编译原理实验——语义分析

实验报告

171860667 袁想　　171860664 谢鹏飞

一、实现功能

(1)根据假设 1-7 实现语义分析并识别要求的 17 种错误

(2)将结构体间的等价机制由名等价改为结构等价(要求 2.3)，完成 17 种错误检查。

二、重要数据结构

(1) 表示类型的结构 TypeInfo,定义在 semantic.h 中

```
typedef enum {BASIC, ARRAY, STRUCT, FUNC, ERROR} Kind;

typedef struct TypeInfo
{
    Kind kind;
    union
    {
        int basic; //BASIC
        struct
        {
            struct TypeInfo *element;
            int size;
        } array; //ARRAY
        struct FieldList *structure; //STRUCT
        struct Function *function; //FUNCTION
        int errortype; //ERROR
    } info;
} TypeInfo;
```

枚举类型 Kind 指明一个变量的具体类型(BASIC 为基本类型 int 或 float，ARRAY 为数组，STRUCT 为结构体，FUNC 为函数，ERROR 为错误，用于在向上层语句返回类型时如果当前类型是一个错误的类型（例如变量未定义，操作符不匹配等），用该类型告知上层语句这里出现错误)，根据 kind 取值决定访问联合体 info 中的哪个部分。

(2) 表示结构体中域的结构 FieldList,定义在 semantic.h 中

```
typedef struct FieldList
{
    char name[33];
    int linenumber;
    struct TypeInfo *typeinfo;
    struct FieldList *next;
} FieldList;
```

该结构中包括域的名字，域所在的行号（用于报错），域的类型信息以及指向结构体中定义的下一个域的指针。

(3) 表示函数的结构 Function,定义在 semantic.h 中

```
typedef struct Function
{
    int paranum;
    struct ParameterList *paralist;
    struct TypeInfo *functype;
} Function;

typedef struct ParameterList
{
    char name[33];
    int linenumber;
    struct TypeInfo *typeinfo;
    struct ParameterList *next;
} ParameterList;
```

函数结构体中有 paranum 表示函数参数的个数，paralist 是指向函数参数表的指针，paralist 中每一个参数是一个 ParamaterList 结构体，将存储参数的名字，定义所在的行号(报

错时使用），参数的类型信息以及指向函数参数表中下一个定义的参数信息的指针（如果后面没有了就是空指针）。functype 存储函数返回类型信息（由函数定义的 Specifier 获得）。

(4) 符号表结构 HashNode，定义在 semantic.h 中

```
typedef struct HashNode
{
    char name[33];
    struct TypeInfo *typeinfo;
    struct HashNode *next;
} HashNode;
```

该结构用来存放定义的变量、数组、结构体和函数的信息。每一个符号表中的符号结构将存放符号的名字 name，符号具体类型信息 typeinfo 和指向下一个符号节点的指针 next。符号表采用散列表，散列函数使用实验指导所给的参考函数，处理散列表冲突的办法为 open hashing，即在相应数组元素下面挂一个链表，通过头插入的方式插入到对应的数组元素链表中。符号表的定义如下：

```
struct HashNode* symboltable[HASH_NUMBER+1];
```

三、实验设计思路与实现

(1) 语义分析代码框架

为每个非终结符创建一个函数，将语法树中对应的节点作为参数传入相应函数，一些节点函数也会返回信息给父节点，以实现符号表的构建和错误检查。

```
void semantic_analysis(TreeNode *root);
void extdeflist(TreeNode *p);
void extdef(TreeNode *p);
```

程序通过 semantic_analysis 函数进入语义分析，相当于语法树中的 program 节点，按照文法产生式推导的规则从语法树中依次向下调用相应函数。例如 extdeflist 函数将传入的节点 p 的第一个子节点 p->children[0]作为参数调用 extdef，将 p 的第二个子节点 p->children[1]作为参数调用 extdeflist，即对应产生式 ExtDefList -> ExtDef ExtDefList。如果 p 为空，说明是 ExtDefList 推出空串，那么函数直接返回。

一些有关变量或函数类型声明和定义的产生式的处理函数（例如 Specifier 的处理函数，结构体定义中的 DefList 产生式的处理函数将处理后得到的类型信息或者域的信息作为相应的结构体指针返回上一层调用该产生式处理的函数，进而将这些信息传递给后面需要用到这些信息的处理函数中。例如 Specifier 处理函数如下：

```
TypeInfo* specifier(TreeNode *p);
```

在后面例如处理 FunDec 时就会用到这个类型信息，将其传入 FunDec 处理函数，因此 FunDec 函数将会接受一个指向类型信息的指针：

```
void fundec(TreeNode *p, TypeInfo *type);
```

