

编译原理

**实验名称**

学 院： 詹天佑

专 业： 计算机科学与技术

学生姓名： 程维森

学 号： 21231264

北京交通大学

2023年12月

1. **程序功能描述**

以下是程序的主要组成部分和功能概述：

**组件：**

语法规则：

语法规则在gra数组中定义。

它包括诸如 A->S，S->V=E 等规则。

解析表：

解析表用action和Goto数组表示。

action包含基于当前状态和输入符号的移入（s）和归约（r）动作。

Goto包含非终结符号的状态转移。

解析函数：

check(): 根据输入字符串和语法规则执行解析过程。

performReduction(): 处理解析过程中的规约步骤。

printStacks()，printlist(): 用于打印解析过程中的堆栈内容和当前状态的函数。

GUI 组件：

程序使用FLTK（快速轻量级工具包）构建了一个简单的图形用户界面（GUI）。

GUI包括一个供用户输入句子的输入字段和一个触发语法分析的按钮。

**功能：**

语法分析：

用户在GUI中输入字符串，触发语法分析。

程序根据提供的语法规则和输入字符串执行LR(0)分析。

它遵循LR(0)分析算法，使用解析表（移入-归约动作和Goto转换）来确定输入字符串是否符合语法。

GUI 交互：

create\_gui(): 设置FLTK窗口、输入字段和分析按钮。

grammarAnalysis(): 从GUI中获取输入字符串，启动语法分析过程，并显示结果。

控制台输出：

程序在解析过程中向控制台（stdout）输出调试和分析信息。

错误处理：

程序处理了一些基本错误，比如遇到不正确的句子或无法接受输入。

**总体而言：**

该程序将一个简单的LR(0)分析器实现与基本的GUI用户交互结合在一起。它根据提供的语法规则对输入字符串进行解析，并在控制台中显示分析结果。GUI允许用户输入需要进行语法分析的句子。

1. **主要数据结构**

|  |  |
| --- | --- |
| 变量及类型 | 用途 |
| vector<string> gra | 存储语法规则 |
| struct node { int num; char c; } action[110][110] | 存储动作表，包含动作编号和操作符类型 |
| int Goto[110][110] | 存储Goto表的状态转移信息 |
| string s | 全局变量，存储输入的字符串 |
| map<char, int>Vtcot | 映射终结符到列数的映射表 |
| map<char, int>Vncot | 映射非终结符到列数的映射表 |
| vector<int>stack1 | 存储状态栈 |
| vector<char>stack2 | 存储符号栈 |

