**黑龙江大学**

**实 验 报 告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **课程名称** | 软件构造 | | | | | | |
| **实验项目名称** | 基于预测分析法的语法分析程序的构造 | | | | | | |
| **实验时间**  **（日期及节次）** | 4月25日5-6节 | | | | | | |
| **专业** | 软件工程 | | **学生所在学院** | | | 软件学院 | |
| **年级** | 2022级 | | **学号** | | | 20225958 | |
| **姓名** | 李嘉富 | | **指导教师** | | | 徐为 | |
| **实验室名称** | 4-108 | | | | | | |
| **实验成绩** | **预习情况** | **操作技术** | | **实验报告** | **附加：综合创新能力** | | **实验**  **综合成绩** |
|  |  | |  |  | |  |
| **教师签字** |  | | | | | | |

**黑龙江大学教务处**

1. 实验目的

1、掌握LL（1）分析法的基本原理。

2、掌握FIRST集、FOLLOW集、SELECT集的计算方法和过程。

3、掌握预测分析表的构造方法。

4、掌握使用预测分析表进行语法分析的过程。

二、实验环境

编程环境：Visual Studio 2022

运行环境: Visual Studio 2022

三、实验内容

对于某个给定的表达式文法（满足LL（1）文法的条件），使用高级语言（c、c++、java、python）编写程序，完成下列内容：

1、计算文法中每个文法符号以及每个产生式右部符号串的FIRST集。

2、计算文法中每个非终结符的FOLLOW集。

3、计算文法中每个产生式的SELECT集。

4、构造预测分析表设计算法编程实现判断输入的符号串是否满足文法定义。

实验通过求出first，follow，select集，进而得出预测分析表，再通过预测分析表进行符号串的分析，通过各种函数，如使用set集，哈希表等，采用了字符串以及数组等数据结构。主要思想为利用课堂上学习的构造定理，求出各种集合，以构造出表。

Grammar.txt文件如下：

E->TG

G->+TG

G->~

T->FS

S->\*FS

S->~

F->(E)

F->i

四、实验结果（测试用例、实验结果）

pat：（存储预测分析表）

VN/VT # ( ) \* + i

E ERR ->TG ERR ERR ERR ->TG

F ERR ->(E) ERR ERR ERR ->i

G ->~ ERR ->~ ERR ->+TG ERR

S ->~ ERR ->~ ->\*FS ->~ ERR

T ERR ->FS ERR ERR ERR ->FS

result：（存储对于文法的分析过程）

步骤 分析式 剩余输入串 所用产生式

1 #E i+i\*i#

2 #GT i+i\*i# E->TG

3 #GSF i+i\*i# T->FS

4 #GSi i+i\*i# F->i

5 #GS +i\*i# i匹配

6 #G +i\*i# S->~

7 #GT+ +i\*i# G->+TG

8 #GT i\*i# +匹配

9 #GSF i\*i# T->FS

10 #GSi i\*i# F->i

11 #GS \*i# i匹配

12 #GSF\* \*i# S->\*FS

13 #GSF i# \*匹配

14 #GSi i# F->i

15 #GS # i匹配

16 #G # S->~

17 # # G->~

五、实验总结

在这次实验中，我学会了许多新的知识，首先由于需要书写的功能过多，我运用了很多我以前没有用过的函数，如set， map等，让我极大提高了在书写代码时的便利性。在写代码的同时，我也又一次的复习运用了我所学的知识，帮助我在接下来的学习和考试中能够熟练的运用这些知识。这次实验教会我许多，希望以后还有机会能够进行整体的实现。

附录（源代码）

题目1源代码：

void pretreatment() {

// 遍历文法哈希表

for (auto iter = grammar.begin(); iter != grammar.end(); ++iter) {

// 符号分类

Vn.insert(iter->first); // 非终结符集

// 终结符集合

string str = iter->second;

for (size\_t i = 0; i < str.length(); i++){

if (str[i] != '|' && (str[i] < 'A' || str[i] > 'Z')){

Vt.insert(string(1, str[i]));

}

}

}

cout << endl << "符号分类" << endl;

// 输出终结符和非终结符集合

cout << "开始符号：" << S << endl;

cout << "非终结符集Vn = " << "{";

for (auto iter = Vn.begin(); iter != Vn.end(); ) {

cout << \*iter;

if ((++iter) != Vn.end()){

cout << ",";

}

}

cout << "}" << endl;

cout << "终结符集Vt = " << "{";

for (auto iter = Vt.begin(); iter != Vt.end(); ) {

cout << \*iter;

if ((++iter) != Vt.end()){

cout << ",";

}

}

cout << "}" << endl;

}

题目2源代码：

// 求某一个非终结符的First集合

set<char> findFirstBV(string vn) {

set<char> results; // first集存储

vector<string> rights = getRights(vn); // 获取右部

if (!rights.empty()) {// 如果右部不为空

// 遍历右部集合（每一个右部分别求解first，加入到该非终结符的first集合中）

for (auto iter = rights.begin(); iter != rights.end(); ++iter) {

string right = \*iter;

// 遍历当前右部： //如果第一个字符是终结符，加入first集合并且跳出循环；

//如果是非终结符，则递归；

//如果非终结符可以推空还需要循环处理该右部的下一字符

for (auto ch = right.begin(); ch != right.end(); ++ch) {

if (isVn(\*ch)){ // 如果是非终结符，就要递归处理

//先查first集合。如果已经有了就不需要重复求解

if (FIRST.find(string(1, \*ch)) == FIRST.end()){ // fisrt集合中不存在

// 递归调用

set<char> chars = findFirstBV(string(1, \*ch));

results.insert(chars.begin(), chars.end());

