Министерство науки и высшего образования

Московский Авиационный институт
(Национальный исследовательский
университет)

Институт №8
«Информационные технологии и
прикладная математика»

Кафедра 813
«Компьютерная математика»

Курсовая работа по дисциплине «Фундаментальные алгоритмы»

Тема: «Исследование и реализация структур данных: список и АВЛ-дерево»

Студент	Заиц Георгий Тиодорович
Группа	М8О-211Б-19
Преподаватель	Романенков Александр Михайлович
Дата	20 мая 2021г.

Оценка: _____

Подпись преподавателя: _____

Подпись студента: _____

Москва 2021

Содержание

1 Описание задачи

Разработайте приложение для обработки данных переписи населения. Придумайте форму опроса (или найдите готовую в сети).

Основным требованием является достаточный объем вопросов (не менее 20), на которые подразумеваются различные виды ответа: в виде числа, с единичным или множественным выбором из предложенных вариантов, либо в виде развернутого текста. Ваше приложение должно обеспечить возможность хранения и обработки результатов опроса. Количество хранимых заполненных форм достаточно велико ($>$ 500'000 шт.).

Функционал приложения должен обеспечить возможность поиска форм по различным критериям (минимум 5 различных критериев), статистическую обработку результатов с подсчетом абсолютных и относительных частот для каждого пункта, который могли выбрать участники опроса. Реализуйте возможность добавления результата опроса, удаления результата опроса.

Для демонстрации работы реализуйте генератор ответов, выдающий случайным образом заполненные формы опроса (генерация должна быть реализована посредством паттерна "фабричный метод").

Реализуйте функционал обработки данных таким образом, чтобы тип коллекции, в которой будут храниться ваши данные, являлся параметром.

Продемонстрируйте обработку данных с использованием std::list и собственной реализации АВЛ-дерева (с компаратором-стратегией).

2 Решение поставленной задачи

Решение задачи разбито на несколько заголовочных файлов. Последовательное описание каждого представлено ниже.

2.1 PopulationCensus.h

В данном файле реализован класс *PopulationCensus.h*, в котором представлены все поля формы (39 полей содержат значения: в виде чис-

ла, с единичным и множественным выбором из предложенных вариантов, строки с ответом).

Для полей с единичным и множественным выбором используются *enum* и контейнер *std::set<enum>* соответственно.

В приватных полях данного класса также представлена генерация для каждого из значений. Генерация происходит с использованием библиотеки *random*. Пример генерации одного из полей формы:

```
static typesOfLivelihoods generateTypesOfLivelihoods()
{
    static std::random_device rd;
    std::default_random_engine generator(rd());
    std::uniform_int_distribution<unsigned int>
     ↪  distribution(0, 13);
    auto answer =
     ↪  static_cast<typesOfLivelihoods>(distribution(generator));
    return answer;
}
```

Генерация налоговых форм реализована с использованием паттерна "Фабричный метод".

Фабричный метод (*Factory Method*) - это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.
Представлен статической функцией *randomGenerateObjectPopulationCensus()*, которая возвращает умный указатель на созданную налоговую форму. В данной функции для каждого поля формы вызывается генерация, таким образом функция возвращает указатель на полностью сгенерированную форму.

Реализованы функции подстановки строк вместо перечисляемых типов, а также перегружен оператор вставки в поток для всех полей класса, чтобы пользователь мог создавать собственные формы.

Для класса *PopulationCensus* реализованы дружественные классы компараторов, которые в дальнейшем используются для поиска данных

в форме.

Для пяти полей, по которым в дальнейшем будет производиться поиск реализованы методы доступа (сеттеры), посредством которых, в форме будет заполняться одно выбранное поле.

2.2 customTypes.h

В данный файл вынесены реализации всех перечислений, структуры и функций для вывода значений *enum* строкой.

Для полей со множественным выбором используется контейнер из стандартной библиотеки шаблонов *set*, в котором хранятся все выбранные пользователем (или сгенерированные) значения.

Пример реализации одного из перечислений.

```
1 enum positionAtWork
2 {
3     employed, /* работающий по найму */
4     businessOwner, /* владелец(совладелец) собственного
      ↪  предприятия(дела) */
5     individualEntrepreneur, /* индивидуальный предприниматель
      ↪  */
6     selfEmployed, /* самозанятый */
7     familyBusiness, /* помогающий на семейном предприятии */
8     otherwise /* иное */
9 };
```

Также представлены пять функций для заполнения одного поля в форме, которые вызываются в поиске и удалении. Удалении происходит по следующему алгоритму, пользователь вводит значение и начинается поиск заданного поля среди форм, если он был найден, производится удаление. Для поиска пользователь также задает значение одного из полей, все найденные значения заносятся в коллекцию *list*.

