**软件学院实践报告（编译原理）**

**课程编号： 3152100131 实践课程名称：编译原理 学年：2018-2019 学期：秋**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **学生姓名** | | 白烨淞 | **学号** | 2016522050 |
| **指导教师姓名** | | 修佳鹏 | 完成时间 | 2018.12.11 |
| **实验名称** | | **语法编译器** | | |
| **实验目的** | 为了让学生更加清楚了解语法分析的过程，通过动手实践，使学生对构造编译系统的基本理论、编译程序的基本结构有更为深入的理解和掌握；使学生掌握编译程序设计的基本方法和步骤；能够设计实现编译系统的重要环节。同时增强编写和调试程序的能力。 | | | |
| **实验要求** | （1）从控制台读入，每条产生式占用一行  （2）文法为LL(1)文法  从键盘上输入待分析的符号串，采用 LL(1)分析算法判断该符号串是否为该文法的句子。 | | | |
| **实验环境** | Windows10  VS2017 | | | |
| **设计方案** | （1）first集的算法思想如果产生式右部第一个字符为终结符，则将其计入左部first集 如果产生式右部第一个字符为非终结符执行以下步骤求该非终结符的first集将该非终结符的非$first集计入左部的first集若存在$，则将指向产生式的指针右移若不存在$，则停止遍历该产生式，进入下一个产生式若已经到达产生式的最右部的非终结符，则将$加入左部的first集处理数组中重复的first集中的终结符  （2）follow集的算法思想 对于文法G中每个非终结符A构造FOLLOW(A)的办法是,连续使用下面的规则,直到每个FOLLOW不在增大为止.对于文法的开始符号S,置#于FOLLOW(S)中;若A->aBb是一个产生式,则把FIRST(b)\{ε}加至FOLLOW(B)中;若A->aB是一个产生式,或A->aBb是一个产生式而b=>ε(即ε∈FIRST(b))则把FOLLOW(A)加至FOLLOW(B)中  （3）生成预测分析表的算法思想构造分析表M的算法是： 对文法G的每个产生式A->a执行第二步和第三步;对每个终结符a∈FIRST(a),把A->a加至M[A,a]中;若ε∈FIRST(a),则把任何b∈FOLLOW(A)把A->a加至M[A,b]中;把所有无定义的M[A,a]标上出错标志.  （4）对符号串的分析过程 预测分析程序的总控程序在任何时候都是按STACK栈顶符号X和当前的输入符号行事的,对于任何(X,a),总控程序 每次都执行下述三种可能的动作之一;若X=a=”#”,则宣布分析成功,停止分析过程.若X=a≠”#”,则把X从STACK栈顶逐出,让a指向下一个输入符号.若X是一个非终结符,则查看分析表M,若M[A,a]中存放着关于X的一个产生式,那么,首先把X逐出STACK栈顶,然后把产生式的右部符号串按反序一一推进STACK栈(若右部符号为ε,则意味着不推什么东西进栈).在把产生式的右部符号推进栈的同时应做这个产生式相应得语义动作,若M[A,a]中存放着”出错标志”,则调用出错诊察程序ERROR. | | | |
| **成源程序** | (详见附录) | | | |
| **实验结果** |  | | | |
| **实验分析** | 首先实现集合FIRST(X)构造算法和集合FOLLOW(A)构造算法，再根据FIRST和FOLLOW集合构造出预测分析表，并对指定的句子打印出分析栈的分析过程，判断是否为该文法的句子。 | | | |
| **心得体会** | 通过此次实验，让我了解到如何设计、编制并调试语法分析程序，加深对语法分析原理的理解；熟悉了构造语法分析程序的手工方式的相关原理，使用某种高级语言（例如Python语言）直接编写此法分析程序。另外，也让我重新熟悉了C++语言的相关内容，加深了对C++语言的用途的理解。 | | | |

**附录**

#include "pch.h"

#include <iostream>

#include<string>

#include<map>

#include<vector>

#include<stack>

#include<set>

#include<cstring>

using namespace std;

map<char, int>getnum;

char get\_char[100]; //获得对应字符

vector<string>proce;

int table[100][100]; //预测分析表

int num = 0; int numvt = 0; //numvt是终结符集合，0是‘#’，numvt表空字

string first[100];

string follow[200];

string word;

void Input()

{

memset(table, -1, sizeof(table));

getnum['#'] = 0;

get\_char[0] = '#';

cout << "请输入所有的终结符：" << endl;

char x;

do

{

cin >> x;

getnum[x] = ++num;

get\_char[num] = x;

} while (cin.peek() != '\n');

numvt = ++num;

getnum['@'] = numvt; //kong zi

get\_char[num] = ('@');

cout << "请输入所有非终结符：" << endl;

do

{

cin >> x;

getnum[x] = ++num;

get\_char[num] = x;

} while (cin.peek() != '\n');

cout << "输入产生式集合（空字用‘@’表示）,以‘end’结束:" << endl;

string pro;

while (cin >> pro && pro != "end")

{

string ss;

ss += pro[0];

for (int i = 3; i < pro.size(); i++)

{

if (pro[i] == '|')

{

proce.push\_back(ss);

ss.clear(); ss += pro[0];

}

else

{

ss += pro[i];

}

}

proce.push\_back(ss);

}

}

void Intersection(string &a, string b) //a=a or b 取a,b交集赋值给a

{

set<char>se;

for (int i = 0; i < a.size(); i++)

se.insert(a[i]);

for (int i = 0; i < b.size(); i++)

se.insert(b[i]);

string ans;

set<char>::iterator it;

for (it = se.begin(); it != se.end(); it++)

ans += \*it;

a = ans;

