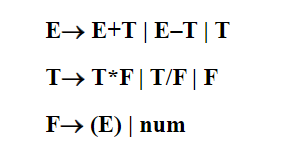
**C语言词法分析程序 实验报告**

#### 一、实验题目及要求

**1、语法分析程序的设计与实现**

编写语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生**。**



**2、实现方式**

方法1：编写递归调用程序实现自顶向下的分析。

.

方法2：编写LL(1)语法分析程序，要求如下。（必做）

(1) 编程实现算法4.2，为给定文法自动构造预测分析表。

(2) 编程实现算法4.1，构造LL(1)预测分析程序。

方法3：编写语法分析程序实现自底向上的分析，要求如下。（必做）

(1) 构造识别该文法所有活前缀的DFA。

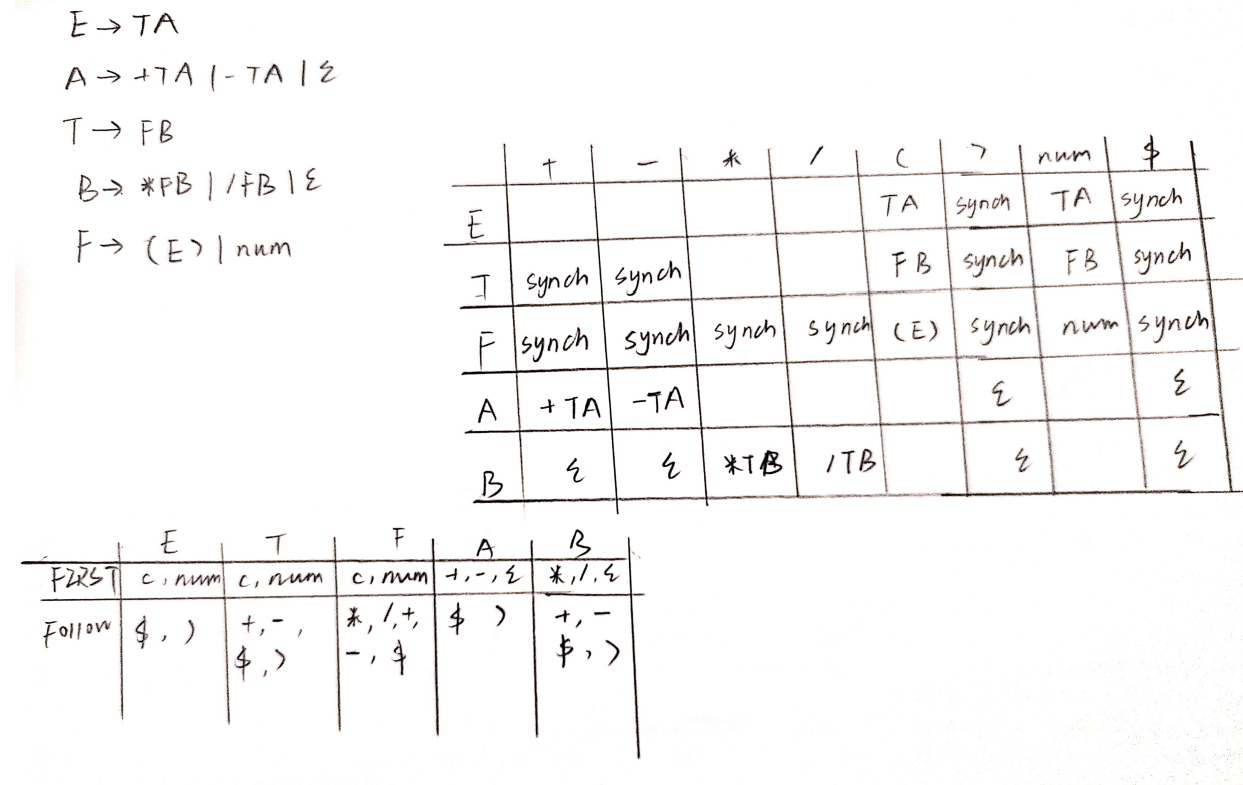
(2) 构造该文法的LR分析表。

(3) 编程实现算法4.3，构造LR分析程序。

方法4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

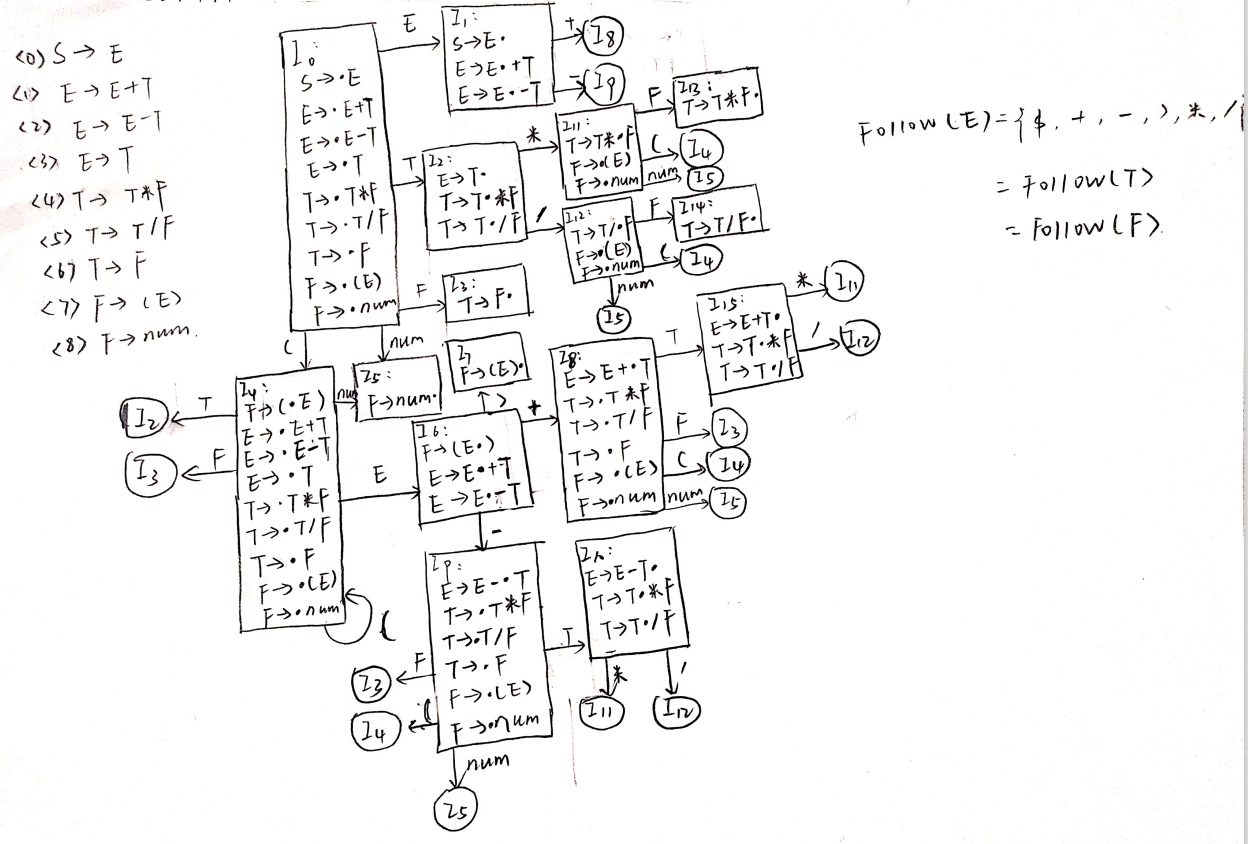
#### 二、程序设计说明

**1.LL(1)语法分析表（带错误处理）**

****

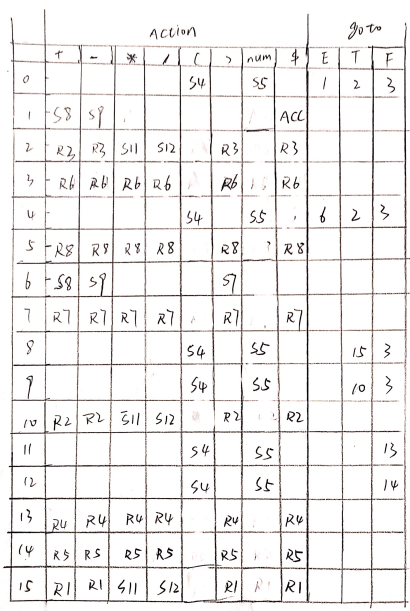
由于原本文法存在左递归，需要先消除左递归得到左侧的文法。如上图所示构建带错误处理的LL分析表。

**2.LR（SLR）项目集规范族及识别文法所有活前缀的DFA**

****

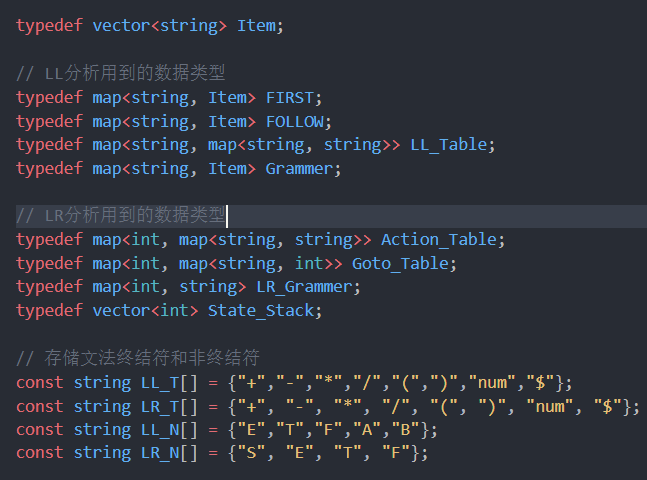
如上图所示，将文法进行拓广得到左边一列拓广文法共9个产生式。将其从I0开始构建项目集规范族和识别文法所有活前缀的DFA，共16个状态。在I1、I2、I10和I15状态存在移进和规约同时存在的情况，但不存在冲突，为SLR型文法。

