```cpp
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <cctype>
using namespace std;
char data[256];

// trim from start
static inline std::string &ltrim(std::string &s) {
    s.erase(s.begin(), std::find_if(s.begin(), s.end(),
            std::not1(std::ptr_fun<int, int>(std::isspace))));
    return s;
}

// trim from end
static inline std::string &rtrim(std::string &s) {
    s.erase(std::find_if(s.rbegin(), s.rend(),
            std::not1(std::ptr_fun<int, int>(std::isspace))).base(), s.end())
    return s;
}

// trim from both ends
static inline std::string &trim(std::string &s) {
    return ltrim(rtrim(s));
}

static inline vector<string> split(const string &s, const string pat) {
  vector<string> v;
  if(s.empty()) {
    return v;
  }
  int i=0, j=0;
  while(i<s.size() && (j = s.find(pat, i)) != string::npos) {
    if(i!=j) {
      auto tok = s.substr(i, j-i);
      v.push_back(tok);
    }
    v.push_back(pat);
    i = j+pat.size();
  }
  if(i<s.size() && i!=j) {
    auto tok = s.substr(i, j-i);
    if(!tok.empty()) {
      v.push_back(tok);
    }
  }

  return v;
}

int main()
{
  string s = R"(
    //@TCEMBED

  )";
  stringstream ss(s);
  vector<string> v;
  while(!ss.eof()) {
    ss.getline(data, 256);
    string str(data);
    trim(str);
    if(!str.empty()) {
      if(!(str[0] == str[1] && str[0] == '/')) {
        v.push_back(str);
      }
    }
  }
  if(v.size() == 0) {
    cerr << "You have not entered anything." << endl;
  } else {
    string raw;
    vector<string> vs;
    for(auto &x : v) {
      stringstream st(x);
      while(!st.eof()) {
        string temp;
        st >> temp;
        xint i=0, j=0;
        while(i < temp.size()) {
          switch(temp[i]) {
            case '{':
              if(i != j) {
                vs.push_back(temp.substr(j, i-j));
              }
              vs.push_back("{");
              j = i+1;
              break;
            case ',':
              if(i != j) {
                vs.push_back(temp.substr(j, i-j));
              }
              vs.push_back(",");
              j = i+1;
              break;
            case '}':
              if(i != j) {
                vs.push_back(temp.substr(j, i-j));
              }
              vs.push_back("}");
              j = i+1;
              break;
            case ';':
              if(i != j) {
                vs.push_back(temp.substr(j, i-j));
              }
              vs.push_back(";");
              j = i+1;
              break;
            default:
              break;
          }
          i++;
        }
        if(i != j) {
          vs.push_back(temp.substr(j, i-j));
        }
        raw += temp + " ";
        // raw += temp;
      }
    }
    int i=vs.size()-1;
    while(i-1>=0 && vs[i] == vs[i-1] && vs[i] == ";") {
      vs.pop_back();
      i--;
    }
    if(raw.find("enum") == string::npos) {
      cerr << "You have not declared enum type." << endl;
      return 1;
    } else if(raw.find("CourseMode") == string::npos) {
      cerr << "CourseMode is not an enum type." << endl;
      return 1;
    } else if(raw.find("RESIDENTIAL") == string::npos) {
      cerr << "RESIDENTIAL is not an identifier of CourseMode." << endl;
      return 1;
    } else if(raw.find("ONLINE") == string::npos) {
      cerr << "ONLINE is not an identifier of CourseMode." << endl;
      return 1;
    } else if(raw.find("HYBRID") == string::npos) {
      cerr << "HYBRID is not an identifier of CourseMode." << endl;
      return 1;
    } else if(vs.size() != 10) {
      cerr << "incorrect" << endl;
      return 1;
    } else {
      if(vs[0] != "enum" || vs[2] != "{" || vs[4] != "," || vs[6] != "," || v
        cerr << "Invalid syntax" << endl;
        return 1;
      } else if(vs[1] != "CourseMode") {
        cerr << "incorrect" << endl;
        return 1;
      } else if(vs[3] != "RESIDENTIAL" || vs[5] != "ONLINE" || vs[7] != "HYBR
        cerr << "Order of the identifiers should be maintained." << endl;
        return 1;
      } else {
        cout << "correct" << endl;
        return 0;
      }
    }
  }
  return 1;
}
```