**《编译原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | **龙俊桦** | | | **年级** | **2017** |
| **学号** | | **20174302** | | | **专业、班级** | **计算机科学与技术05班** |
| **实验名称** | **语义分析程序的设计与实现** | | | | | |
| **实验时间** | **2020.11.05** | | **实验地点** | **A主410** | | |
| **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；   二）实验要求：   1. 自由选用程序设计语言（java，python，c/c++）作为实现语言，手工编写语法分析程序。 2. 提交实验报告及源代码。实验报告需严格遵循学校文档规范，内容包含对应文法、语法分析测试用例。   三）C语言子集:  数据类型: **int**, 无符号整数, 取值范围0-9999  int a;  int a,b;  int a = 1;  算术运算符: **+**,**-**  a = b + 1;  a = b + c;  赋值运算符: **=**  a = 1;  关系运算符: **==** ,**>**,**<,<>,>=,<=**  a = (b==c);  a = (b>c);  a = (b<c);  逻辑运算符: **&&**, **||, !**  a = (b&&c);  a = (b||c);  a = (!b);  条件语句: **if**  if(a==b)  {  };  循环语句: **while**  while(a==b)  {  };  输入,输出: **get**,**put**  get(a);  put(a);  语句结束符: **;**  **加分项：**  条件语句 **if else**  if(a==b)  {  };  else  {  }; | | | | | | |
| 三、实验过程或算法（源程序）   1. 语义分析程序总体说明   这一次实验的语义分析器基本上完成了类型int和bool的类型检查，完成了变量未定义和重定义的检查，以及bool以及int的非法赋值的检查。完成了一个语义分析器的基本功能。   1. 语义分析说明 2. 构建抽象语法树(AST)   上一次的实验2最后输出了一棵分析树，为了更好地利用这棵分析树，同时避免去修改已有的实验2的代码，我选择了根据分析树先构造出一棵语法分析树(AST),代码主要见generateAST.cpp，下面展示部分代码。  我构建了一个结构体来作为AST的节点。  struct ASTnode{      int kind;      vector<ASTnode\*> sons;      int tokenIndex;      string tokenString;  };  然后就是遍历分析树，遍历的过程中，构建出这一棵AST，展示部分代码如下。  例如，这是在处理到ASSIGN节点时编写的代码：  void processAssign(TreeNode\* assign, ASTnode\* father){      ASTnode\* variable = new ASTnode();      variable->kind = VARIABLE;      variable->sons.clear();      variable->tokenIndex = assign->sons[0]->tokenIndex;      variable->tokenString = assign->sons[0]->tokenString;      father->sons.push\_back(variable);      processExpression(assign->sons[2],father);  }  按照这种方法，针对每一个产生式编写一个相应的过程，便可以将一棵分析树转换为一棵抽象语法树，然后便可以开始进行语义分析了。  下面展示构建抽象语法树的结果：  假设我们输入程序    构建出的语法树(分析树)为:    根据分析树构建出的AST为：    将AST画出来，为：     1. 编写符号表相关函数   然后是编写符号表相关函数。这里我们建立一个符号表，每一个表项叫做tableItem, 代码如下，具体代码请见symbolTable.cpp：  struct tableItem{      int type;      string name;      stack<pair<int,int>> vals;      int dscope;  };  我使用了一个map来存放每一个变量名以及与其对应的item。  然后就是编写符号表的插入，删除等相关内容，下面展示enterTable函数相关代码：     1. 语义分析   符号表相关函数也写完了，下面便可以开始进行语义分析了，相关代码见SemanticAnalyzer.cpp。  通过对AST进行遍历以及对于不同种类的ASTnode编写分析代码，可以实现语义分析，下面是对于int类型的node进行语义分析的代码。     1. 错误处理   在进行语义分析时，我会返回一个叫做errorInfo的结构体，其内容大致如下所示：    当发生错误时，我可以获取到相应的错误的Index，然后通过    这一行代码标记出错误位置，并且通过errorInfo.errorDetail变量来获取错误的具体信息。 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程   1. 必选测试样例: 2. 简易计算器   //输入数据num1,num2,op，根据op确定操作进行运算，最后输出运算结果ans  int num1,num2,op,ans;  get(num1,num2,op);  if(op==0)  {  ans = num1 + num2;  };  if(op==1)  {  ans = num1 - num2;  };  if(op==2)  {  ans = num1 & num2;  };  if(op==3)  {  ans = num1 | num2;  };  put(ans);  对程序进行语义分析结果如下：  打印的AST如下图所示：    语法分析结果如下：    结果显示程序存在语义错误，即ans可能没有赋值。  分析程序我们可以看到，ans没有被初始化，所以有可能ans没有进入任何一个if，最后就没有值，所以这里的语义分析是正确无误的。   1. 跑马灯   //循环输入op，改变输出结果out，输入0则结束程序  int num0,num1,out,op;  num1 = 3333;  num2 = 6666;  num3 = 9999;  op = 1;  while(op>0)  {  if(op==1)  {  out = num1;  };  if(op==2)  {  out = num2;  };  if(op==2)  {  out = num3;  };  put(out);  get(op);  };  对程序进行语义分析结果如下：  打印的AST如下图所示：    语义分析结果如下：    可以看到，程序没有语义错误。   1. 自定义测试样例(包含自定义的词法错误类型): 2. 布尔赋值错误   输入程序为：    进行语义分析结果如下：     1. 变量未定义错误   输入程序为：    进行语义分析结果如下：     1. 变量重定义错误     进行语义分析结果如下：     1. 为表达式“（2+3”生成错误信息“丢失右括号”   输入为：    输出分析结果:  因为这是语法错误，所以在语法分析阶段给出结果。     1. 为表达式“2 3”生成错误信息“丢失运算符”；   输入为：    输出分析结果：  因为这是语法错误，所以在语法分析阶段给出结果。     1. 为表达式“（2+）”生成错误信息“丢失操作数”；   输入为：    输出分析结果：  因为这是语法错误，所以在语法分析阶段给出结果。 | | | | | | |
| 五、实验总结  这次的语义分析我主要分成了三个部分来做，首先是将语法树转换为抽象语法树，然后是编写符号表相关函数，最后是对抽象语法树进行语义分析。  其中最为复杂的部分应该就是语义分析时对于符号表的操作，因为你要考虑到变量是否被定义了，或者是变量是否没有被定义或是没有被赋值，所以在编写时要考虑各种情况。  最后看来我没有使用语法制导翻译的方法来进行语义分析，因为我想利用起来上一次实验输出的语法树，同时我也不想去修改上一次编写好的语法分析过程，所以最后我的方案就是选择了直接将语法树转换为了抽象语法树，然后再进行语义分析，最后也还是基本满足了实验要求，收获很大。 | | | | | | |