

[一、项目内容 2](#_Toc135941000)

[二、项目目的 3](#_Toc135941001)

[三、项目文档 3](#_Toc135941002)

[（一）需求分析 3](#_Toc135941003)

[（二）代码设计 4](#_Toc135941004)

[1. 算法设计 4](#_Toc135941005)

[2、架构设计 7](#_Toc135941006)

[3、数据结构设计 8](#_Toc135941007)

[4、平台实现设计 9](#_Toc135941008)

[（三）程序实现 9](#_Toc135941009)

[1. 代码 9](#_Toc135941010)

[2. 界面 9](#_Toc135941011)

[3. 优化 9](#_Toc135941012)

[（四）程序测试 10](#_Toc135941013)

[1、测试设计 10](#_Toc135941014)

[2、测试用例与结果 10](#_Toc135941015)

[3、测试评估 11](#_Toc135941016)

[四、实验总结（心得体会） 12](#_Toc135941017)

[（一）编码心得 12](#_Toc135941018)

[1. “节点”和“结点” 12](#_Toc135941019)

[2. 爆栈 12](#_Toc135941020)

[3.值与引用 12](#_Toc135941021)

[（二）学习心得 12](#_Toc135941022)

[（三）提升空间 13](#_Toc135941023)

[五、参考文献 13](#_Toc135941024)

[六、项目自评 13](#_Toc135941025)

# 一、项目内容

（1）以文本文件的方式输入某一高级程序设计语言的所有单词对应的**正则表达式**，系统需 要提供一个操作界面，让用户打开某一语言的所有单词对应正则表达式文本文件，该文本文 件的具体格式可根据自己实际的需要进行定义。

（2）需要提供窗口以便用户可以查看转换得到的 NFA（可用状态转换表呈现）

（3）需要提供窗口以便用户可以查看转换得到的 DFA（可用状态转换表呈现）

（4）需要提供窗口以便用户可以查看转换得到的最小化 DFA（可用状态转换表呈现）

（5）需要提供窗口以便用户可以查看转换得到的词法分析程序（该分析程序需要用C语言描述）

（6）对要求(5)得到的源程序进行编译生成一个可执行程序，并以该高级程序设计语言的一 个源程序进行测试，输出该源程序的单词编码。需要提供窗口以便用户可以查看该单词编 码。

（7）对系统进行测试:

（A）先以 TINY 语言的所有单词的正则表达式作为文本来测试，生成一个 TINY 语言的词法分析源程序；

（B）接着对这个词法分析源程序利用 C/C++编译器进行编译，并生成可执行程序；

（C）以 sample.tny 来测试，输出该TINY语言源程序的单词编码文件sample.lex。

（8）要求应用程序为Windows界面

（9）书写完善的软件文档

# 二、项目目的

1、了解正则表达式和有限自动机的基本概念和原理；

2、掌握NFA、DFA和最小化DFA的算法及其实现方式；

3、熟悉C语言的基本语法和数据结构，以及如何将自动生成的状态转换表转换为可执行的词法分析程序；

4、学习程序设计和开发的过程，包括需求分析、系统设计、编码实现、测试调试等环节；

5、学会交互式界面的设计和实现，以及与用户进行良好的交互体验。

# 三、项目文档

## （一）需求分析

要完成这个作业，我需要做到：

1. 使用Qt和C++，编写具有交互窗口的可执行程序。窗口需要支持打开本地文件、显示图表、显示文本框
2. 了解题目中各个专有名词的定义，例如正则表达式、NFA、DFA、词法分析程序、可执行程序、单词编码
3. 编写高级程序设计语言的所有标识符对应的正则表达式
4. 了解词法分析程序的转化算法（包括Thompson算法、子集构造算法、Hopcroft-Karp算法）、代码自动生成算法
5. 了解TINY语言，能够独立设计测试用例并检验程序的正确性

