1.$GrammarToFA$

2.$DetermineNFAorDFA$

3.$NFAToDFA$

4.$MinimizeDFA$

5.$RunMin-DFAModel$

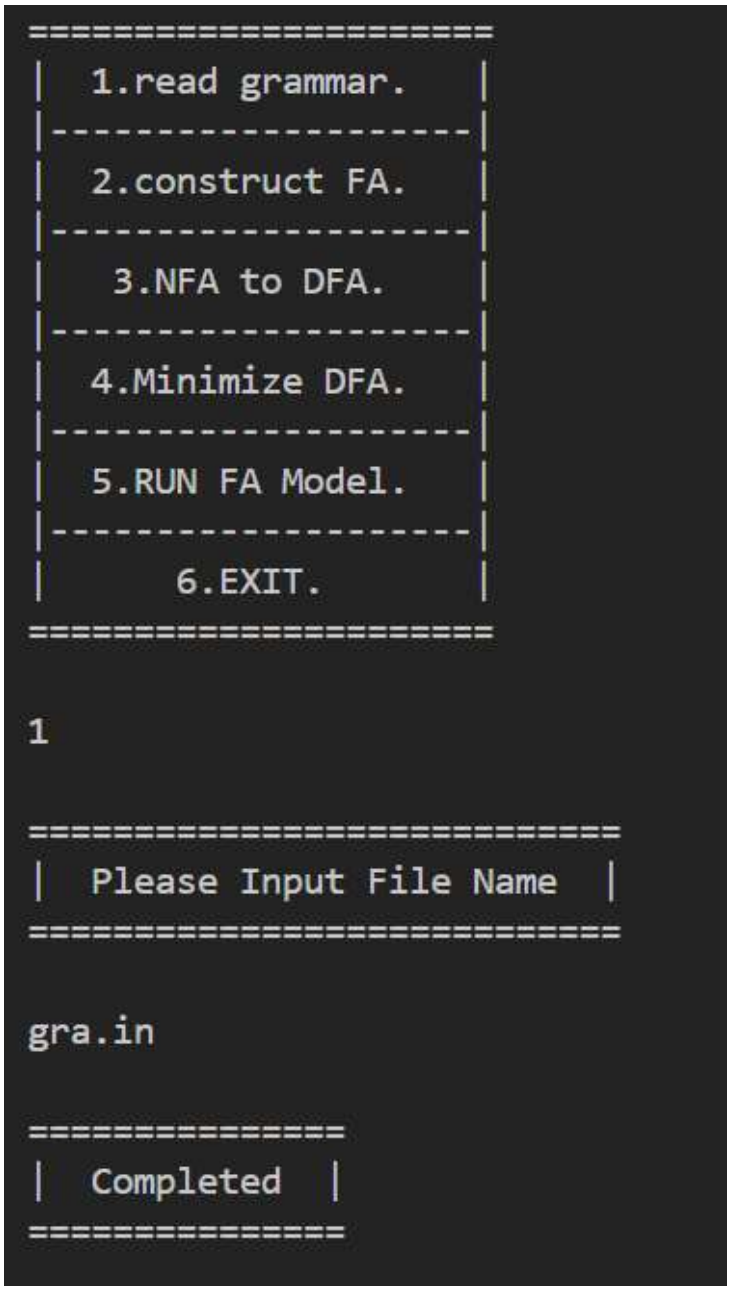
运行方式

$Windows$环境下，配置$g++$编译器，版本$g++(GCC)12.2.0$
$WORKDIR\ fa$下，运行$run.bat$ 完成编译和打开运行

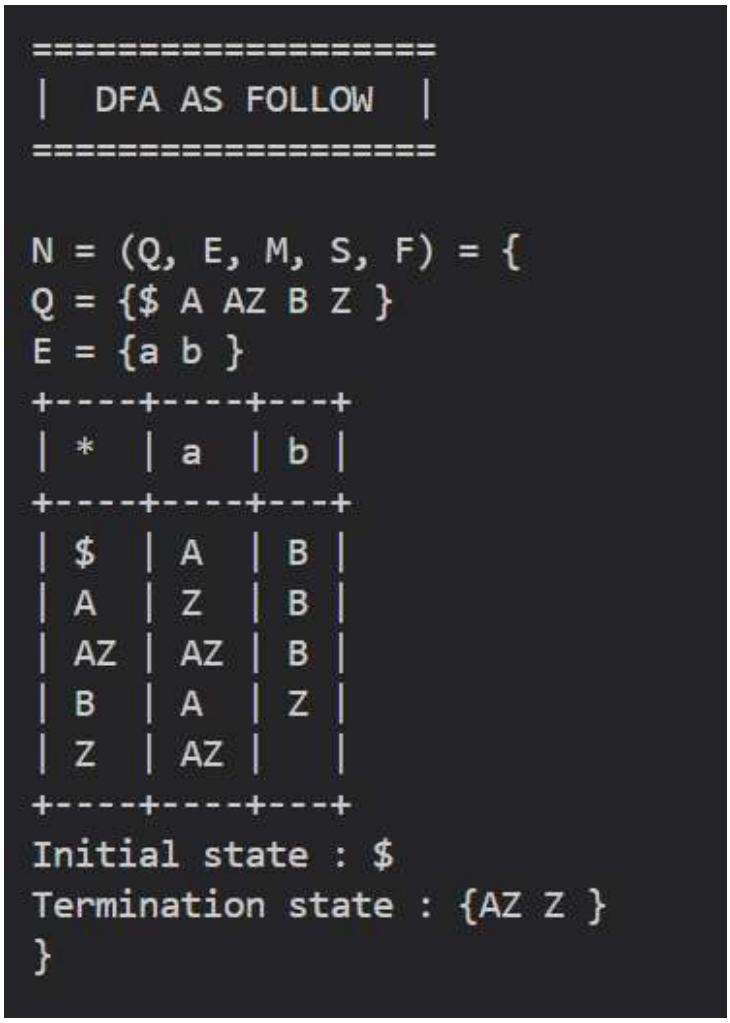
测试样例

输入过程为：

1
$gra.in$


1.read grammar.
2.construct FA.
3.NFA to DFA.
4.Minimize DFA.
5.RUN FA Model.
6.EXIT.
1
Please Input File Name
gra.in
Completed
2

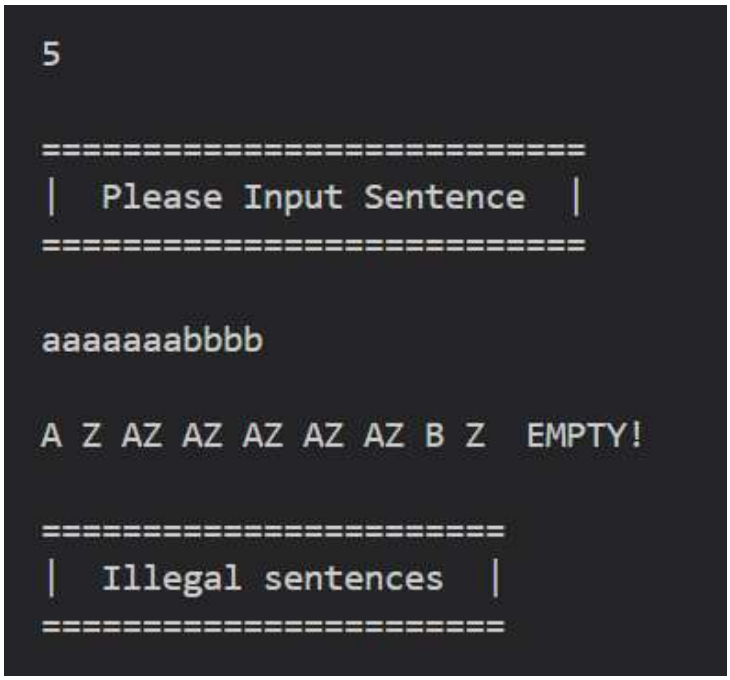
```
=================
|  FA AS FOLLOW  |
=================

N = (Q, E, M, S, F) = {
Q = {$ A B Z}
E = {a b }
+---+----+---+
| * | a  | b |
+---+----+---+
| $ | A  | B |
| A | Z  | B |
| B | A  | Z |
| Z | ZA |   |
+---+----+---+
Initial state : $
Termination state : Z
};

===========
|  Is NFA  |
===========
```


==================
| DFA AS FOLLOW |
==================
N = (Q, E, M, S, F) = {
Q = {$ A AZ B Z }
E = {a b }
+----+----+---+
| * | a | b |
+----+----+---+
$	A	B
A	Z	B
AZ	AZ	B
B	A	Z
Z	AZ	
+----+----+---+
Initial state : $
Termination state : {AZ Z }
}

```
============================
|  Split 'AZ' From {AZ Z }  |
============================

============================
|  Split '$' From {$ A B }  |
============================

==========================
|  Split 'A' From {A B }  |
==========================

======================
|  Min DFA AS FOLLW  |
======================

N = (Q, E, M, S, F) = {
Q = {AZ $ Z A B }
E = {a b }
+----+----+---+
| *  | a  | b |
+----+----+---+
| AZ | AZ | B |
| $  | A  | B |
| Z  | AZ |   |
| A  | Z  | B |
| B  | A  | Z |
+----+----+---+
Initial state : $
Termination state : {AZ Z }
}
```