FIRST.insert(pair<string, set<char>>(string(1, \*ch), chars));

}

else { //把该集合全部加到firsts

set<char> chars = FIRST[string(1, \*ch)];

results.insert(chars.begin(), chars.end());

}

// 如果这个字符可以推空，且后面还有字符，那么还需要处理下一个字符

if (canToEmpty(\*ch) && (iter + 1) != rights.end()){

continue;

}

else

break; // 否则直接退出遍历当前右部的循环

}

else { // 如果不是非终结符，直接把这个字符加入first集合，并且跳出

// 这一步会把前面的空也加进去

results.insert(\*ch);

break;

}

}

}

}

// 最后，如果该终结符不能推空，就删除空

if (!canToEmpty(vn[0])){

results.erase('~');//删除集合中的~

}

return results;

}

// 求解First集，使用哈希表存储

void findFirst() {

// 遍历非终结符集合，构建哈希表

for (auto iter = Vn.begin(); iter != Vn.end(); ++iter) {

string vn = \*iter; // 获取非终结符

set<char> firsts = findFirstBV(vn); // 存放一个Vn的first集

FIRST.insert(pair<string, set<char>>(vn, firsts));

}

// 显示输出

cout << endl << "FISRT集分析" << endl;

for (auto iter = FIRST.begin(); iter != FIRST.end(); ++iter) {

cout << "FIRST(" << iter->first << ")" << "= ";

set<char> sets = iter->second;

printSet(sets);

}

}

// 单个非终结符符求解其Follow集合

set<char> findFollowBV(string vn) {

set<char> results; // 存储求解结果

if (vn == S) {// 如果是开始符号

results.insert('#'); // 把结束符加进去，因为有语句#S#

}

// 遍历文法所有的右部集合

for (auto iter = formulas.begin(); iter != formulas.end(); ++iter){

string right = (\*iter).substr(3); // 获取当前右部

// 遍历当前右部，看是否含有当前符号

for (auto i\_ch = right.begin(); i\_ch != right.end();){

if (\*i\_ch == vn[0]) { // 如果vn出现在了当前右部

if ((i\_ch + 1) == right.end()){ // vn是当前右部最后一个字符

results.insert('#'); // 加入结束符

break;

}

else { // vn后面还有字符,遍历

while (i\_ch != right.end()){

++i\_ch;// 指针后移

if (\*i\_ch == vn[0]){

--i\_ch;

break;

}

if (isVn(\*i\_ch)){ // 如果该字符是非终结符，把first集中的非空元素加进去

set<char> tmp\_f = FIRST[string(1, \*i\_ch)];

tmp\_f.erase('~'); // 除去空

results.insert(tmp\_f.begin(), tmp\_f.end());

// 还要该字符可否推空，需要考虑是否继续循环

if (canToEmpty(\*i\_ch)){

if ((i\_ch + 1) == right.end()){ // 如果是最后一个字符，加入#

results.insert('#');

break;// 跳出循环

}

// 继续循环

}

else // 否则跳出循环

break;

}

else { // 如果该字符是终结符

results.insert(\*(i\_ch)); // 加入该字符

break; // 跳出循环

}

}

}

}

else {

++i\_ch;

}

}

}

return results;

}

// 完善Follow集合

void completeFollow(string vn) {

// 遍历文法所有的右部集合

for (auto iter = formulas.begin(); iter != formulas.end(); ++iter){

string right = (\*iter).substr(3); // 获取当前右部

// 遍历当前右部，看是否含有当前符号

for (auto i\_ch = right.begin(); i\_ch != right.end();){

char vn\_tmp = \*i\_ch;

if (vn\_tmp == vn[0]) { // 如果vn出现在了当前右部

if ((i\_ch + 1) == right.end()) {// vn是当前右部最后一个字符

char left = (\*iter)[0];

set<char> tmp\_fo = FOLLOW[string(1, left)]; // 获取左部的follow集合

set<char> follows = FOLLOW[string(1, vn\_tmp)]; // 获取自己的原来的follow集合

follows.insert(tmp\_fo.begin(), tmp\_fo.end());

FOLLOW[vn] = follows; // 修改

break;

}

else { // 不是最后一个字符，就要遍历之后的字符看是否可以推空

while (i\_ch != right.end()){

++i\_ch; // 指针后移

if (canToEmpty(\*i\_ch)){

if ((i\_ch + 1) != right.end()){ // 不是最后一个元素

continue;

}

else { // 最后一个也能推空,则把左部加进去

char left = (\*iter)[0];

set<char> tmp\_fo = FOLLOW[string(1, left)]; // 左部的follow集合

set<char> follows = FOLLOW[string(1, vn\_tmp)]; // 当前符号的follow集合

follows.insert(tmp\_fo.begin(), tmp\_fo.end());

FOLLOW[vn] = follows;

break;

}

}

else // 如果不能推空，就退出循环

break;

}

}

}

++i\_ch; // 遍历寻找vn是否出现

}

}

}

// 求解Follow集，使用哈希表存储

void findFollow() {

// 遍历所有非终结符，依次求解follow集合

for (auto iter = Vn.begin(); iter != Vn.end(); ++iter) {

string vn = \*iter; // 获取非终结符

set<char> follows = findFollowBV(vn); // 求解一个Vn的follow集

FOLLOW.insert(pair<string, set<char>>(vn, follows)); // 存储到哈希表

}

// 完善follow集合直到follow不再增大

int old\_count = getFS();

int new\_count = -1;

while (old\_count != new\_count) {// 终结符在变化，反复这个过程直到follow集合不再增大

old\_count = getFS();

