```cpp
#pragma once

/// ¿¿¿¿¿BAL dataset
class BALProblem {
public:
    /// load bal data from text file
    explicit BALProblem(const std::string &filename, bool use_quaternions = false);

    ~BALProblem() {
        delete[] point_index_;
        delete[] camera_index_;
        delete[] observations_;
        delete[] parameters_;
    }

    /// save results to text file
    void WriteToFile(const std::string &filename) const;

    /// save results to ply pointcloud
    void WriteToPLYFile(const std::string &filename) const;

    void Normalize();

    void Perturb(const double rotation_sigma,
                 const double translation_sigma,
                 const double point_sigma);

    int camera_block_size() const { return use_quaternions_ ? 10 : 9; }

    int point_block_size() const { return 3; }

    int num_cameras() const { return num_cameras_; }

    int num_points() const { return num_points_; }

    int num_observations() const { return num_observations_; }

    int num_parameters() const { return num_parameters_; }

    const int *point_index() const { return point_index_; }

    const int *camera_index() const { return camera_index_; }

    const double *observations() const { return observations_; }

    const double *parameters() const { return parameters_; }

    const double *cameras() const { return parameters_; }

    const double *points() const { return parameters_ + camera_block_size() * num_cameras_; }

    /// camera¿¿¿¿¿¿
    double *mutable_cameras() { return parameters_; }

    double *mutable_points() { return parameters_ + camera_block_size() * num_cameras_; }

    double *mutable_camera_for_observation(int i) {
        return mutable_cameras() + camera_index_[i] * camera_block_size();
    }

    double *mutable_point_for_observation(int i) {
        return mutable_points() + point_index_[i] * point_block_size();
    }

    const double *camera_for_observation(int i) const {
        return cameras() + camera_index_[i] * camera_block_size();
    }
```

```
68
69      const double *point_for_observation(int i) const {
70          return points() + point_index_[i] * point_block_size();
71      }
72
73  private:
74      void CameraToAngelAxisAndCenter(const double *camera,
75                                      double *angle_axis,
76                                      double *center) const;
77
78      void AngleAxisAndCenterToCamera(const double *angle_axis,
79                                      const double *center,
80                                      double *camera) const;
81
82      int num_cameras_;
83      int num_points_;
84      int num_observations_;
85      int num_parameters_;
86      bool use_quaternions_;
87
88      int *point_index_;      // ¿¿observation¿¿¿point index
89      int *camera_index_;     // ¿¿observation¿¿¿camera index
90      double *observations_;
91      double *parameters_;
92  };
```