

東南大學

编译原理课程设计

设计报告

组长： 陈一鸣

成员： 吴嘉诚

宋晓伟

东南大学计算机科学与工程系

二〇一八年六月

设计名称	SeuLex&SeuYacc 程序设计		
完成时间	2018.6.10	验收时间	2018.6.15
本组成员情况			
学 号	姓 名	承 担 的 任 务	成 绩
09015422	陈一鸣	正规表达式到 NFA 的转换算法实现 DFA 的最小化算法实现 SeuLex 应用：DFA->分析代码 上下文无关文法对应 LR(1)文法的下推自动机 PDA 的构造	
09015424	吴嘉诚	.l 文件的解析 正规表达式的预处理与转化 正规表达式中缀转后缀的实现 LR(1)文法的下推自动机到相应分析表的构造	
09015429	宋晓伟	多个 NFA 的合并 NFA 状态的合并 DFA 的构建：NFA->DFA Yacc 输入文件（.y 文件）的解析 LR(1)总控程序的构造（查表程序）	

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）

Lex:

//seulex.l

```
%{
#include <iostream>
#include <string>
#include <stdio.h>

FILE * yyin;
FILE * yyout;
%}

number [0-9]
letter [a-zA-Z]
id {letter}+
integer {number}*

%%

void      return "VOID";
int       return "INT";
char      return "CHAR";
double    return "DOUBLE";
switch    return "SWITCH";
break     return "BREAK";
case      return "CASE";
do        return "DO";
while     return "WHILE";
for       return "FOR";
continue  return "CONTINUE";
if        return "IF";
else      return "ELSE";
return    return "RETURN";
"{"       return "{";
"}"       return "}";
"("       return "(";
")"       return ")";
"["       return "[";
"]"       return "]";
"+"       return "+";
"-"       return "-";
"^"       return "^";
"/"       return "/";
```

```

"="          return "=";
"=="        return "==";
"!"          return "!";
";"          return ";";
{id}         return "IDENTIFIER";
{integer}    return "INTEGER";
%%

bool isChaDig(char c)
{
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9'))
        return true;
    else return false;
}

int main()
{
    printf("please input the test program in input.txt\n");
    system("input.txt");
    yyin = fopen("input.txt", "r");
    yyout = fopen("output.txt", "w");
    char yy[20];
    yy[0] = '\n';
    int length=0;
    char be='a';
    char c;
    while (fscanf(yyin,"%c", &c) != EOF)
    {
        if (isChaDig(be) != isChaDig(c) || (!isChaDig(c) && !isChaDig(be) && c != be) || (c ==
'\(' && be == '\(') || (c == '\)' && be == '\)'))
        {
            if (yy[0] != '\n')
            {
                for (int i = 0; i < length; i++)
                    fputc(yy[i], yyout);
                fputc('\t', yyout);
                string result = analysis(yy, length);
                for (int i = 0; i < result.size(); i++)
                    fputc(result[i], yyout);
                fputc('\n', yyout);
            }
            if (c != '\t' && c != ' ')
            {

```

```

        yy[0] = c;
        length=1;
    }
    else
    {
        yy[0] = '\n';
    }
}
else
{
    yy[length]= c;
    length++;
}
be = c;
}
for (int i = 0; i < length; i++)
    fputc(yy[i], yyout);
fputc('\t', yyout);
string result = analysis(yy, length);
for (int i = 0; i < result.size();i++)
    fputc(result[i], yyout);
fputc('\n', yyout);
fclose(yyin);
fclose(yyout);
system("output.txt");
return 0;
}

```

Yacc:

//minic 部分

%union

```

{
    int ival;
}

```

%token <ival> IDENTIFIER FOR WHILE IF ELSE

%token <ival> INT FLOAT LONG SHORT CHAR DOUBLE SIGNED UNSIGNED

%token <ival> EQU LESS GREAT NE

%token <ival> NUM

%%

```

source_file : type function_name '(' parament_list ')' compound_statement
            | type function_name '(' ')' compound_statement
            ;

```

type : INT

| FLOAT

```

| LONG
| SHORT
| CHAR
| DOUBLE
| SIGNED
| UNSIGNED
;

function_name : identifier ;

identifier : IDENTIFIER ;

parament_list : type identifier ',' parament_list
               | type identifier
               ;

compound_statement : '{' statement_list '}'
                   ;

statement_list : statement
               | statement_list statement ;

statement : assign_statement
          | selection_statement
          | iteration_statement
          ;

assign_statement : assign_expression ';'
                 | initial_expression ';'
                 ;

assign_expression : identifier operator identifier
                  ;

operator : '+'
         | '-'
         | '*'
         | '/'
         ;

initial_expression : type identifier
                  | type identifier '=' NUM
                  | type identifier '=' identifier
                  | type identifier '=' assign_expression

```

```

        | identifier '=' assign_expression
        ;

selection_statement : IF '(' relate_expression ')' compound_statement
        ;

relate_expression : identifier relation_operator identifier
        ;

relation_operator : EQU
        | LESS
        | GREAT
        | NE
        ;

iteration_statement : WHILE '(' relate_expression ')' compound_statement
        ;

//calc 部分
%{
%}
%union
{
    int ival;
    float fval;
}
%token <ival> DIGIT
%type <ival>  expr0 expr
%left '+' '-'
%left '*' '/'
%%
expr0 :  expr    {$$=$1;cout<<"最终运算结果为: "<<$1<<endl;}
        ;
expr :  expr '-' expr    {$$=$1-$3;}
        |  expr '+' expr    {$$=$1+$3;}
        |  expr '*' expr    {$$=$1*$3;}
        |  expr '/' expr    {$$=$1/$3;}
        |  DIGIT          {$1=yyval.ival;$=$1;} ;
%%

1.2 编译功能
（所完成的功能项目及对应的程序单元）
Lex:
(1)Lex 输入文件的解析;
void main();/*前半部分:解析 Lex*/

```

(2)正规表达式的解析;

```
int specialSign(char c);/*判断特殊符号*/  
void ChangeRe(string &re);/*去掉[]*/  
void ChangetoRe(string &re);/*将标识符变为 RE*/  
bool isChaDig(char c);/*判断当前字符是不是字母或数字*/  
void addRemovedConcatenationOP(string &re);/*添加.*/  
string infixToPostfix(string expression);/*中缀表达式变后缀表达式*/
```

(3)一个正规表达式到 NFA 的转换算法实现;

```
void produceNfa(const string &re,vector<list<Node>> &tnfa,vector<int> &isTer,int index);/*创建 NFA*/
```

```
void modifyTer(vector<int> &is_t,unsigned int state,int value);/*把最后一个状态设为终态*/
```

(4)多个 NFA 的合并;

```
void joinNfa(vector<list<Node>> &nfa1,const vector<list<Node>> &nfa2);/*合并 NFA*/  
void joinIster(vector<int> &is_t1,const vector<int> &is_t2);/*合并终态*/
```

