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 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337052 | 姓名 | 杜尔鑫 |
| 电话 | 15626281204 | Email | duerx@mail2.sysu.edu.cn |
| 开始日期 | 2019-03-15 | 完成日期 | 2019-03-15 |

1. 实验题目

**1.1 Description**

输入开始符号，非终结符，终结符，产生式，LL(1)分析表  
输出LL(1)分析表

此题需要提交实验报告；“实验报告用“学号+姓名+72”

**1.2 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，各产生式的序号，产生式的左边和右边符号，空格符分隔；

LL(1)分析表中的产生式个数，序号，行符号，列符号，产生式编号，空格符分隔；

输入一个算术式符号串，用#结束

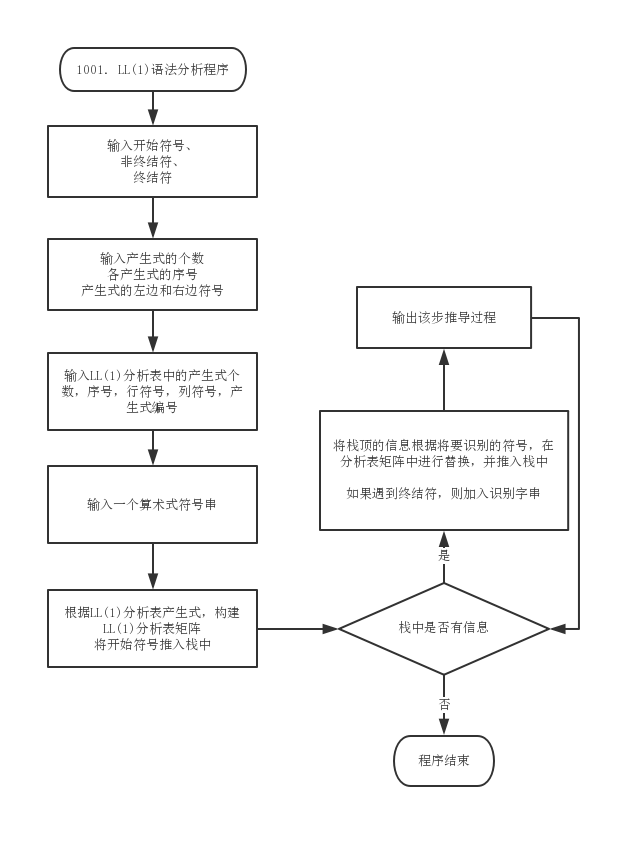
**1.3 Output**

输出推导过程，每一步一行，中间“ & ”前是已经识别的子串，后是栈中信息。

2. 算法描述(介绍程序模块功能；流程图)

书本（62、63页）LL（1）分析程序的总控算法

1. 输入输入开始符号、非终结符、终结符；
2. 输入产生式的个数、各产生式的序号、产生式的左边和右边符号；
3. 输入LL(1)分析表中的产生式个数，序号，行符号，列符号，产生式编号；
4. 输入一个算术式符号串；
5. 根据LL(1)分析表产生式，构建LL(1)分析表矩阵；将开始符号推入栈中；
6. 判断栈中是否有信息，如果没有，程序结束;如果有，进入7)；
7. 将栈顶的信息根据将要识别的符号，在分析表矩阵中进行替换，并推入栈中；如果遇到终结符，则加入识别字串；
8. 输出该步推导过程，返回6)。



3. 测试数据(2组)

**3.1 Standard Example**

**3.1.1 Input**

|  |
| --- |
| E  6 E A T B F D  9 + - \* / ( ) x y z  13  1 E TA  2 A +TA  3 A -TA  4 A k  5 T FB  6 B \*FB  7 B /FB  8 B k  9 F (E)  10 F D  11 D x  12 D y  13 D z  25  1 E ( 1  2 E x 1  3 E y 1  4 E z 1  5 A + 2  6 A - 3  7 A ) 4  8 A # 4  9 T ( 5  10 T x 5  11 T y 5  12 T z 5  13 B + 8  14 B - 8  15 B \* 6  16 B / 7  17 B ) 8  18 B # 8  19 F ( 9  20 F x 10  21 F y 10  22 F z 10  23 D x 11  24 D y 12  25 D z 13  (x+(y-x\*z)\*(y+x\*z))+x/z# |

