

**教学上机实验报告**

**课程名称：**

**任课教师：**

**学生学号：**

**学生姓名：**

**专业班级：**

**2021 ～ 2022学年 第 一 学期**

|  |
| --- |
| **河南理工大学**  **教学上机实验报告评价分值标准** |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 序号 | 评价指标 | 分值 | 评价等级及参考分值 | | | | | 评价分 | | 优 | 良 | 中 | 合格 | 差 | | 1 | 实验报告内容完整充实 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 2 | 实验内容书写规范、字迹工整认真 | 10 | 10 | 8 | 7 | 6 | 3 |  | | 3 | 实验过程叙述详细、概念正确，语言表达准确，结构严谨，调理清楚，逻辑性强，自己努力完成，没有抄袭。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 4 | 对实验过程中存在的问题分析详细透彻、深刻、全面、规范、，结合实验内容，有自己的个人见解和想法，并能结合该实验提出相关问题，给出解决方法。 | 30 | 30 | 26 | 23 | 20 | 10 |  | | 5 | 实验结果、分析和结论正确无误 | 20 | 20 | 17 | 15 | 13 | 6 |  | | 总得分 | | | | | | | |  |   签名（签章）：  日期： 年 月 日 |
|  |

|  |
| --- |
| **河南理工大学教学上机实验报告** |
| 上机时间 2021 年 月 日 |
| **实验题目：**  关于上下文无关文法的程序实现 |
| **实验目的和要求：**  目的：  理解文法在语言描述中的作用；加深对文法的理解；掌握基于文法产生句子的  实现方法。  要求：  （1） 在程序中采用合适的数据类型存放文法中的产生规则集合；  （2） 基于手工的推导过程，利用代码进行实现推导过程；  （3） 需要判断产生的字符串是否与目标句子一致，并展示最终句子。 |
| **实验过程：**  使用c++实现，可以实现对由(i、+、-、\*、/)构成的文法表达式的最左和最右推导规则输出，适合的规则有以下三条：  E → T|E+T|E-T  T → F|T\*F|T/F  F → (E)|i  执行过程如下：  1. 输入待推导的表达式  具体分为七个部分，分别对应  void DeduceE(int l, int r);// 文法第一行  void DeduceT(int l, int r);// 文法第二行  void DeduceF(int l, int r);// 文法第三行  void DeDeduceE(int l, int r);// 文法第一行  void DeDeduceT(int l, int r);// 文法第二行  void DeDeduceF(int l, int r);// 文法第三行  void menu();//目录  main函数使用循环输入表达式选择最左还是最右推导，结束后按1重新输入其他键退出，如果发现不合法输入，重新输入；调用DeduceE或者DeDeduceE(第一行规则的判定)执行最右还是最左推导  2. 对表达式字符串进行第一行文法判定  判断是否有 + or - 运算符括号内除外，是则输出 E→E+T or E→E-T ， + 之前的所有字符返回进行 2 ， + 之后的字符进行 3 ；否则进行下一步。  输出 E→T ，所有字符串进行 3 。  3. 对表达式字符串进行第二行文法判定  判断是否有 \* or / 运算符括号内除外，是则输出 E→E\*T or E→E/T ， + 之前的所有字符返回进行 2 ， + 之后的字符进行 3 ；否则进行下一步。  输出 T→F ，所有字符串进行 4 。  4. 对表达式字符串进行第三行文法判定  第一个字符是否为 ( ，是则输出 F→(E) ，然后将该字符串的 ( 、 ) 删除，返回 2 ；否则输出 F →i ，操作结束。  ```cpp  #include<iostream>  #include<string.h>  using namespace std;  // 全局变量  char str[10];// 存储  int step;// 记录推导步长  void DeduceE(int l, int r);// 文法第一行  void DeduceT(int l, int r);// 文法第二行  void DeduceF(int l, int r);// 文法第三行  void DeDeduceE(int l, int r);// 文法第一行  void DeDeduceT(int l, int r);// 文法第二行  void DeDeduceF(int l, int r);// 文法第三行  void menu();  // 主函数  int main() {  string choose = "1";  while (choose == "1") {  system("cls");  menu();  int error = 1;  while (error == 1) { // 输入句子，简单判错  cout << "请输入该文法的句子：" << endl << ">>";  cin >> str;  for (int i = 0; i < strlen(str); i++) {  if (str[i] != 'i' && str[i] != '+' && str[i] != '\*' && str[i] != '/' && str[i] != '(' &&  str[i] != ')') {  cout << "输入有误请重新输入！