(5)NFA 的确定化和最小化算法实现;

```
void NfaToDfa();/*NFA 生成 DFA*/  
void Eclosure(set<int> &T);/*求闭包*/  
set<int>move(constset<int> &I,int value);/*求通过 value 达到的 NFA 状态的集合*/  
int dfaisTerminated(set<int> &I);/*返回终态对应动作, 如果是非终态返回-1*/  
void minidfa();/*最小化 DFA*/  
bool checkState(vector<set<int>> &A,int begin,int fir,int sec);  
bool equalState(const list<Node> &fir,const list<Node> &sec,vector<set<int>> &A,int begin);/*判断状态是否相等*/
```

(6)返回状态与返回内容的对应;

```
void genAnalysisCode();/*生成词法分析部分的代码*/
```

(7)Lex 应用;

yylex.cpp

Yacc:

(1)Yacc 输入文件的解析, 对应于程序中的 readinputfile()函数; 定义段-规则段-用户自定义

(2)语义动作程序的加入, 对应于程序中的 readAproducer()函数。实现对\$\$ \$1 等语义求值动作的定义。

(3)上下文无关文法到对应 LR(1)下推自动机的构造, 首先定义 LR(1)项的数据结构表示。采用面向对象方法构造。对应程序中的 extension()函数, 进行状态内部扩展。first()和 getfirstSet()函数用于求预测符。

(4)由 LR(1)下推自动机到对应 LR(1)语法分析表的构造, 在构造项目集合时即产生对应的 LR(1)分析表。对应于 produce()函数实现状态间的转换和由项目集到 LR(1)分析表的构造。

(5)LR(1)总控程序的构造(查表程序), 以下函数分别生成相应的输出代码以及需要的文件:
generateParseCode(); generateSemanticActionCode() 生成相应的语义动作函数;
generateMainCode();

(6)符号表的构建与相应管理程序的编写, 程序会生成符号表文件 Symbol_Table.txt。对应程序中的 generateTableCode();

(7)Yacc 应用:

yyparse.cpp,yypab.h

2. 主要特色

1. 完成了实验所要求的功能, 实现了对于普通 C 语言程序和表达式的词法分析与语法分析程序的构造; ‘
2. 根据词法分析与语法分析的步骤, 我们将 Lex 和 Yacc 程序划分如下模块:
Lex:
 - a) Lex 输入文件的解析;
 - b) 正规表达式的解析;
 - c) 一个正规表达式到 NFA 的转换算法实现;
 - d) 多个 NFA 的合并;
 - e) NFA 的确定化和最小化算法实现;
 - f) SeuLex 应用: DFA 到分析代码的转化;Yacc:
 - a) Yacc 输入文件的解析;
 - b) 上下文无关文法到对应 LR(1)文法的下推自动机的构造;
 - c) LR(1)文法的下推自动机到相应分析表的构造;
 - d) LR(1)总控程序的构造(查表程序)以及 Yacc 的应用;
3. 对于普通的 C 语言程序和单纯的表达式文本, 我们分别采用不同的 Yacc 语法文件(minic.y 和 calc.y)来表示。
4. 我们针对不同的需求选取了合适的数据结构来构造程序:
 - a) **结构体 struct**: 用于构建一个在整体上具有实际意义的结构, 如 Node 代表结点, 由该结点入边上的值和该结点的状态值组成;
 - b) **向量数组 vector**: 用于存储一系列有序或无序的数据, 如 nfalsTer 中存放的为所有的终态结点的状态值;
 - c) **关联容器 map 以及 hash_map**: 提供一对一的多组数据的存储功能, 如 tsymTable 中存储的是每一个终结符所对应的数字编号;
 - d) **双向链表 list**: 用于线性方式存储数据集合, 如 Lex 中对于每一个结点建立一个 list 来存储其可以一步达到的点的集合;
 - e) **集合 set**: 用于存储一系列有序且无重复的数据, 如 leftTable 和 rightTable 中分别有序存放着满足左结合和右结合的符号。

3 概要设计与详细设计

(由总到分地介绍编译器的设计,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)

Lex:

词法分析程序 Lex 主要的处理思路是:

步骤 1: 读入.l 文件进行解析, 存储 RE、Action 以及输入字符集等;

步骤 2: 对 RE 表达式和对应 Action 部分的预处理;

步骤 3: 通过 RE 和 Action 构造唯一的 NFA;

步骤 4: 通过 NFA 以及输入字符集构造 DFA 并将其最小化;

步骤 5: 通过 DFA 和对应终态节点对应的 Action 生成最终的词法分析程序。

下面我们项目在具体实现这些环节所构建的模块:

- a) Main.cpp: 实现整个项目的框架和处理流程, 处理到相应步骤调用相应类的对象进行变量存储和处理; 整合了.l 文件的解析过程(步骤 1)、和预处理(步骤 2)
- b) 函数 produceNfa(): 实现通过每一条 RE 和 Action 构造一个小的 NFA 自动机以及将所有小的自动机合并为唯一的 NFA 自动机。(步骤 3)
- c) 函数 NfaToDfa(): 根据 NFA 构造 DFA。主要运用了子集构造法。(步骤 4)
- d) 函数 genAnalysisCode(): 生成词法分析部分的代码(步骤 5)

Yacc:

语法分析程序 Yacc 主要的处理思路是:

步骤 1: 读入.y 文件进行解析,对于定义段-规则段-用户自定义的规范文件进行解析; 采用了状态跳转的方法来进行控制输入此部分具体所做的工作如下:

1. 读取定义段, 存储语义值类型定义, 存储终结符和非终结符语义值类型, 算符优先级定义, 左右结合性定义。
2. 读取规则段, 完成产生式集合的生成。如有语义动作, 完成语义动作的解析及存储
3. 读取用户自定义段, 直接输出到文件。

步骤 2: 上下文无关文法到对应 LR(1)下推自动机的构造, 首先定义 LR(1)项的数据结构表示。对于已有状态进行内部的扩展, 并且求预测符;

步骤 3: 由 LR(1)下推自动机到对应 LR(1)语法分析表的构造, 在构造项目集合时即产生对应的 LR(1)分析表, 主要操作步骤类似手工求取项目集族并且根据项目集族来产生对应的分析表。

步骤 4: 实现 LR(1)总控程序(查表程序), 符号表的构建与相应管理程序的编写, 程序会生成符号表文件 Symbol_Table.txt。

下面我们项目在具体实现这些环节所构建的模块:

- a) 函数 readinputfile(): 实现整个项目的框架和处理流程, 处理到相应步骤调用相应类的对象进行变量存储和处理; 整合了.y 文件的解析过程(步

骤 1)

