项目测试报告

任务二

1、准备工作

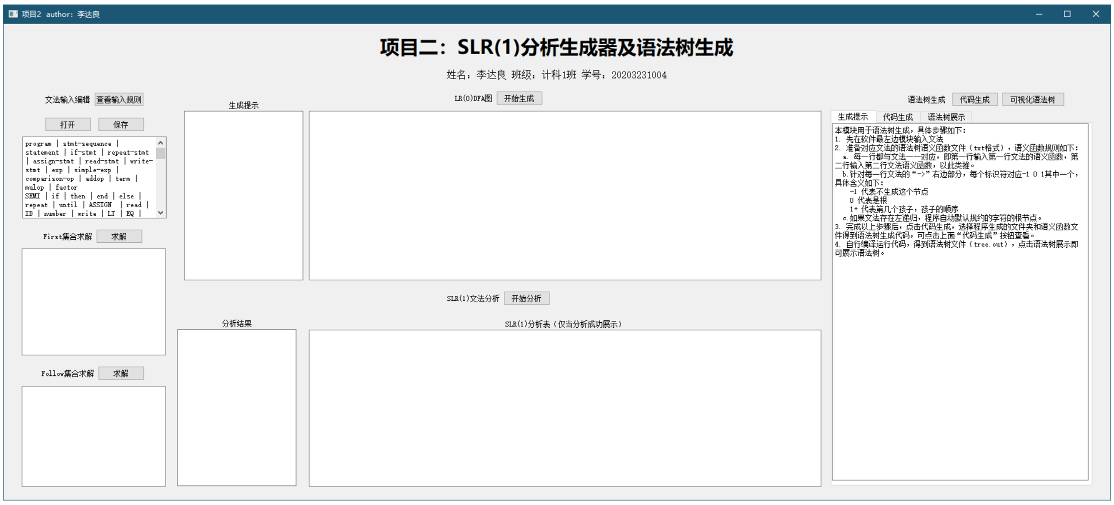
tiny 的文法：

```
program | stmt-sequence | statement | if-stmt | repeat-stmt | assign-stmt | read-stmt | write-stmt | exp | simple-exp | comparison-op | addop | term | mulop | factor
SEMI | if | then | end | else | repeat | until | ASSIGN  | read | ID | number | write | LT | EQ | LTEQ | NE | RTEQ | RT | PLUS | MINUS | MULTIPLY | DIVIDE | MOD | LPAN | RPAN | NUMBER
program -> stmt-sequence
stmt-sequence -> stmt-sequence SEMI statement
stmt-sequence -> statement
statement -> if-stmt
statement -> repeat-stmt
statement -> assign-stmt
statement -> read-stmt
statement -> write-stmt
if-stmt -> if exp then stmt-sequence end
if-stmt -> if exp then stmt-sequence else stmt-sequence end
repeat-stmt -> repeat stmt-sequence until exp
assign-stmt -> ID ASSIGN exp
read-stmt -> read ID
write-stmt -> write exp
exp -> simple-exp comparison-op simple-exp
exp -> simple-exp
comparison-op -> LT
comparison-op -> EQ
comparison-op -> LTEQ
comparison-op -> NE
comparison-op -> RTEQ
comparison-op -> RT
simple-exp -> simple-exp addop term
simple-exp -> term
addop -> PLUS
addop -> MINUS
term -> term mulop factor
term -> factor
mulop -> MULTIPLY
mulop -> DIVIDE
mulop -> MOD
factor -> LPAN exp RPAN
factor -> NUMBER
factor -> ID
```

语义函数：

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

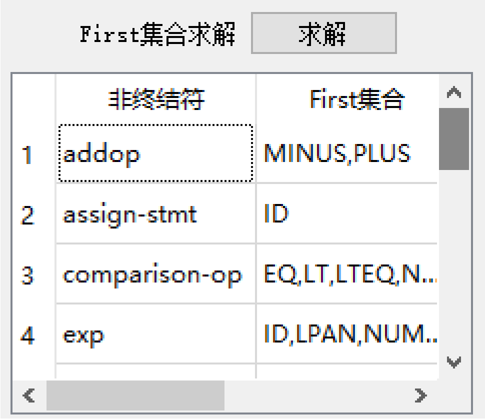
输入文法：



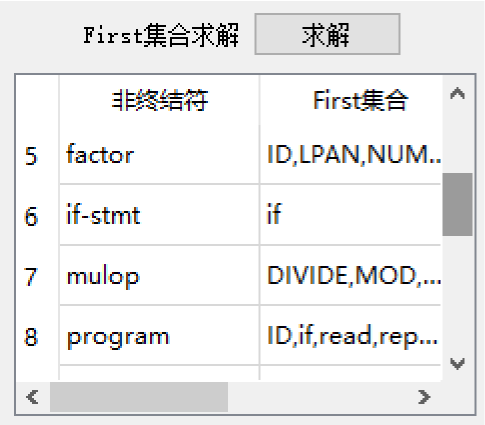
```
program | stmt-sequence |
statement | if-stmt | repeat-stmt
| assign-stmt | read-stmt | write-
stmt | exp | simple-exp |
comparison-op | addop | term |
mulop | factor
SEMI | if | then | end | else |
repeat | until | ASSIGN  | read |
ID | number | write | LT | EQ |
```

