

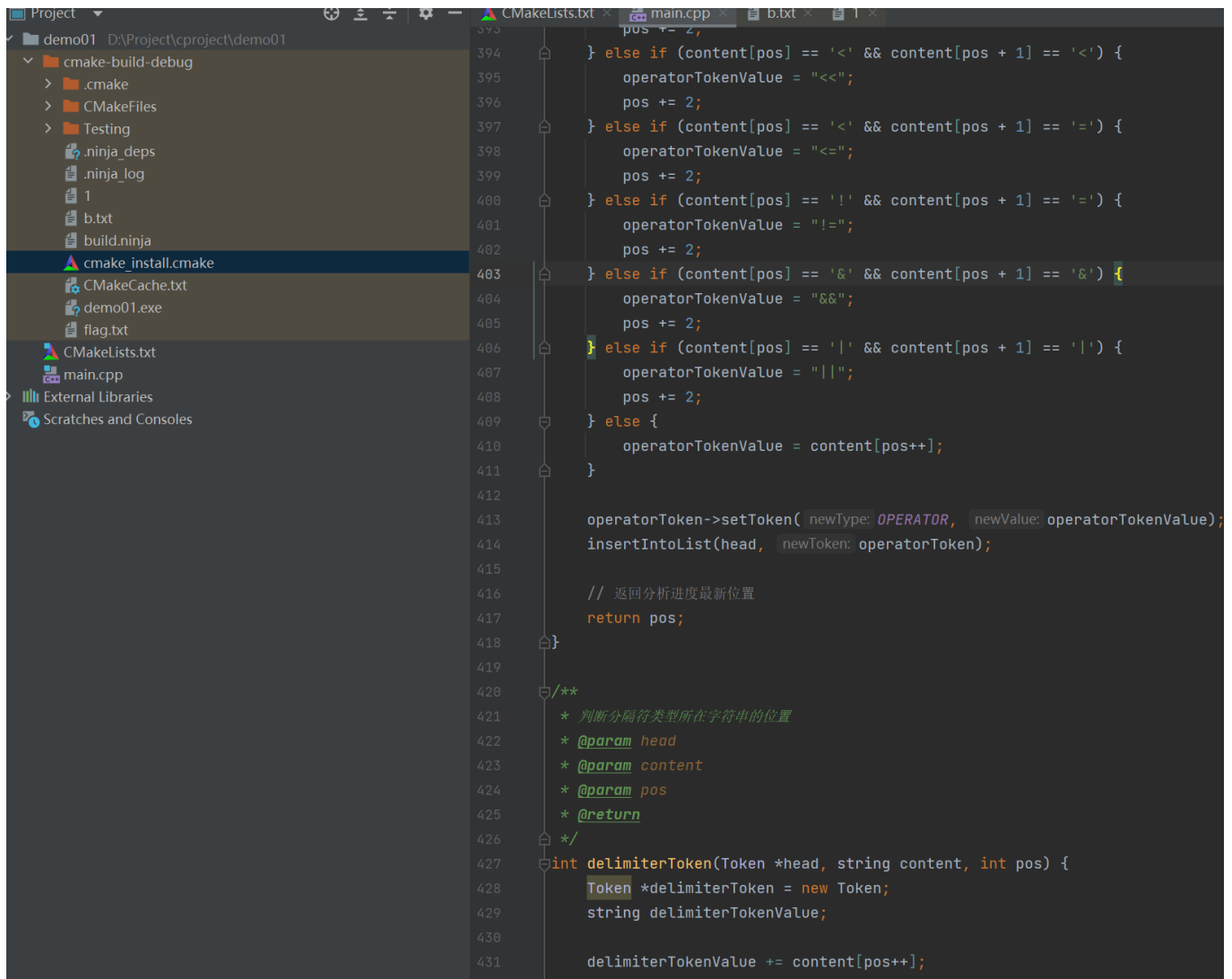
设计报告

输出样例

```
# => 分隔符
include => 标识符
< => 运算符
iostream => 标识符
> => 运算符
using => 关键字
namespace => 关键字
std => 标识符
; => 分隔符
int => 关键字
main => 标识符
( => 分隔符
) => 分隔符
{ => 分隔符
cout => 标识符
<< => 运算符
"hello world" => 串
<< => 运算符
endl => 标识符
; => 分隔符
return => 关键字
0 => 数字
; => 分隔符
} => 分隔符
```

编译方法

cpp 的可执行程序通过在 clion 中 cmake 命令编译。



```
393 pos += 2;
394 } else if (content[pos] == '<' && content[pos + 1] == '<') {
395     operatorTokenValue = "<<";
396     pos += 2;
397 } else if (content[pos] == '<' && content[pos + 1] == '=') {
398     operatorTokenValue = "<=";
399     pos += 2;
400 } else if (content[pos] == '!' && content[pos + 1] == '=') {
401     operatorTokenValue = "!=";
402     pos += 2;
403 } else if (content[pos] == '&' && content[pos + 1] == '&') {
404     operatorTokenValue = "&&";
405     pos += 2;
406 } else if (content[pos] == '|' && content[pos + 1] == '|') {
407     operatorTokenValue = "||";
408     pos += 2;
409 } else {
410     operatorTokenValue = content[pos++];
411 }
412
413 operatorToken->setToken( newType: OPERATOR, newValue: operatorTokenValue);
414 insertIntoList(head, newToken: operatorToken);
415
416 // 返回分析进度最新位置
417 return pos;
418 }
419
420 /**
421  * 判断分隔符类型所在字符串的位置
422  * @param head
423  * @param content
424  * @param pos
425  * @return
426  */
427 int delimiterToken(Token *head, string content, int pos) {
428     Token *delimiterToken = new Token;
429     string delimiterTokenValue;
430
431     delimiterTokenValue += content[pos++];
```

python 的前端可执行程序通过 auto py to exe 工具打包，将两个 exe 合并为一个。



原理

整体框架

通过 python 的 tkinter 桌面程序框架，通过 `os.popen()` (系统命令) 执行本地编译好的 `cpp` 词法分析程序。本地的 `cpp` 写成命令程序，通过 `argv` 接收命令行参数 (`cLexer.exe <输入文件路径>`)，来实现词法分析器的功能。

cpp词法分析器设计思路

将每一个词当作一个 Token，在顺序遍历源文件的时候依次通过分析当前开头的字符可能会匹配哪种类型的 Token，如果可能匹配多种类型的 Token，就再顺序遍历到下一个字符，再次判断此时可能匹配哪些 Token（状态转移的思路）

```
// 存储token的类型
enum TokenType {
    KEYWORD = 1,
    IDENTIFIER,
    NUM,
    STR,
    OPERATOR,
    DELIMITER
};

//存储 token 信息的结构体
struct Token {
    enum TokenType type;
    string value;
    Token *next = nullptr;

    void setToken(TokenType newType, string newValue) {
        this->type = newType;
        this->value = newValue;
    }
};
```

startCharType 函数最能说明这个设计思路的原理。看开头的字符符合哪些 Token 的可能性。

```
int startCharType(char ch) {
    int type = 0;
    if (isDigit(ch)) {
        type = 1;
    } else {
        if (ch == '"' || ch == '\n') {
            type = 2;
        } else {
            if (isOperator(ch)) {
                type = 3;
            } else {
                if (isDelimiter(ch)) {
                    type = 4;
                } else {
                    if (isAlpha(ch)) {
                        type = 5;
                    } else {
                        if (ch == '\n') {
                            type = 6;
                        } else {

```

```
    type = 7;
  }
}
}
}
}
}
return type;
}
```