**编译原理课程实验报告**

**实验4：词法分析+语法分析+语义分析**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 李国建 | | 院系 | | 计算机学院 | | | 学号 | | | | 1160300426 | | |
| 任课教师 | | 辛明影 | | | | 指导教师 | 辛明影 | | | | | | | |
| 实验地点 | | 格物208 | | | | 实验时间 | 2019/04/28 | | | | | | | |
| 实验课表现 | | 出勤、表现得分 | |  | | 实验报告  得分 |  | | | 实验总分 | | |  | |
| 操作结果得分 | |  | |
| **一、需求分析** | | | | | | | | | | | 得分 | | |  |
| 要求：阐述系统所要完成的功能。  将词法分析、语法分析和语义分析结合起来，基本功能如下：  **词法分析：**  识别以下几类单词：   * 标识符（由大小写字母、数字以及下划线组成，但必须以字母或者下划线开头） * 关键字（①类型关键字：整型、浮点型、布尔型、记录型；②分支结构中的if和else；③循环结构中的do和while； * 运算符（①算术运算符；②关系运算符**；③**逻辑运算） * 界符（①用于赋值语句的界符，如“=”；②用于句子结尾的界符，如“；”； * 常数（无符号整数和浮点数等） * 注释（/\*……\*/形式）   **语法分析：**  （1）能识别以下几类语句：   * 声明语句（变量声明**）** * 表达式及赋值语句（简单赋值） * 分支语句：if\_then\_else * 循环语句：do\_while   （2）要求编写自动计算CLOSURE(I)和GOTO函数的程序，并自动生成LR分析表。(选做)  （3）具备简单语法错误处理能力，能准确给出错误所在位置，并采用可行的错误恢复策略。输出的错误提示信息格式如下：  Error at Line [行号]：[说明文字](选做)  （4）系统的输入形式：要求可以通过文件导入文法和测试用例,测试用例要涵盖“实验内容”第（1）条中列出的各种类型的语句，并设置一些语法错误。  （5）系统的输出分为两部分：一部分是打印输出语法分析器的LR分析表。另一部分是打印输出语法分析结果，既输出归约时的产生式序列  **语义分析：**  （1）能分析以下几类语句，并生成中间代码（三地址指令和四元式形式）：   * 声明语句（包括变量声明、数组声明、记录声明和过程声明**）** * 表达式及赋值语句（包括数组元素的引用和赋值） * 分支语句：if\_then\_else * 循环语句：do\_while * 过程调用语句   （2）具备语义错误处理能力，包括变量或函数重复声明、变量或函数引用前未声明、运算符和运算分量之间的类型不匹配（如整型变量与数组变量相加减）等错误，能准确给出错误所在位置，并采用可行的错误恢复策略。输出的错误提示信息格式如下：  Error at Line [行号]：[说明文字]  （3）系统的输入形式：要求能够通过文件导入测试用例。测试用例要涵盖第（1）条中列出的各种类型的语句，以及第（2）条中列出的各种类型的错误。  （4）系统的输出分为两部分：一部分是打印输出符号表。另一部分是打印输出三地址指令和四元式序列，格式如下图所示（以输入语句“while a<b do if c<d then x=y+z else x=y-z”为例）：  1 : ( *j*<, *a* , *b* , 3 ) if *a* < *b* goto 3  2 : ( *j* , - , - , 11) goto 11  3 : ( *j*<, *c* , *d* , 5 ) if *c* < *d* goto 5  4 : ( *j* , - , - , 8 ) goto 8  5 : ( + , *y* , *z* , *t1*) *t1* = *y* + *z*  6 : ( = , *t1* , - , *x* ) *x* = *t1*  7 : ( *j* , - , - , 1 ) goto 1  8 : ( - , *y* , *z* , *t2*) *t2* = *y* - *z*  9 : ( = , *t2* , - , *x* ) *x* = *t2*  10 : ( *j* , - , - , 1 ) goto 1  11: | | | | | | | | | | | | | | |
| **二、文法设计** | | | | | | | | | | | 得分 | | |  |
| 要求：给出如下语言成分所对应的语义动作   * 声明语句（变量声明 * 表达式及赋值语句 * 分支语句：if\_then\_else * 循环语句：do\_while   声明语句：  D -> T L ; {enter(L.name, T.type, offset);offset = offset + T.width}  T -> int {T.type = int, T.width = 4}  T -> float {T.type = float, T.width = 4}  T -> double {T.type = double, T.width = 8}  L -> id {L.name = id}  表达式及赋值语句：  S -> id = E ; {p = lookup(id.name);  if p != nil then gencode(p` = ` E.addr) else error}  E -> E + E {E.addr = newtemp; gencode(E.addr` = `E1.addr `+` E2.addr)}  E -> E \* E {E.addr = newtemp; gencode(E.addr` = `E1.addr `\*` E2.addr)}  E -> - E {E.addr = newtemp; gencode(E.addr `=` -E.addr)}  E -> ( E ) {E.addr = E1.addr}  E -> id {p = lookup(id.name); if p != nil then E.addr = p else error}  E -> num { E.addr = num }  分支和循环语句：  B -> B COMP B {B.addr = newtemp;  gencode(B.addr `=` B1.addr COMP.addr B2.addr)}  B -> not B {B.truelist = B1.falselist; B.false = B1.truelist}  B -> ( B ) {B.truelist = B1.truelist; B.falselist = B1.falselist}  B -> E > E | E < E | E <= E | E >= E{B.truelist = makelist(nextquad);  B.falselist = makelist(nextquad + 1);  gencode(`if` E1.addr relop.op E2.addr `goto -`);  gencode(`goto -`)}  B -> true {B.truelist = makelist(nextquad); gencode(`goto -`)}  B -> false {B.falselist = makelist(nextquad); gencode(`goto -`)}  COMP -> or {COMP.quad = nextquad}  COM -> and {COMP.quad = nextquad}  TH -> then {TH.quad = nextquad}  WH -> while { WH.quad = nextquad }  DO -> do {DO.quad = nextquad}  S -> if B TH S {backpatch(B.truelist, TH.quad);  S.nextlist = merge(B.falselist, S.nextlist)}  S -> WH B DO S {backpatch(S1.nextlist, WH.quad);backpatch(B.truelist, DO.quad); S.nextlistB.falselist; gencode(`goto` WH.quad)}  S -> A {S.nextlist = nil} | | | | | | | | | | | | | | |
| **三、系统设计** | | | | | | | | | | | 得分 | | |  |
| 要求：分为系统概要设计和系统详细设计。  （1）系统概要设计：给出必要的系统宏观层面设计图，如系统框架图、数据流图、功能模块结构图等以及相应的文字说明。  （2）系统详细设计：对如下工作进行展开描述   * 核心数据结构的设计 * 主要功能函数说明 * 程序核心部分的程序流程图   （1）  数据流图：    源程序既可以从词法分析到语法分析再到语义分析，也可以直接通过词法分析到达语法制导翻译，最后都是输出四元组。  本程序将语法翻译和语义翻译结合起来，没有生成中间的语法树，直接利用词法单元，生成了中间代码。  功能模块结构图：    （2）   * 核心数据结构的设计  1. 栈：   为了存储规约时候的状态和词法缓存区序列  **public** **static** Stack<Integer> *stateStack* = **new** Stack<Integer>(); // 状态栈  **public** **static** Stack<String> *symbolStack* = **new** Stack<String>(); // 符号栈  语义分析，需要新加一个属性值，因此需要一个新的栈——属性栈  **public** **static** Stack<Attribute> *valueStack* = **new** Stack<Attribute>(); // 属性值栈  2. 属性类：  **public** **class** Attribute {  **private** String value;  **private** Integer width;  **private** List<Integer> truelist;  **private** List<Integer> falselist;  **private** Integer quad;  **private** List<Integer> nextlist;  }  属性栈中存储的数据类型，val 是值，width是声明语句中使用的，其他类型为控制语句回填时使用。  token ：  **public** **class** Token {  **private** String word;  **private** String code;  **private** String value;  }  存储每一个单词及其种别码，如果是标识符，value存储的是其在符号表中的入口。  3．队列：  存储输入的词法单元缓冲区  **public** **static** Queue<String> *buffer* = **new** LinkedList<String>(); // 缓冲区  4． Map：  **private** **static** Map<String, String> *KEYKIND*; // 编码  **static** {  *KEYKIND* = **new** HashMap<String, String>();  *KEYKIND*.put("(", "SLP");  *KEYKIND*.put(")", "SRP");  ……  }  存储所有的关键字、运算符、界符……  5. List:  **private** **static** List<Token> *tokenList* = **new** ArrayList<Token>();  存储token序列，程序中还有其他List，这里只是一个代表。  6. 数组：  存储 Action 表 和 Goto 表：  **public** **static** String[][] *ACTION*;  **public** **static** String[][] *GOTO*;   * 主要功能函数说明   **1.读取源码文件：**  **public** **static** String readFile(String fileName)  该函数将输入的源码文件中全部的符号存储在一个字符串中，返回这个字符串。  **2.