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。

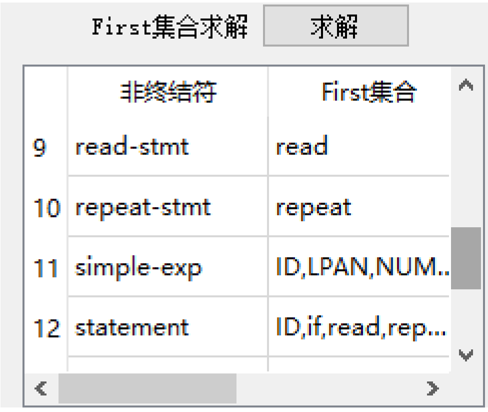
2、求解 first 集合



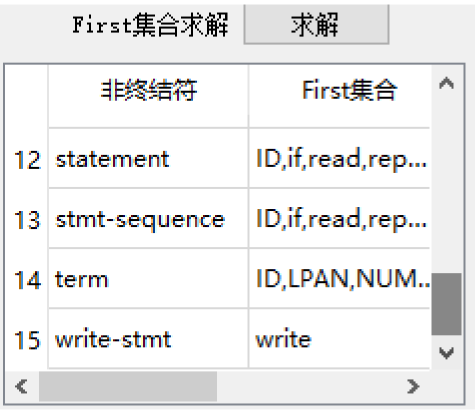
	非终结符	First集合
1	addop	MINUS,PLUS
2	assign-stmt	ID
3	comparison-op	EQ,LT,LTEQ,N...
4	exp	ID,LPAN,NUM...



	非终结符	First集合
5	factor	ID,LPAN,NUM...
6	if-stmt	if
7	mulop	DIVIDE,MOD,...
8	program	ID,if,read,rep...

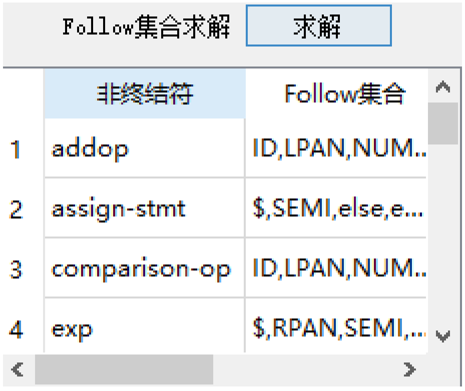


	非终结符	First集合
9	read-stmt	read
10	repeat-stmt	repeat
11	simple-exp	ID,LPAN,NUM...
12	statement	ID,if,read,rep...

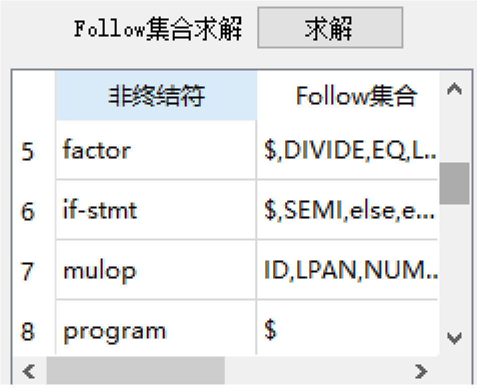


	非终结符	First集合
12	statement	ID,if,read,rep...
13	stmt-sequence	ID,if,read,rep...
14	term	ID,LPAN,NUM...
15	write-stmt	write

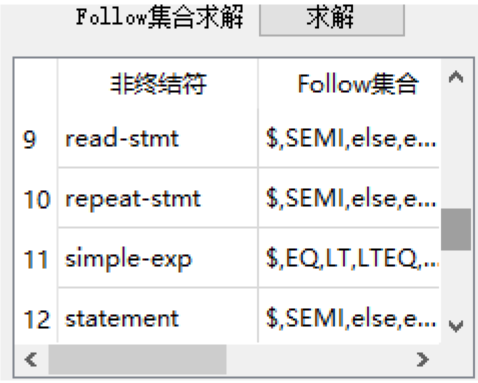
3、求解 follow 集合



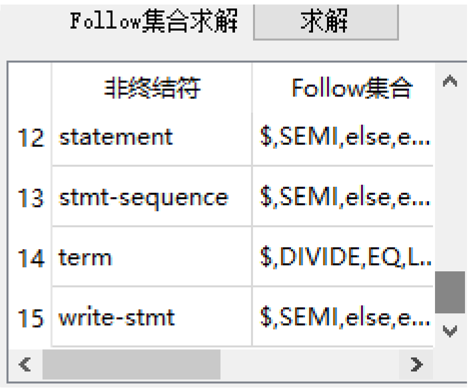
	非终结符	Follow集合
1	addop	ID,LPAN,NUM..
2	assign-stmt	$,SEMI,else,e...
3	comparison-op	ID,LPAN,NUM..
4	exp	$,RPAN,SEMI,...



	非终结符	Follow集合
5	factor	$,DIVIDE,EQ,L..
6	if-stmt	$,SEMI,else,e...
7	mulop	ID,LPAN,NUM..
8	program	$

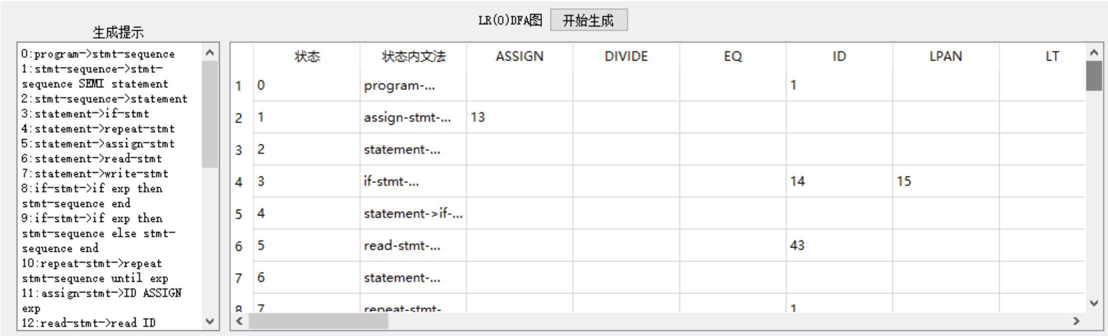


	非终结符	Follow集合
9	read-stmt	$,SEMI,else,e...
10	repeat-stmt	$,SEMI,else,e...
11	simple-exp	$,EQ,LT,LTEQ,...
12	statement	$,SEMI,else,e...