gra 存储语法规则的向量。

action 存储动作表，其中每个元素是一个结构体，包含动作编号和操作符类型。

Goto 存储Goto表的状态转移信息。

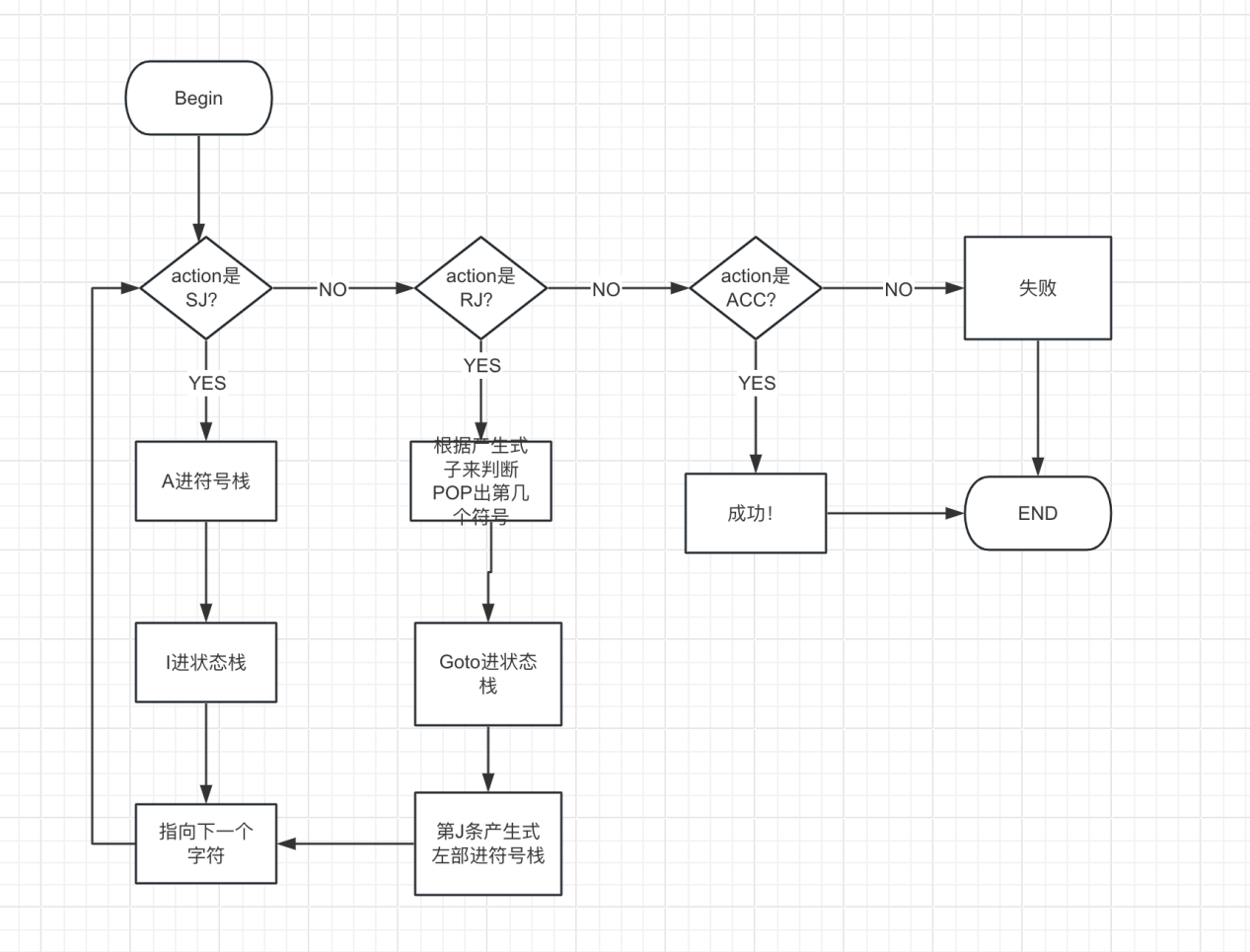
s 是一个全局变量，用于存储输入的字符串。

Vtcot 和 Vncot 是映射表，将终结符和非终结符映射到对应的列数。

stack1 和 stack2 分别用于存储状态栈和符号栈，在语法分析过程中用于存储状态和符号。

1. **程序结构描述**
2. **设计方法**

程序流程图如图所示：



程序流程图

1. **函数定义及函数间的调用关系**

|  |  |
| --- | --- |
| 函数名称 | 函数功能描述 |
| void init() | 初始化动作表、Goto表和映射表等数据结构 |
| void printStacks() | 打印状态栈、符号栈和剩余字符串的内容 |
| void performReduction() | 处理规约步骤，修改状态栈和符号栈 |
| void printlist() | 打印状态栈、符号栈和剩余字符串的内容，表示规约动作 |
| void check() | 执行LR(0)语法分析，检查输入字符串的语法正确性 |
| void grammarAnalysis() | 从GUI获取输入字符串，进行语法分析，并输出结果 |
| void create\_gui() | 创建FLTK图形界面和相关控件，设置GUI功能和回调函数 |
| int main() | 主函数，程序入口，进行初始化并启动FLTK事件循环 |

main() 函数是程序入口，在其中调用 init() 进行初始化，然后调用 create\_gui() 创建图形界面和相关控件，并启动 FLTK 事件循环。

create\_gui() 函数内部设置了一个按钮回调函数，回调函数内部调用 grammarAnalysis() 函数进行语法分析。

grammarAnalysis() 函数从 GUI 获取输入字符串，然后调用 check() 函数执行语法分析。

check() 函数内部根据输入字符串执行 LR(0)语法分析，在解析过程中调用了 performReduction()、printStacks()、printlist() 等函数进行状态栈和符号栈的处理和打印。

performReduction()、printStacks()、printlist() 函数主要被 check() 函数内部调用，用于执行规约步骤和打印解析过程中的信息。

