**编译原理与设计实验报告**

**姓名/学号**： 鲍吴迪/1120173588

# 实验名称

语义分析实验

# 实验目的

(1)熟悉 C 语言的语义规则，了解编译器语义分析的主要功能;

(2)掌握语义分析模块构造的相关技术和方法，设计并实现具有一定分析 功能的 C 语言语义分析模块;

(3)掌握编译器从前端到后端各个模块的工作原理，语义分析模块与其他 模块之间的交互过程。

# 实验内容

语义分析阶段的工作为基于语法分析获得的分析树构建符号表，并进行语义检查。如果存在非法的结果，请将结果报告给用户，其中语义检查的内容主要包括:

⚫ 变量使用前是否进行了定义;  
⚫ 变量是否存在重复定义;  
⚫ break 语句是否在循环语句中使用;  
⚫ 函数调用的参数个数和类型是否匹配;

⚫ 函数使用前是否进行了定义或者声明;

⚫ 运算符两边的操作数的类型是否相容;

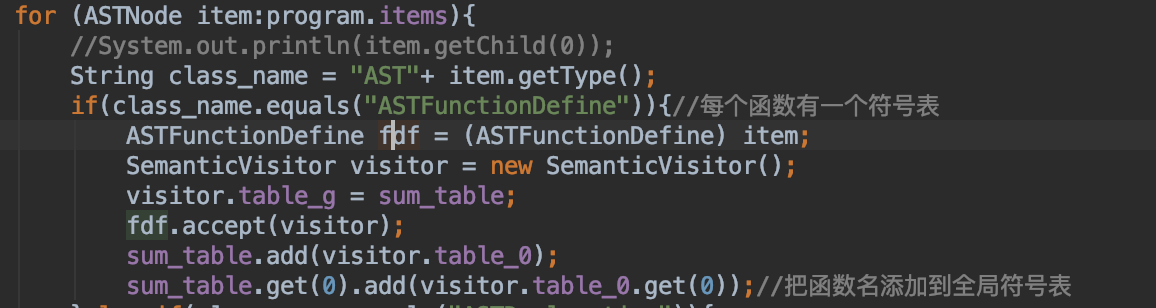
⚫ 数组访问是否越界;  
⚫ goto 的目标是否存在;

⚫... 本次语义检查的前(1)-(3)为要求完成内容，而其余为可选内容。

# 实验步骤

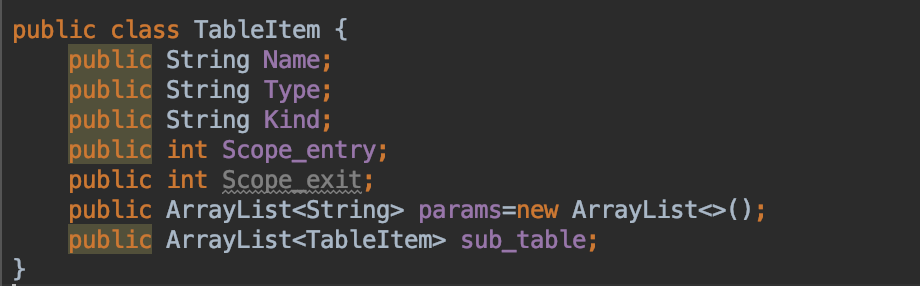
1. 对AST语法树的遍历

获取到AST语法树根结点program后，首先编写SemanticVisitor类实现ASTVisitor接口，然后通过SemanticVisitor对象对program的子节点进行访问，从而对其进行深度优先遍历。例如子节点为FunctionDefine时：



