编译原理实验二

111220174 张赫
mattzhang9@gmail.com

总体说明:

我在本次实验中使用SDT，即在语法树构建完成时完成语义分析。这种方式虽然需要同时在.y文件和另一个存放其他相关函数的.c文件中进行添加或者修改，但可以省去一些SDD跑树并因此递归的麻烦。虽然遇到了很多麻烦，一方面是需要从底层向上大量传值，另一方面是递归decList，但最终还是把问题圆满地解决了。

所用到的数据结构都定义在ASTNode.h文件中，这些结构大体按照实验指导攻略里设计。为函数和变量做了两个符号表链表，为定义了的struct做了一个类型链表，对于匿名struct采用按规定方式命不合法名一块加入到struct类型链表。

编译的话使用make parser进行编译。如果出现语法错，大部分情况下会直接出现段错误，这点很难防范。如果出现语义错，同行的代码会被跳过，由此该行其他一些正确的元素也会被忽略。关于注意事项已经在README.md中说明。语义分析器经受了一些基础测试和一些稍稍复杂一点的测试没有发现问题，但有可能会在经历复杂测试的时候出错，如果出错请联系我，谢谢。

添加语句位置:

1.声明语句：都添加在类似 Ext : Specifier ExtDecList的地方，通过遍历该定义子树将定义的变量加入到符号表。 该部分的难点在于想到该在这个地方添加语句以及正确地遍历树。在更底层添加语句会造成巨大的传值麻烦，在这里的话整个语句的语法树已经构建好可以很遍历地处理声明语句。对于struct型，在StructSpecifier的时候就将该类型添加到struct链表，然后一直用类型(type)传值达到和其他类型的统一。这样一来，Specifier的值被转化为类型，方便对变量符号进行存储和检查。

2.函数语句：因为在声明形参的时候整个函数的树还没有建好，所以在FunDec后面就要把形参加入到符号表。然后再对CompSt子树进行分析，将声明的变量加入符号表并检查返回值。如果返回值不符则删除函数。

3.表达式语句(Exp)：这部分的错误情况很多，而且都要在底层检查好不向上层返回。在传值的过程中，依然传类型(type)，毕竟类型才是核心。为了使分析器可以在出错的情况下还继续分析，将出错的Exp的type.u.basic设为-1以显示其错误，并在每个归约式中对于出错的Exp自动忽略不加处理。对于除了赋值以外的其他Exp双目操作，都是先检查类型然后归约，对于数组、调用函数和结构体也是先检查是否符合类型要求。对于赋值语句，还要检查是否符合左值。

一些具体函数:

总体来说，因为大部分的处理都在比较上层的地方，所以可以复用的代码比较多，这部分介绍一些比较主要的函数。对于数据类型，除了Type的理解和指导要求不同外其他是一样的。我代码中的Type就是Type_而非Type_*。

1.bool isEqualType(Type* t1, Type* t2)

该函数判断两个类型是否相同。对于基础类型，很简单可以判断。对于数组，判断维数后递归判断其声明类型（以为只能是基础或结构体了）。对于结构体，判断其存储的每个域的类型是否相同即可。

2.bool insertSymbol(Symbol** head, Symbol* s)

该函数将符号插入符号表。在本次实现中用链表实现。但因为接口比较统一，改为其他数据结构也很方便简单。但考虑到意义不大所以没有选择哈希表之类的数据结构。

3.Symbol* varDectoSym(ASTNode* spec, ASTNode *vardec)

该函数用spec中存储的type将vardec转化为符号然后传给调用者加以利用。这个函数的使用频率很高，因为定义部分实际上是整个实验最复杂的部分，而且这个函数架起了类型和变量之间的桥梁。而且因为数组的定义和使用的递归方式不同，将定义部分的数组链表进行了转置以方便Exp部分的数组使用。

4.void insertComp(ASTNode* deflist)

这个函数目前用来将comp中声明部分中声明的变量插入符号表并检查函数返回值。具体做法是遍历子树。其他诸如定义函数、定义结构体、检查函数形参等工作也是同样地原理遍历子树，后面将不加赘述。检查返回值就是匹配所有stmt推导成return的语句中Exp的类型并记录是否有return(因为不能不返回)。

5.Type* handleExp(ASTNode* n1, ASTNode* n2)

配合检查Exp的类型，尤其是对于带入'+'之类操作符的类型检查。对于函数和数组的调用会在.y文件中进一步检查。数组的推导Exp -> Exp LB EXP RB 因为在存储的时候进行过链表转置，在此处$$->ntype = *($1->ntype.u.array.elem)即可，不用单独存储维数等信息。

6.char* getArg(Symbol* func)

这个函数用来返回func函数的参数类型用来进行语义错误提示。因为函数参数列表声明的时候是从头部插入，所以如果直接迭代返回参数类型是相反的。于是用数组做了一个类似栈的结构将每个形参的类型的字符串化表达返回。除此之外，对于实参列表的返回也是相同的原理。

总结:

本次实验花了4整天时间。在初期的时候走过很多弯路，语法树节点和符号表节点的设计也改过很多次，但所幸最后还是成功了。本次实验真正用于调试的时间并不是很多。其一是因为没有采用SDD的方式在整个语法树建完以后再进行语义分析，这样减小了因为整个太过庞大而造成的代码混乱。其二是每写完一部分内容都会进行测试，保证每个部分的正确性。我觉得在设计上比较错误的是数组的存储。用链表做不仅低效而且写起来很麻烦，完全不如直接用纯字符串存储，在对数组的处理上也花了最多时间。指导中的那种设计或许看上去结构更清楚，但实际操作的时候会麻烦很多。