	非终结符	Follow集合
12	statement	$,SEMI,else,e...
13	stmt-sequence	$,SEMI,else,e...
14	term	$,DIVIDE,EQ,L..
15	write-stmt	$,SEMI,else,e...

4、LR0

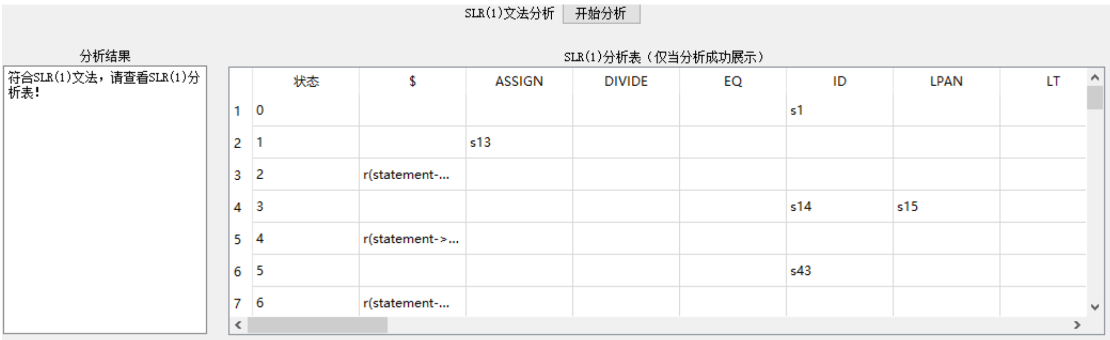


```
0:program->stmt-sequence
1:stmt-sequence->stmt-
sequence SEMI statement
2:stmt-sequence->statement
3:statement->if-stmt
4:statement->repeat-stmt
5:statement->assign-stmt
6:statement->read-stmt
7:statement->write-stmt
8:if-stmt->if exp then
stmt-sequence end
9:if-stmt->if exp then
stmt-sequence else stmt-
sequence end
10:repeat-stmt->repeat
stmt-sequence until exp
11:assign-stmt->ID ASSIGN
exp
12:read-stmt->read ID
```

	状态	状态内文法	ASSIGN	DIVIDE	EQ	ID	LPAN	LT
1	0	program-...				1		
2	1	assign-stmt-...	13					
3	2	statement-...						
4	3	if-stmt-...				14	15	
5	4	statement->if-...						
6	5	read-stmt-...				43		
7	6	statement-...						
8	7	repeat-stmt-				1		

项目太多，未能展示完全

5、SLR1 表



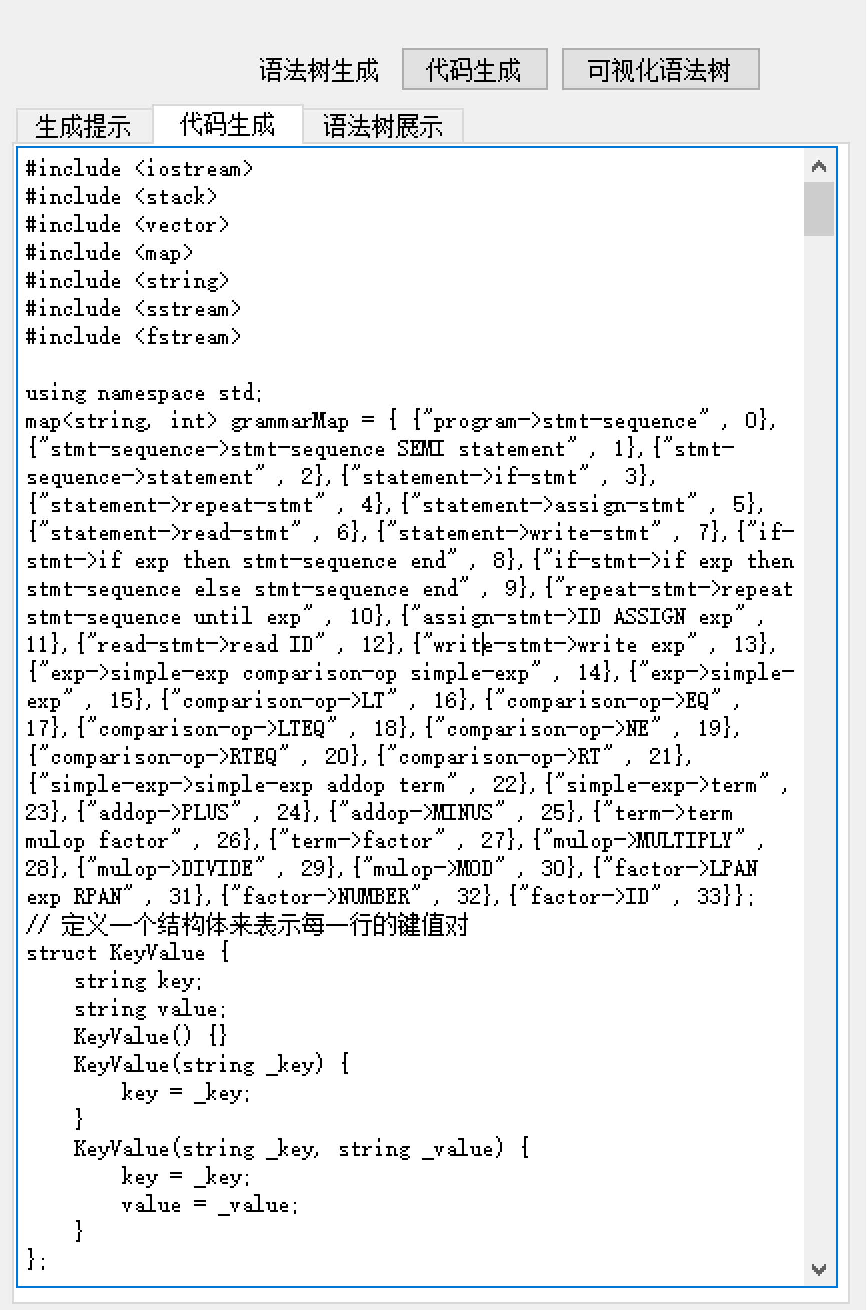
符合SLR(1)文法，请查看SLR(1)分析表!