}

string get\_f(int vn, int & has\_0) //dfs:vn能推出的不含空字的vt集合，并且判断vn能否推出空字

{

if (vn == numvt)has\_0 = 1;

if (vn < numvt)return first[vn];

string ans;

for (int i = 0; i < proce.size(); i++)

{

if (getnum[proce[i][0]] == vn)

ans += get\_f(getnum[proce[i][1]], has\_0);

}

return ans;

}

void getfirst()

{

for (int i = 1; i <= numvt; i++) //终结符，first集是其本身。

{

first[i] += ('0' + i);

}

for (int j = 0; j < proce.size(); j++) //扫描所有产生式

{

int k = 0; int has\_0 = 0; //k扫瞄该产生式

do {

has\_0 = 0;

k++;

if (k == proce[j].size()) //推到最后一个了，则附加空字

{

first[getnum[proce[j][0]]] += ('0' + numvt);

break;

} //合并之

Intersection(first[getnum[proce[j][0]]], get\_f(getnum[proce[j][k]], has\_0));

}

while (has\_0); //到无法推出空字为止

}

}

void print\_first()

{

cout << "first集:" << endl;

for (int i = 1; i <= num; i++)

{

cout << "first [" << get\_char[i] << "]: ";

for (int j = 0; j < first[i].size(); j++)

cout << get\_char[first[i][j] - '0'] << " ";

cout << endl;

}

cout << endl;

}

void getfollow()

{

for (int i = 0; i < 2; i++)

{

Intersection(follow[getnum[proce[0][0]]], "0"); //先添加‘#’；

for (int j = 0; j < proce.size(); j++) //扫所有产生式

{

for (int jj = 1; jj < proce[j].size(); jj++) //每个非终结符的follow集

{

if (getnum[proce[j][jj]] <= numvt)continue; //vt无follow集

int k = jj; int has\_0;

do

{

has\_0 = 0;

k++;

if (k == proce[j].size()) //都能推出空字，follow集=产生式左边的vn，

{

Intersection(follow[getnum[proce[j][jj]]], follow[getnum[proce[j][0]]]);

break;

}

Intersection(follow[getnum[proce[j][jj]]], get\_f(getnum[proce[j][k]], has\_0));

} while (has\_0);

}

}

}

}

void gettable() //得预测分析表

{

for (int i = 0; i < proce.size(); i++) //扫所有产生式

{

if (proce[i][1] == '@') //直接推出空字的，特判下（follow集=产生式左边的vn中元素填）

{

string flw = follow[getnum[proce[i][0]]];

for (int k = 0; k < flw.size(); k++)

{

table[getnum[proce[i][0]]][flw[k] - '0'] = i;

}

}

string temps = first[getnum[proce[i][1]]];

for (int j = 0; j < temps.size(); j++) //考察first集

{

if (temps[j] != ('0' + numvt))

{

table[getnum[proce[i][0]]][temps[j] - '0'] = i;

}

else //有空字的，考察follw集

{

string flw = follow[getnum[proce[i][1]]];

for (int k = 0; k < flw.size(); k++)

{

table[getnum[proce[i][0]]][flw[k] - '0'] = i;

}

}

}

}

}

string get\_proce(int i) //由对应下标获得对应产生式。

{

if (i < 0)return " "; //无该产生式

string ans;

ans += proce[i][0];

ans += "->";

//ans+=(proce[i][0]+"->"); 注意这样不行！思之即可。

for (int j = 1; j < proce[i].size(); j++)

ans += proce[i][j];

return ans;

}

void print\_table()

{

cout << "预测分析表：" << endl;

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

cout << '\t' << get\_char[i];

cout << endl;

for (int i = numvt + 1; i <= num; i++)

{

cout << get\_char[i];

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

{

cout << '\t' << get\_proce(table[i][j]);

}

cout << endl;

}

cout << endl;

}

void print\_follow()

{

cout << "follow集：" << endl;

for (int i = numvt + 1; i <= num; i++)

{

cout << "follow [" << get\_char[i] << "]: ";

for (int j = 0; j < follow[i].size(); j++)

cout << get\_char[follow[i][j] - '0'] << " ";

cout << endl;

}

cout << endl;

}

bool analyze() //总控，分析字word的合法性，若合法，输出所有产生式。

{

stack<char>sta;

sta.push('#'); sta.push(proce[0][0]);

int i = 0;

while (!sta.empty())

{

int cur = sta.top();

sta.pop();

if (cur == word[i]) //是终结符，推进

{

i++;

}

else if (cur == '#') //成功，结束

{

return 1;

}

else if (table[getnum[cur]][getnum[word[i]]] != -1) //查表

{

int k = table[getnum[cur]][getnum[word[i]]];

cout << proce[k][0] << "->";

for (int j = 1; j < proce[k].size(); j++)

cout << proce[k][j];

cout << endl;

for (int j = proce[k].size() - 1; j > 0; j--) //逆序入栈

{

if (proce[k][j] != '@')

sta.push(proce[k][j]);

}

}

else //失败！

{

return 0;

}

}

return 1;

}

int main()

{

Input();

getfirst();

getfollow();

gettable();

print\_first();

print\_follow();

print\_table();

cout << "请输入字：" << endl;

cin >> word;

if (analyze())

cout << "succeed!该字有效，所用产生式如上。" << endl;

else cout << "error!" << endl;

return 0;

}