**编译原理实验报告**

实验名称：\_\_\_\_\_\_\_\_SLR(1)语法分析器\_\_\_\_\_\_\_\_

实验成绩：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

日期： 2021年 12月 12 日

**一、实验题目**

设计编制一个SLR(1)语法分析器。

**二、实验内容**

程序输入/输出示例（以下仅供参考）：  
对下列文法，用SLR（1）分析法对任意输入的符号串进行分析：

(0)S'→E

(1)E→E+T

(2)E→T

(3)T→T\*F

(4)T→F

(5)F→(E)

(6)F→i

  输出的格式如下：  
  (1)SLR（1）分析程序，编制人：姓名，学号，班级  
  (2)输入一个以#结束的符号串(包括+—\*/（）i#)：在此位置输入符号串  
  (3)输出过程如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 步骤 | 状态栈 | 符号栈 | 剩余输入串 | ACTION | GOTO |
| 1 | 0 | # | i+i\*i# | R5 |  |

 (4)输入符号串为非法符号串(或者为合法符号串)

  备注：

  1.在“所用产生式”一列中如果对应有推导则写出所用产生式；如果为匹配终结符则写明匹配的终结符；如分析异常出错则写为“分析出错”；若成功结束则写为“分析成功”。

  2.在此位置输入符号串为用户自行输入的符号串。

  注意：

  1.表达式中允许使用运算符（±\*/）、分割符（括号）、字符i，结束符#；

  2.如果遇到错误的表达式，应输出错误提示信息（该信息越详细越好）；

  3.对学有余力的同学，测试用的表达式事先放在文本文件中，一行存放一个表达式，同时以分号分割。同时将预期的输出结果写在另一个文本文件中，以便和输出进行对照。

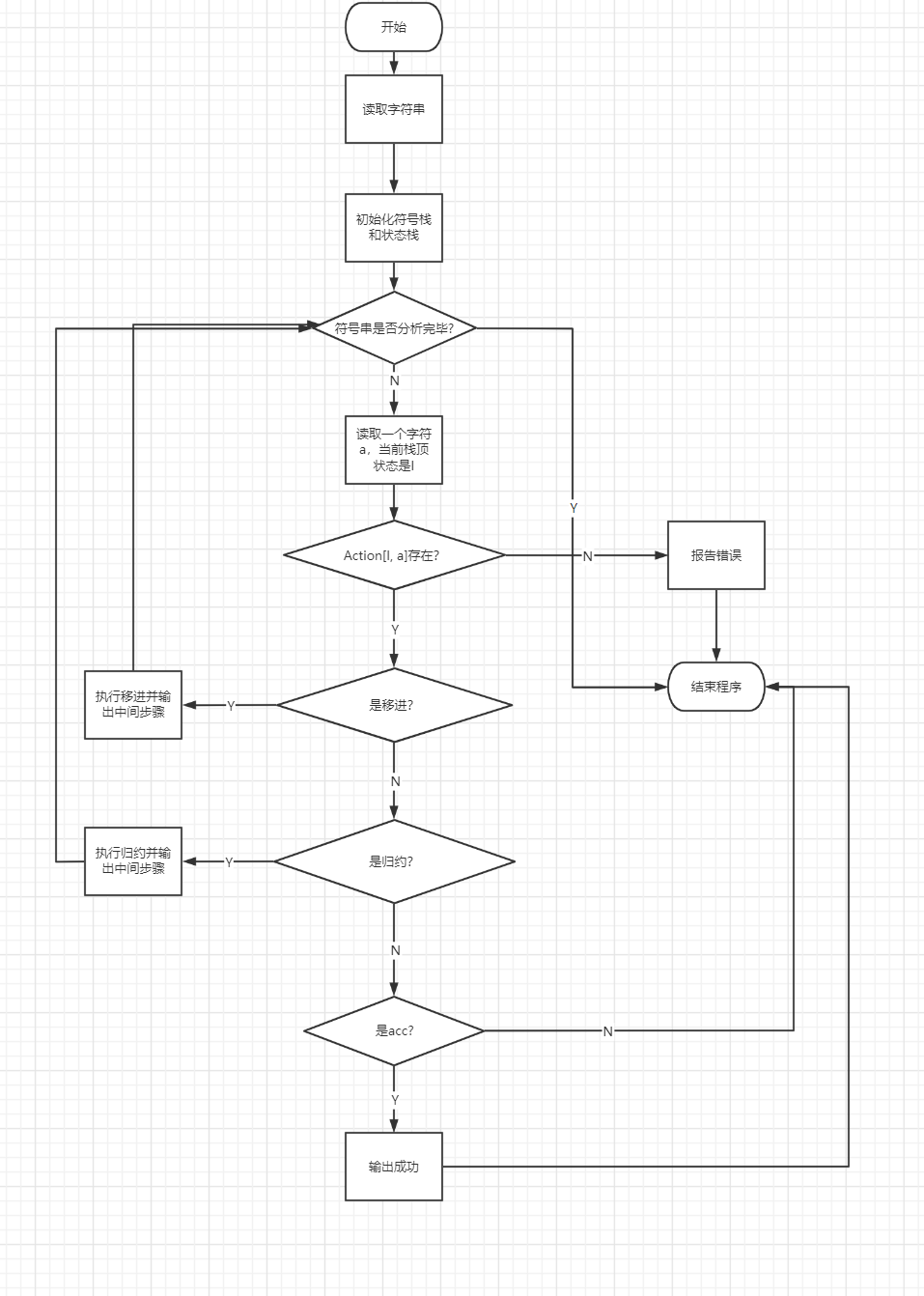
**三、实验步骤**

（包括基本设计思路、流程框图、算法设计、函数相关说明、输入与输出以及程序运行结果）

基本设计思路：首先将action表和Goto表通过数组的形式存储下来，使用0表示错误，设置两个栈，分别为状态栈和符号栈，同时将产生式集合存储到vector中，方便查询；这这里并没有真的将状态栈和符号栈使用真正的栈来存储，而是使用了vector模拟栈，这里主要考虑到要输出栈中的内容，因此需要可以随机访问栈里的内容，因此选择vector来模拟栈。

分析过程就是读入符号，查看action表，若没有动作，就报错，否则执行移入或者归约，移入操作就是将符号压入符号栈，更新状态栈，查看下一个字符，归约则需要找到对于的产生式，并计算产生式右部的长度，来弹出状态，并计算查看Goto表压入新的状态，以及更新符号栈，这个过程不会改变当前字符。

流程框图：



函数相关说明：

init:初始化状态站和符号栈，初始化当前输入字符指针和步骤数记录

error:输出错误

printStep:输出中间步骤

getPos:获取终结符或非终结符对应的索引

shift:进行移进操作

reduction:执行归约

analyse:入口函数，根据输入的字符串进行SLR(1)分析

程序运行结果： 自己运行 i+i\*i#和 i+(i+\*i\*)#