// 再次遍历非终结符，如果出现在右部最末端的，把左部的follow集加进来

for (auto iter = Vn.begin(); iter != Vn.end(); ++iter) {

string vn = \*iter; // 获取非终结符

completeFollow(vn);

}

new\_count = getFS();

}

// 显示输出

cout << endl << setfill('=') << "FOLLOW集分析" << endl;

for (auto iter = FOLLOW.begin(); iter != FOLLOW.end(); ++iter) {

cout << "FOLLOW(" << iter->first << ")" << "= ";

set<char> sets = iter->second;

printSet(sets);

}

}

// 单个表达式求解Select集合

set<char> findSelectBF(string formula) {

set<char> results; // 存储结果

// 得到产生式的left和right

char left = formula[0]; // 左部

string right = formula.substr(3); // 右部

// 遍历右部产生式，首先分析右部第一个字符:right[0]

for (auto iter = right.begin(); iter != right.end(); ++iter){

// 如果是非终结符：把first(right[0])-'~'加入到results；如果还可以推空，则要继续continue

if (isVn(\*iter)){

set<char> chs = FIRST.find(string(1, \*iter))->second; // 得到该符号的first、

chs.erase('~'); // 去除空符

results.insert(chs.begin(), chs.end()); // 加入select

if (canToEmpty(\*iter)){ // 如果可以推空，继续处理下一个字符加入到select集合

if ((iter + 1) == right.end()){ // 当前是最后一个字符，则把follow(left)加入results,然后break

set<char> chs = FOLLOW.find(string(1, left))->second; // 得到左部的follow

results.insert(chs.begin(), chs.end()); // 加入select

}

else { // 继续处理下一字符

continue;

}

}

else

break; // 该字符不可以推空，退出循环

}

else {// 如果是终结符：（如果为空符，则把follow(left)加入results，否则直接把该符号加入到results）,然后break

if (\*iter == '~') {// 如果是空

set<char> chs = FOLLOW.find(string(1, left))->second; // 得到左部的follow

results.insert(chs.begin(), chs.end()); // 加入select

}

else

results.insert(\*iter); // 直接加入select

break; // 退出循环

}

}

return results;

}

// 求解Select集，使用哈希表存储

void findSelect() {

// 遍历表达式集合

for (auto iter = formulas.begin(); iter != formulas.end(); ++iter) {

string formula = \*iter; // 获取表达式

set<char> selects = findSelectBF(formula); // 存放一个Vn的first集

SELECT.insert(pair<string, set<char>>(formula, selects)); // 插入到哈希表，提高查询效率

}

// 显示输出

cout << endl << "SELECT集分析" << endl;

for (auto iter = SELECT.begin(); iter != SELECT.end(); ++iter) {

cout << "SELECT(" << iter->first << ")" << "= ";

set<char> sets = iter->second;

printSet(sets);

}

}

题目3源代码：

// 判断是否为LL(1)分析

void isLL1() {

for (auto i1 = SELECT.begin(); i1 != SELECT.end(); ++i1){

for (auto i2 = SELECT.begin(); i2 != SELECT.end(); ++i2){

char left1 = (i1->first)[0]; // 获取左部2

char left2 = (i2->first)[0]; // 获取左部2

if (left1 == left2){ // 左部相等

if (i1->first != i2->first){ //表达式不一样

if (isIntersect(i1->second, i2->second)) { // 如果select集合有交集

// 不是LL1文法

cout << "不符合LL(1)文法，请重试" << endl;

exit(0);

}

}

}

}

}

// 是LL（1）文法

cout << "符合LL(1)文法" << endl;

}

// 构造预测分析表

void makeTable() {

cout << "构造分析表" << endl;

// 遍历select集合，对于键：分为left和->right;

// 对于值，遍历后单个字符ch：把left和ch配对作为TABLE的键，而->right作为值

char left\_ch;

string right;

set<char> chars;

for (auto iter = SELECT.begin(); iter != SELECT.end(); ++iter){ // 遍历select集合

left\_ch = iter->first[0]; // 获取左部

right = iter->first.substr(1); // 获取->右部

chars = iter->second;

//一个个放入

for (char ch : chars) { // 遍历终结符

TABLE.insert(pair<pair<char, char>, string>(pair<char, char>(left\_ch, ch), right));

}

}

// 输出分析表

printTable();

}

题目4源代码：

// 分析字符串

void LL1Analyse() {

cout << "构造完成，请输入您要分析的字符串" << endl;

ofstream ofs;

ofs.open("result.txt", ios::out);

string str; // 输入串

cin >> str;

str.push\_back('#'); // 末尾加入结束符

/\*cout << setw(16) << left << setfill(' ') << "步骤";

cout << setw(16) << left << setfill(' ') << "分析栈";

cout << setw(16) << left << setfill(' ') << "剩余输入串";

cout << setw(16) << left << setfill(' ') << "分析情况" << endl;\*/

ofs << setw(12) << left << setfill(' ') << "步骤";

ofs << setw(12) << left << setfill(' ') << "分析式";

ofs << setw(12) << left << setfill(' ') << "剩余输入串";

ofs << setw(12) << left << setfill(' ') << "所用产生式"<< endl;

stack<char> stack\_a; // 分析栈

stack\_a.push('#'); // 末尾符进栈

stack\_a.push(S[0]); // 开始符号进栈

// 初始化显示数据

int step = 1; // 步骤数

stack<char> stack\_remain = stack\_a; // 剩余分析栈

string str\_remain = str; // 剩余分析串

string str\_situation = "待分析"; // 分析情况

// 初始数据显示

/\*cout << setw(16) << left << setfill(' ') << step;

cout << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

cout << setw(16) << left << setfill(' ') << str\_remain << endl;\*/

ofs << setw(16) << left << setfill(' ') << step;

ofs << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

ofs << setw(16) << left << setfill(' ') << str\_remain << endl;