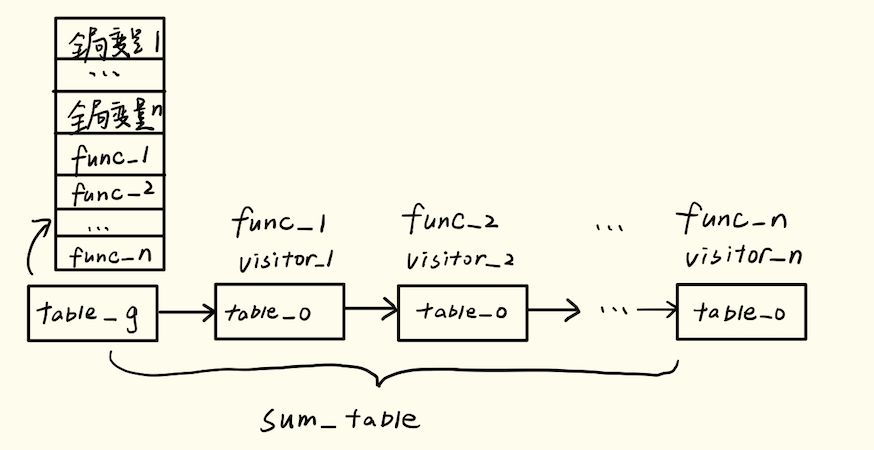
1. 符号表的建立

采用列表实现符号表的组织与建立，首先创建TableItem类，作为符号表中的项：



* name为变量名/函数名等，
* type为数据类型（如int），
* kind为声明的种类（如VariableDeclarator/FunctionDefine），
* scope\_entry/scope\_exit为其作用域开始/结束位置，
* params仅当该项为函数声明时有作用，列表里包括该函数的每个变量类型，如int f（int a，int b）；则params为[int，int]，
* sub\_table标识函数内部是否有封闭的子作用域，若存在，则为子作用域建立新的table，并加入父作用域的sub\_table属性中。

符号表为ArrayList<TableItem>,每个visitor有一个独自的符号表table\_0,用sum\_table（类型为ArrayList< ArrayList<TableItem>>）组织所有的符号表，其中sum\_table[0]为全局符号表，sum\_table[1…n]为每个函数的符号表。

