



编译原理实验报告

|  |  |
| --- | --- |
| 专 业 班 级 | 计算机科学与技术21-1班 |
| 学生姓名及学号 | 2021214710 杨程锦 |
| 任 课 教 师 | 李宏芒老师 |
| 实验指导教师 | 李宏芒老师 |
| 实验地点 | 科教楼C304 |
| 2022 ~2023 学年第二学期 | |

## 实验一 词法分析设计

1. **实验目的**

通过本实验的编程实践，使学生了解词法分析的任务，掌握词法分析程序设计的原理和构造方法，使学生对编译的基本概念、原理和方法有完整的和清楚的理解，并能正确地、熟练地运用。

1. **实验内容**

用 VC++/VB/JAVA 语言实现对 C 语言子集的源程序进行词法分析。通过输入源程序从左到右对字符串进行扫描和分解，依次输出各个单词的内部编码及单词符号自身值；若遇到错误则显示“Error”，然后跳过错误部分继续显示 ；同时进行标识符登记符号表的管理。

以下是实现词法分析设计的主要工作：

1. 可手动输入多行字符并读入进行分析
2. 统计行数和列数用于错误单词的定位
3. 删除空格类字符，包括回车、制表符空格
4. 按拼写单词（关键字、标识符、常数、运算符、关系运算符、分界符号），并用（内码，属性）二元式表示
5. 如果发现错误则报告出错
6. 根据需要是否填写标识符表供以后各阶段使用
7. 识别注释
8. 判别浮点数和整数

**三、实验开发环境**

编程语言：python

集成开发环境：PyCharm

**四、程序结构描述**

|  |  |
| --- | --- |
| **判断条件** | **作用** |
| str[i].isdigit() | 字符为数字 |
| str[i].isalpha() | 字符为字母 |
| str[i] in s | 字符为算术运算符 |
| str[i] in t | 字符为关系运算符 |
| str[i] in r | 字符为分界符 |
| str[i] in r | 字符为空格 |
| str[i] == '\n' | 字符为换行符 |
| else | 报错 |

1. **详细的算法描述**

程序通过遍历该词法中的每一个字符来判断该字符是否为特定的关键字。For循环来挨个遍历每一个字符，在循环内部通过一系列的判断来检测他们是否为关键字。

1. **给出软件的测试方法和测试结果**

输入词法：



输出结果：

文本

描述已自动生成

1. **实验总结**

此次的实验，我通过亲自动手敲代码实现了词法分析的过程，让我对这方面的知识有了更加深刻的理解。首先我根据编译原理指导书对整个实验进行构思，在有一个大概的框架之后，便开始着手写代码了。每写完一部分后我就会测试一遍，遇到问题之后就会想办法去解决，通过看书，重新整理思路等等方式去解决，使得我对整个词法分析的过程越来越熟悉了，同时也对我解决问题，思考问题的能力有了提升。由于大二下学期没有安排专门的编程课，导致我的编程能力有所下降，但是经过这次的训练，对我编程水平的提升也有着巨大的帮助。

