将该字符加到 postFixExpr 尾部；如果遇到一元运算符*+?，就将该运算符直接加到 postFixExpr 尾部；如果遇到左括号(，就将其直接压栈；如果遇到右括号)，就将栈内运算符逐个弹出并加到 postFixExpr 尾部直至遇到左括号(；如果遇到|就将栈前部分的|和.运算符全部弹出并加到 postFixExpr 尾部；如果遇到.，就将栈顶前部分的|运算符全部弹出并加到 postFixExpr 尾部。扫描结束后将栈中运算符逐个弹出并加到 postFixExpr 尾部，直至栈空。上述转换过程会顺便将|转为 132，将.转为 131，将?转为 130，将+转为 129，将*转为 128。
2. Table 类封装了状态转移表的所有操作，其中有一个数据成员 t，其类型为 HashSet<Integer>的二维数组，共 128 列，表示 ASCII 码为 0~127 的 128 个字符，行数是动态的。t[s][c]中的内容就表示在状态 s 遇到字符 c 后所能到达的所有状态。成员函数 add(), set().get(), copyLineByRef() 的实现都很简单。
3. NFA 类用一个成员变量 protected Table trsTbl 来表示其状态转移表。成员函数 or(), and(), star(), oneMore(), oneNone() 分别接受一个或两个 NFA 类型的参数并返回一个 NFA 类型，上述函数分别对应于 NFA 的|. *+?等运算，上述函数都是 Thompson's construction 的实现。NFA 的构造函数接受一个 String 类型的参数 regExpr，表示一个正规表达式的后缀形式，构造函数的功能是生成该正规式的 NFA，函数逐个扫描 regExpr 中的内容，如果遇到字符（也就是 ASCII 码小于 128 的），就构造一个只有两

个状态的 NFA 表示这一个字符，然后将这个 NFA 压栈；如果遇到运算符（也就是 ASCII 码大于等于 128 的），就从栈中弹出一个或两个 NFA（视该运算符为一元或二元而定），然后调用 `or()`, `and()`, `star()`, `oneMore()`, `oneNone()` 等函数构造新 NFA，并将新 NFA 压栈。扫描结束后留在栈内的 NFA 就是最终结果。

4. MergedNFA 类的构造函数的参数为 NFA 的数组，其功能是将若干 NFA 合并，使用的算法类似于 Thompson' s construction 中的|运算。该类包含一个数据成员 `private int finalStates[]`; `finalStates[i]` 表示第 i 状态为中止状态时所对应的正规式编号，-1 表示状态 i 不是中止状态。

5. DFA 类负责对 NFA 进行确定化和最小化。

成员函数 `private Set<Integer> epsilonClosure(Set<Integer> T)` 计算 epsilon 闭包。用的是课本上的算法，这个算法类似于图的遍历算法。类中有一个成员变量 `private HashMap<Integer, HashSet<Integer>> espClsTbl` 用来保存已经计算过的单个 NFA 状态的 epsilon 闭包，避免重复计算。

构造函数中用变量 `HashMap<Set<Integer>, Set<Integer>[]> Dtran` 暂时存放生成的状态转移表，这样存放虽然内存开销较大但却为下面的计算带来了便利，用变量 `HashMap<Set<Integer>, Integer> Dstates` 将若干 NFA 编号的集合对应到一个 DFA 编号。接下实现课本中的 subset construction，构造出一个 DFA 都存在变量 `HashMap<Set<Integer>, Set<Integer>[]> Dtran` 中如下图所示

	a	b
{0, 1, 3}	{0, 1, 2, 3}	{1, 2, 4}	...
{0, 1, 2, 3}
{1, 2, 4}
.....

`HashMap<Set<Integer>, Integer> Dstates = new HashMap<Set<Integer>, Integer>()` 是上述格子中集合到数字的一个映射，方便下面将上述状态转移表里的集合全部换为数字。

成员函数 `public void minimize()` 负责将 DFA 最小化，首先将所有状态按照中止状态所对应的正规式的不同而划分为 N+1 类（N 为正规式数目），存放于变量 `HashSet<HashSet<Integer>> pi` 中，然后按照书上算法，考察 pi 的每一个元素 G，此元素为 DFA 状态的集合，如果有一个状态 s 接收到任意一个字符 c 后所转移到的状态不再 G 中，就将 G 分为 {s} 和 G-{s}。重复上述分裂过程，直至不再产生新的集合。然后 pi 中的一个集合代表 DFA 最小化后的一个状态。然后将集合变换为数字，填到状态转移表 `private int trsTbl[][]` 中。

