

東南大學

编译原理课程设计

seuLex 设计报告

成员： 09013430 任杰文

09013429 黄路遥

09013413 钱鑫

东南大学计算机科学与工程学院

二 016 年 5 月

设计任务名称		SeuLex	
完成时间		<u>2016/5/18</u>	验收时间
本组成员情况			
学 号	姓 名	承 担 的 任 务	成 绩
09013430	<u>任杰文</u>	<u>Lex 全部任务：分析，设计，实现。</u> <u>相关文档报告编写。</u>	
09013429	<u>黄路遥</u>	<u>YACC</u>	
09013413	<u>钱鑫</u>	<u>YACC</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)

Cminus.1 文件内容如下:

```
%{  
#include <fstream>  
using namespace std;  
typedef enum  
{  
    INT = 1,  
    FLOAT,  
    ID,  
    STRUCT,  
    IF,  
    ELSE,  
    RETURN,  
    NUM,  
    LPAREN,  
    RPAREN,  
    LCOMMENT,  
    RCOMMENT,  
    LBRACK,  
    RBRACK,  
    ASSIGN,  
    SEMI,  
    COMMA,  
    MINUS,  
    PLUS,  
    TIMES,  
    OVER,  
    MOD,  
    EQ,  
    CHAR,  
    BOOL,  
    DOUBLE,  
    VOID,  
    _NULL,  
    TRUE,  
    FLASE,  
    WHILE,  
    PLUSASSIGN,  
    MINUSASSIGN,
```

```

    TIMESASSIGN,
    OVERASSIGN,
    MODASSIGN,
    LOGICNOT,
    LOGICAND,
    LOGICOR,
    NEQ,
    LT,
    GT,
    LEQ,
    GEQ,
    NL,
    SPACE,
    ERROR
} TokenType;

int lineno = 1;
}%

alpha      [A-Za-z]
digit      [0-9]
alphanum   [A-Za-z0-9]

%%

"int"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tPrimitive Type\n";return INT;
"char"     cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tPrimitive Type\n";return CHAR;
"bool"     cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tPrimitive Type\n";return BOOL;
"float"    cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tPrimitive Type\n";return FLOAT;
"double"   cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tPrimitive Type\n";return DOUBLE;
"void"     cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tPrimitive Type\n";return VOID;

"NULL"     cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tInternal Constant\n";return _NULL;
"true"     cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tInternal Constant\n";return TRUE;
"false"    cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tInternal Constant\n";return FLASE;

"if"       cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tFlow Controller\n";return IF;
"else"     cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tFlow Controller\n";return ELSE;
"while"    cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tFlow Controller\n";return WHILE;
"return"   cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tFlow Controller\n";return RETURN;

{alpha}{alphanum}*   cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tIdentifier\n";return ID;
{digit}+("."{digit})? cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tNumber\n";return NUM;

"="           cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tAssign Operator\n";return ASSIGN;
"+="         cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tAssign Operator\n";return PLUSASSIGN;

```

```

"-= "      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tAssign Operator\n";return
MINUSASSIGN;
"*="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tAssign Operator\n";return
TIMESASSIGN;
"/="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tAssign Operator\n";return
OVERASSIGN;
"%="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tAssign Operator\n";return MODASSIGN;

"- "      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tArithmetic Operator\n";return MINUS;
"+"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tArithmetic Operator\n";return PLUS;
"*"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tArithmetic Operator\n";return TIMES;
"/"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tArithmetic Operator\n";return OVER;
%"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tArithmetic Operator\n";return MOD;

"! "      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tLogical Operator\n";return LOGICNOT;
"&&"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tLogical Operator\n";return LOGICAND;
"||"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tLogical Operator\n";return LOGICOR;

"=="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tRelation Operator\n";return EQ;
"<"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tRelation Operator\n";return NEQ;
">"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tRelation Operator\n";return LT;
"<="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tRelation Operator\n";return GT;
">="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tRelation Operator\n";return LEQ;
"!="      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tRelation Operator\n";return GEQ;

","      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return
COMMA;
";"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return SEMI;
"{"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return
LCOMMENT;
"}"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return
RCOMMENT;
"("      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return LPAREN;
")"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return RPAREN;
"["      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return LBRACK;
"]"      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tSeparating Character\n";return RBRACK;

\n      lineno++;return NL;
[" "\t\r]      /*ignore white space*/ return SPACE;
.      cout<<seuLexLastLex<<"\t at line "<<lineno<<"\t---\tBad Character\n"; return -1; return
ERROR;
%%

int main()