1. **实验心得体会**

通过本次实验，自己动手编写SLR(1)语法分析器，对SLR(1)语法分析过程有了更加深刻的理解，SLR(1)语法分析是一种自底向上的语法分析过程，通过对句子不断进行移进和归约从而归约到开始符号，则表示该句子可以被该文法接受，符号栈中的内容是一个可行前缀，如果当前符号栈中的内容没有句柄，则会继续移入，直到出现句柄，就会指行归约。SLR(1)相较于LR(0)对归约有了更严格的要求，要求可以归约的终结符要出现在产生式左部非终结符的FOLLOW集合中。

在具体编写中，并没有去自动计算FIRST集合FOLLOW集合LR(0)项目集构造，这些计算过程比较繁琐，但算法并不复杂，与手动计算过程无异，无非代码中要使用很多的循环来实现，这也是本次实验的不足之处，没有讲上述是三个过程进行计算，而是直接给出ACTION表和GOTO表。

**五、源程序清单（代码）**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <string>

#include <vector>

#include <stack>

#include <cstdlib>

using namespace std;

class SLR1

{

private:

string m\_actionTable[12][6] = {"S5", "0", "0", "S4", "0", "0",

"0", "S6", "0", "0", "0", "acc",

"0", "r2", "S7", "0", "r2", "r2",

"0", "r4", "r4", "0", "r4", "r4",

"S5", "0", "0", "S4", "0", "0",

"0", "r6", "r6", "0", "r6", "r6",

"S5", "0", "0", "S4", "0", "0",

"S5", "0", "0", "S4", "0", "0",

"0", "S6", "0", "0", "S11", "0",

"0", "r1", "S7", "0", "r1", "r1",

"0", "r3", "r3", "0", "r3", "r3",

"0", "r5", "r5", "0", "r5", "r5"};

int m\_gotoTable[12][3] = {1, 2, 3,

0, 0, 0,

0, 0, 0,

0, 0, 0,

8, 2, 3,

0, 0, 0,

0, 9, 3,

0, 0, 10,

0, 0, 0,

0, 0, 0,

0, 0, 0,

0, 0, 0};

string m\_VT = "i+\*()#";

string m\_VN = "ETF";

vector<string> m\_production = {"E->E+T", "E->T", "T->T\*F", "T->F", "F->(E)", "F->i"};

vector<int> m\_stateStack;

vector<char> m\_symbolStack;

int m\_curPos;

int m\_step;

string m\_target;

char m\_errorMsg[100];

void init(string &s)