## （二）代码设计

### 1. 算法设计

#### 1.1. 正则表达式转C++源代码【语法分析+语义分析】

##### 1.1.1.正则表达式转NFA

一条正则表达式就像算术表达式：字符之间用运算符连接起来，从而能够得出某种词法。同样地，可以用栈解析正则表达式：当遇到一般字符和左括号‘(’时，压入栈中；当遇到其它运算符时，按照运算规则pop、处理、再push；最后将栈内的所有元素pop并连接，得到NFA图。

|  |  |  |  |
| --- | --- | --- | --- |
| 单个字符 | **一般字符** | a |  |
| **特殊含义字符** | digit | 表示数字，替换成(0|1|2|3|4|5|6|7|8|9)，再按照一般字符处理 |
| lettle | 表示字母，替换成(a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z)，再按照一般字符处理 |
| . | 表示除了\n以外的任意字符，替换成(char)1，做词法分析的时候遇到(char)1则对应任意字符 |
| **转义符** | \ | 将下一个字符强制当作一般字符（用于解决字符二义性问题） |
| **运算符** | 一元运算符 | ? |  |
| \* |  |
| 二元运算符 | | |  |
| 连接 |  |  |

\*注释：蓝色、实线表示是新建的，黑白、虚线表示原有的。

NFA图的特性是：只有一个起点，一个终点。

**运算符**之间存在优先级问题，（）＞?=\*＞丨。

##### 1.1.2. NFA转DFA

DFA的特点是：不存在以ε（注释：ε又为epsilon，可以理解为无条件转移，可以说它是个假的转移条件）为转移条件的状态转移。因此也确保了一个转移条件只对应一个下一节点。所以，NFA转DFA，说大白话就是把其中的epslion全部去掉。

采用子集构造法：

|  |  |
| --- | --- |
| **步骤** | **示意图** |
| 读取NFA |  |
| ①从NFA的起始节点开始，搜索经过ε能到达的所有节点，合并这些被ε连接通的全部节点，作为DFA的起始节点【深度/广度优先搜索】 |  |
| ②遍历DFA起始节点里的所有节点id1，再遍历每个转移条件condition，如果存在id1--condition-->id2，则构造节点id2 |  |
| ③对于每个id2，再次搜索经过ε能到达的所有节点id3，id2和id3构成了DFA的新节点 |  |
| ④以新节点为所谓的起始节点，重复步骤②③，直至不再有新节点产生 |  |
| ⑤简化DFA，就是为每个DFA节点分配唯一的ID。 |  |

注意，简化DFA之后，DFA表的第一列的ID不也许不再是递增的。

##### 1.1.3. DFA转最小化DFA

|  |  |
| --- | --- |
| **步骤** | **示意图** |
| 读入DFA |  |
| ①先按照是否为终态节点，划分接受状态集和非接受状态集 |  |
| ②遍历每个状态集，遍历每个转移条件，遍历状态集里的每个id（注意这个遍历顺序）。设id--conditon-->id’，存在两个不同的id，其id’属于不通的状态集，则将两者划分开来。 |  |
| ③重复②，直至不再有新的状态集产生。 |  |
| ④构造新的DFA，为每个状态集赋予新的ID， |

##### 1.1.4. 最小化DFA转C++代码

我提前在VS 2022写了一版可执行的词法分析demo，把demo里固定不变的部分保存为template.txt，需要改变的部分再动态生成并拼接，最后就可以实现生成目标代码。

