编译原理实验报告二

151220135 许丽军 xulj.cs@gmail.com

零、实验进度描述

完成了所有的必做内容和选做要求**2.1 2.3**

一、实验内容

对不存在词法错误和语法错误的C--源代码,进行语义分析和类型检查,并打印所有的语义错误

二、实现细节

符号表

- 符号表采用散列表+open hashing方式
- hash函数采用hash_pjw()
- 将变量、结构体和函数都组织到同一张表中
- 符号表的表项的数据结构定义如下：

```
typedef struct symNode{
    const char *name;
    enum{Variable,Structure,Function} kind;
    union{
        Type type;      //for variable
        FieldList field;    //for struct definition
        ParaList_Ret para_ret;  // for function definition
    }info;
    int lineno ;    // for function : if defined then lines=0 ; else declartion
    struct symNode* tail;
}symNode;
```

- 域lineno用来记录未定义的函数的最后一次声明的行号（故为简化实现，错误类型18的行号只会显示最后一次的行号）；若已经声明，则清零
- 相关函数接口：
  - 初始化符号表
  - 查表
  - 查表（在函数定义和声明时检查是否存在不一致）
  - 填表
  - 检查表（检查是否存在没有定义的函数声明）

```
void initSymTable();
bool searchSymTable(const char *,int,void **,int);
int searchSymTable_f(const char *,ParaList_Ret,int);
int insertSymTable(const char *,int ,void *,int);
void checkSymTable();
```

语义分析

- 实现思路
  从ExtDef开始，为语法树的每一个非叶子节点定义一个对应的函数，用来对以此叶子节点为根的子树进行分析，并调用子节点对应的函数.
  通过函数的返回值和参数来传递节点的综合属性、继承属性以及其他相关信息.
- DefList的不同出现位置
  语法单元DefList可能出现在 ***structSpecifier -> struct OpTag { DefList }*** 以及 ***CompSt -> { DefList }***

1. 当出现在前者时，DefList中出现的标识符ID为结构体的域，不需要作为变量记录在符号表中（虽然假设7中允许作为变量记录在符号表中；
2. 当出现在后者时，DefList中出现的标志符ID为局部变量，需要作为变量记录在符号表中
   本实验代码中根据调用者的不同，通过flag区分开了这两种情况，即即使违反假设7的源代码也可以正确分析

```
static void DefList(Node *p,FieldList head,int flag)
```

> 假设7：结构体中的域不与变量重名，并且不同结构体中的域互不重名。

类型检查

- 定义类型的数据结构如下

```
typedef struct Type_* Type;
typedef struct FieldList_* FieldList;

struct Type_{
    enum {BASIC,ARRAY,STRUCTURE} kind;
    union{
        enum {INT,FLOAT} basic;
        struct {
            Type elem;
            int size;
        }array;
        FieldList structure;
    };
};

struct FieldList_{
    const char *name;
    Type type;
    FieldList tail;
};
typedef FieldList ParaList_Ret;
```

- 函数的返回值和参数通过复用结构体的FieldList记录在符号表中,其中第一个域的名字定为"return"
- 类型检查相关函数接口（用于Stmt中检查类型问题）

```
bool typeEq(Type,Type);
bool fieldEq(FieldList,FieldList);
bool isInt(Type);
bool isFloat(Type);
bool isStruct(Type);
Type isField(FieldList,const char *);
Type isArray(Type);
```

三、编译和运行

- 编译并生成可执行目标文件：make
- 运行可执行目标文件： ./parser your_file_name 或 make test [TESTFILE = your_file_name]
- 以pretest/目录下的所有文件为输入，批量运行： make test-all
- 清除中间生成文件：make clean-temp
- 清除所有生成文件: make clean