**附录：程序核心代码**

|  |
| --- |
| for line in open("test.txt", 'r', encoding="UTF-8"):  str = line  i = 0  while i < len(str):  if jud1 != 1:  if str[i].isdigit(): # 字符为数字  jud = 1 # 用于判断该单词是不是纯数字  str1 = '' # 用于暂时存储单词  while i < len(str):  if str[i].isdigit():  str1 += str[i]  i += 1  elif str[i].isalpha():  str1 += str[i]  i += 1  jud = 0  else: break  if jud == 1: # 全为数字，构建输出  if str1 in u: u.append(str1)  mes = [str1, f'(1,{str1})', '常数', f'({hang}, {lie})']  else: # 夹杂有字母  mes = [str1, 'Error', 'Error', f'({hang}, {lie})']  lie += 1  message = tep.join(mes)  output.append(message)  elif str[i].isalpha(): # 字符为字母  str1 = ""  while i < len(str):  if str[i].isdigit() or str[i].isalpha():  str1 += str[i]  i += 1  else: break  if str1 in k:  mes = [str1, f'(1,{str1})', '关键字', f'({hang}, {lie})']  elif not str1 in v:  v.append(str1)  mes = [str1, f'(6,{str1})', '标识符', f'({hang}, {lie})']  else:  mes = [str1, f'(6,{str1})', '标识符', f'({hang}, {lie})']  lie += 1  message = tep.join(mes)  output.append(message)  elif (str[i] in s): # 字符为算术运算符  str1 = ""  if str[i] == '/' and str[i+1] == '/':  i = len(str)  elif str[i] == '/' and str[i+1] == '\*':  jud1 = 1  else:  while i < len(str):  if str[i] in s:  str1 += str[i]  i += 1  else: break  if str1 in s:  mes = [str1, f'(1,{str1})', '算术运算符', f'({hang}, {lie})']  else:  mes = [str1, 'Error', 'Error', f'({hang}, {lie})']  lie += 1  message = tep.join(mes)  output.append(message)  elif (str[i] in t): # 字符为关系运算符  str1 = ""  while i < len(str):  if str[i] in t:  str1 += str[i]  i += 1  else: break  if str1 in t: mes = [str1, f'(1,{str1})', '关系运算符', f'({hang}, {lie})']  else: mes = [str1, 'Error', 'Error', f'({hang}, {lie})']  lie += 1  message = tep.join(mes)  output.append(message)  elif (str[i] in r): # 字符为分界符  str1 = str[i]  i += 1  mes = [str1, f'(1,{str1})', '分界符', f'({hang}, {lie})']  lie += 1  message = tep.join(mes)  output.append(message)  elif (str[i] == ' '): i += 1 #字符为空格  elif (str[i] == '\n'):  hang += 1  i += 1  continue  else:  i += 1  mes = [str1, 'Error', 'Error', f'({hang}, {lie})']  lie += 1  message = tep.join(mes)  output.append(message)  else:  while str[i] != '\*' or str[i+1] != '/':  if (i < len(str)-2):  i += 1  else: break  if str[i] == '\*' or str[i+1] == '/':  jud1 = 0  hang -= 1  i += 2  else: break |

## 实验二 LL(1)分析法

**一、实验目的**

通过完成预测分析法的语法分析程序，了解预测分析法和递归子程序法的区

别和联系。使学生了解语法分析的功能，掌握语法分析程序设计的原理和构造方

法，训练学生掌握开发应用程序的基本方法。有利于提高学生的专业素质，为培

养适应社会多方面需要的能力。

**二、实验内容**

1、根据某一文法编制调试 LL （ 1 ）分析程序，以便对任意输入的符号串

进行分析。

2、构造预测分析表，并利用分析表和一个栈来实现对上述程序设计语言的分

析程序。

3、分析法的功能是利用 LL（1）控制程序根据显示栈栈顶内容、向前看符号

以及 LL（1）分析表，对输入符号串自上而下的分析过程。

**三、实验开发环境**

编程语言：C++

集成开发环境：vs完成核心代码的编写，QT完成图形界面编写

**四、程序结构描述**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| LL1 | 属性 | set | first\_set | 文法符号的FIRST集 |
| set | follow\_set | 文法符号的FOLLOW集 |
| sest | grammarMap | 产生式左边和右边映射 |
| set | VT | 终结符集合 |
| set | VN | 非终结符结合 |
| set | table | 预测分析表 |
| 方法 | void | getGrammarMap() | 构建产生式左边和右边映射 |
| void | getVT() | 构建终结符集合 |
| void | getVN() | 构建非终结符集合 |
| Void | getFirstSet() | 构建FIRST集 |
| void | getStrFirst() | 构建字符串FIRST集 |
| void | getFollow() | 构建FOLLOW集 |
| void | getPredictiveAnalysisTable() | 构建预测分析表 |