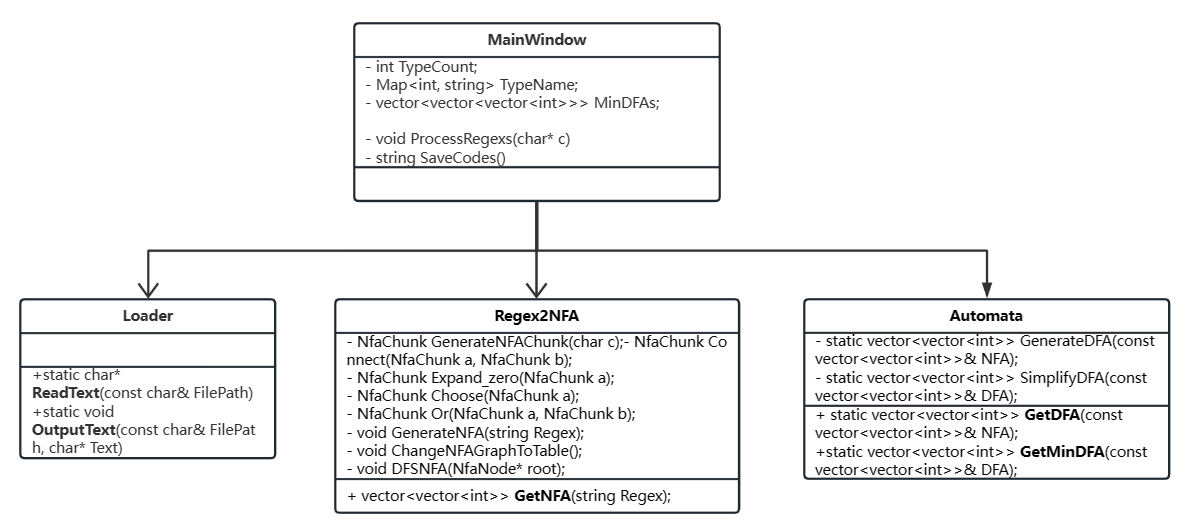
#### （2）源代码转单词编码【词法分析】

整体思路：源代码里的每个字符都对应了自动机里的一个转移条件，每读入一个字符，就开始执行一次状态转移。遍历所有DFA，如果从某个字符开始，可以转移到某个DFA的终态，则输出字段、判断词法；如果对于每个DFA，该字符都不能到达终态，则跳过该字符。

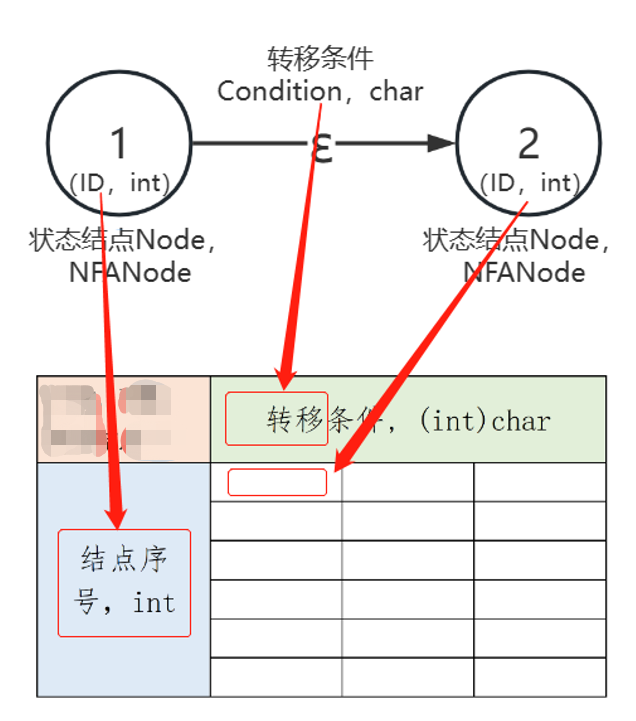
因为程序需要读取一段完整的代码，而cin输入的时候遇到空格或者换行符会自动停止读入。于是我直接读取文件流，用户需要做的是输入源码码文件的路径。

### 2、架构设计

#### 2.1. 代码框架



#### 2.2. NFA图和NFA表格式



### 3、数据结构设计

#### 3.1. NFA图

在解析正则表达式的过程中，NFA最好用**图**结构表示。定义图结点结构体如下：

|  |
| --- |
| struct NfaNode {  int id;  vector<NfaNode\*> nextNode; // 下一结点  map<NfaNode\*, char> transition; // 下一结点 与对应的 转移条件  }; |

为了在栈中统一处理正则表达式的字符（NFA）和运算符，我自定义了能同时容纳两者的结构体：

|  |
| --- |
| // NFA块  struct NfaChunk {  char op; // 若为#，说明这仅是结点块；若不为#，说明这是运算符，后面的结点元素无实际作用  NfaNode\* start;  NfaNode\* end;  } |

#### 3.2. 表

在转化NFA的过程中，NFA最好用**表格**表示，易于处理。之后的DFA也需要用到表格处理。考虑到表格大小的动态性，但又需要按行、按列遍历，需要记录终态，规定表格类型为vector<int>格式如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 行数，列数，起始节点（唯一），终态结点（不唯一）……vector<int> | 转移条件, vector<(int)char> | | |
| 状态结点Node，vector<int> | 单元格Cell, vector<int> |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### 4、平台实现设计