	状态	$	ASSIGN	DIVIDE	EQ	ID	LPAN	LT
1	0					s1		
2	1		s13					
3	2	r(statement-...						
4	3					s14	s15	
5	4	r(statement->...						
6	5					s43		
7	6	r(statement-...						

项目太多，未能展示完全

6、语法树代码生成

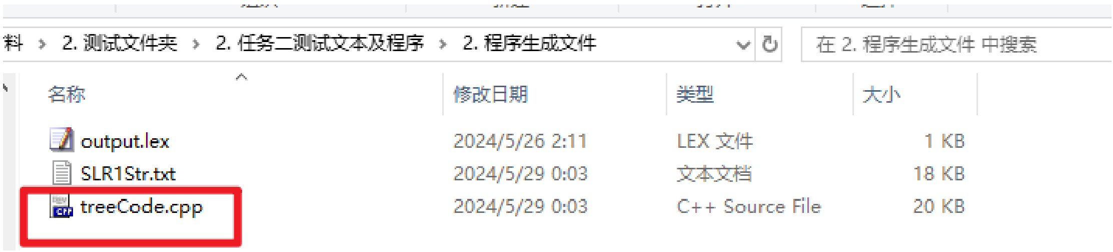
注意：lex 所在路径即为代码生成路径



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

using namespace std;
map<string, int> grammarMap = { {"program->stmt-sequence" , 0},
{"stmt-sequence->stmt-sequence SEMI statement" , 1},{"stmt-
sequence->statement" , 2},{"statement->if-stmt" , 3},
{"statement->repeat-stmt" , 4},{"statement->assign-stmt" , 5},
{"statement->read-stmt" , 6},{"statement->write-stmt" , 7},{"if-
stmt->if exp then stmt-sequence end" , 8},{"if-stmt->if exp then
stmt-sequence else stmt-sequence end" , 9},{"repeat-stmt->repeat
stmt-sequence until exp" , 10},{"assign-stmt->ID ASSIGN exp" ,
11},{"read-stmt->read ID" , 12},{"write-stmt->write exp" , 13},
{"exp->simple-exp comparison-op simple-exp" , 14},{"exp->simple-
exp" , 15},{"comparison-op->LT" , 16},{"comparison-op->EQ" ,
17},{"comparison-op->LTEQ" , 18},{"comparison-op->NE" , 19},
{"comparison-op->RTEQ" , 20},{"comparison-op->RT" , 21},
{"simple-exp->simple-exp addop term" , 22},{"simple-exp->term" ,
23},{"addop->PLUS" , 24},{"addop->MINUS" , 25},{"term->term
mulop factor" , 26},{"term->factor" , 27},{"mulop->MULTIPLY" ,
28},{"mulop->DIVIDE" , 29},{"mulop->MOD" , 30},{"factor->LPAN
exp RPAN" , 31},{"factor->NUMBER" , 32},{"factor->ID" , 33}};
// 定义一个结构体来表示每一行的键值对
struct KeyValue {
    string key;
    string value;
    KeyValue() {}
    KeyValue(string _key) {
        key = _key;
    }
    KeyValue(string _key, string _value) {
        key = _key;
        value = _value;
    }
};
```

7、查看语法树生成代码文件

PS:请用 C++11 以上进行编译



料 › 2. 测试文件夹 › 2. 任务二测试文本及程序 › 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
output.lex	2024/5/26 2:11	LEX 文件	1 KB
SLR1Str.txt	2024/5/29 0:03	文本文档	18 KB
treeCode.cpp	2024/5/29 0:03	C++ Source File	20 KB

具体语法树代码如下：

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

using namespace std;
map<string, int> grammarMap = { {"program->stmt-sequence" , 0},{"stmt-sequence->stmt-
sequence SEMI statement" , 1},{"stmt-sequence->statement" , 2},{"statement->if-
stmt" , 3},{"statement->repeat-stmt" , 4},{"statement->assign-stmt" ,
5},{"statement->read-stmt" , 6},{"statement->write-stmt" , 7},{"if-stmt->if exp then
stmt-sequence end" , 8},{"if-stmt->if exp then stmt-sequence else stmt-sequence
end" , 9},{"repeat-stmt->repeat stmt-sequence until exp" , 10},{"assign-stmt->ID
ASSIGN exp" , 11},{"read-stmt->read ID" , 12},{"write-stmt->write exp" ,
13},{"exp->simple-exp comparison-op simple-exp" , 14},{"exp->simple-exp" ,
15},{"comparison-op->LT" , 16},{"comparison-op->EQ" , 17},{"comparison-op->LTEQ" ,
18},{"comparison-op->NE" , 19},{"comparison-op->RTEQ" , 20},{"comparison-op->RT" ,
21},{"simple-exp->simple-exp addop term" , 22},{"simple-exp->term" ,
23},{"addop->PLUS" , 24},{"addop->MINUS" , 25},{"term->term mulop factor" ,
26},{"term->factor" , 27},{"mulop->MULTIPLY" , 28},{"mulop->DIVIDE" ,
29},{"mulop->MOD" , 30},{"factor->LPAN exp RPAN" , 31},{"factor->NUMBER" ,
32},{"factor->ID" , 33} };
// 定义一个结构体来表示每一行的键值对
struct KeyValue {
    string key;
    string value;
    KeyValue() {}
    KeyValue(string _key) {
        key = _key;
    }
    KeyValue(string _key, string _value) {
        key = _key;
        value = _value;
    }
};

// 函数读取并分隔每一行的键值对
vector<KeyValue> readKeyValuePairs(const string& filename) {
    ifstream file(filename);
    vector<KeyValue> keyValuePairs;

    if (!file) {
        return keyValuePairs;
    }

    string line;
    while (getline(file, line)) {
        // 使用 stringstream 将每一行分隔成键和值
        stringstream ss(line);
        string key, value;
        getline(ss, key, ':'); // 以 ':' 分隔键
        getline(ss, value);    // 获取剩下的作为值

        // 去除键和值两端的空格
        key.erase(0, key.find_first_not_of(" \t"));
        key.erase(key.find_last_not_of(" \t") + 1);
        value.erase(0, value.find_first_not_of(" \t"));
        value.erase(value.find_last_not_of(" \t") + 1);

        // 将键值对添加到向量中
        keyValuePairs.push_back({ key, value });
    }

    file.close();
    return keyValuePairs;
}

// 定义语法树节点结构
struct BTreeNode {
    string kind;
    string value;
    vector<BTreeNode*> nodeList;
    BTreeNode(string kind, string val) :kind(kind), value(val) {}
};

stack<BTreeNode*> treeStack;    // 树节点栈
stack<KeyValue> strStack; // 分析栈
stack<int> stateStack; // 状态栈
```

