**《编译原理》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | | **年级** |  |
| **学号** | |  | | | **专业、班级、** | **计科卓越班** |
| **实验名称** | **语义分析** | | | | | |
| **实验时间** |  | | **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：9  评价教师签名：张敏 | | | | | | |
| 一、实验目的  掌握语义分析基础，实现语义分析器 | | | | | | |
| 二、实验项目内容  1.在词法及语法分析程序的基础上，编写一个程序对使用类C语言书写的源代码进行语法制导翻译，语义分析及类型检查，并打印分析结果。  2.程序要能够检查源代码中是否符合以下语义要求：  （1）最低要求3.1：  【1】能够实现对整型（int）和浮点型（float）变量的类型检查，两类变量不能相互赋值及运算；仅整型及浮点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，并绘制出相应抽象语法树；  【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）其他要求3.2:  【1】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；  【2】函数仅能定义一次、程序中所有变量均不能重名、函数不可嵌套定义；  【3】定位源代码中的错误位置。  （3）其他要求3.3:  【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；  【2】编写一个FreeSymbolTable函数，在main函数的最后调用此函数时，可以讲整个符号表的内存释放，避免内存泄露。 | | | | | | |
| 三、实验过程或算法（源程序）  **1.语义分析总体实现思想**  在实验2的语法分析里，我们已经建立了一个语法分析树。考虑到bison在进行语法分析时是自底向上的，因此如果在建立语法分析树的同时进行语义分析，则我们无法确定程序中变量的作用域。因此我们可以先建立语法分析树，然后再自顶向下地遍历语法分析树，采用深度优先搜索的方式，建立变量符号表、函数符号表和结构体符号表，对树中节点进行语义分析。  实现功能：  支持八进制、十进制、十六进制整数、十进制浮点数，数据类型定义为int和float。支持一维数组。可定义函数和结构体，能对变量、函数、结构体是否重名进行检查，并定位代码中错误的位置。能够实现对整型和浮点型的类型检查。  **2.确定当前作用域**  当进入不同的函数或者块时，作用域会发生变化。我们采用如下的代码来定义当前的作用域。      我们假设初始作用域为0。global\_scope为全局的作用域变量，scope\_stack为一个作用域变量栈，top\_scope指向栈顶。调用语义分析程序时，向当前节点传递此时的作用域值。每次进入一个新的块时，如函数或者{}里时，将当前的scope存入scope\_stack中，即scope\_stack[top\_scope++]=scope，然后改变scope，scope=++global\_scope。当退出一个块时，出栈操作，有scope = scope\_stack[--top\_scope]。这样每次进入一个块时，当前作用域被赋新值，当退出一个块时，当前作用域被置为上一层作用域值。  **3.语法分析树节点类型**    当节点类型为数组时，记录数组元素的类型和数组大小；当节点类型为结构体或函数名时，记录结构体名或函数名。  当比较两个节点类型是否相同时，调用typeCheck函数。    **4.语法分析树节点**    **5.建立变量符号表**    采用哈希表的方式，对变量名称求哈希值，符号表中的每一项存储变量名称、类型和作用域。当不同变量（如名称不同、名称相同但作用域不同）的哈希值相同时，采用链表的方式进行存储。这样在查找变量是否已定义时，可以快速的确定变量在符号表中的位置。  **insertVariable**函数：调用have\_symbol函数和add\_symbol函数，当变量重定义时，输出错误信息。    **have\_symbol**函数：遍历符号表判断变量是否已存在，如果存在，则返回变量类型；否则返回NULL。    **add\_symbol**函数：将变量存入符号表中。    **6.建立函数符号表**    采用哈希表的方式，对函数名求哈希值，存储方式和变量符号表一致。  **insertFunction**：对名为function\_definition的语法分析树节点进行分析，确定函数的名字、返回值类型、作用域、是函数定义还是函数声明、参数个数以及每个参数的类型。然后调用have\_func函数判断相同类型的函数是否已存入函数符号表中，根据返回值输出对应的错误信息。如果have\_fun返回值合理，则调用add\_fun存储函数信息到函数符号表中。      **have\_fun**函数：遍历函数符号表，当在符号表中找到函数名相同并且参数个数也相同的项时，则遍历形参列表，判断每个形参类型是否相同。如果这些都相同，则当此时为函数定义并且符号表中存储的也为已定义的函数时，则insertFunction函数输出错误信息：“函数重定义”；当此时为函数定义而符号表中存储的是未定义的函数时，则判断二者的返回类型是否相同，如果不同，则insertFunction函数输出错误信息：“函数声明和定义不一致”；当此时位函数声明并且符号表中存储的也是未定义的函数时，则判断二者返回类型是否相同，如果不同，则insertFunction函数输出错误信息：“两次函数声明不一致”。      **add\_func**函数：将函数信息存入函数符号表中，过程与将变量存入变量符号表中类似。    **6.建立结构体符号表**    采用哈希表的方式，对结构体名求哈希值，在结构体符号表中的存储原理与变量符号表类似。vari\_list存储在结构体中声明的变量的名字和类型，根据我们的文法，在结构体中只允许声明(int a;)，不允许定义(int a=1;)。  **insertStructure**函数：对名为struct\_specifier的语法分析树节点进行分析，调用have\_struct函数判断结构体是否已定义，调用get\_struct\_vari函数得到结构体里声明的变量链表，调用add\_struct函数将结构体存入结构体符号表中。    **have\_struct**函数：判断是否已定义名称相同的结构体，如果已定义，返回1；如果未定义，返回0.    **get\_struct\_vari**函数：对名为struct\_declaration\_list的语法分析树节点进行分析，采用深度优先搜索的方式，得到结构体中声明的变量名和类型。        **add\_struct**函数：将已定义的结构体信息存入结构体符号表中，原理和将变量存入变量符号表中类似。    **7.语义分析**  根据建立的语法分析树，采用深度优先搜索的方式，自顶向下的进行分析。在.y文件中，初始调用semantic\_analysis($$,0)，0为初始作用域。根据分析树节点的名字，递归的调用semantic\_analysis函数，对每一个节点进行处理。如对translation\_unit节点，在语法阶段定义的规则为：    对应的语义分析分析代码为：    对于function\_definition节点    第0，1规则对应函数定义，在进入函数时对作用域压栈，同时保存函数返回类型，然后递归分析parameter\_list和compound\_statement节点。当分析完毕compound\_statement节点后出栈。    对于2，3规则对应函数声明，3中parameter\_list是一个单独的作用域，在分析parameter\_list节点前压栈，在分析完毕后出栈。    对于变量声明节点：    第1条规则中在对declarator和logical\_expression节点调用semantic\_analysis程序后，需要调用typeCheck函数比较二者节点类型是否相同，如果不同则输出错误信息：“等号两边类型不匹配”。  对于logical\_expression节点：  对”+、-、\*、/、>、<”等操作符两边的表达式进行语义分析，确定节点类型，然后比较两边节点类型是否相同，如果不同则输出错误信息：“操作符两边的类型不匹配”。  当表达式对应规则为函数调用时，    需要判断是否已定义对应的函数，对节点调用functionCheck函数，遍历函数符号表，当找到相同名称，相同形参个数并且相同形参类型的函数时，返回函数类型，如果没找到在logical\_expression节点类型置为UNKNOWN，同时输出错误信息：“未定义的函数”。  当表达式对应规则为结构体调用时，    需要判断第一个子节点logical\_expression的类型是否为STRUCTURE，如果是，调用structCheck函数，判断该结构体已定义变量中是否存在名为IDENTIFIER->content的变量，如果存在，则返回变量类型，如果不存在，则父节点logical\_expression类型置为UNKNOWN，同时输出错误信息：“结构体中不存在对应变量”。  当在表达式中使用变量时，    需要调用have\_symbol函数判断相同作用域中变量符号表里是否已存在名为IDENTIFIER->content的变量，如果存在，则返回变量类型，并对primary\_expression类型赋值，如果不存在，在置primary\_expression类型为UNKNOWN，同时输出错误信息“未定义的变量”。 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程  **错误检测**：    输出错误信息    **测试语义分析**：  1.  int a;  int b;    2.  int func(int a, int b){  }  int func(int a, int c){  }    3.  void func(int a, int b);  int func(int a, int c){  }    4.  void func(int a, int b);  int func(int a, int c);    5.  struct A{  int a;  };  struct A{  int a,b;  };    6.  int a=1.0;    7.  struct A func(){  int a=0;  int b[10];  b[3]=4;  }    8.  int main(){  int a=12;  int b;  if(a>0)  b=12;  else  b=-12;  return;  }    9.  int main(){  int a=1; float b=12.0;  a+b;  }    10.  int main(){  int a[10];  a[0]=1;  a[2.0]=4;  }    11.  int main(){  func();  }    12.  int main(){  int a=10,b=13;  c=12;  }    13.  int main(){  int a;  a[12]=13;  }    14.  int main(){  int a;  a.b=13;  }    15.  struct A{  int a;  int b;  }  int main(){  struct A stu;  stu.a=13;  stu.c=15;  } | | | | | | |