项目测试报告

任务三

一、项目任务一测试

1、准备工作

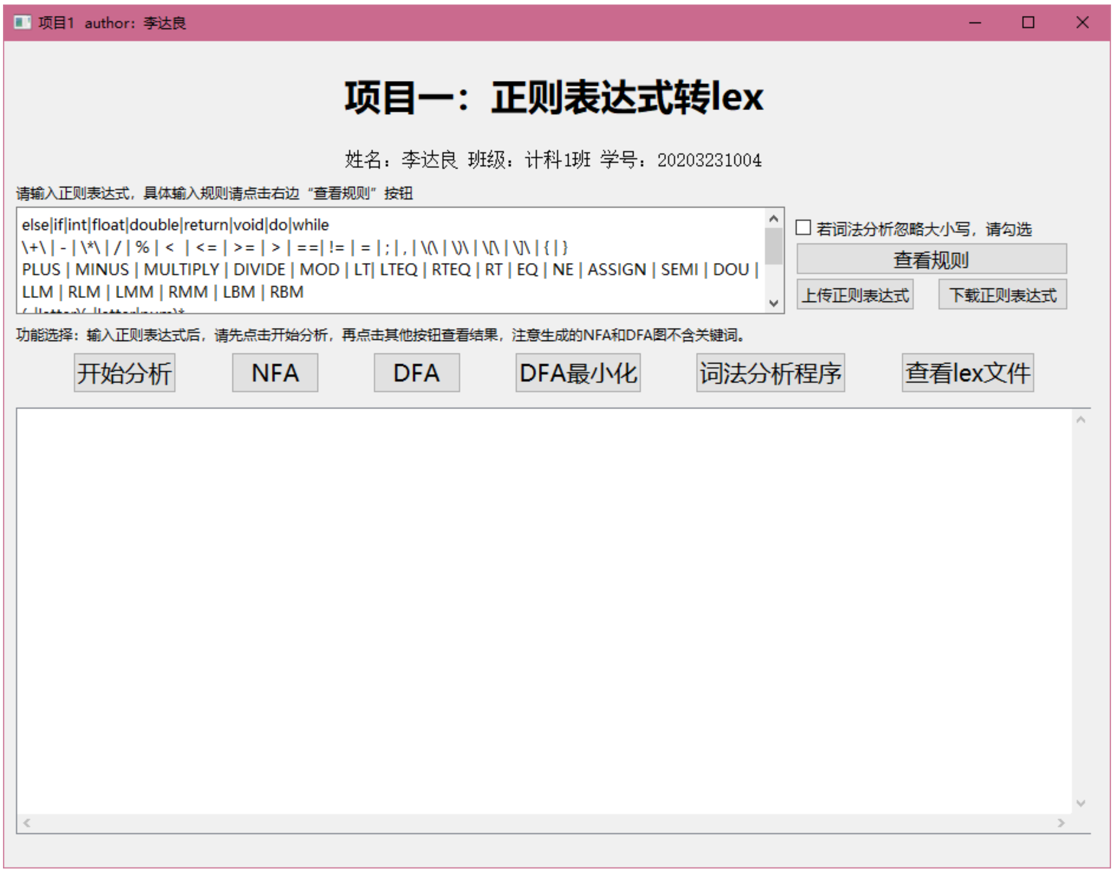
minic 的正则表达式：

```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\
| \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ |
NE | ASSIGN | SEMI | DOU | LLM | RLM | LMM | RMM | LBM | RBM
(_|letter)(_|letter|num)*
num num* (.num num *)?
//~\n
```

sample.tny 文件：

```
//Sample program
//In MiniC language-computes factorial
//}
int x; // an integer
int fact;
double y;
int arr[100];
void test(int a, int b)
{
  a = 1;
  b = 2;
  return ;
}
void main(void)
{
  x = 2;
  y = 3;
  if ( x > 1 )  //don't compute if x <= 0
      do
          fact = fact * x;
      while (fact >= 1);
  else
    x = 1;
  return ; //return void
}
```

输入正则表达式：



项目一：正则表达式转lex

姓名：李达良 班级：计科1班 学号：20203231004

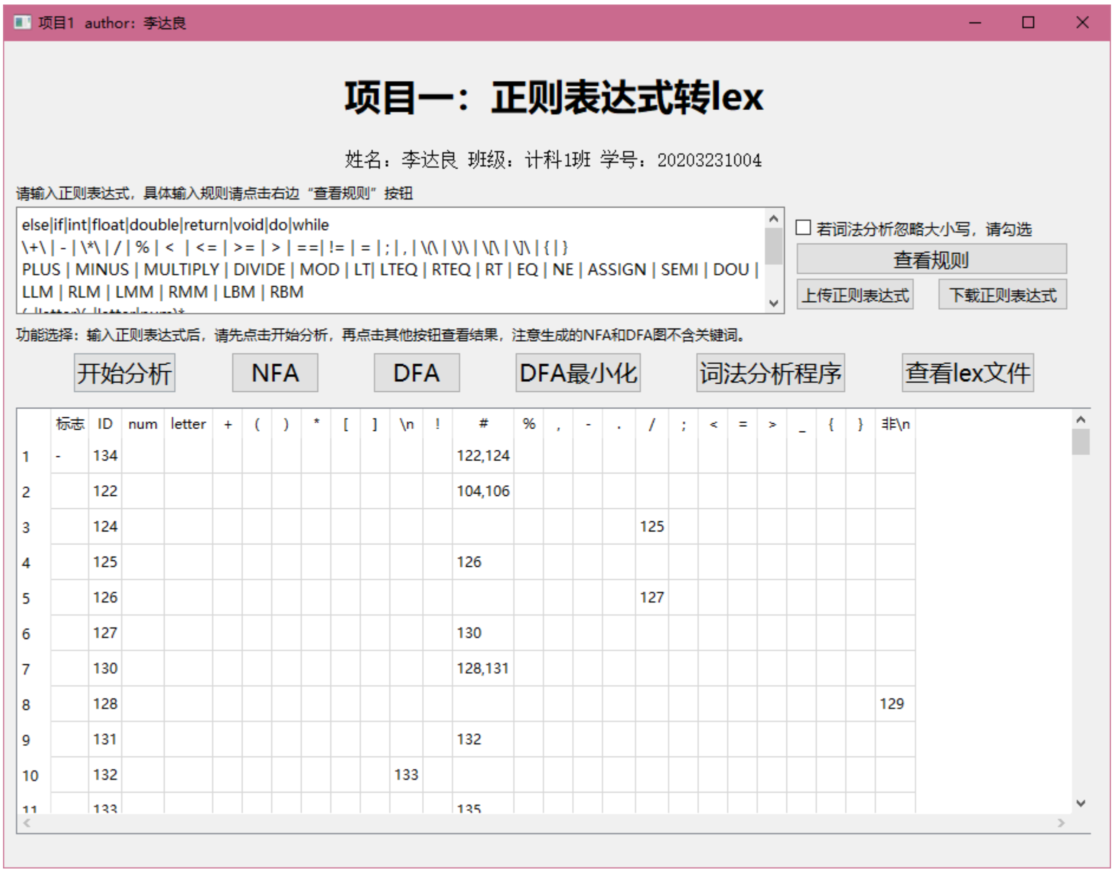
请输入正则表达式，具体输入规则请点击右边“查看规则”按钮

```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU |
LLM | RLM | LMM | RMM | LBM | RBM
```