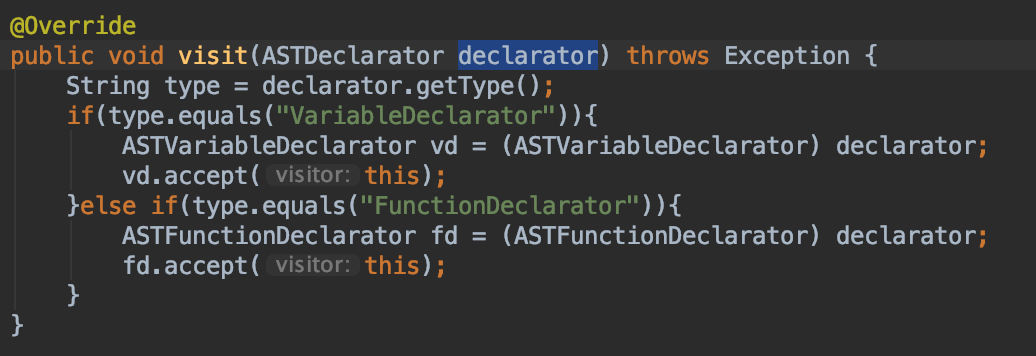


1. 表项的添加

定义变量时，通过多重visit()将identifier加入符号表中，返回时为表项添加type和kind等信息。

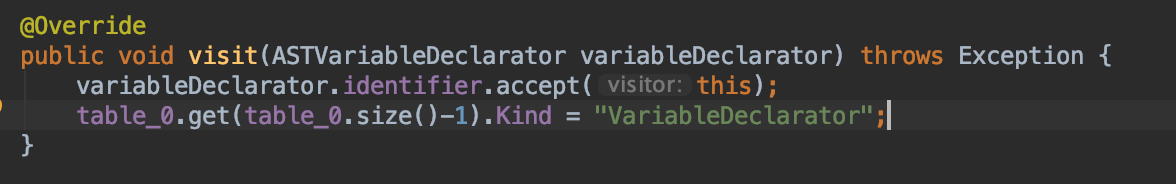
如定义int a时，首先调用visit（declarator）：

（判断定义的是变量还是函数）

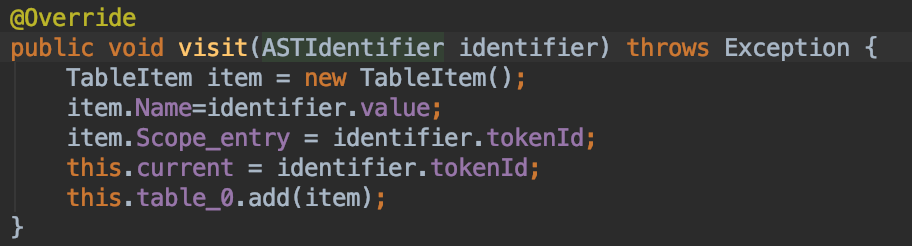


再调用visit（VariableDeclarator）：

（在identifier返回后将添加的表项kind设为VariableDeclarator）



再调用visit（Identifier）添加表项：