词法分析器入口**  **public** **void** parser()  分析词法的函数。  **3.处理以不同类别符号开始的单词**  **public** **int** dealStartedDigit(**int** i)  输入每个单词的开始字符在整个符号串中的位置（下标），输出该单词最后一个字符的位置（下标）。期间，识别出相应的单词后，调用对应的处理函数。  **4.求First 集：**  // 计算字符串的 First 集  **public** **static** List<String> first\_str(List<String> x);  // 计算一个字符的 First 集  **public** **static** List<String> first\_sig(String x);  **5.求文法闭包：**  // 递归计算文法闭包  **public** **static** List<String> closure(List<String> I);  // 计算一层闭包  **public** **static** List<String> CLOSURE(List<String> I);  **6.求 Goto 函数：**  // 计算转移函数 GO  **public** **static** List<String> Go(List<String> I, String x);  **7.求 Action 和 Goto 表：**  // 创建 LR  **public** **static** **void** createLR();  **8.利用 LR（1）进行规约：**  LR 类中的重要函数（语义分析，规约时使用）：  // 利用生成的 ACTION 和 GOTO 表对源码进行规约  **public** **static** **boolean** analyzeLR(List<String> input)  **9.赋值语句的语义分析：**  // 赋值语句  *assignment*(trans\_state);  **10.布尔表达式的语义分析：**  // 布尔表达式  *boolAnaly*(trans\_state);  **11.条件循环语句的语义分析：**  // 条件、循环语句  *jump*(trans\_state);  **12.回填技术的三个函数：**  /\*\*  \* makelist(i)  \* 创建一个只包含 i 的新链表  \* \*/  **public** **static** List<Integer> makelist(Integer i)  /\*\*  \* merge(p1, p2)  \* 合并由 p1 和 p2 指向的两个链表，并返回新链表  \* \*/  **public** **static** List<Integer> merge(List<Integer> p1, List<Integer> p2)  /\*\*  \* backpatch(p, i)  \* 将 i 插入到链表 p 中的每一条语句中，作为该语句的目标标号  \* \*/  **public** **static** **void** backpatch(List<Integer> p, Integer i)   * 程序核心部分的程序流程图   词法分析部分流程图：    语法分析部分流程图：    语义分析四大功能：    系统的程序流程图 | | | | | | | | | | | | | | |
| **四、系统实现及结果分析** | | | | | | | | | 得分 | | | | |  |
| 要求：对如下内容展开描述。   1. 系统实现过程中遇到的问题   问题1：如何构造自动机并根据自动机实现词法分析？  解决方法：虽然分析的时候需要先构造自动机再进行分析，但是写程序，可以直接写分析，没必要一定要有自动机再分析，只要功能实现了即可。  问题2：求闭包时，Java 集合类型在遍历时，无法添加元素。  解决方法：利用递归的方式来计算闭包。  问题3：规约时，每个元素的属性值不一定只有一个，但只能对应属性栈中一个。  解决方法：新建一个属性类，存储元素属性，多属性的元素可以存储在类中的属性中。  （2）针对一测试程序输出其语义分析结果  测试程序：  int a;  int b;  int c;  int d;  a=1;  b=2;  c=3;  d=10;  while(a<10)  {  a = a+4;  b = b+5;  if(a+b>c+d)  {  c = c+9;  }  }  **词法分析，输出的token：**    **语法分析，状态转换表：**    **语义分析生成的中间代码，输出结果：**  机器代码：    （3）输出针对此测试程序经过语义分析后的**符号表**  **符号表：**    （4）输出针对此测试程序对应的**错误报告**  **词法分析时的错误：**  测试程序：    错误报告：    **语法分析时的错误：**  不规范的程序：  int i;  i=i+1;  if() {  i=i+1;  } else {  i=i-1;  }  do {  i=i+2;  }while(true);  错误报告：  字符 ‘)’ 规约错误，程序不规范。  **语义分析时的错误：**  带有错误的测试程序：  int a;  int a;  a=1;  b=2;  输出：  变量重复声明!!!  变量未声明!!!  （5）分析实验结果  编译器首先利用词法分析器，生成token序列和空的符号表（存储变量）。  将生成的token 存储在文件中。  然后语法分析首先读取产生式，生成 Action 表 和 Goto 表，然后读取文件中的 token序列，进行规约，规约的同时进行语义分析。  进行语义分析时，程序首先识别出 a, b, c, d 四个变量，根据它们的声明，判断它们的类型值。规约到声明语句时，对类型做检查，然后进行操作，更新符号表。  然后，规约时，规约到赋值语句，边识别边生成代码。  然后，识别到 while 循环语句，进行 while 语义处理；  识别到 if 语句时，进行 条件语句语义处理  处理跳转语句时，同时利用回填技术确定跳转的语句标号 | | | | | | | | | | | | | | |
| 指导教师评语：  日期： | | | | | | | | | | | | | | |