运行环境：Windows x64

开发语言：C++ 11

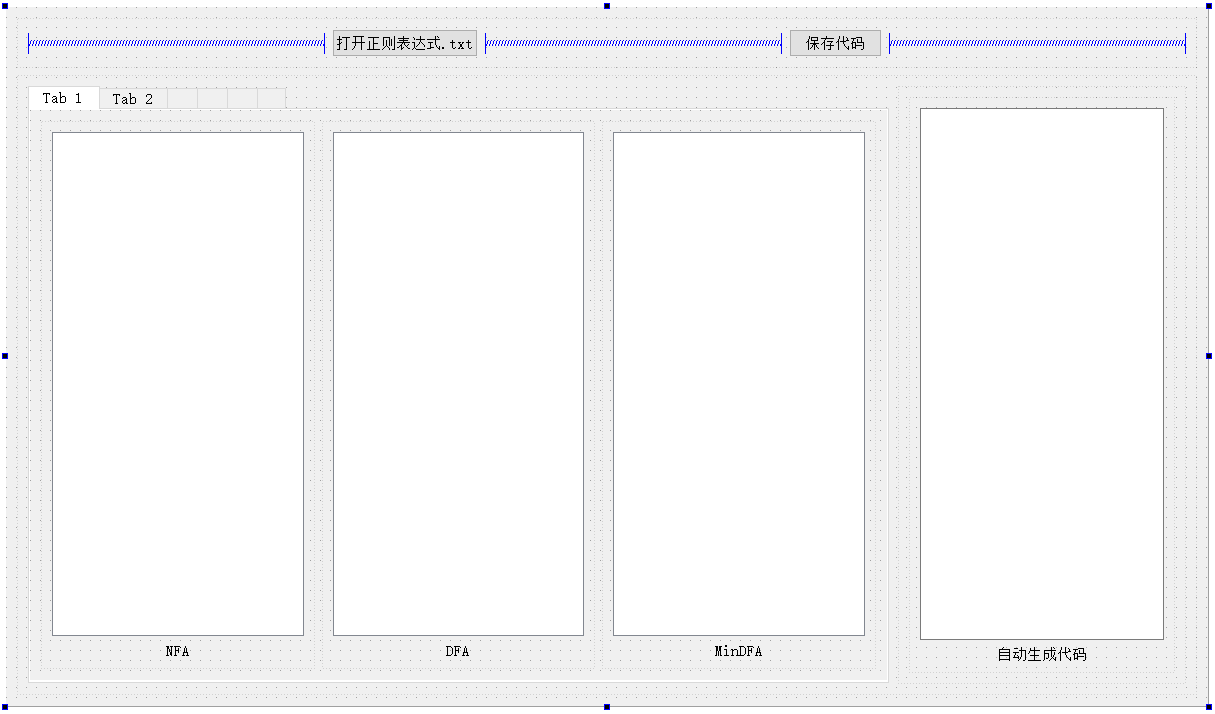
编码工具：Qt 4.12.2，Visual Studio 2022

## （三）程序实现

### 1. 代码

* 请查看ComprehensiveExp1\SourceCode\里的源码，笔者已经尽自己的能力提到了命名的可读性
* 大部分函数声明已标注brief英文注释，将鼠标停留在上面即可查看
* 考虑到笔者能力有限，函数内的注释仍保留了全中文

### 2. 界面



### 3. 优化

#### 3.1. 容器初始化

|  |  |  |
| --- | --- | --- |
|  | **代码** | **解释** |
| **优化前** | vector < vector<int>> table;  vector<int> cell;  for(…)  **cell.clear();**  **for(…)**  **cell.push\_back(i);**  table.push\_back(cell); | 在生成词法分析程序的源码时，需要将MinDFA转储出去，就需要大量的赋值代码。  优化后的语法显然简洁很多。 |
| **优化后** | vector < vector<int>> table;  for(…)  cell.push\_back(**{1, 1, 2, }**); |