□ 若词法分析忽略大小写，请勾选

功能选择：输入正则表达式后，请先点击开始分析，再点击其他按钮查看结果，注意生成的NFA和DFA图不含关键词。

2、NFA



项目一：正则表达式转lex

姓名：李达良 班级：计科1班 学号：20203231004

请输入正则表达式，具体输入规则请点击右边“查看规则”按钮

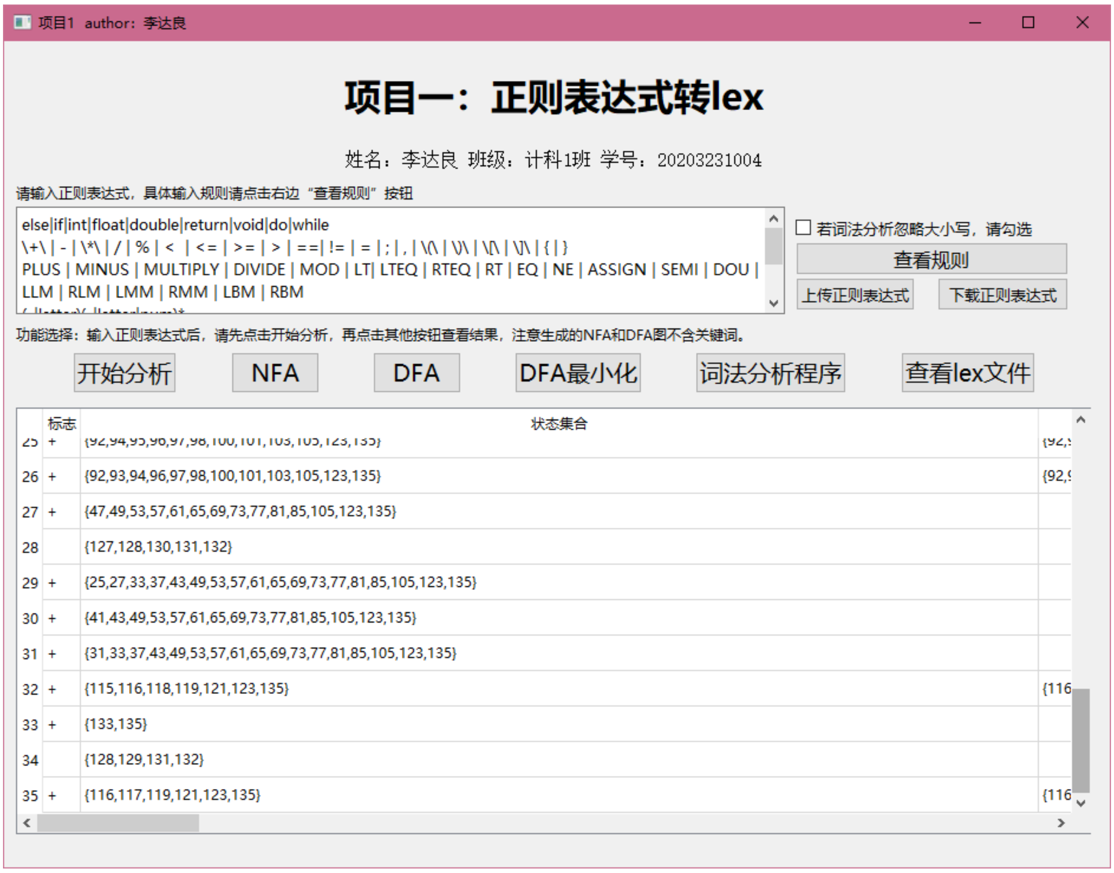
```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU |
LLM | RLM | LMM | RMM | LBM | RBM
```

□ 若词法分析忽略大小写，请勾选

功能选择：输入正则表达式后，请先点击开始分析，再点击其他按钮查看结果，注意生成的NFA和DFA图不含关键词。

	标志	ID	num	letter	+	(	)	*	[	]	\n	!	#	%	,	-	.	/	;	<	=	>	_	{	}	非\n
1	-	134											122,124													
2		122											104,106													
3		124																125								
4		125											126													
5		126																127								
6		127											130													
7		130											128,131													
8		128																								129
9		131											132													
10		132									133															
11		133											135													

3、DFA 图



项目一：正则表达式转lex

姓名：李达良 班级：计科1班 学号：20203231004

请输入正则表达式，具体输入规则请点击右边“查看规则”按钮

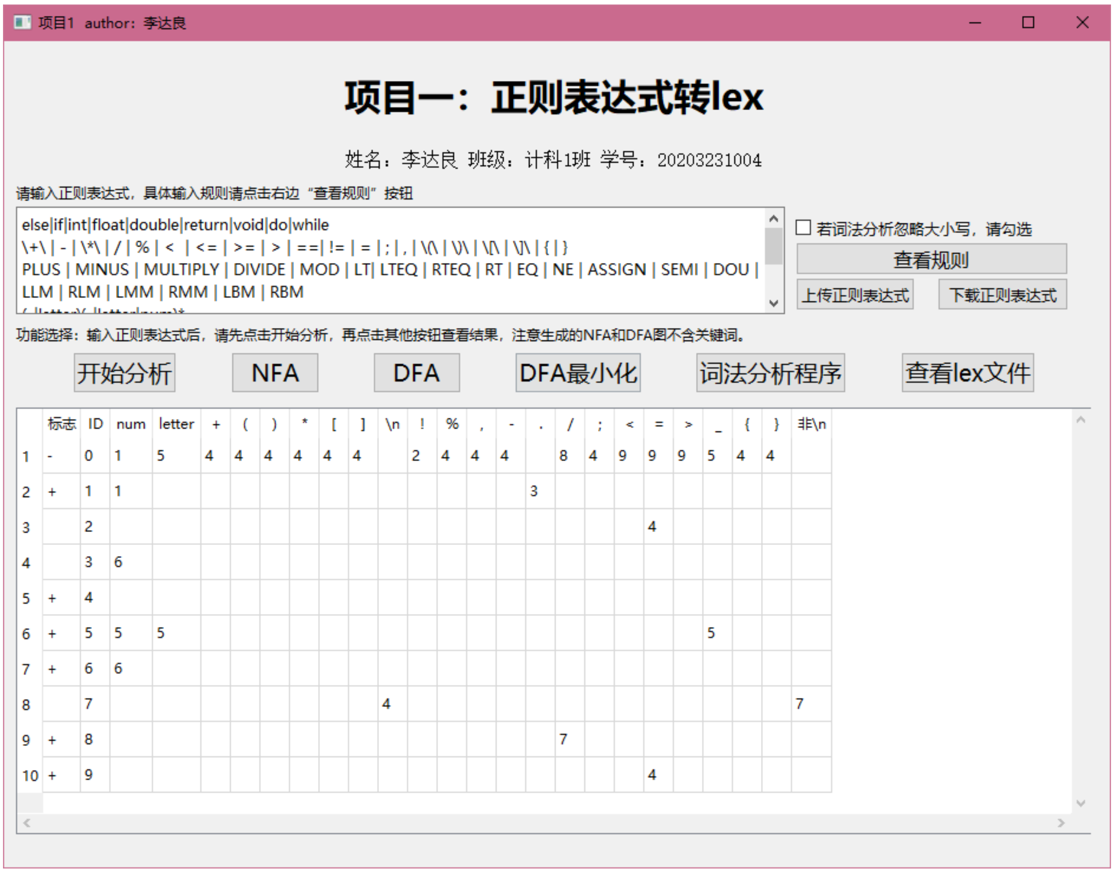
```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU |
LLM | RLM | LMM | RMM | LBM | RBM
```

□ 若词法分析忽略大小写，请勾选

功能选择：输入正则表达式后，请先点击开始分析，再点击其他按钮查看结果，注意生成的NFA和DFA图不含关键词。

	标志	状态集合	
25	+	{92,94,95,96,97,98,100,101,103,105,123,135}	{92,9
26	+	{92,93,94,96,97,98,100,101,103,105,123,135}	{92,9
27	+	{47,49,53,57,61,65,69,73,77,81,85,105,123,135}	
28		{127,128,130,131,132}	
29	+	{25,27,33,37,43,49,53,57,61,65,69,73,77,81,85,105,123,135}	
30	+	{41,43,49,53,57,61,65,69,73,77,81,85,105,123,135}	
31	+	{31,33,37,43,49,53,57,61,65,69,73,77,81,85,105,123,135}	
32	+	{115,116,118,119,121,123,135}	{116
33	+	{133,135}	
34		{128,129,131,132}	
35	+	{116,117,119,121,123,135}	{116

