项目测试报告

任务一

1、准备工作

tiny 的正则表达式：

```
letter = [a-zA-Z]
digit  = [0-9]
_number100=digit+
_identifier200=letter(letter|digit)*
_keyword300 = if | then | else | end | repeat | read | until | write
_special400S = \+ | - | \* | / | ^ | < | <> | <= | >= | > | = | ;
| :=
_annotation500={( digit | letter | _special )*}
```

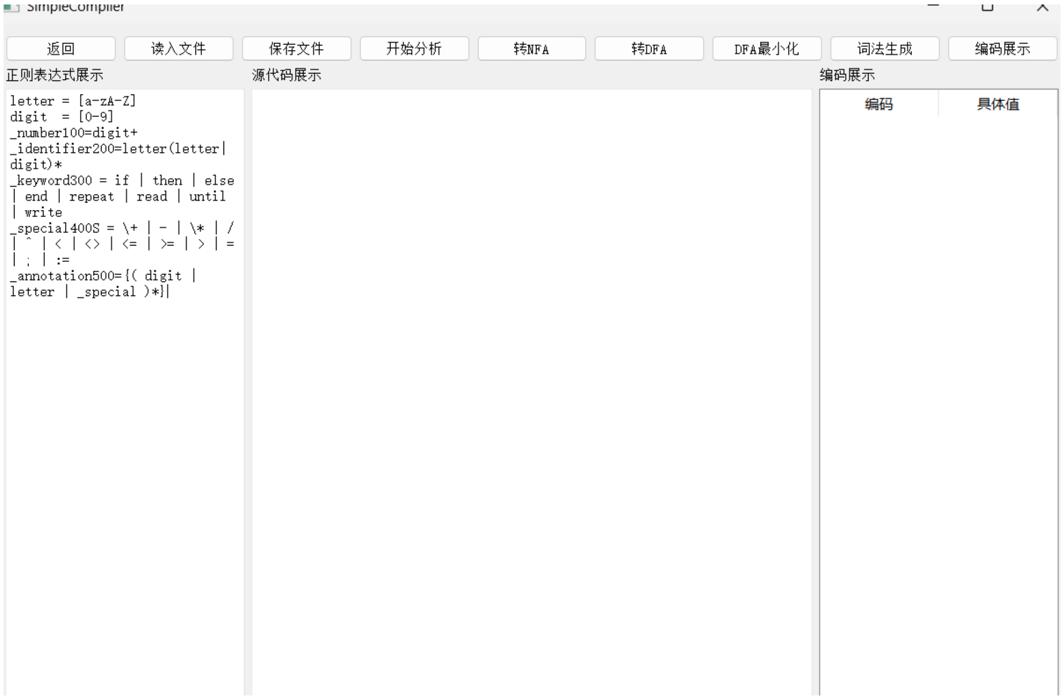
sample.tny 文件：

```
read x ;
{
   多行注释
}
if x < 10 then
    y := x * 3 / 2 ;
    write y
end
else
    repeat
        x := x + 1 - 2
    until x = 0
end ;
{符号部分}
if  x <= 99 then
    x := 1
end
else
    x := 1
end ;

if  x <> 100 then
    x := 1
end ;
if  x > 99 then
    x := 1
end ;
if  x >= 100 then
    x := x ^ 2
```

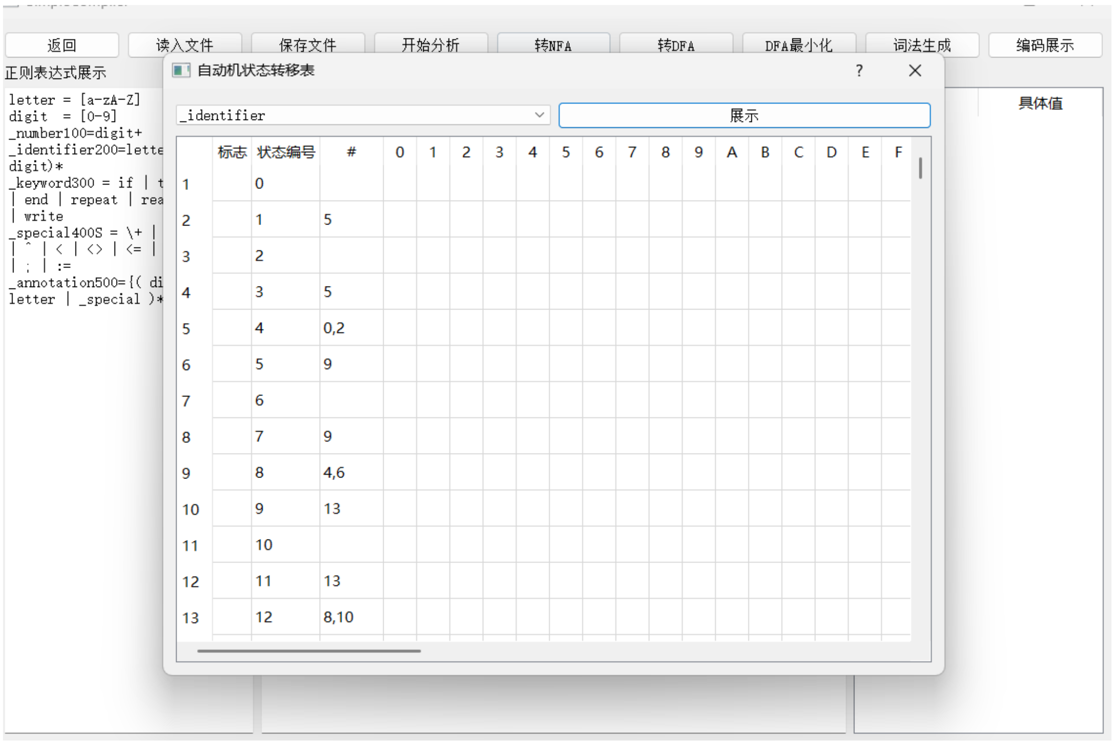
```
end
else
    x := 0
end
```

输入正则表达式：



```
letter = [a-zA-Z]
digit  = [0-9]
_number100=digit+
_identifier200=letter(letter|
digit)*
_keyword300 = if | then | else
| end | repeat | read | until
| write
_special400S = \+ | - | \* | /
| ^ | < | <> | <= | >= | > | =
| ; | :=
_annotation500={( digit |
letter | _special )*}|
```