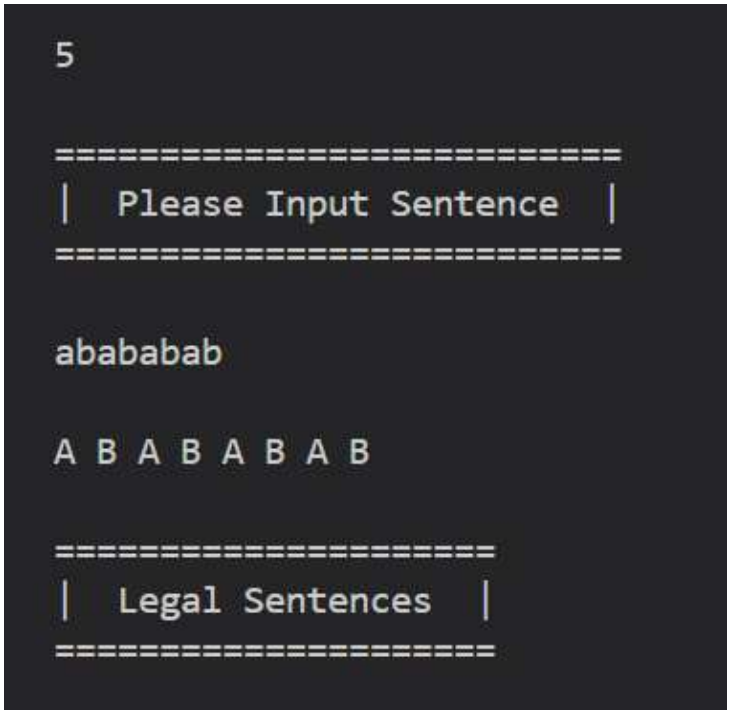
5

aaaaaaabbbb


5
==========
| Please Input Sentence |
==========
aaaaaaabbbb
A Z AZ AZ AZ AZ AZ B Z EMPTY!
==========
| Illegal sentences |
==========

5

abababab


5
==========
| Please Input Sentence |
==========
abababab
A B A B A B A B
==========
| Legal Sentences |
==========

1.*Grammar To FA*

```cpp
struct FA
{
  bool is_dfa;
  std::vector<char>  Q;
  std::vector<char> E;
  std::map<std::pair<std::string, char>, std::string> M;
  char S;
  char F;
};

struct DFA
{
  std::vector<std::string>  Q;
  std::vector<char> E;
  std::map<std::pair<std::string, char>, std::string> M;
  char S;
  std::vector<std::string> F;
};
```

NFA 和 DFA 结构体，即为五元组存储方式.

```cpp
std::ifstream in;

  auto read = [&] () {
    shows("Please Input File Name");
    std::string filename;
    std::cin >> filename;
    std::cout << "\n";
    in.open(filename);
  };

  std::vector<std::string> s;
  std::string grammar;

  std::set<char> ct;
  std::set<char> symbol;

  auto getG = [&] () {
    s.clear();
    ct.clear();

    while (in >> grammar) {
      std::cout << grammar << "\n";
      ct.insert(grammar[0]);
      s.push_back(grammar);
    }
    std::cout << "\n";

    int n = (int) ct.size();
    n += 1;
    std::vector<std::vector<std::array<char, 2>>> g(300);
    for (int i = 0; i < (int) s.size(); i++) {
      std::string sg = s[i];

      char l = sg[0];
      if (sg.length() == 5) {
        char N = sg[3];
        char T = sg[4];
        symbol.insert(T);
        g[N].push_back({l, T});
      }
      else if (sg.length() == 4) {
        char T = sg[3];
        symbol.insert(T);
        g['$'].push_back({l, T});
      }
    }
    return g;
  };
```

输入文件名指定正则文法读取路径，使用$c++$容器实例 $std::vector<std::vector<std::array<char,2>>>$ 存储正则文法，存储方式为带权有向图。