```

```

{
    ifstream in("demo.cpp");
    while( seuLex(in) != -1);
    cout<<"Lexing Ended\n";
    system("pause");
    return 0;
}

```

1.2 编译功能

（所完成的项目功能及对应的程序单元）

- FileReader 类: Cminus.1 文件读入, 将其内容分割为 C 语言定义段, 正规定义段, 规则段 (返回正规式及其对应 C++ 程序), 用户子程序段。
- FormatRegExp:

根据 FileReader 类提供的正规定义、正规式, 处理输入正规式, 将正规式转化为标准表达形式:

只含有运算符: 连接 ·, 或 |, 闭包*, 正闭包+, 一次或空 ?, 左小括号 (, 右小括号)。

并且为了将普通符号与运算符区分开, 用 ASCII 码不用的常量整数替换表示为:

```

const char STAR = 128;
const char ONE_OR_MORE = 129;
const char ONE_OR_NONE = 130;
const char AND = 131;
const char OR = 132;
const char LEFT_LITTLE_BRACKET = 133;
const char RIGHT_LITTLE_BRACKET = 134;

```

将标准化的正规表达式转为后缀形式, 同时去掉小括号。方便求 NFA 运算。

- TransTable 类: 存放 NFA 状态转化表。
- NFA 类: 计算后缀正规表达式, 通过 Thompson's Algorithm 将之转化为对应的 NFA (初始状态标号默认 0, 状态数, 状态转化表, Token 识别映射表)。
- MergedNFA 类: 派生自 NFA, 合并多个 NFA (初始状态标号默认 0, 状态数, 状态转化表, Token 识别映射表)
- DFA 类: 对合并的 NFA 进行确定化, 并最小化 DFA (初始状态标号, 状态数, 状态转化表, Token 识别映射表)
- CodeGenerator 类:
 - 生成程序框架;
 - 根据 Cminus.1 语言定义段生成包含头文件, 变量声明等代码;
 - 根据生成 DFA 生成状态转化表, Token 识别映射表;
 - 根据初始状态标号, 初始化初始状态;
 - 根据 Cminus.1 规则段生成识别对应 Token 执行的代码;
 - 根据 Cminus.1 用户子例程段生成用户子例程;

2. 主要特色

1) 正规定义段可接受定义形式:

- a) alpha [A-Za-z]
- b) digit [0-9]
- c) alphanum [A-Za-z0-9]

2) 规则段中支持多种运算符:

- 连接: 省略表示
- 或: | 表示
- 闭包: *表示
- 正闭包: +表示
- 一次或一次以上重复: ? 表示
- 一定范围的字符或运算: 如[A-Z]表示 A|B|……|Z
- 表示引用正规定义: 如{alpha}表示引用[A-Za-z]
- 表示优先级: ()
- 运算符内容需作为普通符号使用只需加" " : 如 "|" 解释为字符串"|", 而不表示或运算. 从而能够识别左右中括号 [];
- 通配符表示任意字符: .
- 生成的 C++程序文件名为 yylex.cpp
- 生成程序中有提供 int seuLex(instream & in, ostream &out)函数原型, 返回值为 cminus.1 中定义的 Token 类型, 文件结束时, 返回-1; 供语法分析程序调用。