表达式应只含有“i,+,\*,/,(,)”等字符！" << endl;  error = 1;  break;  } else  error = 0;  }  }  int choice;  cout << "请选择" << endl << "1.最右" << endl << "2.最左" << endl << ">>";  cin >> choice;  switch (choice) {  case 1:  step = 1;  DeduceE(0, strlen(str));// 进行文法的推导  break;  case 2:  step = 1;  DeDeduceE(0, strlen(str));// 反向进行文法的推导  break;  }  cout << endl << "推导完毕！" << endl << endl;  cout << "输入 1 继续推导其他句子，输入其他退出。" << endl;  cin >> choose;  }  }  void menu() {  string P[3][4];  P[0][0] = "E";  P[0][1] = "T";  P[0][2] = "E+T";  P[0][3] = "E-T";  P[1][0] = "T";  P[1][1] = "F";  P[1][2] = "T\*F";  P[1][3] = "T/F";  P[2][0] = "F";  P[2][1] = "(E)";  P[2][2] = "i";  cout << "文法如下：" << endl;// 输出文法  for (int i = 0; i < 3; i++) {  if (i < 2)  cout << P[i][0] << " → " << P[i][1] << "|" << P[i][2] << "|" << P[i][3];  else  cout << P[i][0] << " → " << P[i][1] << "|" << P[i][2];  cout << endl;  }  }  // 首字符 F，第三行文法  void DeduceF(int l, int r) {  char a[10];  int i, j = 0;  if (str[l] == '(') { // 判断是否含括号  cout << "step " << step << ":" << "F->(E)" << endl;  step++;  for (i = 0; i < strlen(str); i++)  if (str[i] != ')')  a[j++] = str[i];  memset(str, 0, sizeof(str));  for (i = 0; i < strlen(a); i++)  str[i] = a[i];  DeduceE(l + 1, strlen(str));// 返回文法第一行  } else { // 无括号，直接输出值 i  cout << "step " << step << ":" << "F->i" << endl;  step++;  }  }  // 首字符 T，第二行文法  void DeduceT(int l, int r) {  int bracket = 0;// 记录括号  int symbol = 0;// 记录符号  for (int i = r - 1; i >= l; i--) { // 判断字符串是否含括号和运算符  if (str[i] == ')')  bracket++;  if (str[i] == '(')  bracket--;  if (!bracket) { // 没有括号或不在括号内部，则进行运算符操作（不对括号内的内容进行运算符处理）  if (str[i] == '\*') {  symbol = 1;  cout << "step " << step << ":" << "T->T\*F" << endl;  step++;  DeduceF(l, i);// 转去进行文法第三行判断  DeduceT(i + 1, r);// 继续进行文法第一行判断  break;  } else if (str[i] == '/') {  symbol = 1;  cout << "step " << step << ":" << "T->T/F" << endl;  step++;  DeduceF(l, i);  DeduceT(i + 1, r);  break;  }  }  }  if (!symbol) { // 字符串既没括号，也没字符  cout << "step " << step << ":" << "T->F" << endl;  step++;  DeduceF(l, r);// 转去进行文法第三行判断  }  }  // 首字符 E，第一行文法  void DeduceE(int l, int r) {  int bracket = 0;// 记录括号  int symbol = 0;// 记录符号  for (int i = r - 1; i >= l; i--) { // 判断字符串是否含括号和运算符  if (str[i] == ')')  bracket++;  if (str[i] == '(')  bracket--;  if (!bracket) { // 没有括号或不在括号内部，则进行运算符操作（不对括号内的内容进行运算符处理）  if (str[i] == '+') {  symbol = 1;  cout << "step " << step << ":" << "E->E+T" << endl;  step++;  DeduceT(l, i);// 转去进行文法第二行判断  DeduceE(i + 1, r);// 继续进行文法第一行判断  break;  } else if (str[i] == '-') {  symbol = 1;  cout << "step " << step << ":" << "E->E-T" << endl;  step++;  DeduceT(l, i);  DeduceE(i + 1, r);  break;  }  }  }  if (!symbol) { // 字符串既没括号，也没字符  cout << "step " << step << ":" << "E->T" << endl;  step++;  DeduceT(l, r);// 转去进行文法第二行判断  }  }  // 首字符 F，第三行文法  void DeDeduceF(int l, int r) {  char a[10];  int i, j = 0;  if (str[l] == '(') { // 判断是否含括号  cout << "step " << step << ":" << "F->(E)" << endl;  step++;  for (i = 0; i < strlen(str); i++)  if (str[i] != ')')  a[j++] = str[i];  memset(str, 0, sizeof(str));  for (i = 0; i < strlen(a); i++)  str[i] = a[i];  DeDeduceE(l + 1, strlen(str));// 返回文法第一行  } else { // 无括号，直接输出值 i  cout << "step " << step << ":" << "F->i" << endl;  step++;  }  }  // 首字符 T，第二行文法  void DeDeduceT(int l, int r) {  int bracket = 0;// 记录括号  int symbol = 0;// 记录符号  for (int i = r - 1; i >= l; i--) { // 判断字符串是否含括号和运算符  if (str[i] == ')')  bracket++;  if (str[i] == '(')  bracket--;  if (!bracket) { // 没有括号或不在括号内部，则进行运算符操作（不对括号内的内容进行运算符处理）  if (str[i] == '\*') {  symbol = 1;  cout << "step " << step << ":" << "T->T\*F" << endl;  step++;  DeDeduceT(i + 1, r);// 继续进行文法第一行判断  DeDeduceF(l, i);// 转去进行文法第三行判断  break;  } else if (str[i] == '/') {  symbol = 1;  cout << "step " << step << ":" << "T->T/F" << endl;  step++;  DeDeduceT(i + 1, r);  DeDeduceF(l, i);  break;  }  }  }  if (!symbol) { // 字符串既没括号，也没字符  cout << "step " << step << ":" << "T->F" << endl;  step++;  DeDeduceF(l, r);// 转去进行文法第三行判断  }  }  // 首字符 E，第一行文法  void DeDeduceE(int l, int r) {  int bracket = 0;// 记录括号  int symbol = 0;// 记录符号  for (int i = r - 1; i >= l; i--) { // 判断字符串是否含括号和运算符  if (str[i] == ')')  bracket++;  if (str[i] == '(')  bracket--;  if (!bracket) { // 没有括号或不在括号内部，则进行运算符操作（不对括号内的内容进行运算符处理）  if (str[i] == '+') {  symbol = 1;  cout << "step " << step << ":" << "E->E+T" << endl;  step++;  DeDeduceE(i + 1, r);// 继续进行文法第一行判断  DeDeduceT(l, i);// 转去进行文法第二行判断  break;  } else if (str[i] == '-') {  symbol = 1;  cout << "step " << step << ":" << "E->E-T" << endl;  step++;  DeDeduceE(i + 1, r);  DeDeduceT(l, i);  break;  }  }  }  if (symbol == 0) { // 字符串既没括号，也没字符  cout << "step " << step << ":" << "E->T" << endl;  step++;  DeDeduceT(l, r);// 转去进行文法第二行判断  }  }  ``` |
| **实验结果：** |