## （四）程序测试

### 1、测试设计

对于用户输入的每个词法规则文本，做如下规定：

①用括号括住正则表达式的每个单词，避免二义性。例如digit\*表示接收d、i、g、i、和任意个t，而(digit)\*表示接收任意个数字。

②对于每种类型的关键词，在第一行写类型名，第二行写正则表达式，下一个关键词另起一行声明。

③“.”表示除了换行符以外的任意字符。

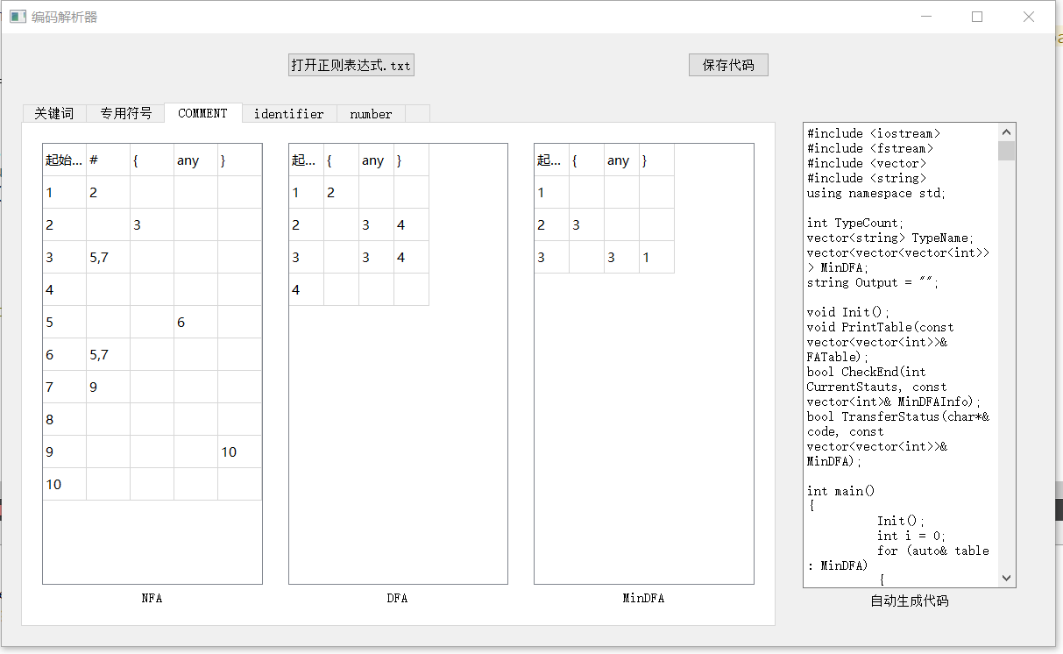
④“\”是转义符。例如“\\*”表示字符\*，“\*”表示正则表达式运算符。例如“\n”表示换行符，“n”表示字母n。

### 2、测试用例与结果

#### TINY.txt

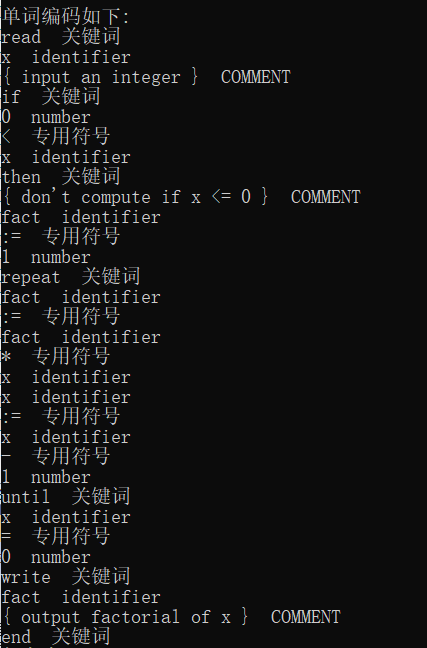
|  |
| --- |
| KEYWORD  if|then|else|end|repeat|until|read|write|IF|THEN|ELSE|END|REPEAT|UNTIL|READ|WRITE  OPERATOR  +|–|\*|/|<|=|;|:=  IDENTIFER  letter letter\*  NUMBER  digit digit\*  SPACE  | |  COMMENT  {\*} |

中间过程产生的数据量过大，不便展示，此处仅展示最后的部分的最小化DFA。



#### SAMPLE1.TNY

|  |
| --- |
| read x; { input an integer }  if 0 < x then { don't compute if x <= 0 }  fact := 1;  repeat  fact := fact \* x;  x := x - 1  until x = 0;  write fact { output factorial of x }  end |



### 3、测试评估

测试结果达到了笔者期望。

# 四、实验总结（心得体会）

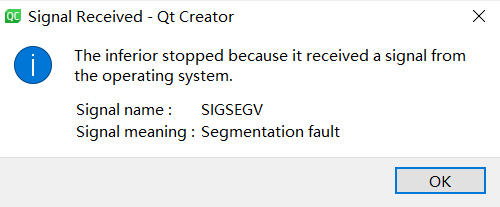
## （一）编码心得

### 1. “节点”和“结点”

我在写文档的过程中，突然发现自己分不清“节点”和“结点”。

我在[这篇博客](https://blog.csdn.net/qq_42270373/article/details/83758928)中找到了参考：节点被认为是一个实体，有处理能力，比如，网络上的一台计算机；而结点只是一个交叉点，像“结绳记事”，打个结，做个标记，仅此而已，还有就是，要记住：**一般算法中的点都是结点**。