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) SeuLex 共包含 7 个类文件, 一个 header.h 文件和一个 main 函数文件;
- 2) 其中 header.h 中声明了包含的头文件, 以及定义了常量:
//用 ASCII 码不用的常量整数表示: 闭包、正闭包、或空串、与、或、空串

```
const char STAR = 128;  
const char ONE_OR_MORE = 129;  
const char ONE_OR_NONE = 130;  
const char AND = 131;  
const char OR = 132;  
const char LEFT_LITTLE_BRACKET = 133;  
const char RIGHT_LITTLE_BRACKET = 134;  
//定义空串 ascii 码为 0  
const char EPSILON = 0;  
//定义 NFA 状态转化表列数, 即 NFA 可接收符号数 (为 ASCII 码 0 - 127 )  
const int NUM_OF_COLUMNS = 128;
```
- 3) main 函数文件定义了程序的入口, 调用各个类, 将读入的 cminus.l 解析, 构造 NFA, DFA, 生成词法分析程序。
- 4) FileReader 类: Cminus.l 文件读入, 将其内容分割为 C 语言定义段, 正规定义段, 规则段 (返回正规式及其对应 C++ 程序), 用户子程序段。
- 5) FormatRegExp: 根据 FileReader 类提供的正规定义、正规式, 处理输入正规式, 将正规式转化为内码表示运算符的标准后缀表达形式;
- 6) TransTable 类: 存放 NFA 状态转化表。
- 7) NFA 类: 利用 FormatRegExp 提供的后缀表达式, 通过 Thompson's Algorithm 将之转化为对应的 NFA (初始状态标号默认 0, 状态数, 状态转化表, Token 识别映射表)。
- 8) MergedNFA 类: 派生自 NFA, 合并多个构造好的 NFA (初始状态标号默认 0, 状态数, 状态转化表, Token 识别映射表)
- 9) DFA 类: 对 MergedNFA 进行确定化, 并最小化 DFA (初始状态标号, 状态数, 状态转化表, Token 识别映射表)
- 10) CodeGenerator 类: 根据 FileReader, DFA 结果生成相应的词法分析程序。

3.2 详细设计

(以描述数据结构及算法实现为主)

- 1) **FileReader** 类: **Cminus.l** 文件读入, 将其内容分割为 C 语言定义段, 正规定义段, 规则段 (返回正规式及其对应 C++ 程序), 用户子程序段。

函数原型如下:

```
//返回 C 语言定义部分代码
vector<string> getDefPart();
//返回正规定义段
map<string,string> getRegDefPart();
//返回正规表达式
vector<string> getRegularExpression();
//返回正规表达式后对应代码
vector<string> getCode();
//返回用户定义子例程
vector<string> getLastPart();
```

- 2) **FormatRegExp**: 根据 **FileReader** 类提供的正规定义、正规式, 处理输入正规式, 将正规式转化为内码表示运算符的标准后缀表达形式;

函数原型如下:

```
FormatRegExp( const map<string,string> &RDP );
~FormatRegExp();
//将正规表达式中缀转后缀
string postfix( string regExpr );
//预处理: 处理双引号, 大括号, 中括号, 通配符, 转义字符: "{}[]\ " ;
//并将运算符转为内码: 只包含内码表示的 ( ) + * ? | &
string preProcess(string s);
//处理中括号, 将中括号中内容用 | 运算符连接起来, 并用 ( ) 表示
//优先运算
string proBrackets( string s );
//处理转义字符
char escape( char d );
//一元运算符
bool isUniOp(char c);
//二元运算符
bool isBinOp(char c);
```

成员变量:

```
//cminus.l 正规定义
map<string,string> regDefPart;
```

算法描述: 首先用正规定义部分初始化类; 此类提供唯一一个 **public:** 成员函数 **string postfix(string regExpr);** 此函数首先调用预处理 **string preProcess(string s);** 处理并去掉双引号, 大括号, 中括号, 通配符, 转义字

符: "{}[].; 并将运算符转为内码: 只包含内码表示的 (、)、+、*、?、|、连接(&) ; 在中缀转后缀时, 利用堆栈和算符的优先级进行转化。

3) **TransTable** 类: 存放 NFA 状态转化表。

成员变量:

//状态转化表行数 (状态数)

int numRows;

//存储 hash_set<int>指针的二维数组

vector<vector<hash_set<int>*> > tableItems;

此类用作 NFA 状态转化表, 由于每个状态接收同一个字符时可能有不同的转移状态, 所以用一个 Item 是集合的二维数据来存放, 为了空间换时间, 使用 hash_set;