// 遍历所输入的句子，一个个字符分析

for (auto iter = str.begin(); iter != str.end();) {

char a = \*iter; // 当前终结符送a

char X = stack\_a.top(); // 栈顶元素送X

if (isVt(X)) {// 如果X是Vt终结符，栈顶元素出栈，然后读取下一个字符

if (X == a){ // 和输入字符匹配

stack\_a.pop(); // 移除栈顶元素

// 从剩余分析串中移除本元素

for (auto i\_r = str\_remain.begin(); i\_r != str\_remain.end(); i\_r++){

if (\*i\_r == a) {

str\_remain.erase(i\_r);

break; // 只删除第一个,

}

}

// 重新组装提示字符串

string msg = string(1, a) + "匹配";

str\_situation = msg;

// 读取下一个字符

++iter;

}

else { // 无法匹配，分析出错

cout << "分析出错:" << X << "和" << a << "不匹配" << endl;

exit(-1); // 出错退出

}

}

else if (X == '#'){ // 文法分析结束

if (a == '#'){ // 当前符号也是最后一个符号,接受分析结果

cout << "分析结束" << endl;

exit(0);

}

else {

cout << "分析出错，文法结束输入串未结束" << endl;

exit(-1);

}

}

else { // X 就是非终结符了

// 查看TABLE（X，a）是否有结果

if (TABLE.find(pair<char, char>(X, a)) == TABLE.end()) {//如果找不到

if (!canToEmpty(X)){ // 也不能是空

cout << "分析出错,找不到表达式" << endl;

exit(-1); // 失败退出

}

else { // 可以是空，

stack\_a.pop(); // 移除栈顶元素

str\_situation.clear();

str\_situation.push\_back(X);

str\_situation = str\_situation + "->";

str\_situation = str\_situation + "~";

}

}

else {

stack\_a.pop();// 先将当前符号出栈

string str = TABLE.find(pair<char, char>(X, a))->second.substr(2); // 获取表达式并且逆序进栈(除去->)

// 重新组装字符串

str\_situation.clear();

str\_situation.push\_back(X);

str\_situation = str\_situation + "->";

str\_situation = str\_situation + str;

reverse(str.begin(), str.end());

for (auto iiter = str.begin(); iiter != str.end(); ++iiter){

if (\*iiter != '~'){

stack\_a.push(\*iiter);

}

}

//继续识别该字符

}

}

// 重置显示数据

++step; // 步骤数加1

stack\_remain = stack\_a; // 置剩余栈为当前栈

// 每次循环显示一次

/\*cout << setw(16) << left << setfill(' ') << step;

cout << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

cout << setw(16) << left << setfill(' ') << str\_remain;

cout << setw(16) << left << setfill(' ') << str\_situation << endl;\*/

ofs << setw(16) << left << setfill(' ') << step;

ofs << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

ofs << setw(16) << left << setfill(' ') << str\_remain;

ofs << setw(16) << left << setfill(' ') << str\_situation << endl;

}

ofs.close();

}

总源代码：

#include <iostream>

#include <iomanip>

#include <stdio.h>

#include <fstream> // 文件流

#include <string> // 字符串

#include <algorithm> // 字符串处理

#include <unordered\_map> // 哈希表

#include <map> // 图

#include <stack> // 栈

#include <set> // 集

#include <vector> //容器

using namespace std;

unordered\_map<string, string> grammar;// 文法集合哈希表

string S; // 开始符

set<string> Vn; // 非终结符集

set<string> Vt; // 终结符集

set<string> formulas; // 产生式集。求解select时方便遍历

unordered\_map<string, set<char>> FIRST;// FIRST集合

unordered\_map<string, set<char>> FOLLOW;// FOLLOW集合

unordered\_map<string, set<char>> SELECT;// Select集合

map<pair<char, char>, string> TABLE;// 预测分析表

void readFile(); // 读取文件

void pretreatment(); // 预处理，简化、符号分类等

set<char> findFirstBV(string vn); // 某个非终结符的First集合（递归求解）

void findFirst(); // 使用哈希表存储

set<char> findFollowBV(string vn); // 某个非终结符的Follow集合（递归求解）

void findFollow(); // 使用哈希表存储

set<char> findSelectBF(string formula); // 某个产生式的Select集合（递归求解）

void findSelect(); // 使用哈希表存储

void isLL1(); // 判断是否为LL(1)分析

void makeTable(); // 构造预测分析表

void LL1Analyse(); // 分析字符串

// 根据左部获取产生式的右部（集合）

vector<string> getRights(string left);

// 判断是终结符还是非终结符

bool isVn(char v);

bool isVt(char v);

// 判断某个非终结符能否推出空

bool canToEmpty(char v);

//判断两个字符set的交集是否为空

bool isIntersect(set<char> a, set<char> b);

// 输出分析表

void printTable();

// 得到逆序字符串

string getStackRemain(stack<char> stack\_remain);

// 显示输出一个char集

void printSet(set<char> sets);

// 求FOLLOW集合中的元素个数（用于判断：直到follow集合不再增大）

int getFS();

int main() {

//1、读取文法并且简单处理

readFile();

//2、找First集

findFirst();

//3、找Follow集

findFollow();

//4、找Select集

findSelect();