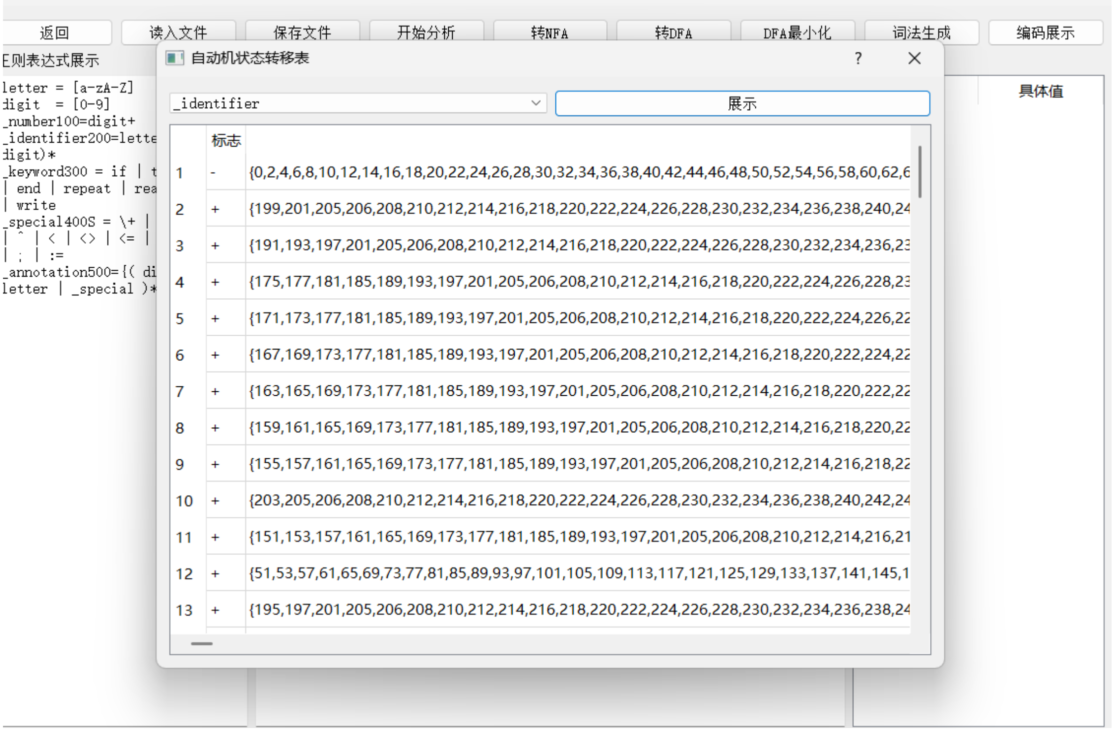
2、NFA



```
letter = [a-zA-Z]
digit  = [0-9]
_number100=digit+
_identifier200=lette
digit)*
_keyword300 = if | t
| end | repeat | rea
| write
_special400S = \+ |
| ^ | < | <> | <= |
| ; | :=
_annotation500={( di
letter | _special )*
```

	标志	状态编号	#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		0																	
2		1	5																
3		2																	
4		3	5																
5		4	0,2																
6		5	9																
7		6																	
8		7	9																
9		8	4,6																
10		9	13																
11		10																	
12		11	13																
13		12	8,10																

3、DFA 图

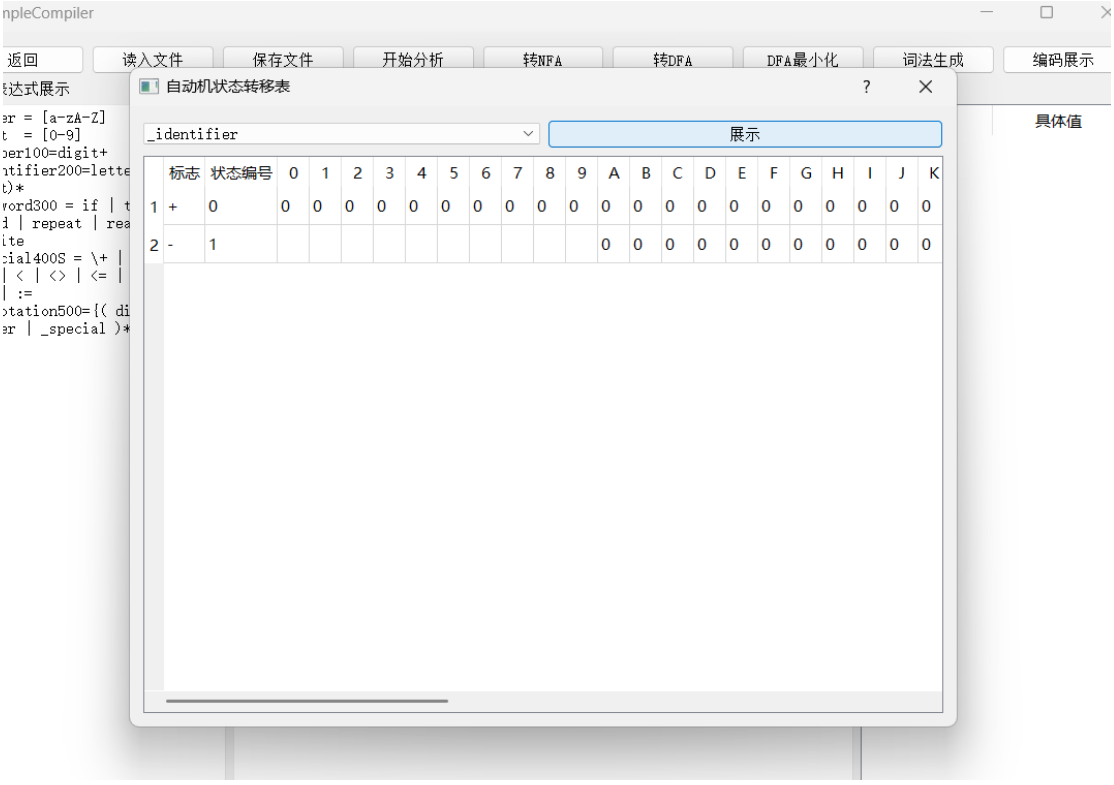


E则表达式展示

```
letter = [a-zA-Z]
digit  = [0-9]
_number100=digit+
_identifier200=lette
digit)*
_keyword300 = if | t
| end | repeat | rea
| write
_special400S = \+ |
| ^ | < | <> | <= |
| ; | :=
_annotation500={( di
letter | _special )*
```

	标志	
1	-	{0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,6
2	+	{199,201,205,206,208,210,212,214,216,218,220,222,224,226,228,230,232,234,236,238,240,24
3	+	{191,193,197,201,205,206,208,210,212,214,216,218,220,222,224,226,228,230,232,234,236,23
4	+	{175,177,181,185,189,193,197,201,205,206,208,210,212,214,216,218,220,222,224,226,228,23
5	+	{171,173,177,181,185,189,193,197,201,205,206,208,210,212,214,216,218,220,222,224,226,22
6	+	{167,169,173,177,181,185,189,193,197,201,205,206,208,210,212,214,216,218,220,222,224,22
7	+	{163,165,169,173,177,181,185,189,193,197,201,205,206,208,210,212,214,216,218,220,222,22
8	+	{159,161,165,169,173,177,181,185,189,193,197,201,205,206,208,210,212,214,216,218,220,22
9	+	{155,157,161,165,169,173,177,181,185,189,193,197,201,205,206,208,210,212,214,216,218,22
10	+	{203,205,206,208,210,212,214,216,218,220,222,224,226,228,230,232,234,236,238,240,242,24
11	+	{151,153,157,161,165,169,173,177,181,185,189,193,197,201,205,206,208,210,212,214,216,21
12	+	{51,53,57,61,65,69,73,77,81,85,89,93,97,101,105,109,113,117,121,125,129,133,137,141,145,1
13	+	{195,197,201,205,206,208,210,212,214,216,218,220,222,224,226,228,230,232,234,236,238,24