2.*Determine NFA or DFA*

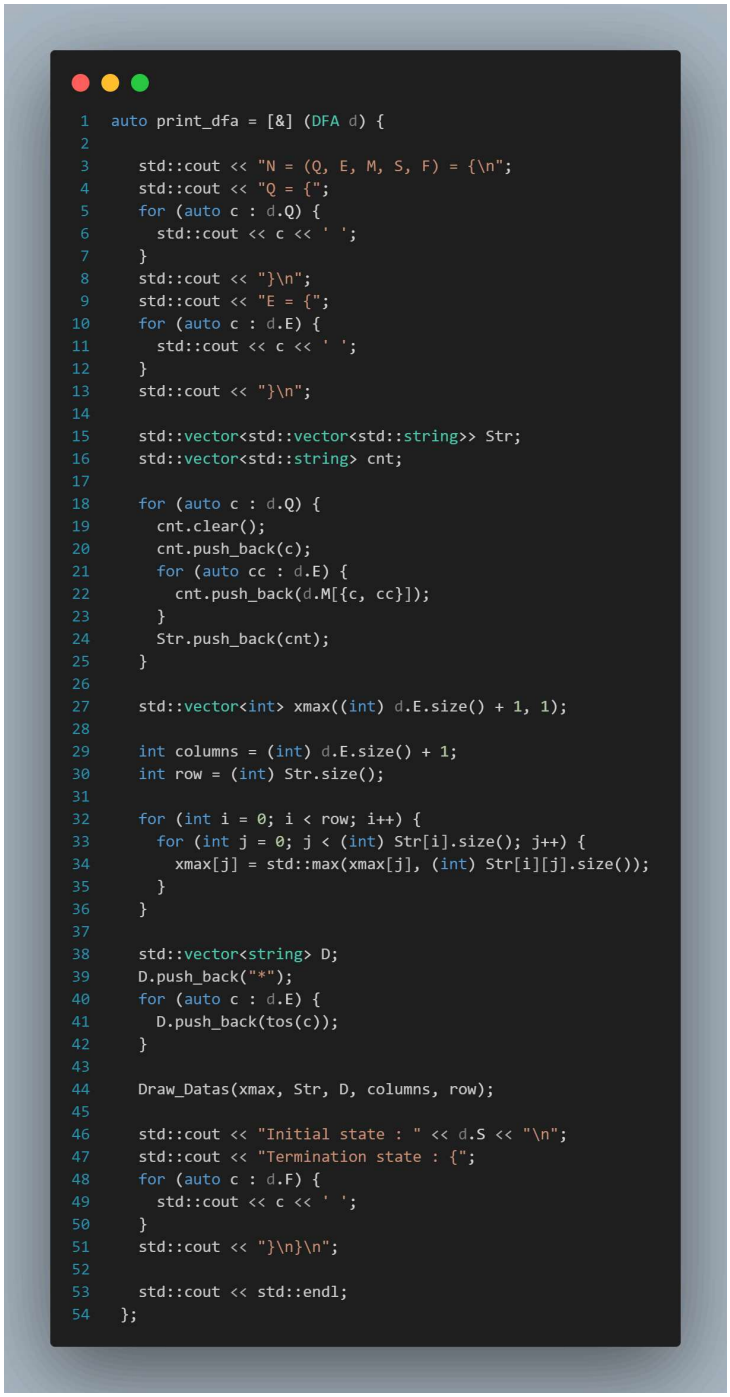
```
auto print = [&] (std::vector<std::vector<std::array<char, 2>>> g) {
    shows("FA AS FOLLOW");
    bool is_dfa = true;
    int n = (int) ct.size();
    std::cout << "N = (Q, E, M, S, F) = {\n";
    std::cout << "Q = {$";
    for (auto c : ct) {
      std::cout << ' ' << c;
    }
    std::cout << "}\n";

    // symbol
    std::cout << "E = {";
    for (auto c : symbol) {
      std::cout << c << ' ';
    }
    std::cout << "}\n";

    std::map<char, int> mp;
    std::map<char, int> smx;
    for (auto c : g) {
      if (c.empty()) {
        continue;
      }
      mp.clear();
      int mx = 0;
      for (auto cc : c) {
        mp[cc[1]] += 1;
        smx[cc[1]] = std::max(smx[cc[1]], mp[cc[1]]);
        if (smx[cc[1]] >= 2) {
          is_dfa = false;
        }
      }
    }
    std::vector<int> ssmx;
    ssmx.push_back(1);
    for (auto c : symbol) {
      ssmx.push_back(smx[c]);
    }
    std::vector<char> cct(ct.begin(), ct.end());
    int index = 1;
    std::vector<char> title(symbol.begin(), symbol.end());
    std::vector<std::string> ntitle;
    ntitle.push_back("*");
    for (auto t : title) {
      std::string cnt = "";
      cnt += t;
      ntitle.push_back(cnt);
    }
    std::vector<std::vector<std::string>> menu;
    for (int i = 0; i < 200; i++) {
      if (g[i].empty()) {
        continue;
      }
      std::vector<std::string> cnt;
      cnt.clear();
      std::string fr = "";
      fr += (char) (i);
      cnt.push_back(fr);
      std::map<char, std::vector<char>> mmp;
      for (auto cc : g[i]) {
        mmp[cc[1]].push_back(cc[0]);
      }
      for (auto cc : title) {
        std::string cs = "";
        for (auto ccc : mmp[cc]) {
          cs += ccc;
        }
        cnt.push_back(cs);
      }
      menu.push_back(cnt);
    }

    Draw_Datas(ssmx, menu, ntitle, (int) ntitle.size(), (int) menu.size());

    std::cout << "Initial state : $\n";
    std::cout << "Termination state : " << s[0][0] << "\n";

    std::cout << "};\n\n";

    std::map<std::pair<std::string, char>, std::string> r;

    int ind = 0;

    for (auto c : menu) {
      int indx = 0;
      for (int i = 1; i < (int) c.size(); i++) {
        r[{c[0], title[indx++]}] = c[i];
      }
    }

    FA f;
    f.is_dfa = is_dfa;
    f.Q = cct;
    f.E = title;
    f.M = r;
    f.S = '$';
    f.F = s[0][0];

    return f;
  };
```