//5、判断是否是LL1文法

isLL1();

//6、构建分析表

makeTable();

//7、分析字符串

LL1Analyse();

return 0;

}

// 读取文件

void readFile() {

cout << endl << "请输入文件名：";

char file[100];

cin >> file;

cout << endl << "文法读取" << endl;

// ifstream文件流打开文件

ifstream ifs;

ifs.open(file, ios::in);

if (!ifs.is\_open()){

cout << "打开文件失败";

exit(-1);

}

string line;

bool isGet = false;

while (getline(ifs, line)) {

if (!isGet){

// 得到开始符

S = line[0];

isGet = true;

}

formulas.insert(line); // 得到所有表达式

cout << line << endl;

// 如果哈希表中已经存在该键,加在后面

for (auto iter = grammar.begin(); iter != grammar.end(); ++iter) {

if (iter->first == string(1, line[0])){

iter->second = iter->second + "|" + line.substr(3);

break;

}

}

// 往存储文法的哈希表中插入键值对

grammar.insert(pair<string, string>(string(1, line[0]), line.substr(3)));

}

cout << "~表示空" << endl;

ifs.close();

pretreatment();

}

// 简单处理：符号分类、输出显示

void pretreatment() {

// 遍历文法哈希表

for (auto iter = grammar.begin(); iter != grammar.end(); ++iter) {

// 符号分类

Vn.insert(iter->first); // 非终结符集

// 终结符集合

string str = iter->second;

for (size\_t i = 0; i < str.length(); i++){

if (str[i] != '|' && (str[i] < 'A' || str[i] > 'Z')){

Vt.insert(string(1, str[i]));

}

}

}

cout << endl << "符号分类" << endl;

// 输出终结符和非终结符集合

cout << "开始符号：" << S << endl;

cout << "非终结符集Vn = " << "{";

for (auto iter = Vn.begin(); iter != Vn.end(); ) {

cout << \*iter;

if ((++iter) != Vn.end()){

cout << ",";

}

}

cout << "}" << endl;

cout << "终结符集Vt = " << "{";

for (auto iter = Vt.begin(); iter != Vt.end(); ) {

cout << \*iter;

if ((++iter) != Vt.end()){

cout << ",";

}

}

cout << "}" << endl;

}

// 求某一个非终结符的First集合

set<char> findFirstBV(string vn) {

set<char> results; // first集存储

vector<string> rights = getRights(vn); // 获取右部

if (!rights.empty()) {// 如果右部不为空

// 遍历右部集合（每一个右部分别求解first，加入到该非终结符的first集合中）

for (auto iter = rights.begin(); iter != rights.end(); ++iter) {

string right = \*iter;

// 遍历当前右部： //如果第一个字符是终结符，加入first集合并且跳出循环；

//如果是非终结符，则递归；

//如果非终结符可以推空还需要循环处理该右部的下一字符

for (auto ch = right.begin(); ch != right.end(); ++ch) {

if (isVn(\*ch)){ // 如果是非终结符，就要递归处理

//先查first集合。如果已经有了就不需要重复求解

if (FIRST.find(string(1, \*ch)) == FIRST.end()){ // fisrt集合中不存在

// 递归调用

set<char> chars = findFirstBV(string(1, \*ch));

results.insert(chars.begin(), chars.end());

FIRST.insert(pair<string, set<char>>(string(1, \*ch), chars));

}

else { //把该集合全部加到firsts

set<char> chars = FIRST[string(1, \*ch)];

results.insert(chars.begin(), chars.end());

}

// 如果这个字符可以推空，且后面还有字符，那么还需要处理下一个字符

if (canToEmpty(\*ch) && (iter + 1) != rights.end()){

continue;

}

else

break; // 否则直接退出遍历当前右部的循环

}

else { // 如果不是非终结符，直接把这个字符加入first集合，并且跳出

// 这一步会把前面的空也加进去

results.insert(\*ch);

break;

}

}

}

}

// 最后，如果该终结符不能推空，就删除空

if (!canToEmpty(vn[0])){

results.erase('~');//删除集合中的~

}

return results;

}

// 求解First集，使用哈希表存储

void findFirst() {

// 遍历非终结符集合，构建哈希表

for (auto iter = Vn.begin(); iter != Vn.end(); ++iter) {

string vn = \*iter; // 获取非终结符

set<char> firsts = findFirstBV(vn); // 存放一个Vn的first集

FIRST.insert(pair<string, set<char>>(vn, firsts));

}

// 显示输出

cout << endl << "FISRT集分析" << endl;

for (auto iter = FIRST.begin(); iter != FIRST.end(); ++iter) {

cout << "FIRST(" << iter->first << ")" << "= ";

set<char> sets = iter->second;

printSet(sets);

}

}

// 单个非终结符符求解其Follow集合

set<char> findFollowBV(string vn) {

set<char> results; // 存储求解结果

if (vn == S) {// 如果是开始符号

results.insert('#'); // 把结束符加进去，因为有语句#S#

}

// 遍历文法所有的右部集合

for (auto iter = formulas.begin(); iter != formulas.end(); ++iter){

string right = (\*iter).substr(3); // 获取当前右部

// 遍历当前右部，看是否含有当前符号

for (auto i\_ch = right.begin(); i\_ch != right.end();){

if (\*i\_ch == vn[0]) { // 如果vn出现在了当前右部

if ((i\_ch + 1) == right.end()){ // vn是当前右部最后一个字符

results.insert('#'); // 加入结束符

break;

}

else { // vn后面还有字符,遍历

while (i\_ch != right.end()){

++i\_ch;// 指针后移

if (\*i\_ch == vn[0]){

--i\_ch;

break;