4、DFA 图最小化

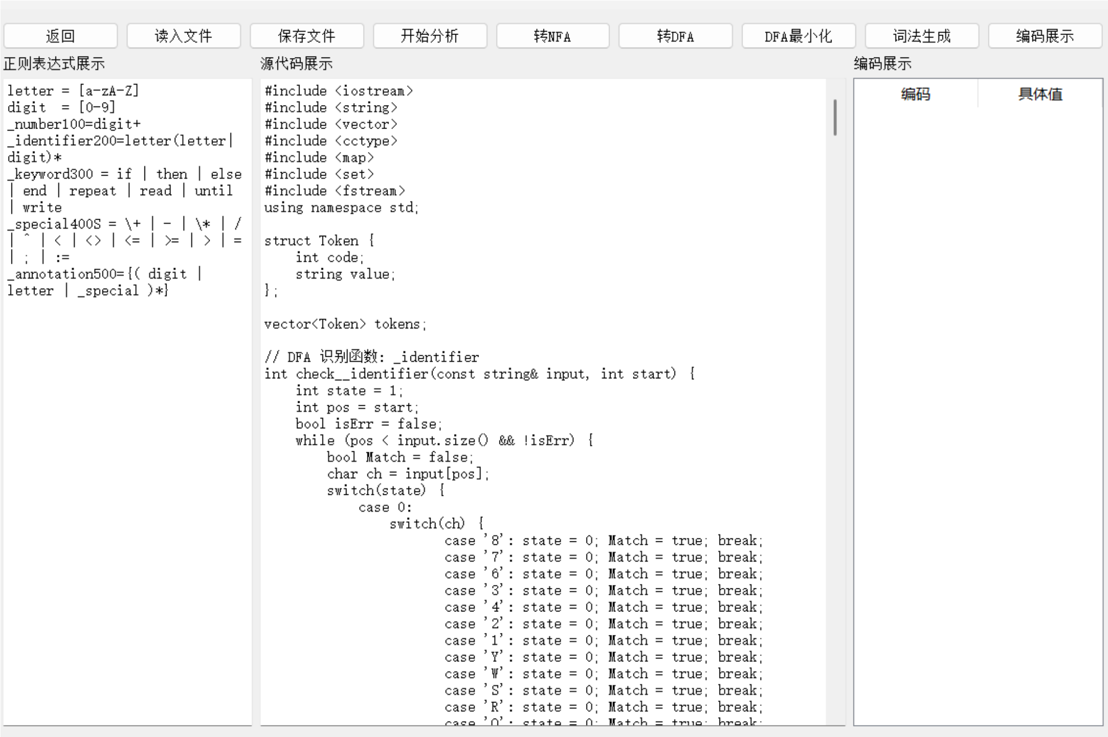


表达式展示

```
er = [a-zA-Z]
t  = [0-9]
ber100=digit+
ntifier200=lette
t)*
word300 = if | t
d | repeat | rea
ite
cial400S = \+ |
| < | <> | <= |
| :=
otation500={( di
er | _special )*
```

	标志	状态编号	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K
1	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	-	1											0	0	0	0	0	0	0	0	0	0	0

5、生成词法程序



```
letter = [a-zA-Z]
digit  = [0-9]
_number100=digit+
_identifier200=letter(letter|
digit)*
_keyword300 = if | then | else
| end | repeat | read | until
| write
_special400S = \+ | - | \* | /
| ^ | < | <> | <= | >= | > | =
| ; | :=
_annotation500={( digit |
letter | _special )*}
```

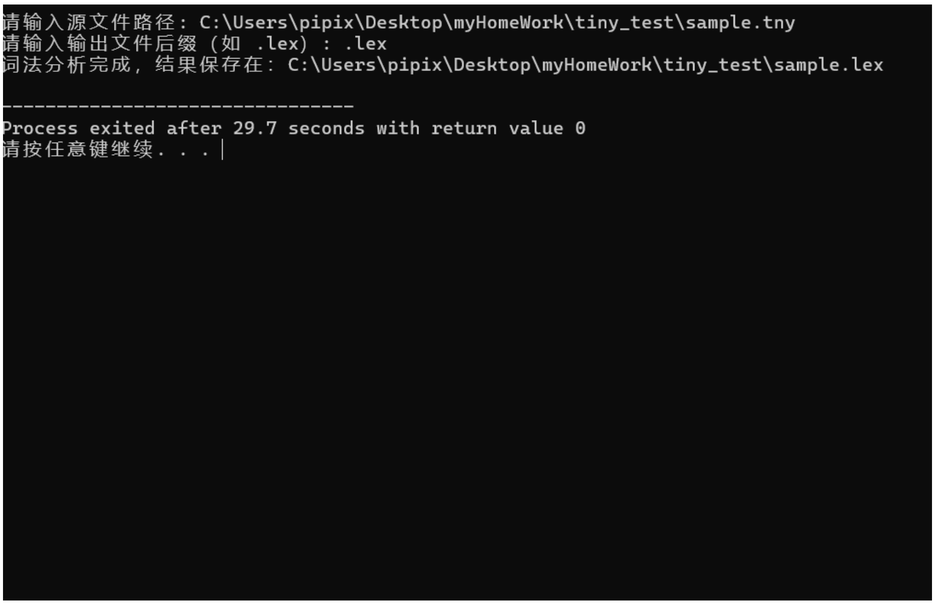
```
#include <iostream>
#include <string>
#include <vector>
#include <cctype>
#include <map>
#include <set>
#include <fstream>
using namespace std;

struct Token {
    int code;
    string value;
};

vector<Token> tokens;

// DFA 识别函数: _identifier
int check__identifier(const string& input, int start) {
    int state = 1;
    int pos = start;
    bool isErr = false;
    while (pos < input.size() && !isErr) {
        bool Match = false;
        char ch = input[pos];
        switch(state) {
            case 0:
                switch(ch) {
                       case ’8’: state = 0; Match = true; break;
                       case ’7’: state = 0; Match = true; break;
                       case ’6’: state = 0; Match = true; break;
                       case ’3’: state = 0; Match = true; break;
                       case ’4’: state = 0; Match = true; break;
                       case ’2’: state = 0; Match = true; break;
                       case ’1’: state = 0; Match = true; break;
                       case ’Y’: state = 0; Match = true; break;
                       case ’W’: state = 0; Match = true; break;
                       case ’S’: state = 0; Match = true; break;
                       case ’R’: state = 0; Match = true; break;
                       case ’Q’: state = 0; Match = true; break;
```

