编译原理实验报告

实验项目二：语义分析和类型检查

• 实验目的

在词法分析和语法分析程序的基础上编写一个程序，对 C--源代码进行语义分析和类型检查，并打印分析结果。

• 实验设计

（一）设计思路

我在实验中的思路基本是在理解实验手册上讲解基本思想后进行的，首先按照实验手册上给出的例子设计了一个类型 Type 结构，用于记录不同类型的数据，区分基本类型，函数类型，数组类型和结构类型。之后，再建立一个符号表用于存储不同的符号信息。最后，我采用对已经生成好的语法树遍历的方法进行语义分析，按照附录 A 中提供的语义分析结构，逐步地向下递归处理。当遇到变量的定义时，将其类型和名字等信息存入符号表；当在下面的语句中运用到某符号时，需要先进行查表工作，然后再做一些语义分析和类型检查等处理。总的来说，由于本次实验需要处理的情况有很多，所以代码量比较大，但我还是坚持到最后，并顺利将任务做完。

（二）数据结构

这次实验的重点就是符号表的设计，因此一个好的数据结构可以使我们在设计过程中更加简单，方便，其中 Type 数据结构如下所示：

```
7  //typedef struct Type* Type;
8  typedef struct Type{
9      enum {BASIC,ARRAY,STRUCTURE,FUNCTION} kind;
10     union {
11         int basic; //1-INT,2-FLOAT
12         struct {
13             struct Type* elem;
14             int size;
15         } array;
16         struct {
17             struct structField* field;
18             char name[30];
19         }structure;
20         struct Func* func;
21     } u;
22 }Type;
```

图 1 类型 Type 数据结构

首先根据实验讲义的例子进行了一些拓展，利用一个枚举类型 kind 来表示这个 Type 属于哪一种类型，之后利用一个联合来存储一个类型应该具有的信息，例如对于一个结构类型需要将它的结构名字和结构体域中的定义的变量记录下来。这样一个 Type 结构就可以将一个变量的类型信息完整的存储下来。其中，用于存储结构类型信息和函数类型信息的数据结构定义如下：

```
typedef struct funcPara{
    Type* type;
    char* name;
    struct funcPara* next_para;
}funcPara;

typedef struct Func{
    Type* return_type;
    funcPara* para;
    int para_num;
}Func;

typedef struct structField{
    char* name;
    Type* type;
    struct structField* next;
}structField;
```

图 2 函数，结构体数据结构

在定义好类型的数据结构之后，就可以开始符号表结构的设计了，我们的符号表每一个节点中需要存储变量的类型，名字，所在的行数，以及该变量所在的作用域。数据结构如下图：

```
typedef struct semanNode{
    Type* type;
    char name[30];
    int line;
    int level;
    struct semanNode* next_in_hash;
    //struct semanNode* next_in_scope;
}semanNode;
```

图 3 符号数据结构

然后，我利用定义好的符号结构设计了一个开散列的 hash 表作为符号表的数据结构，每当遇到一个新定义的变量时，按他的名字进行 hash 值计算，插入 hash 表的相应位置。当遇到冲突时，利用开散列的处理方法，即在该节点后不断链入新的节点，来解决冲突。符号 Hash 表定义如下：

```
semanNode* hashtable[101];
structNode* struct_hashtable[101];
```

图 4 符号表定义

我最开始设计的结构主要就是这些，但是我在后面实现的过程中发现，结构体类型是有名字的，但我觉得把结构体的类型也放在符号表中有些不合适，因此我之后又设计了一个结构体的符号表，其基本类型与上面的符号表类似，用来记录定义的结构体，当之后用结构体定义变量时，首先在结构体符号表中进行查找，然后返回该结构类型 Type。这样就基本将我在这次实验中使用到的数据结构进行了完整的介绍。

（三）功能实现

（1）在定义好数据结构之后，就要开始语义分析这一功能的实现了。我发现在变量定

义阶段的主要任务就是讲一个个符号放入符号表中，首先对 specifier 进行分析，将其 Type 信息作为继承信息传给分析变量的部分，然后将变量的名字和类型等信息都存入符号表中，方便后面的查找和类型检查。

这里面我觉得主要有三个重点，第一个就是解决 hash 表冲突的问题，当两个变量的名字 hash 值相等时，会将这两个符号放入同一个符号表位置，所以需要我们利用链表的方法解决这一冲突，将新加入的变量连入原来的变量串中，从而处理冲突。第二个重点也是我的选做部分，就是要解决变量定义的作用域，这里我采用的方法并不是利用栈的方法，每次压栈，而是设置了一个 level 变量表示作用域的等级并将这个变量存入符号的信息中，在处理 CompSt 时，每一次遇到一个左花括号就将 level 加 1，遇到右括号就将 level 减 1 同时将这一作用域中的临时变量从符号表删除，用 level 来区分不同作用域的。经过测试，可以做到不同作用域同名变量的区分。最后一个重点就是要在定义时对变量的重定义检查，这里也需要先查表，发现没有重定义之后再将该变量插入表中。

（2）在将变量定义好之后就要在之后的使用中，对使用时的类型进行检查了。主要的检查任务基本都在处理 Exp 时进行处理。大部分设计我都根据测试样例和附录 A 进行设计和检查。对 Type 中记录的信息进行比较，从而检查两个变量的类型。这部分中相对比较复杂的对于函数类型的比较，因为在引用函数时需要在使用时检查参数的数量和类型。这里我采用的方式是在处理 Args 时，将一个函数类型的 Type 作为参数传入，然后每处理一个参数就会将这个 Type 类型的 func 变量加入一个参数。用这样的方式更新的函数类型再与符号表中记录的类型进行比较，从而解决这一问题。

• 实验中遇到问题及解决办法

本次实验相对代码量比较大，所以我在实验中也多次遇到 segment fault 这样的 bug，有的时候是由于对 hash 表的数组越界操作导致的，但大多数时候是由于在树的遍历或访问类型中元素时，没有对一个空对象进行判断，导致经常会访问一个空结构中的变量，从而造成段错误。本来这些错误都不是很难调的 bug，但由于有的时候工程量比较大，只能靠一些输出来找到 bug 的位置，然后才会发现这些问题。所以大部分时间都用在了查找 bug 出现位置上，在找出问题位置之后就可以相对轻松的解决这些问题。

• 实验总结

经过本次实验，了解了编译器中语义分析和类型检查实现的基本原理，虽然我设计的语义分析和类型检查肯定还存在着不足，但我从中确实学习到了很多。从自己设计一个系统的数据结构，到递归的检查语义分析的各个类型，其中都让我在对原有知识的理解更加深入。尤其是这两次实验使我对多叉树这一常用数据结构的建立和修改遍历等操作更加熟悉，这些都会对我将来的程序设计起到很大的作用。

• 编译方法

（1） 进入 Lab/Code 目录

（2） 在命令行中输入：make parser，生成可执行文件 parser

（3） 在命令行中输入：./parser <测试文件路径>，对测试文件进行语义分析