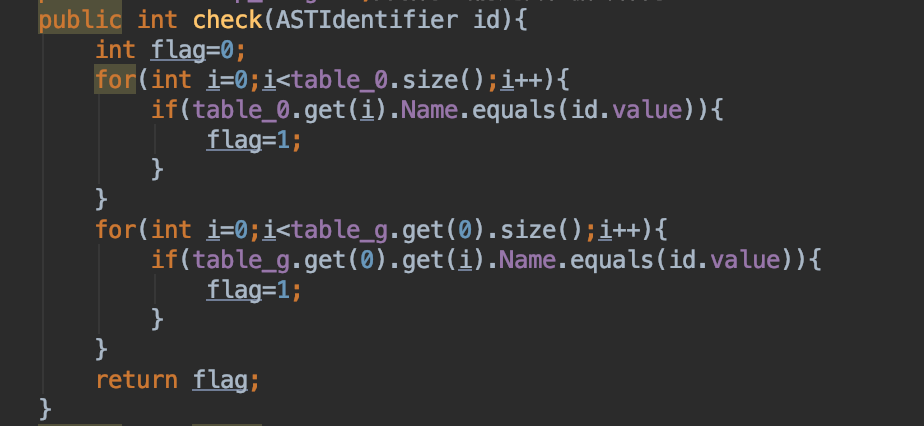


1. 变量使用前是否定义

在使用变量/调用函数时分别遍历当前符号表和全局符号表，若没有查找到相同记录，则报错。

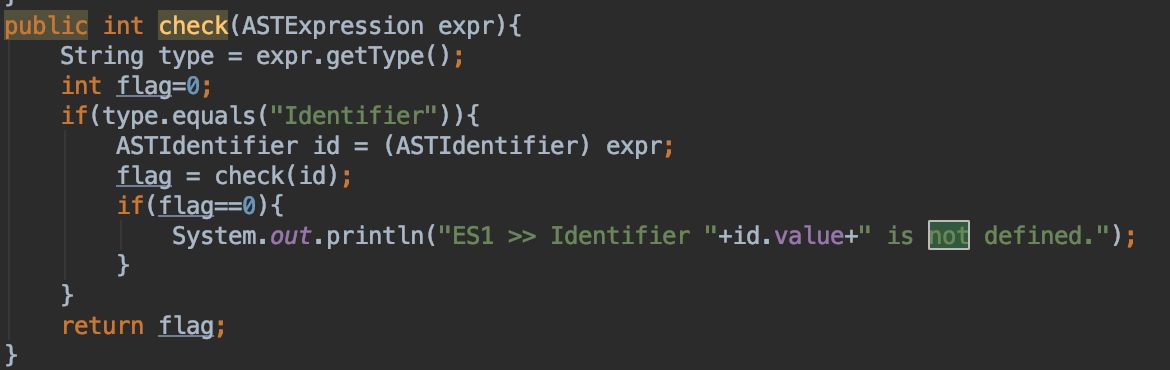
4.1 校验函数

下为通过check（）函数检测当前变量是否已定义。返回flag=1代表查找到声明记录，flag=0代表未声明。



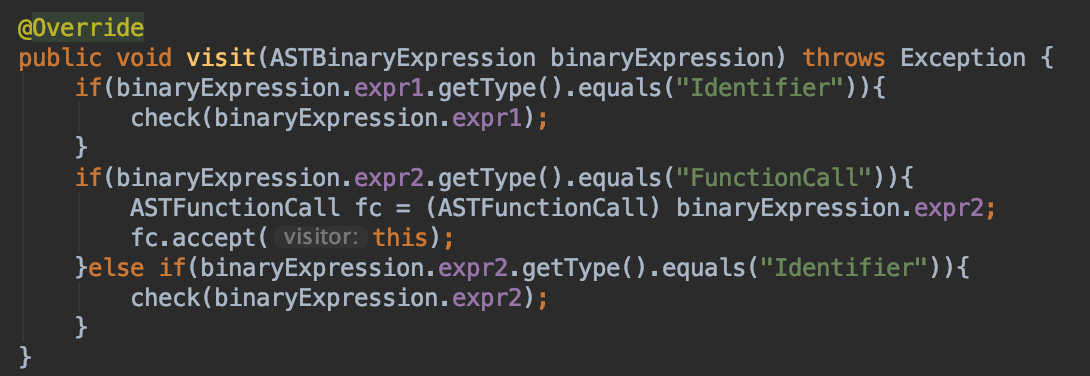
对于一元表达式，直接check其identifier属性，对于二元表达式，先判断是变量调用还是函数调用，若是变量调用，则分别check其expr1和expr2.若是函数调用则进入visit（FunctionCall），对函数名进行check。