6、编译 cpp 文件并运行



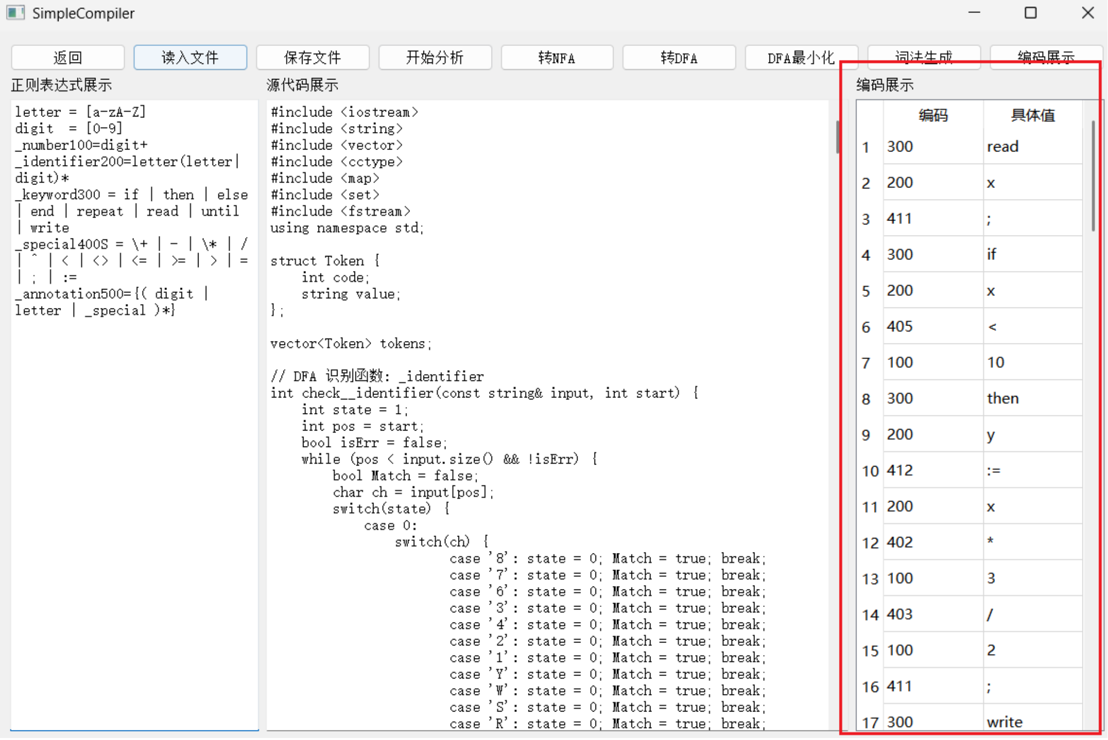
```
请输入源文件路径： C:\Users\pipix\Desktop\myHomeWork\tiny_test\sample.tny
请输入输出文件后缀（如 .lex）： .lex
词法分析完成，结果保存在： C:\Users\pipix\Desktop\myHomeWork\tiny_test\sample.lex

--------------------------------
Process exited after 29.7 seconds with return value 0
请按任意键继续. . .
```

上图可知：编译成功

并成功生成 lex 文件：

7、查看 lex 文件



正则表达式展示

```
letter = [a-zA-Z]
digit  = [0-9]
_number100=digit+
_identifier200=letter(letter|
digit)*
_keyword300 = if | then | else
| end | repeat | read | until
| write
_special400S = \+ | - | \* | /
| ^ | < | <> | <= | >= | > | =
| ; | :=
_annotation500={( digit |
letter | _special )*}
```

源代码展示

```
#include <iostream>
#include <string>
#include <vector>
#include <cctype>
#include <map>
#include <set>
#include <fstream>
using namespace std;

struct Token {
    int code;
    string value;
};

vector<Token> tokens;

// DFA 识别函数: _identifier
int check__identifier(const string& input, int start) {
    int state = 1;
    int pos = start;
    bool isErr = false;
    while (pos < input.size() && !isErr) {
        bool Match = false;
        char ch = input[pos];
        switch(state) {
            case 0:
                switch(ch) {
                       case '8': state = 0; Match = true; break;
                       case '7': state = 0; Match = true; break;
                       case '6': state = 0; Match = true; break;
                       case '3': state = 0; Match = true; break;
                       case '4': state = 0; Match = true; break;
                       case '2': state = 0; Match = true; break;
                       case '1': state = 0; Match = true; break;
                       case 'Y': state = 0; Match = true; break;
                       case 'W': state = 0; Match = true; break;
                       case 'S': state = 0; Match = true; break;
                       case 'R': state = 0; Match = true; break;
```

编码展示

	编码	具体值
1	300	read
2	200	x
3	411	;
4	300	if
5	200	x
6	405	<
7	100	10
8	300	then
9	200	y
10	412	:=
11	200	x
12	402	*
13	100	3
14	403	/
15	100	2
16	411	;
17	300	write

具体 lex 如下：

300 read 200 x 411 ; 300 if 200 x 405 < 100 10 300 then 200 y 412 := 200 x 402 * 100 3 403 / 100 2 411 ; 300 write 200 y 300 end 300 else 300 repeat 200 x 412 := 200 x 400 + 100 1 401 - 100 2 300 until 200 x 410 = 100 0 300 end 411 ; 300 if 200 x 407 <= 100 99 300 then 200 x 412 := 100 1 300 end 300 else 200 x 412 := 100 1 300 end 411 ; 300 if 200 x 406 <> 100 100 300 then 200 x 412 := 100 1 300 end 411 ; 300 if 200 x 409 > 100 99 300 then 200 x 412 := 100 1 300 end 411 ; 300 if 200 x 408 >= 100 100 300 then 200 x 412 := 200 x 404 ^ 100 2 300 end 300 else 200 x 412 := 100 0 300 end