4、DFA 图最小化

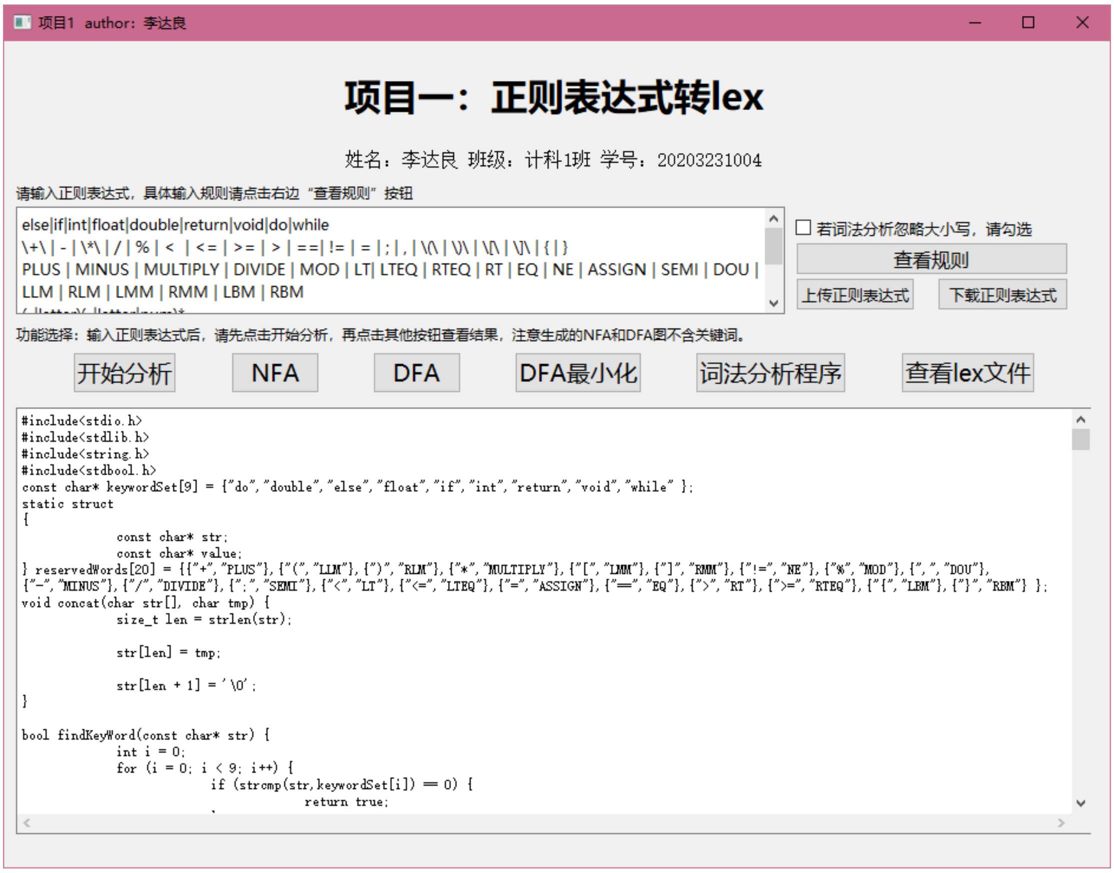

项目1 author：李达良
项目一：正则表达式转lex
姓名：李达良 班级：计科1班 学号：20203231004
请输入正则表达式，具体输入规则请点击右边“查看规则”按钮

```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU |
LLM | RLM | LMM | RMM | LBM | RBM
```

□ 若词法分析忽略大小写，请勾选
查看规则
上传正则表达式
下载正则表达式
功能选择：输入正则表达式后，请先点击开始分析，再点击其他按钮查看结果，注意生成的NFA和DFA图不含关键词。
开始分析
NFA
DFA
DFA最小化
词法分析程序
查看lex文件

	标志	ID	num	letter	+	(	)	*	[	]	\n	!	%	,	-	.	/	;	<	=	>	_	{	}	非\n
1	-	0	1	5	4	4	4	4	4	4		2	4	4	4		8	4	9	9	9	5	4	4	
2	+	1	1													3									
3		2																		4					
4		3	6																						
5	+	4																							
6	+	5	5	5																		5			
7	+	6	6																						
8		7									4														7
9	+	8															7								
10	+	9																		4					

5、生成词法程序


项目1 author：李达良
项目一：正则表达式转lex
姓名：李达良 班级：计科1班 学号：20203231004
请输入正则表达式，具体输入规则请点击右边“查看规则”按钮

```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU |
LLM | RLM | LMM | RMM | LBM | RBM
```

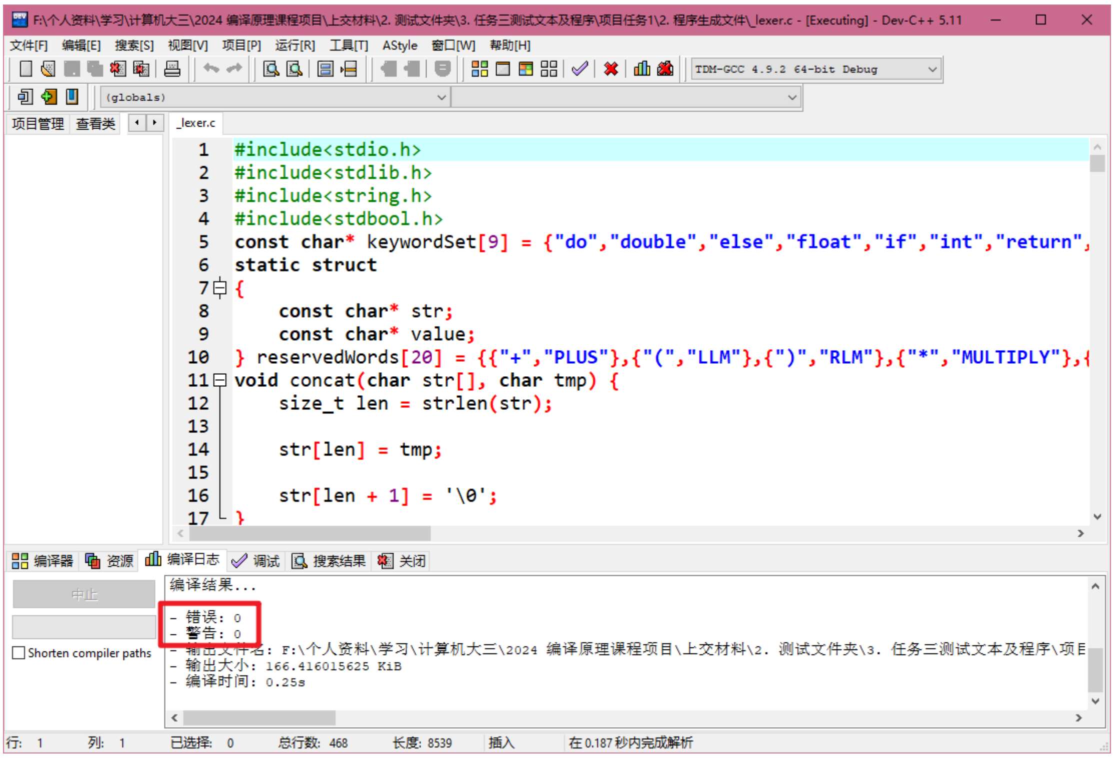
□ 若词法分析忽略大小写，请勾选
查看规则
上传正则表达式
下载正则表达式
功能选择：输入正则表达式后，请先点击开始分析，再点击其他按钮查看结果，注意生成的NFA和DFA图不含关键词。
开始分析
NFA
DFA
DFA最小化
词法分析程序
查看lex文件

