|  |
| --- |
| 编译原理课程设计  实验报告  （第 3 周）  **指导教师：申春**  **年 级： 2117**  **班 级：211711**  **小组编号： 211711-01**  **组长学号姓名：21171102许俊杰**  **组员学号姓名：21171135周智勇**  **组员学号姓名：21171110付昊翀**  **2020年 5 月 13 日**  **计算机科学与技术学院** |

|  |  |  |
| --- | --- | --- |
| **完成实验内容** | | |
| 在源词法分析部分代码基础上增加了大量注释说明，为解决接下来语法分析的多种变量声明问题，用大量文字直观描述词法分析函数  更改部分算法以解决不能正确匹配Begin-End的逻辑错误  更改部分变量常量名称防止在语法分析部分发生冲突  完善修改标识符展示表struct type项  说明词法分析中使用的库函数isdigit和isalpha  详细解释将各种word写入展示表的过程  说明之前易出错的问题（解决逻辑难题：不同type的table可以有相同的index）  //会有Index相同但type不同的table（详见词法分析函数的常数与标识符两部分）  设计语法错误的16种类  根据一句话的开头字符开始往后进行表达式分析（解决逻辑难题：运算符出线的位置会造成不同的错误分析结果，详见问题与解决方案）  //while（1）{  //分支1 如果是变量或整数  //分支1.1 如果在同一行，整数后直接加变量  //分支1.2 如果这个变量或整数与下一个字符不在同一行  //分支2  //分支2.1 如果是运算符  //分支2.2 如果不是运算符  //分支3 如果是数字或变量后加运算符  //分支4 如果数字或变量后加分界符  //分支5 如果table下标>table总数  //分支6 如果数字或变量后什么都没有  }  利用Github平台进行代码的分享使用，学习使用Github建立仓库和上传文件（目前正在学习中） | | |
| **小组成员任务完成情况** | | |
|  |  |  |
| 姓名 | 具体完成任务 | 工作量  百分比 |
|  | 三人共同学习查询程序中使用的多个库函数  cin.getline  isalpha库函数  isdigit库函数  三人共同查询Github的最基本使用方法，仍在学习中 |  |
| 许俊杰 | 初步设计主函数  考虑输入方式  设计多种语法错误类型  编写语法错误输出函数与展示格式  参与共同编写语法分析的表达式语法分析子函数  （解决逻辑难题：详见问题与解决方案）  在github建立team并在其中建立仓库 | 33.3% |
| 周智勇 | 参与共同编写语法分析的表达式语法分析子函数  （解决逻辑难题：详见问题与解决方案）  改进词法分析的bug  在词法分析中增加大量注释使设计语法分析与主函数时可以正确调用并直观发现问题  在github的team中将仓库设置权限，使组员可以pull request，将本地文件上传 | 33.3% |
| 付昊翀 | 参与共同编写语法分析的表达式语法分析子函数  （解决逻辑难题：详见问题与解决方案）  改进词法分析的bug  完善showtable结构体并在table各项中加入之后词法分析所涉及的各种标识符  在Github中上传词法分析部分可调试代码  解决showtable中的index冲突问题 | 33.3% |
| **小组成员协作情况** | | |
| **本程序中：**  **主函数使用了cin.getline( 第一个字符指针(char\*),字符个数(int),结束符'#'(char) );**  **词法分析函数使用了isalpha库函数**  **判断字符是否为英文字母**  **若大写字母返回1，小写字母返回2，非字母返回0**  **词法分析函数使用了isdigit库函数**  **判断是否为十进制数字**  **若阿拉伯数字0-9返回非零值，否则返回0**  三人共同解决了语法分析部分-表达式分析函数中的逻辑难题，尝试多次，并最终给出了相对成功地解决方式  三人互相分享各自的改进方案，并成功找到词法分析中的隐藏问题 | | |
| **实验中出现的问题及解决方案** | | |
| 主要解决**两个逻辑难题**和**一个定义声明中的隐患**  **定义声明中的隐患**  (以下代码出现在前端声明处）  **int BeginEndProgram = 0;**  **//正确情况下应该为0，遇到begin加1，遇到end减1，<0则缺少Begin，>0则缺少End**  之前没有想到会出现两个end或两个begin的情况（尽管这发生的可能性较小）  **难题1.两个table的index项会不会冲突？**  (以下代码出现在table结构体定义处）  **typedef struct ShowTable//某个标识符的展示表**  **{**  **int Index; //某个标识符展示表的下标**  **//注：会有Index相同但type不同的table（详见词法分析函数的常数与标识符两部分）**  **char CharacterInTableName[20]; //某标识符的名称**  **int type;**  **/\*该标识符的类型：1~8：......\*/**  **int line; //该标识符在第几行**  **}Table;**  (以下算法在词法分析函数 **LexicalAnalyzer(char character[], int charLength, int nLine)**中词法分析函数的常数与标识符两部分处）  不同类型的标识符看做不同的分支  在各自分支下对各自的table[i].index操作  因此会有table的type项不同但index项相同  举个例子：  整数5对应table值：  ***table.index=m；table.CharacterInTableName=5；table.type=3（type=3表示整数）***  变量a对应table值：  ***table.index=m；table.CharacterInTableName=a；table.type=2（type=3表示变量）***  **难题2.运算符出现在不同位置时怎样判断错误？