任务一测试完全通过

任务二

1、准备工作

tiny 的文法

program -> stmt-sequence

stmt-sequence -> stmt-sequence ; statement | statement

statement -> if-stmt | repeat-stmt | assign-stmt | read-stmt | write-stmt

if-stmt -> if exp then stmt-sequence end | if exp then stmt-sequence end else stmt-sequence end

repeat-stmt -> repeat stmt-sequence until exp

assign-stmt -> identifier := exp

read-stmt -> read identifier

write-stmt -> write exp

```
exp -> simple-exp comparison-op simple-exp | simple-exp
comparison-op -> < | > | = | <= | <> | >=
simple-exp -> simple-exp addop term | term
addop -> + | -
term -> term mulop factor | factor
mulop -> * | / | % | ^
factor -> ( exp ) | number | identifier
```

tiny 的 sample.lex 编码文本（见上任务一最底部的编码）

tiny 的语义动作表

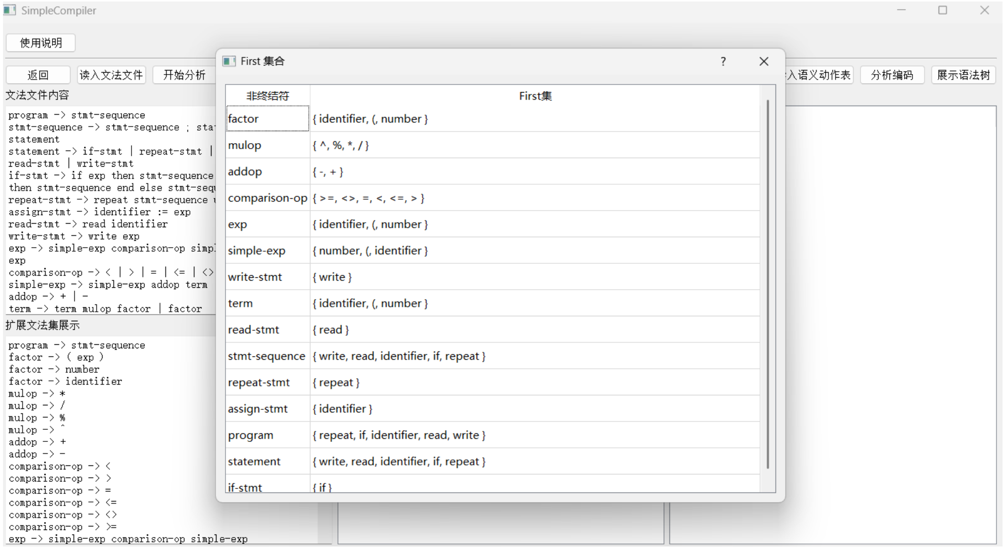
```
program -> stmt-sequence
1
stmt-sequence -> stmt-sequence ; statement
1 0 3
stmt-sequence -> statement
1
statement -> if-stmt
1
statement -> repeat-stmt
1
statement -> assign-stmt
1
statement -> read-stmt
1
statement -> write-stmt
1
if-stmt -> if exp then stmt-sequence end
1 2 0 2 0
if-stmt -> if exp then stmt-sequence end else stmt-sequence end
1 2 0 2 0 0 2 0
repeat-stmt -> repeat stmt-sequence until exp
1 2 0 2
assign-stmt -> identifier := exp
2 1 2
read-stmt -> read identifier
1 2
write-stmt -> write exp
1 2
```

```
exp -> simple-exp comparison-op simple-exp
2 1 2
exp -> simple-exp
1
comparison-op -> <
1
comparison-op -> >
1
comparison-op -> =
1
comparison-op -> <=
1
comparison-op -> <>
1
comparison-op -> >=
1
simple-exp -> simple-exp addop term
2 1 2
simple-exp -> term
1
addop -> +
1
addop -> -
1
term -> term mulop factor
2 1 2
term -> factor
1
mulop -> *
1
mulop -> /
1
mulop -> %
1
mulop -> ^
1
factor -> ( exp )
0 1 0
```

```
factor -> number
1
factor -> identifier
1
```

2、开始测试

① First 集合：



文法文件内容

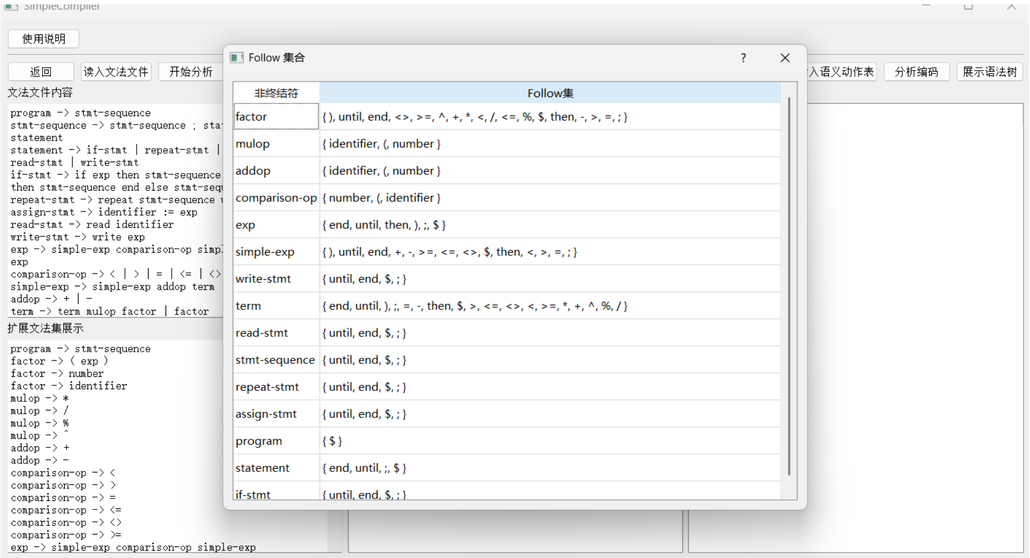
```
program -> stmt-sequence
stmt-sequence -> stmt-sequence ; sta
statement
statement -> if-stmt | repeat-stmt |
read-stmt | write-stmt
if-stmt -> if exp then stmt-sequence
then stmt-sequence end else stmt-seq
repeat-stmt -> repeat stmt-sequence u
assign-stmt -> identifier := exp
read-stmt -> read identifier
write-stmt -> write exp
exp -> simple-exp comparison-op simpl
exp
comparison-op -> < | > | = | <= | <>
simple-exp -> simple-exp addop term
addop -> + | -
term -> term mulop factor | factor
```

扩展文法集展示

```
program -> stmt-sequence
factor -> ( exp )
factor -> number
factor -> identifier
mulop -> *
mulop -> /
mulop -> %
mulop -> ^
addop -> +
addop -> -
comparison-op -> <
comparison-op -> >
comparison-op -> =
comparison-op -> <=
comparison-op -> <>
comparison-op -> >=
exp -> simple-exp comparison-op simple-exp
```

非终结符	First集
factor	{ identifier, (, number }
mulop	{ ^, %, *, / }
addop	{ -, + }
comparison-op	{ >=, <>, =, <, <=, > }
exp	{ identifier, (, number }
simple-exp	{ number, (, identifier }
write-stmt	{ write }
term	{ identifier, (, number }
read-stmt	{ read }
stmt-sequence	{ write, read, identifier, if, repeat }
repeat-stmt	{ repeat }
assign-stmt	{ identifier }
program	{ repeat, if, identifier, read, write }
statement	{ write, read, identifier, if, repeat }
if-stmt	{ if }