- b) 函数 `readAproducer()`: 读取产生式 (步骤 1)
- c) 类 `Item`: 存储产生式, 通过其组成数组来表示项目集族, `int` 类型 `pn` 表示产生式的编号, `int` 类型的 `pos` 表示此项目的移进位置, 另外, `int` 类型的集合 `predictSym` 用于存储此项目的预测符 (步骤 2)
内部函数有:
`getCurrSym()`: 返回下一个将要移进的符号, 如果已经到达最后, 则返回 -1, 表示可归约
`getNextSym()`: 返回再一下要移进的符号, 在进行项目集扩展运算中计算预测符时使用
`getProdN()`: 返回产生式的编号
`nextIsEnd()`: 右部是否到达最后一个符号, 在进行项目集扩展运算时用
- d) 函数 `produce()`: 实现状态间的转换和由项目集到 LR(1)分析表的构造, 存放在二维数组中, 产生的 LR(1)语法分析表 `produceActionTable` 也包括 `goto` 动作, 行数为 LR(1)状态数, 列数为文法非终结符与终结符总数(步骤 3)
- e) 函数 `generateParseCode()`: 生成 yacc 语法分析文件 `yyparse.cpp`
函数 `generateTableCode()`: 生成符号表文件 `Symbol_Table.txt`
函数 `generateSemanticActionCode()`: 生成语义动作代码
函数 `generateSVCode()`: 生成给终结符赋默认值的代码

3.2 详细设计

(以描述数据结构及算法实现为主)

Lex:

数据结构:

```
struct Node//节点
{
    unsigned int value;//连接下一个节点边上的值
    unsigned int state;//下一个节点的状态
};
vector<list<Node>> nfa;//vector[i]表示从节点 i 出发的所有边
vector<list<Node>> dfa;//vector[i]表示从节点 i 出发的所有边
hash_map<string,string> idTore;//存储定义段中标识名到正则式的映射
hash_map<int,int> nfaTer2Action;//存储 NFA 终态到 action 表头对应内容。
hash_map<int,int> dfaTer2Action;//存储 DFA 终态到 action 表头对应内容
vector<string> actionTable;//存储正则式的 action
vector<int> nfaIsTer;//标记 NFA 中哪些是终态
map<set<int>,int> dfanodetable;
vector<int> dfaIsTer;//标记 dfa 中哪些是终结态。
```

算法实现:

- 添加连接操作符 (`.`): 从文件相应位置读取正规表达式, 对正规表达式添加 '`'`操作符号:如果`*`、`)`、字符后面是字符或`(`, 则在此符号后加`'`;
- 中缀转后缀, 参考以下算法
 - 1.遇到操作数: 直接输出 (添加到后缀表达式中)
 - 2.栈为空时, 遇到运算符, 直接入栈

- 3.遇到左括号：将其入栈
 - 4.遇到右括号：执行出栈操作，并将出栈的元素输出，直到弹出栈的是左括号，左括号不输出。
 - 5.遇到其他运算符：加减乘除：弹出所有优先级大于或者等于该运算符的栈顶元素，然后将该运算符入栈
 - 6.最终将栈中的元素依次出栈，输出。
- NFA 构造：使用了栈，当遇到非操作符时就为单个字符构造 nfa 对象并压栈，再判断操作符是单目还是双目，分别进行不同的操作。
 - 根据 NFA 构造 DFA（子集法）：
 1. 令 $I_0 = \epsilon_CLOSURE(S_0)$ 作为 DFA 的初始状态。其中 S_0 为 NFA 初始状态集。
 2. 若 DFA 中的每个状态都经过本步骤处理过.则转步骤 3;否则任选一个未经本步骤处理的 DFA 状态 S_i ，对每一个 $a \in \Sigma$,进行下述处理：
 - (1) 计算 $S_j = S_i a$
 - (2) 若 $S_j \neq \Phi$ ，则令 $f(S_i, a) = S_j$
 若 S_j 不为当前 DFA 的状态，则将其作为 DFA 的一个状态。转步骤 2。
 3. 若 $S' = [S_1, \dots, S_n]$ 是 A 的一个状态,且其中存在一个 S_i 是 A' 的终止状态,则令 S' 为 A 的终止状态。
 - DFA 化简（最小化算法）：
 1. 把状态集 Q 划分为终态集和非终态集: $\Pi = \{\{\text{非终态}\}, \{\text{终态}\}\}$ 。因为终态能识别 e，而非终态不能，所以它们是可区别的；
 2. 按顺序选定集合中的第一个状态 A，并再选取另一个状态 B 与其比较，若是他们不是等价状态，则将 B 移入后面的集合中，若是等价状态则继续验证下一个状态，一遍遍历完后，将会从头再次遍历，直至每一个集合中的状态都是等价状态且不可再分；
 3. 在最终的由各子集组成的状态集合中，在每个子集中任取一个状态做“代表”，而删去子集中其余状态，并把射向其它等价状态的箭弧都改作射向这个做“代表”的状态结中。这样得到的状态转换图所对应的 DFA M' 就是接受 L(M) 的具有最少状态的 DFA；

Yacc:

数据结构:

```
vector<Producer> producerSet;//存储所有产生式的向量，在读入产生式时进行初始化！
map<int,vector<int>> hpSet;//以产生式的左部为关键字，以对应产生式的编号为内容的哈希表。
map<int,int> terminSet;//存储终结符,并且要在其中加入结束符号#
map<int,int> nonterminSet;//存储非终结符,并同时存储一个到 action 表头的对应关系
vector<vector<int>> actionTable;//存储动作表，其中第二维大小必须已知。
map<string,int> tsymTable;//这个表是终结符的串到相应数字编码的映射
map<string,int> ntsymTable;
map<int,int> priorityTable;//优先级表
vector<int> producerPrioTable;//产生式的优先级表，和上面的优先级表联合使用
set<int> leftTable;//左结合表
```

```

set<int> rightTable;//右结合表
map<int,string> symclaTable;//存储语义值类型的表
vector<string> produceActionTable;//存储语义动作的表
//关于类 Item 的定义
class Item
{
public:
    friend bool operator <(const Item &it1,const Item &it2);
    Item(int producerNum,int currentPos);
    int getCurrSym();//返回下一个将要移进的符号，如果已经到达最后，则返回
-1，表示可归约。
    int getNextSym() const;//返回再一下要移进的符号，在进行项目集扩展运算中
计算预测符时使用。
    int getProdN();//返回产生式的编号。
    bool isEnd();//是否右部已经全部移进。:
    bool nextIsEnd() const;//右部是否到达最后一个符号，在进行项目集扩展运算
时用。
    void setpredictSym(const set<int> &predictSymbol);
    const set<int> &getpredictSym() const;
    void move();
private:
    int pn;//存储产生式的编号。
    int pos;//存储此项目的移进位置。
    set<int> predictSym;//存储此项目的预测符，作为公共可以让外部函数 first 直
接使用
};