}

if (isVn(\*i\_ch)){ // 如果该字符是非终结符，把first集中的非空元素加进去

set<char> tmp\_f = FIRST[string(1, \*i\_ch)];

tmp\_f.erase('~'); // 除去空

results.insert(tmp\_f.begin(), tmp\_f.end());

// 还要该字符可否推空，需要考虑是否继续循环

if (canToEmpty(\*i\_ch)){

if ((i\_ch + 1) == right.end()){ // 如果是最后一个字符，加入#

results.insert('#');

break;// 跳出循环

}

// 继续循环

}

else // 否则跳出循环

break;

}

else { // 如果该字符是终结符

results.insert(\*(i\_ch)); // 加入该字符

break; // 跳出循环

}

}

}

}

else {

++i\_ch;

}

}

}

return results;

}

// 完善Follow集合

void completeFollow(string vn) {

// 遍历文法所有的右部集合

for (auto iter = formulas.begin(); iter != formulas.end(); ++iter){

string right = (\*iter).substr(3); // 获取当前右部

// 遍历当前右部，看是否含有当前符号

for (auto i\_ch = right.begin(); i\_ch != right.end();){

char vn\_tmp = \*i\_ch;

if (vn\_tmp == vn[0]) { // 如果vn出现在了当前右部

if ((i\_ch + 1) == right.end()) {// vn是当前右部最后一个字符

char left = (\*iter)[0];

set<char> tmp\_fo = FOLLOW[string(1, left)]; // 获取左部的follow集合

set<char> follows = FOLLOW[string(1, vn\_tmp)]; // 获取自己的原来的follow集合

follows.insert(tmp\_fo.begin(), tmp\_fo.end());

FOLLOW[vn] = follows; // 修改

break;

}

else { // 不是最后一个字符，就要遍历之后的字符看是否可以推空

while (i\_ch != right.end()){

++i\_ch; // 指针后移

if (canToEmpty(\*i\_ch)){

if ((i\_ch + 1) != right.end()){ // 不是最后一个元素

continue;

}

else { // 最后一个也能推空,则把左部加进去

char left = (\*iter)[0];

set<char> tmp\_fo = FOLLOW[string(1, left)]; // 左部的follow集合

set<char> follows = FOLLOW[string(1, vn\_tmp)]; // 当前符号的follow集合

follows.insert(tmp\_fo.begin(), tmp\_fo.end());

FOLLOW[vn] = follows;

break;

}

}

else // 如果不能推空，就退出循环

break;

}

}

}

++i\_ch; // 遍历寻找vn是否出现

}

}

}

// 求解Follow集，使用哈希表存储

void findFollow() {

// 遍历所有非终结符，依次求解follow集合

for (auto iter = Vn.begin(); iter != Vn.end(); ++iter) {

string vn = \*iter; // 获取非终结符

set<char> follows = findFollowBV(vn); // 求解一个Vn的follow集

FOLLOW.insert(pair<string, set<char>>(vn, follows)); // 存储到哈希表

}

// 完善follow集合直到follow不再增大

int old\_count = getFS();

int new\_count = -1;

while (old\_count != new\_count) {// 终结符在变化，反复这个过程直到follow集合不再增大

old\_count = getFS();

// 再次遍历非终结符，如果出现在右部最末端的，把左部的follow集加进来

for (auto iter = Vn.begin(); iter != Vn.end(); ++iter) {

string vn = \*iter; // 获取非终结符

completeFollow(vn);

}

new\_count = getFS();

}

// 显示输出

cout << endl << setfill('=') << "FOLLOW集分析" << endl;

for (auto iter = FOLLOW.begin(); iter != FOLLOW.end(); ++iter) {

cout << "FOLLOW(" << iter->first << ")" << "= ";

set<char> sets = iter->second;

printSet(sets);

}

}

// 单个表达式求解Select集合

set<char> findSelectBF(string formula) {

set<char> results; // 存储结果

// 得到产生式的left和right

char left = formula[0]; // 左部

string right = formula.substr(3); // 右部

// 遍历右部产生式，首先分析右部第一个字符:right[0]

for (auto iter = right.begin(); iter != right.end(); ++iter){

// 如果是非终结符：把first(right[0])-'~'加入到results；如果还可以推空，则要继续continue

if (isVn(\*iter)){

set<char> chs = FIRST.find(string(1, \*iter))->second; // 得到该符号的first、

chs.erase('~'); // 去除空符

results.insert(chs.begin(), chs.end()); // 加入select

if (canToEmpty(\*iter)){ // 如果可以推空，继续处理下一个字符加入到select集合

if ((iter + 1) == right.end()){ // 当前是最后一个字符，则把follow(left)加入results,然后break

set<char> chs = FOLLOW.find(string(1, left))->second; // 得到左部的follow

results.insert(chs.begin(), chs.end()); // 加入select

}

else { // 继续处理下一字符

continue;

}

}

else

break; // 该字符不可以推空，退出循环

}

else {// 如果是终结符：（如果为空符，则把follow(left)加入results，否则直接把该符号加入到results）,然后break

if (\*iter == '~') {// 如果是空

set<char> chs = FOLLOW.find(string(1, left))->second; // 得到左部的follow

results.insert(chs.begin(), chs.end()); // 加入select

}

else

results.insert(\*iter); // 直接加入select

break; // 退出循环

}

}

return results;

