编译原理实验阶段 2 实验报告

一、实验进展

这个阶段的实验我的项目实现了语义分析的功能，能对实验教程中的前 17 种错误类型进行检查和输出错误，在选做内容方面，代码实现了 2.2，在前 17 种错误类型中均能如测试样例所示输出错误检查结果。在代码中这部分内容的实现主要体现在使用了 symbol.c 文件构造了符号表、类型表以及由类型的域形成的链表的结构，并在其中实现了基于这些表的功能函数。然后在 c_analyse.c 中实现对代码的语义分析功能，通过嵌套的函数和文法从 Program 开始在语法树上从上往下分析，最后再 main.c 中调用结构的初始化和 AnalyseProgram 函数来实现语义分析功能。

二、编译方法

和阶段 1 一样先输入命令

bison -d syntax.y

和

flex lexical.l

然后输入命令

gcc main.c common.c syntax.tab.c syntax_tree.c c_analyse.c symbol.c -lfl -ly -o parser

生成语法分析器 parser，最后

./parser (文件名)

即可使用 parser 对文件进行错误检查。

三、实验思考

在 analyse 函数的实现过程中，由于语言文法中每一个元素的子元素的种类很多，所以当时实现时经常发生错误，比如 StructSpecifier 后跟的子树中有 OptTag 这个元素，这个 OptTag 作为第二个子结点而可能为空集，当时我没有注意到它可能为空的可能性而直接默认将第二个子结点作为 OptTag 的分析，第四个子结点作为 DefList 的分析，于是就发生了错误，后面我为了能减少函数对子结点的操作并纠正这个错误，我使用了倒数第二个子结点作为 DefList 就能规避这个问题。

还有在 TYPE 类型结构中包含了域表，而域表中也有 TYPE 类型

```
struct FieldList_{
        char *name;
        TYPE *type;
        struct FieldList_ *next, *prev;
};

typedef struct TYPE_{
        TypeKind kind;
        union{
                int basic;
                struct{
                        struct TYPE *elem;
                        int size;
                }array;
                FieldList structure;
        };
};
```

一开始我觉得这没有什么问题，后来才发现这个情况下不管先定义谁都不对，这时我请教了同学知道了可以现在文件头部将这两个结构先 typedef 了。就不会有先后定义的问题了。

这个阶段的实验中的错误信息输出要求将一些不是终结符号的语法树节点的内容也输出出来，这个部分我在阶段 1 时没有进行处理，导致一开始时这样的输出会输出 NULL，于是我修改了 syntax.y 的内容，让它在构建语法树时将子结点中包含的终结符号的内容也传输给父节点。让类似于 exp 的节点也能记录它的子结点的信息。