2.3 decorator.h

В данном заголовочном файле реализован шаблонный класс *Decorator* посредством применения паттерна "Декоратор". Декоратор *Decorator* пред-

ставляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Декоратор — это структурный паттерн проектирования, который позволяет динамически подключать к объектам новую функциональность. Благодаря реализации этого класса обращение к собственной и к контейнеру из стандартной библиотеки шаблонов имеет одинаковый интерфейс.

В классе реализованы виртуальные функции добавления, удаления, поиска, статистической обработки информации (для каждого типа данных реализована своя функция подсчета абсолютных и относительных частот, возвращающая контейнер *map*). Дочерние классы от данного будут иметь собственные реализации этих функции.

```
template <class TypeData>
class Decorator
{
public:
    Decorator() = default;
    virtual void add(std::shared_ptr<TypeData> data) = 0;
    virtual void remove() = 0;
    virtual void countingStatistics() = 0;
    virtual std::list<std::shared_ptr<TypeData>> find() = 0;
    virtual ~Decorator() = default;
};
```

Тип *shared_ptr* — это смарт-указатель в стандартной библиотеке C++, который предназначен для ситуаций, когда управлять временем существования объекта в памяти требуется нескольким владельцам. После инициализации указателя *shared_ptr* его можно копировать, передавать по значению в аргументах функций и присваивать другим экземплярам *shared_ptr*. Все экземпляры указывают на один и тот же объект и имеют общий доступ к одному "блоку управления который увеличивает и уменьшает счетчик ссылок, когда указатель *shared_ptr* добавляется, выходит из области действия или сбрасывается. Когда счетчик ссылок достигает нуля, блок управления удаляет ресурс в памяти и самого себя.

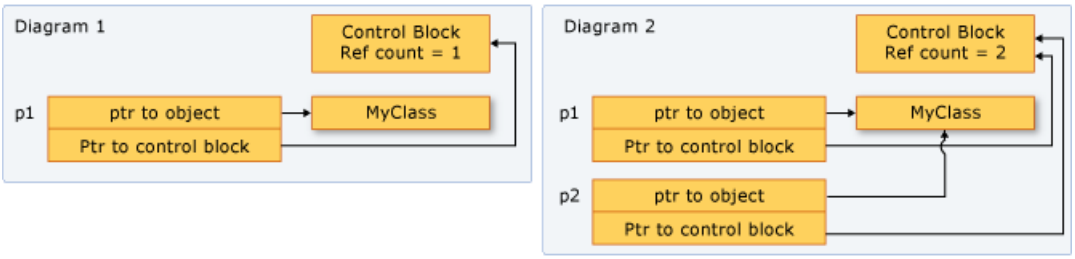

Рис. 1: На схеме выше показано несколько экземпляров *shared_ptr*, указывающих на одно расположение в памяти.

2.4 strategy.h

В данном заголовочном файле реализован шаблонный класс *Strategy* посредством применения паттерна "Стратегия".

Паттерн "Стратегия"(*Strategy*) представляет шаблон проектирования, который определяет набор алгоритмов, инкапсулирует каждый из них и обеспечивает их взаимозаменяемость. В зависимости от ситуации мы можем легко заменить один используемый алгоритм другим. При этом замена алгоритма происходит независимо от объекта, который использует данный алгоритм.

```
1 template <typename T>
2 class Strategy{
3 public:
4     virtual ~Strategy() = default;
5     virtual int compare(const T& left, const T& right) const =
      ↪  0;
6 };
```

2.5 decoratorList.h

В данном файле класс *DecoratorList* наследуется от класса *Decorator*. В приватном поле класса находится двусвязный список стандартной коллекции шаблонов, в котором хранятся умные указатели на анкеты.

Все функции реализуются посредством использования итераторов и встроенных в stl-коллекцию методов. В функции удаления пользователь выбирает, удаляются все вхождения или первое, также выбирается одно из 5 предложенных полей (номер приписного участка, имя, место рождения, гражданство, номер бланка), создается соответствующий

компаратор и далее, если значение было найдено, форма удаляется, иначе при выборе удаления первого вхождения выводится сообщение "Данные не найдены". В функции поиска компаратор задается в соответствии выбранному из пяти предложенных полей. Далее, если список для найденных значений оказывается пустым, пользователю сообщается, что поиск не дал результатов, иначе функция возвращает заполненный список.