{

m\_target = s;

m\_stateStack.push\_back(0);

m\_symbolStack.push\_back('#');

m\_curPos = 0;

m\_step = 1;

}

void error()

{

cout << "Error\t" << m\_errorMsg << endl;

exit(0);

}

void printStep(int stateNum, int flag = 0)

{

cout << m\_step++ << '\t';

for (int i = 0; i < m\_stateStack.size(); i++)

cout << m\_stateStack[i];

cout << "\t";

for (int i = 0; i < m\_symbolStack.size(); i++)

cout << m\_symbolStack[i];

cout << '\t';

cout << m\_target.substr(m\_curPos) << '\t';

if (flag == 0)

cout << "\tS" << stateNum << endl;

else if (flag == 1)

{

// 执行归约

reduction(stateNum);

cout << "\tr" << stateNum << '\t';

if (m\_stateStack.back() == 0)

error();

else cout << m\_stateStack.back() << endl;

}

else if (flag == 2)

cout << "\tacc" << endl;

else

error();

}

int getPos(char ch, bool isVT = true)

{

string::size\_type res;

if (isVT)

res = m\_VT.find(ch);

else

res = m\_VN.find(ch);

if (res == string::npos)

{

sprintf(m\_errorMsg, "The input symbol string contains illegal characters!");

error();

}

return res;

}

void shift(int stateNum)

{

char symbol = m\_target[m\_curPos++];

m\_stateStack.push\_back(stateNum);

m\_symbolStack.push\_back(symbol);

}

void reduction(int stateNum)

{

string production = m\_production[stateNum - 1];

int len = production.size() - 3;

for (int i = 0; i < len; i++)

{

m\_stateStack.pop\_back();

m\_symbolStack.pop\_back();

}

m\_symbolStack.push\_back(production[0]);

int pos = getPos(production[0], false);

int topState = m\_stateStack.back();

int newState = m\_gotoTable[topState][pos];

// goto不存在

if (newState == 0)

sprintf(m\_errorMsg, "Goto(%d, %c) is null!", topState, production[0]);

m\_stateStack.push\_back(newState);

}

public:

void analyse(string &s)

{

init(s);

while (m\_curPos < m\_target.size())

{

int t = getPos(m\_target[m\_curPos]);

int i = m\_stateStack.back();

if (m\_actionTable[i][t] == "acc")

{

printStep(i, 2);

break;

}

// action不存在

else if (m\_actionTable[i][t] == "0")

{

sprintf(m\_errorMsg, "Action(%d, %d) is null!", i, t);

printStep(i, 3);

break;

}

else if (m\_actionTable[i][t][0] == 'S')

{

int stateNum = 0;

for (int j = 1; j < m\_actionTable[i][t].size(); j++)

stateNum = stateNum \* 10 + m\_actionTable[i][t][j] - '0';

printStep(stateNum, 0);

shift(stateNum);

}

else

{

int stateNum = 0;

for (int j = 1; j < m\_actionTable[i][t].size(); j++)

stateNum = stateNum \* 10 + m\_actionTable[i][t][j] - '0';

// 归约过程放在打印中,因为需要输出Goto值和操作前的栈状态

printStep(stateNum, 1);

}

}

}

};

string s;

int main()

{

SLR1 analyer;

cout << endl << "SLR(1)分析程序,编制人:偏偏喜欢你" << endl;

cout << "输入一个以#结束的符号串(包括+—\*/()i#):" << endl;

cin >> s;

cout << "输出过程如下:" << endl;

cout << "步骤"

<< "\t"

<< "状态栈"

<< "\t"

<< "符号栈"

<< "\t"

<< "剩余输入串"

<< "\t"

<< "Action"

<< "\t"

<< "Goto" << endl;

analyer.analyse(s);

return 0;

}