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
const char* keywordSet[9] = {"do","double","else","float","if","int","return","void","while" };
static struct
{
            const char* str;
            const char* value;
} reservedWords[20] = {{"+","PLUS"},{"(","LLM"},{")","RLM"},{"*","MULTIPLY"},{"[","LMM"},{"]","RMM"},{"!=","NE"},{"%","MOD"},{",","DOU"},
{"-","MINUS"},{"/","DIVIDE"},{";","SEMI"},{"<","LT"},{"<=","LTEQ"},{"=","ASSIGN"},{"==","EQ"},{">","RT"},{">=","RTEQ"},{"{","LBM"},{"}","RBM"} };
void concat(char str[], char tmp) {
            size_t len = strlen(str);

            str[len] = tmp;

            str[len + 1] = '\0';
}

bool findKeyWord(const char* str) {
            int i = 0;
            for (i = 0; i < 9; i++) {
                        if (strcmp(str,keywordSet[i]) == 0) {
                                    return true;
```

代码太长，具体可查看：
2. 测试文件夹\1. 任务一测试文本及程序\2. 程序生成文件中的_lexer.c 文件

6、编译_lexer.c 文件并运行

注意：sample.tny 必须和该程序放在同一个文件夹下。



F:\个人资料\学习\计算机大三\2024 编译原理课程项目\上交材料\2. 测试文件夹\3. 任务三测试文本及程序\项目任务1\2. 程序生成文件_lexer.c - [Executing] - Dev-C++ 5.11

TDM-GCC 4.9.2 64-bit Debug

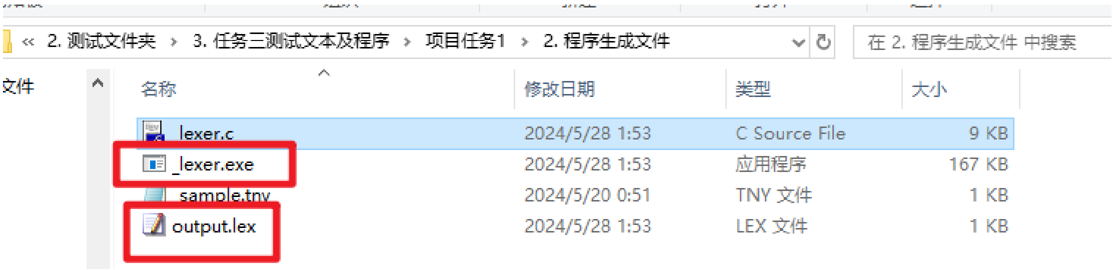
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
const char* keywordSet[9] = {"do","double","else","float","if","int","return",
static struct
{
    const char* str;
    const char* value;
} reservedWords[20] = {{"+","PLUS"},{"(","LLM"},{")","RLM"},{"*","MULTIPLY"},{
void concat(char str[], char tmp) {
    size_t len = strlen(str);

    str[len] = tmp;

    str[len + 1] = '\0';
}
```

```
编译结果...

- 错误: 0
- 警告: 0
- 输出文件名: F:\个人资料\学习\计算机大三\2024 编译原理课程项目\上交材料\2. 测试文件夹\3. 任务三测试文本及程序\项目
- 输出大小: 166.416015625 KiB
- 编译时间: 0.25s
```

行: 1 列: 1 已选择: 0 总行数: 468 长度: 8539 插入 在 0.187 秒内完成解析

上图可知：编译成功
并成功生成 lex 文件：

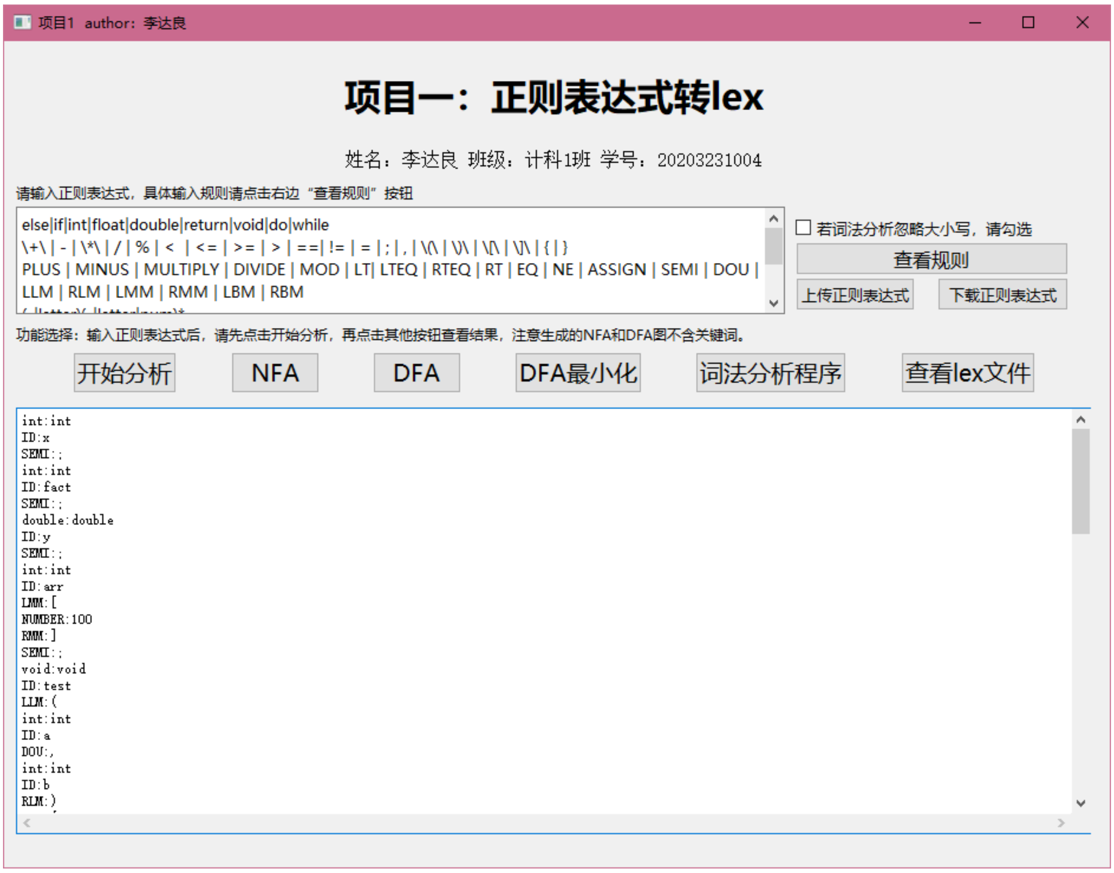
7、查看 lex 文件



« 2. 测试文件夹 › 3. 任务三测试文本及程序 › 项目任务1 › 2. 程序生成文件

在 2. 程序生成文件 中搜索

名称	修改日期	类型	大小
_lexer.c	2024/5/28 1:53	C Source File	9 KB
_lexer.exe	2024/5/28 1:53	应用程序	167 KB
sample.tny	2024/5/20 0:51	TNY 文件	1 KB
output.lex	2024/5/28 1:53	LEX 文件	1 KB



项目一：正则表达式转lex

姓名：李达良 班级：计科1班 学号：20203231004

请输入正则表达式，具体输入规则请点击右边“查看规则”按钮

```
else|if|int|float|double|return|void|do|while
\+\ | - | \*\ | / | % | <  | <= | >= | > | ==| != | = | ; | , | \(\ | \)\ | \[\ | \]\ | { | }
PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LT| LTEQ | RTEQ | RT | EQ | NE | ASSIGN | SEMI | DOU |
LLM | RLM | LMM | RMM | LBM | RBM
```

