|  |
| --- |
| 编译原理课程设计  实验报告  **指导教师：刘华虓**  **年 级： 2019**  **班 级：211927**  **小组编号： 9**  **组长学号姓名：朱雨廷**  **组员学号姓名：许馨月**  **组员学号姓名：**  **2022年 4 月 27 日**  **计算机科学与技术学院** |

|  |  |  |
| --- | --- | --- |
| **完成实验内容** | | |
|  | | |
| **小组成员任务完成情况** | | |
| 姓名 | 具体完成任务 | 工作量  百分比 |
| 朱雨廷 |  |  |
| 许馨月 |  |  |
|  |  |  |
| **小组成员协作情况** | | |
|  | | |
| **实验平台与编程语言** | | |
| **实验平台：vs2019 Qt5.9**  **编程语言：C++** | | |
| **实验方案设计** | | |
| 求解predict集合   1. 思想：   由于SNL语法中的终极符和非终极符是一个单词而非字母，同时一个非终极符会对应多个文法，先将文法分解，拆成**map<string, set<vector<string>>>**的形式，然后根据算法依次求解每个文法符号的first集合从而求解符号串的first，然后求每个文法符号的follow集合，最后结合follow集和first集得出每个产生式的predict集。   1. 用到的数据结构：   **map<string, set<vector<string>>> grammars;//文法**  文法分解 string为左侧非终结符 set为该符号产生的一条/多条符号串 vector存着一条符号串里的每个单词  **map<string, set<string>> first;//符号的first集**  string为符号 set则存着first集合  **map<string, set<string>> follow;//符号的follow集**  string为符号 set则存着follow集合  **map<int, set<string>> first1;// 符号串的first集**  int为序号(1-104) set则存着first集合  **map<int, set<string>> predict; //产生式的predict集**  int为序号(1-104) set则存着predict集合  **typedef struct node {**  **int num;//产生式号**  **string vn;//左边的非终极符**  **vector<string> production;//单词组合**  **set<string> firstset;**  **set<string> predictset;**  **} gramm;**  一条产生式可分成一个gramm   1. 流程图： 2. 求predict主流程图  1. 求文法字符的first集合 first(X)  1. 求产生式的first集合 first(beta)  1. 求所有非终极符的follow集合 follow(A)  1. 求产生式的predict集 predict(A->beta)   LL1语法分析   1. 思想：显式采用栈，分为两个部分，一个是语法分析表(记录规则)，一个是语法分析驱动程序(根据规则采取动作) 涉及四个栈：符号栈、语法树栈、操作符栈、操作数栈。   语法分析驱动程序分为以下四步：  a.分析栈的栈顶元素是终极符，则看其是否与输入流的头符相匹配，如果成功，去掉栈顶元素．并读下一个单词；若匹配不成功，则报错。  b.栈顶是非终极符，则用栈顶和输入流的当前单词去查当前矩阵，如果查的值是产生式编号，则把对应的产生式右部逆序压入栈中；如果查得的值为.错误信息，则报错。  c.栈已空，输入流不空，这时输入流报错。  d.若栈已空，输入流也空，则语法分析成功。   1. 数据结构：   **int LL1Table[MaxSymbolValue+5][MaxSymbolValue+5]**//符号表  **static vector<string> errors** // 保存所有的语法错误信息  **static vector<Symbol> symbolStack** // 符号栈 ,存放 语法符号 Vt 和 Vn  **static vector<TreeNode \*\*> syntaxStack**  // 语法树栈 , 存放指向儿子或兄弟节点的指针的地址 处理声明部分和语句部分  **static vector<TreeNode \*> operatorStack** // 操作符栈  **static vector<TreeNode \*> numStack**  // 操作数栈   1. 流程图： 2. 主流程图  1. LL1分析主体部分LL1parse() | | |
| **程序界面及运行截图** | | |
| LL1生成树：  成功时：  调试信息：  不成功，有错误：  调试信息： | | |
| **源程序核心代码** | | |
| 求predict集主函数  **void getAllPredict() {**  **get\_grammars();**  **getallFirst();**  **getFollow();**  **getFollow();**  **getFollow();**  **getFollow();**  **getFollow();**  **getFollow();**  **getFollow();**  **getFirst1();**  **getPredict();**  **outputPredict(predictfile);**  **if(isInitPredict) return ;**  **isInitPredict=true;**  **}**  LL1主函数：  **TreeNode\* getLL1Tree() {**  **// token 序列读进来**  **tokens = getTokenList();**  **initReadOneToken();**  **initProducts(); // 千万不要忘了 init !!!!!**  **getAllPredict(); //initPredict();**  **if(tokens.size()==0) return NULL; // 如果词法分析错误, 那么不进行语法分析**  **bool errorflag=LL1parse(); //LL1主要功能**  **const char fileName[] = "D:/CODE\_Qt/txt/ll1syntaxTree.txt";**  **if ( errorflag==1 ){**  **outputParserTree(rt , fileName);**  **FILE \* fn=fopen(errorFileName,"a"); // 控制台信息输出到 debug\_info 文件 , 除词法分析都是 a**  **fputs(">>> LL1语法分析正确 !\n",fn);**  **fclose(fn);**  **}else{**  **rt=NULL;**  **outputParserTree(rt , fileName);**  **outputParserLL1Errors(errorFileName);**  **}**  **return rt;**  **}** | | |