自定义 $print$ 函数输出图关键信息，并将图传给$mysql$式表格输出函数，判断是否DFA方式为如果穿的值里有一列长度不为一，既有多个选择时，即为NFA。

DFA Output

```cpp
void Draw_line(vector<int> xmax,int columns) {
  for (int i = 0; i < columns; i++) {
    cout << "+-";
    for (int j = 0; j <= xmax[i]; j++) {
      cout << '-';
    }
  }
  cout << '+' << endl;
}

void Draw_Datas(vector<int> xmax, vector<vector<string>> Str,vector<string> D,int columns,int row) {
  Draw_line(xmax,columns);
  for (int i = 0; i < D.size(); i++) {
    cout << "| " << setw(xmax[i]) << setiosflags(ios::left) << setfill(' ') << D[i] << ' ';
  }
  cout << '|' << endl;
  Draw_line(xmax, columns);
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < columns; j++) {
      cout << "| " << setw(xmax[j]) << setiosflags(ios::left) << setfill(' ');
      cout << Str[i][j] << ' ';
    }
    cout << '|'<<endl;
  }
  Draw_line(xmax, columns);
}
```

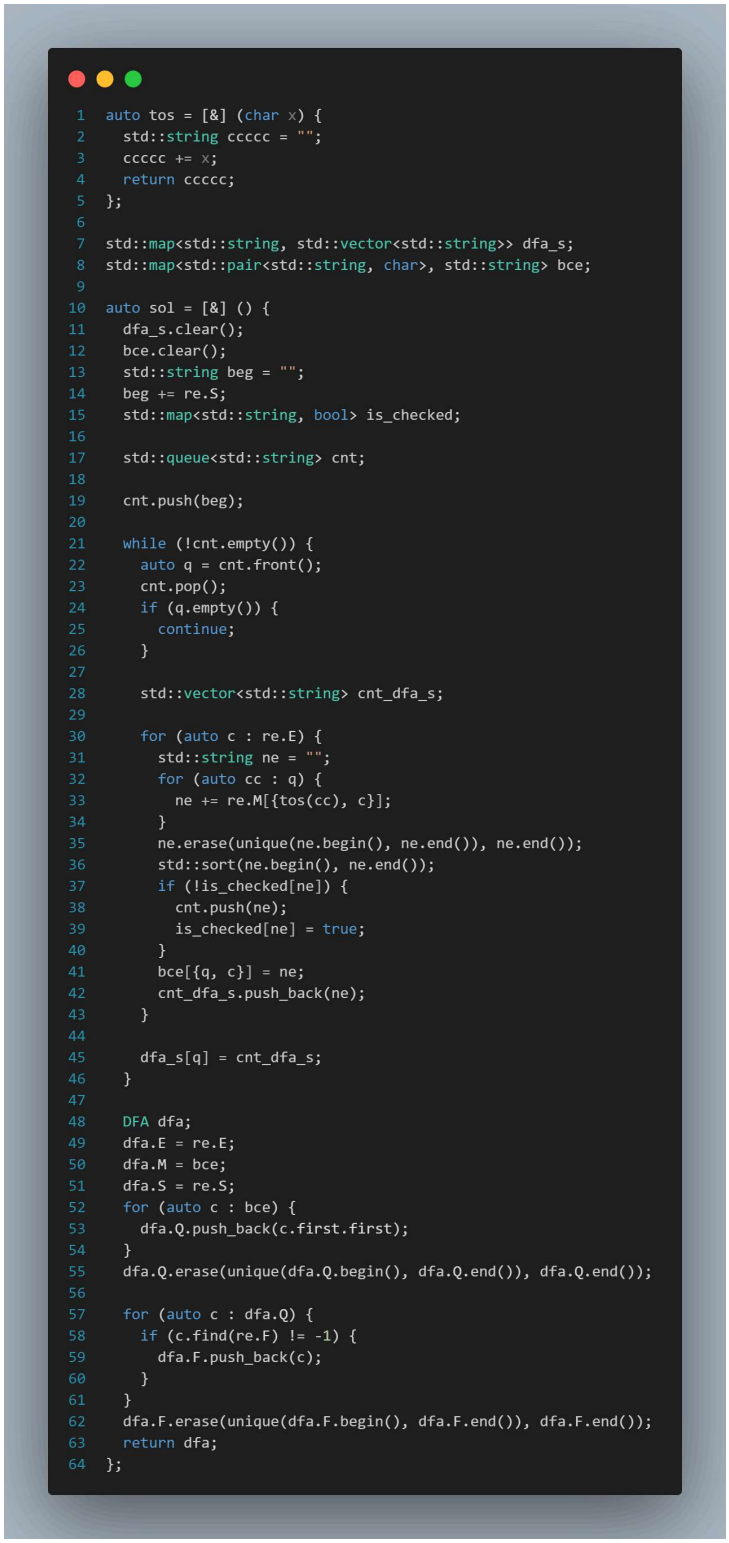
实现表格的格式化输出，借鉴$mysql$命令行数据库输出格式完成。



```cpp
auto print_dfa = [&] (DFA d) {

    std::cout << "N = (Q, E, M, S, F) = {\n";
    std::cout << "Q = {";
    for (auto c : d.Q) {
      std::cout << c << ' ';
    }
    std::cout << "}\n";
    std::cout << "E = {";
    for (auto c : d.E) {
      std::cout << c << ' ';
    }
    std::cout << "}\n";

    std::vector<std::vector<std::string>> Str;
    std::vector<std::string> cnt;

    for (auto c : d.Q) {
      cnt.clear();
      cnt.push_back(c);
      for (auto cc : d.E) {
        cnt.push_back(d.M[{c, cc}]);
      }
      Str.push_back(cnt);
    }

    std::vector<int> xmax((int) d.E.size() + 1, 1);

    int columns = (int) d.E.size() + 1;
    int row = (int) Str.size();

    for (int i = 0; i < row; i++) {
      for (int j = 0; j < (int) Str[i].size(); j++) {
        xmax[j] = std::max(xmax[j], (int) Str[i][j].size());
      }
    }

    std::vector<string> D;
    D.push_back("*");
    for (auto c : d.E) {
      D.push_back(tos(c));
    }

    Draw_Datas(xmax, Str, D, columns, row);

    std::cout << "Initial state : " << d.S << "\n";
    std::cout << "Termination state : {";
    for (auto c : d.F) {
      std::cout << c << ' ';
    }
    std::cout << "}\n}\n";

    std::cout << std::endl;
  };
```