(2) 符号表构建

首先初始化散列表，定义散列表大小为 0x3fff+1，将其中所有指针初始化为 NULL。之后在处理非终结符节点的函数中如果遇到变量或者函数的定义信息，如在 FunDec 中遇到 ID LP VarList RP 或者 ID LP RP，都是函数定义，此时根据 ID 查表，如果在散列表中查到该名称所对应的变量或函数，说明函数名重复，报错，如果没有查到，则根据调用 FunDec 前先调用的 Specifer 函数返回并传入 FunDec 处理函数的 TypeInfo*指针（即相应函数的类型信息）以及如果有 VarList，根据调用 VarList 处理函数之后返回的参数表指针，来构建一个新的散列节点 node 并将其插入散列表。

(3) 错误检查

错误类型 3、4、15、16、17 会出现在定义变量、结构体、函数定义的开头部分（包括函数的返回类型，函数名字，参数表）以及结构体中域的定义的这些定义部分。其余的错误类型会出现在函数的函数体部分，即大括号中间的语句部分。

错误 1、2、3、4、14、15、16、17 均可以通过查符号表的方式解决，根据当前分析到的语法树中的节点处理函数中获取的变量名或者函数名使用辅助工具函数 checksymbol 函数进行查表，如果是查询相关函数名，则第二个参数为 1，否则为 0。(因为结构体、数组和基本类型变量是不允许互相重名的，因此这三种可以放在一起判断是否重名)。该函数返回 1 表示查到该符号，返回 0 表示没有查到：

```
int checksymbol(char *name, int isfunc);
```

定义部分出现的错误将会通过在插入符号表之前进行上面的操作判断是否已经存在该符号名称，有就报重名错误，没有就插入该符号信息。对于结构体内部的域名，将会对每个域名进行检查，特别是一个语句可能定义多个域（例如 int i,j,k;）将通过结构体中的 Def 处理返回的 FieldList* 遍历进行重名检查。

函数体（语句部分）的错误将会在 Exp 产生式处理中处理。针对 Exp 的产生式划分了多个处理函数。如果在 Exp 中调用变量、函数或结构体时没有查到表中对应项，则报错误 1、2、14；14 即遍历结构体中的域名，没有与当前访问相同的就报错。如果查找到但使用方法不对，判断方法就是通过处理 Exp 函数返回的对应变量类型是否为期望的类型，比如对非结构体类型的变量使用“.”操作符，对非数组使用“[]”，将报出相应错误。其中值得一提的是错误 6，赋值号左边的操作数不是左值的情况。我们在语法树结构 TreeNode 中添加一项 int isleftexp 表示该节点是否为左值，当 Exp->ID,Exp->Exp DOT ID,Exp->Exp LB Exp RB 时，分别为访问变量，访问结构体域和访问数组时，将传入的语法树节点 p（对应上面三个产生式左边的 Exp)中的 isleftexp 置为 1 表示为左值，其余情况均置 0 表示不是左值。在处理 Exp->Exp ASSIGNOP Exp 的函数 expassignop 中判断传入的参数节点 p 的第一个子节点，即 ASSIGNOP 左侧的 Exp 的 isleftexp 是否为 1 即可，不为 1 则报错误 6。同时，每个处理 Exp 产生式的函数都会返回一个类型。如果没有语义错误将会返回对应类型，例如 Exp RELOP Exp 将会返回一个 type 为 BASIC 的 int 类型。如果出现语义错误，将会返回 type 为 ERROR 的存储错误号的出错类型，告知上层语句这里出现了错误。

(4) 将结构体间的等价改为结构等价(要求 2.3)

在 Exp 相关函数中，如果一个表达式涉及两个 Exp，如 Exp->Exp ASSIGNOP Exp 则需要判断这两个 Exp 的类型是否相同，此时调用自己编写的 isEqualType 函数判断。该函数中判断传入的两个 type 是否相同，返回 1 表示相同，返回 0 表示不同。对于 BASIC，看 info 中的 basic 是否一样；对于 ARRAY，对两个 ARRAY 的 array 中的 element 继续调用 isEqualType 判断；对于结构体，则通过遍历两个结构体的 FieldList 依次判断这两个结构体中的成员是否类型相同（用 isEqualType)。如果当前判断的两个域类型不同，或者此时一个结构体还有域，另外一个已经遍历结束（当前访问的指向域的指针为空），那么返回 0。没有触发返回 0 就返回 1，表示域的类型全部相同。对于函数，判断其返回类型 functype 是否相同。

四、编译说明

利用给定的 Makefile 文件，将其置于源码的相同路径下，在终端执行 make 即可。或者依次输入如下指令：flex lexical.l，bison -d -v syntax.y gcc main.c，syntax.tab.c tree.c semantic.c -lfl -ly -o parser。