4.2 一元表达式校验



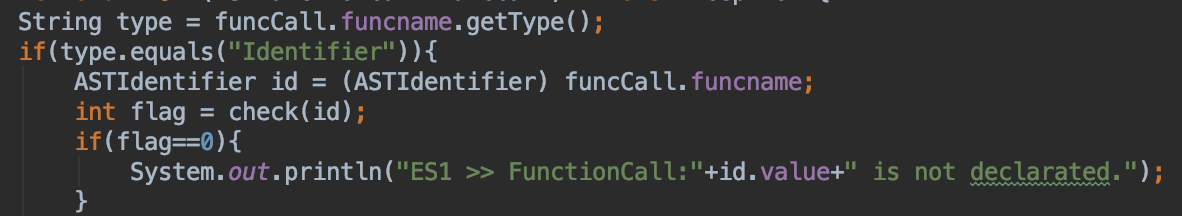
一元表达式校验是否定义

4.3 二元表达式校验



二元表达式校验是否定义

4.4 函数调用校验

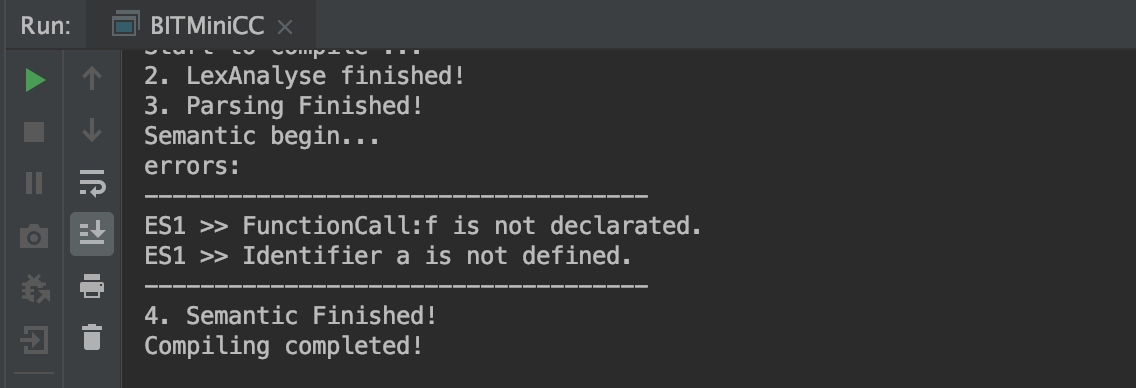


函数校验是否定义

* 1. 实验结果：

使用0\_var\_not\_defined进行测试。

1. //int f();
2. **int** main()
3. {
4. **int** res = f();
5. **int** c = a+1;
6. **return** 0;
7. }
8. **int** f(){
9. **return** 1;
10. }

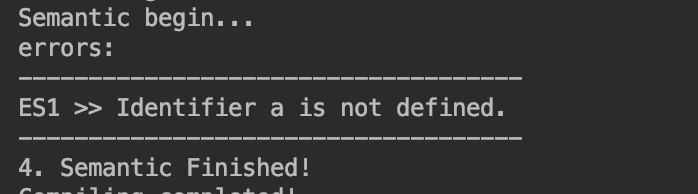


查看当前全局符号表，其中有main和f两个表项，由于在分析main时f还没有进行声明，所以先报错f is not declarated.



全局符号表

将开头的int f();取消注释后：



由于f已经声明，不再报未函数未定义错误。

1. 变量是否重复定义

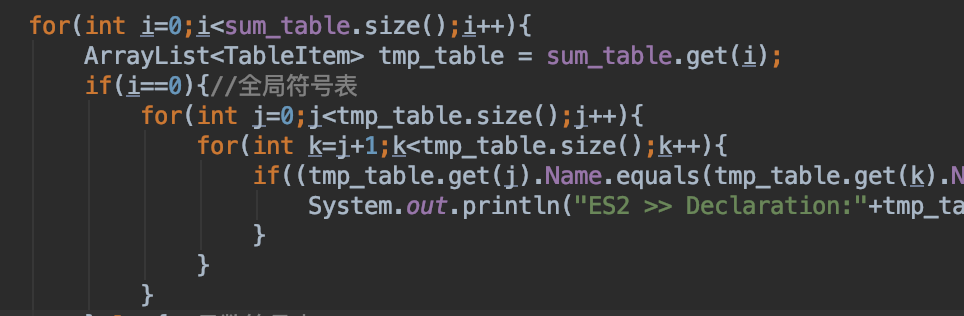
5.1 变量重复定义

由于在访问到变量声明variableDeclarator时，不知道它最终类型是函数声明还是变量声明（返回时才添加函数声明类型），所以在声明完成后再对全局符号表和函数符号表进行遍历，若存在相同的符号表项则为重复定义错误。

