①

```
class Year {
private:
    int year;
    std::vector<int> months;
    bool vis;
public:
    void set_vis(); void set_nevis(); int month_num(int);
    Year(int year): year(year) {
        if(year %4 == 0) vis = true;
        if(vis) set_vis();
        else set_nevis();
    };
    Year & operator=(Year new) { year = new.year;
                                 return *this; } };
void Year::set_vis() {
    months = {31, 29, 31, 30, ... };
}
void Year::set_nevis() {
    months = {31, 28, 31, .... };
}
int i;
int Year::month_num(int day) {
    for(i=0; day > months[i]; i++) day -= months[i];
    return i;
}
```

```
int main() {
    Year year(2000);
    std::cout << year.month_num/20);
    return 0;
}
```

(2.1)

```
#include <iostream>
#include <list>
#include <algoritm>
#include <string>

int main() {
    std::list<std::string> data = { "abba", ... };
    for(std::list<std::string>::const_iterator iter = data.cbegin();
        iter != data.cend(); ++iter)
    if (std::equal((*iter).begin(), (*iter).begin+(*iter).size/2,
            (*iter).rbegin()))
        std::cout << "Palindrome - " << *iter << <<std::endl;
}
```

(2.2) -

```
bool is_palindrom(const std::string &str) {
    auto left = str.begin();
    auto right = str.end();
    while (left < right) {
        if (*left != *right) return false;
        ++left;
        --right;
    }
    return true;
}
```

```
void print_palindrom(const std::vector<std::string> &data) {
    auto found_it = std::find_if(data.begin(), data.end(),
    [&data](const std::string &st) { return is_palindrom(str);
    });
    while(found_it != data.end()) {
        std::cout << *found_it << std::endl;
        auto found_it = std::find_if(found_it+1,
            data.end(), [&data](const std::string &str) {
                return is_palindrom(str);
            });
    }
}
```

③

```
#include <iostream>
#include <mutex>
#include <thread>
std::mutex mtx;
void print(int flag) {
    mtx.lock();
    for(int i=0; i< 10; i++) {
        if(i%2==flag)
            std::cout << i << "  ";
        else continue;
    }
    mtx.unlock();
}

int main() {
    thread th1(print, true);
    thread th2(print, false);
    th1.join();
    th2.join();
    return 0;
}
```