② Follow 集合：



文法文件内容

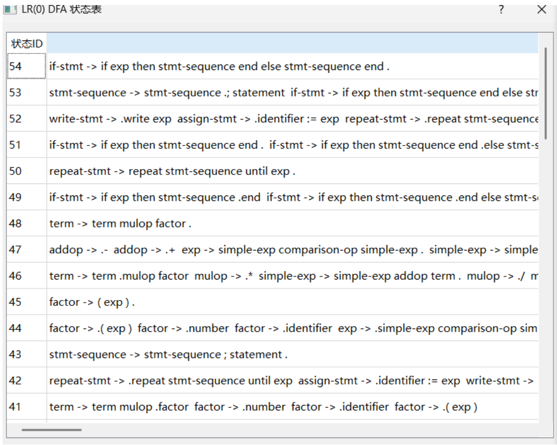
```
program -> stmt-sequence
stmt-sequence -> stmt-sequence ; sta
statement
statement -> if-stmt | repeat-stmt |
read-stmt | write-stmt
if-stmt -> if exp then stmt-sequence
then stmt-sequence end else stmt-seq
repeat-stmt -> repeat stmt-sequence u
assign-stmt -> identifier := exp
read-stmt -> read identifier
write-stmt -> write exp
exp -> simple-exp comparison-op simpl
exp
comparison-op -> < | > | = | <= | <>
simple-exp -> simple-exp addop term
addop -> + | -
term -> term mulop factor | factor
```

扩展文法集展示

```
program -> stmt-sequence
factor -> ( exp )
factor -> number
factor -> identifier
mulop -> *
mulop -> /
mulop -> %
mulop -> ^
addop -> +
addop -> -
comparison-op -> <
comparison-op -> >
comparison-op -> =
comparison-op -> <=
comparison-op -> <>
comparison-op -> >=
exp -> simple-exp comparison-op simple-exp
```

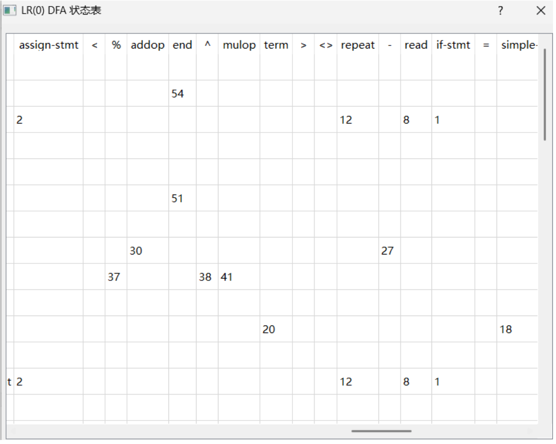
非终结符	Follow集
factor	{), until, end, <>, >=, ^, +, *, <, /, <=, %, $, then, -, >, =, ; }
mulop	{ identifier, (, number }
addop	{ identifier, (, number }
comparison-op	{ number, (, identifier }
exp	{ end, until, then,), ;, $ }
simple-exp	{), until, end, +, -, >=, <=, <>, $, then, <, >, =, ; }
write-stmt	{ until, end, $, ; }
term	{ end, until,), ;, =, -, then, $, >, <=, <>, <, >=, *, +, ^, %, / }
read-stmt	{ until, end, $, ; }
stmt-sequence	{ until, end, $, ; }
repeat-stmt	{ until, end, $, ; }
assign-stmt	{ until, end, $, ; }
program	{ $ }
statement	{ end, until, ;, $ }
if-stmt	{ until, end, $, ; }

③ LR0-DFA：



LR(0) DFA 状态表

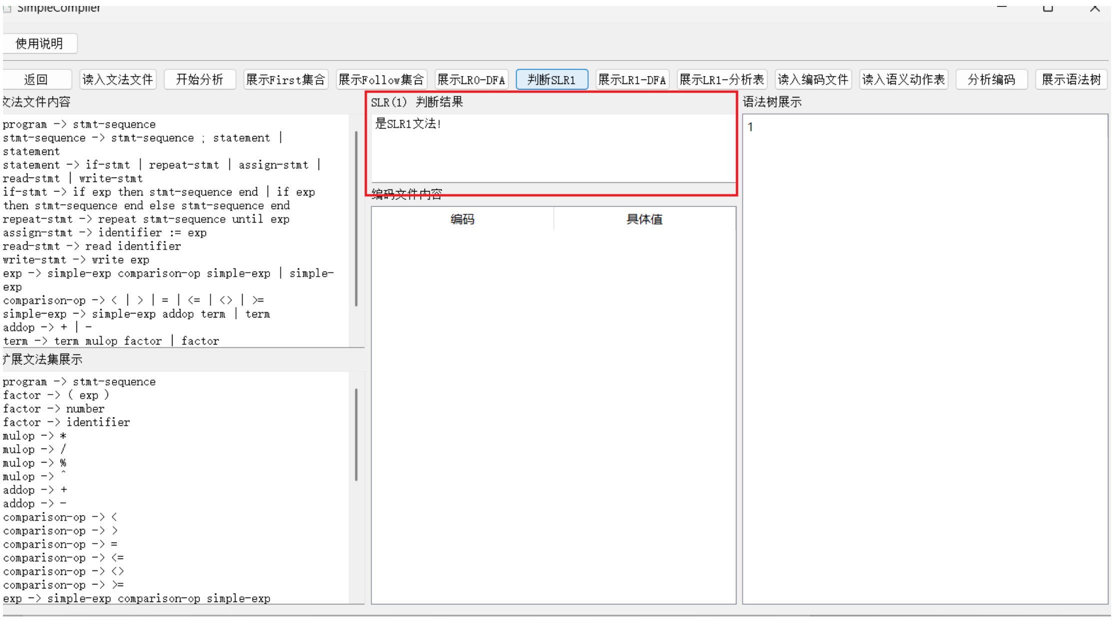
状态ID	
54	if-stmt -> if exp then stmt-sequence end else stmt-sequence end .
53	stmt-sequence -> stmt-sequence .; statement if-stmt -> if exp then stmt-sequence end else str
52	write-stmt -> .write exp assign-stmt -> .identifier := exp repeat-stmt -> .repeat stmt-sequence
51	if-stmt -> if exp then stmt-sequence end . if-stmt -> if exp then stmt-sequence end .else stmt-s
50	repeat-stmt -> repeat stmt-sequence until exp .
49	if-stmt -> if exp then stmt-sequence .end if-stmt -> if exp then stmt-sequence .end else stmt-s
48	term -> term mulop factor .
47	addop -> .- addop -> .+ exp -> simple-exp comparison-op simple-exp . simple-exp -> simple
46	term -> term .mulop factor mulop -> .* simple-exp -> simple-exp addop term . mulop -> ./ m
45	factor -> (exp) .
44	factor -> .(exp) factor -> .number factor -> .identifier exp -> .simple-exp comparison-op sim
43	stmt-sequence -> stmt-sequence ; statement .
42	repeat-stmt -> .repeat stmt-sequence until exp assign-stmt -> .identifier := exp write-stmt ->
41	term -> term mulop .factor factor -> .number factor -> .identifier factor -> .(exp)