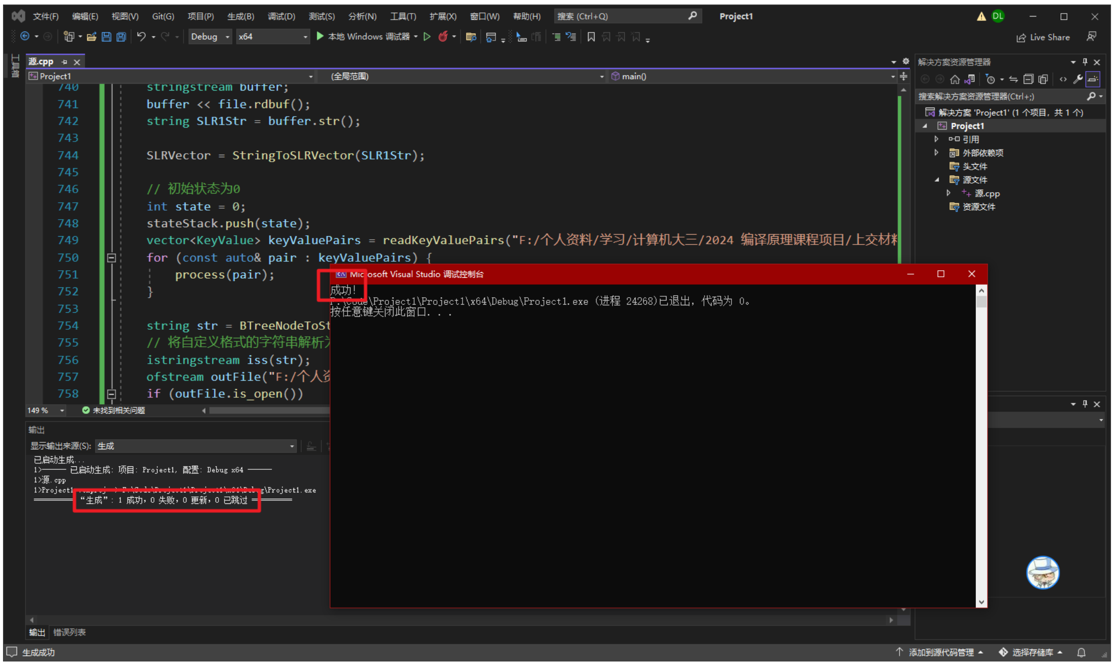
```
// 定义数据结构
struct SLRUnit
{
    string index;
    map<string, string> m;
};

vector<SLRUnit> SLRVector;

// 将自定义格式的字符串解析成 SLRVector
vector<SLRUnit> StringToSLRVector(const string& str)
{
    vector<SLRUnit> vec;
    SLRUnit unit;

    istringstream iss(str); // 创建一个输入流
    string line;

    int index = 0;

    while (getline(iss, line))
    {
        if (line == "SLRUnit")
        {
            unit = SLRUnit(); // 创建一个新的 SLRUnit
            unit.index = to_string(index);  // 标记序号
        }
        else if (line == "{")
        {
            // 忽略
        }
        else if (line == "}")
        {
            vec.push_back(unit); // 将完成的 SLRUnit 添加到 vector 中
        }
        else if (line.size() > 0)
        {
            // 解析键值对
            size_t pos = line.find(": ");
            if (pos != string::npos)
            {
                string key = line.substr(pos + 2); // 获取键的内容
                getline(iss, line); // 读取下一行，这一行应该是值
                pos = line.find(": ");
                if (pos != string::npos)
                {
                    string value = line.substr(pos + 2); // 获取值的内容
                    unit.m[key] = value; // 添加到当前 SLRUnit 中
                }
            }
        }
    }

    return vec;
}


// 将 BTreeNode 转换为自定义格式的字符串
string BTreeNodeToString(BTreeNode* node, int depth = 0)
{
    ostringstream oss; // 创建一个输出流
    string indent(depth * 4, ' '); // 缩进

    oss << indent << "BTreeNode\n";
    oss << indent << "{\n";
    oss << indent << "    kind: " << node->kind << "\n";
    oss << indent << "    value: " << node->value << "\n";

    if (!node->nodeList.empty())
    {
        oss << indent << "    nodeList:\n";
        for (const auto& child : node->nodeList)
        {
            oss << BTreeNodeToString(child, depth + 1); // 递归转换子节点
        }
    }

    oss << indent << "}\n";

    return oss.str(); // 返回输出流中的字符串
}
// program->stmt-sequence
string fun0() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();
    treeStack.push(newNode0);
    return "program";
}

// stmt-sequence->stmt-sequence SEMI statement
string fun1() {
    BTreeNode* newNode2 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol1 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode1 = new BTreeNode("SEMI", symbol1);

    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* rootNode = new BTreeNode("-1", "stmt-sequence");
    rootNode->nodeList.push_back(newNode0);
    rootNode->nodeList.push_back(newNode2);
    treeStack.push(rootNode);
    return "stmt-sequence";
}

// stmt-sequence->statement
string fun2() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "stmt-sequence";
}

// statement->if-stmt
string fun3() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "statement";
}

// statement->repeat-stmt
string fun4() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "statement";
}

// statement->assign-stmt
string fun5() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "statement";
}

// statement->read-stmt
string fun6() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "statement";
}

// statement->write-stmt
string fun7() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "statement";
}

// if-stmt->if exp then stmt-sequence end
string fun8() {
    string symbol4 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode4 = new BTreeNode("end", symbol4);

    BTreeNode* newNode3 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol2 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode2 = new BTreeNode("then", symbol2);

    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("if", symbol0);

    newNode0->nodeList.push_back(newNode1);
    newNode0->nodeList.push_back(newNode3);
    treeStack.push(newNode0);
    return "if-stmt";
}

// if-stmt->if exp then stmt-sequence else stmt-sequence end
string fun9() {
    string symbol6 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode6 = new BTreeNode("end", symbol6);

    BTreeNode* newNode5 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol4 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode4 = new BTreeNode("else", symbol4);