**五、详细的算法描述**

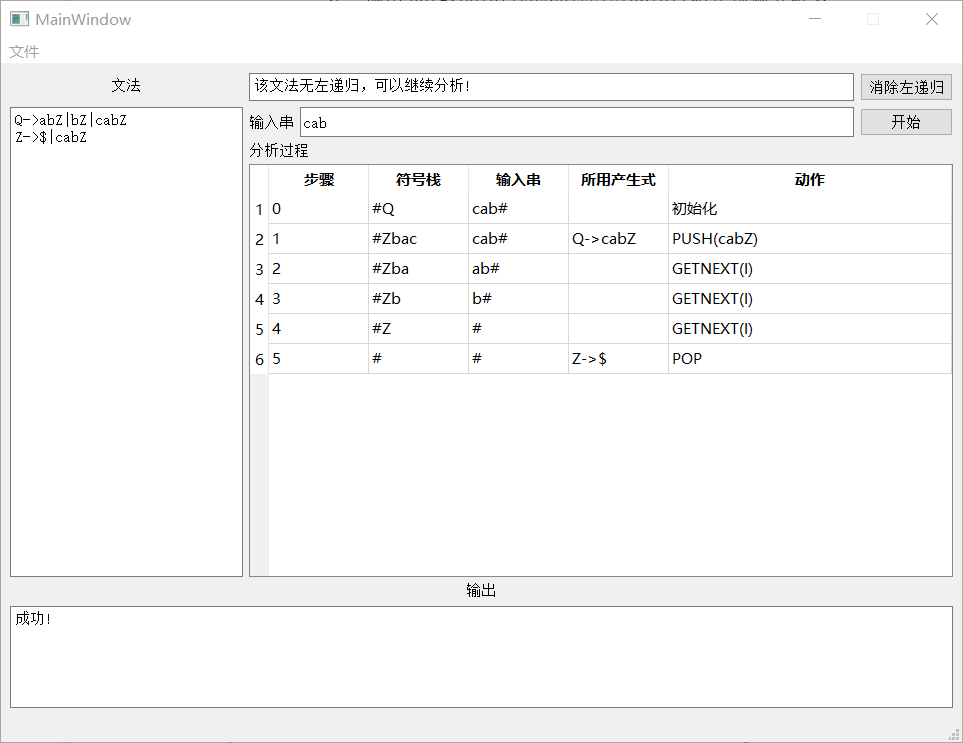
图示

描述已自动生成图示

描述已自动生成

* + - 1. 首先从文件中读取文法信息
      2. 调用LL1类中的方法getGrammarMap()将非终结符和其产生式右部构建映射关系。
      3. 调用getVT()和getVN()获得终结符和非终结符的集合
      4. 调用getFirstSet()构建文法符号的FIRST集
      5. 调用getFollow()构建非终结符的FOLLOW集
      6. 调用getPredictiveAnalysisTable()构建预测分析表
      7. 按照上面LL1预测分析过程进行分析

**六、给出软件的测试方法和测试结果**



图一、LL1预测分析过程

文本, 信件

描述已自动生成

图二、程序生成地FIRST集

图表

描述已自动生成

图三、程序生成地FOLLOW集合

散点图

描述已自动生成

图四、程序生成地预测分析表

**七、实验总结**

通过此次的实验，我收获很多，学习到了很多。

1.我对LL1文法更加的熟悉，知识点理解更加的深刻。通过自己手敲代码以及通过“遇到问题-解决问题”这一过程，我对整个LL1的逻辑和过程更加清楚了。

2.本次的实验代码量巨大，代码写了400行左右，在写代码的过程中，通常需要对某些变量设计复杂的数据结构，以及使用这些数据结构等，都让我的代码能力得到提升，此外，我还使用之前不曾使用的C++的新特性。

3.因为我在写代码的过程中遇到了很多的问题，让我对代码的调试能力得到了提升，同时也对我的思考能力，解决问题的能力等有着很大提升。

4.消除左递归算法的实现对于我来说还是比较困难，是先后我感觉自己的认识又加深了几分。

**附录：程序核心代码**

|  |
| --- |
| // 预测分析过程  stack<std::string> stackLL;      string stackStr{};      stackLL.push("#");      stackStr += '#';      auto pos = v\_LL[0].find("->");      std::string begin = v\_LL[0].substr(0, pos);      stackLL.push(begin);      stackStr += begin;      // print to test      std::cout << "------- stack init ------" << std::endl;      std::cout << stackLL.top() << std::endl;      std::cout << inputStr << std::endl;      // 预测分析      cout << "步骤\t\t" << "符号栈\t\t" << "输入串\t\t" << "所用产生式" << endl;      int stepNum = 0;      int index = 0; // 用来存取输入      cout << to\_string(stepNum) << "\t\t" << stackStr << "\t\t" << inputStr << endl;      bool flag = true;      string a = inputStr.substr(index, 1);      while (flag)      {          string stack\_top = stackLL.top();          stackLL.pop();          stackStr.pop\_back();          auto pair\_tmp = pair<string, string>(stack\_top, a);          if (ll1.isInVT(stack\_top)) {              if (stack\_top == a) {                  a = inputStr.substr(++index, 1);              }              else{                  cout << "ERROR" << endl;                  break;              }          }          else if (stack\_top == "#") {              if (stack\_top == a) {                  flag = false;              }              else {                  cout << "ERROR" << endl;                  break;              }          }          else if (!ll1.table[pair\_tmp].empty()) {              string right{};              string s\_tmp = ll1.table[pair\_tmp];              auto pos1 = s\_tmp.find("->");              right = s\_tmp.substr(pos1+2);              vector<string> v\_tmp{};              LL1::splitString(v\_tmp, right);              for (int i=int(v\_tmp.size()-1); i>=0; i--) {                  if (v\_tmp[i] == "$")                      continue;                  stackLL.push(v\_tmp[i]);                  stackStr += v\_tmp[i];              }          }          else {              cout << "ERROR" << endl;              break;          }      } |