| **实验分析：**  实验使我对上下文无关文法如何理解、使用规则有了基本了解，且尝试根据带判断表达式判断应该哪条文法规则，懂得通过单步执行调试cpp程序。总之，该程序设计还有所欠缺，没有实现直观的推导过程，因为没有引入栈或者其他数据结构对过程进行存储。  **出错分析：**  例如，对于第三行规则的判定，如何识别括号，一开始没有定义变量标记括号，使用ASCII码识别，对于单层括号使用，但是嵌套的括号就不适用了，出现错误。 |
| **实验成绩：**  日期： 年 月 日 |

|  |
| --- |
| **河南理工大学教学上机实验报告** |
| 上机时间 2021 年 11 月 28 日 |
| **实验题目：**  词法分析器的设计 |
| **实验目的和要求：**  目的**：**  理解词法分析在编译程序中的作用；加深对有穷自动机模型的理解；掌握词法分析程序的实现方法和技术。  要求**：**  1.待分析的简单的语法  关键字： if else for while main return int  运算符和界符： = + - \* / < <= > >= <> = ; ( ) #  其他单词是标识符 id 和整型常数 num，通过以下正规式定义：  id=l(l|d)\*  num=dd\*  空格、注释：在词法分析中要去掉。  2.设计各种单词符号对应的种别编码 |
| **实验过程：**  **需求分析**  根据此法规则，识别字，返回相应的种别码存入暂存变量，循环输出它和对应数组中的token，忽略空白字符，若读入不能识别的字，返回”显示有误”，并显示错误位置。  **分析词法规则**  <字母>::=a | b | c | ... | x | y | z  <数字>::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  <标识符>::=<字母> | <标识符><字母> | <标识符><数字>  <数字序列>::＝<数字> | <数字序列><数字>  <关系运算符>::＝< | > | != | >= | <= | ==  <运算符>::＝+ | - | \* | /  <分界符>::＝, | ; | ( | ) | { | }  <关键字>::＝main | if | else | do | while | for | switch | case | int | double | float | long | void  **种别码**   |  |  | | --- | --- | | "main,"if","else","do","while","for","switch",  "case","int","double","float","long","void" | 3 | | 标识符，注释内容 | 2 | | 数字 | 1 | | ( | 16 | | ) | 17 | | { | 33 | | } | 34 | | + | 18 | | ++ | 29 | | - | 19 | | -- | 30 | | = | 22 | | == | 23 | | > | 24 | | >= | 25 | | < | 26 | | <= | 27 | | ; | 28 | | " | 29 | | , | 35 | | ! | 36 | | != | 37 | | # | 38 | | \* | 20 | | \*/ | 32 | | / | 21 | | /\* | 31 | | // | 39 | | 其它 | 识别有误 |   **设计思路**   1. 若读到的是字母，就再接着读，直到读到的既不是字母也不是数字也不是下划线，并将读到的写入到token数组； 2. 若读到的是数字，直到读到的不是数字或小数点，将读到的写入到token数组； 3. 若读到的是<、>、=，则再读入下一位，若为=，则该运算符为<=、>=、==，若为其他字符，则返回<、>、=的种别码； 4. 若读到的是/，则读下一位，若为\*，则说明之后为注释内容，一直读入直到读入\*，并判断下一位是否为/，是则注释结束，不是继续读入下一位；若读入\n，行数加一，若读入的字符与以上都不匹配，则“识别有误”，并输出错误行数。 5. 对于字母，循环比较其是否是关键字之一，若是则返回种别码3，不是则返回种别码2。   **基本数据类型**  char ch;  char token[13];  **函数设计**  void getbc(); //跳过空格,当读入的ch为’\n’,则行数row加1  void m\_getch(); //从缓冲区读入下一个字符  int scanner(); //判断种别码  void init\_token(); //对token进行初始化  int digit(); //判断ch中的字符是否为数字  int letter(); //判断ch中的字符是否为字母  int reserve(); //在判断关键词或是标识符之前先进行是否是字母的判断，对token中的字符串和keyword数组中的字符串匹配，若是一个关键字则返回3，否则返回标识符2  **具体代码**  ```cpp  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <cstdio>  #include <iostream>  using namespace std;  const int max\_word = 505;  char token[12];  char in[105];  FILE\* fin, \* fout;  int cnt = 0, token\_num = 0;  int row = 1;  int flag = 0;  char ch;  char datain[255];  //关键字  const char