4) **NFA** 类: 利用 **FormatRegExp** 提供的后缀表达式, 通过 **Thompson's Algorithm** 将之转化为对应的 NFA (初始状态标号默认 0, 状态数, 状态转化表, Token 识别映射表)。

主要成员函数:

//构造函数

NFA();

NFA(int _numState);

NFA(string regExpr , const map<string,string> ®DefPart);

//NFA 运算

NFA* or(NFA *m, NFA *n);

NFA* and(NFA *m, NFA *n);

NFA* star(NFA *m);

NFA* oneMore(NFA *m);

NFA* oneNone(NFA *m);

在用正规式构造 NFA 时, 用正规式和正规定义初始化, NFA 调用 **FormatRegExp** 类, 标准化正规表达式, 并中缀转后缀, 操作符用 header 中的内码替换; 然后利用后缀表达式和堆栈构造 NFA。

5) **MergedNFA** 类: 派生自 **NFA**, 合并多个构造好的 NFA (初始状态标号默认 0, 状态数, 状态转化表, Token 识别映射表)

成员函数:

//合并 NFA

MergedNFA (const vector<NFA*> &nfas);

//返回当前 NFA 状态识别的 Token 编号, 低编号具有高优先级

int statePatten(int s);

算法描述: NFA 重新编号, 计算合并 NFA 状态数, 设置识别 TOKEN 的接收态集合, 构造新的状态转化表, 添加生成 NFA 初态到各 NFA 初态的 EPSILON 边, 复制其它转化函数;

6) **DFA** 类: 对 **MergedNFA** 进行确定化, 并最小化 **DFA** (初始状态标号, 状态数, 状态转化表, Token 识别映射表)

函数原型:

```

        //用 NFA 构造 DFA
        DFA( MergedNFA *pnfa);
        //返回当前 DFA 状态识别 Token 编号，低编号具有高优先级
        int statePatten( hash_set<int>* curStateOfDFA );
        //返回集合 T,接收 a 后转化状态集
        hash_set<int>* move(hash_set<int>* T, char a);
        //返回状态 i 的 EPSILON 闭包
        hash_set<int>* epsilonClosure ( int i );
        //返回一个 NFA 状态集合的 EPSILON 闭包
        hash_set<int>* epsilonClosure( hash_set<int>* curSetOfNFASStates );
        //返回 DFA 状态数
        int getNumStates();
        //返回状态转化表
        vector<vector<int >>* getTable();
        //返回 DFA 识别模式表
        vector<int >* getStatePattern();

        //DFA 最小化
        void minimize();
        //根据出边划分子集
        bool partition( vector<set<int>* >* statesOfMinimizedDFA,
map<int,int>& oriStateToMinimizedState);
        //获取 DFA 初始状态
        int getEnterState();

```

算法描述：

NFA 确定化：定义：DFA 闭包、DFA 闭包标号、闭包之间的转化映射关系（表），vector 为 128 维，对应 128 个 ASCII 码；初始化：将 EPSILON 闭包标号，作为 DFA 的状态、添加 DFA 状态、添加闭包间映射关系；当存在新的 DFA 状态时，遍历所有现有 DFA 状态，为每一个 DFA 状态添加转化函数，并把出现的新状态加入 DFA 集合中：把出现的新状态加入 DFA 集合中：求当前转换的 EPSILON 闭包，加入新的 EPSILON 闭包作为 DFA 状态，标号、加入新的 DFA 状态、添加闭包间映射关系。DFA 状态数、初始化状态转化表、初始化识别 TOKEN 表。遍历所有 DFAState，标记当前 DFA 状态所识别的 token，构造状态转化表。

DFA 最小化：计算 DFA 可识别 Token 数，定义：最小化 DFA 状态集合及其编号、原 DFA 状态映射到最小化 DFA 的某个状态；初始化：初始化非终态与终态集合；插入最小化 DFA 初始状态集，并初始化新旧状态映射表；自顶向下划分；构造新状态转化表、新状态与识别 Token 映射表；更新 DFA。

