正则表达式引擎

基本的数据结构定义

核心思路是读取正则表达式以后生成对应的NFA，NFA中有边和状态两个结构。边的结构记录了它的起点和终点，同时记录了匹配的其他需求。

```
//用于处理'^'字符
enum { NEXCLUDED = false, EXCLUDED = true };
//用于处理预处理类型, 0-128以内ASCII字符直接匹配
enum { LCASES=256, UCASES=257, NUM=258, EPSILON=259, ANY=260, WS=261 };
class Edge
{
public:
    State *start;
    State *end;
    int type;
    int exclude;
    Edge(State *s, State *e, int t, bool ex = NEXCLUDED) :start(s), end(e), type(t), exclude(ex) {};
}
```

状态有预备，成功和失败两种，同时每个状态维护两个向量，向量存储了出边和入边的指针。

```
enum { READY = -1, SUCCESS = 1, FAIL = 0};

class State
{
public:
    int status;
    std::vector<Edge *> InEdges;
    std::vector<Edge *> OutEdges;
}
```

Nfa类会存储一个正则表达式，同时存储NFA的起点和终点，并使用了两个链表来维护NFA的边和状态，同时用一个链表来存储匹配成功的字符串。两个静态的字符串指针用于记录文件和正则表达式字符串的读取状态，因为是静态常量，因此都只会对文件内容和正则表达式扫描一次，避免在匹配成功的字符串中再匹配子串。

```
char *regex;
    State *Start;
    State *End;
    std::list<Edge *> edgeList;
    std::list<State *> stateList;
    std::list<char> matchedChar;
    static char *regRead;
    static char *fileRead;
```

处理方式

关键的部分在于匹配字符串时采取的思路，尤其是特殊字符的生成NFA的方式，这个不同于课本上最开始的NFA生成算法，而是基于读取字符串的过程，同时避免了字符串的回退，采用了及时处理方案，使得处理更加简单的同时避免生成冗余状态，兼顾了时间和空间效率。