}

// 求解Select集，使用哈希表存储

void findSelect() {

// 遍历表达式集合

for (auto iter = formulas.begin(); iter != formulas.end(); ++iter) {

string formula = \*iter; // 获取表达式

set<char> selects = findSelectBF(formula); // 存放一个Vn的first集

SELECT.insert(pair<string, set<char>>(formula, selects)); // 插入到哈希表，提高查询效率

}

// 显示输出

cout << endl << "SELECT集分析" << endl;

for (auto iter = SELECT.begin(); iter != SELECT.end(); ++iter) {

cout << "SELECT(" << iter->first << ")" << "= ";

set<char> sets = iter->second;

printSet(sets);

}

}

// 判断是否为LL(1)分析

void isLL1() {

for (auto i1 = SELECT.begin(); i1 != SELECT.end(); ++i1){

for (auto i2 = SELECT.begin(); i2 != SELECT.end(); ++i2){

char left1 = (i1->first)[0]; // 获取左部2

char left2 = (i2->first)[0]; // 获取左部2

if (left1 == left2){ // 左部相等

if (i1->first != i2->first){ //表达式不一样

if (isIntersect(i1->second, i2->second)) { // 如果select集合有交集

// 不是LL1文法

cout << "不符合LL(1)文法，请重试" << endl;

exit(0);

}

}

}

}

}

// 是LL（1）文法

cout << "符合LL(1)文法" << endl;

}

// 构造预测分析表

void makeTable() {

cout << "构造分析表" << endl;

// 遍历select集合，对于键：分为left和->right;

// 对于值，遍历后单个字符ch：把left和ch配对作为TABLE的键，而->right作为值

char left\_ch;

string right;

set<char> chars;

for (auto iter = SELECT.begin(); iter != SELECT.end(); ++iter){ // 遍历select集合

left\_ch = iter->first[0]; // 获取左部

right = iter->first.substr(1); // 获取->右部

chars = iter->second;

//一个个放入

for (char ch : chars) { // 遍历终结符

TABLE.insert(pair<pair<char, char>, string>(pair<char, char>(left\_ch, ch), right));

}

}

// 输出分析表

printTable();

}

// 输出预测分析表

void printTable() {

// 输出分析表

ofstream ofs;

ofs.open("pat.txt", ios::out);

//cout << setw(9) << left << setfill(' ') << "VN/VT";

ofs << setw(9) << left << setfill(' ') << "VN/VT";

set<string> vts = Vt;

vts.erase("~");

vts.insert("#");

for (string str : vts) {// 遍历终结符

//cout << setw(12) << left << setfill(' ') << str;

ofs << setw(15) << left << setfill(' ') << str;

}

//cout << endl << endl;

ofs << endl << endl;

for (string vn : Vn){

//cout << setw(7) << left << setfill(' ') << vn;

ofs << setw(7) << left << setfill(' ') << vn;

for (string vt : vts){ // 遍历终结符

if (TABLE.find(pair<char, char>(vn[0], vt[0])) == TABLE.end()){ //如果找不到

//cout << setw(12) << left << "ERR" << " ";

ofs << setw(12) << left << "ERR" << " ";

}

else {

//cout << setw(12) << left << TABLE.find(pair<char, char>(vn[0], vt[0]))->second << " ";

ofs << setw(12) << left << TABLE.find(pair<char, char>(vn[0], vt[0]))->second << " ";

}

}

//cout << endl;

ofs << endl;

}

ofs.close();

cout << "已完成储存" << endl;

}

// 分析字符串

void LL1Analyse() {

cout << "构造完成，请输入您要分析的字符串" << endl;

ofstream ofs;

ofs.open("result.txt", ios::out);

string str; // 输入串

cin >> str;

str.push\_back('#'); // 末尾加入结束符

/\*cout << setw(16) << left << setfill(' ') << "步骤";

cout << setw(16) << left << setfill(' ') << "分析栈";

cout << setw(16) << left << setfill(' ') << "剩余输入串";

cout << setw(16) << left << setfill(' ') << "分析情况" << endl;\*/

ofs << setw(12) << left << setfill(' ') << "步骤";

ofs << setw(12) << left << setfill(' ') << "分析式";

ofs << setw(12) << left << setfill(' ') << "剩余输入串";

ofs << setw(12) << left << setfill(' ') << "所用产生式"<< endl;

stack<char> stack\_a; // 分析栈

stack\_a.push('#'); // 末尾符进栈

stack\_a.push(S[0]); // 开始符号进栈

// 初始化显示数据

int step = 1; // 步骤数

stack<char> stack\_remain = stack\_a; // 剩余分析栈

string str\_remain = str; // 剩余分析串

string str\_situation = "待分析"; // 分析情况

// 初始数据显示

/\*cout << setw(16) << left << setfill(' ') << step;

cout << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

cout << setw(16) << left << setfill(' ') << str\_remain << endl;\*/

ofs << setw(16) << left << setfill(' ') << step;

ofs << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

ofs << setw(16) << left << setfill(' ') << str\_remain << endl;

// 遍历所输入的句子，一个个字符分析

for (auto iter = str.begin(); iter != str.end();) {

char a = \*iter; // 当前终结符送a

char X = stack\_a.top(); // 栈顶元素送X

if (isVt(X)) {// 如果X是Vt终结符，栈顶元素出栈，然后读取下一个字符

if (X == a){ // 和输入字符匹配

stack\_a.pop(); // 移除栈顶元素

// 从剩余分析串中移除本元素

for (auto i\_r = str\_remain.begin(); i\_r != str\_remain.end(); i\_r++){

if (\*i\_r == a) {

str\_remain.erase(i\_r);

break; // 只删除第一个,

}

}

// 重新组装提示字符串

string msg = string(1, a) + "匹配";

str\_situation = msg;