**3.LR（SLR）分析表**

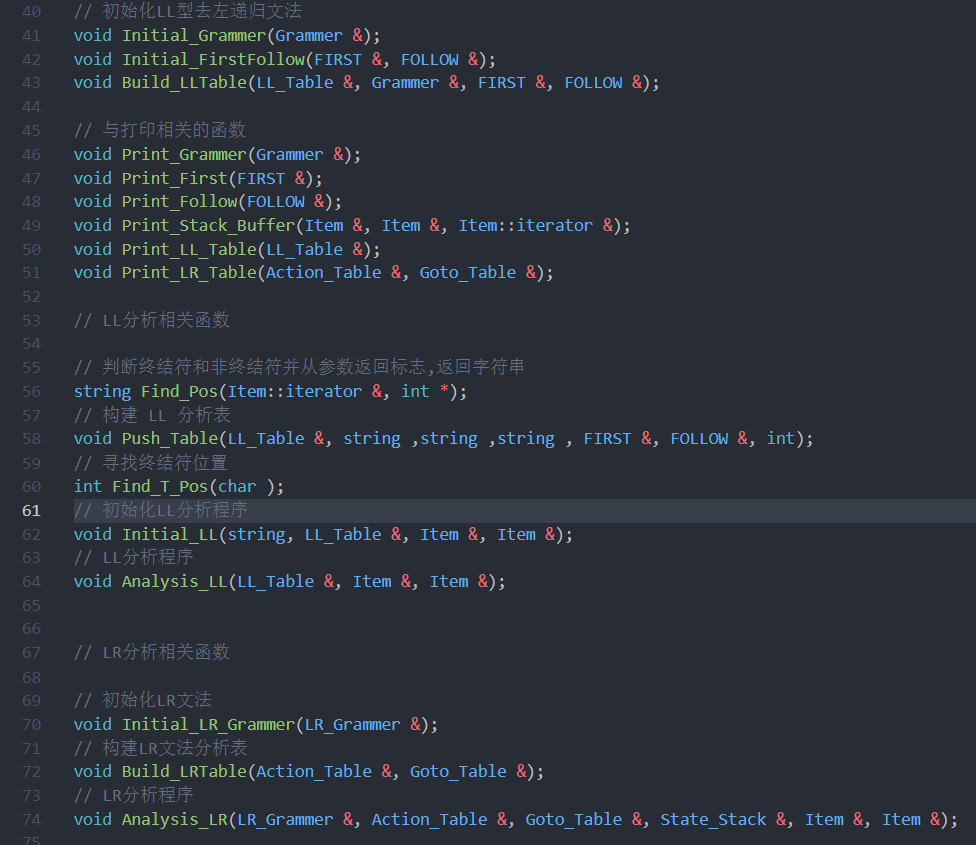
****

如图所示，根据识别文法活前缀的DFA构造LR文法分析表。

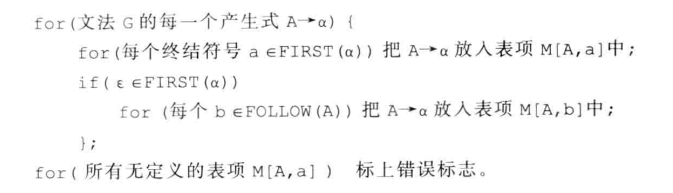
**4.数据结构与类型定义设计**

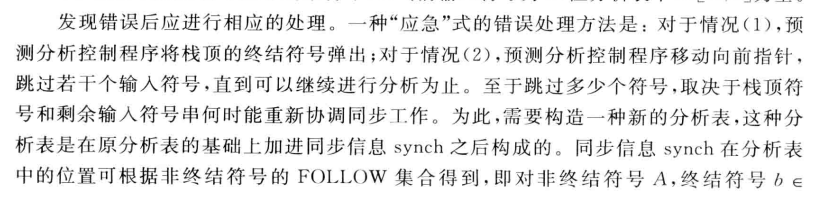
****

**5.函数定义设计**

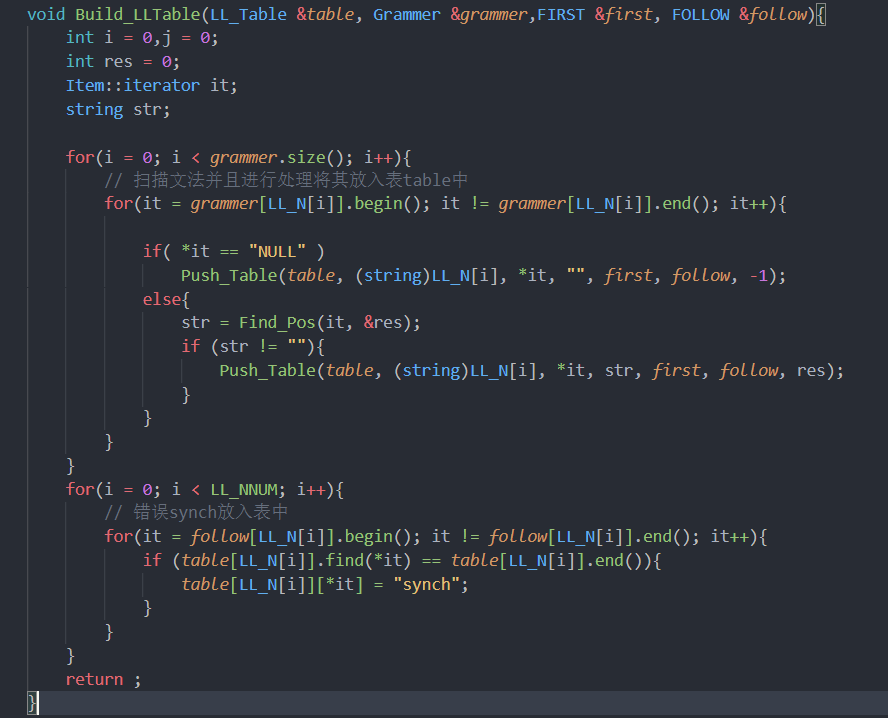
****

**6.LL预测分析表构造**

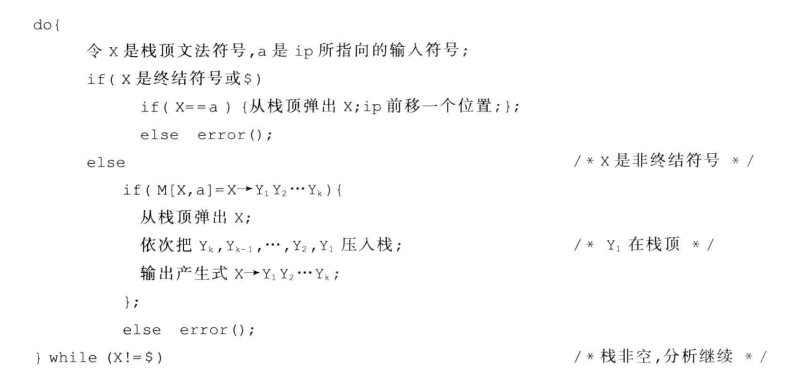
****

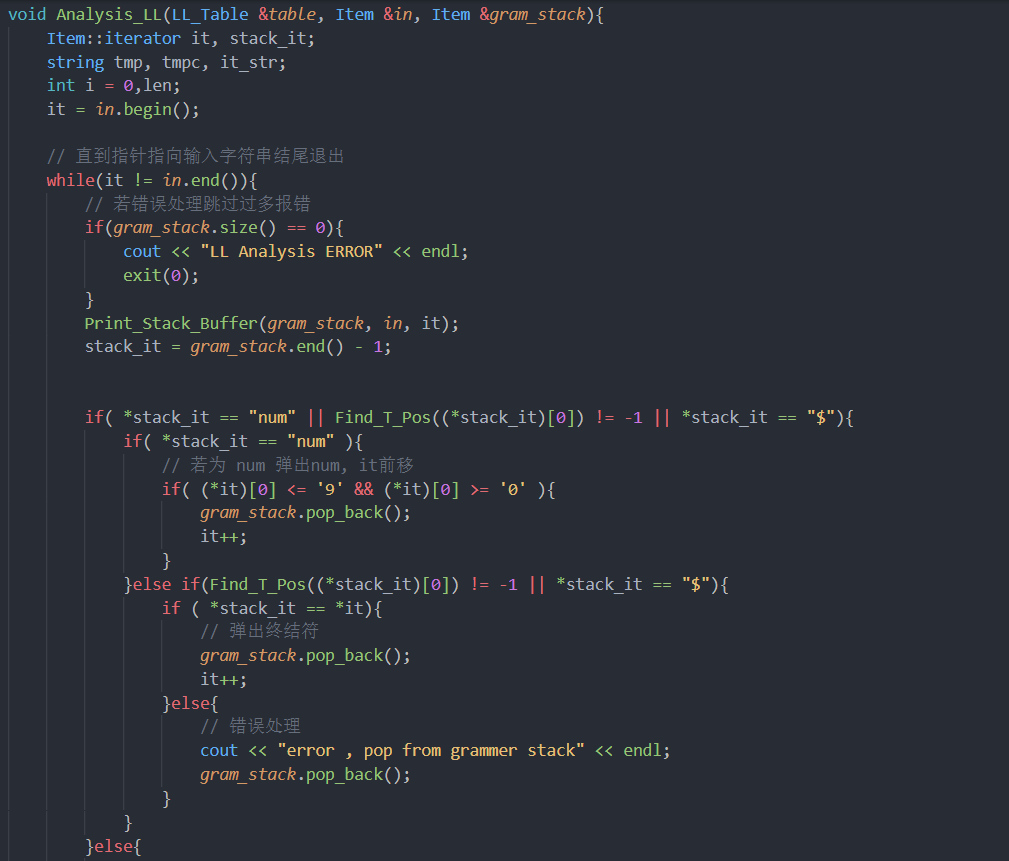
****

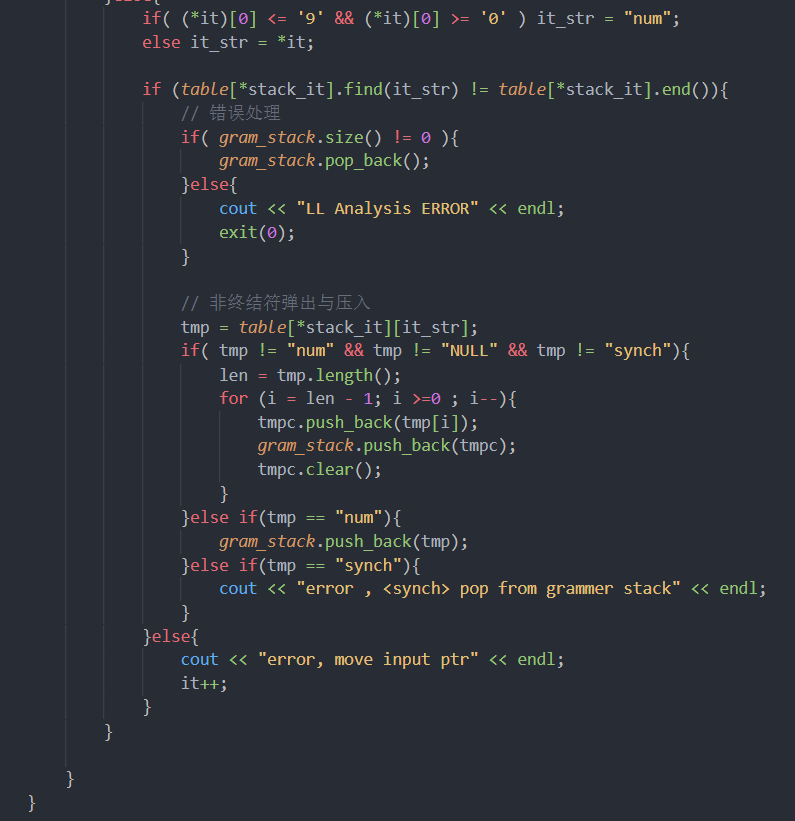
****



**7.LL分析过程**

****

****

****

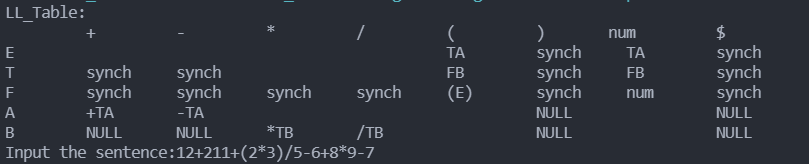
**8.LR分析过程**

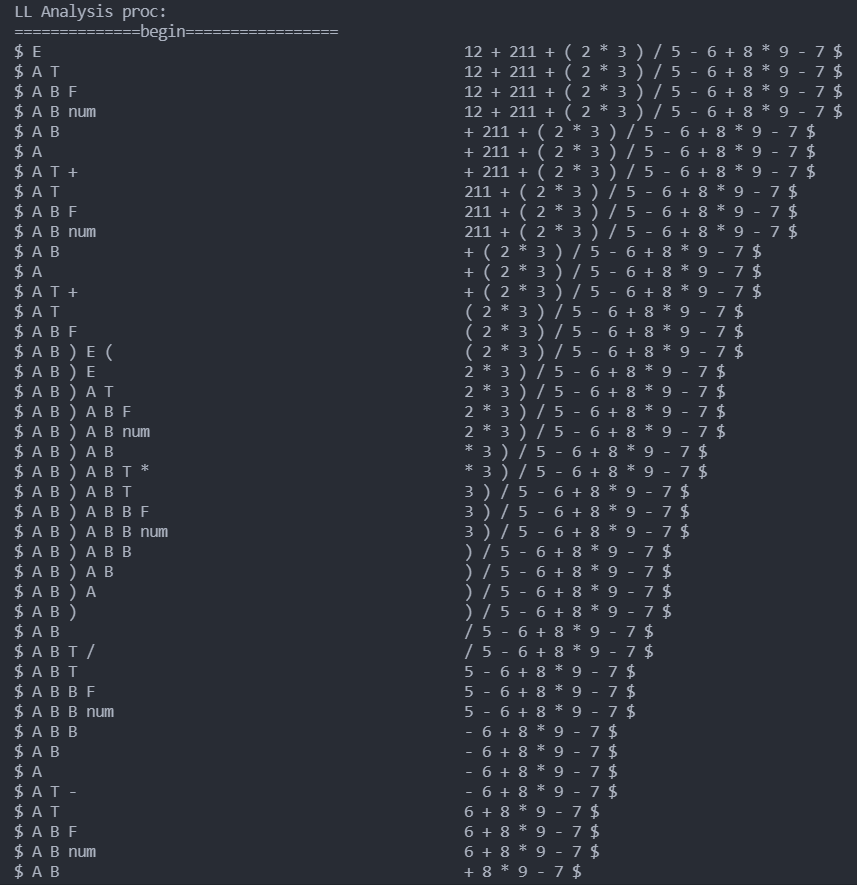
****

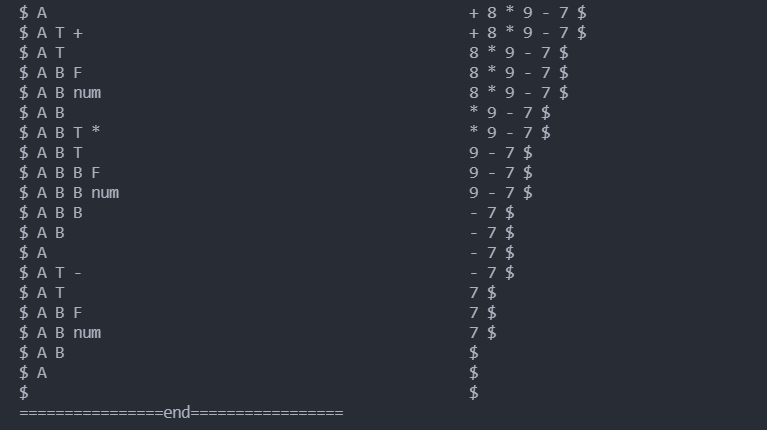
#### 四、测试报告

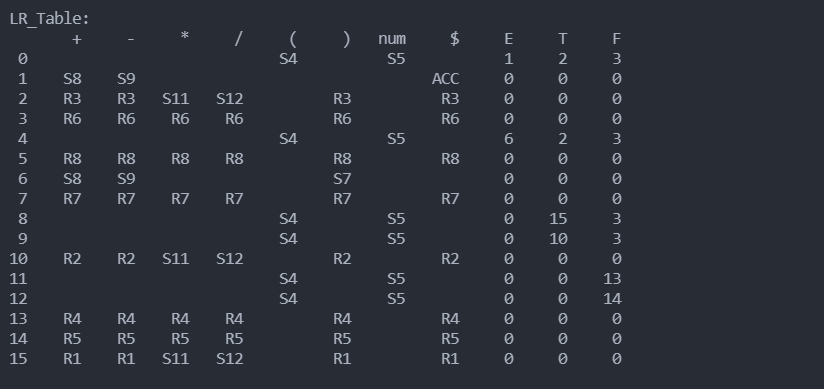
输入：12+211+(2\*3)/5-6+8\*9-7

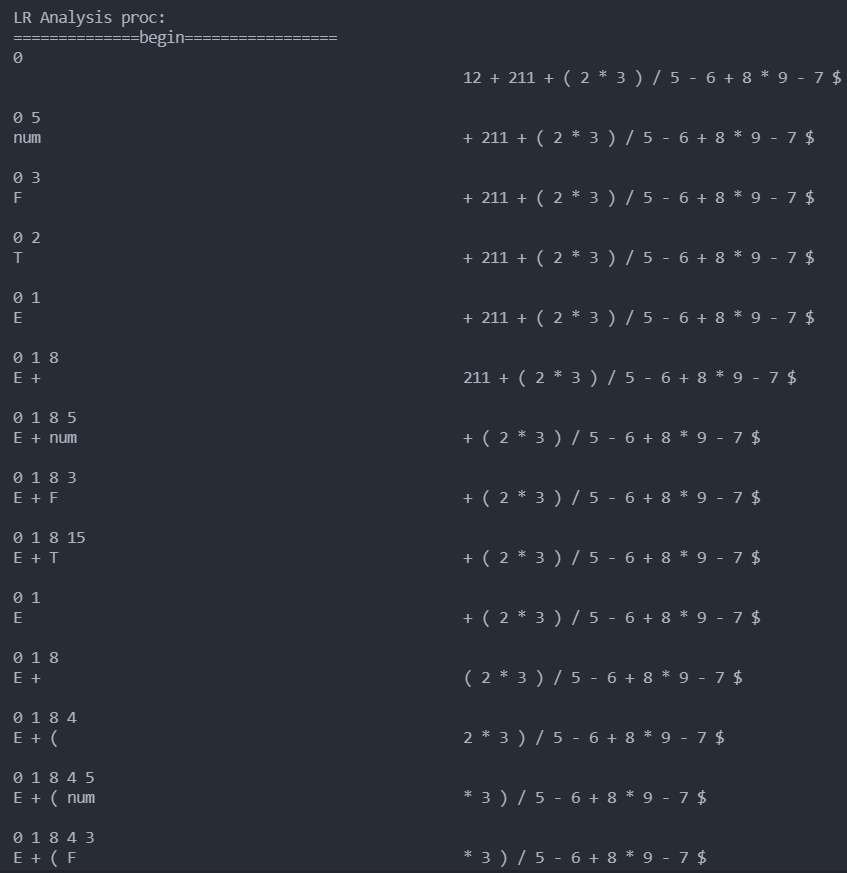
输出：

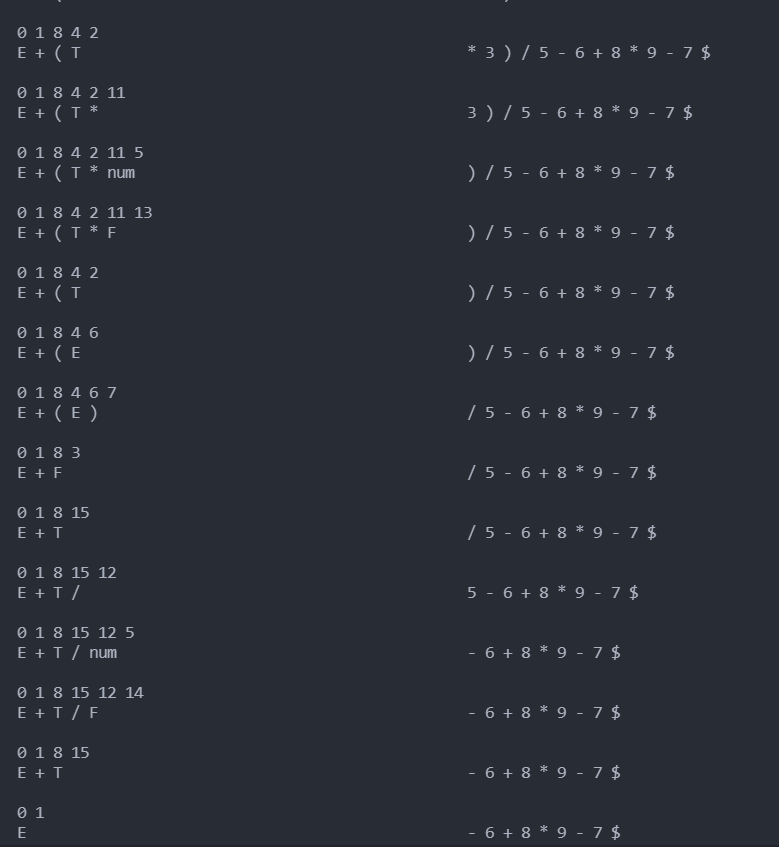


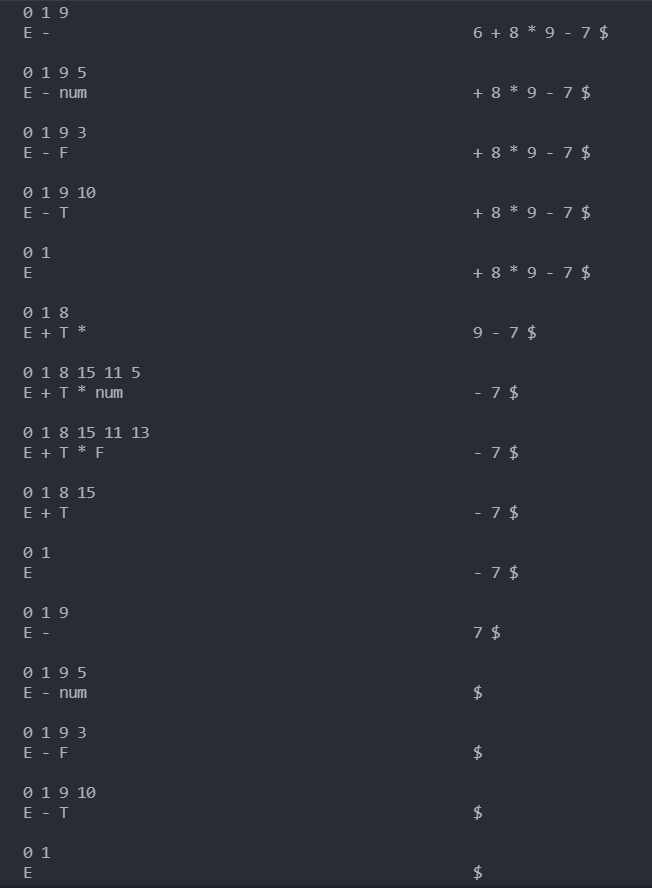










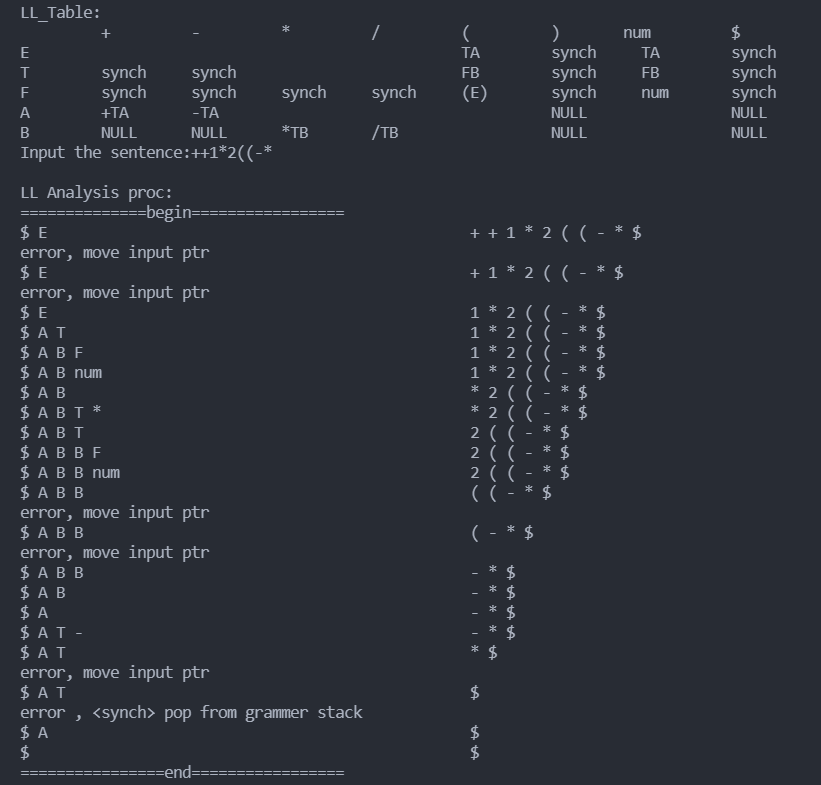


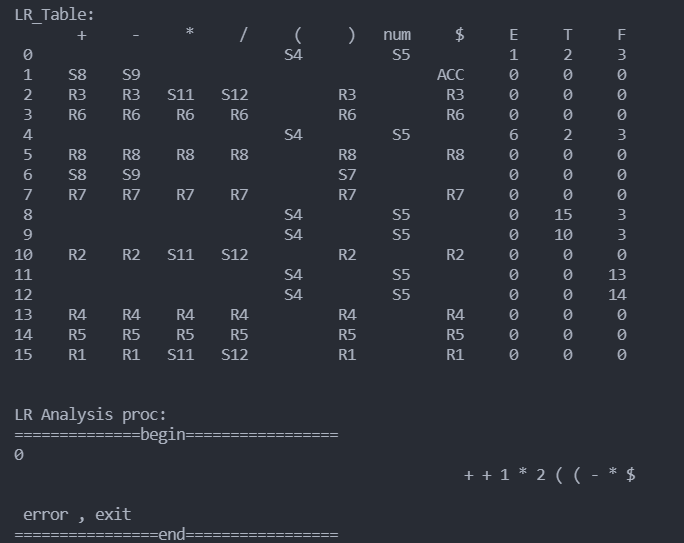


结果分析：改程序能够正确的给出LL和LR分析表并且进行LL和LR分析。

输入二：++1\*2((-\*

输出二：





结果分析：改程序能够正确进行LL的错误处理，对LR能够识别错误并且停止分析。

#### 五、源代码

|  |
| --- |
| #include <iostream>  #include <cstring>  #include <cstdlib>  #include <vector>  #include <map>  #include <stack>  #include <sstream>  #include <iomanip>  using namespace std;  #define TNUM 7      // 终结符个数  #define LL\_NNUM 5   // LL文法非终结符个数  #define LR\_NNUM 4   // LR文法非终结符个数  #define STATENUM 16 // LR项目集规范族个数(状态个数)  #define LINEMAX 50  // 输出时一行缓冲区大小  typedef vector<string> Item;  // LL分析用到的数据类型  typedef map<string, Item> FIRST;  typedef map<string, Item> FOLLOW;  typedef map<string, map<string, string>> LL\_Table;  typedef map<string, Item> Grammer;  // LR分析用到的数据类型  typedef map<int, map<string, string>> Action\_Table;  typedef map<int, map<string, int>> Goto\_Table;  typedef map<int, string> LR\_Grammer;  typedef vector<int> State\_Stack;  // 存储文法终结符和非终结符  const string LL\_T[] = {"+","-","\*","/","(",")","num","$"};  const string LR\_T[] = {"+", "-", "\*", "/", "(", ")", "num", "$"};  const string LL\_N[] = {"E","T","F","A","B"};  const string LR\_N[] = {"S", "E", "T", "F"};  // 初始化LL型去左递归文法  void Initial\_Grammer(Grammer &);  void Initial\_FirstFollow(FIRST &, FOLLOW &);  void Build\_LLTable(LL\_Table &, Grammer &, FIRST &, FOLLOW &);  // 与打印相关的函数  void Print\_Grammer(Grammer &);  void Print\_First(FIRST &);  void Print\_Follow(FOLLOW &);  void Print\_Stack\_Buffer(Item &, Item &, Item::iterator &);  void Print\_LL\_Table(LL\_Table &);  void Print\_LR\_Table(Action\_Table &, Goto\_Table &);  // LL分析相关函数  // 判断终结符和非终结符并从参数返回标志,返回字符串  string Find\_Pos(Item::iterator &, int \*);  // 构建 LL 分析表  void Push\_Table(LL\_Table &, string ,string ,string , FIRST &, FOLLOW &, int);  // 寻找终结符位置  int Find\_T\_Pos(char );  // 初始化LL分析程序  void Initial\_LL(string, LL\_Table &, Item &, Item &);  // LL分析程序  void Analysis\_LL(LL\_Table &, Item &, Item &);  // LR分析相关函数  // 初始化LR文法  void Initial\_LR\_Grammer(LR\_Grammer &);  // 构建LR文法分析表  void Build\_LRTable(Action\_Table &, Goto\_Table &);  // LR分析程序  void Analysis\_LR(LR\_Grammer &, Action\_Table &, Goto\_Table &, State\_Stack &, Item &, Item &);    int main()  {      int i = 0;      string input;      /\* LL Grammer Analysis var \*/      Grammer LL\_grammer;      FIRST LL\_first;      FOLLOW LL\_follow;      LL\_Table LL\_table;      Item in, LL\_gram\_stack;      /\* LR Grammer Analysis var \*/      LR\_Grammer LR\_grammer;      Action\_Table action\_table;      Goto\_Table goto\_table;      State\_Stack state\_stack;      Item symbol\_stack;        /\* LL Grammer Analysis \*/      Initial\_Grammer(LL\_grammer);      Initial\_FirstFollow(LL\_first, LL\_follow);      Build\_LLTable(LL\_table, LL\_grammer, LL\_first, LL\_follow);      Print\_LL\_Table(LL\_table);      cout << "Input the sentence:";      cin >> input;      Initial\_LL(input, LL\_table, in, LL\_gram\_stack);      cout << endl << "LL Analysis proc:" << endl;      cout << "==============begin=================" << endl;      Analysis\_LL(LL\_table, in, LL\_gram\_stack);      cout << "================end=================" << endl << endl;      /\* LR Grammer Analysis \*/      Initial\_LR\_Grammer(LR\_grammer);      Build\_LRTable(action\_table, goto\_table);      Print\_LR\_Table(action\_table,goto\_table);      cout << "LR Analysis proc:" << endl;      cout << "==============begin=================" << endl;      Analysis\_LR(LR\_grammer, action\_table, goto\_table, state\_stack, symbol\_stack, in);      cout << "================end=================" << endl;      return 0;  }  void Print\_Grammer(Grammer &*grammer*)  {      int i = 0;      Item::iterator it;      for (i = 0; i < 5; i++){          for (it = *grammer*[LL\_N[i]].begin(); it != *grammer*[LL\_N[i]].end(); it++){              cout << \*it << " ";          }          cout << endl;      }  }  void Print\_First(FIRST &*first*)  {      int i = 0;      Item::iterator it;      for (i = 0; i < 5; i++)      {          for (it = *first*[LL\_N[i]].begin(); it != *first*[LL\_N[i]].end(); it++)          {              cout << \*it << " ";          }          cout << endl;      }  }  void Print\_Follow(FOLLOW &*follow*)  {      int i = 0;      Item::iterator it;      for (i = 0; i < 5; i++)      {          for (it = *follow*[LL\_N[i]].begin(); it != *follow*[LL\_N[i]].end(); it++)          {              cout << \*it << " ";          }          cout << endl;      }  }  void Initial\_Grammer(Grammer &*grammer*){      Item item;      item.push\_back("TA");  *grammer*[LL\_N[0]] = item;      item.clear();      item.push\_back("FB");  *grammer*[LL\_N[1]] = item;      item.clear();      item.push\_back("(E)");item.push\_back("num");  *grammer*[LL\_N[2]] = item;      item.clear();      item.push\_back("+TA");item.push\_back("-TA");item.push\_back("NULL");  *grammer*[LL\_N[3]] = item;      item.clear();      item.push\_back("\*TB");item.push\_back("/TB");item.push\_back("NULL");  *grammer*[LL\_N[4]] = item;      item.clear();  }  void Initial\_FirstFollow(FIRST &*first*, FOLLOW &*follow*){      /\* make First set begin\*/      Item item;      item.push\_back("(");item.push\_back("num");  *first*[LL\_N[0]] = item;  *first*[LL\_N[1]] = item;  *first*[LL\_N[2]] = item;      item.clear();      item.push\_back("+");item.push\_back("-");item.push\_back("NULL");  *first*[LL\_N[3]] = item;      item.clear();      item.push\_back("\*");item.push\_back("/");item.push\_back("NULL");  *first*[LL\_N[4]] = item;      item.clear();      /\* make First set finish\*/      /\* make FOLLOW set begin\*/      item.push\_back("$");      item.push\_back(")");  *follow*[LL\_N[0]] = item;  *follow*[LL\_N[3]] = item;      item.clear();      item.push\_back("+");item.push\_back("-");item.push\_back("$");item.push\_back(")");  *follow*[LL\_N[1]] = item;  *follow*[LL\_N[4]] = item;      item.clear();      item.push\_back("\*");item.push\_back("/");item.push\_back("+");item.push\_back("-");item.push\_back("$");item.push\_back(")");  *follow*[LL\_N[2]] = item;      item.clear();      /\* make FOLLOW set finish\*/  }  void Build\_LLTable(LL\_Table &*table*, Grammer &*grammer*,FIRST &*first*, FOLLOW &*follow*){      int i = 0,j = 0;      int res = 0;      Item::iterator it;      string str;      for(i = 0; i < *grammer*.size(); i++){          // 扫描文法并且进行处理将其放入表table中          for(it = *grammer*[LL\_N[i]].begin(); it != *grammer*[LL\_N[i]].end(); it++){                if( \*it == "NULL" )                  Push\_Table(*table*, (string)LL\_N[i], \*it, "", *first*, *follow*, -1);              else{                  str = Find\_Pos(it, &res);                  if (str != ""){                      Push\_Table(*table*, (string)LL\_N[i], \*it, str, *first*, *follow*, res);                  }              }          }      }      for(i = 0; i < LL\_NNUM; i++){          // 错误synch放入表中          for(it = *follow*[LL\_N[i]].begin(); it != *follow*[LL\_N[i]].end(); it++){              if (*table*[LL\_N[i]].find(\*it) == *table*[LL\_N[i]].end()){  *table*[LL\_N[i]][\*it] = "synch";              }          }      }      return ;  }  string Find\_Pos(Item::iterator &*it*, int \**res*){      int i = 0;      string tmp;      if ('a' <= (\**it*).c\_str()[0] && (\**it*).c\_str()[0] <= 'z')          tmp = \**it*;      else          tmp = (\**it*).c\_str()[0];      for(i = 0; i < LL\_NNUM; i++){          if(tmp == LL\_N[i]){              \**res* = 0;              return LL\_N[i];          }      }      for(i = 0; i < TNUM; i++){          if(tmp == LL\_T[i]){              \**res* = 1;              return LL\_T[i];          }      }        return "";  }  void Push\_Table(LL\_Table &*table*, string *N*, string *now*, string *str*, FIRST &*first*, FOLLOW &*follow*, int *tag*){      int i = 0, j = 0;      Item::iterator it;      if( *tag* == 0 ){          for (it = *first*[*str*].begin(); it != *first*[*str*].end(); it++){  *table*[*N*][\*it] = *now*;          }      }else if(*tag* == 1){  *table*[*N*][*str*] = *now*;      }else{          for(it = *follow*[*N*].begin(); it != *follow*[*N*].end(); it++){  *table*[*N*][\*it] = *now*;          }      }  }  int Find\_T\_Pos(char *c*){      int i = 0;      string tmp(1,*c*);      for(i = 0; i < TNUM; i++){          if(tmp == LL\_T[i]){              return i;          }      }      return -1;  }  void Initial\_LL(string *input*, LL\_Table &*table*, Item &*in*, Item &*gram\_stack*){      int len, i, j, pos;      string tmp;      len = *input*.length();      for(i = 0; i < len; i++){          tmp.clear();          if(*input*[i] <= '9' && *input*[i] >= '0'){              while (*input*[i] <= '9' && *input*[i] >= '0'){                  tmp += *input*[i];                  i++;              }  *in*.push\_back(tmp);              i -= 1;          }else{              pos = Find\_T\_Pos(*input*[i]);              if (pos == -1){                  cout << "error input" << endl;                  exit(0);              }else{                  tmp = *input*[i];  *in*.push\_back(tmp);              }          }      }  *in*.push\_back("$");  *gram\_stack*.push\_back("$");  *gram\_stack*.push\_back("E");  }  void Print\_Stack\_Buffer(Item &*grammer*, Item &*in*, Item::iterator &*it*){      int i;      Item::iterator tmp;      stringstream out;      for(tmp = *grammer*.begin(); tmp != *grammer*.end(); tmp++){          out << \*tmp << " ";      }      for(i = out.str().length();i < LINEMAX; i++){          out << " ";      }      for(tmp = *it*; tmp != *in*.end(); tmp++){          out << \*tmp << " ";      }      cout << out.str() << endl;  }  void Analysis\_LL(LL\_Table &*table*, Item &*in*, Item &*gram\_stack*){      Item::iterator it, stack\_it;      string tmp, tmpc, it\_str;      int i = 0,len;      it = *in*.begin();      // 直到指针指向输入字符串结尾退出      while(it != *in*.end()){          // 若错误处理跳过过多报错          if(*gram\_stack*.size() == 0){              cout << "LL Analysis ERROR" << endl;              exit(0);          }          Print\_Stack\_Buffer(*gram\_stack*, *in*, it);          stack\_it = *gram\_stack*.end() - 1;            if( \*stack\_it == "num" || Find\_T\_Pos((\*stack\_it)[0]) != -1 || \*stack\_it == "$"){              if( \*stack\_it == "num" ){                  // 若为 num 弹出num, it前移                  if( (\*it)[0] <= '9' && (\*it)[0] >= '0' ){  *gram\_stack*.pop\_back();                      it++;                  }              }else if(Find\_T\_Pos((\*stack\_it)[0]) != -1 || \*stack\_it == "$"){                  if ( \*stack\_it == \*it){                      // 弹出终结符  *gram\_stack*.pop\_back();                      it++;                  }else{                      // 错误处理                      cout << "error , pop from grammer stack" << endl;  *gram\_stack*.pop\_back();                  }              }          }else{              if( (\*it)[0] <= '9' && (\*it)[0] >= '0' ) it\_str = "num";              else it\_str = \*it;              if (*table*[\*stack\_it].find(it\_str) != *table*[\*stack\_it].end()){                  // 错误处理                  if( *gram\_stack*.size() != 0 ){  *gram\_stack*.pop\_back();                  }else{                      cout << "LL Analysis ERROR" << endl;                      return ;                  }                  // 非终结符弹出与压入                  tmp = *table*[\*stack\_it][it\_str];                  if( tmp != "num" && tmp != "NULL" && tmp != "synch"){                      len = tmp.length();                      for (i = len - 1; i >=0 ; i--){                          tmpc.push\_back(tmp[i]);  *gram\_stack*.push\_back(tmpc);                          tmpc.clear();                      }                  }else if(tmp == "num"){  *gram\_stack*.push\_back(tmp);                  }else if(tmp == "synch"){                      cout << "error , <synch> pop from grammer stack" << endl;                  }              }else{                  cout << "error, move input ptr" << endl;                  it++;              }          }        }  }  void Print\_LL\_Table(LL\_Table &*LL\_table*){      int i = 0, j = 0;      cout << "LL\_Table:" << endl;      for(i = 0; i < TNUM + 1; i++){          cout << setw(10) << LL\_T[i];      }      cout << endl;      for(i = 0; i < LL\_NNUM ; i++){          cout << LL\_N[i] << "        ";          for(j = 0; j < TNUM + 1; j++){              if (*LL\_table*[LL\_N[i]].find(LL\_T[j]) != *LL\_table*[LL\_N[i]].end()){                  cout << setiosflags(ios::left) << setw(10) << *LL\_table*[LL\_N[i]][LL\_T[j]];              }else{                  cout << setw(10) << " ";              }          }          cout << endl;      }  }  void Initial\_LR\_Grammer(LR\_Grammer &*grammer*){  *grammer*[0] = "S->E";  *grammer*[1] = "E->E+T";  *grammer*[2] = "E->E-T";  *grammer*[3] = "E->T";  *grammer*[4] = "T->T\*F";  *grammer*[5] = "T->T/F";  *grammer*[6] = "T->F";  *grammer*[7] = "F->(E)";  *grammer*[8] = "F->num";  }  void Build\_LRTable(Action\_Table &*action\_t*, Goto\_Table &*goto\_t*){      int i = 0,j = 0;      string R\_term[] = {"+","-","\*","/",")","$"};      map<int,string> tmp\_map;      tmp\_map[2] = "3";      tmp\_map[3] = "6";      tmp\_map[5] = "8";      tmp\_map[7] = "7";      tmp\_map[10] = "2";      tmp\_map[13] = "4";      tmp\_map[14] = "5";      tmp\_map[15] = "1";      for(i = 0; i < STATENUM; i++){          if(tmp\_map.find(i) != tmp\_map.end()){              for(j = 0; j < 6; j++){  *action\_t*[i][R\_term[j]] = "R" + tmp\_map[i];              }          }      }    *action\_t*[0]["("] = "S4";*action\_t*[0]["num"] = "S5";  *action\_t*[1]["+"] = "S8";*action\_t*[1]["-"] = "S9";*action\_t*[1]["$"] = "ACC";  *action\_t*[2]["\*"] = "S11";*action\_t*[2]["/"] = "S12";  *action\_t*[4]["("] = "S4";*action\_t*[4]["num"] = "S5";  *action\_t*[6]["+"] = "S8";*action\_t*[6]["-"] = "S9";*action\_t*[6][")"] = "S7";  *action\_t*[8]["("] = "S4";*action\_t*[8]["num"] = "S5";  *action\_t*[9]["("] = "S4";*action\_t*[9]["num"] = "S5";  *action\_t*[10]["\*"] = "S11";*action\_t*[10]["/"] = "S12";  *action\_t*[11]["("] = "S4";*action\_t*[11]["num"] = "S5";  *action\_t*[12]["("] = "S4";*action\_t*[12]["num"] = "S5";  *action\_t*[15]["\*"] = "S11";*action\_t*[15]["/"] = "S12";  *goto\_t*[0]["E"] = 1;*goto\_t*[0]["T"] = 2;*goto\_t*[0]["F"] = 3;  *goto\_t*[4]["E"] = 6;*goto\_t*[4]["T"] = 2;*goto\_t*[4]["F"] = 3;  *goto\_t*[8]["T"] = 15;*goto\_t*[8]["F"] = 3;  *goto\_t*[9]["T"] = 10;*goto\_t*[9]["F"] = 3;  *goto\_t*[11]["F"] = 13;  *goto\_t*[12]["F"] = 14;  }  void Print\_LR\_Table(Action\_Table &*action\_t*, Goto\_Table &*goto\_t*){      int i = 0, j = 0;      cout << "LR\_Table:" << endl;      cout << setiosflags(ios::right);      cout << "  ";      for(i = 0; i < TNUM + 1; i++){          cout << setw(6) << LR\_T[i];      }      for(i = 1; i < LR\_NNUM ; i++){          cout << setw(6) << LR\_N[i];      }      cout << endl;      for(i = 0; i < STATENUM ; i++){          