**《编译原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | **刘玉莽** | | | **年级** | **2018级** |
| **学号** | | **20184261** | | | **专业、班级** | **物联网02班** |
| **实验名称** | **语义分析程序的设计与实现** | | | | | |
| **实验时间** | **5/27** | | **实验地点** | **A主404** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □综合性** | | |
| 教师评价：  完成实验内容，达到实验要求，实验数据和结果正确，报告内容详实。  程序质量得分：7，实验报告得分：2，实验项目总得分：  评价教师签名：张敏 | | | | | | |
| 一、实验目的  1、掌握语法制导定义和语法翻译方案  2、编写语义分析程序，实现实验内容要求的语义分析器 | | | | | | |
| 二、实验项目内容  一）实验内容  设计并实现实验一中的C语言子集的语法制导翻译程序，语义分析及类型检查，并打印分析结果。要求实现以下功能：  1、必做项：   1. 能够实现对整型（int）及布尔变量的类型检查，两类变量不能相互赋值及运算；仅整型变量才能参与算术运算； 2. 实现整数计算器的语义动作定义及语法制导翻译，并绘制出相应语法树； 3. 为表达式“（2+3”生成错误信息“丢失右括号”； 4. 为表达式“2 3”生成错误信息“丢失运算符”； 5. 为表达式“（2+）”生成错误信息“丢失操作数”；   2）可选项：  能判断源代码是否符合以下语义假设并给出相应错误具体位置；   1. 过程/函数仅能定义一次、程序中所有变量均不能重名、过程/函数不可嵌套定义； 2. 定位源代码中的错误位置。 3. 能检查结构体中域是否与变量重名，不同结构体中域是否重名； | | | | | | |
| 1. 实验过程或算法（源程序） 2. **语义分析程序总体说明：**本次实验完成了实验中**要求的所有必做与选做要求**，即可以完成下面列出的功能：   必做项：   1. 能够实现对整型（int）及布尔变量的类型检查，两类变量不能相互赋值及运算；仅整型变量才能参与算术运算； 2. 实现整数计算器的语义动作定义及语法制导翻译，并绘制出相应语法树； 3. 为表达式“（2+3”生成错误信息“丢失右括号”； 4. 为表达式“2 3”生成错误信息“丢失运算符”； 5. 为表达式“（2+）”生成错误信息“丢失操作数”；   可选项：  能判断源代码是否符合以下语义假设并给出相应错误具体位置；   1. 过程/函数仅能定义一次、程序中所有变量均不能重名、过程/函数不可嵌套定义； 2. 定位源代码中的错误位置。 3. 能检查结构体中域是否与变量重名，不同结构体中域是否重名； 4. **语法制导翻译方法说明：**采用**递归下降**的方法进行语法分析，因为相比于其他分析方法，递归下降的方法更适于手工编码，当然这个就需要**消除左递归。** 5. 对于消除了左递归的整数计算器的语义动作定义及语法如下：   <expr> ::= <term> <expr\_tail>  <expr\_tail> ::= + <term> <expr\_tail>  | - <term> <expr\_tail>  | <empty>  <term> ::= <factor> <term\_tail>  <term\_tail> ::= \* <factor> <term\_tail>  | / <factor> <term\_tail>  | <empty>  <factor> ::= ( <expr> )  | Num   1. **错误处理：** 2. **对于类型检查：**   首先如下面所示，读于程序读入的每个标识符都会有一个type字段，标明其类型。    那么在遇到算数运算的时候，就可以通过type字段检验类型是否是整型。     1. **为表达式“（2+3”生成错误信息“丢失右括号”**   参考上面对于文法的定义，括号的出现定义在factor中，那么就可以在这里当出现了左括号后检查是否存在右括号。     1. **为表达式“2 3”生成错误信息“丢失运算符”**   定义了flagOperator变量，用这个变量来指示在四则运算的时候运算符是否缺失，         1. **为表达式“（2+）”生成错误信息“丢失操作数”**   这个的判断是如果在递归下降中一直没找到操作数的时候，就可以判定这个运算表达式丢失了操作数     1. **过程/函数仅能定义一次、程序中所有变量均不能重名、过程/函数不可嵌套定义**         如上所示，对于不能重复定义函数名，结构体，以及局部变量的要求，所采取的策略都是一样的，每检测到一个这样的变量，都会检测对应的class字段，如果已经存在，那么就说明这个变量已经存在，能够这样判断的原因是因为我们在词法分析阶段，定义的词法分析器在判断标识符时是像下面这样工作的    每次都存存储标识符字段最开始的地方查询，通过hash字段判断两个标识符是不是一样的，如果一样的话，id\_ind就会停止加1.  对于函数不能嵌套定义的问题，这里采取的方式实在已有的文法分析基础上，当分析到一个区域已经定义了函数的时候，里面如果在定义函数的话，这个程序就会给出错误，     1. **定位源代码中的错误位置。**     对于所有的报错与提示信息，这里都给出了错误所在的行。   1. **能检查结构体中域是否与变量重名，不同结构体中域是否重名；**   解决方法与5）中的解决方法一致。   1. **程序说明：符号表说明、语法分析树构造过程** 2. **符号表说明**     程序对于每一个token，都依据上面的定义给出了相应的标识   1. **语法分析树构造过程**     构造    打印 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程   1. **测试样例及结果展示：** 2. 包含各种错误的测试样例，包括**整型与字符型变量做四则运算结构体和函数重复定义，变量重复定义，函数嵌套定义，运算表达式丢失右括号、丢失运算符、丢失操作数。**   #include <stdio.h>  struct strc {  char name[20];  int id;  int val;  char test[3];  };  struct strc {  int testStruct;  };  int fun(){  int a1, b1;  int min(){  return a1 + b1;  }  return 1;  }  int fun(){  int testFun;  return 1;  }  int main(){  // find the minimum val  int a, b, c, d;  int a, b;  char cha1;  int arr[3];  a = d + (b + c);  a = c + cha1;  c = (b + d;  b = a c;  d = (2+);  @ // undefined  if(a == b){  c = 2 \* a;  }  else if(a < b){  c = a + b;  }  while (a)  {  a--;  }  return 0;  }   1. **测试结果**   为了结果报错位置清楚，再将重点出错位置标明      结果      这里对于下面的正确的表达式，其语法分析树如下       1. **正确的测试样例**   #include <stdio.h>  struct strc {  char name[20];  int id;  int val;  char test[3];  };  int fun(){  int a1, a2;  return 1;  }  int main(){  // find the minimum val  int a, b, c, d;  char cha1;  int arr[3];  a = d + (b + c);  if(a == b){  c = 2 \* a;  }  else if(a < b){  c = a + b;  }  while (a)  {  a--;  }  return 0;  }  **测试结果**        可以看到对于相应的表达式这里也给出了正确的语法分析树 | | | | | | |
| 五、实验总结   1. **实验过程中所遇到的问题及解决办法**     刚开始上面写的是token == ‘\*’到时程序陷入了循环，最后对比才发现应该改为token == Mul，问题解决。   1. **实验中还遇到了有关内存的问题**   这种问题真的时头大，后面就也不知道改了些什么，他又好了。 | | | | | | |