    BTreeNode* newNode3 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol2 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode2 = new BTreeNode("then", symbol2);

    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("if", symbol0);

    newNode0->nodeList.push_back(newNode1);
    newNode0->nodeList.push_back(newNode3);
    newNode0->nodeList.push_back(newNode5);
    treeStack.push(newNode0);
    return "if-stmt";
}

// repeat-stmt->repeat stmt-sequence until exp
string fun10() {
    BTreeNode* newNode3 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol2 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode2 = new BTreeNode("until", symbol2);

    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("repeat", symbol0);

    newNode0->nodeList.push_back(newNode1);
    newNode0->nodeList.push_back(newNode3);
    treeStack.push(newNode0);
    return "repeat-stmt";
}

// assign-stmt->ID ASSIGN exp
string fun11() {
    BTreeNode* newNode2 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol1 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode1 = new BTreeNode("ASSIGN", symbol1);

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("ID", symbol0);

    newNode1->nodeList.push_back(newNode0);
    newNode1->nodeList.push_back(newNode2);
    treeStack.push(newNode1);
    return "assign-stmt";
}

// read-stmt->read ID
string fun12() {
    string symbol1 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode1 = new BTreeNode("ID", symbol1);

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("read", symbol0);

    newNode0->nodeList.push_back(newNode1);
    treeStack.push(newNode0);
    return "read-stmt";
}

// write-stmt->write exp
string fun13() {
    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("write", symbol0);

    newNode0->nodeList.push_back(newNode1);
    treeStack.push(newNode0);
    return "write-stmt";
}

// exp->simple-exp comparison-op simple-exp
string fun14() {
    BTreeNode* newNode2 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();
    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    newNode1->nodeList.push_back(newNode0);
    newNode1->nodeList.push_back(newNode2);
    treeStack.push(newNode1);
    return "exp";
}

// exp->simple-exp
string fun15() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "exp";
}

// comparison-op->LT
string fun16() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("LT", symbol0);

    treeStack.push(newNode0);
    return "comparison-op";
}

// comparison-op->EQ
string fun17() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("EQ", symbol0);
    treeStack.push(newNode0);
    return "comparison-op";
}

// comparison-op->LTEQ
string fun18() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("LTEQ", symbol0);

    treeStack.push(newNode0);
    return "comparison-op";
}

// comparison-op->NE
string fun19() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("NE", symbol0);

    treeStack.push(newNode0);
    return "comparison-op";
}

// comparison-op->RTEQ
string fun20() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("RTEQ", symbol0);

    treeStack.push(newNode0);
    return "comparison-op";
}

// comparison-op->RT
string fun21() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("RT", symbol0);
    treeStack.push(newNode0);
    return "comparison-op";
}

// simple-exp->simple-exp addop term
string fun22() {
    BTreeNode* newNode2 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* rootNode = new BTreeNode("-1", "simple-exp");
    rootNode->nodeList.push_back(newNode1);
    rootNode->nodeList.push_back(newNode0);
    rootNode->nodeList.push_back(newNode2);
    treeStack.push(rootNode);
    return "simple-exp";
}

// simple-exp->term
string fun23() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "simple-exp";
}

// addop->PLUS
string fun24() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("PLUS", symbol0);

    treeStack.push(newNode0);
    return "addop";
}

// addop->MINUS
string fun25() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("MINUS", symbol0);