```

算法实现:

● 项目集族和分析表的构造

1. 构造 0 号产生式 $S' \rightarrow S$ ，并且对其进行扩展，构造出 0 号项目集族。同时初始化分析表，其列数是已知的，为终结符和非终结符个数之和。
2. 对于各个产生式可能的预测符逐一进行分析，并且移动读头（表示移进），否则，则表示需要归约，在表中填入对应的移进还是归约对应的数字，对于移进——归约冲突，通过优先级和运算符的结合性进行裁定。先看优先级的高低，再根据结合性判断。栈内低优先级，栈外高优先级，应该移进；栈内高优先级，栈外低优先级，应该归约；在此之后，符号若左结合，则进行归约。
3. 然后再次进行扩展，观察是否得到新的项目集族，同时为其标号作为状态号，对应于分析表中的行号。
4. 对于新产生的项目集族，需要判断它是否已经出现过了，若未出现，则将其放入待处理队列。

由于每次只处理一个项目集族，针对其可能的预测符，相对应的可以填完分析表中的一行，该算法在队列为空时处理完成。

4 使用说明

4.1 SeuLex 使用说明

1. 将需要分析的语言所对应的.l 文件放入项目目录下;
2. 执行 Lex.exe, 生成 yylex.cpp 文件;
3. 建立新项目 LexTest, 将 yylex.cpp 放入项目中;
4. 编译并执行当前项目, 生成词法分析程序;
5. 输入需要分析的代码, 得到词法分析结果。

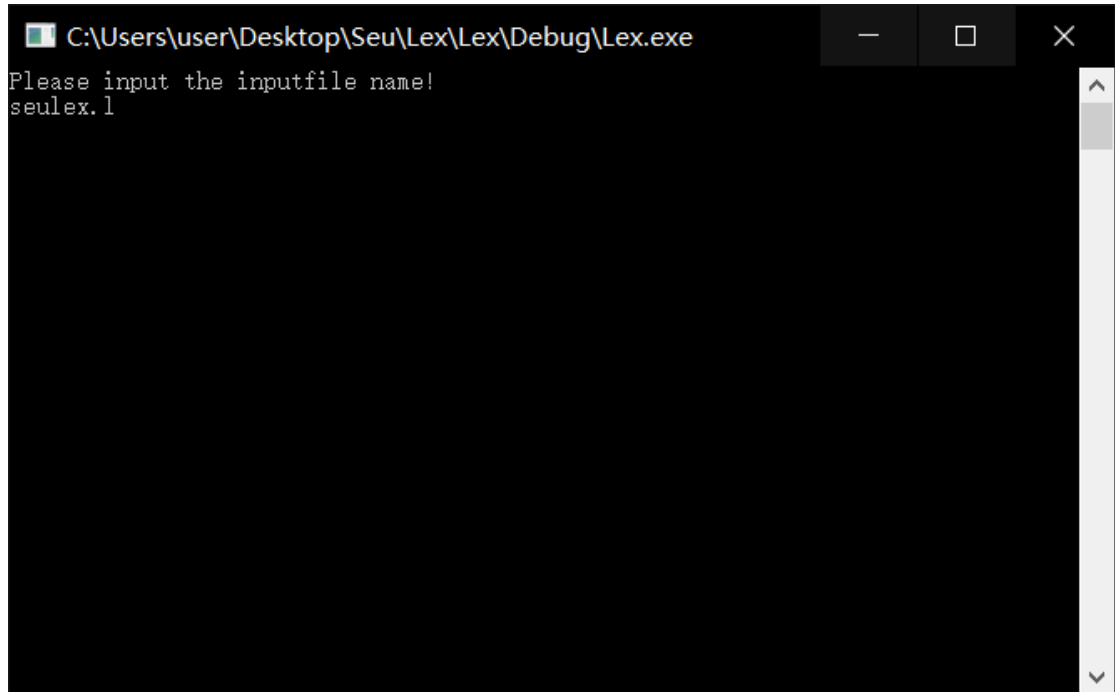
4.2 SeuYacc 使用说明

1. 将需要分析的语言对应的.y 文件放入项目目录下;
2. 执行 Yacc.exe, 生成符号表 Symbol_Table.txt 和语法分析文件 yytab.h、yyparse.cpp;
3. 建立新项目 YaccTest, 将 yylex.cpp 和 yytab.h、yyparse.cpp 以及 yylexmore.cpp 放入项目中;
4. 编译并执行当前项目, 生成语法分析程序;
5. 输入需要分析的代码文件或表达式文件, 得到语法分析结果。