keyWord[13][20] = { "main","if","else","do","while","for","switch",  "case","int","double","float","long","void"  };  void init\_token();  void getbc();  void m\_getch();  int digit();  int letter();  int reserve();  int scanner();  int main()  {  int temp;  char input[30];  cout << "请输入源文件名:";  for (;;) {  cin >> input;  if ((fin = freopen(input, "r", stdin)) != NULL) {  cout << "从 " << input << " 读取" << endl;  while ((scanf("%[^`]s", &datain)) != EOF)  printf(datain);  break;  }  else  cout << "路径输入有误" << endl;  }  freopen(input, "r", stdin);  while (1) {  temp = scanner();  if (temp == -1) {  break;  }  switch (temp) {  case -10:  cout << "第 " << row << " 行 识别有误." << endl;  break;  default:  cout << "第 " << row << " 行 <" << temp << "," << token << ">" << endl;  break;  }  }  return 0;  }  void init\_token()  {  int i;  for (i = 0; i < 12; i++) {  token[i] = 0;  }  }  void getbc()  {  while (ch == ' ' || ch == '\t' || ch == '\n') {  if (ch == '\n') {  row++;//顺便记录行数  }  ch = getc(fin);  }  }  void m\_getch()  {  token[token\_num++] = ch;  ch = getc(fin);  }  int digit()  {  if (ch >= '0' && ch <= '9')  return 1;  else  return 0;  }  int letter()  {  if (ch >= 'a' && ch <= 'z' || ch >= 'A' && ch <= 'Z')  return 1;  else  return 0;  }  int reserve()  {  for (int i = 0; i < 13; i++) {  if (strcmp(token, keyWord[i]) == 0) {  //3为关键词  return 3;  }  }  //2为标识符  return 2;  }  int scanner()  {  init\_token();  if (flag == 0) {  ch = getc(fin);  }  flag = 1;  getbc();  token\_num = 0;  if (letter()) {  while (letter() || digit()) {  m\_getch();  }  token[token\_num++] = '\0';  reserve();  }  //是数字  else if (digit()) {  while (digit() || ch == '.') {  m\_getch();  }  return 1;  }  else {  token[token\_num++] = ch;  switch (ch) {  case '(':  ch = getc(fin);  return 16;  case ')':  ch = getc(fin);  return 17;  case '{':  ch = getc(fin);  return 33;  case '}':  ch = getc(fin);  return 34;  case '+':  ch = getc(fin);  if (ch == '+') {  token[token\_num++] = ch;  ch = getc(fin);  return 29;  }  else {  return 18;  }  case '-':  ch = getc(fin);  if (ch == '-') {  m\_getch();  return 30;  }  else {  return 19;  }  case '\*':  ch = getc(fin);  if (ch == '/') {  m\_getch();  return 32;  }  else {  return 20;  }  case '/':  ch = getc(fin);  if (ch == '\*') {  m\_getch();  return 31;  }else if (ch == '/') {  m\_getch();  return 39;  }  else {  return 21;  }  case '=':  ch = getc(fin);  if (ch == '=') {  m\_getch();  return 