**3.1.2 Output**

|  |
| --- |
| # & E#  # & TA#  # & FBA#  # & (E)BA#  #( & E)BA#  #( & TA)BA#  #( & FBA)BA#  #( & DBA)BA#  #( & xBA)BA#  #(x & BA)BA#  #(x & A)BA#  #(x & +TA)BA#  #(x+ & TA)BA#  #(x+ & FBA)BA#  #(x+ & (E)BA)BA#  #(x+( & E)BA)BA#  #(x+( & TA)BA)BA#  #(x+( & FBA)BA)BA#  #(x+( & DBA)BA)BA#  #(x+( & yBA)BA)BA#  #(x+(y & BA)BA)BA#  #(x+(y & A)BA)BA#  #(x+(y & -TA)BA)BA#  #(x+(y- & TA)BA)BA#  #(x+(y- & FBA)BA)BA#  #(x+(y- & DBA)BA)BA#  #(x+(y- & xBA)BA)BA#  #(x+(y-x & BA)BA)BA#  #(x+(y-x & \*FBA)BA)BA#  #(x+(y-x\* & FBA)BA)BA#  #(x+(y-x\* & DBA)BA)BA#  #(x+(y-x\* & zBA)BA)BA#  #(x+(y-x\*z & BA)BA)BA#  #(x+(y-x\*z & A)BA)BA#  #(x+(y-x\*z & )BA)BA#  #(x+(y-x\*z) & BA)BA#  #(x+(y-x\*z) & \*FBA)BA#  #(x+(y-x\*z)\* & FBA)BA#  #(x+(y-x\*z)\* & (E)BA)BA#  #(x+(y-x\*z)\*( & E)BA)BA#  #(x+(y-x\*z)\*( & TA)BA)BA#  #(x+(y-x\*z)\*( & FBA)BA)BA#  #(x+(y-x\*z)\*( & DBA)BA)BA#  #(x+(y-x\*z)\*( & yBA)BA)BA#  #(x+(y-x\*z)\*(y & BA)BA)BA#  #(x+(y-x\*z)\*(y & A)BA)BA#  #(x+(y-x\*z)\*(y & +TA)BA)BA#  #(x+(y-x\*z)\*(y+ & TA)BA)BA#  #(x+(y-x\*z)\*(y+ & FBA)BA)BA#  #(x+(y-x\*z)\*(y+ & DBA)BA)BA#  #(x+(y-x\*z)\*(y+ & xBA)BA)BA#  #(x+(y-x\*z)\*(y+x & BA)BA)BA#  #(x+(y-x\*z)\*(y+x & \*FBA)BA)BA#  #(x+(y-x\*z)\*(y+x\* & FBA)BA)BA#  #(x+(y-x\*z)\*(y+x\* & DBA)BA)BA#  #(x+(y-x\*z)\*(y+x\* & zBA)BA)BA#  #(x+(y-x\*z)\*(y+x\*z & BA)BA)BA#  #(x+(y-x\*z)\*(y+x\*z & A)BA)BA#  #(x+(y-x\*z)\*(y+x\*z & )BA)BA#  #(x+(y-x\*z)\*(y+x\*z) & BA)BA#  #(x+(y-x\*z)\*(y+x\*z) & A)BA#  #(x+(y-x\*z)\*(y+x\*z) & )BA#  #(x+(y-x\*z)\*(y+x\*z)) & BA#  #(x+(y-x\*z)\*(y+x\*z)) & A#  #(x+(y-x\*z)\*(y+x\*z)) & +TA#  #(x+(y-x\*z)\*(y+x\*z))+ & TA#  #(x+(y-x\*z)\*(y+x\*z))+ & FBA#  #(x+(y-x\*z)\*(y+x\*z))+ & DBA#  #(x+(y-x\*z)\*(y+x\*z))+ & xBA#  #(x+(y-x\*z)\*(y+x\*z))+x & BA#  #(x+(y-x\*z)\*(y+x\*z))+x & /FBA#  #(x+(y-x\*z)\*(y+x\*z))+x/ & FBA#  #(x+(y-x\*z)\*(y+x\*z))+x/ & DBA#  #(x+(y-x\*z)\*(y+x\*z))+x/ & zBA#  #(x+(y-x\*z)\*(y+x\*z))+x/z & BA#  #(x+(y-x\*z)\*(y+x\*z))+x/z & A#  #(x+(y-x\*z)\*(y+x\*z))+x/z & # |