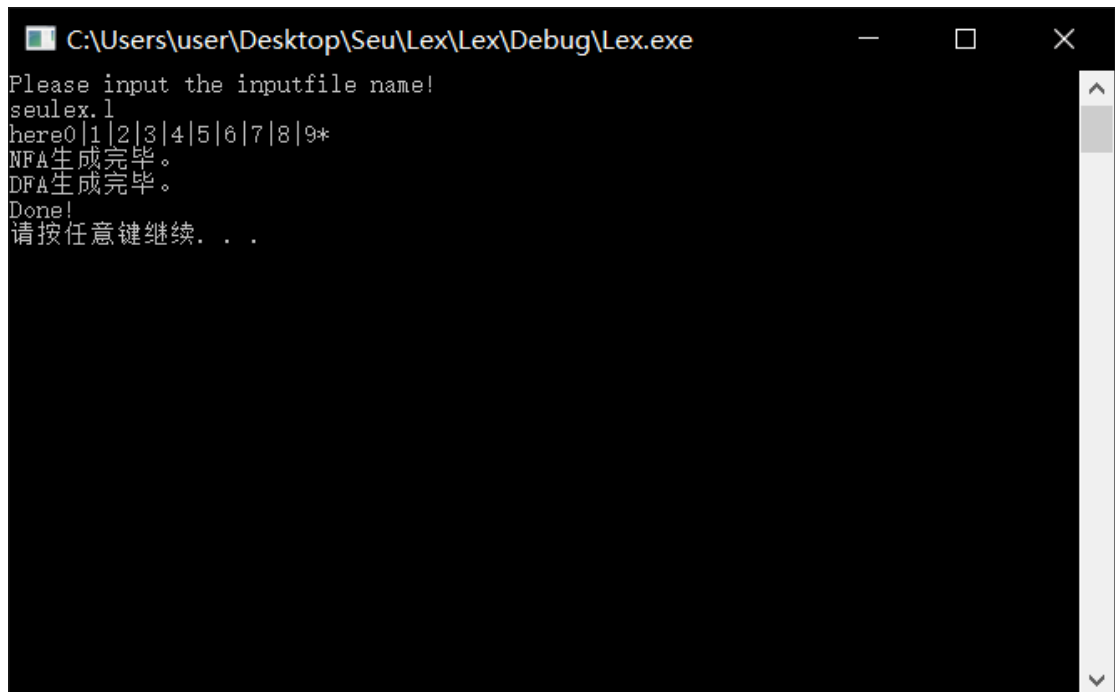
5 测试用例与结果分析

4.1 SeuLex 使用说明

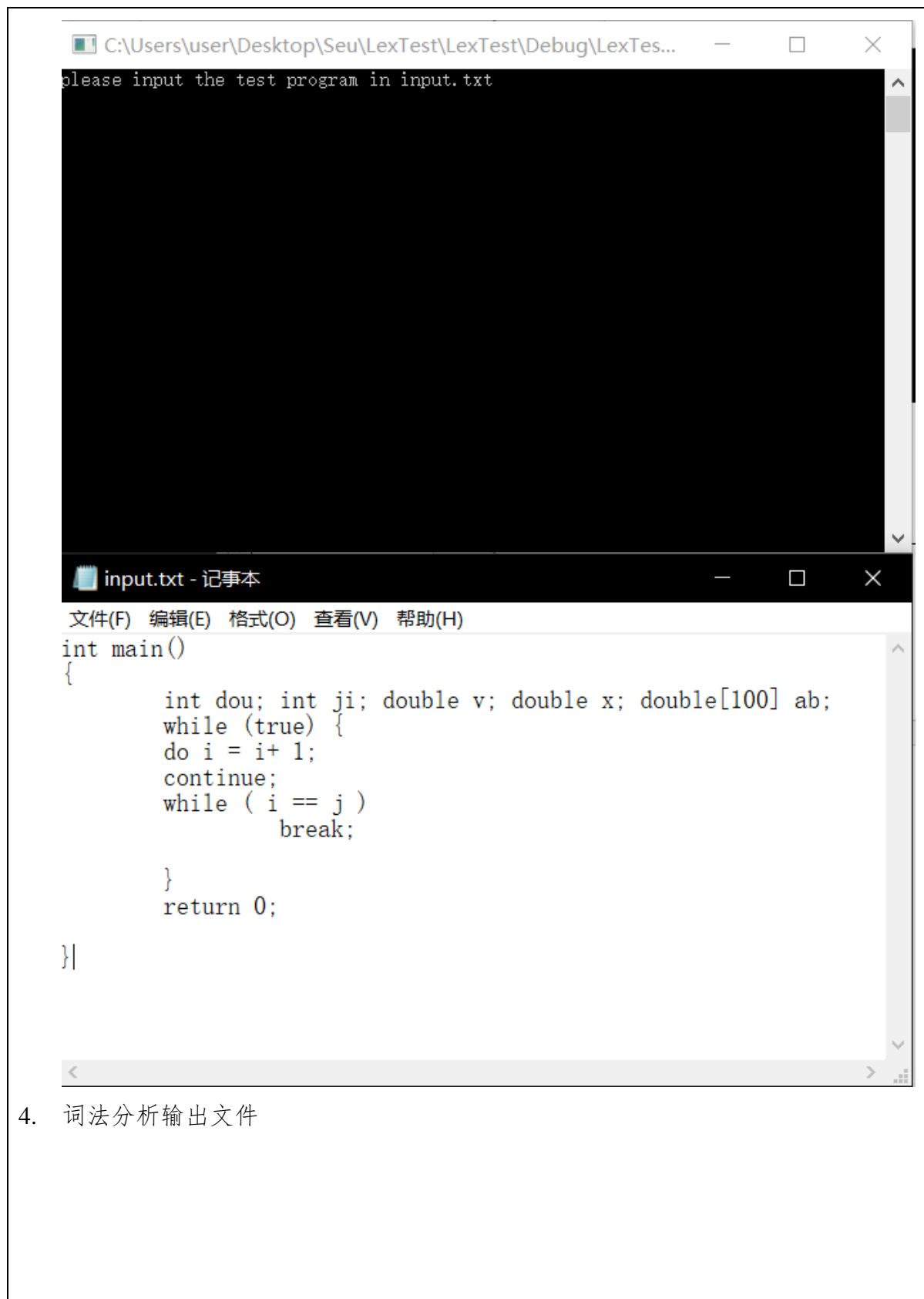
1. 输入.l 文件

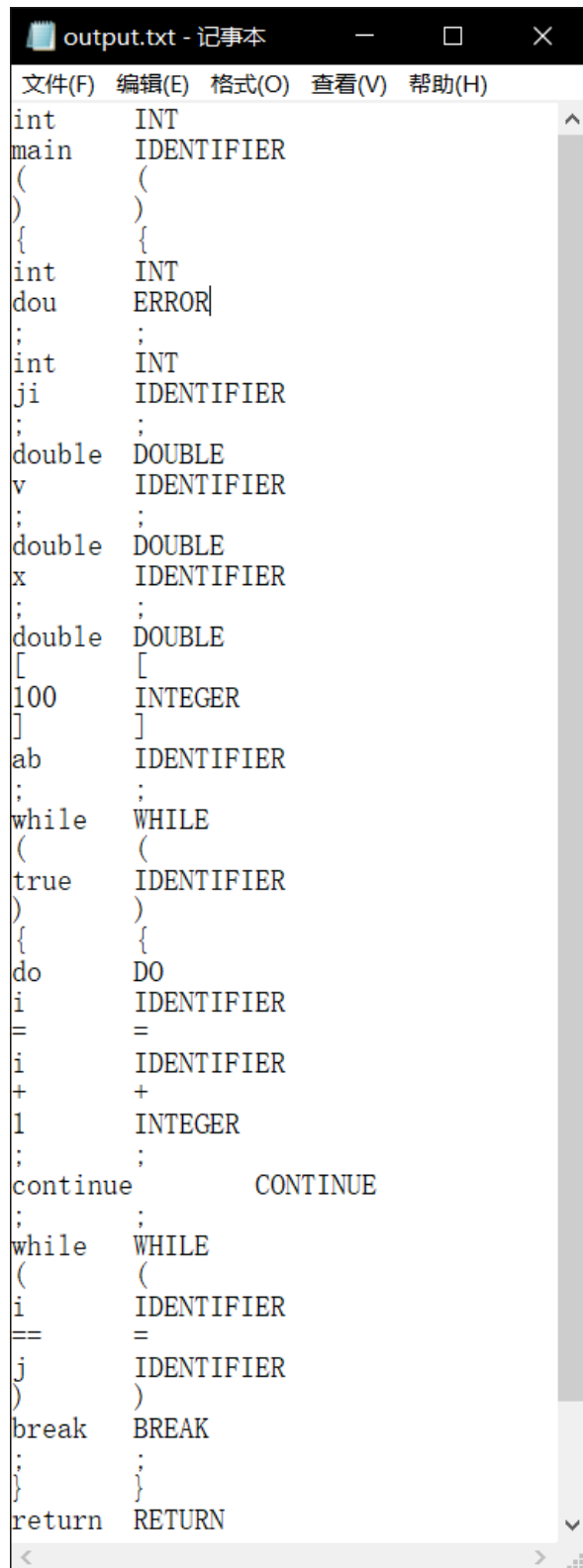


2. 生成完毕



3. 词法分析的输入文件

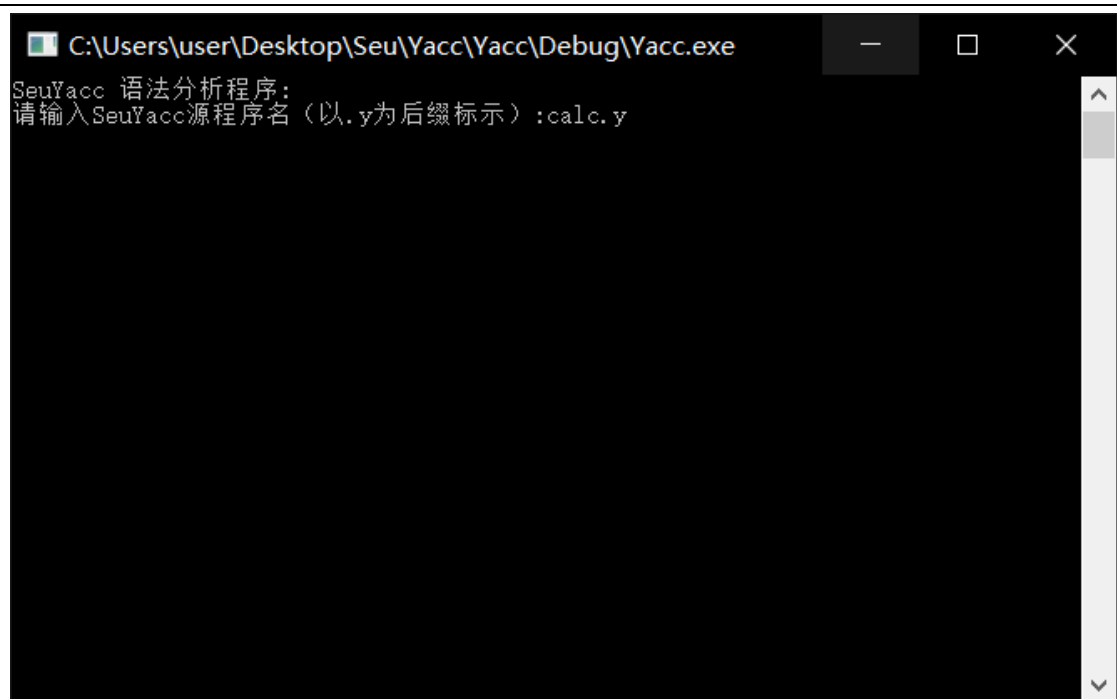




```
int      INT
main     IDENTIFIER
(        (
)        )
{        {
int      INT
dou      ERROR
;        ;
int      INT
ji       IDENTIFIER
;        ;
double   DOUBLE
v        IDENTIFIER
;        ;
double   DOUBLE
x        IDENTIFIER
;        ;
double   DOUBLE
[        [
100      INTEGER
]        ]
ab       IDENTIFIER
;        ;
while    WHILE
(        (
true     IDENTIFIER
)        )
{        {
do       DO
i        IDENTIFIER
=        =
i        IDENTIFIER
+        +
1        INTEGER
;        ;
continue          CONTINUE
;        ;
while    WHILE
(        (
i        IDENTIFIER
==       =
j        IDENTIFIER
)        )
break    BREAK
;        ;
}        }
return   RETURN
```

4.2 SeuYacc 使用说明

1. .y 文件的输入 (以 calc.y 为例)



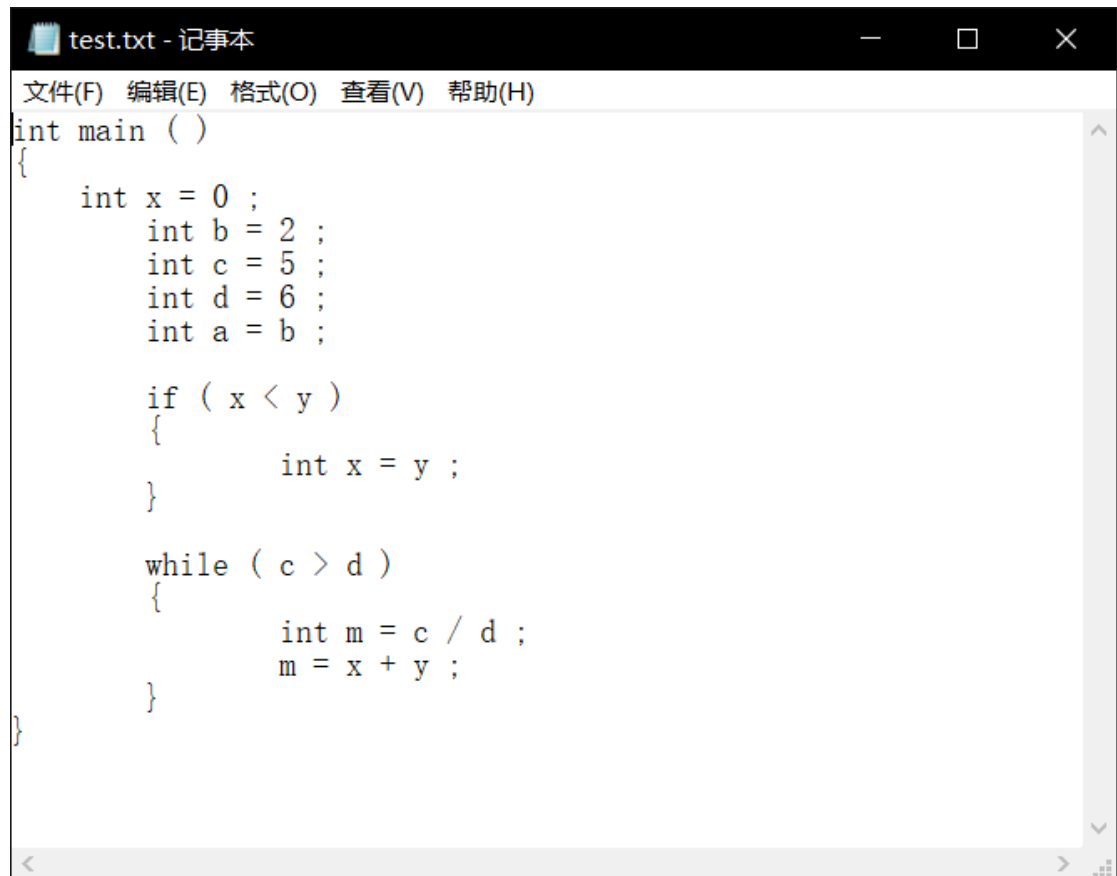
2. 生成的项目集族

```
C:\Users\user\Desktop\SeuYacc\Yacc\Debug\Yacc.exe
SeuYacc 语法分析程序:
请输入SeuYacc源程序名 (以 .y为后缀标示): calc.y

-----ItemSet -----
Item
-----
PID:0      LEFT:34999      RIGHT:35000
predict:34999
-----
PID:1      LEFT:35000      RIGHT:35001
predict:34999
-----
PID:2      LEFT:35001      RIGHT:35001  30002  35001
predict:30001
-----
PID:2      LEFT:35001      RIGHT:35001  30002  35001
predict:30002
-----
PID:2      LEFT:35001      RIGHT:35001  30002  35001
predict:30003
-----
PID:2      LEFT:35001      RIGHT:35001  30002  35001
predict:30004
-----
PID:2      LEFT:35001      RIGHT:35001  30002  35001