实现 $Print-DFA$ 函数，作用为格式化输出DFA。

3.$NFA\ To\ DFA$

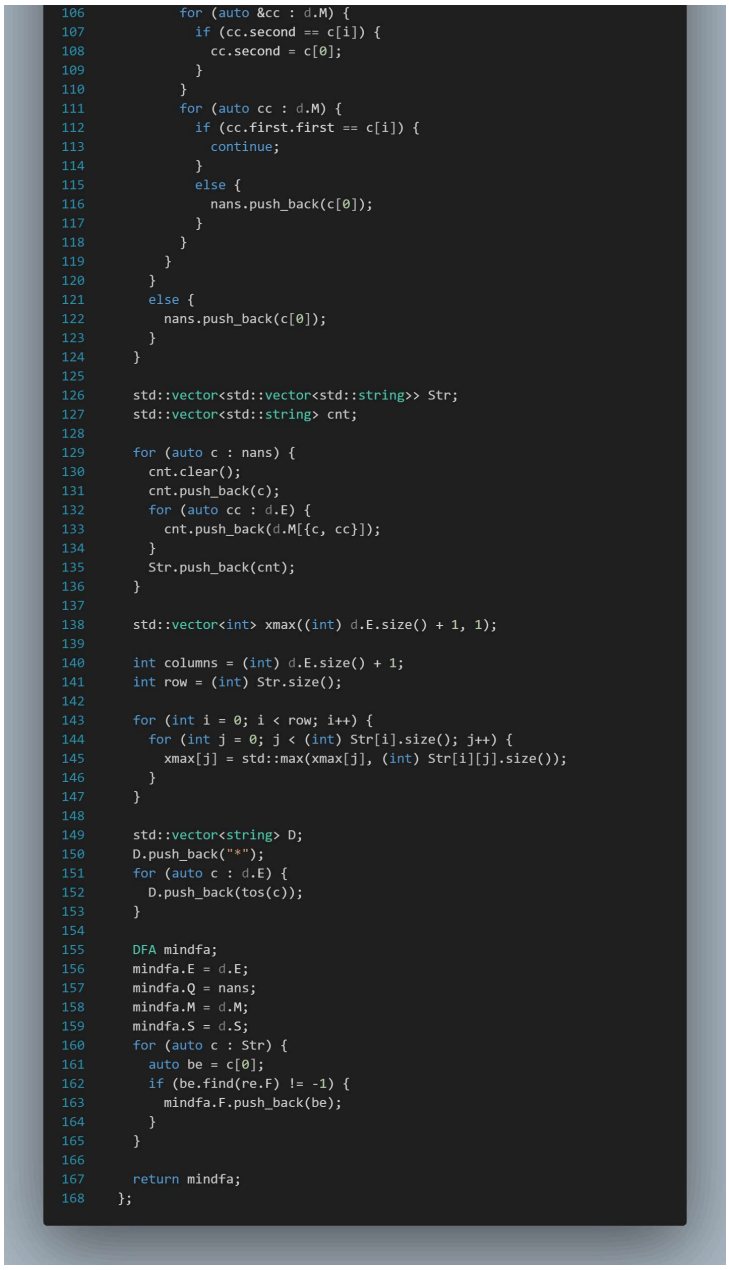


```cpp
auto tos = [&] (char x) {
  std::string ccccc = "";
  ccccc += x;
  return ccccc;
};

std::map<std::string, std::vector<std::string>> dfa_s;
std::map<std::pair<std::string, char>, std::string> bce;

auto sol = [&] () {
  dfa_s.clear();
  bce.clear();
  std::string beg = "";
  beg += re.S;
  std::map<std::string, bool> is_checked;

  std::queue<std::string> cnt;

  cnt.push(beg);

  while (!cnt.empty()) {
    auto q = cnt.front();
    cnt.pop();
    if (q.empty()) {
      continue;
    }

    std::vector<std::string> cnt_dfa_s;

    for (auto c : re.E) {
      std::string ne = "";
      for (auto cc : q) {
        ne += re.M[{tos(cc), c}];
      }
      ne.erase(unique(ne.begin(), ne.end()), ne.end());
      std::sort(ne.begin(), ne.end());
      if (!is_checked[ne]) {
        cnt.push(ne);
        is_checked[ne] = true;
      }
      bce[{q, c}] = ne;
      cnt_dfa_s.push_back(ne);
    }

    dfa_s[q] = cnt_dfa_s;
  }

  DFA dfa;
  dfa.E = re.E;
  dfa.M = bce;
  dfa.S = re.S;
  for (auto c : bce) {
    dfa.Q.push_back(c.first.first);
  }
  dfa.Q.erase(unique(dfa.Q.begin(), dfa.Q.end()), dfa.Q.end());

  for (auto c : dfa.Q) {
    if (c.find(re.F) != -1) {
      dfa.F.push_back(c);
    }
  }
  dfa.F.erase(unique(dfa.F.begin(), dfa.F.end()), dfa.F.end());
  return dfa;
};
```