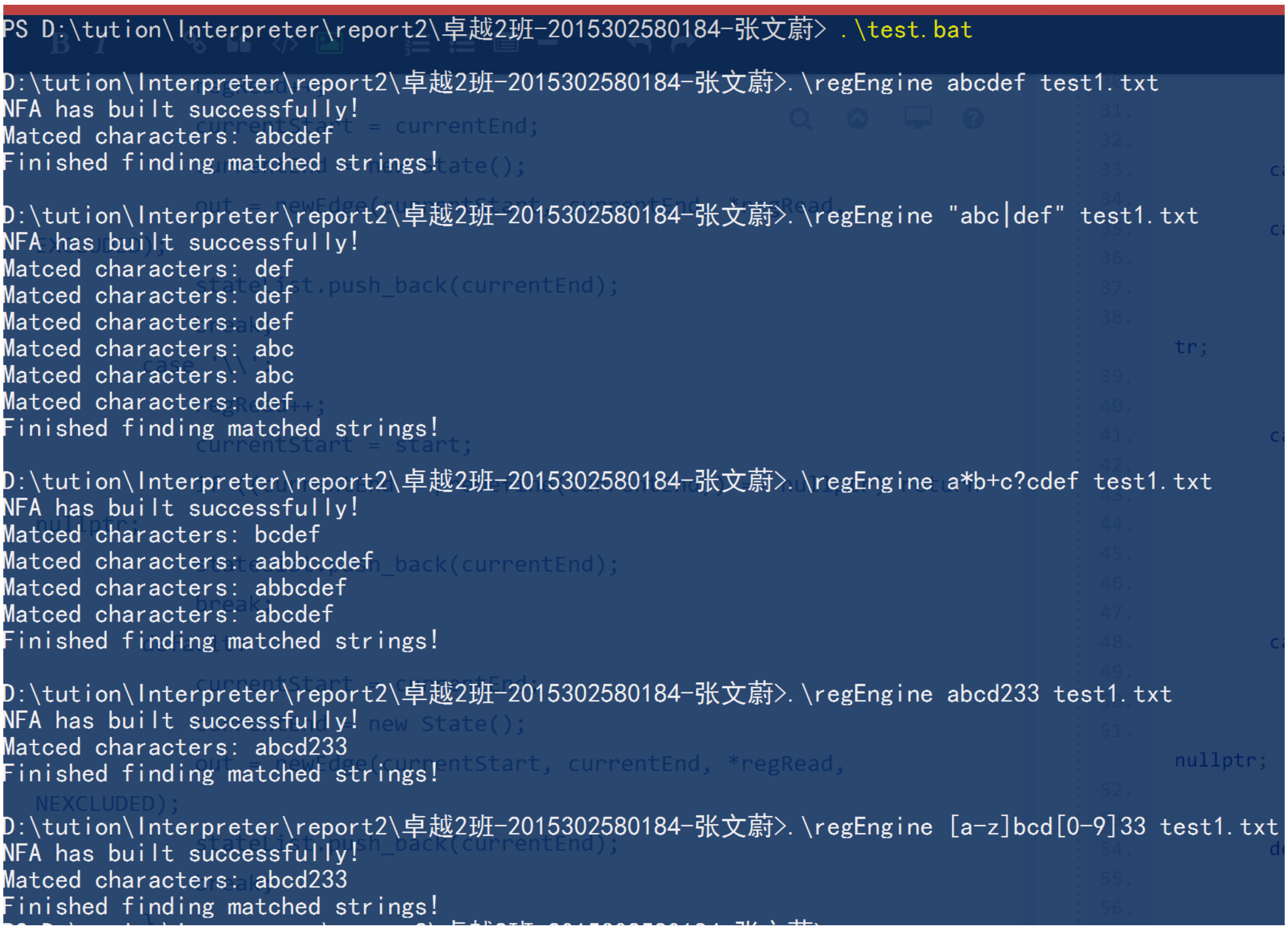
```
switch (*regRead) {
        case '.':   /* any */
            currentStart = currentEnd;
            currentEnd = new State();
            out = newEdge(currentStart, currentEnd, ANY, NEXCLUDED);
            stateList.push_back(currentEnd);
            break;
        case '|':   // alternate
            regRead++;
            currentStart = start;
            alternate= regex2nfa(regRead, start);
            currentEnd->merge(alternate);
            stateList.remove(alternate);
            regRead--;
            break;
        case '?':   // zero or one
            out = newEdge(currentStart, currentEnd, EPSILON, NEXCLUDED);
            break;
        case '*':   // zero or more
            alternate = currentEnd;
            currentStart->merge(alternate);
            stateList.remove(alternate);
            currentEnd = currentStart;
            break;
        case '+':   /* one or more */
```

```
            out = newEdge(currentEnd, currentEnd, edgeList.back()->type, NEXCLUDED);
            break;
        case '(':
            regRead++;
            currentStart = currentEnd;
            currentEnd = regex2nfa(regRead, currentEnd);
            break;
        case ')':
            return currentEnd;
        case '[':
            regRead++;
            currentStart = currentEnd;
            if((currentEnd = group(currentEnd)) == nullptr) return nullptr;
            stateList.push_back(currentEnd);
            break;
        case '^':
            regRead++;
            currentStart = currentEnd;
            currentEnd = new State();
            out = newEdge(currentStart, currentEnd, *regRead, EXCLUDED);
            stateList.push_back(currentEnd);
            break;
        case '\\':
            regRead++;
            currentStart = start;
            if ((currentEnd = preDefine(currentEnd)) == nullptr) return nullptr;
            stateList.push_back(currentEnd);
            break;
        default:
            currentStart = currentEnd;
            currentEnd = new State();
            out = newEdge(currentStart, currentEnd, *regRead, NEXCLUDED);
            stateList.push_back(currentEnd);
            break;
        }
```

基于图示的方式说明NFA的生成方式可能更加直观，本文有待完善。

结果

目前只测试通过了下图中的若干用例，可以说只跑通了基本的功能，但是代码中对所有的要求其实都有一个实现（其实就是时间紧ddl多没有完成debug的工作，逃~），希望能在下周前提交一个更加完善，能实现所有需求的版本！



```
PS D:\tution\Interpreter\report2\卓越2班-2015302580184-张文蔚> .\test.bat

D:\tution\Interpreter\report2\卓越2班-2015302580184-张文蔚>.\regEngine abcdef test1.txt
NFA has built successfully!
Matced characters: abcdef
Finished finding matched strings!

D:\tution\Interpreter\report2\卓越2班-2015302580184-张文蔚>.\regEngine "abc|def" test1.txt
NFA has built successfully!
Matced characters: def
Matced characters: def
Matced characters: def
Matced characters: abc
Matced characters: abc
Matced characters: def
Finished finding matched strings!

D:\tution\Interpreter\report2\卓越2班-2015302580184-张文蔚>.\regEngine a*b+c?cdef test1.txt
NFA has built successfully!
Matced characters: bcdef
Matced characters: aabbccdef
Matced characters: abbcdef
Matced characters: abcdef
Finished finding matched strings!

D:\tution\Interpreter\report2\卓越2班-2015302580184-张文蔚>.\regEngine abcd233 test1.txt
NFA has built successfully!
Matced characters: abcd233
Finished finding matched strings!

D:\tution\Interpreter\report2\卓越2班-2015302580184-张文蔚>.\regEngine [a-z]bcd[0-9]33 test1.txt
NFA has built successfully!
Matced characters: abcd233
Finished finding matched strings!
```