5.2 函数重复定义

由于函数定义设计到函数定义和声明（functionDefine和functionDeclarator）的区分问题，与变量重复定义的判别存在区别。

对于函数声明，将其加入全局符号表，kind属性设为functionDeclarator，声明完成后和变量声明一起进行遍历查找重复项。

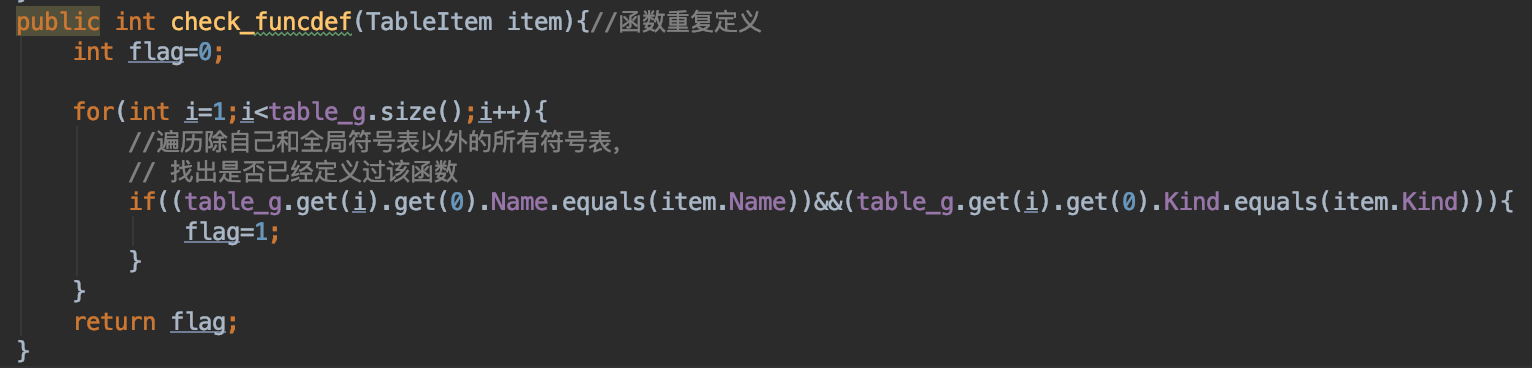


遍历全局符号表，查找重复项



遍历当前符号表，查找重复项

对于函数定义，将其加入全局符号表，并创立该函数的函数符号表，函数符号表第一项为该函数定义，kind属性设为functionDefine，

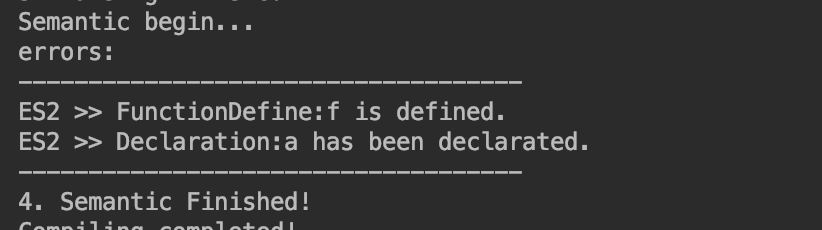


（其实可以只遍历全局符号表，当时写的时候没有把每个函数加入全局符号表，因此遍历了除自己和全局符号表以外的每个符号表）由于每个函数符号表的第一项都是函数本身，因此能够通过遍历所有符号表查找重复项。

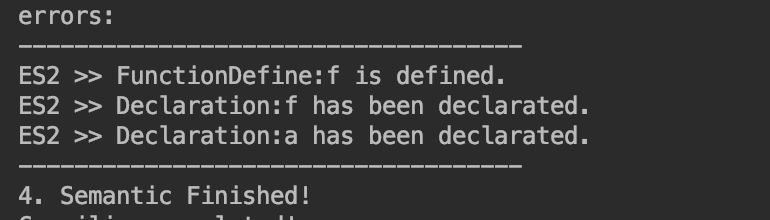
5.3 实验结果

测试代码为1\_var\_defined\_again

1. **int** f();
2. //int f();
3. **int** f(){
4. **return** 1;
5. }
7. **int** f(){
8. **return** 2;
9. }
11. **int** main()
12. {
13. **int** a;
14. **int** a;
16. **int** b;
17. {
18. **int** b;
19. }
20. **return** 0;
21. }