predict:34999
-----
PID:3      LEFT:35001      RIGHT:35001  30001  35001
predict:30001
-----
PID:3      LEFT:35001      RIGHT:35001  30001  35001
predict:30002
-----
PID:3      LEFT:35001      RIGHT:35001  30001  35001
predict:30003
-----
PID:3      LEFT:35001      RIGHT:35001  30001  35001
predict:30004
-----
PID:3      LEFT:35001      RIGHT:35001  30001  35001
predict:34999
-----
PID:4      LEFT:35001      RIGHT:35001  30003  35001
predict:30001
-----
PID:4      LEFT:35001      RIGHT:35001  30003  35001
predict:30002
-----
PID:4      LEFT:35001      RIGHT:35001  30003  35001
predict:30003
-----
PID:4      LEFT:35001      RIGHT:35001  30003  35001
predict:30004
-----
PID:4      LEFT:35001      RIGHT:35001  30003  35001
predict:34999
-----
PID:5      LEFT:35001      RIGHT:35001  30004  35001
predict:30001
-----
PID:5      LEFT:35001      RIGHT:35001  30004  35001
predict:30002
-----
PID:5      LEFT:35001      RIGHT:35001  30004  35001
predict:30003
-----
PID:5      LEFT:35001      RIGHT:35001  30004  35001
predict:30004
-----
PID:5      LEFT:35001      RIGHT:35001  30004  35001
predict:34999
-----
PID:6      LEFT:35001      RIGHT:30000
predict:30001
-----
PID:6      LEFT:35001      RIGHT:30000
predict:30002
-----
PID:6      LEFT:35001      RIGHT:30000
predict:30003
-----
PID:6      LEFT:35001      RIGHT:30000
predict:30004
-----
PID:6      LEFT:35001      RIGHT:30000
predict:34999
-----

state = 0
state = 1
state = 2
state = 3
state = 4
state = 5
state = 6
state = 7
state = 8
state = 9
state = 10
state = 11
成功生成! 可以在C编译器里进行语法解析了...
请按任意键继续...
```

