<https://github.com/rsy56640/Assignment-in-WHUISS/tree/master/Principle_of_Compiling/2nd>

作者：任思远 2016302580320 卓越二班

刘瑞康 2016302580242 卓越二班

工作分配：

任思远：负责整体流程，

主要实现： separate\_one\_line()；parsing()主要部分；

刘瑞康：辅助整体流程实现，测试和完善实现细节

主要实现： parse\_cmd()； preprocess()；parsing() –json部分；

源码位于Yamlite/src/文件夹下，使用cmake-gui.exe进行生成，generator选项为 Visual Studio 14 2015，生成结果为Yamlite.sln。

测试运行均在Yamlite/test/文件夹下进行。

为了便于测试，在代码中加入了display部分，即输出有效行的信息，并且输出所有的 key-value信息，key为value所对应的的目录（含嵌套）。

实现部分

整体思路：解析命令，逐行扫描，然后再具体进行行内分析，分析结束后根据命令进行对应的查找判断或输出操作。

刘瑞康：

1. 命令行解析：

\_STD tuple<YAMLlite::option, const char\*, const char\*> parse\_cmd(int argc, char \*argv[])

功能：

检查并根据命令行参数数量和参数具体内容确定要进行的操作，若参数个数非法或内容非法，输出错误类型，终止程序

检查文件名是否包含非法字符 { '/', ':', '\*', '?', '"', '<', '>', '|' }

检查文件名后缀是否符合要求 (.yml)

解析完成后返回包含必要信息的tuple以供下一步处理

1. 预处理：

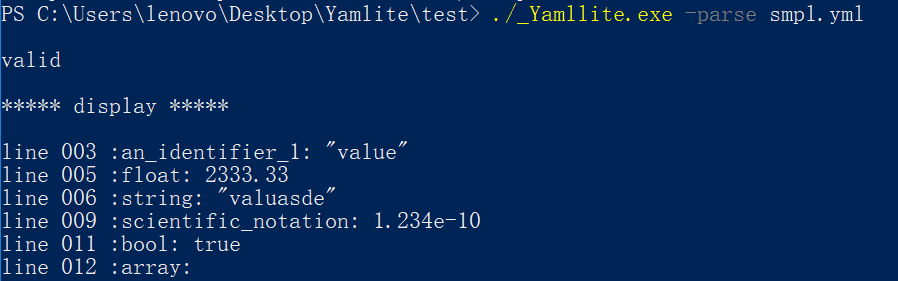
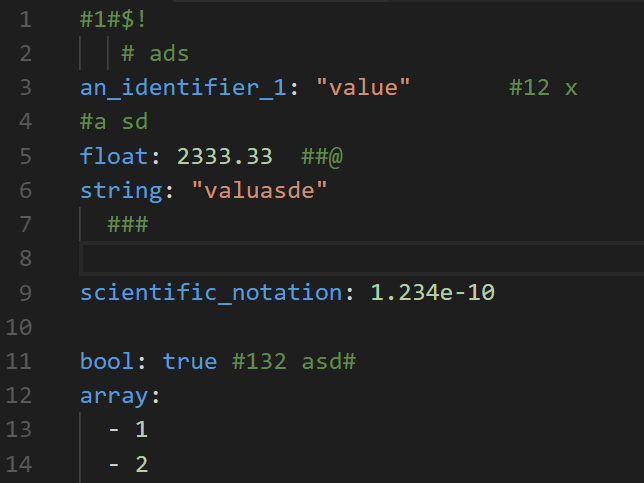
\_STD tuple<std::size\_t, const char\*, std::size\_t> preprocess(std::size\_t line\_num, const char\* c)  
功能：