**3.2 Addition Test**

**3.2.1 Input**

|  |
| --- |
| E  6 E A T B F D  9 + - \* / ( ) x y z  13  1 E TA  2 A +TA  3 A -TA  4 A k  5 T FB  6 B \*FB  7 B /FB  8 B k  9 F (E)  10 F D  11 D x  12 D y  13 D z  25  1 E ( 1  2 E x 1  3 E y 1  4 E z 1  5 A + 2  6 A - 3  7 A ) 4  8 A # 4  9 T ( 5  10 T x 5  11 T y 5  12 T z 5  13 B + 8  14 B - 8  15 B \* 6  16 B / 7  17 B ) 8  18 B # 8  19 F ( 9  20 F x 10  21 F y 10  22 F z 10  23 D x 11  24 D y 12  25 D z 13  (x+y+(x\*z)\*(y/x-z))+x/z# |

**3.2.2 Output**

|  |
| --- |
| # & E#  # & TA#  # & FBA#  # & (E)BA#  #( & E)BA#  #( & TA)BA#  #( & FBA)BA#  #( & DBA)BA#  #( & xBA)BA#  #(x & BA)BA#  #(x & A)BA#  #(x & +TA)BA#  #(x+ & TA)BA#  #(x+ & FBA)BA#  #(x+ & DBA)BA#  #(x+ & yBA)BA#  #(x+y & BA)BA#  #(x+y & A)BA#  #(x+y & +TA)BA#  #(x+y+ & TA)BA#  #(x+y+ & FBA)BA#  #(x+y+ & (E)BA)BA#  #(x+y+( & E)BA)BA#  #(x+y+( & TA)BA)BA#  #(x+y+( & FBA)BA)BA#  #(x+y+( & DBA)BA)BA#  #(x+y+( & xBA)BA)BA#  #(x+y+(x & BA)BA)BA#  #(x+y+(x & \*FBA)BA)BA#  #(x+y+(x\* & FBA)BA)BA#  #(x+y+(x\* & DBA)BA)BA#  #(x+y+(x\* & zBA)BA)BA#  #(x+y+(x\*z & BA)BA)BA#  #(x+y+(x\*z & A)BA)BA#  #(x+y+(x\*z & )BA)BA#  #(x+y+(x\*z) & BA)BA#  #(x+y+(x\*z) & \*FBA)BA#  #(x+y+(x\*z)\* & FBA)BA#  #(x+y+(x\*z)\* & (E)BA)BA#  #(x+y+(x\*z)\*( & E)BA)BA#  #(x+y+(x\*z)\*( & TA)BA)BA#  #(x+y+(x\*z)\*( & FBA)BA)BA#  #(x+y+(x\*z)\*( & DBA)BA)BA#  #(x+y+(x\*z)\*( & yBA)BA)BA#  #(x+y+(x\*z)\*(y & BA)BA)BA#  #(x+y+(x\*z)\*(y & /FBA)BA)BA#  #(x+y+(x\*z)\*(y/ & FBA)BA)BA#  #(x+y+(x\*z)\*(y/ & DBA)BA)BA#  #(x+y+(x\*z)\*(y/ & xBA)BA)BA#  #(x+y+(x\*z)\*(y/x & BA)BA)BA#  #(x+y+(x\*z)\*(y/x & A)BA)BA#  #(x+y+(x\*z)\*(y/x & -TA)BA)BA#  #(x+y+(x\*z)\*(y/x- & TA)BA)BA#  #(x+y+(x\*z)\*(y/x- & FBA)BA)BA#  #(x+y+(x\*z)\*(y/x- & DBA)BA)BA#  #(x+y+(x\*z)\*(y/x- & zBA)BA)BA#  #(x+y+(x\*z)\*(y/x-z & BA)BA)BA#  #(x+y+(x\*z)\*(y/x-z & A)BA)BA#  #(x+y+(x\*z)\*(y/x-z & )BA)BA#  #(x+y+(x\*z)\*(y/x-z) & BA)BA#  #(x+y+(x\*z)\*(y/x-z) & A)BA#  #(x+y+(x\*z)\*(y/x-z) & )BA#  #(x+y+(x\*z)\*(y/x-z)) & BA#  #(x+y+(x\*z)\*(y/x-z)) & A#  #(x+y+(x\*z)\*(y/x-z)) & +TA#  #(x+y+(x\*z)\*(y/x-z))+ & TA#  #(x+y+(x\*z)\*(y/x-z))+ & FBA#  #(x+y+(x\*z)\*(y/x-z))+ & DBA#  #(x+y+(x\*z)\*(y/x-z))+ & xBA#  #(x+y+(x\*z)\*(y/x-z))+x & BA#  #(x+y+(x\*z)\*(y/x-z))+x & /FBA#  #(x+y+(x\*z)\*(y/x-z))+x/ & FBA#  #(x+y+(x\*z)\*(y/x-z))+x/ & DBA#  #(x+y+(x\*z)\*(y/x-z))+x/ & zBA#  #(x+y+(x\*z)\*(y/x-z))+x/z & BA#  #(x+y+(x\*z)\*(y/x-z))+x/z & A#  #(x+y+(x\*z)\*(y/x-z))+x/z & # |