6. DFA 的模拟函数（也就是 DFA 状态转移表的驱动函数）我是在 VC 下先调试成功后，再将其写到 CodeGen 类中的，该类中的 `public void genDriver(String rules[])` 函数，基本上全是输出语句，输出的是我实现 VC 下写好的驱动函数，以及用户规则段（通过数组 rules 传进来的）。

CodeGen 类中的 `public void genTable(int t[][], String name)` 函数负责生成 DFA 的状态转移表到 C++ 源程序文件，该函数的实现很容易。

7. LexFileReader 类中的 `public int getRegularExpressionNumber()` 函数中，逐行读入文本，遇到“%%”后开始计数，再次遇到“%%”停止计数，其中当遇到“{”符号时继续往下读，直到读到“}”才为计数器加一，其他情况默认每个非空行算一条，最后返回 int 型数据作为正规式个数。`public String getRegDef(String s)` 函数中，逐行读入文本，遇到“%”后开，遇到“%%”停止，动态建立两个 String 类的数组，tab1[] 存放名称，tab2[] 存放对应地

表达式,函数输入一个String类数据,并逐个与tab[]中的成员对比,若相等则取出对应的tab2[]中的数据返回.public String[] getRegularExpression()函数中,用getRegularExpressionNumber()一样的方法识别每个正规式,并存入String类数组然后返回,其中,遇到"\"时int型temp++,遇到"\"则temp--,只有在temp等于零时读到空格或TAB才算结束.public String[] getCCode()函数中,用getRegularExpressionNumber()一样的方法识别每个正规式,并将其后面地C代码段存入String类数组然后返回.public String getCDefPart()和public String getCSubRoutine()函数都是逐行读入,以特定字符串开头和结尾,最后返回String类型数据.

8. Formatter类对输出的原始代码格式化便于查看,SeuLex_Generated_Code.cpp为原始输出文件,formatter.cpp是格式化后的文件.函数format为详细的格式化方法,具体思想如下: inFile与outFile为输入输出流;buffer数组存储输出字符;i原来判断文件是否为空;j的量与'{' '}'相关,如与到一个'{' j+1,遇到'}' j-1;根据j的值确定在每次换行时加入几个空格来调整格式;分3种情况:如遇'\n'与'{'连续,输出'\n'后再输出j-1个'\t';如遇'\n',输出'\n'后再输出j个'\t';其他情况输出原始文件的字符.

4 使用说明

4.1 SeuLex 使用说明

1. 输入 Lex 文件的注意事项见主要特色部分。
2. 在命令行中敲入 `java SeuLex *.l`，其中*.l 为 lex 文件名
3. 程序将输出名为 `SeuLex_Generated_Code.cpp` 的 C++源程序文件
4. 生成的词法分析程序提供函数
`int seuLex(std::istream& cin = std::cin, std::ostream& cout = std::cout)`，以供语法分析程序调用
5. 生成的词法分析程序提供一个全局变量 `std::string seuLexLastLex`；用来存放最近一次分析出来的单词
6. 生成的 C++源程序在 VC++中有些时候会编译不过，但用 G++编译通常没有问题。