□ 若词法分析忽略大小写，请勾选

功能选择：输入正则表达式后，请先点击开始分析，再点击其他按钮查看结果，注意生成的NFA和DFA图不含关键词。

```
int:int
ID:x
SEMI:;
int:int
ID:fact
SEMI:;
double:double
ID:y
SEMI:;
int:int
ID:arr
LMM:[
NUMBER:100
RMM:]
SEMI:;
void:void
ID:test
LLM:(
int:int
ID:a
DOU:,
int:int
ID:b
RLM:)
```

具体 lex 如下：

```
int:int
ID:x
SEMI:;
int:int
ID:fact
SEMI:;
double:double
ID:y
SEMI:;
int:int
ID:arr
LMM:[
NUMBER:100
RMM:]
SEMI:;
void:void
ID:test
LLM:(
int:int
ID:a
DOU:,
int:int
```

```
ID:b
RLM:)
LBM:{
ID:a
ASSIGN:=
NUMBER:1
SEMI:;
ID:b
ASSIGN:=
NUMBER:2
SEMI:;
return:return
SEMI:;
RBM:}
void:void
ID:main
LLM:(
void:void
RLM:)
LBM:{
ID:x
ASSIGN:=
NUMBER:2
SEMI:;
ID:y
ASSIGN:=
NUMBER:3
SEMI:;
if:if
LLM:(
ID:x
RT:>
NUMBER:1
RLM:)
do:do
ID:fact
ASSIGN:=
ID:fact
MULTIPLY:*
ID:x
SEMI:;
while:while
LLM:(
ID:fact
RTEQ:>=
NUMBER:1
RLM:)
SEMI:;
else:else
ID:x
ASSIGN:=
NUMBER:1
SEMI:;
return:return
SEMI:;
RBM:}
EOF:EOF
```

对照 minic 源程序，可知解析完全正确。

测试结果

任务三-项目一的测试完全通过

二、项目任务二的测试

1、准备工作

minic 的文法：

```
program | definition-list | definition | variable-definition | function-definition | type-indicator | parameters | compound-stmt | parameter-list | parameter | local-definitions | statement-list | statement | expression-stmt | condition-stmt | dowhile-stmt | return-stmt | expression | simple-expression | variable | additive-expression | relop | term | addop | mulop | factor | call | arguments | argument-list
ID | SEMI | LMM | RMM | int | float | double | void | LLM | RLM | if | else | do | while | return | LTEQ | LT | RT | RTEQ | EQ | NE | PLUS | MINUS | MULTIPLY | DIVIDE | MOD | NUMBER | DOU | LBM | RBM | ASSIGN
program -> definition-list
definition-list -> definition-list definition
definition-list -> definition
definition -> variable-definition
definition -> function-definition
variable-definition -> type-indicator ID SEMI
variable-definition -> type-indicator ID LMM NUMBER RMM SEMI
type-indicator -> int
type-indicator -> float
type-indicator -> double
type-indicator -> void
function-definition -> type-indicator ID LLM parameters RLM compound-stmt
parameters -> parameter-list
parameters -> void
parameter-list -> parameter-list DOU parameter
parameter-list -> parameter
parameter -> type-indicator ID
parameter -> type-indicator ID LMM RMM
compound-stmt -> LBM local-definitions statement-list RBM
local-definitions -> local-definitions variable-definition
local-definitions -> @
statement-list -> statement-list statement
statement-list -> @
statement -> expression-stmt
statement -> compound-stmt
statement -> condition-stmt
statement -> dowhile-stmt
statement -> return-stmt
expression-stmt -> expression SEMI
expression-stmt -> SEMI
```

```
condition-stmt -> if LLM expression RLM statement
condition-stmt -> if LLM expression RLM statement else statement
dowhile-stmt -> do statement while LLM expression RLM SEMI
return-stmt -> return SEMI
return-stmt -> return expression SEMI
expression -> variable ASSIGN expression
expression -> simple-expression
variable -> ID
variable -> ID LMM expression RMM
simple-expression -> additive-expression relop additive-expression
simple-expression -> additive-expression
relop -> LTEQ
relop -> LT
relop -> RT
relop -> RTEQ
relop -> EQ
relop -> NE
additive-expression -> additive-expression addop term
additive-expression -> term
addop -> PLUS
addop -> MINUS
term -> term mulop factor
term -> factor
mulop -> MULTIPLY
mulop -> DIVIDE
mulop -> MOD
factor -> LLM expression RLM
factor -> variable
factor -> call
factor -> NUMBER
call -> ID LLM arguments RLM
arguments -> argument-list
arguments -> @
argument-list -> argument-list DOU expression
argument-list -> expression
```

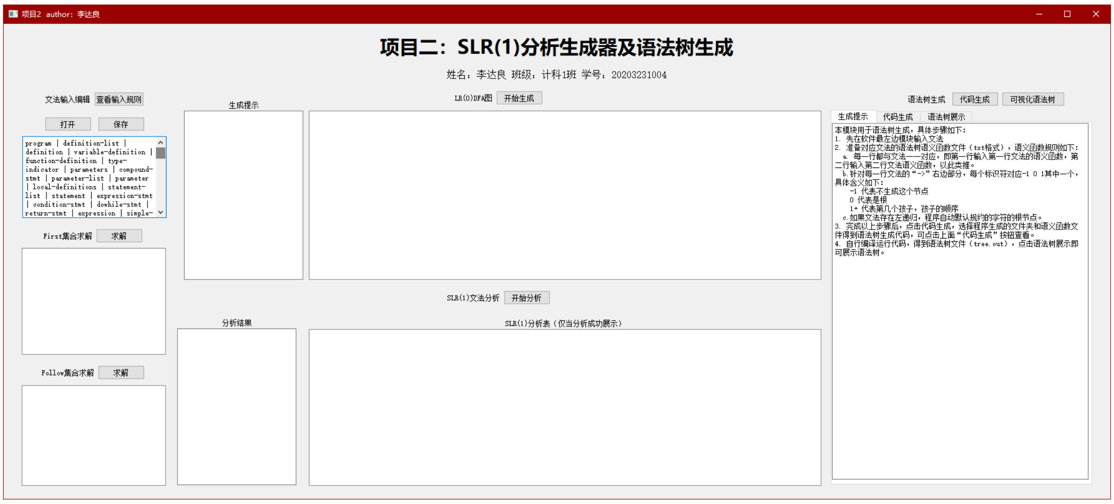
语义函数：

```
0
0 1
0
0
0
1 0 -1
1 0 -1 2 -1 -1
```

```
0
0
0
0
1 0 -1 2 -1 3
0
0
0 -1 1
0
1 0
1 0 -1 -1
-1 0 1 -1
0 1
-1
0 1
-1
0
0
0
0
0
0 -1
-1
0 -1 1 -1 2
0 -1 1 -1 2 -1 3
0 1 -1 -1 2 -1 -1
0 -1
0 1 -1
1 0 2
0
0
0 -1 1 -1
1 0 2
0
0
0
0
0
0
0
0 1 2
0
0
0
```

```
0 1 2
0
0
0
0
-1 0 -1
0
0
0
0 -1 1 -1
0
-1
0 -1 1
0
```

输入文法：



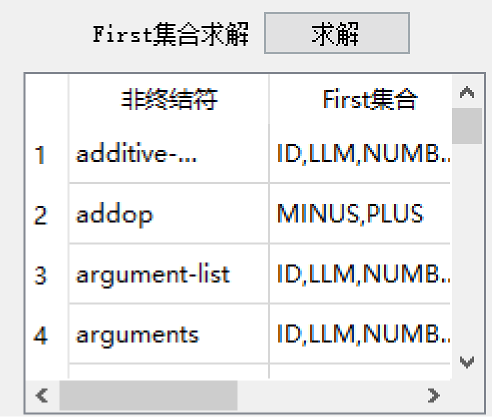
项目二：SLR(1)分析生成器及语法树生成

姓名：李达良 班级：计科1班 学号：20203231004

```
program | definition-list |
definition | variable-definition |
function-definition | type-
indicator | parameters | compound-
stmt | parameter-list | parameter
| local-definitions | statement-
list | statement | expression-stmt
| condition-stmt | dowhile-stmt |
return-stmt | expression | simple-
```