3. 普通 C 语言程序的测试



```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
int main ( )
{
    int x = 0 ;
    int b = 2 ;
    int c = 5 ;
    int d = 6 ;
    int a = b ;

    if ( x < y )
    {
        int x = y ;
    }

    while ( c > d )
    {
        int m = c / d ;
        m = x + y ;
    }
}
```

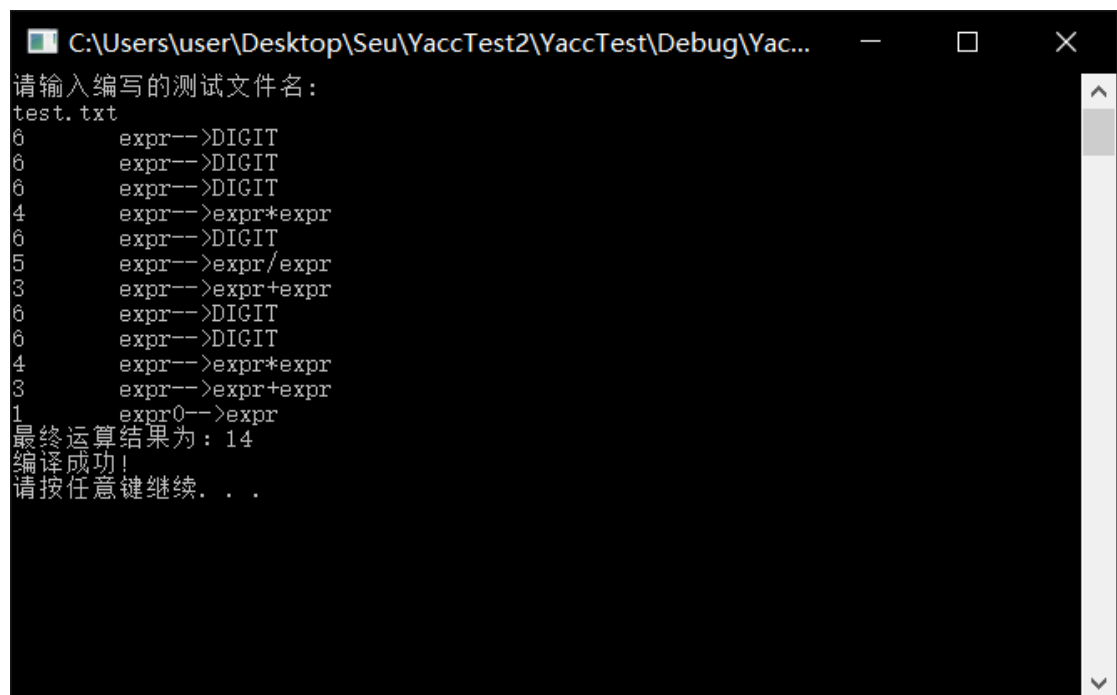
```
C:\Users\user\Desktop\Seu\YaccTest\YaccTest\Debug\YaccTest.exe
请输入编写的测试文件名:
test.txt
3      type-->INT
12     identifier-->IDENTIFIER
11     function_name-->identifier
3      type-->INT
12     identifier-->IDENTIFIER
29     initial_expression-->typeidentifier=NUM
22     assign_statement-->initial_expression;
18     statement-->assign_statement
16     statement_list-->statement
3      type-->INT
12     identifier-->IDENTIFIER
29     initial_expression-->typeidentifier=NUM
22     assign_statement-->initial_expression;
18     statement-->assign_statement
17     statement_list-->statement_liststatement
3      type-->INT
12     identifier-->IDENTIFIER
29     initial_expression-->typeidentifier=NUM
22     assign_statement-->initial_expression;
18     statement-->assign_statement
17     statement_list-->statement_liststatement
3      type-->INT
12     identifier-->IDENTIFIER
29     initial_expression-->typeidentifier=NUM
22     assign_statement-->initial_expression;
18     statement-->assign_statement
17     statement_list-->statement_liststatement
3      type-->INT
12     identifier-->IDENTIFIER
12     identifier-->IDENTIFIER
30     initial_expression-->typeidentifier=identifier
22     assign_statement-->initial_expression;
18     statement-->assign_statement
17     statement_list-->statement_liststatement
12     identifier-->IDENTIFIER
36     relation_operator-->LESS
12     identifier-->IDENTIFIER
34     relate_expression-->identifierrelation_operatoridentifier
3      type-->INT
12     identifier-->IDENTIFIER
12     identifier-->IDENTIFIER
30     initial_expression-->typeidentifier=identifier
22     assign_statement-->initial_expression;
18     statement-->assign_statement
16     statement_list-->statement
15     compound_statement-->[statement_list]
33     selection_statement-->IF(relate_expression) compound_statement
19     statement-->selection_statement
17     statement_list-->statement_liststatement
12     identifier-->IDENTIFIER
37     relation_operator-->GREAT
12     identifier-->IDENTIFIER
34     relate_expression-->identifierrelation_operatoridentifier
3      type-->INT
12     identifier-->IDENTIFIER
12     identifier-->IDENTIFIER
27     operator-->/
12     identifier-->IDENTIFIER
23     assign_expression-->identifieroperatoridentifier
31     initial_expression-->typeidentifier=assign_expression
22     assign_statement-->initial_expression;
18     statement-->assign_statement
16     statement_list-->statement
12     identifier-->IDENTIFIER
12     identifier-->IDENTIFIER
12     identifier-->IDENTIFIER
24     operator-->+
12     identifier-->IDENTIFIER
23     assign_expression-->identifieroperatoridentifier
32     initial_expression-->identifier=assign_expression
22     assign_statement-->initial_expression;
18     statement-->assign_statement
17     statement_list-->statement_liststatement
15     compound_statement-->[statement_list]
39     iteration_statement-->WHILE(relate_expression) compound_statement
20     statement-->iteration_statement
17     statement_list-->statement_liststatement
15     compound_statement-->[statement_list]
2     source_file-->typefunction_name() compound_statement
编译成功!
请按任意键继续. . .
```