1. **程序测试**

测试用例1：

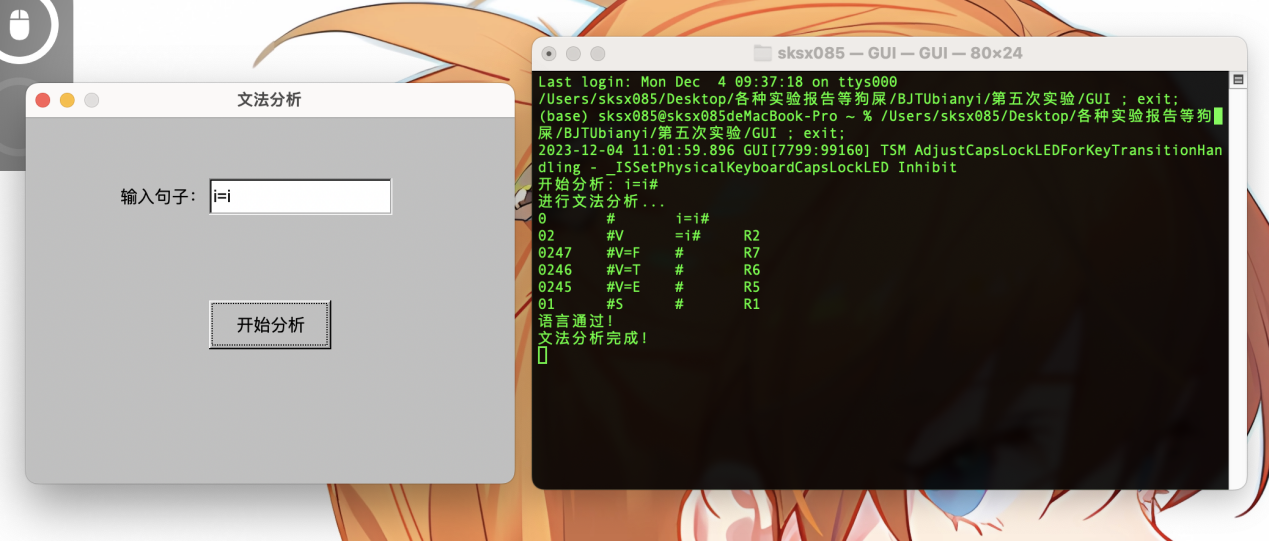
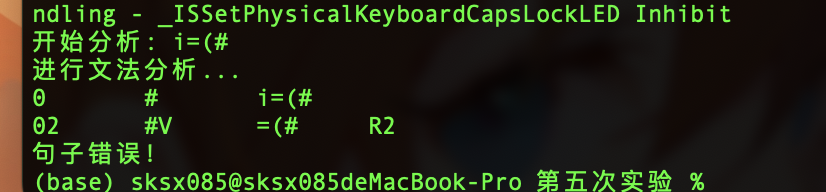
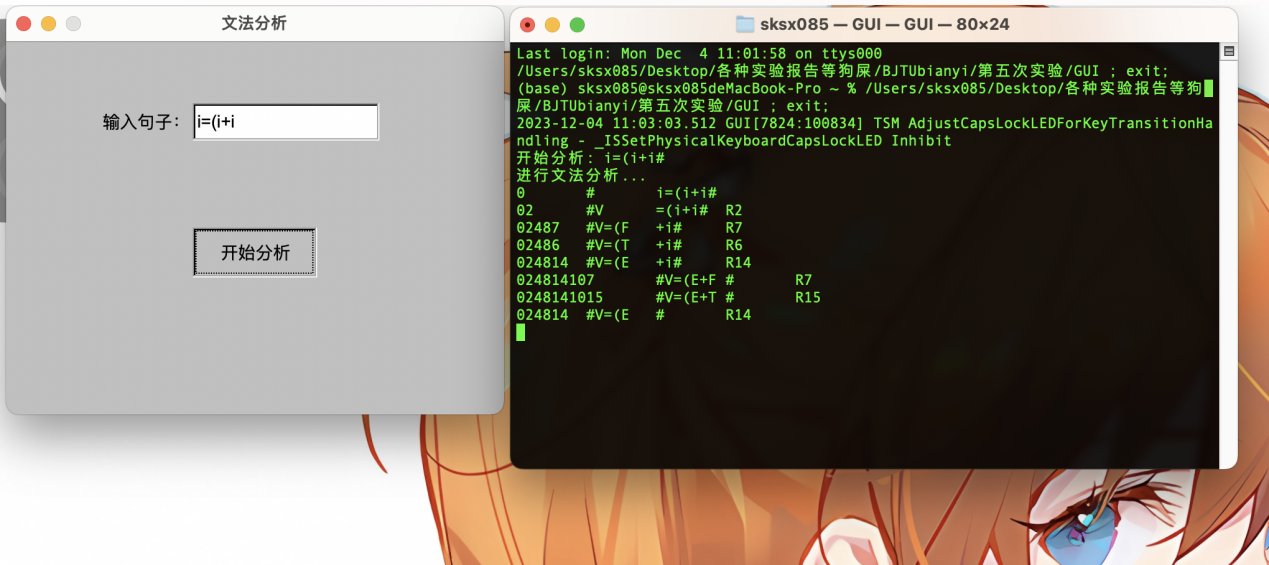


