report

学号：211220133

姓名：李睿

程序功能

- 完成了必做部分。
- 完成了选做2.3，即识别单行注释和跨行注释并去除。

程序实现概要

抽象语法树结点

用枚举类型表示结点的类型，由于`syntax.tab.h`中定义过一些结点的枚举类型，所以`ast.h`中的枚举类型前面都会有一个`_`用于区分。

由于抽象语法树需要存储不同类型结点的不同信息，为了节省空间，用`union`存储不同类型的数据，通过结点的类型来取得有意义的属性值。

语法树的打印

在`C99`中，枚举类型可以转换为`int`类型，因此用`nametable`数组来映射每个类型的名字，方便后续的打印操作。

打印的实现就是简单的递归。

抽象语法树的构建

`ast.h`向外提供了`intNode`,`floatNode`,`relNode`,`idNode`,`normalNode`接口，分别用于构建不同类型的抽象语法树结点。以上每种结点实际上"继承"自虚节点，都会调用一个`newNode`来生成一个"基类"，填写一些必要的字段。不过这里出现了一些小小的问题，在使用`yylloc.first_line`时，会出现行数不正确的情况。后面用`@$.first_line`，得到了正确的结果。

`ast.h`还向外提供了`addchild`接口，用于向已有的结点添加子节点。在`syntax.y`中，便是用`$$=xxNode();addchild($$,$1);addchild...`的形式进行抽象语法树的构建的。

token的属性类型

token的属性类型就是`void*`，即不同类型的抽象语法树结点。以上的操作可以帮助我们构造非终结符结点。那么终结符结点就需要在`lexical.l`中进行构建。直接使用正则表达式匹配对应的词法单元，但是`TYPE`,`ID`,`INT`,`FLOAT`,`GT`,`GE`,`LT`,`LE`,`EQ`,`NE`等符号有特殊的语义，因此需要调用对应的抽象语法树结点的对应接口进行构建。

程序编译

本阶段没有更改`Makefile`，直接使用`make parser`命令即可编译。