实现 sol函数，实现NFA 到 DFA 的转换，具体办法为不断将新的状态添加到队列末尾，直至无新状态的产生，即为DFA ，返回值为 DFA 五元组。

4.*Minimize DFA*

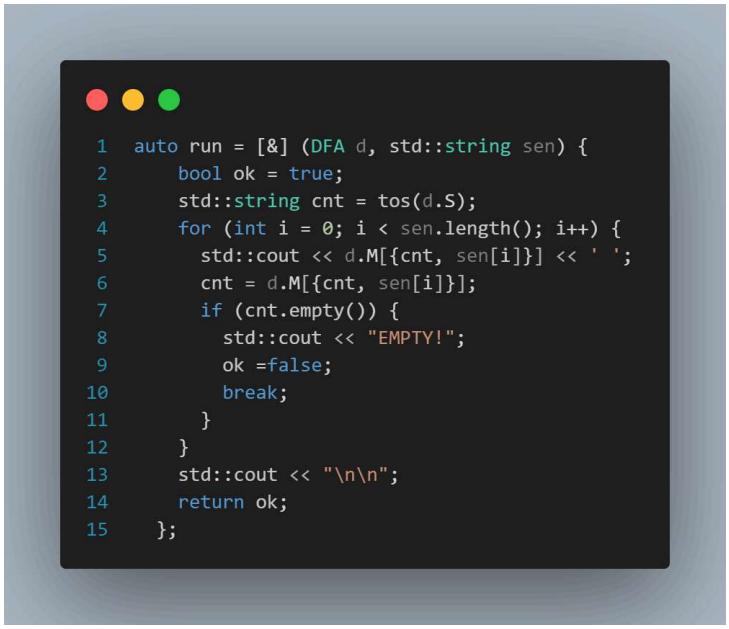
```
auto min = [&] (DFA d) {
    std::queue<std::vector<std::string>> state;

    state.push(d.F);
    std::vector<std::string> nonter;

    for (auto c : d.Q) {
      bool is_ter = false;
      for (auto cc : d.F) {
        if (cc == c) {
          is_ter = true;
          break;
        }
      }
      if (!is_ter) {
        nonter.push_back(c);
      }
    }
    nonter.erase(unique(nonter.begin(), nonter.end()), nonter.end());

    std::map<std::string, std::string> f;

    for (auto c : d.Q) {
      f[c] = c;
    }

    std::function<std::string(std::string)> get = [&] (std::string gs) {
      return gs == f[gs] ? gs : f[gs] = get(f[gs]);
    };

    for (int i = 1; i < (int) d.F.size(); i++) {
      f[get(d.F[i])] = get(d.F[0]);
    }

    for (int i = 1; i < (int) d.F.size(); i++) {
      f[get(nonter[i])] = get(nonter[0]);
    }

    state.push(nonter);

    std::map<std::vector<std::string>, bool> is_vised;

    std::vector<std::vector<std::string>> ans;

    ans.clear();

    while (!state.empty()) {
      auto q = state.front();
      state.pop();
      if (is_vised[q]) {
        continue;
      }
      else {
        is_vised[q] = true;
      }
      bool is_ok = true;
      for (int i = 1; i < (int) q.size(); i++) {
        for (auto c : d.E) {
          if (get(d.M[{q[0], c}]) != get(d.M[{q[i], c}])) {
            is_ok = false;
            break;
          }
        }
      }
      if (!is_ok) {
        std::vector<std::string> l;
        l.push_back(q[0]);
        std::vector<std::string> r;
        for (int i = 1; i < (int) q.size(); i++) {
          r.push_back(q[i]);
        }
        std::string rm = "Split \'" + q[0] + "\' From {";
        for (auto c : q) {
          rm += c;
          rm += ' ';
        }
        rm += "}";
        shows(rm);
        state.push(r);

        auto scnt = state;

        f.clear();
        for (auto c : d.Q) {
          f[c] = c;
        }
        while (!scnt.empty()) {
          auto cq = scnt.front();
          scnt.pop();