图4.1 测试用例1

测试用例2：



1. 附件（源代码列表）

#include <FL/Fl.H>

#include <FL/Fl\_Window.H>

#include <FL/Fl\_Button.H>

#include <FL/Fl\_Input.H>

#include <FL/Fl\_Output.H>

#include <iostream>

#include <stack>

#include <vector>

#include <map>

using namespace std;

vector<string> gra;

struct node

{

int num;

char c;

} action[110][110];

int Goto[110][110];

string s;

map<char, int> Vtcot;

map<char, int> Vncot;

int flag = 0;

void init()

{

for (int i = 0; i < 110; i++)

for (int j = 0; j < 110; j++)

action[i][j].num = -1;

for (int i = 0; i < 110; i++)

for (int j = 0; j < 110; j++)

Goto[i][j] = -1;

Vtcot['='] = 0;

Vtcot['+'] = 1;

Vtcot['-'] = 2;

Vtcot['\*'] = 3;

Vtcot['/'] = 4;

Vtcot['('] = 5;

Vtcot[')'] = 6;

Vtcot['i'] = 7;

Vtcot['#'] = 8;

Vncot['A'] = 0;

Vncot['S'] = 1;

Vncot['E'] = 2;

Vncot['T'] = 3;

Vncot['F'] = 4;

Vncot['V'] = 5;

gra.push\_back("A->S");

gra.push\_back("S->V=E");

gra.push\_back("E->E+T");

gra.push\_back("E->E-T");

gra.push\_back("E->T");

gra.push\_back("T->T\*F");

gra.push\_back("T->T/F");

gra.push\_back("T->F");

gra.push\_back("F->(E)");

gra.push\_back("F->i");

gra.push\_back("V->i");

action[0][7].c = 's', action[0][7].num = 3;

action[1][8].c = 'r', action[1][8].num = 0;

action[2][0].c = 's', action[2][0].num = 4;

action[3][0].c = 'r', action[3][0].num = 10;

action[4][5].c = 's', action[4][5].num = 8;

action[4][7].c = 's', action[4][7].num = 9;

action[5][1].c = 's', action[5][1].num = 10;