忽略以#开头的行内注释

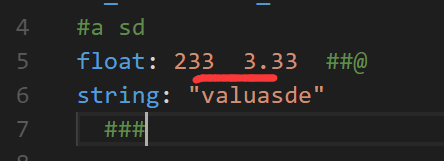
检查空格段的数量——除去开头缩进中含有的空格，文字段间最多包含一段空格

忽略结尾的空格

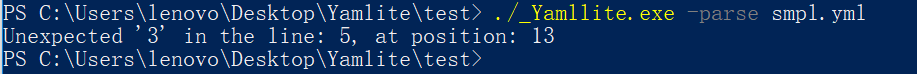
返回处理后的内容，以供下一步处理



如果一行中出现多个空格段，在预处理时报错，如：



将在命令行输出如下错误信息



任思远：

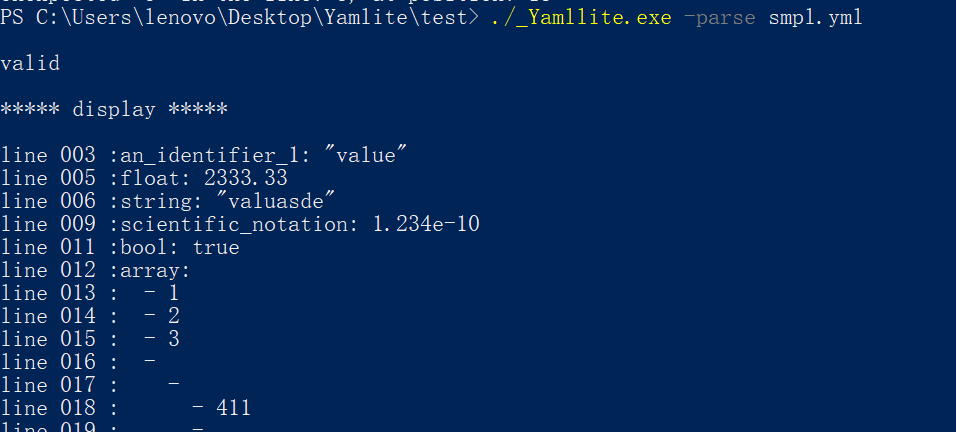
1. 扫描一行： void seperate\_one\_line(std::size\_t line\_num, const char\* c, std::size\_t size)  
   事实上，每一行只有4种类型：
   * kv : 8, "key: value" key: string(value ""), number, scientific\_notation, bool
   * key : 4, "key:" kv (but value is array or kv)
   * value : 2, "- value" trivial value type of the array
   * new\_line : 1, "-" to start a new line, indicating the value is "kv" or "array"  
     enum class line\_t { kv = 8, key = 4, value = 2, new\_line = 1 };

先统计有多少个 indent (1个 indent 为2个 space)，然后再看下一个是啥 如果是 ‘-’，就说明是 value 或 new\_line， 否则尝试解释为 key 或 kv。  
最主要的工作全部由两个函数 key\_scan 和 value\_scan 完成。  
（注：科学计数法那里写的不是很完善，没怎么做异常处理）

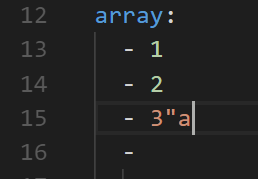
函数执行完成后，所有有效行的信息都会被保存到一个table里面（table是一个vector<line>，用于后续的遍历过程）。

通过display操作来展现separate\_one\_line的结果：

无效行都被除去，并且有效行的信息全部原样展现（不包含注释）



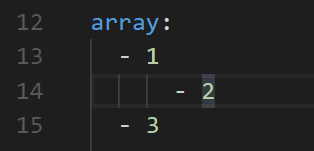
异常处理：在此函数中，保证所有行的格式都是正确的





更进一步，我添加了一个提前扫描缩进的异常处理：

如果某一行的缩进比上一行多了不只一层，则会报错。





1. 解析： void parsing(option opt, const char\* search\_path)  
   先做 -parse，这一块的大致思路是：  
   遍历所有行，用一个栈来存储 indent，如果小于栈顶的 indent，那么尝试 reduce。  
   这里面有2个结构：

using \_key = std::string;

using \_value = std::string;

\_STD unordered\_map<\_key, \_value> find\_table;

find\_table 用来存储所有的值 与其对应的目录，存储过程是在扫描中进行的。也就是说，-parse 之后，如果 option 为 -find，只需要在 find\_table 中查找 search\_path 即可。

typedef struct path {

YAMLlite::line l;

std::string cur\_path;

std::string parent\_path;

int array\_level;// -1 if no - array, store array\_level for reduce

path(YAMLlite::line \_l, const std::string& \_cur\_path, const std::string& \_parent\_path, int ary\_level = -1)

:l(\_l), cur\_path(\_cur\_path), parent\_path(\_parent\_path), array\_level(ary\_level)

{}

};

\_STD stack<path> stk;

stk 用来存储每一行的信息，包括本行信息，以及当前目录和父目录，还有数组的层次。stk中的元素都位于同一段中，段结尾处全部 reduce。  
其实一开始只有 key 和 new\_line 要存储当前目录，但是后来改了一下，存储所有行，于是对于 kv 或 value，就存储父目录(当然这两个同时也要根据上一项的（当前/父）目录来存储进 find\_table )。  
我做了一个表来表示每一行可接受的状态：

/\*

\* 0和1指的是这一行与上一行缩进的差， 若 < 0 记为 -1, then reduce

\* 左列是上一行的类型，值是这一行可以接受的类型。

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\* |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| 0 | 1 | -1 |