4. 程序清单

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <string>  #include <unistd.h>  using namespace std;  struct Vns\_Struct { int Nv; string VN[10];} ;  struct Vts\_Struct { int Nt; string VT[10];} ;  struct ps\_Struct { int Np; string PL[20], PR[20];} ;  struct llps\_Struct { int Np; string PL[30], PR[30]; int psNum[30];} ;  string S;  string expression;  int main(int argc, char const \*argv[])  {      Vns\_Struct Vns;      Vts\_Struct Vts;      ps\_Struct ps;      llps\_Struct llps;      string VnsString = "", VtsString = "";      cin >> S;      cin >> Vns.Nv;      for (int i = 0; i < Vns.Nv; ++i)      {          cin >> Vns.VN[i];          VnsString = VnsString + Vns.VN[i];      }      cin >> Vts.Nt;      for (int i = 0; i < Vts.Nt; ++i)      {          cin >> Vts.VT[i];          VtsString = VtsString + Vts.VT[i];      }      cin >> ps.Np;      for (int i = 0; i < ps.Np; ++i)      {          int num;          cin >> num >> ps.PL[i] >> ps.PR[i];      }      cin >> llps.Np;      for (int i = 0; i < llps.Np; ++i)      {          int num;          cin >> num >> llps.PL[i] >> llps.PR[i] >> llps.psNum[i];      }      cin >> expression;      // middel process      Vts.VT[Vts.Nt++] = "#";      string processString[Vns.Nv][20];      for (int i = 0; i < Vns.Nv; ++i)          for (int z = 0; z < Vts.Nt; ++z)              for (int j = 0; j < llps.Np; ++j)                  if (llps.PL[j] == Vns.VN[i] && llps.PR[j] == Vts.VT[z])                      processString[i][z] = ps.PR[llps.psNum[j] - 1];      string done, nodone = S;      int pointer = 0;      while (nodone != "")      {          cout << "#" << done << " & " << nodone << "#" << endl;          int found = VtsString.find(nodone[0]);          if (found != std::string::npos)          {              done += nodone[0];              nodone = nodone.substr(1, nodone.length());              pointer++;          }          else          {              int VnsIndex = VnsString.find(nodone[0]);              int VtsIndex = VtsString.find(expression[pointer]);              string toString = processString[VnsIndex][VtsIndex];              nodone.replace(nodone.begin(), nodone.begin() + 1, toString);          }          while (nodone[0] == 'k')              nodone = nodone.substr(1, nodone.length());      }      cout << "#" << done << " & " << nodone << "#" << endl;      return 0;  } |