5 测试用例与结果分析

1. 测试用 [Cminus.1](#) 文件，在本报告所在目录下。

```
C:\WINDOWS\system32\cmd.exe
D:\My_Documents\Eclipse_Workspace\seuLex2\bin>java SeuLex Cminus.1
Constructing NFAs...
Job Done! Got NFA with 4 States ---- "int"
Job Done! Got NFA with 5 States ---- "char"
Job Done! Got NFA with 5 States ---- "bool"
Job Done! Got NFA with 6 States ---- "float"
Job Done! Got NFA with 7 States ---- "double"
Job Done! Got NFA with 5 States ---- "void"
Job Done! Got NFA with 5 States ---- "null"
Job Done! Got NFA with 5 States ---- "true"
Job Done! Got NFA with 6 States ---- "false"
Job Done! Got NFA with 3 States ---- "if"
Job Done! Got NFA with 5 States ---- "else"
Job Done! Got NFA with 6 States ---- "while"
Job Done! Got NFA with 7 States ---- "return"
Job Done! Got NFA with 453 States ---- <alpha><alphanum>*
Job Done! Got NFA with 80 States ---- <digit>+<\.<digit>+>?
Job Done! Got NFA with 2 States ---- "="
Job Done! Got NFA with 3 States ---- "+="
Job Done! Got NFA with 3 States ---- "-="
Job Done! Got NFA with 3 States ---- "*="
Job Done! Got NFA with 3 States ---- "/="
Job Done! Got NFA with 3 States ---- "%="
Job Done! Got NFA with 2 States ---- "-"
Job Done! Got NFA with 2 States ---- "+"
```

```
C:\WINDOWS\system32\cmd.exe
Job Done! Got NFA with 2 States ---- "*"
Job Done! Got NFA with 2 States ---- "/"
Job Done! Got NFA with 2 States ---- "%"
Job Done! Got NFA with 2 States ---- "!"
Job Done! Got NFA with 3 States ---- "&&"
Job Done! Got NFA with 3 States ---- "||"
Job Done! Got NFA with 3 States ---- "=="
Job Done! Got NFA with 2 States ---- "<"
Job Done! Got NFA with 2 States ---- ">"
Job Done! Got NFA with 3 States ---- "<="
Job Done! Got NFA with 3 States ---- ">="
Job Done! Got NFA with 3 States ---- "!="
Job Done! Got NFA with 2 States ---- ","
Job Done! Got NFA with 2 States ---- ";"
Job Done! Got NFA with 2 States ---- "<"
Job Done! Got NFA with 2 States ---- ">"
Job Done! Got NFA with 2 States ---- "<"
Job Done! Got NFA with 2 States ---- ">"
Job Done! Got NFA with 2 States ---- "\n"
Job Done! Got NFA with 10 States ---- [\t\r]
Job Done! Got NFA with 503 States ---- .
Merging NFAs...
Job Done! Got Merged NFA with 1181 States
Constructing DFA...
Job Done! Got DFA with 264 States
```

```
C:\WINDOWS\system32\cmd.exe
Minimizing DFA...
Job Done! Got Minimized DFA with 109 States
Generating Code...
Job Done! Got Origin C++ Code in temp.data
Formatting Code...
Job Done! Got Formatted C++ Code in SeuLex_Generated_Code.cpp

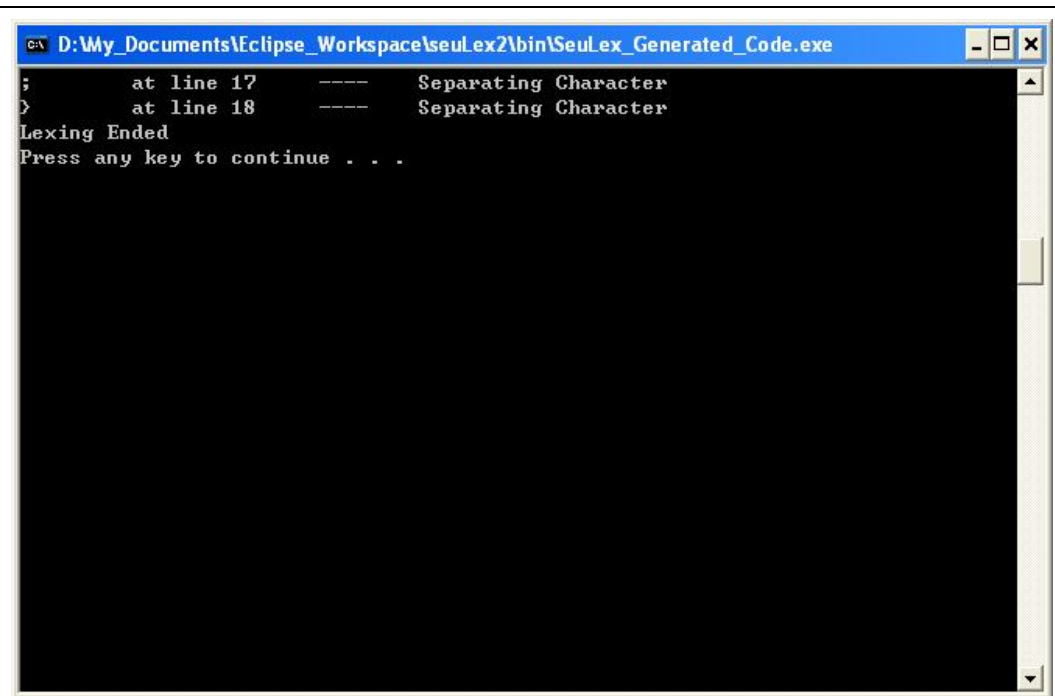
D:\My_Documents\Eclipse_Workspace\seuLex2\bin>pause
Press any key to continue . . .
```