将第二行的int f();取消注释后，则f既有重复定义，也有重复声明。



1. Break语句是否在循环语句中使用

6.1 设置循环体标识变量

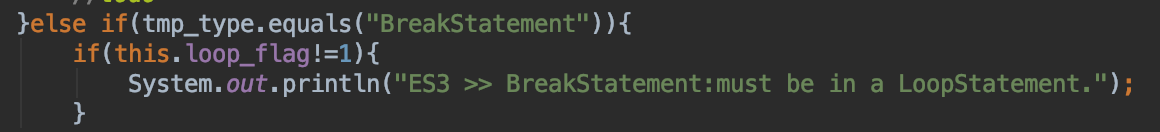
在SemanticVisitor类里设置一个变量loop\_flag,初始值为0，用于标识当前访问到的节点是否在循环体内。

当访问到iterationStatement节点时，令loop\_flag=1;访问结束后再重新置零。



6.2 break语句的访问

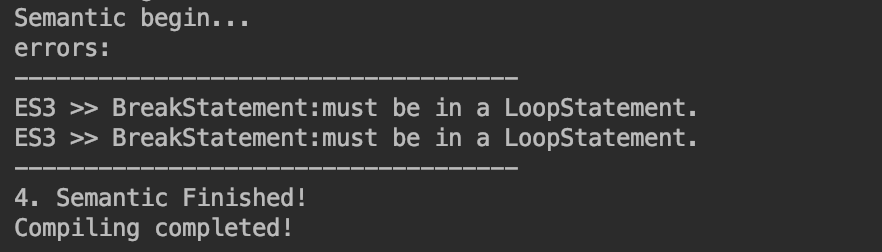
访问到breakStatement节点时，通过判断loop\_flag的值判断当前是否在循环体内。



6.3 实验结果

测试代码为2\_break\_not\_in\_loop.c

1. **int** main()
2. {
3. **break**;
5. **for**(;;){
6. **break**;
7. }
9. **break**;
11. **return** 0;
12. }



7. 函数调用的参数个数和类型是否匹配;