// 读取下一个字符

++iter;

}

else { // 无法匹配，分析出错

cout << "分析出错:" << X << "和" << a << "不匹配" << endl;

exit(-1); // 出错退出

}

}

else if (X == '#'){ // 文法分析结束

if (a == '#'){ // 当前符号也是最后一个符号,接受分析结果

cout << "分析结束" << endl;

exit(0);

}

else {

cout << "分析出错，文法结束输入串未结束" << endl;

exit(-1);

}

}

else { // X 就是非终结符了

// 查看TABLE（X，a）是否有结果

if (TABLE.find(pair<char, char>(X, a)) == TABLE.end()) {//如果找不到

if (!canToEmpty(X)){ // 也不能是空

cout << "分析出错,找不到表达式" << endl;

exit(-1); // 失败退出

}

else { // 可以是空，

stack\_a.pop(); // 移除栈顶元素

str\_situation.clear();

str\_situation.push\_back(X);

str\_situation = str\_situation + "->";

str\_situation = str\_situation + "~";

}

}

else {

stack\_a.pop();// 先将当前符号出栈

string str = TABLE.find(pair<char, char>(X, a))->second.substr(2); // 获取表达式并且逆序进栈(除去->)

// 重新组装字符串

str\_situation.clear();

str\_situation.push\_back(X);

str\_situation = str\_situation + "->";

str\_situation = str\_situation + str;

reverse(str.begin(), str.end());

for (auto iiter = str.begin(); iiter != str.end(); ++iiter){

if (\*iiter != '~'){

stack\_a.push(\*iiter);

}

}

//继续识别该字符

}

}

// 重置显示数据

++step; // 步骤数加1

stack\_remain = stack\_a; // 置剩余栈为当前栈

// 每次循环显示一次

/\*cout << setw(16) << left << setfill(' ') << step;

cout << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

cout << setw(16) << left << setfill(' ') << str\_remain;

cout << setw(16) << left << setfill(' ') << str\_situation << endl;\*/

ofs << setw(16) << left << setfill(' ') << step;

ofs << setw(16) << left << setfill(' ') << getStackRemain(stack\_remain);

ofs << setw(16) << left << setfill(' ') << str\_remain;

ofs << setw(16) << left << setfill(' ') << str\_situation << endl;

}

ofs.close();

}

// 根据左部返回某一产生式的右部集合

vector<string> getRights(string left){

vector<string> rights;

if (grammar.find(left) == grammar.end()){ // 语法中没有这一项，直接返回空

return rights;

}

else {

string str = grammar.find(left)->second;

str = str + '|'; // 末尾再加一个分隔符以便截取最后一段数据

size\_t pos = str.find('|');//find函数的返回值，若找到分隔符返回分隔符第一次出现的位置，

//否则返回npos

//此处用size\_t类型是为了返回位置

while (pos != string::npos){

string x = str.substr(0, pos);//substr函数，获得子字符串

rights.push\_back(x); // 存入right容器

str = str.substr(pos + 1); // 更新字符串

pos = str.find('|'); // 更新分隔符位置

}

return rights;

}

}

// 判断是终结符还是非终结符

bool isVn(char v) {

if (v >= 'A' && v <= 'Z') {

return true;

}

else {

return false;

}

}

bool isVt(char v) {

if (isVn(v) || v == '#' || v == '|') {

return false;

}

return true;

}

// 判断某个非终结符能否推出空

bool canToEmpty(char vn) {

vector<string> rights = getRights(string(1, vn)); // vn可能推出的右部集

for (auto i = rights.begin(); i != rights.end(); ++i){ // 遍历右部集合（如果前面的右部可以推空可提前跳出,不然就要看到最后）

string right = \*i; // 此为一个右部

// 遍历这个右部

for (auto ch = right.begin(); ch != right.end(); ++ch) {

if ((\*ch) == '~'){// 如果ch为空，说明可以推空（因为不可能存在右部是"εb"这样的情况，不需要看是否是最后一个字符）

return true;

}

else if (isVn(\*ch)) { // 如果是vn则需要递归

if (canToEmpty(\*ch)){// 如果可以推空

// 而且是最后一个字符，则返回true

//可能存在"AD"A->εD不能推空的情况，所以需要看是否最后一个字符

if ((ch + 1) == right.end()){

return true;

}

continue; // 当前字符可以推空，但不是最后一个字符，无法确定能否推空，还需要看右部的下一个字符

}

else // 如果不可以推空，说明当前右部不可以推空，需要看下一个右部

break;

}

else // 如果是非空vt说明目前右部不能推空，需要看下一个右部

break;

}

}

return false;

}

// 判断两个字符set的交集是否为空

bool isIntersect(set<char> as, set<char> bs) {

for (char a : as) {

for (char b : bs) {

if (a == b)

{

return true;

}

}

}

return false;

}

// 得到逆序字符串

string getStackRemain(stack<char> stack\_remain) {

string str;

while (!stack\_remain.empty()){

str.push\_back(stack\_remain.top());

stack\_remain.pop();

}

reverse(str.begin(), str.end());

return str;

}

// 显示输出一个char集

void printSet(set<char> sets) {

cout << "{ ";

for (auto i = sets.begin(); i != sets.end();) {

cout << \*i;

if (++i != sets.end()){

cout << " ,";

}

}

cout << " }" << endl;

}

// 求FOLLOW集合中的元素个数

int getFS() {

int count = 0;

for (auto iter = FOLLOW.begin(); iter != FOLLOW.end(); ++iter) {

count = count + iter->second.size();

}

return count;

}

//G=E',S=T'