    treeStack.push(newNode0);
    return "addop";
}

// term->term mulop factor
string fun26() {
    BTreeNode* newNode2 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    BTreeNode* rootNode = new BTreeNode("-1", "term");
    rootNode->nodeList.push_back(newNode1);
    rootNode->nodeList.push_back(newNode0);
    rootNode->nodeList.push_back(newNode2);
    treeStack.push(rootNode);
    return "term";
}
```

```
// term->factor
string fun27() {
    BTreeNode* newNode0 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    treeStack.push(newNode0);
    return "term";
}

// mulop->MULTIPLY
string fun28() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("MULTIPLY", symbol0);

    treeStack.push(newNode0);
    return "mulop";
}

// mulop->DIVIDE
string fun29() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("DIVIDE", symbol0);

    treeStack.push(newNode0);
    return "mulop";
}

// mulop->MOD
string fun30() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("MOD", symbol0);

    treeStack.push(newNode0);
    return "mulop";
}
```

```
// factor->LPAN exp RPAN
string fun31() {
    string symbol2 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode2 = new BTreeNode("RPAN", symbol2);

    BTreeNode* newNode1 = treeStack.top();
    strStack.pop();
    stateStack.pop();
    treeStack.pop();

    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("LPAN", symbol0);

    treeStack.push(newNode1);
    return "factor";
}

// factor->NUMBER
string fun32() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("NUMBER", symbol0);

    treeStack.push(newNode0);
    return "factor";
}

// factor->ID
string fun33() {
    string symbol0 = strStack.top().value;
    strStack.pop();
    stateStack.pop();
    BTreeNode* newNode0 = new BTreeNode("ID", symbol0);

    treeStack.push(newNode0);
    return "factor";
}

// 定义一个存储函数指针的数组
```

```
string(*funcArray[])() =
{ fun0,fun1,fun2,fun3,fun4,fun5,fun6,fun7,fun8,fun9,fun10,fun11,fun12,fun13,fun14,fun15,fun16,fun17,fun18,fun19,fun20,fun21,fun22,fun23,fun24,fun25,fun26,fun27,fun28,fun29,fun30,fun31,fun32,fun33 };

void process(const KeyValue& line) {
    string key = line.key;
    string value = line.value;

    // 拿到当前栈的状态
    int state = stateStack.top();



    // 找到下一个状态字符串
    string nextStateStr = SLRVector[state].m[key == "EOF" ? "$" : key];
    int nextState;
    switch (nextStateStr[0]) {
    case 's':   // 下一步
        // 放入字符栈
        strStack.push(line);
        nextState = stoi(nextStateStr.substr(1));
        stateStack.push(nextState);
        break;
    case 'r':   // 要规约了
    {
        string res;
        size_t startPos = nextStateStr.find("("); // 查找左括号的位置
        size_t endPos = nextStateStr.find(")");   // 查找右括号的位置

        // 如果找到了左右括号
        if (startPos != string::npos && endPos != string::npos) {
            startPos++; // 从左括号的下一个位置开始提取内容
            res = nextStateStr.substr(startPos, endPos - startPos);
        }
        int i = grammarMap[res];
        string left = funcArray[i]();   // 调用对应的函数
        state = stateStack.top();
        nextStateStr = SLRVector[state].m[left];
        nextState = stoi(nextStateStr);
        stateStack.push(nextState);
        strStack.push(KeyValue(left));
        // 继续放入当前字符
        process(line);
        break;
    }
    case 'A':   // ACCEPT
        // 生成语法树
        cout << "成功！";
        break;
    default:
        cout << "状态表出错！";

    }

}

int main() {

    ifstream file("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/2.任务二测试文本及程序/2.程序生成文件/SLR1Str.txt");
    stringstream buffer;
    buffer << file.rdbuf();
    string SLR1Str = buffer.str();

    SLRVector = StringToSLRVector(SLR1Str);

    // 初始状态为0
    int state = 0;
    stateStack.push(state);
    vector<KeyValue> keyValuePairs = readKeyValuePairs("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/2.任务二测试文本及程序/2.程序生成文件/output.lex");
    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/2.任务二测试文本及程序/2.程序生成文件/tree.out");
    if (outFile.is_open())
    {
        outFile << str;
        outFile.close();
    }
    return 0;
}
```

