语义分析实验报告

151220125 吴佳玮

邮箱：498279281@qq.com

一、 完成功能

实现了对C—语言的语义分析及错误检测，能够检查17种错误，实现了2.2的要求，支持变量作用域，内部变量屏蔽外部变量，作用域结束变量消亡

二、 实验环境

实验编写在 MacOS 系统下，使用flex 2.5.35和bison 2.3编译通过并运行。提交之前在ubuntu 16.04环境下编译通过并测试成功

//macOS下编译需要-ll参数而非-lfl参数，在提交前已将Makefile恢复为-lfl

三、 实现解释

1、类型表示

```
struct Type_
{
  enum { BASIC, ARRAY, STRUCTURE } kind;
  union
  {
    // 基本类型 1为int 0为float
    int basic;
    // 数组类型信息包括元素类型与数组大小构成
    struct { Type* elem; int size; } array;
    // 结构体类型信息是一个链表
    struct {char *name; FieldList *fl;} structure;
  } u;
};

struct FieldList_
{
  char* name; // 域的名字
  Type* type; // 域的类型
  FieldList* tail; // 下一个域
};
```

Type的结构体基本使用参考书中的结构，只有在structure域中添加了char *name来应对结构体名等价的要求。此外typedef没有将struct Type_ *定义为新的类型，因为害怕忘记指针类型，所以定义时还是采用Type *形式。

2、符号表

```
struct symbol_ {
    int depth;
    char name[32];
    enum{Func,Variable,Struct} kind;
    int lineNum;
    FuncMessage* funcMessage;
    Type* variableMessage;
    //哈希表中解决哈希冲突的链表指针
    Symbol* hashLast;
    Symbol* hashNext;
    //维护变量每一层作用域的链表指针
    Symbol* stackNext;
};

struct FuncMessage_
{
    Type* returnType;//返回值类型
    FieldList* input;//参数
};

struct stacklist {
  Symbol* firstSymbol;
  StackList *next;
};
```

因为要完成要求2.2，所以采用参考书中的十字链表结构，横向hash表用来查询，纵向栈链接用来控制作用域。其中funcMessage用来存储函数相关信息（参数，返回值），variableMessage存储的是变量和结构体定义相关的信息

符号表对外开放的接口有且只有

stackAdd();

压栈，进入新的深度

stackDelete();

退栈，删除所有原深度符号

stAdd(…);

在符号表中当前深度添加符号

stSearch(…);

在符号表中搜索符号

当前深度以全局变量currentDepth管理

hash值计算为：名字中各个字符ascii码相加再mod 256

语义分析

在lab1产生的语法树上重新遍历进行语法分析，从根节点开始分辨子节点类型并调用对应函数进行分析。

特殊思路

1） 在CompSt函数中只有退栈没有压栈，因为每次压栈都在CompSt更高一级实现，因为对于函数定义，形参也应定义在新的深度中，而函数符号本身应该定义在较浅深度。

2） 结构体定义时仍然采用压栈退栈的方式来检测同一结构体中域名重复问题

3） 每次出现语义错误时有返回值得函数会返回NULL，进而导致同一行的一些其他错误发生（比如 x = 1,x未定义导致返回Type为NULL，同时导致了赋值语句左右类型不一致的错误），每次选择将错误都打印出来。

4） 对于结构体中域的定义，会有选择的舍弃返回NULL的子节点并将其他的串联在一起，防止结构体的域链表被某个错误打断。（如 struct a{ int i; int i,j; }; a接收的域链表应包含i和j，不应该由于第二个i重定义导致j被舍弃）

5） 左值相关错误从语法角度检查