本模块用于语法树生成，具体步骤如下：
1. 先在软件最左边模块输入文法
2. 准备对应文法的语法树语义函数文件（txt格式），语义函数规则如下：
  a. 每一行都与文法一一对应，即第一行输入第一行文法的语义函数，第二行输入第二行文法语义函数，以此类推。
  b.针对每一行文法的“->”右边部分，每个标识符对应-1 0 1其中一个，具体含义如下：
    -1 代表不生成这个节点
    0 代表是根
    1+ 代表第几个孩子，孩子的顺序
  c.如果文法存在左递归，程序自动默认规约的字符的根节点。
3. 完成以上步骤后，点击代码生成，选择程序生成的文件夹和语义函数文件得到语法树生成代码，可点击上面“代码生成”按钮查看。
4. 自行编译运行代码，得到语法树文件（tree.out），点击语法树展示即可展示语法树。

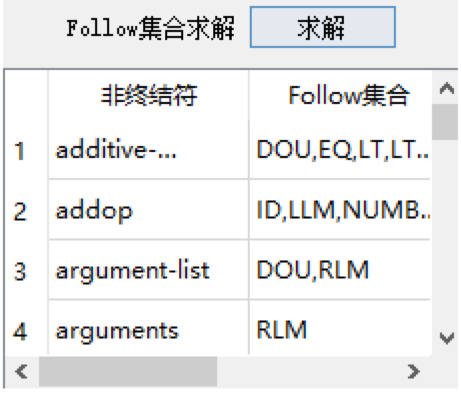
2、求解 first 集合



	非终结符	First集合
1	additive-...	ID,LLM,NUMB...
2	addop	MINUS,PLUS
3	argument-list	ID,LLM,NUMB...
4	arguments	ID,LLM,NUMB...

项目过多，在此不一一展示。

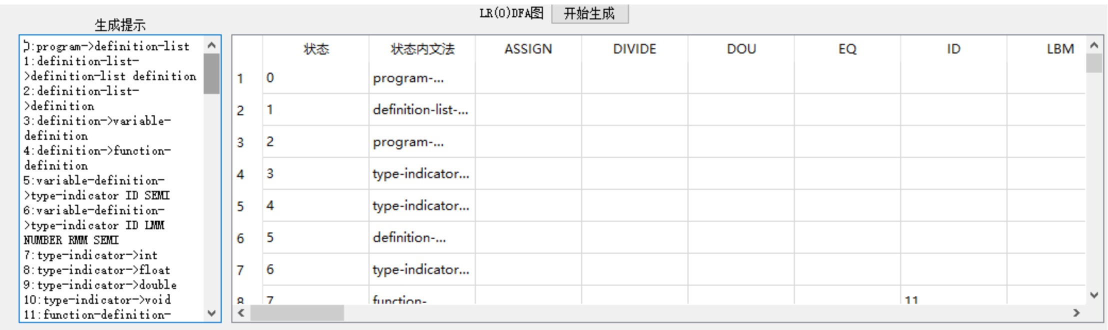
3、求解 follow 集合



	非终结符	Follow集合
1	additive-...	DOU,EQ,LT,LT..
2	addop	ID,LLM,NUMB..
3	argument-list	DOU,RLM
4	arguments	RLM

项目过多，在此不一一展示。

4、LR0

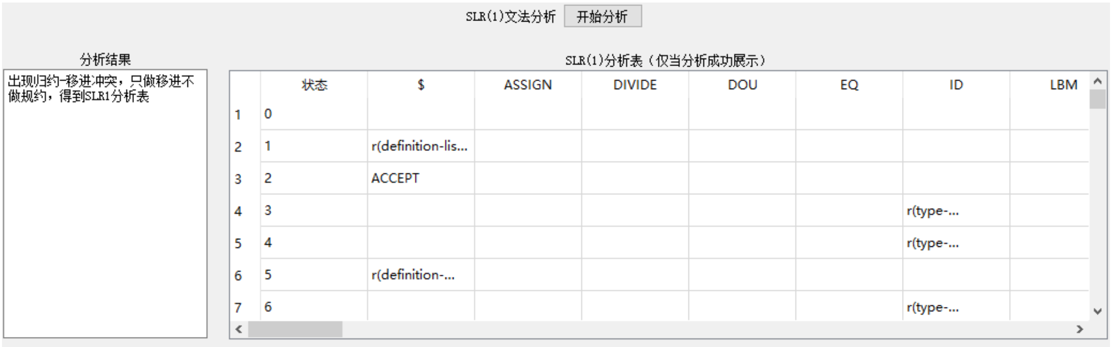


```
0:program->definition-list
1:definition-list-
>definition-list definition
2:definition-list-
>definition
3:definition->variable-
definition
4:definition->function-
definition
5:variable-definition-
>type-indicator ID SEMI
6:variable-definition-
>type-indicator ID LMM
NUMBER RMM SEMI
7:type-indicator->int
8:type-indicator->float
9:type-indicator->double
10:type-indicator->void
11:function-definition-
```

	状态	状态内文法	ASSIGN	DIVIDE	DOU	EQ	ID	LBM
1	0	program-...						
2	1	definition-list-...						
3	2	program-...						
4	3	type-indicator...						
5	4	type-indicator...						
6	5	definition-...						
7	6	type-indicator...						
8	7	function-					11	