编译运行：

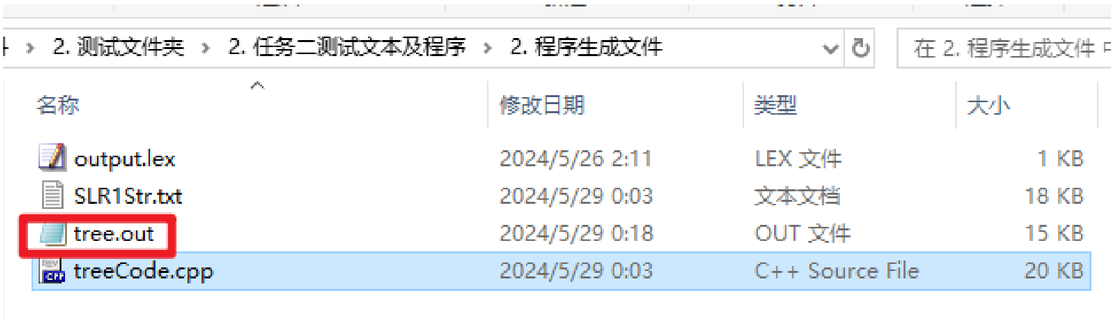


```
    stringstream buffer;
    buffer << file.rdbuf();
    string SLR1Str = buffer.str();

    SLRVector = StringToSLRVector(SLR1Str);

    // 初始状态为0
    int state = 0;
    stateStack.push(state);
    vector<KeyValue> keyValuePairs = readKeyValuePairs("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
    for (const auto& pair : keyValuePairs) {
        process(pair);
    }

    string str = BTreeNodeToSt
    // 将自定义格式的字符串解析为
    istringstream iss(str);
    ofstream outFile("F:/个人资
    if (outFile.is_open())
```

```
成功!
F:\Code\Project1\Project1\x64\Debug\Project1.exe (进程 24268)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 ==========
```

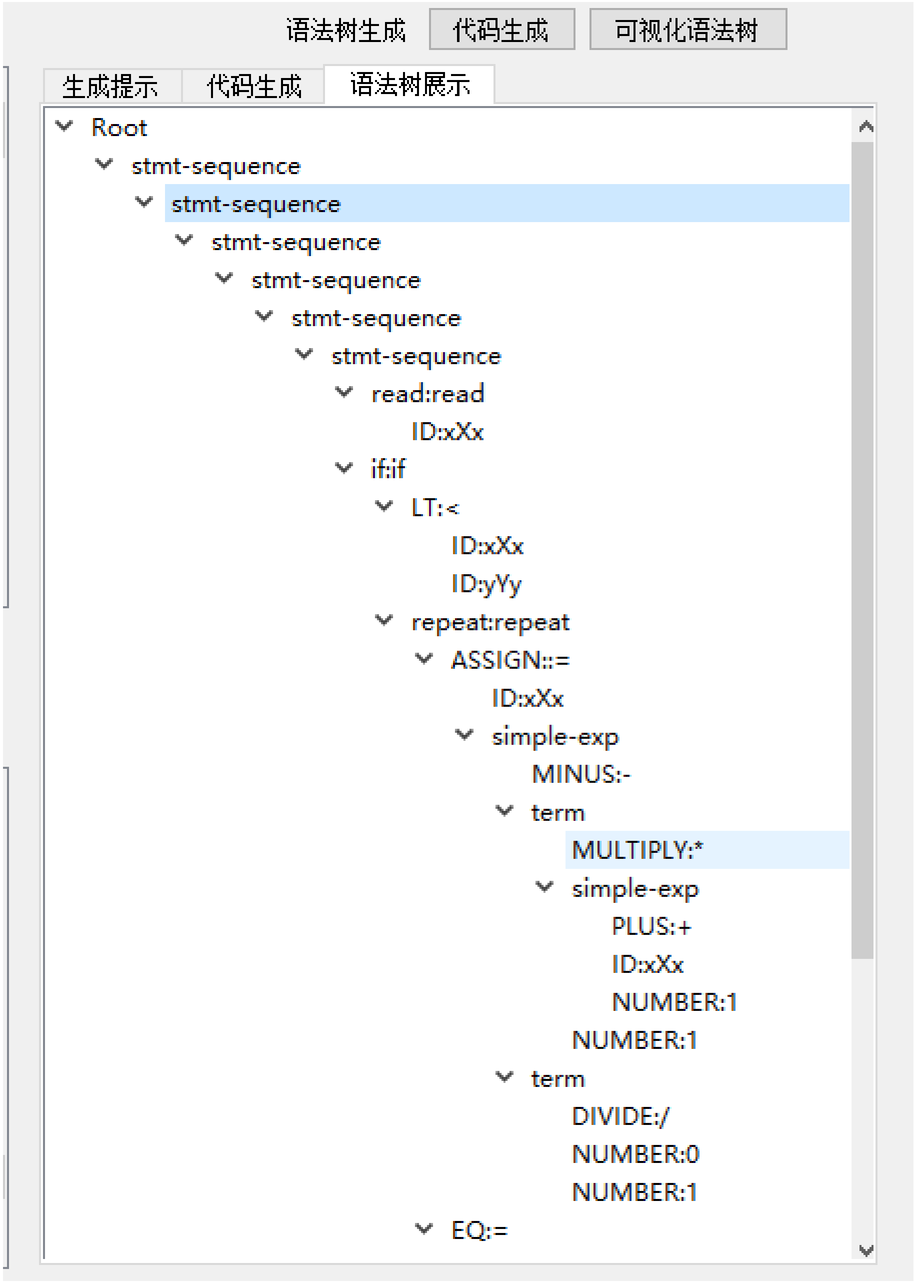
得到语法树：



2. 测试文件夹 › 2. 任务二测试文本及程序 › 2. 程序生成文件

名称	修改日期	类型	大小
output.lex	2024/5/26 2:11	LEX 文件	1 KB
SLR1Str.txt	2024/5/29 0:03	文本文档	18 KB
tree.out	2024/5/29 0:18	OUT 文件	15 KB
treeCode.cpp	2024/5/29 0:03	C++ Source File	20 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
                      MINUS:-
                      term
                        MULTIPLY:*
                        simple-exp
                          PLUS:+
                          ID:xXx
                          NUMBER:1
                        NUMBER:1
                      term
                        DIVIDE:/
                        NUMBER:0
                        NUMBER:1
                  EQ:=
```

测试结果

任务二测试完全通过