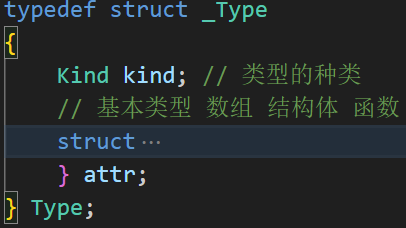
编译原理实验2报告

姓名：朱宸慷 学号：2021110908

1. 程序实现的功能
2. 所使用的数据结构

文本

描述已自动生成参考实验指导书，定义了Type和FieldList结构体，如下图所示

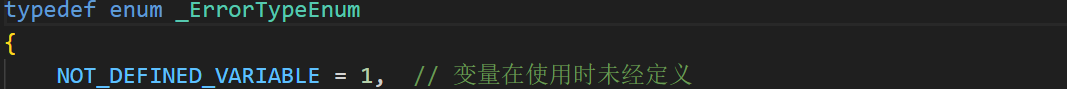
其中Type类型用于描述语义分析中每一个Token的种类以及属性。种类分为基本类型和数组、结构体、函数共四种，每个种类都有对应的属性，例如基本属性有int和float，数组属性有数组元素类型和数组大小，结构体属性有结构体名和结构体域，函数属性有参数个数、参数域以及返回值类型。其中种类以及基本属性通过枚举定义。

作用域FieldList用于解析结构体和函数时，将它们和普通的符号区分开。例如在解析结构体时，结构体成员保存在其type->attr.structure的作用域中，来表明存在归属关系。同理，在解析函数时，也要将相关信息保存在函数的符号表项中。

基于上述两数据结构，可以构建符号表项结构体。符号表项保存了本符号的FieldList和在符号表中具有相同哈希值的下一个符号表项指针，其中该符号的哈希值是符号的name属性通过哈希函数映射的无符号数，符号的name属性保存在该符号的fieldList中。

将符号表保存为符号表项的一维数组哈希表，对表项的索引通过哈希函数，根据传入符号的name寻找下标。符号表中的每个表项通过拉链技术保存，从而可以查找相同哈希值的表项。

1. 定义了错误类型枚举类，建立了错误号和错误名的关系



1. 错误检查的实现

**（错误3，4）变量（包括数组、指针、结构体）或过程名重复声明：**

会保存在符号表中的符号一共有三种，即结构体名、函数名以及普通的变量名。根据实验要求，将结构体成员以及函数的参数也视为普通定义的变量。

首先介绍普通变量的声明：变量的名字最终会在产生式VarDec->ID中终结符ID的字面值中给出。VarDec的祖先节点是ExtDef，该变量的类型在产生式ExtDef -> Specifier ExtDecList SEMI 中由非终结符Specifier给出。因此，将变量的类型递归传入VarDec中，即可获得完整的表项结构体。根据变量名在符号表中寻找该表项，若已经存在相同哈希值的表项，则比较名字，如果名字相同，则产生重复声明；若比较到最后都未产生重复，则将该项插入符号表。

文本

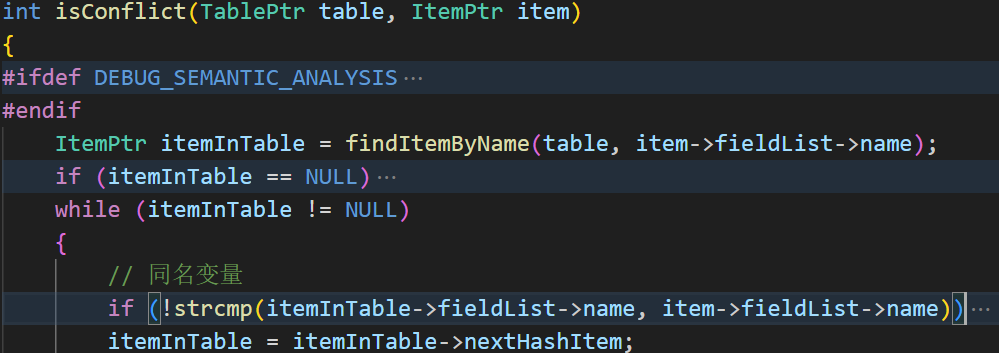
描述已自动生成

在定义结构体和函数变量的时候也遵循类似的思路。结构体定义的入口函数在Specifier中，产生式Specifier -> StructSpecifier -> STRUCT OptTag LC DefList RC，结构体的名字将会在OptTag中给出。类似的，函数的名字在产生式FunDec -> ID LP VarList RP中的ID给出。在声明该变量时，同样按照上文查找是否重复。

**（错误1，2，17）变量（包括数组、指针、结构体）或过程未经声明就使用：**

首先介绍类型检查的方法：在产生式中，终结符ID一定对应着符号表中的一个符号表项，因此，每当产生式遇到一个ID，就可以向符号表询问该ID对应的类型；除此之外，代码中还会有直接写明字面值的常量，对应着词法单元INT和FLOAT类型。

完成上述符号声明后，即可检查变量使用时未经声明的错误。对变量和函数的使用仅会在Exp -> 产生式中出现，其中Exp -> ID是使用普通变量，在此处检查普通变量未声明就使用的问题。当获取到ID的字面值name时，通过封装好的函数isConflict()来查找该name是否已经出现在符号表中，若无，则此处变量未定义，产生报错信息。



