



# 一、项目内容

mini-c 语言作为测试

（1）以 mini-c 的词法进行测试，并以至少一个 mini-c 源程序进行词法分析的测试(该mini-c 源程序需要自己根据 mini-c 词法和语法编写出来，类似于 sample.tny

（2）以 mini-c 的语法进行测试，并以测试步骤(1) 的源程序所生成的单词编码文件进行语法分析，生成对应的语法树

# 二、项目目的

进一步验证词法分析和语法分析算法的正确性和鲁棒性。

# 三、项目文档

## 1. 词法分析测试

### 1.1. 测试数据

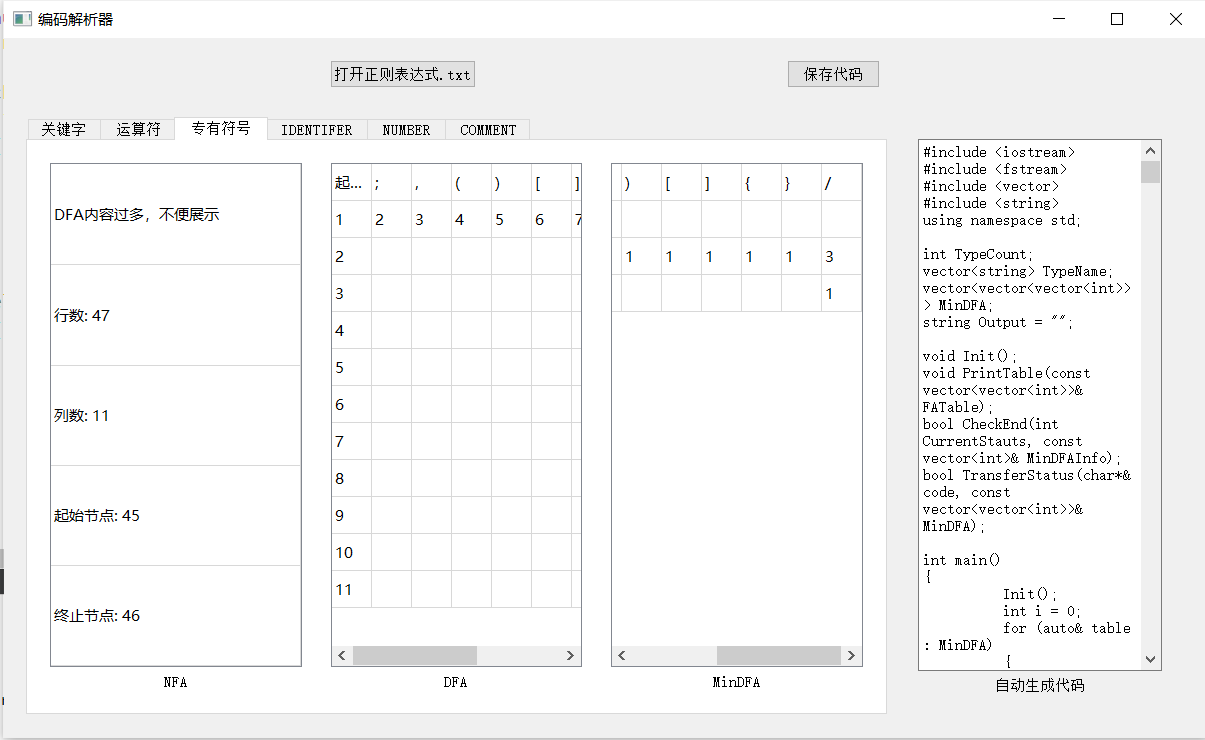
…\ComprehensiveExp1\TestData\C++.txt

|  |
| --- |
| KEYWORD  (asm)|(auto)|(bool)|(break)|(case)|(catch)|(char)|(class)|(const)|(const\_cast)|(continue)|(default)|(delete)|(do)|(double)|(dynamic\_cast)|(else)|(enum)|(explicit)|(export)|(extern)|(false)|(float)|(for)|(friend)|(goto)|(if)|(inline)|(int)|(long)|(mutable)|(namespace)|(new)|(operator)|(private)|(protected)|(public)|(register)|(reinterpret\_cast)|(return)|(short)|(signed)|(sizeof)|(static)|(static\_cast)|(struct)|(switch)|(template)|(this)|(throw)|(true)|(try)|(typeof)|(typeid)|(typename)|(union)|(unsigned)|(using)|(virtual)|(void)|(volatile)|(wchar\_t)|(while)  OPERATOR  +|-|\\*|/|%|>|<|=|(<=)|(>=)|(==)|(!=)|&|\||!|^|~|(++)|(--)  IDENTIFER  (\_|letter)(\_|letter|digit)\*  NUMBER  (digit)(digit)\*(\.(digit)(digit)\*)?  COMMENT  //.\* |

…\ComprehensiveExp1\TestData\SAMPLE2.txt

|  |
| --- |
| #include <iostream>  using namespace std;  // test  int main()  {  float \_i = 1.3;  cout << \_i;  return 0;  } |

### 1.2. 测试结果



### 1.3. 测试分析

#### 1.3.1. 检查最小DFA生成结果

#### 1.3.2. 判断注释

#### 1.3.3 判断标识符

#### 1.3.4. 判断数字

#### 1.3.5. 判断关键字

#### 1.3.6. 判断运算符

## 2. 语法分析测试

### 2.1. 测试数据

…\ComprehensiveExp2\TestData\C++.txt

|  |
| --- |
|  |

…\ComprehensiveExp2\TestData\SAMPLE2.cpp

|  |
| --- |
|  |

### 2.2. 测试结果

### 2.3. 测试分析

#### 2.3.1. 化简文法

输入的文法规则本就不存在冗余，因此这个步骤没有改变测试数据，测试结果符合预期。

#### 2.3.2. 消除直接左递归

|  |  |  |
| --- | --- | --- |
| 测试数据 | 预期结果 | 实际结果 |
|  |  |  |

#### 2.3.3. 消除间接左递归

#### 2.3.4. 消除直接左公因子

#### 2.3.5. 消除间接左公因子

#### 2.3.8. 产生语法树

这一步同时验证了产生first集和follow集和LL1分析表。

# 四、实验总结

# 五、参考文献

# 六、项目自评