### 2. 爆栈



情况一：指针问题，把新开辟的数组赋值给了错误的变量a，然后再把这个变量a赋值给别的变量b，就出问题了。

情况二：在疯狂Debug又终止又Debug的过程中，内存会抽搐，随便运行一行赋值代码都会栈溢出。这种情况下重启电脑就好了。

### 3.值与引用

3.1.函数参数

直接传容器的开销是很大的，这意味着编译器需要再复制一份容器的实例。最好的方法是传引用，一来，引用本身是地址，开销小；二来，可以通过引用直接修改容器的值，也避免了传返回值时的二次开销。如果想确保不修改容器的值，在参数前加const即可。

3.2.容器引用的迭代器

在for-auto语句中，我们也可以用3.1.的函数参数的思路去理解返回值。例如vector< vector<int>> Table;。①for(auto cell : Table)，cell是Table里每个vector<int>的**深拷贝**的返回值，开销较大，且不允许修改Table的值；②for(auto& cell : Table)，cell是Table里每个vector<int>的引用的返回值，是**地址**，开销较小，且**允许修改**Table的值；③for(const auto& cell : Table)，cell是Table里每个vector<int>的引用的返回值，是地址，开销较小，且**不允许**修改Table的值。

### 4.Qt特性

鬼故事：Qt的函数不写返回值不会报错。

但这只是在debug模式下能正常运行。但是，在release模式下，程序会因此产生崩溃且没有任何提示。

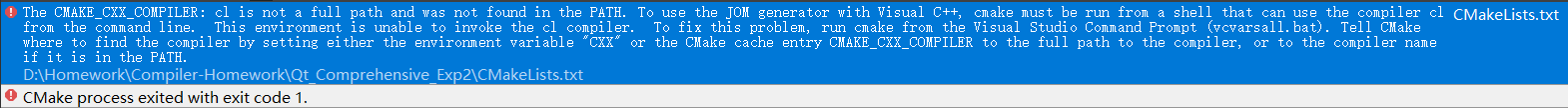
## （二）学习心得

通过完成编译原理课程的大作业，我对词法分析有了更深入的了解和认识。首先，我明白了词法分析的重要性，词法分析是编译器的第一步，它将源代码转换成一个个的单词，为后续的语法分析提供了基础。其次，我学会了使用正则表达式和有限自动机来描述和实现词法分析器，包括手工实现和使用程序生成实现代码的方式。正则表达式是一种用于描述字符串的模式语言，通过正则表达式可以方便地描述词法单元的规则。而有限自动机是一种描述有限状态的图模型，它可以通过状态转移的方式实现对输入字符串的识别和分析。最后，我还学习了词法分析器的实现方法。通过大作业的实践，我深刻体会到了词法分析器的工作原理和实现方法，这对我的编译原理和软件开发能力的提升具有重要意义。

## （三）提升空间

#### 1、CMake

采用更主流的CMake去构建项目，可以满足其它不同平台的开发者去更方便地查看我的代码、运行程序、并提出修改意见。但我的Qt一直报错如下：



考虑到开发优先级的问题，我没有解决这个问题，我采用了qmake。

在未来的项目中，我希望能解决这个问题。

# 五、参考文献

老师的课件。

# 六、项目自评

1.项目完成情况

实现读入一系列正则表达式，构建NFA、DFA、最小化DFA，并生成对应的c++词法分析代码，解析出.lex文件。实现交互界面，书写了使用说明、测试文档、设计文档。

2.自评分数

95分

3.原因说明

（1）独立完成：我完全独立地完成了这个项目，并书写了简洁明了的文档，绘制了算法示意图，没有抄袭或借鉴其他同学或外部资源的代码。

（2）理解深入：在完成这个项目的过程中，我对词法定义、自动机等概念有了更深入的理解和掌握，获取了这门课应该学到的知识。

（3）功能完善：我的项目能够很好地完成词法分析的任务，能够接收输入的正则表达式，支持转义符、特殊含义字符，构造状态机，进行词法分析，打印结果。我还实现了一些额外的功能，如错误处理和代码优化等。