**中山大学数据科学与计算机学院本科生实验报告**

课程名称：编译器构造实验 任课教师：陈炬桦 教学助理（TA）：

|  |  |  |  |
| --- | --- | --- | --- |
| 学年度 | 2018-2019 | 学期 | 第二学期 |
| 年级 | 2016级 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337327 | 姓名 | 郑映雪 |
| 电话 | 18956073450 | Email | 646352420@qq.com |
| 开始日期 | 2019.3.15 | 完成日期 | 2019.3.15 |

**实验题目**

## 1. Description

输入开始符号，非终结符，终结符，产生式

压缩自产生式文法和不可达文法后，按非终结符顺序输出产生式；

## 2. Input

输入开始符号；

非终结符个数，非终结符，空格符分隔；

终结符个数，终结符，空格符分隔；

产生式的个数，各产生式的左边和右边符号，空格符分隔；

## 3. Output

G[开始符号]：

按非终结符顺序输出各产生式；

**算法描述**

## 1. 流程总览（绿色部分为本实验添加部分）

## 2.各模块功能

本次实验是在上一次实验的基础上进行的。因此以下四个模块与上一次实验相同：

**a.初始模块**

在进行文法输出之前，需要进行准备工作。首先，对终结符、非终结符和产生式建立表,并进行输入操作。考虑到输出是比较复杂的串集，所以输出集定义为string类型的数组result比较方便。

**b.判重模块**

对于输入的每一个产生式，都有左部和右部，我们想把这些产生式归结起来，就需要根据左部进行分类。一个很方便的办法就是，对于每个输入的产生式的左部，都在result进行判重操作。如果result里已经有该左部的产生式信息（即result[index][0]=PL[0]）,则直接在该项的字符串上进行“加”操作，加上分隔符“|”和右部。如果result里没有该左部，就在result里建立新项，并对于该新项进行左部、等于符号和右部的“加”操作。

**c.输出模块**

题目要求按照非终结符输入的顺序来输出。所以我们在归类完毕之后，还需要按照非终结符输入顺序对result里的符合该非终结符的项进行输出，以达到按非终结符输入顺序重新输出的目的。

**d. 额外模块**

本题没有要求进行非法输入的判断。但是在生活中，非法输入的判断是需要进行的。对于每一个输入的左部和右部，我们要判断是否在非终结符∪终结符列表中。由于题目本身没有要求，所以此处为额外模块的部分，但仍为必要的部分。

对于本次实验中添加的部分，即自产生式文法和不可达文法的判断：

**e.自产生式文法判断**

根据自产生式文法（E::=E类型）的定义，我们只要在输入左部和右部时进行相等判断即可。如果左部和右部相等，直接不加入判重模块中，也不加入左部和右部的数组中。同时输出已经删除的左部和右部作为提示。

**f.不可达式文法判断**

根据不可达式文法（左部在其他产生式右部均未出现）的定义，可以在判重结束之后，对于每个result串，进行下面的判断：

①遍历所有除自身以外的result串。

②如果当前result元素串里有当前result的第一个字母（即左部），则代表并非不可达。因为在自身以外的result串里如果出现了自身左部的非终结符，那么一定是在该串的右部，所以这个非终结符是可达的。

③如果遍历完毕也未找到一个符合条件的result串，则说明这个非终结符是不可达的，要把它从result里删去。

至此，两个文法的判断都已完成，最终进行输出。

**测试数据(2组)（采用上个实验的测试数据）**

|  |  |  |
| --- | --- | --- |
| 测试数据 | 测试结果 | 结果判定 |
|  | （因为是即时输出所以截图会截到相邻的输入） | 正确 |
|  |  | 正确 |

**程序清单**

1. #include <iostream>
2. #include <string>
3. **using** **namespace** std;
4. **struct** { **int** Nv; string VN[10]; } Vns = { 0};
5. **struct** { **int** Nt; string VT[10]; } Vts = { 0};
6. **struct** { **int** Np; string PL[20], PR[20]; } ps = { 0 };
7. string S;
8. **int** judge(string left, string right, string result[], **int** len) {
9. **for** (**int** j = 0; j < len; ++j) {
10. **if** (left[0] == result[j][0]) **return** j;
11. }
12. **return** -1;
13. }
14. **int** main(){
15. cin >> S>>Vns.Nv;
16. **for** (**int** i = 0; i < Vns.Nv; ++i)
17. cin >> Vns.VN[i];
18. cin >> Vts.Nt;
19. **for** (**int** i = 0; i < Vts.Nt; ++i)
20. cin >> Vts.VT[i];
22. cin >> ps.Np;
23. string result[20];
24. **int** len=0;
25. **for** (**int** i = 0; i < ps.Np; ++i) {
26. cin >> ps.PL[i] >> ps.PR[i];
27. **if** (ps.PL[i] == ps.PR[i]) {
28. cout << "delete self production:" << ps.PL[i] << "::=" << ps.PR[i]<<endl;
29. **continue**;
30. }
31. **int** index = judge(ps.PL[i], ps.PR[i], result, len);
32. **if** (index!=-1) {
33. result[index] += " | ";
34. result[index] += ps.PR[i];
35. }
36. **else** {
37. result[len] = ps.PL[i] + "::=" + ps.PR[i];
38. len++;
39. }
40. }
41. cout << "unreached Vn:";
42. **for** (**int** i = 0; i < len; ++i) {
43. **int** flag1 = 0;
44. **for** (**int** j = 0; j < len; ++j) {
45. **if** (i == j) **continue**;
46. **if** (result[i][0] == S[0]) flag1 = 1;
47. **for** (**int** k = 0; k < result[j].size(); ++k) {
48. **if** (result[i][0] == result[j][k]) flag1 = 1;
49. }
51. }
52. **if** (!flag1) {
53. cout << result[i][0]<<endl;
54. **for** (**int** k2 = 0; k2 < ps.Np; ++k2) {
55. **if** (result[i][0] == ps.PL[k2][0])
56. cout << "delete production:" << ps.PL[k2] << "::=" << ps.PR[k2]<<endl;
57. }
58. cout << "delete VN:" << result[i][0] << endl;
60. result[i] = "";
61. }
62. }
63. cout << "G[" << S << "]:" << endl;
65. **for** (**int** j = 0; j < Vns.Nv; ++j) {
66. **for** (**int** i = 0; i < len; ++i) {
67. **if** (Vns.VN[j][0] == result[i][0]){
68. cout << result[i] << endl;
69. }
71. }
72. }
73. system("pause");
74. }