**  (以下代码和算法分析均为语法分析部分-表达式分析子函数）  **//分支1 如果是变量或整数**  **//分支1.1 如果在同一行，整数后直接加变量**  **GrammaError(7, TableIndex); //则中间缺少运算符**  **//分支1.2 如果这个变量或整数与下一个字符不在同一行**  **出错信息：该语句缺少分号**  **//分支2**  **//分支2.1 如果是运算符**  **GrammaError(13, TableIndex);//则缺少变量**  **//分支2.2 如果不是运算符**  **GrammaError(6, TableIndex); //则该处应为变量或常量**  **//分支3 如果是数字或变量后加运算符**  **//分支4 如果数字或变量后加分界符**  **//分支5 如果table下标>table总数**  **//分支6 如果数字或变量后什么都没有**  **分析：分支2.1判断运算符（输出错误） 与 分支3判断运算符（成功，向bool类型函数return1） 并不冲突，原因在于：**  **当第一个字符读入时如果是运算符，才会调用分支2.1说明这句话以运算符开头，这肯定是错误的**  **当第一个字符读入时如果是整数或变量才会进入分支1，之后再用分支1.1或1.2判断是否有错，如果没错则会直接越过分支1.1和1.2，也会越过分支2，直接进入分支3判断是否为运算符，此时如果是运算符，那么一定是变量或常数后才接运算符而不是以运算符开头的，所以分支2.1和分支3虽然判断语句相同，但不可能同时出现**  **还有一种特殊情况是开头为运算符，后面又接了一个运算符，这种判断分别在两次循环中分别执行，所以不会运行时出现错误，而这种情况在第一个运算符输入时就已经判断处错误了，所以只需输出错误类型即可，不必再管有几个运算符，这句话一定是错的**  这个算法思路并不是非常完美，但是是经过多次改正之后认为相对来讲比较妥当的，如果后续有新的思路解决变量与运算符之间问题的想法会随时尝试调整  具体算法实现详见程序代码 | | |
| **程序界面及运行截图** | | |
| 本次完成大致分四部分  1.修改的定义与声明部分  2.修改的showtable表项  3.语法错误-输出函数  4.语法错误-表达式分析函数  修改的定义与声明部分  修改的showtable表项  语法错误-输出函数  语法错误-表达式分析函数 | | |
| **已完成部分的源程序代码** | | |
| **/\*本程序中：**  **主函数使用了cin.getline( 第一个字符指针(char\*),字符个数(int),结束符'#'(char) );**  **词法分析函数使用了isalpha库函数**  **判断字符是否为英文字母**  **若大写字母返回1，小写字母返回2，非字母返回0**  **词法分析函数使用了isdigit库函数**  **判断是否为十进制数字**  **若阿拉伯数字0-9返回非零值，否则返回0**  **\*/**  **#define CodeMaxLength 500 //最大代码长度**  **#define VarMaxNumber 100 //变量最大个数**  **#define ConstMaxNumber 100 //常量最大个数**  **#define KeywordMaxNumber 33 //关键字数量**  **#define OpSingleMaxNumber 8 //单目运算符最大个数**  **#define OpDoubleMaxNumber 4 //双目运算符最大个数**  **#define EndMaxNumber 11 //界符最大个数**  **typedef struct ShowTable//某个标识符的展示表**  **{**  **int Index; //某个标识符展示表的下标**  **//注：会有Index相同但type不同的table（详见词法分析函数的常数与标识符两部分）**  **char CharacterInTableName[20]; //某标识符的名称**  **int type;**  **/\***  **该标识符的类型：**  **1 关键字**  **2 变量标识符**  **3 整数**  **4 小数**  **5 单目运算符**  **6 双目运算符**  **7 分界符**  **8 无法识别的其他字符**  **\*/**  **int line; //该标识符在第几行**  **}Table;**  **int TableNum = 0; //展示表总数**  **char Word[VarMaxNumber][20]; //变量表，用于存放变量名**  **char Digit[VarMaxNumber][20]; //数字表，用于存放数字**  **int VarNum = 0; //变量表的下标**  **int ConstNum = 0; //常量表的下标**  **bool errorFlag = 0; //错误标志**  **int TableIndex = -1; //Table表的下标索引**  **int BeginEndProgram = 0;**  **//正确情况下应该为0，遇到begin加1，遇到end减1，<0则缺少Begin，>0则缺少End**  **int ifCount = 0; //遇到if加1**  **Table\* table = new Table[CodeMaxLength];**  **/\***  **词法分析**  **输入的字符（串）可能为**  **空格**  **tab**  **换行符**  **标识符（关键字或非关键字）**  **常数**  **运算符**  **\*/**  **void LexicalAnalyzer(char character[], int charLength, int nLine)**  **//根据一句话的开头字符开始往后进行表达式分析**  **bool Expression()**  **{**  **/\***  **table.type的值（int）=**  **1 关键字**  **2 变量标识符**  **3 整数**  **4 小数**  **5 单目运算符**  **6 双目运算符**  **7 分界符**  **8 无法识别的其他字符**  **\*/**  **while (1)**  **{**  **//分支1 如果是变量或整数**  **if (table[TableIndex].type == 2 || table[TableIndex].type == 3)**  **{**  **//分支1.1 如果在同一行，整数后直接加变量**  **if (table[TableIndex].type == 3 && table[TableIndex + 1].type == 2 && table[TableIndex].line == table[TableIndex + 1].line)**  **{**  **GrammaError(7, TableIndex); //则中间缺少运算符**  **//TableIndex = TableIndex + 2;这是缺少双目运算符的情况**  **TableIndex++;**  **}**    **//分支1.2 如果这个变量或整数与下一个字符不在同一行**  **if (table[TableIndex].line != table[TableIndex + 1].line)**  **{**  **// Gerror(14,TableIndex); //出错信息：该语句缺少分号**  **return 1;**  **}**  **TableIndex++;**  **}**    **//分支2**  **else**  **{**  **//分支2.1 如果是运算符**  **if (table[TableIndex].type == 5 || table[TableIndex].type == 6)**  **{**  **GrammaError(13, TableIndex);//则缺少变量**  **TableIndex++;**  **}**  **//分支2.2 如果不是运算符**  **else**  **{**  **GrammaError(6, TableIndex); //则该处应为变量或常量**  **}**  **TableIndex++;**  **}**  **//分支3 如果是数字或变量后加运算符**  **if (table[TableIndex].type == 5 || table[TableIndex].type == 6)**  **TableIndex++;**  **//分支4 如果数字或变量后加分界符**  **else if (table[TableIndex].type == 7)**  **return 1;**  **//分支5 如果table下标>table总数**  **else if (TableIndex >= TableNum) //|| )**  **{**  **GrammaError(14, TableIndex); //出错信息：该语句缺少分号**  **return 1;**  **}**  **//分支6 如果数字或变量后什么都没有**  **else**  **{**  **GrammaError(5, TableIndex); //出错信息：此处应为运算符**  **TableIndex++;**  **}**  **}**  **}**  **void GrammaError(int errorType, int nIndex)**  **{**  **errorFlag = 1;**  **switch (errorType)**  **{**  **case 1:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex].CharacterInTableName << " 应该为赋值号：= \n";**  **break;**  **case 2:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex].CharacterInTableName << " 应为变量 \n";**  **break;**  **case 3:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex].CharacterInTableName << " 应为逗号 \n";**  **break;**  **case 4:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex].CharacterInTableName << " 应为分号 \n";**  **break;**  **case 5:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex].CharacterInTableName << " 应为运算符 \n";**  **break;**  **case 6:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex].CharacterInTableName << " 应为变量或常量 \n";**  **break;**  **case 7:**  **cout << "第" << table[TableIndex].line << "行 "**  **<< table[nIndex].CharacterInTableName << "与"**  **<< table[TableIndex + 1].CharacterInTableName**  **<< "之间缺少运算符 \n";**  **break;**  **case 8:**  **cout << "第" << table[nIndex].line << "行：" << table[nIndex + 1].CharacterInTableName << " 应为'(' \n";**  **break;**  **case 9:**  **cout << "第" << table[TableIndex].line << "行 "**  **<< table[TableIndex].CharacterInTableName << "与" << table[nIndex + 1].CharacterInTableName << "之间缺少'(' \n";**  **break;**  **case 10:**  **cout << "第" << table[TableIndex - 1].line**  **<< " 行: 缺少'then'" << endl;**  **break;**  **case 11:**  **cout << "第" << table[TableIndex].line << " 行:"**  **<< table[nIndex].CharacterInTableName << "应为then \n";**  **break;**  **case 12:**  **cout << "第" << table[TableIndex].line << " 行: end 后不能接 " << table[TableIndex].CharacterInTableName << endl;**  **break;**  **case 13:**  **cout << "第" << table[nIndex].line << "行： "**  **<< table[nIndex - 1].CharacterInTableName << "与 "**  **<< table[TableIndex].CharacterInTableName << "之间缺少变量 \n";**  **break;**  **case 14:**  **cout << "第" << table[nIndex].line << "行 "**  **<< table[nIndex].CharacterInTableName << "后缺少';' \n";**  **break;**  **case 15:**  **cout << "第" << table[TableIndex].line << " 行:"**  **<< table[nIndex].CharacterInTableName << "应为')' \n";**  **break;**  **case 16:**  **cout << "第" << table[TableIndex].line**  **<< " 行,begin 后不能接 "**  **<< table[TableIndex].CharacterInTableName << endl;**  **break;**  **}**  **}** | | |