\* | kv : 8 | 0000 | 0000 | 1111 |

\* | key : 4 | 0000 | 1111 | 0000 |

\* | value : 2 | 0011 | 0000 | 1111 |

\* | new\_line : 1 | 0000 | 1111 | 0000 |

\* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

\*\*/

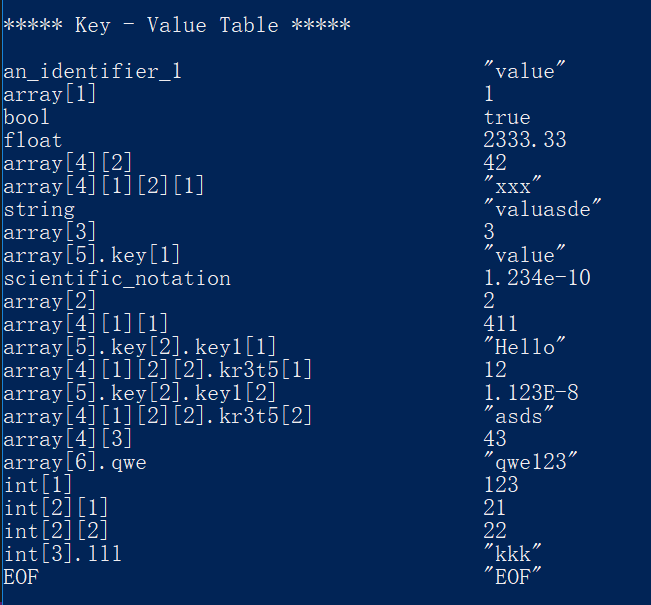
举例：如果上一行是 value，并且这一行的 indent 多了 1，那么结果就是 0000 报错；如果这一行的 indent 与上一行同级，则结果是 0011，即这一行只能是 value 或 new\_line 类型。  
具体写起来是这样：首先看 stk 是否为空，即开始新的一段。  
当 stk 非空时，比较这一行与上一行的缩进，分为三种情况：

1. 相同
2. 多1层缩进
3. 少于上一行，那么尝试 reduce

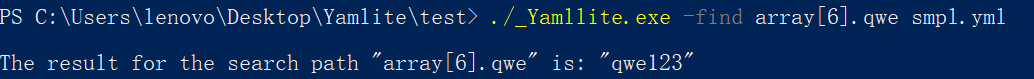
其实1和2大同小异，就是先查表，然后再根据可能的情况进行分析。中间有个地方比较累，就是 stk 的记录，因为一开始我想的是 只记录 key 和 new\_line，因为这两个才算是新的目录，但后来写的时候忘了，stk记录的是每一行，所以又强行记录了 kv 和 value。为此我还专门为 find\_table 和 stk 封装了4个函数，专门用来记录。一开始 stk 存的信息还少，后来加了记录函数，又重构了两次后就清晰多了。只要搞清楚什么情况下选择 "当前目录" 还是 "父目录" 就好。

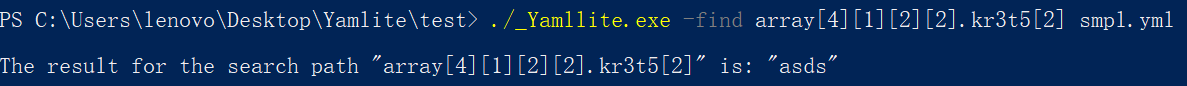
至于3 reduce，还是先查表，然后 stk 不断弹出，直到与当前 indent 同级别。（当然，注意到：如果最后 stk 只剩1个元素，即判断为开启新的一段，将最后一个元素手动弹出）看表的话虽然4种情况都是会出现，但其实还是可以更进一步分层，我在表后面的注释已经写了。如果 stk 为空，那么新的一行只能是 kv 或 key，其他就是非空的情况了。 其实这一块我偷了个懒，讲道理我觉得一个分析器应该把这个结构记下来，也就是当 reduce 一整段的时候应该把这一段的全部信息存储到某个结构中，并且这种数据结构可以完整地刻画出这一段的信息。但因为作业中要求的操作只有 -find，而所有 value 和 其对应的目录 已经被记录在 find\_table 中，所以我就没写。。。 (￣ェ￣;)

关于把value全部扔进 find\_table 中，有一个好处，就是不需要再额外解析从命令行传来的 search\_path。以下是 find\_table 中的值，（因为hash是无序容器，所以看起来没有整体的结构了，为了改进这一点，可以考虑在hash插入的操作时绑定一个额外的插入操作，插入一个有序容器中，这样结构就是一样的了。不过我没写。。。）



-find 操作：







刘瑞康：

1. -json 部分

前要条件：已经通过了 -parse ，假设格式已经保证正确。  
主要思路：遍历所有行，同时用一个栈来辅助存储 ' ) ' 和 ' } ' 的 缩进层级，延迟到合适的位置弹出栈并输出到json文件。（输出格式参照给出的样例格式来进行）

功能：按照已处理好的表中信息格式化输出到json文件