action[5][2].c = 's', action[5][2].num = 11;

action[5][8].c = 'r', action[5][8].num = 1;

action[6][1].c = 'r', action[6][1].num = 4;

action[6][2].c = 'r', action[6][2].num = 4;

action[6][3].c = 's', action[6][3].num = 12;

action[6][4].c = 's', action[6][4].num = 13;

action[6][6].c = 'r', action[6][6].num = 4;

action[6][8].c = 'r', action[6][8].num = 4;

action[7][1].c = 'r', action[7][1].num = 7;

action[7][2].c = 'r', action[7][2].num = 7;

action[7][3].c = 'r', action[7][3].num = 7;

action[7][4].c = 'r', action[7][4].num = 7;

action[7][6].c = 'r', action[7][6].num = 7;

action[7][8].c = 'r', action[7][8].num = 7;

action[8][7].c = 's', action[8][7].num = 9;

action[9][1].c = 'r', action[9][1].num = 9;

action[9][2].c = 'r', action[9][2].num = 9;

action[9][3].c = 'r', action[9][3].num = 9;

action[9][4].c = 'r', action[9][4].num = 9;

action[9][6].c = 'r', action[9][6].num = 9;

action[9][8].c = 'r', action[9][8].num = 9;

action[10][5].c = 's', action[10][5].num = 8;

action[10][7].c = 's', action[10][7].num = 9;

action[11][5].c = 's', action[11][5].num = 8;

action[11][7].c = 's', action[11][7].num = 9;

action[12][5].c = 's', action[12][5].num = 8;

action[12][7].c = 's', action[12][7].num = 9;

action[13][5].c = 's', action[13][5].num = 8;

action[13][7].c = 's', action[13][7].num = 9;

action[14][1].c = 's', action[14][1].num = 10;

action[14][2].c = 's', action[14][2].num = 11;

action[14][6].c = 's', action[14][6].num = 19;

action[15][1].c = 'r', action[15][1].num = 2;

action[15][2].c = 'r', action[15][2].num = 2;

action[15][3].c = 's', action[15][3].num = 12;

action[15][4].c = 's', action[15][4].num = 13;

action[15][6].c = 'r', action[15][6].num = 2;

action[15][8].c = 'r', action[15][8].num = 2;

action[16][1].c = 'r', action[16][1].num = 3;

action[16][2].c = 'r', action[16][2].num = 3;

action[16][3].c = 's', action[16][3].num = 12;

action[16][4].c = 's', action[16][4].num = 13;

action[16][6].c = 'r', action[16][6].num = 3;

action[16][8].c = 'r', action[16][8].num = 3;

action[17][1].c = 'r', action[17][1].num = 5;

action[17][2].c = 'r', action[17][2].num = 5;

action[17][3].c = 'r', action[17][3].num = 5;

action[17][4].c = 'r', action[17][4].num = 5;

action[17][6].c = 'r', action[17][6].num = 5;

action[17][8].c = 'r', action[17][8].num = 5;

action[18][1].c = 'r', action[18][1].num = 6;

action[18][2].c = 'r', action[18][2].num = 6;

action[18][3].c = 'r', action[18][3].num = 6;

action[18][4].c = 'r', action[18][4].num = 6;

action[18][6].c = 'r', action[18][6].num = 6;

action[18][8].c = 'r', action[18][8].num = 6;

action[19][1].c = 'r', action[19][1].num = 8;

action[19][2].c = 'r', action[19][2].num = 8;

action[19][3].c = 'r', action[19][3].num = 8;

action[19][4].c = 'r', action[19][4].num = 8;

action[19][6].c = 'r', action[19][6].num = 8;

action[19][8].c = 'r', action[19][8].num = 8;

Goto[0][1] = 1;

Goto[0][5] = 2;

Goto[4][2] = 5;

Goto[4][3] = 6;

Goto[4][4] = 7;

Goto[8][2] = 14;

Goto[8][3] = 6;

Goto[8][4] = 7;

Goto[10][3] = 15;