- 7) **CodeGenerator 类：**根据 FileReader，DFA 结果生成相应的词法分析程序。
函数原型：

```

        //打开文件，并包含头文件
        CodeGenerator(string fileName);
        ~CodeGenerator();

```

```
//根据规则，生成驱动程序,默认入口状态为 0
void genDriver( vector<string> rules,int enterState = 0 );
//将二维 vector 转化为二维常量数组
void genTable(vector<vector<int> > t,string);
//将 vector 转化为常量数组
void genVector(vector<int> v,string);
```

算法描述：首先，用 `yySeuLex.cpp` 文件初始化类，打开写入文件；生成一些包含头文件的代码；根据 `Cminus.l` 语言定义段生成 `C` 语言定义段；根据 DFA 状态转化表生成 DFA 状态转化表；根据 DFA 状态 Token 映射表生成 DFA 识别 token 查找表；根据 `Cminus.l` 规则段生成驱动程序；根据 `Cminus.l` 用户子例程段生成用户子例程段；

4 使用说明

4.1 SeuLex 使用说明

1. **cminus.l** 定义规则：

正规定义段可接受定义形式：

- a) alpha [A-Za-z]
- b) digit [0-9]
- c) alphanum [A-Za-z0-9]

规则段中支持多种运算符：

- 连接：省略表示
- 或：| 表示
- 闭包：*表示
- 正闭包：+表示
- 一次或一次以上重复：? 表示
- 一定范围的字符或运算：如[A-Z]表示 A|B|……|Z
- 表示引用正规定义：如{alpha}表示引用[A-Za-z]
- 表示优先级：()
- 运算符内容需作为普通符号使用只需加" "：如“|”解释为字符串“|”，而不表示或运算
- 通配符表示任意字符：.
- 生成的 C++程序文件名为 yylex.cpp