23;  }  else {  return 22;  }  case '>':  ch = getc(fin);  if (ch == '=') {  m\_getch();  return 25;  }  else {  return 24;  }  case '<':  ch = getc(fin);  if (ch == '=') {  m\_getch();  return 27;  }  else {  return 26;  }  case ';':  ch = getc(fin);  return 28;  case '"':  ch = getc(fin);  return 29;  case '!':  ch = getc(fin);  if (ch == '=') {  m\_getch();  return 37;  }  else {  return 36;  }  case '#':  ch = getc(fin);  if (ch == '#') {  m\_getch();  return -2;  }  else {  return 38;  }  case ',':  ch = getc(fin);  return 35;  case EOF:  return -1;  default:  ch = getc(fin);  return -10;  }  }  }  ```  测试文本文件data.txt  int main()  {  int b=10,e;  float c=0.01,f=110;  f+=b;  int i=0;  while(i==10){  f++;  i++;  }  e=f+c;  } |
| **实验结果：** |
| **实验分析：**  整个实验其实就是一个wordcount的过程，只不过不是对单词拼写种类进行统计，而是先人为设定单词对应的种别码，再进行扫描进行分类，扫描的过程是对源程序代码部分从左到右进行的扫描，识别的最小语法单位是符号或者单词，关键字、运算符、界符等等，使用状态转换图明确了编程的方向，图中每一个状态对应一段代码，遇到分支和重复调用的状态，使用条件判断和循环结构即可。  **出错分析：**  对于超前搜索，考虑了>、<、/、\*和类关键字的标识符几种情况，后面是否还有其他符号比如=、/、\*或其它字符，对于符号，使用条件分支即可，其中，对于C语言中的注释符//，读取下一位直到'\n'，对于'/\*'，则匹配下一位是'\*'，并且再下一位是'/'。 |
| **实验成绩：**  日期： 年 月 日 |

|  |
| --- |
| **河南理工大学教学上机实验报告** |
| 上机时间 2021 年 月 日 |
| **实验题目：**  实验三-自上而下的语法分析其的设计 |
| **实验目的和要求：**  目的：  根据具体的文法设计语法分析程序，对输入的符号串进行匹配判断，选择最有  代表性的语法分析方法，如递归下降分析法、预测分析法，加深对分析方法的  理解。  要求:  最基本的要求，能对一个算术表达式(a-1130)/b+c 做自上而下的语法分析。文法  的产生规则如下：  <表达式> ::= [+|-]<项>{<加法运算符> <项>}  <项> ::= <因子>{<乘法运算符> <因子>}  <因子> ::= <标识符>|<无符号整数>| ‘(’<表达式>‘)’  <加法运算符> ::= +|-  <乘法运算符> ::= \*|/  要求根据所选择的方法，对输入的字符串进行匹配分析，并给出对应匹配过程的结果，  最终给出句子是否语法正确的结论。 |
| **实验过程：**  **重要函数：**  **void getVN(string\* Gra)**  我们希望找出每一个非终结符的FIRST集合和FOLLOW集合，首先需要构造函数找出非终结符，非终结符是否有重复出现  void getVN(string\* Gra) {  for (int i = 0; i < Gra\_count; i++) {  VN.push\_back(Gra[i][0]);  }  }  **void getVT(string\* Gra)**  同理，也需要找出终结符，这里出现了一个问题，如果产生式中给出的是一个个位以上的数字，首先可以是重复的，其次，由于VT和VN都存储在vector容器中，类型为char，  所以无法直接将连接完的字符数字直接push\_back到容器中，需要通过string的c\_str方法先转换再存储到新定义的字符指针变量中，可是无法push\_back地址，只能将指针加入vector中，这样只能加入数字的首字符；所以使用insert函数，将数字部分加到结果尾部。  void getVT(string\* Gra) {  string temp = "";  char\* p;  for (int i = 0; i < Gra\_count; i++) {  for (int j = 3; j < Gra[i].length(); j++) {  if (int(Gra[i][j]) >= 48 && int(Gra[i][j]) <= 57) {  if (Gra[i][j] != ' ') {  temp += Gra[i][j];  strcpy(p, temp.c\_str());  }  }  else {  if (judgeVT(Gra[i][j])) {  VT.push\_back(Gra[i][j]);  }  }  }  }  VT.insert(VT.end(), p, p + strlen(p));  }  **void getFirst()**  1.