Goto[10][4] = 7;

Goto[11][3] = 16;

Goto[11][4] = 7;

Goto[12][4] = 17;

Goto[13][4] = 18;

}

void printStacks(const vector<int> &stack1, const vector<char> &stack2, const string &s)

{

for (int j = 0; j < stack1.size(); j++)

{

cout << stack1[j];

}

cout << "\t";

for (int j = 0; j < stack2.size(); j++)

{

cout << stack2[j];

}

cout << "\t" << s << "\n";

}

void performReduction(vector<int> &stack1, vector<char> &stack2, const vector<string> &gra, int num1, int &i)

{

char c1 = stack2.back();

for (int k = 3; k < gra[action[stack1.back()][num1].num].size(); k++)

{

if (stack2[stack2.size() - (gra[action[stack1.back()][num1].num].size() - k)] != gra[action[stack1.back()][num1].num][k])

{

cout << "失败！未被接纳！\n";

exit(-1);

}

}

int tmp = stack1.back();

int temp = gra[action[stack1.back()][num1].num].size();

for (int k = 3; k < temp; k++)

{

stack2.pop\_back();

stack1.pop\_back();

}

stack2.push\_back(gra[action[tmp][num1].num][0]);

stack1.push\_back(Goto[stack1.back()][Vncot[stack2.back()]]);

}

void printlist(vector<int> &stack1, int &i, vector<char> &stack2)

{

for (int j = 0; j < stack1.size(); j++)

{

cout << stack1[j];

}

cout << "\t";

for (int j = 0; j < stack2.size(); j++)

{

cout << stack2[j];

}

cout << "\t";

cout << s.substr(i) << "\t"

<< "R" << stack1.back() << "\n";

}

void check()

{

vector<int> stack1;

stack1.push\_back(0);

vector<char> stack2;

stack2.push\_back('#');

int i = 0;

int num = 1;

while (true)

{

if (num == 1)

printStacks(stack1, stack2, s.substr(i));

num++;

if (stack2.size() == 2 && stack2[1] == 'S' && s[i] == '#')

{

cout << "语言通过！\n";

return;

}

int num1 = Vtcot[s[i]];

if (action[stack1.back()][num1].num != -1)

{

if (action[stack1.back()][num1].c == 's')

{

stack1.push\_back(action[stack1.back()][num1].num);

stack2.push\_back(s[i]);

i++;

}

else if (action[stack1.back()][num1].c == 'r')

{

performReduction(stack1, stack2, gra, num1, i);

printlist(stack1, i, stack2);

}

else

{

cout << "句子错误！\n";

exit(-1);

}

}

else

{

cout << "句子错误！\n";

exit(-1);

}

}

}

void grammarAnalysis(Fl\_Input \*input, Fl\_Output \*output)

{

s = input->value();

s += '#'; // 将输入的文本赋值给全局变量 s，并添加终止符号 '#'

cout << "开始分析: " << s << endl;

cout << "进行文法分析..." << endl;

check();

cout << "文法分析完成！" << endl;

exit(0);

}

void create\_gui()

{

Fl\_Window \*window = new Fl\_Window(400, 300, "文法分析");

Fl\_Input \*input = new Fl\_Input(150, 50, 150, 30, "输入句子：");

Fl\_Button \*button = new Fl\_Button(150, 150, 100, 40, "开始分析");

button->callback([](Fl\_Widget \*widget, void \*data)

{

Fl\_Input \*input = (Fl\_Input \*)data;

Fl\_Output \*output = (Fl\_Output \*)widget->parent()->child(2); // 获取输出控件

grammarAnalysis(input, output); },

input);

window->end();

window->show();

}

int main()

{

init();

Fl::lock(); // 确保多线程安全性

create\_gui();

return Fl::run();

}

// g++ -std=c++11 -I /usr/local/include -o GUI GUI.cpp -L /usr/local/lib -lfltk -lX11 -lXext -lm

// export LIBRARY\_PATH=/opt/X11/lib:$LIBRARY\_PATH