项目太多，未能展示完全

5、SLR1 表



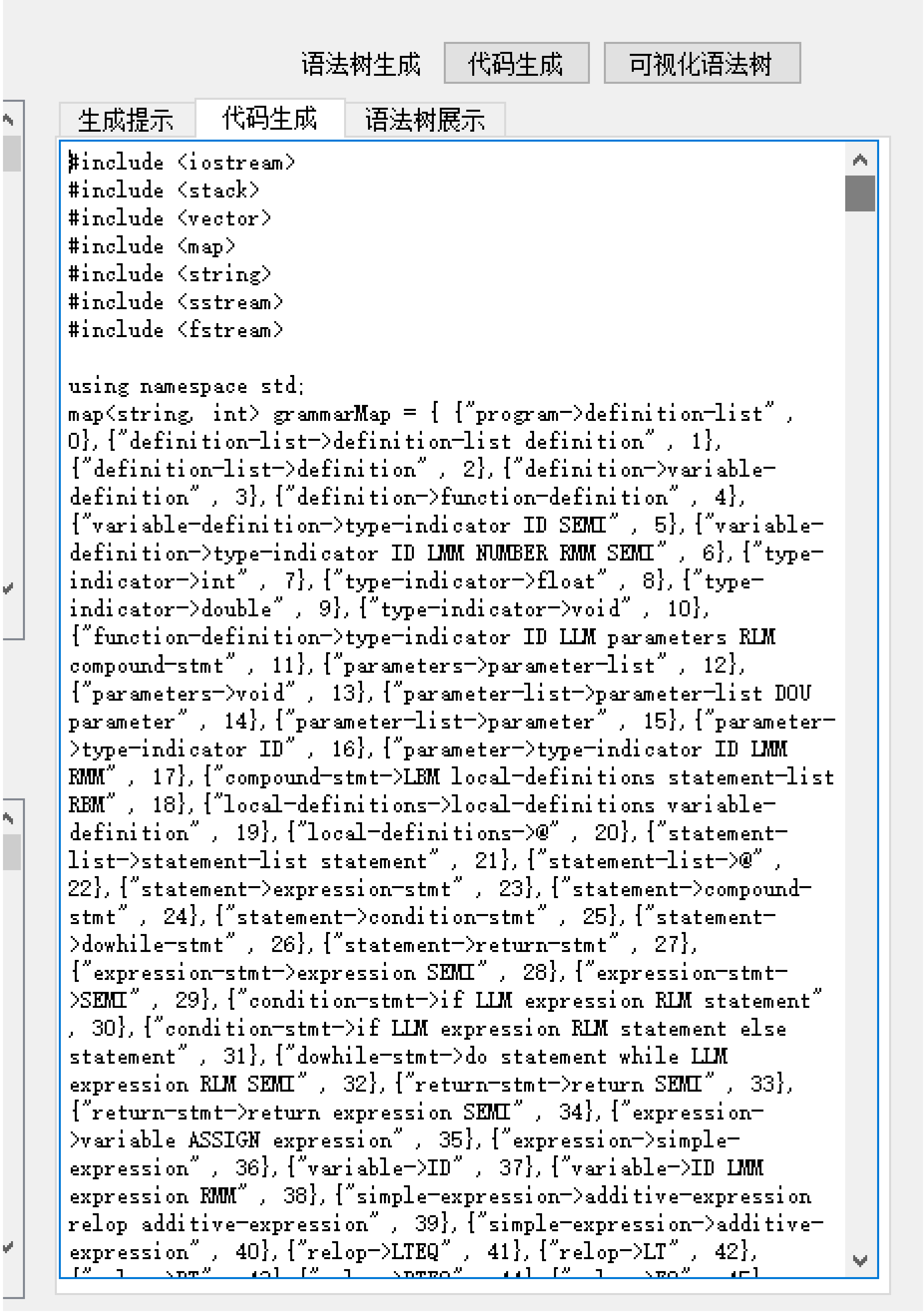
出现归约-移进冲突，只做移进不做规约，得到SLR1分析表

	状态	$	ASSIGN	DIVIDE	DOU	EQ	ID	LBM
1	0							
2	1	r(definition-lis...						
3	2	ACCEPT						
4	3						r(type-...	
5	4						r(type-...	
6	5	r(definition-...						
7	6						r(type-...	

项目太多，未能展示完全
出现移进归约冲突，我们做只移进不规约的处理

6、语法树代码生成

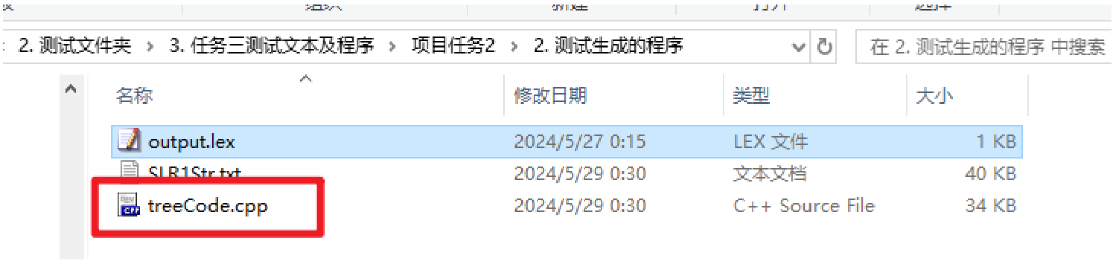
注意：lex 所在路径即为代码生成路径



```
#include <iostream>
#include <stack>
#include <vector>
#include <map>
#include <string>
#include <sstream>
#include <fstream>

using namespace std;
map<string, int> grammarMap = { {"program->definition-list" ,
0},{"definition-list->definition-list definition" , 1},
{"definition-list->definition" , 2},{"definition->variable-
definition" , 3},{"definition->function-definition" , 4},
{"variable-definition->type-indicator ID SEMI" , 5},{"variable-
definition->type-indicator ID LMM NUMBER RMM SEMI" , 6},{"type-
indicator->int" , 7},{"type-indicator->float" , 8},{"type-
indicator->double" , 9},{"type-indicator->void" , 10},
{"function-definition->type-indicator ID LLM parameters RLM
compound-stmt" , 11},{"parameters->parameter-list" , 12},
{"parameters->void" , 13},{"parameter-list->parameter-list DOU
parameter" , 14},{"parameter-list->parameter" , 15},{"parameter-
>type-indicator ID" , 16},{"parameter->type-indicator ID LMM
RMM" , 17},{"compound-stmt->LBM local-definitions statement-list
RBM" , 18},{"local-definitions->local-definitions variable-
definition" , 19},{"local-definitions->@" , 20},{"statement-
list->statement-list statement" , 21},{"statement-list->@" ,
22},{"statement->expression-stmt" , 23},{"statement->compound-
stmt" , 24},{"statement->condition-stmt" , 25},{"statement-
>dowhile-stmt" , 26},{"statement->return-stmt" , 27},
{"expression-stmt->expression SEMI" , 28},{"expression-stmt-
>SEMI" , 29},{"condition-stmt->if LLM expression RLM statement"
, 30},{"condition-stmt->if LLM expression RLM statement else
statement" , 31},{"dowhile-stmt->do statement while LLM
expression RLM SEMI" , 32},{"return-stmt->return SEMI" , 33},
{"return-stmt->return expression SEMI" , 34},{"expression-
>variable ASSIGN expression" , 35},{"expression->simple-
expression" , 36},{"variable->ID" , 37},{"variable->ID LMM
expression RMM" , 38},{"simple-expression->additive-expression
relop additive-expression" , 39},{"simple-expression->additive-
expression" , 40},{"relop->LTEQ" , 41},{"relop->LT" , 42},
```

7、查看语法树生成代码文件

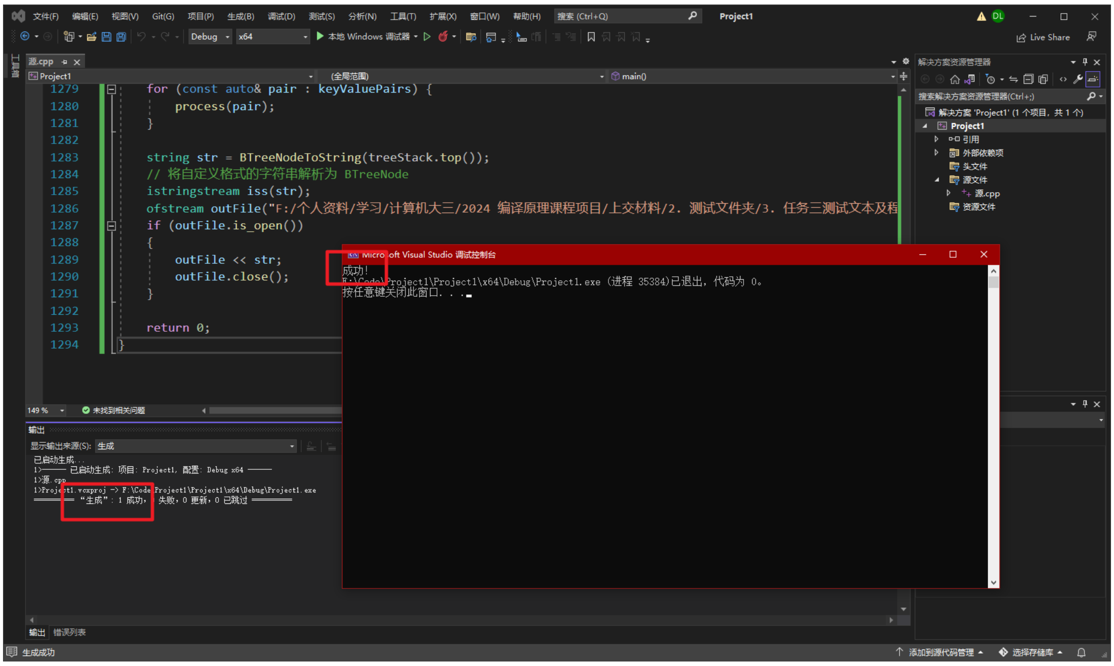
PS:请用 C++11 以上进行编译


2. 测试文件夹 › 3. 任务三测试文本及程序 › 项目任务2 › 2. 测试生成的程序
在 2. 测试生成的程序 中搜索

名称	修改日期	类型	大小
output.lex	2024/5/27 0:15	LEX 文件	1 KB
SLR1Str.txt	2024/5/29 0:30	文本文档	40 KB
treeCode.cpp	2024/5/29 0:30	C++ Source File	34 KB

