C++ challenge

Note: Don't care if the answers compile or not (ex due to missing includes or typos, etc).

#1

Given the following *partial* class declarations:

```
class RecognizerModel {
    public:
        RecognizerModel() {}; // doesn't do anything
        ~RecognizerModel();   // quick destroy (doesn't block)

        // train_with_latest_data - blocks for several seconds until complete
        void train_with_latest_data();

        // identify_person - Caller must ensure that train_with_latest_data() was already called/finished
        //   identify_person() is thread-safe but may not be called in parallel to train_with_latest_data()
        Person identify_person(PersonInfo info);
};

/*
  // Typical RecognizerModel use example:
  RecognizerModel model;
  model.train_with_latest_data();

  // Model is now ready to be used to identify people from many threads
  startThreads();
  // Mulitple threads will now call model.identify_person() in parallel to identify people
*/

class Recognizer {
    public:
        Recognizer();
        ~Recognizer();

        // queue_retrain - should queue (or start) another call to train_with_latest_data(), but return immediately
        void queue_retrain();

        // identify_person - should call RecognizerModel::identify_person() on most recently trained model
        Person identify_person(PersonInfo info);

    private:
        unique_ptr<RecognizerModel> m_model;
};

// Single global Recognizer instance
Recognizer g_recognizer;
```

Consider: Every time someone is spotted by our camera, we potentially create a new person profile or add photos to an existing person profile. Our face recognition model needs to be retrained at that point so that it can make use of the new data to recognize people going forward, so some thread calls:

```
g_recognizer.queue_retrain();
```

Implement `Recognizer` (*not* `RecognizerModel`). Feel free to add to its class declaration . Notes:

- Training a model via `train_with_latest_data()` takes several seconds
- Different threads (ex. corresponding to different cameras) may call `g_recognizer.queue_retrain()` frequently, simultaneously, etc.
- `g_recognizer.queue_retrain()` must not block (practically).
- Multiple parallel calls to `g_recognizer.queue_retrain()` should never result in more than one thread executing `RecognizerModel::train_with_latest_data()` at the same time.
- If `RecognizerModel::train_with_latest_data()` is in progress, and one or more calls to

`g_recognizer.queue_retrain()` are made, then another call to `RecognizerModel::train_with_latest_data()` will need to occur after the first is complete (since the data has changed again).

- While retraining, the system must continue to be able to recognize people with the previously trained model. **I.e. `g_recognizer.identify_person()` must not block (practically) and must return a valid prediction from the most recently trained model.**
- Different threads (ex. corresponding to different cameras) may call `g_recognizer.identify_person()` frequently, simultaneously, etc.

#2

What is this code trying to accomplish? Any problems with it?

```
vector<string> tag_types;
tag_types.push_back("Known");
tag_types.push_back("VIP");
tag_types.push_back("Suspect");
tag_types.push_back("Employee");
...
tag_types.push_back("Friend");
// tag_types.size() == 200

...

try {
    auto p = g_recognizer.identify_person(inf);
    if(!p) {
        throw "Person not identified";
    }
    for(unsigned i = 0; i < tag_types.size(); ++i) {
        if(p.tags & uint64_t(pow(2,i))) {
            cout << tag_types[i] << endl;
        }
    }
} catch(const exception& e) {
    ...
}
```