## 实验三 LR(1)分析法

1. **实验目的**

构造 LR(1)分析程序，利用它进行语法分析，判断给出的符号串是否为该文法识别的句子，了解 LR（K）分析方法是严格的从左向右扫描，和自底向上的语法分析方法。

**二、实验内容**

对下列文法，用 LR（1）分析法对任意输入的符号串进行分析：

（1）E-> E+T

（2）E->T

（3）T-> T\*F

（4）T->F

（5）F-> (E)

1. F-> i

**三、实验开发环境**

开发工具：java

编程语言：IntelliJ IDEA

**四、程序结构描述：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类** | **成员类别** | **类型** | **成员名** | **描述** |
| LR1 | 属性 | map | firstSet | 文法符号的FIRST集 |
| map | follow\_set | 文法符号的FOLLOW集 |
| map | grammarMap | 产生式左边和右边映射 |
| set | VT | 终结符集合 |
| vector | VN | 非终结符结合 |
| string | begin | 文法开始符号 |
| map | CLOSURE | 项目闭包 |
| map | GO | DFA |
| map | setId | 项目id |
| set | C | 项目集族 |
| set | charSet | 所有符号的集合 |
| map | ACTION | ACTION表 |
| map | GOTO | GOTO表 |
| 方法 | void | getGrammarMap() | 构建产生式左边和右边映射 |
| void | getVT() | 构建终结符集合 |
| void | getVN() | 构建非终结符集合 |
| void | getFirstSet() | 构建FIRST集 |
| void | getStrFirst() | 构建字符串FIRST集 |
| void | getFollow() | 构建FOLLOW集 |
| void | getProgram() | 构建项目集合 |
| void | getCharSet() | 获得所有符号 |
| void | getTable() | 构建ACTION和GOTO表 |
| Program | 属性 | string | prog | 项目当前状态(产生式加点) |
| vector | leftStr | 产生式左边部分 |
| vector | rightStr | 产生式右边部分 |
| string | LFStr | 展望串 |
| Unsinged long | pos | 状态点的位置 |
| 方法 | void | addDot() | 在产生式中加点表示状态 |
| void | splitString | 对字符串进行切分 |

**程序实现步骤：**

* + - 1. 从文件中读取文法信息，并进行拓广
      2. 调用getGrammarMap()函数对产生式进行拆分，构建非终结符与产生式右部的映射关系
      3. 调用getVT()和getVN()获得VT和VN
      4. 调用getFirstSet()构建FIRST集合
      5. 调用getCharSet()获得所有文法符号
      6. 调用getProgram()构造项目闭包和项目集族
      7. 调用getTable()构建ACTION和GOTO表

**五、算法及设计思想**

**1.项目集族的构造**

构造LR(1)项目集族的办法本质与构造LR(0)项目集族的办法是一样的，首先需要两个函数CLOSURE函数和GO函数。

CLOSURE函数的构造方法如下：

1. I的任何项目都属于ClOSURE(I)
2. 若项目[A->α\* Bβ,a]属于CLOSURE(I),B->ξ是一个产生式，那么，对于FIRST(βa)中的每一个终结符b,如果[B->ξ,b]原来不在CLOSURE(I)中，则把它加进去
3. 重复步骤（2），直至CLOSURE(I)不再增大为止。

构造的过程都比较容易理解，比较麻烦的是怎么实现CLOSURE(I)不再增大，在这里，用了一点小技巧，设置i从前往后遍历CLOSURE(I)每一个元素用来实现CLOSURE(I)不断增大，设置j指向CLOSURE(I)的最后一个元素，如果i和j不相等，说明CLOSURE(I)还在不断增大，以此来实现CLOSURE(I)的构造。

GO函数的构造如下：

GO(I,X) = CLOSURE(J)