LR(0) DFA 状态表

	assign-stmt	<	%	addop	end	^	mulop	term	>	<>	repeat	-	read	if-stmt	=	simple-
					54											
	2										12		8	1		
					51											
				30								27				
			37			38	41									
								20								18
t	2										12		8	1		

④ 判断是否为 SLR1：



文法文件内容

```
program -> stmt-sequence
stmt-sequence -> stmt-sequence ; statement |
statement
statement -> if-stmt | repeat-stmt | assign-stmt |
read-stmt | write-stmt
if-stmt -> if exp then stmt-sequence end | if exp
then stmt-sequence end else stmt-sequence end
repeat-stmt -> repeat stmt-sequence until exp
assign-stmt -> identifier := exp
read-stmt -> read identifier
write-stmt -> write exp
exp -> simple-exp comparison-op simple-exp | simple-
exp
comparison-op -> < | > | = | <= | <> | >=
simple-exp -> simple-exp addop term | term
addop -> + | -
term -> term mulop factor | factor
```

扩展文法集展示

```
program -> stmt-sequence
factor -> ( exp )
factor -> number
factor -> identifier
mulop -> *
mulop -> /
mulop -> %
mulop -> ^
addop -> +
addop -> -
comparison-op -> <
comparison-op -> >
comparison-op -> =
comparison-op -> <=
comparison-op -> <>
comparison-op -> >=
exp -> simple-exp comparison-op simple-exp
```

SLR(1) 判断结果

是SLR1文法!

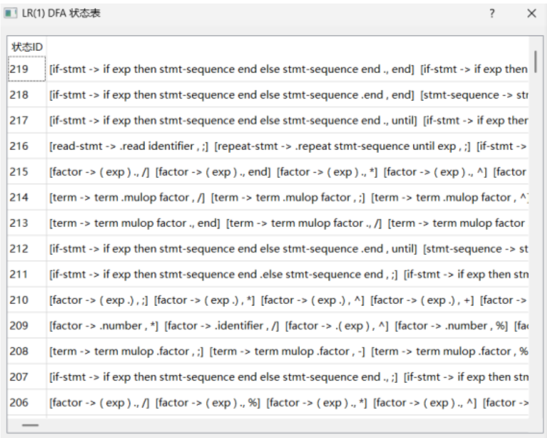
编码文件内容

编码	具体值

语法树展示

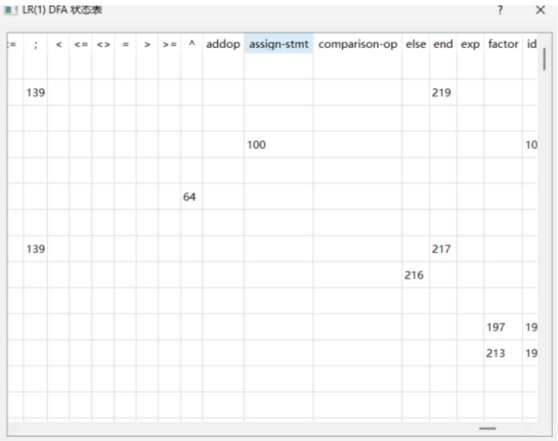
1

⑤ LR1-DFA：



LR(1) DFA 状态表

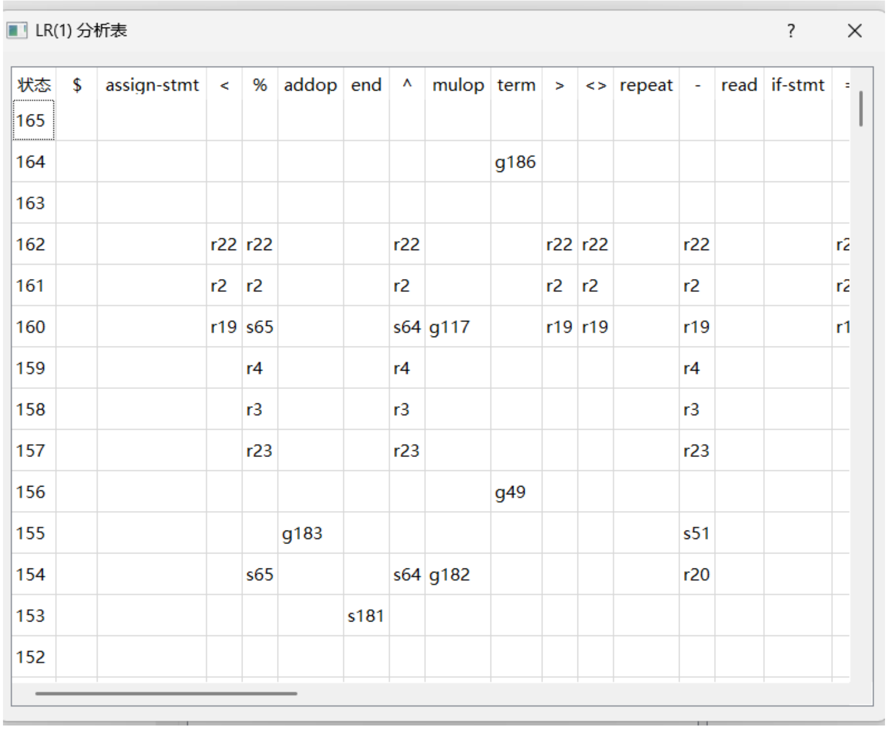
状态ID	
219	[if-stmt -> if exp then stmt-sequence end else stmt-sequence end ., end] [if-stmt -> if exp then
218	[if-stmt -> if exp then stmt-sequence end else stmt-sequence .end , end] [stmt-sequence -> str
217	[if-stmt -> if exp then stmt-sequence end else stmt-sequence end ., until] [if-stmt -> if exp ther
216	[read-stmt -> .read identifier , ;] [repeat-stmt -> .repeat stmt-sequence until exp , ;] [if-stmt ->
215	[factor -> (exp) ., /] [factor -> (exp) ., end] [factor -> (exp) ., *] [factor -> (exp) ., ^] [factor
214	[term -> term .mulop factor , /] [term -> term .mulop factor , ;] [term -> term .mulop factor , ^
213	[term -> term mulop factor ., end] [term -> term mulop factor ., /] [term -> term mulop factor
212	[if-stmt -> if exp then stmt-sequence end else stmt-sequence .end , until] [stmt-sequence -> st
211	[if-stmt -> if exp then stmt-sequence end .else stmt-sequence end , ;] [if-stmt -> if exp then stn
210	[factor -> (exp .) , ;] [factor -> (exp .) , *] [factor -> (exp .) , ^] [factor -> (exp .) , +] [factor ->
209	[factor -> .number , *] [factor -> .identifier , /] [factor -> .(exp) , ^] [factor -> .number , %] [fa
208	[term -> term mulop .factor , ;] [term -> term mulop .factor , -] [term -> term mulop .factor , %
207	[if-stmt -> if exp then stmt-sequence end else stmt-sequence end ., ;] [if-stmt -> if exp then stn
206	[factor -> (exp) ., /] [factor -> (exp) ., %] [factor -> (exp) ., *] [factor -> (exp) ., ^] [factor ->