对函数的调用是形如Exp->ID LP RP和Exp->ID LP RP的产生式右部，进入该产生式时同样检查ID是否存在符号表中即可。该产生式可能涉及函数参数的传入，对非函数类型使用()运算符等问题，相关错误在下文中讨论。

**（错误5，7）运算分量类型不匹配：**

对于错误5：赋值号两边的表达式类型不匹配，共有两种可能的情况，即初始化赋值和表达式赋值。当一个变量声明时，有产生式Dec -> VarDec ASSIGNOP Exp进行初始化赋值。此时可以通过VarDec获取该变量的类型，并于Exp返回的表达式类型进行对比。根据变量类型的不同，此处处理思路也有不同——若变量是普通类型，则比较两边是否同为整型数或者浮点数；若变量是一个数组，则直接报告该错误。对于表达式赋值，有产生式形如Exp -> ID ASSIGNOP Exp，此时检查ID的类型和Exp返回的类型是否相同，若不同，同样报告该错误。

错误7的处理思路与错误5类似，共有两种，对于形如Exp -> Exp 操作符 Exp的产生式，检查两个Exp返回的类型是否相同即可，除了基本类型不同之外，还有可能出现数组与结构体做运算，这也是不匹配的一种；对于形如Exp -> MINUS Exp和Exp -> NOT Exp的产生式，此处要求不能对函数与结构体使用该类运算符，因此只需要检查右侧Exp返回的类型是否是整型数或浮点数，

**（错误6，12，10，11，13，9，8）操作符与操作数之间的类型不匹配**

错误6在上文赋值检查的同时进行，如果ASSIGNOP左侧的Exp返回了整型或者浮点型就报告该错误。

错误12，10在形如Exp -> Exp LB Exp RB的产生式中检查。该产生式用于访问数组元素，此时要求左侧Exp的类型必须为数组，右侧Exp的类型必须为整型数。若左侧不为数组类型的变量，则报告错误10；若右侧不为整型数，则报告错误12.

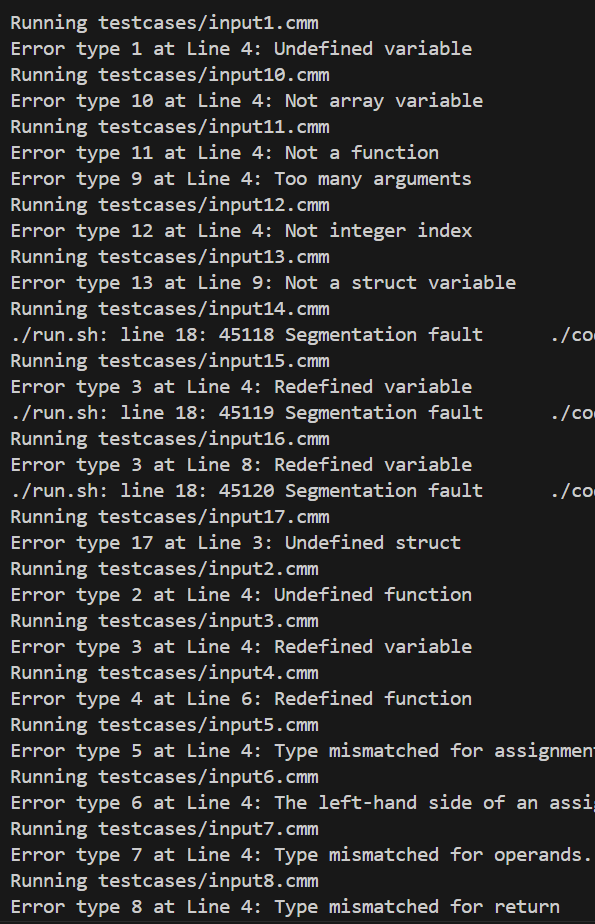
错误11在产生式Exp -> ID LP Args RP中检查，思路同上，只需要检查ID的类型是否是函数。此处产生式中不一定有参数，若有参数，可以通过比较函数的argc字段和Args所提供的参数数量是否相同，检查错误9。

错误13在产生式Exp -> Exp DOT ID中检查，思路同上，检查Exp是否是结构体类型即可。若不是，则报告错误13.

错误8在产生式Stmt -> RETURN Exp SEMI，此处产生式左部来自ExtDef -> Specifier FunDec CompSt，CompSt -> LC DefList StmtList RC。因此，可以由父节点传入函数的返回值类型，在Stmt中，根据函数的返回值类型与Exp提供的类型是否相同来判断，若不同，则报告该错误。

1. 程序的编译方法
2. 自动化执行makefile

通过使用makefile自动执行。在code文件夹下先make clean再运行make即可编译本实验的可执行文件。除此之外，笔者还搭配了一个bash脚本用于自动化执行makefile以及输入测试用例进行测试。运行./run即可在控制台打印所有结果。

1. 结果展示
2. 共完成了14种错误的识别，包括必做错误类型：1，3，5，6，7，10，12和未要求的2，4，8，9，11，13，17
3. 其中错误类型14、15、16因为结构体指针赋值问题产生段错误，笔者在实验时间内暂时无法解决。