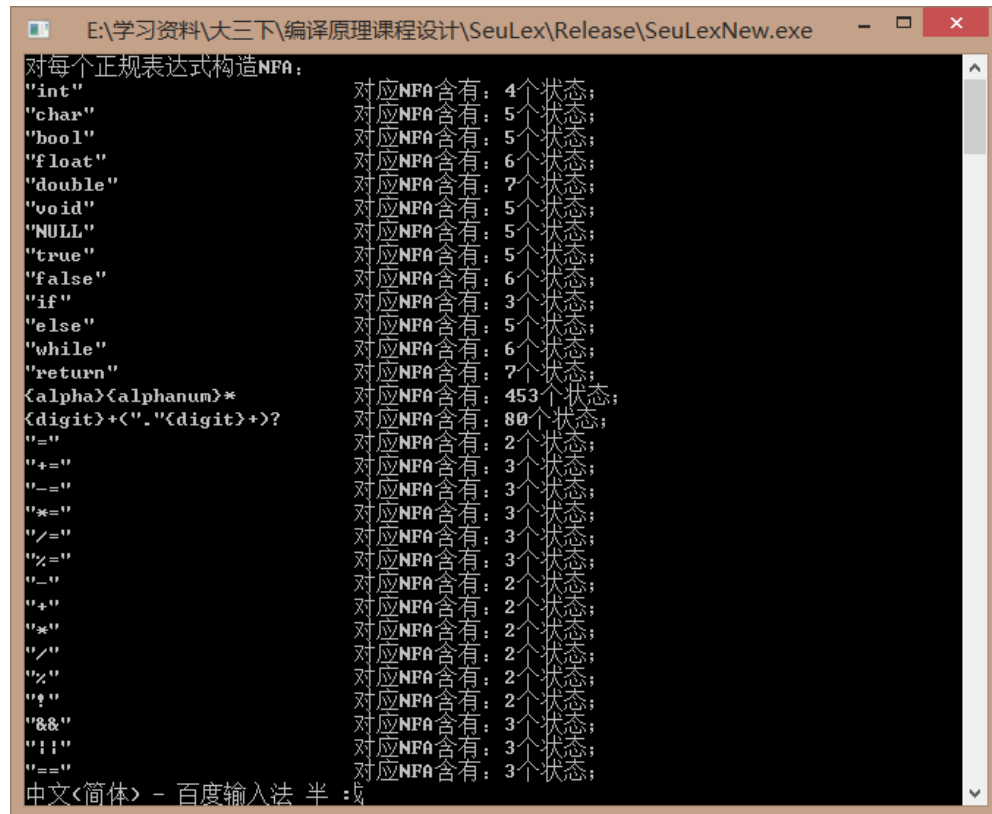
2. 程序输出词法分析文件为：**yylex.cpp**；

3. 生成程序中有提供 **int seuLex(instream & in, ostream &out)**函数原型，返回值为 **cminus.l** 中定义的 **Token** 类型，文件结束时，返回-1；供语法分析程序调用；

4. VS2012 编译通过；

5 测试用例与结果分析

1. 测试用例词法定义为 cminus.l。具体内容见文件中；
2. 测试用例示例程序为 demo.cpp，具体内容见文件中；
3. 生成 yylex，距离代码见文件；
4. Lex 运行结果屏幕截图：



```
E:\学习资料\大三下\编译原理课程设计\SeuLex\Release\SeuLexNew.exe
对每个正规表达式构造NFA:
"int"          对应NFA含有: 4个状态;
"char"         对应NFA含有: 5个状态;
"bool"         对应NFA含有: 5个状态;
"float"        对应NFA含有: 6个状态;
"double"       对应NFA含有: 7个状态;
"void"         对应NFA含有: 5个状态;
"NULL"         对应NFA含有: 5个状态;
"true"         对应NFA含有: 5个状态;
"false"        对应NFA含有: 6个状态;
"if"           对应NFA含有: 3个状态;
"else"         对应NFA含有: 5个状态;
"while"        对应NFA含有: 6个状态;
"return"       对应NFA含有: 7个状态;
<alpha><alphanum>* 对应NFA含有: 453个状态;
<digit>+<"."<digit>+)? 对应NFA含有: 80个状态;
"="           对应NFA含有: 2个状态;
"+="          对应NFA含有: 3个状态;
"-="          对应NFA含有: 3个状态;
"*="          对应NFA含有: 3个状态;
"/="          对应NFA含有: 3个状态;
"%="          对应NFA含有: 3个状态;
"_"           对应NFA含有: 2个状态;
"++"          对应NFA含有: 2个状态;
"*"           对应NFA含有: 2个状态;
"/"           对应NFA含有: 2个状态;
"%"           对应NFA含有: 2个状态;
"&&"          对应NFA含有: 3个状态;
"||"          对应NFA含有: 3个状态;
"=="          对应NFA含有: 3个状态;
```

```
E:\学习资料\大三下\编译原理课程设计\SeuLex_09013430任杰文\...
"%" 对应NFA含有：2个状态；
"/" 对应NFA含有：2个状态；
"%" 对应NFA含有：2个状态；
"%" 对应NFA含有：2个状态；
"&&" 对应NFA含有：3个状态；
"!!" 对应NFA含有：3个状态；
"==" 对应NFA含有：3个状态；
"<" 对应NFA含有：2个状态；
">" 对应NFA含有：2个状态；
"<=" 对应NFA含有：3个状态；
">=" 对应NFA含有：3个状态；
"!=" 对应NFA含有：3个状态；
"." 对应NFA含有：2个状态；
";" 对应NFA含有：2个状态；
"<" 对应NFA含有：2个状态；
">" 对应NFA含有：2个状态；
"<=" 对应NFA含有：2个状态；
">=" 对应NFA含有：2个状态；
"[" 对应NFA含有：2个状态；
"]" 对应NFA含有：2个状态；
"\n" 对应NFA含有：2个状态；
[" \"\t\r"] 对应NFA含有：18个状态；
. 对应NFA含有：506个状态；
合并NFA：
NFA合并完成，当前NFA有：1196个状态
构造NFA：
构造NFA完成，当前DFA有：265个状态
最小化NFA：
最小化NFA完成，当前DFA有：92个状态
生成词法分析程序 yySeuLex.cpp：
词法分析程序 yySeuLex.cpp生成成功！
请按任意键继续。 . . .
中文<简体> - 百度输入法 半：
```