2. 生成 [SeuLex_Generated_Code.cpp](#) 文件，在本报告所在目录下。
3. 编译此 C++ 文件，生成 [SeuLex_Generated_Code.exe](#) 可执行文件，在本报告所在目录下。
4. Cminus 源程序 [test.c](#)，在本报告所在目录下
5. 运行 [SeuLex_Generated_Code.exe](#)，得到如下结果

```
D:\My_Documents\Eclipse_Workspace\seuLex2\bin\SeuLex_Generated_Code.exe
int      at line 1      ---- Primitive Type
main     at line 1      ---- Identifier
<        at line 1      ---- Separating Character
>        at line 1      ---- Separating Character
<        at line 2      ---- Separating Character
double   at line 3      ---- Primitive Type
p        at line 3      ---- Identifier
=        at line 3      ---- Assign Operator
1.2354   at line 3      ---- Number
;        at line 3      ---- Separating Character
float    at line 4      ---- Primitive Type
a        at line 4      ---- Identifier
=        at line 4      ---- Assign Operator
10.55    at line 4      ---- Number
;        at line 4      ---- Separating Character
int      at line 5      ---- Primitive Type
b        at line 5      ---- Identifier
=        at line 5      ---- Assign Operator
0        at line 5      ---- Number
;        at line 5      ---- Separating Character
bool     at line 6      ---- Primitive Type
flag     at line 6      ---- Identifier
=        at line 6      ---- Assign Operator
true     at line 6      ---- Internal Constant
;        at line 6      ---- Separating Character
```

```
D:\Wyy_Documents\Eclipse_Workspace\seuLex2\bin\SeuLex_Generated_Code.exe
while    at line 7      ----    Flow Controller
<        at line 7      ----    Separating Character
a        at line 7      ----    Identifier
<=       at line 7      ----    Relation Operator
b        at line 7      ----    Identifier
!!       at line 7      ----    Logical Operator
!        at line 7      ----    Logical Operator
<        at line 7      ----    Separating Character
p        at line 7      ----    Identifier
!=       at line 7      ----    Relation Operator
0        at line 7      ----    Number
>        at line 7      ----    Separating Character
&&       at line 7      ----    Logical Operator
flag     at line 7      ----    Identifier
>        at line 7      ----    Separating Character
<        at line 8      ----    Separating Character
flag     at line 9      ----    Identifier
=        at line 9      ----    Assign Operator
a        at line 9      ----    Identifier
>        at line 9      ----    Relation Operator
b        at line 9      ----    Identifier
;        at line 9      ----    Separating Character
p        at line 10     ----    Identifier
*=       at line 10     ----    Assign Operator
a        at line 10     ----    Identifier
```

```
D:\Wyy_Documents\Eclipse_Workspace\seuLex2\bin\SeuLex_Generated_Code.exe
;        at line 10     ----    Separating Character
a        at line 11     ----    Identifier
-=       at line 11     ----    Assign Operator
1        at line 11     ----    Number
;        at line 11     ----    Separating Character
>        at line 12     ----    Separating Character
if       at line 13     ----    Flow Controller
<        at line 13     ----    Separating Character
p        at line 13     ----    Identifier
>=       at line 13     ----    Relation Operator
10       at line 13     ----    Number
!!       at line 13     ----    Logical Operator
false    at line 13     ----    Internal Constant
>        at line 13     ----    Separating Character
p        at line 14     ----    Identifier
=        at line 14     ----    Assign Operator
10       at line 14     ----    Number
;        at line 14     ----    Separating Character
else     at line 15     ----    Flow Controller
p        at line 16     ----    Identifier
=        at line 16     ----    Assign Operator
100      at line 16     ----    Number
;        at line 16     ----    Separating Character
return   at line 17     ----    Flow Controller
0        at line 17     ----    Number
```



```
C:\ D:\My_Documents\Eclipse_Workspace\seuLex2\bin\SeuLex_Generated_Code.exe
; at line 17 ---- Separating Character
> at line 18 ---- Separating Character
Lexing Ended
Press any key to continue . . .
```

6 课程设计总结（包括设计的总结和需要改进的内容）

在做 seuLex 的过程中，我们对上学期所学的此法分析部分更下熟悉了。通过实现编译原理课本中的一些算法，我们的程序设计能力也得到提高。在做项目的过程中我们合作解决问题的能力也得到锻炼。

本程序对输入文件的错误控制能力比较弱，这是需要改进的地方。

7 教师评语

签名：_____

附：包含源程序、可运行程序、输入数据文件、输出数据文件、答辩所做 PPT 文件、本设计报告等一切可放入光盘的内容的光盘。