编译原理实验儿：语义分析

实验目标

在实验一词法和语法分析的基础上，利用实验一构造的语法树进行语义分析

实验步骤

- 对语法树进行遍历
- 在遍历的过程中构造符号表
- 在不同的语法单元节点进行不同的语义检查操作
- 输出语义错误信息

遍历语法树

```
static void ExtDefList(Node node);
static void ExtDef(Node node);
static void ExtDecList(Node node, Type type, List *head);
static Type FunDec(Node node, Type returnType);
static void CompSt(Node node, Type returnType);
static void StmtList(Node node, Type returnType);
static void Stmt(Node node, Type returnType);
static void VarList(Node node, List *head);
static void ParamDec(Node node, List *head);
static Type Specifier(Node node);
static Type StructSpecifier(Node node);
static void DefList(Node node, List *head, int structFlag);
static void Def(Node node, List *head, int structFlag);
static void DecList(Node node, Type type, List *head, int
structFlag);
static void Dec(Node node, Type type, List *head, int structFlag);
static void VarDec(Node node, Type type, List *head, int
structFlag);
static Type Exp(Node node);
static void Args(Node node, List *head);
```

对于每种语法节点定义处理函数

构造符号表

```
struct Type_ {
    enum { BASIC, ARRAY, STRUCTURE, FUNCTION } kind;
    union {
        enum { INT, FLOAT } basic;
        struct { Type elem; int size; } array;
        struct { char *name; List domain; } structure;
        struct { char *name; int line; Type type; List param; }
function;
    } info;
    enum { LEFT, RIGHT, BOTH } assign;
};
```

结构体 `Type_` 表示符号类型，包含基础类型、数组、结构体、函数，并根据不同类型记录信息用于检查语义错误。

实现了类型等价检查函数。

还定义了链表结构体用于储存符号，实现了链表操作相关函数。

由于我的选做实验要求实现符号的作用域，我这里选择使用栈的方式实现：

```
struct Stack_ {
    List list;
    Stack next;
};
```

定义了栈的结构体，栈中元素是一个链表表示一层作用域里的符号。

语义检查

分别对17中语义错误进行检查，包括类型不匹配，重定义等

```
...
static void ExtDef(Node node) {
    Type type = Specifier(getChild(node, 0));
```

```
    Node second = getChild(node, 1);
    if (strcmp(second->nodeName, "ExtDecList") == 0) { // 变量声明
        List head = NULL;
        ExtDecList(second, type, &head);
    } else if (strcmp(second->nodeName, "FunDec") == 0) { // 函数定
义
        push_stack(&symbolTable);
        Type func = FunDec(second, type);
        pop_stack(&symbolTable);
        if (find_in_stack(symbolTable, func->info.function.name)
!= NULL) {
            // !ERROR
            printError(4, node->lineNum, "Redefined function.");
        }
        insert_in_top(symbolTable, func->info.function.name,
func);
        push_stack(&symbolTable);
        FunDec(second, type);
        CompSt(getChild(node, 2), type);
        pop_stack(&symbolTable);
    } else if (strcmp(second->nodeName, "SEMI") == 0) { // 仅类型声
明
        return;
    }
}
...
```

总结

在实验的过程中，我遇到过因为考虑不充分而产生的bug。

比如在赋值语句中，当函数调用在赋值号左边时，Exp应返回函数类型，从而产生报错；但当函数出现在赋值号右边时，应该返回函数的返回值类型用于表达式求值。

还有对于某些可能为空的语法节点，要在遍历之前进行检查，不然容易出现对空指针解引用引发的程序崩溃问题。