5. 词法分析结果频幕截图：

```
C:\Windows\system32\cmd.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
int      at line 3      ---- Primitive Type
arr      at line 3      ---- Identifier
[        at line 3      ---- Separating Character
]        at line 3      ---- Separating Character
=        at line 3      ---- Assign Operator
{        at line 3      ---- Separating Character
0        at line 3      ---- Number
,        at line 3      ---- Separating Character
5        at line 3      ---- Number
}        at line 3      ---- Separating Character
;        at line 3      ---- Separating Character
int      at line 4      ---- Primitive Type
testInt  at line 4      ---- Identifier
=        at line 4      ---- Assign Operator
10000    at line 4      ---- Number
;        at line 4      ---- Separating Character
double   at line 5      ---- Primitive Type
testDouble at line 5      ---- Identifier
=        at line 5      ---- Assign Operator
1.000000 at line 5      ---- Number
;        at line 5      ---- Separating Character
float    at line 6      ---- Primitive Type
testFloat at line 6      ---- Identifier
=        at line 6      ---- Assign Operator
10000.86 at line 6      ---- Number
;        at line 6      ---- Separating Character
bool     at line 7      ---- Primitive Type
testBool at line 7      ---- Identifier
=        at line 7      ---- Assign Operator
false    at line 7      ---- Internal Constant
;        at line 7      ---- Separating Character
while    at line 8      ---- Flow Controller
{        at line 8      ---- Separating Character
testInt  at line 8      ---- Identifier
<=       at line 8      ---- Relation Operator
testDouble at line 8      ---- Identifier
中文<简体> - 百度输入法 半 : Logical Operator
```



```
C:\Windows\system32\cmd.exe

>      at line 8      ---- Separating Character
<      at line 9      ---- Separating Character
testInt at line 10     ---- Identifier
+=      at line 10     ---- Assign Operator
1       at line 10     ---- Number
;       at line 10     ---- Separating Character
testDouble at line 11  ---- Identifier
-=      at line 11     ---- Assign Operator
testInt at line 11     ---- Identifier
;       at line 11     ---- Separating Character
testBool at line 12    ---- Identifier
=       at line 12     ---- Assign Operator
false   at line 12     ---- Internal Constant
;       at line 12     ---- Separating Character
>      at line 13     ---- Separating Character
testBool at line 14    ---- Identifier
=       at line 14     ---- Assign Operator
true    at line 14     ---- Internal Constant
;       at line 14     ---- Separating Character
if      at line 15     ---- Flow Controller
<      at line 15     ---- Separating Character
testBool at line 15    ---- Identifier
>      at line 15     ---- Separating Character
<      at line 15     ---- Separating Character
testInt at line 16     ---- Identifier
=       at line 16     ---- Assign Operator
10      at line 16     ---- Number
;       at line 16     ---- Separating Character
>      at line 17     ---- Separating Character
else    at line 18     ---- Flow Controller
<      at line 18     ---- Separating Character
testDouble at line 19  ---- Identifier
+=      at line 19     ---- Assign Operator
8888.888 at line 19    ---- Number
;       at line 19     ---- Separating Character
>      at line 20     ---- Separating Character
return  at line 21     ---- Flow Controller
0       at line 21     ---- Number
;       at line 21     ---- Separating Character
>      at line 22     ---- Separating Character
Lexing Ended
中文<简体> - 百度输入法 半 : Logical Operator
```

6 课程设计总结（包括设计的总结和需要改进的内容）

在本次编译原理 **Lex** 和 **Yacc** 项目完成过程中，我们组的成员都对编译原理及程序构造过程有了更深一步的了解。同时，我们也进一步提高了自己编写程序的能力，并且也锻炼了自己与他人合作的能力。

程序还有些不足的地方，比如，**lex** 中为了识别正规式中支持的运算符，在很多部分需要对运算符进行特殊处理，给程序设计的模块合理划分带来一定问题，不利于程序的拓展。

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。