对每个产生式进行扫描，如果能计算FIRST集则计算并标志已经计算  2.对于没有计算FIRST的非终结字符，如果能计算FIRST集则计算并标志已经计算。  3.重复执行2，直至所有的终结符的FIRST集计算完成。  实现过程：   1. 按行遍历所有产生式，如果开始就碰到终结符，直接insert；否则从左到右遍历右边的产生式，遇到终结符就break； 2. 如果不是终结符，就递归求该非终结符的First集   getFirst(Grammar[i][j]);  set<char>::iterator it;  int ind = getVNIndex(Grammar[i][j]);  for (it = First[ind].begin(); it != First[ind].end(); it++) {  if (\*it == '$')  flag = 1;  else  First[getVNIndex(x)].insert(\*it);  }  **void getFollow()**  1.对于文法开始符号E，置#于FOLLOW（E）中;  2.若A→aBb是一个产生式，则把FIRST(b)\{e}加至FOLLOW(B)中；  3.若A→aB是一个产生式，或A→aBβ是一个产生式,而b \*推导出e (即e∈FIRST(b))，则把FOLLOW(A)加至FOLLOW(B)中。  实现过程：  1.通过遍历所有文法产生式（所有行），找到非终结符x出现的位置  for (int i = 0; i < Gra\_count; i++) {  int index = -1;  for (int j = 3; j < Grammar[i].length(); j++) {  if (Grammar[i][j] == x) {  index = j;  break;  }  }  2.如果找到了x并且不是最后一个字符，对于下一个字符next：Grammar[i][index + 1]，判断其是否为终结符，如果是终结符，直接添加进x的follow集，否则遍历下一个字符的first集：如果有空串，标记1，否则加入FOLLOW集，判断下一个。  for (set<char>::iterator it = First[getVNIndex(next)].begin(); it != First[getVNIndex(next)].end(); it++) {  if (\*it == '$')  flag = 1;  else  Follow[getVNIndex(x)].insert(\*it);  }  如果有空串且左边不为它本身（防止死循环），则头部的follow集是x的follow集  char temp = Grammar[i][0];  if (flag && temp != x) {  getFollow(temp);  for (set<char>::iterator it = Follow[getVNIndex(temp)].begin(); it != Follow[getVNIndex(temp)].end(); it++)  Follow[getVNIndex(x)].insert(\*it);  }  如果在产生是的尾部，则产生式左部的Follow集应该加入到x的Follow集中  if (index != -1 && index == Grammar[i].length() - 1 && x != Grammar[i][0]) {  char temp = Grammar[i][0];  getFollow(temp);  for (set<char>::iterator it = Follow[getVNIndex(temp)].begin(); it != Follow[getVNIndex(temp)].end(); it++) {  Follow[getVNIndex(x)].insert(\*it);  }  **创建预测分析表**  void getAnalysisList() {  for (int i = 0; i < Gra\_count; i++) {  char temp = Grammar[i][3];  //如果右边第一个符号为终结符  if (judgeVT(temp)) {  //不为空串，直接更新分析表  if (temp != '$') {  AnalysisList[getVNIndex(Grammar[i][0])][getVTIndex(temp)] = i;  }  //为空串，将follow集加入，更新分析表  else {  for (set<char>::iterator it = Follow[getVNIndex(Grammar[i][0])].begin(); it != Follow[getVNIndex(Grammar[i][0])].end(); it++) {  //if (\*it == '#') {  // //最后一个+1的索引为#号  // AnalysisList[getVNIndex(Grammar[i][0])][VT.size()] = i;  // continue;  //}  AnalysisList[getVNIndex(Grammar[i][0])][getVTIndex(\*it)] = i;  }  }  }  //右边第一个为非终结符  else {  for (set<char>::iterator it = First[getVNIndex(temp)].begin(); it != First[getVNIndex(temp)].end(); it++) {  