编译原理实验二实验报告

191220163 计算机科学与技术系 张木子苗

完成的实验要求

在本次实验中，我完成了**全部的必做任务**，代码**可以通过所有的必做内容样例。**没有完成选做内容。

提供的测试方法

在Test目录下，我一共准备了17个代码文件用于测试：

`test1.cmm` 到 `test17.cmm`，对应实验指导 pdf 里的样例1-17

为了方便测试，我为我的代码写了个简单的脚本文件，`Code/Test.sh`，在**Code目录下命令行界面输入**：

```
./Test.sh
```

即可自动编译并且执行对上述17个测试用例的语义分析，打印分析结果。

同时，为了方便助教测试其它用例，我为还提供了另外一个脚本文件，
`Code/Test_one_testcase.sh`，**将新的测试用例放在Test目录下，在Code目录下命令行界面输入**：

```
./Test_one_testcase.sh the_name_of_testfile.cmm
```

即可自动编译并且执行对该测试用例的语义分析，打印分析结果。

代码的实现思路

数据结构

对于类型和链表的实现采用的是实验指导中推荐的结构体 `Type_` 和 `FieldList_`

对于符号表项，其结构体如下：

```
struct Item_
{
    char item_name[100];
    Type item_type;
    Item next_item;
};
```

写完后才发现这个结构体实际上和 `FieldList_` 几乎相同，不过因为难以全部修改还是保留了，也方便区分 `FieldList` 和符号表项 `Item`。

符号表用链表组织，一共有两个全局的符号表：

```
Item Symbol_Table = NULL;
Item Struct_Symbol_Table = NULL;
```

其中 `Struct_Symbol_Table` 是单独结构体内部的符号表，在分析每个新的结构体定义时把旧结构体符号表制空。该符号表用于判断结构体中的域是否重复定义。

实现思路

通过遍历语法分析树，进行语法分析。

语法分析树的每一个节点对应一个终结符或非终结符，在之前的处理中，语法分析树用的是孩子兄弟表示法，并且在每个节点上记录了这个文法符号的类型 `token_type`（如 ID、Def、Stmt 等）。

对于每个重要的非终结符，为其专门构造一个分析函数，将该非终结符对应的节点和其他需要的参数传给该分析函数，在函数内根据我们的文法对其进行分析，并且返回在后续分析中需要使用的信息给调用者，例如：

```
/*Handle the definition of top-level functions, structures, and global variables.
(处理最高层函数、结构体、全局变量的定义)*/
void ExtDef(struct Token* node);
/*Parse the Specifiers and return its type (basic or struct).
(解析类型描述符，返回其类型(basic or struct))*/
Type Specifier(struct Token* node);
/*Parse the definition of a single variable (basic or array) and return its corresponding item in the symbol table.
(解析对单个变量的定义(basic or array)，返回其在符号表中对应的项)*/
Item VarDec(struct Token* node, Type type);
```

此处只展示部分函数声明，具体实现细节请查看 `semantic.h` 与 `semantic.c`，注意查看 `semantic.h` 中对函数声明的注释和 `semantic.c` 中对代码对应的产生式的注释，如：

```
/*
    Specifier → StructSpecifier
    StructSpecifier -> STRUCT OptTag LC DefList RC
    StructSpecifier -> STRUCT Tag
    OptTag -> ID | kong
    Tag -> ID
*/
specifier_type->kind = STRUCTURE;
struct Token* struct_specifier = node->firstChild;
if(strcmp(struct_specifier->firstChild->nextSibling->token_type,"Tag")==0)
```

对于报错结果的说明

有的报错可能会导致别的报错的产生。目前发现的是：错误类型1 “变量未定义” 和错误类型14 “结构体内域未定义" 可能会额外导致 错误类型5 ”赋值类型不匹配“ 和 错误类型7 ”操作数不匹配“。

但是可以确保核心的错误都可以找到，并且最早被输出。