具体语法树代码太长，在此不进行展示，请打开 treeCode.cpp 查看

编译运行：

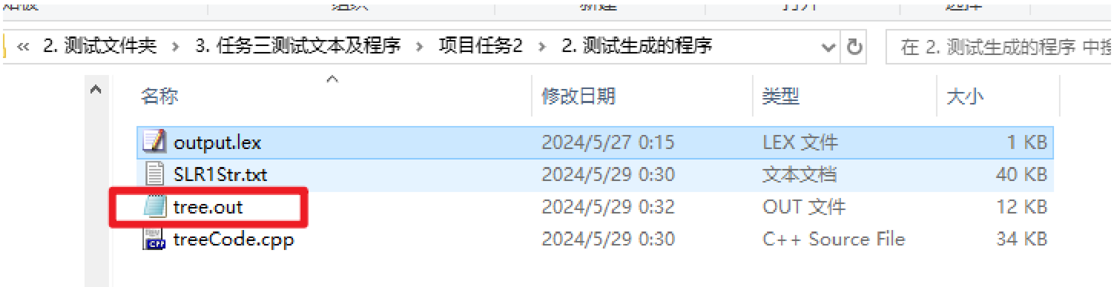


```
    for (const auto& pair : keyValuePairs) {
        process(pair);
    }

    string str = BTreeNodeToString(treeStack.top());
    // 将自定义格式的字符串解析为 BTreeNode
    istringstream iss(str);
    ofstream outFile("F:/个人资料/学习/计算机大三/2024 编译原理课程项目/上交材料/2. 测试文件夹/3. 任务三测试文本及程
    if (outFile.is_open())
    {
        outFile << str;
        outFile.close();
    }

    return 0;
}
```

```
成功!
F:\Code\Project1\Project1\x64\Debug\Project1.exe (进程 35384)已退出，代码为 0。
按任意键关闭此窗口. . .
```

```
已启动生成...
1>------ 已启动生成: 项目: Project1, 配置: Debug x64 ------
1>源.cpp
1>Project1.vcxproj -> F:\Code\Project1\Project1\x64\Debug\Project1.exe
========== “生成”: 1 成功, 0 失败, 0 更新, 0 已跳过 ==========
```

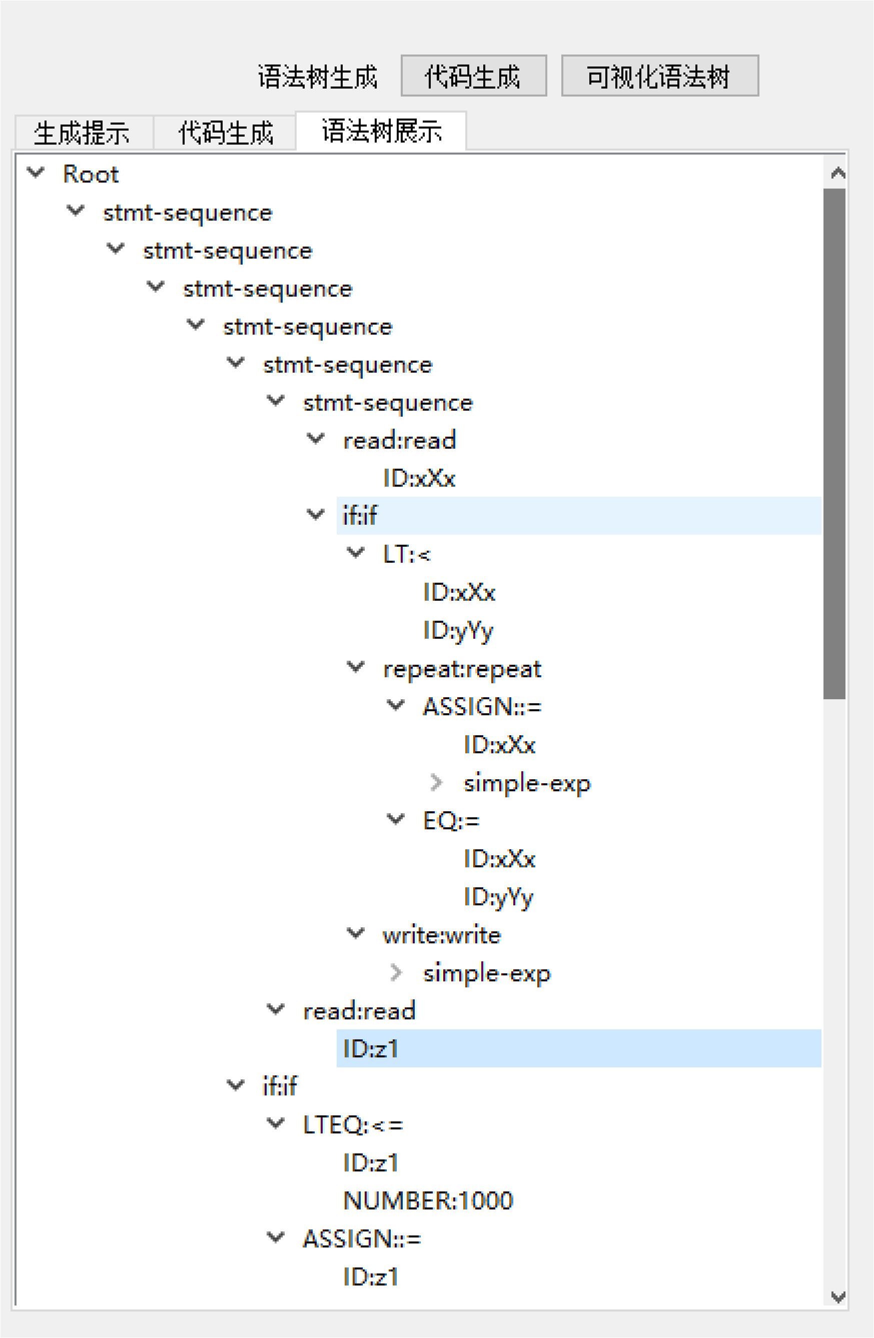
得到语法树：



« 2. 测试文件夹 › 3. 任务三测试文本及程序 › 项目任务2 › 2. 测试生成的程序

名称	修改日期	类型	大小
output.lex	2024/5/27 0:15	LEX 文件	1 KB
SLR1Str.txt	2024/5/29 0:30	文本文档	40 KB
tree.out	2024/5/29 0:32	OUT 文件	12 KB
treeCode.cpp	2024/5/29 0:30	C++ Source File	34 KB

8、可视化语法树



```
Root
  stmt-sequence
    stmt-sequence
      stmt-sequence
        stmt-sequence
          stmt-sequence
            stmt-sequence
              read:read
                ID:xXx
              if:if
                LT:<
                  ID:xXx
                  ID:yYy
                repeat:repeat
                  ASSIGN::=
                    ID:xXx
                    simple-exp
                  EQ:=
                    ID:xXx
                    ID:yYy
                write:write
                  simple-exp
            read:read
              ID:z1
          if:if
            LTEQ:<=
              ID:z1
              NUMBER:1000
            ASSIGN::=
              ID:z1
```

测试结果

任务三-项目二的测试完全通过