Задача 1(Трекер задач)

Реализуйте класс TeamTasks, позволяющий хранить статистику по статусам задач команды разработчиков:

```
// Перечислимый тип для статуса задачи
enum class TaskStatus {
NEW,          // новая
IN_PROGRESS,  // в разработке
TESTING,      // на тестировании
DONE          // завершена
};

// Объявляем тип-синоним для map<TaskStatus, int>,
// позволяющего хранить количество задач каждого статуса
using TasksInfo = map<TaskStatus, int>;

class TeamTasks {
public:
// Получить статистику по статусам задач конкретного разработчика
const TasksInfo& GetPersonTasksInfo(const string& person) const;

// Добавить новую задачу (в статусе NEW) для конкретного разработчитка
void AddNewTask(const string& person);

// Обновить статусы по данному количеству задач конкретного разработчика,
// подробности см. ниже
tuple<TasksInfo, TasksInfo> PerformPersonTasks(
const string& person, int task_count);
};
```

Метод PerformPersonTasks должен реализовывать следующий алгоритм:

1. Рассмотрим все не выполненные задачи разработчика person.
2. Упорядочим их по статусам: сначала все задачи в статусе NEW, затем все задачи в статусе IN_PROGRESS и, наконец, задачи в статусе TESTING.
3. Рассмотрим первые task_count задач и переведём каждую из них в следующий статус в соответствии с естественным порядком: NEW → IN_PROGRESS → TESTING → DONE.
4. Вернём кортеж из двух элементов: информацию о статусах обновившихся задач (включая те, которые оказались в статусе DONE) и информацию о статусах остальных не выполненных задач.

Гарантируется, что task_count является положительным числом. Если task_count превышает текущее количество невыполненных задач разработчика, достаточно считать, что task_count равен количеству невыполненных задач: дважды обновлять статус какой-либо задачи в этом случае не нужно.

Кроме того, гарантируется, что метод GetPersonTasksInfo не будет вызван для разработчика, не имеющего задач.

Пример работы метода PerformPersonTasks

Предположим, что у конкретного разработчика имеется 10 задач со следующими статусами:

- NEW — 3

- IN_PROGRESS — 2
- TESTING — 4
- DONE — 1

Поступает команда PerformPersonTasks с параметром task_count = 4, что означает обновление статуса для 3 задач с NEW до IN_PROGRESS и для 1 задачи с IN_PROGRESS до TESTING. Таким образом, новые статусы задач будут следующими:

- IN_PROGRESS — 3 обновлённых, 1 старая
- TESTING — 1 обновлённая, 4 старых
- DONE — 1 старая

В этом случае необходимо вернуть кортеж из 2 словарей:

- Обновлённые задачи: IN_PROGRESS — 3, TESTING — 1.
- Не обновлённые задачи, исключая выполненные: IN_PROGRESS — 1, TESTING — 4.

Словари не должны содержать лишних элементов, то есть статусов, которым соответствует ноль задач.

Примечание

Обновление словаря одновременно с итерированием по нему может привести к непредсказуемым последствиям. При возникновении такой необходимости рекомендуется сначала в отдельном временном словаре собрать информацию об обновлениях, а затем применить их к основному словарю.

Пример кода

```
1  // Принимаем словарь по значению, чтобы иметь возможность
2  // обращаться к отсутствующим ключам с помощью [] и получать 0,
3  // не меняя при этом исходный словарь
4  void PrintTasksInfo(TasksInfo tasks_info) {
5  cout << tasks_info[TaskStatus::NEW] << " new tasks" <<
6  ", " << tasks_info[TaskStatus::IN_PROGRESS] << " tasks in progress" <<
7  ", " << tasks_info[TaskStatus::TESTING] << " tasks are being tested" <<
8  ", " << tasks_info[TaskStatus::DONE] << " tasks are done" << endl;
9  }
10
11 int main() {
12 TeamTasks tasks;
13 tasks.AddNewTask("Ilia");
14 for (int i = 0; i < 3; ++i) {
15 tasks.AddNewTask("Ivan");
16 }
17 cout << "Ilia's tasks: ";
18 PrintTasksInfo(tasks.GetPersonTasksInfo("Ilia"));
19 cout << "Ivan's tasks: ";
20 PrintTasksInfo(tasks.GetPersonTasksInfo("Ivan"));
21
22 TasksInfo updated_tasks, untouched_tasks;
23
24 tie(updated_tasks, untouched_tasks) =
25 tasks.PerformPersonTasks("Ivan", 2);
26 cout << "Updated Ivan's tasks: ";
27 PrintTasksInfo(updated_tasks);
```

```
28 cout << "Untouched Ivan's tasks: ";
29 PrintTasksInfo(untouched_tasks);
30
31 tie(updated_tasks, untouched_tasks) =
32 tasks.PerformPersonTasks("Ivan", 2);
33 cout << "Updated Ivan's tasks: ";
34 PrintTasksInfo(updated_tasks);
35 cout << "Untouched Ivan's tasks: ";
36 PrintTasksInfo(untouched_tasks);
37
38 return 0;
39 }
```