          for (int i = 1; i < (int) cq.size(); i++) {
            f[get(cq[i])] = get(cq[0]);
          }
        }
        ans.push_back(l);
      }
      else {
        ans.push_back(q);
      }
    }

    std::vector<std::string> nans;

    for (auto c : ans) {
      if ((int) c.size() >= 2) {
        for (int i = 1; i <= (int) c.size(); i++) {
```



```
          for (auto &cc : d.M) {
            if (cc.second == c[i]) {
              cc.second = c[0];
            }
          }
          for (auto cc : d.M) {
            if (cc.first.first == c[i]) {
              continue;
            }
            else {
              nans.push_back(c[0]);
            }
          }
        }
      }
      else {
        nans.push_back(c[0]);
      }
    }

    std::vector<std::vector<std::string>> Str;
    std::vector<std::string> cnt;

    for (auto c : nans) {
      cnt.clear();
      cnt.push_back(c);
      for (auto cc : d.E) {
        cnt.push_back(d.M[{c, cc}]);
      }
      Str.push_back(cnt);
    }

    std::vector<int> xmax((int) d.E.size() + 1, 1);

    int columns = (int) d.E.size() + 1;
    int row = (int) Str.size();

    for (int i = 0; i < row; i++) {
      for (int j = 0; j < (int) Str[i].size(); j++) {
        xmax[j] = std::max(xmax[j], (int) Str[i][j].size());
      }
    }

    std::vector<string> D;
    D.push_back("*");
    for (auto c : d.E) {
      D.push_back(tos(c));
    }

    DFA mindfa;
    mindfa.E = d.E;
    mindfa.Q = nans;
    mindfa.M = d.M;
    mindfa.S = d.S;
    for (auto c : Str) {
      auto be = c[0];
      if (be.find(re.F) != -1) {
        mindfa.F.push_back(be);
      }
    }

    return mindfa;
  };
```

min 函数，实现DFA的最小化，具体方法为将属性集合不断拆分为不可再分，再将相同状态的集合校区，即得到最小化的DFA。

5.$RunMin-DFAModel$



```
auto run = [&] (DFA d, std::string sen) {
    bool ok = true;
    std::string cnt = tos(d.S);
    for (int i = 0; i < sen.length(); i++) {
      std::cout << d.M[{cnt, sen[i]}] << ' ';
      cnt = d.M[{cnt, sen[i]}];
      if (cnt.empty()) {
        std::cout << "EMPTY!";
        ok =false;
        break;
      }
    }
    std::cout << "\n\n";
    return ok;
  };
```

run 函数，实现句子在DFA上的运行，判断是否可以产生这个句子，并返回真值。