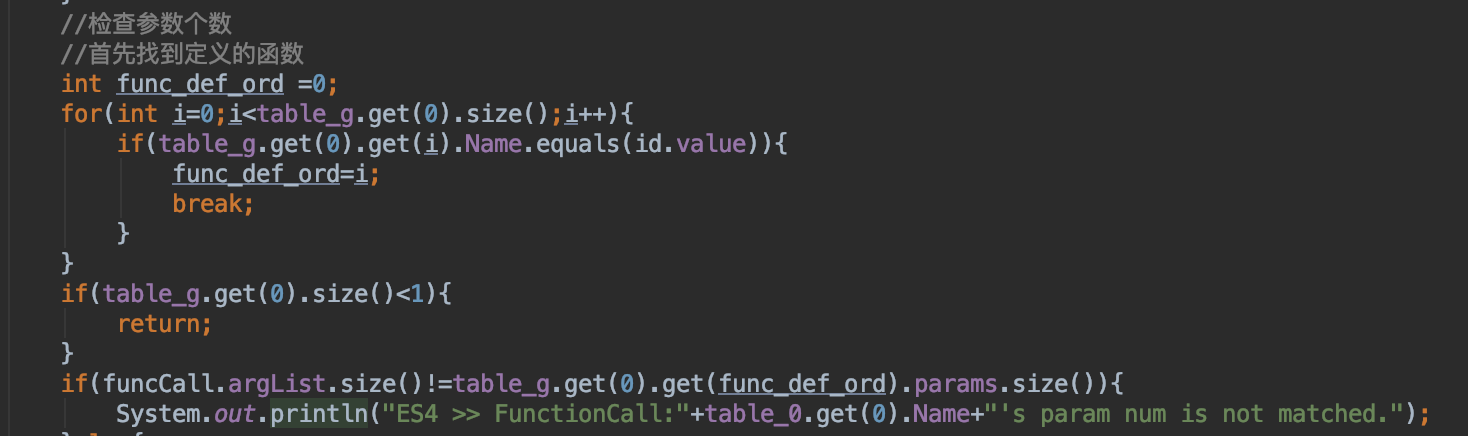
7.1 符号表项中设置参数属性params

详见实验步骤2：符号表建立

7.2 调用时进行参数检查

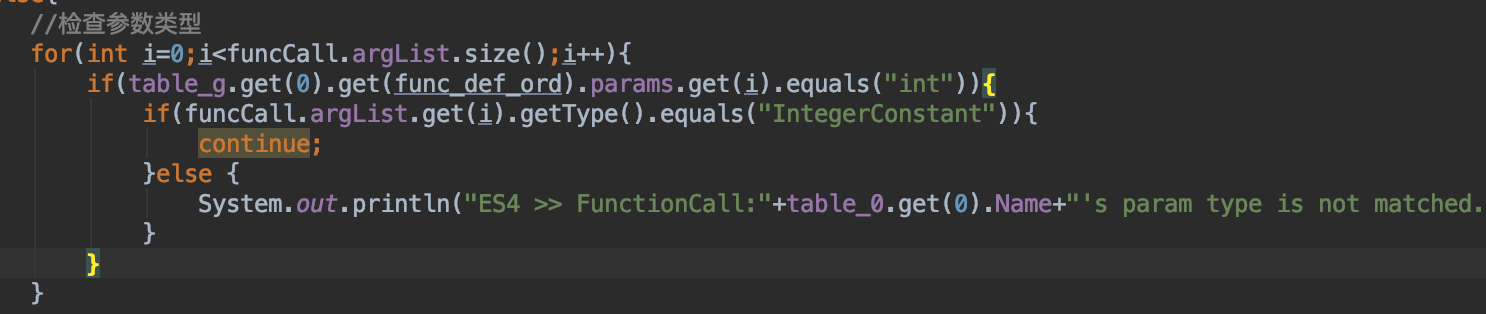
先检查参数个数是否匹配：

在全局符号表中找到该函数的声明，则该项params属性的size（）为参数个数。



再检查参数类型是否匹配：

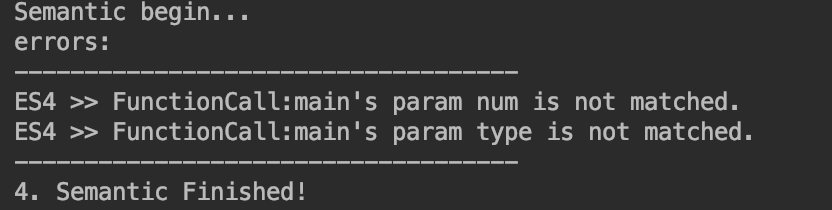
将调用函数所用的参数类型（InteferConstant）和params中的类型(int)进行匹配，不能匹配则报错。



7.3 实验结果

测试代码为3\_func\_arg\_not\_match.c

1. **int** f(**int** a,**int** b){
2. **return** a+b;
3. }
4. **int** main(){
5. **int** res = f(1);
6. res = f(1,0.2);
7. **return** 0;
8. }



# 实验心得体会

这次实验最大的困难应该是老师这次没有给示例代码，从获取节点、节点访问到表项设计、建表等都要自己一步步完成。一开始其实踩了个比较大的坑，没有想到要在visitor类中设置符号表属性，而是通过修改每个节点visit（）函数的调用，把需要用到的符号表传进去。后来发现实在行不通，不仅工作量大、修改表也很困难。于是采用了后来每个visitor实现一个符号表的设计。

遇到的另外一个小麻烦是每次获取到的节点都是父类节点，比如虽然paramsDeclarator.declarator的类型是variableDeclarator，但他accept（visitor）时，调用的还是visit（Declarator）函数，因此每次都有对其类型枚举判断再进行强制转换，比较麻烦，由于时间关系，类型判断也不全面，不能适应更多更复杂的代码。

通过这次实验，对语义分析的流程和方法有了更清楚的认识，对符号表的认识也更具像化了，了解到符号表应具有的信息以及添加表项、查找表项的流程，更加深入的理解了课本中的知识。