其中 J = ｛任何形如[A->αX\*β，a]的项目|[A->α\*Xβ,a]∈I｝

然后构造项目集族C,构造算法是：

BEGIN

C:=｛CLOSURE({[S’->\*S,#]})｝;

REPEAT

FOR C中的每个项目集I和G’的每个符号X DO

IF GO(I,X)非空且不属于C,THEN把GO(I,X)加入C中

UNTIL C不再增大

END

再通过LR(1)项目集族C构造分析表的算法

1. 若项目[A->α\*aβ，b]属于Ik且GO(Ik,a)=Ij,a为终结符，则置ACTION[k,a]为“把状态j和符号a移进栈”，简记为“sj”
2. 若项目[A->α\*，a]属于Ik,则置ACTION[k,a]为“用产生式A->α归约”，简记为“rj”;其中假定A->α为文法G’的第j个产生式。
3. 若项目[S’->\*S,#]属于Ik,则置ACTION[k,#]为“接受”，简记为“acc”
4. 若GO(Ik,A)=Ij，则置GOTO[k,A]=j
5. 分析表中凡不能用规则1至4填入信息的空白栏均填上“出错标志”

**2.LR分析器构造：**

(1)移进：

table[i，a]= Sj：状态 j 移入到状态栈，把a移入到文法符号栈，其中 i,j 表示状态号。

(2)归约：

table[i，a]=rk：当在栈顶形成句柄时，则归约为相应的非终结符 A，即文法中有 A- B 的产生式，若 B 的长度为 R(即|B|=R)，则从状态栈和文法符号栈中自顶向下去掉 R 个符号，即栈指针 SP 减去 R，并把 A 移入文法符号栈内，j=GOTO[i,A]移进状态栈，其中 i 为修改指针后的栈顶状态。

(3)接受 acc:

当归约到文法符号栈中只剩文法的开始符号 S 时，并且输入符号串已结束即当前输入符是'#'，则为分析成功,当table中该项为acc时分析成功接受。

(4)报错:

当遇到状态栈顶为某一状态下出现不该遇到的文法符号时，则报错，说明输

入端不是该文法能接受的符号串。

**六、给出软件的测试方法和测试结果。**

图示

描述已自动生成

图一、总控程序流程图

**图形用户界面

描述已自动生成**

图二、LR1分析过程

**文本

中度可信度描述已自动生成**

图三、程序自动生成的FIRST集

表格

中度可信度描述已自动生成

图四、状态0的项目闭包(其他闭包过多，不展示)

图片包含 图表

描述已自动生成

图五、ACTION表何GOTO表

**七、实验总结**

通过此次的实验，我有很多的收获，也有很多的反省。我对LR1文法更加的熟悉，知识点理解更加的深刻。通过自己手敲代码以及通过“遇到问题-解决问题”这一过程，我对整个LR1的逻辑和过程更加清楚了。因为我在写代码的过程中遇到了很多的问题，让我对代码的调试能力得到了提升，同时也对我的思考能力，解决问题的能力等有着很大提升。这次的实验总体来说完成的有些仓促，怪我平时没有安排好时间，导致出现事情堆在一起手忙脚乱的情况，这个需要我去反省。

**附录：核心代码**

|  |
| --- |
| stack<int> stack\_state{} = new stack<>();      stack\_state.push(0);      stack<string> stack\_char{} = new stack<>();      stack\_char.push("#");      int index = 0;      int stepNum = 0;      while (index < v\_in.size()) {          int state\_top = stack\_state.top();          string a\_str = v\_in[index];          pair<int, string> pair\_tmp = pair<int, string>(state\_top, a\_str);          string actionStr = lr1.ACTION[pair\_tmp];          if (actionStr.empty()) {                break;          }          else if (actionStr == "acc") {              break;          }          else if (actionStr[0] == 's') {              stack\_state.push(stoi(actionStr.substr(1)));              stack\_char.push(a\_str);              index++;              stepNum++;          }          else if (actionStr[0] == 'r') {              string production = lr1.id\_pro[stoi(actionStr.substr(1))];              auto pos = production.find("->");              string left = production.substr(0, pos);              string right = production.substr(pos+2);              auto length = right.size();              while (length > 0) {                  stack\_state.pop();                  stack\_char.pop();                  length--;              }              stack\_char.push(left);              int top\_tmp = stack\_state.top();              pair<int, string> p1 = pair<int, string>(top\_tmp, left);              stack\_state.push(lr1.GOTO[p1]);              stepNum++;          }      } |