编译原理 project1 实验报告

一、实验完成内容

在实验一中，我完成了 flex 要运行文件 lexical.l 和 bison 运行的文件 syntax.y，并将它们的使用联系起来，让 bison 的运行包含 flex，并在其中的代码中实现了 flex 文件对输入文件中的内容读入并识别为终结符号，然后将这些终结符号返回给 bison 文件的功能，以及 bison 中通过对文件的扫描结合返回的终结符号构造出语法树从而对文件的正确性进行判断的功能，同时还能在文件无错误时按照语法树遍历顺序输出的功能。在实验一的教学文档中给出了可识别出的两种错误类型，我将这两种错误类型的识别均实现了出来，对所有的样例均能测试通过，但是还没有实现 error 变量的放置操作，导致大部分有多个错误的程序文件只能检测出一个错误，这个功能我会接下来去完善它。在检测出错误类型 A 时，我的代码能如教程中所示输出错误类型 A 的错误信息以及出错行号，错误类型 B 也是如此。

二、程序的编译运行方式

我提交的项目文件夹中包含若干个源代码文件，在 linux 中进入项目目录并依次执行

```
flex lexical.l
bison -d syntax.y
gcc main.c syntax.tab.c syntax_tree.c common.c -lfl -ly -o parser
```

可编译得到可运行文件 parser，然后用 parser 对需要进行语法分析的文件 test.c 执行

```
./parser test.c
```

即可对 test.c 进行语法分析并根据检测结果输出错误信息或语法树构造。

三、实验心得

在一开始编写 lexical.l 的规则部分的代码时，我没有留意到规则部分的优先级是越在上面的代码最先会被执行，因此我直接就把 int，float，id 的定义规则放在了规则部分的最上方，这样做会导致后面的 type，struct 等终结符号类型都被识别为 id，因此当时编译代码时会出现某某某行代码定义了无效的规则的 warning，当时没有在意这个 warning 直到最终编译的时候发现这个错误并找了很久。

在编写 syntax.y 文件时，由于在规则部分对每一个产生式都要实现构造语法树的结点并确定结点子结点与父结点的关系，所以我一开始是打算在每一种产生式后面的动作中一个一个地实现这些操作，然后我就发现这样的代码写起来太繁琐了，但是如果定义函数代替操作我又发现产生式后面部分作为函数参数的量又有很多种类，也就是说我得写很多种函数，这时我上网找到了...参数的用法，才将本来应该很复杂的代码简化，只定义了一个函数就实现了所有动作的替代。最终实现出来的动作函数如下

```
#define operate(token, head, location, num, ...) \
do { \
        head = asd(location.first_line, num, __VA_ARGS__); \
        (*head).name = toArray(#token); \
}while(0)
Treenode *asd(int location, int num, ...){
        va_list valist;
        int i;
        va_start(valist, num);
        Treenode *p = newnode();
        int flag = 0;
        for (i = 0; i < num; i++){
                Treenode *q = va_arg(valist, Treenode*);
                if(q != NULL && flag == 0) {
                        (*p).lineno = (*q).lineno;
                        flag = 1;
                }
                if(q != NULL) treeAddChild(p, q);
        }
        va_end(valist);
        return p;
}
```

关于语法树结构的实现我也实现了很久，主要是一开始没有想着语法树的结点要实现一个链表结构的子结点结构，而是想着实现一个数组结构来保存子结点，然而编译时的内存溢出让我只好重新实现语法树结构的构造将子结点用链表来保存。在语法树的输出中，由于要判断当前结点的符号类型再决定如何输出，所以我又回到了语法树结点的定义地方加入了对于结点的记录，并在 lexical.l 文件的相应类型的读取中加入了对结点属性的记录，然后还加入了对终结符号结点也就是叶结点的数值记录。在 print 的实现上我也反复更改了很多次。