LR(1) DFA 状态表

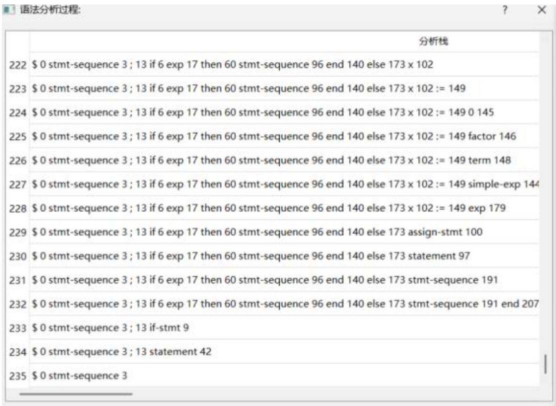
:=	;	<	<=	<>	=	>	>=	^	addop	assign-stmt	comparison-op	else	end	exp	factor	id
	139												219			
										100						10
								64								
	139												217			
												216				
															197	19
															213	19

⑥ LR1-分析表：

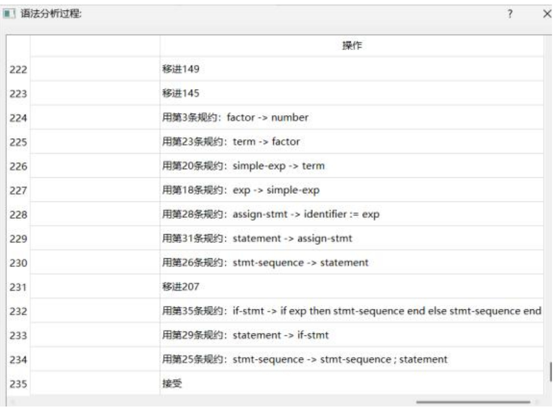


状态	$	assign-stmt	<	%	addop	end	^	mulop	term	>	<>	repeat	-	read	if-stmt	=
165																
164									g186							
163																
162			r22	r22			r22			r22	r22		r22			r2
161			r2	r2			r2			r2	r2		r2			r2
160			r19	s65			s64	g117		r19	r19		r19			r1
159				r4			r4						r4			
158				r3			r3						r3			
157				r23			r23						r23			
156									g49							
155					g183								s51			
154				s65			s64	g182					r20			
153						s181										
152																

⑦ 分析编码过程展示

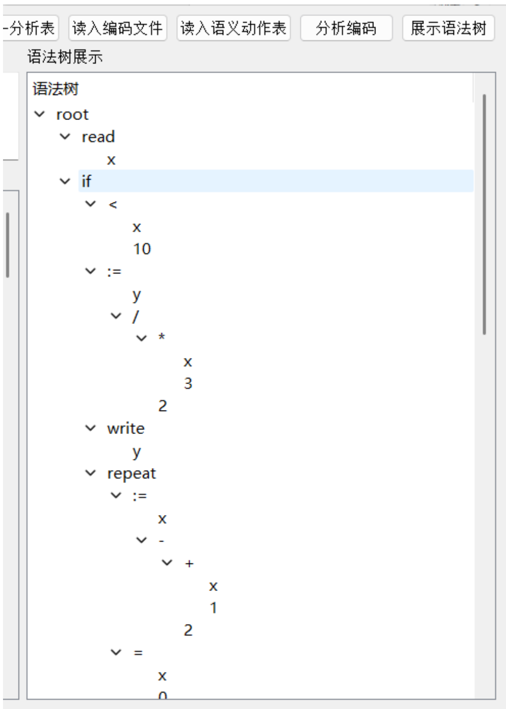


	分析栈
222	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102
223	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102 := 149
224	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102 := 149 0 145
225	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102 := 149 factor 146
226	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102 := 149 term 148
227	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102 := 149 simple-exp 144
228	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 x 102 := 149 exp 179
229	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 assign-stmt 100
230	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 statement 97
231	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 stmt-sequence 191
232	$ 0 stmt-sequence 3 ; 13 if 6 exp 17 then 60 stmt-sequence 96 end 140 else 173 stmt-sequence 191 end 207
233	$ 0 stmt-sequence 3 ; 13 if-stmt 9
234	$ 0 stmt-sequence 3 ; 13 statement 42
235	$ 0 stmt-sequence 3

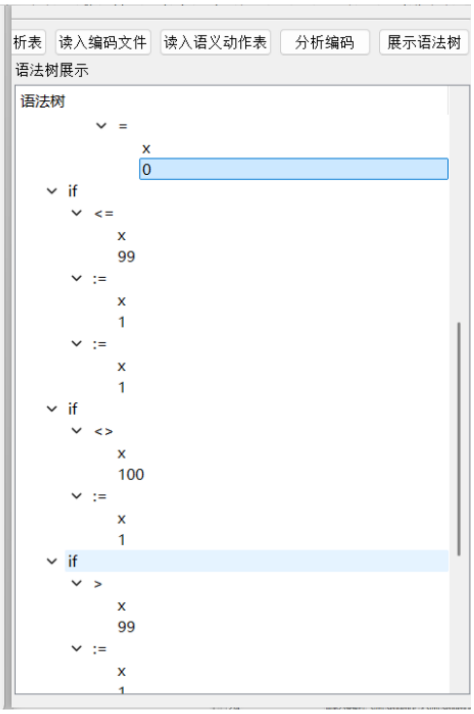


		操作
222		移进149
223		移进145
224		用第3条规约：factor -> number
225		用第23条规约：term -> factor
226		用第20条规约：simple-exp -> term
227		用第18条规约：exp -> simple-exp
228		用第28条规约：assign-stmt -> identifier := exp
229		用第31条规约：statement -> assign-stmt
230		用第26条规约：stmt-sequence -> statement
231		移进207
232		用第35条规约：if-stmt -> if exp then stmt-sequence end else stmt-sequence end
233		用第29条规约：statement -> if-stmt
234		用第25条规约：stmt-sequence -> stmt-sequence ; statement
235		接受

⑧ 语法树展示



```
语法树
root
  read
    x
  if
    <
      x
      10
    :=
      y
      /
        *
          x
          3
        2
    write
      y
    repeat
      :=
        x
        -
          +
            x
            1
          2
      =
        x
        0
```



```
语法树
      =
        x
        0
  if
    <=
      x
      99
    :=
      x
      1
    :=
      x
      1
  if
    <>
      x
      100
    :=
      x
      1
  if
    >
      x
      99
    :=
      x
      1
```

任务二测试完全通过

总结：根据我们最后展示的语法分析过程的结果，以及语法树的展示结果来看，我们本次任务一和任务二的测试是完全通过的。