cout << setiosflags(ios::right) << setw(2) << i;          for(j = 0; j < TNUM + 1; j++){              if (*action\_t*[i].find(LR\_T[j]) != *action\_t*[i].end()){                  cout << setiosflags(ios::right) << setw(6) << *action\_t*[i][LR\_T[j]];              }else{                  cout << "      " << setw(6);              }          }          for(j = 1; j < LR\_NNUM ; j++){              cout << setw(6) << *goto\_t*[i][LR\_N[j]];          }          cout << endl;      }      cout << endl << endl;  }  void Analysis\_LR(LR\_Grammer &*grammer*, Action\_Table &*action\_t*, Goto\_Table &*goto\_t*, State\_Stack &*state\_stack*, Item &*symbol\_stack*, Item &*in*){      // initial state\_stack  *state\_stack*.push\_back(0);      Item::iterator it = *in*.begin(), symbol\_it;      int tmp\_num, i, len;      string tmp\_str, action\_str, reduce\_str;      State\_Stack::iterator state\_it;      while (1){          symbol\_it = *symbol\_stack*.end() - 1;          state\_it = *state\_stack*.end() - 1;          for(i = 0; i < *state\_stack*.size(); i++){              cout << *state\_stack*[i] << " ";          }          cout << endl;          Print\_Stack\_Buffer(*symbol\_stack*, *in*, it);          cout << endl;          if( (\*it)[0] <= '9' && (\*it)[0] >= '0' ){              tmp\_str = "num";          }else{              tmp\_str = \*it;          }          action\_str = *action\_t*[\*state\_it][tmp\_str];          if( action\_str[0] == 'S' ){              // 移进处理              tmp\_num = 0;              for(i = 1; i < action\_str.length(); i++){                  tmp\_num = tmp\_num \* 10 + action\_str[i] - '0';              };  *state\_stack*.push\_back(tmp\_num);  *symbol\_stack*.push\_back(tmp\_str);              it++;          }else if( action\_str[0] == 'R' ){              // 规约处理              tmp\_num = 0;              for(i = 1; i < action\_str.length(); i++){                  tmp\_num = tmp\_num \* 10 + action\_str[i] - '0';              };              if( tmp\_num != 8 ) len = *grammer*[tmp\_num].length() - 3;              else len = 1;              while(len--){  *state\_stack*.pop\_back();  *symbol\_stack*.pop\_back();              }              reduce\_str = *grammer*[tmp\_num][0];  *symbol\_stack*.push\_back(reduce\_str);  *state\_stack*.push\_back(*goto\_t*[\*(*state\_stack*.end() - 1)][reduce\_str]);          }          else if (action\_str == "ACC"){              // 接受              cout << "success accept" << endl;              return ;          }else{              // 错误              cout << " error , exit" << endl;              return ;          }        }  } |