AnalysisList[getVNIndex(Grammar[i][0])][getVTIndex(\*it)] = i;  }  //first集中有空串，把follow集加入  if (First[getVNIndex(temp)].count('$') != 0) {  for (set<char>::iterator it = Follow[getVNIndex(Grammar[i][0])].begin(); it != Follow[getVNIndex(Grammar[i][0])].end(); it++) {  AnalysisList[getVNIndex(Grammar[i][0])][getVTIndex(\*it)] = i;  }  }  }  }  }  **分析**  void Analysis(string s) {  vector<char> left; //剩余输入串  vector<char> stack;//符号栈  cout << std::left << setw(8) << "步骤" << std::left << setw(15) << "符号栈" << std::left << setw(15) << "剩余输入串" << std::left << setw(20) << "所用产生式" << endl;  //把整个串push进剩余符号left  for (int i = 0; i < s.length(); i++) {  left.push\_back(s[i]);  }  left.push\_back('#');  //把#和开始符push进符号栈  stack.push\_back('#');  stack.push\_back(Grammar[0][0]);  //如果剩余输入串长度不为0，就一直循环  int step = 0;  while (left.size() > 0) {  cout << std::left << setw(8) << step++;  //输出分析栈内容  string outputs = "";  for (int i = 0; i < stack.size(); i++) {  outputs += stack[i];  }  cout << std::left << setw(15) << outputs;  outputs = "";  //输出剩余输入串内容  for (int i = 0; i < left.size(); i++) {  outputs += left[i];  }  cout << std::left << setw(15) << outputs;  char f1 = stack[stack.size() - 1];  char f2 = left[0];  //如果可以匹配，并且都为#  if (f1 == f2 && f1 == '#') {  cout << std::left << setw(21) << "Accept!" << endl;  return;  }  //如果可以匹配，并且都为终结符  if (f1 == f2) {  stack.pop\_back();  left.erase(left.begin());  cout << std::left << setw(15) << f1 << " 匹配" << endl;  }  else if (AnalysisList[getVNIndex(f1)][getVTIndex(f2)] != -1 && f1 != '#') {  //如果在预测分析表中有值  int index = AnalysisList[getVNIndex(f1)][getVTIndex(f2)];  stack.pop\_back();  if (Grammar[index][3] != '$') {  for (int i = Grammar[index].length() - 1; i >= 3; i--) {  stack.push\_back(Grammar[index][i]);  }  }  cout << std::left << setw(15) << Grammar[index] << endl;  }  else {  cout << std::left << setw(15) << "error" << endl;  return;  }  }  } |
| **实验结果：** |
| **实验分析：**  1. LL(1)文法的定义  每当在符号栈的栈顶出现非终极符时，要预测用哪个产生式的右部去替换该非终极符；当出现终结符时，判断其与剩余输入串的第一个字符是否匹配，如果匹配，则继续分析，否则报错。LL(1)分析方法要求文法满足如下条件：对于任一非终极符A的两个不同产生式->α,A->β，都要满足下面条件：SELECT(A->α)∩SELECT(A->β)=∅  2. 预测分析表构造  LL(1)分析表的作用是对当前非终极符和输入符号确定应该选择用哪个产生式进行推导。它的行对应文法的非终极符，列对应终极符，表中的值有两种：一是产生式的右部的字符串，一是null。  3. 语法分析程序的构造  LL(1)分析中X为符号栈栈顶元素，a为输入流当前字符，E为给定测试数据的开始符号，#为句子括号即输入串的括号。分析表用一个二位数组M表示，数组元素M[A，a]中的下标A表示非终结符，a为终结符或句子括号‘#’，二维数组中存放的是一条关于A 的产生式，表明当非终结符A向下推导时，面临输入符a时，所采用的候选产生式，当元素内容无产生式时，则表明用A 的左部向下推导时出现了不该出现的符号，因此元素内容转向出错处理的信息。 |
| **实验成绩：**  日期： 年 月 日 |