此处所显示的是所有规约的时候采用的产生式，最后编译成功即意味着状态表进入 ACCEPT 状态。

4. 表达式的测试



```
test.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
2+3*4/6+5*2
```



```
C:\Users\user\Desktop\Seu\YaccTest2\YaccTest\Debug\Yac...
请输入编写的测试文件名:
test.txt
6      expr-->DIGIT
6      expr-->DIGIT
6      expr-->DIGIT
4      expr-->expr*expr
6      expr-->DIGIT
5      expr-->expr/expr
3      expr-->expr+expr
6      expr-->DIGIT
6      expr-->DIGIT
4      expr-->expr*expr
3      expr-->expr+expr
1      expr0-->expr
最终运算结果为: 14
编译成功!
请按任意键继续. . .
```

此处所显示的是所有规约的时候采用的产生式,最后编译成功即意味着状态表进入 ACCEPT 状态,且最终会输出该表达式所得到的结果。

6 课程设计总结（包括设计的总结和需要改进的内容）

这一次的课程设计是建立在上学期的课程实验的基础之上的，但这一次的课程设计主要是针对 **Lex** 和 **Yacc** 应用程序的编写，即所要完成的是通过代码自动生成相应的词法分析程序和语法分析程序，因此在初期我们有些无从下手。在我们仔细梳理了词法分析和语法分析的步骤后，我们借助课本上的一些处理方法，完成了对该课程设计初步的步骤和思路的设计。同时，我们查阅了相关的 **C++** 资料，学习和梳理了一些 **STL** 库内的类的使用方法，构建了适合该课程设计的数据结构，初步完成了课程设计的内容。

在随后的调试过程中，我们对之前所完成的设计和代码重新进行了一次梳理。在测试的过程中，我们发现并解决了一些问题：

1. 最初编写代码的时候，我们只是简单的按照个人的分工，将整个程序划分为几个主要的函数，这使得我们在后来调试的过程中，定位错误的难度大大提高。因此我们将每个人的功能再次进行划分，分成若干个较小的功能，并将它们封装成函数，这大大提高了代码的可读性。
2. 在 **Lex** 部分中，由于我们所构建的 **NFA** 在存储时采用的结构是对每个结点建立一个 **list** 链表存储该结点可以达到的结点，这就会导致由于最终的一个结点并没有出边，因此其对应的链表为空的情况。这就使得我们定义的 **nfaIsTer** 的 **size** 会比 **NFA** 的 **size** 大一个，从而导致在合并 **NFA** 的时候 **size** 数量不匹配。为解决这个问题，我们给每个 **NFA** 的 **size** 自动+1，即相当于给每个 **NFA** 生成一个空的终结点，从而避免了不匹配的问题。
3. 在进行词法分析的时候，我们所采用的 **analysis** 函数的返回值是对应输入的属性值。而 **string** 类型的返回值在进行语法分析的时候并没有 **int** 类型来的方便，因此我们在导入上面生成的 **yylex.cpp** 的同时还会同时导入一个从 **string** 到 **int** 的对应函数用于转换属性和对应的返回值。
4. 在 **Yacc** 部分中，在构建下推自动机的时候，我们考虑通过项目集族的构建来实现，从而实现到分析表的转化。在构建项目集族的过程中，最困难的一步就是如何完善当前的项目集。对于这一点，我们考虑对每个产生式的左部建立表，并将其右部可能达到的产生式均加入该表，实现查找的便捷化。
5. 在显示语法分析的结果的时候，为表现整个程序的构建过程，我们考虑输出在每一次规约的时候所采用的产生式。由于我们之前为了方便已将所有的产生式数字化，因此我们考虑再建立一个产生式的对应表，从而使得在语法分析的结果中输出产生式成为可能。

经过这一次的课程设计，我们不仅复习了算法和数据结构的相关知识，对编译程序的构造过程也有了进一步的了解，同时也提高了自身程序编写的能力。由于这次课程设计的代码量较大，因此需要一个组的同学合作完成，这也是对我们分工合作能力的一次考验和提升。相信这次课程设计一定会对我们之后的学习产生极其深远的影响。

7 教师评语

签名：_____

附录 代码清单

SeuLex 包含如下代码：

1. Lex.cpp
2. seulex.l
3. 生成的 yylex.cpp

SeuYacc 包含如下代码：

1. Yacc.cpp
2. calc.y
3. minic.y
4. yylexmore.cpp
5. 生成的 producer.txt
6. 生成的 Symbol_Table.txt
7. 生成的 yytab.h
8. 生成的 yyparse.cpp

光盘粘贴处

请将光盘装入纸袋中粘贴于此。