Пример реализации одной из функций:

```
void remove() override
    {
        int comparisonResult;
        bool removeAllOccurrences;
        std::shared_ptr<PopulationCensus> censusPtr;

        PopulationCensus* census = nullptr;
        PopulationCensus tmp;
        Strategy<std::shared_ptr<PopulationCensus>> *strategy
        ↪  = nullptr;
        std::cout << "Удалить все совпадения? (если выбрано
        ↪  'нет', то удалится первый найденный элемент):\n"
                  "1. Да\n"
                  "2. Нет\n>";
        std::cin >> comparisonResult;
        if (comparisonResult == 1)
            removeAllOccurrences = true;
        else if (comparisonResult == 2)
            removeAllOccurrences = false;
        else
            throw std::invalid_argument("Invalid input!");

        std::cout << "Выберете поле для удаления:\n";
        fieldSelectionMenu();
        std::cin >> comparisonResult;
        createComparerAndCensus(&census, &strategy,
        ↪  comparisonResult);
        tmp = *census;
        if (removeAllOccurrences) {
            while (true) {
                censusPtr =
                ↪  innerFind(std::shared_ptr<PopulationCensus>
                (std::shared_ptr<PopulationCensus>(census)),
```

```
30                                          strategy);
31                 census = new PopulationCensus(tmp);
32                 if (censusPtr == nullptr)
33                     break;
34                 list.remove(censusPtr);
35             }
36             delete census;
37         }
38         else
39         {
40             censusPtr =
                ↪ innerFind(std::shared_ptr<PopulationCensus>(census),
                ↪ strategy);
41             if (censusPtr == nullptr)
42             {
43                 std::cout << "Данные не найдены";
44                 return;
45             }
46             list.remove(censusPtr);
47         }
48         delete strategy;
49     }
```

2.6 binaryTree.h

В данном заголовочном файле описаны два абстрактных шаблонных класса *BinaryTree<T>* и *TreeNode<T>*.

Класс *TreeNode<T>* представляет собой абстракцию узла дерева поиска. Класс содержит поле с шаблонным типом данных и два конструктора, первый предназначен для инициализации шаблона по умолчанию, второй для создания узла с конкретным значением.

Класс *BinaryTree<T>* представляет собой абстракцию дерева поиска, содержит три чистых виртуальные функции: добавление элемента в дерево, удаление элемента из дерева, поиска элемента в дереве по значению. Конструктор класса принимает в качестве параметра шаблонный компаратор, реализованный в виде объекта стратегии.

```
1 template <typename T>
2 class TreeNode
```

```
{
public:
    T value;
    TreeNode() : TreeNode(T())
    {

    }
    explicit TreeNode(T inputValue)
    {
        value = inputValue;
    }
    ~TreeNode() = default;
};

template <class T>
class BinaryTree {
protected:
    TreeNode<T> *root = nullptr;
    Strategy<T> *strategy_;
    explicit BinaryTree(Strategy<T> *strategy = nullptr) :
     ↪  strategy_(strategy) {}
public:
    virtual void addNode(const T& data) = 0;
    virtual void deleteNode(const T& data) = 0;
    virtual TreeNode<T>* searchByValue(const T& data) = 0;
    virtual ~BinaryTree() = default;
};
```

2.7 decoratorAVL.h

В данном файле класс *DecoratorAVL* наследуется от класса *Decorator*. В приватном поле класса находится односвязный список данных *std::forwar d_ list<std::shaed_ ptr<PopulationCensus» dataList* и упорядоченный ассоциативный контейнер индексов *map std::map<int, BinaryTree<std::share d_ ptr<PopulationCensus>>*> indexMap*, в первом контейнере хранятся данные, анкеты, во втором хранится ключ-значение, первое — уникальный идентификатор для каждого АВЛ дерева, второе указатель на родительский класс *BinaryTree*.

В конструкторе класса создаются пять АВЛ-деревьев с различными компараторами и добавляются в упорядоченный ассоциативный кон-

тейнер *indexMap*. Данная реализация была выбрана для того, чтобы не пересобирать деревья и хранить сразу пять, за счет того, что мы храним указатели на объекты, а не сами объекты они не дублируются и не занимают дополнительное место в памяти. Также хранение сразу пяти деревьев позволяет осуществлять более быстрый поиск за счет того, что на момент поиска данных дерево уже собрано по определенному компаратору и готово к работе.

Так как никаких втроенных методов непредусмотренно, все функции добавления и удаления реализовываются как методы класса *AVLTree*.

АВЛ - дерево представляет собой списковую структуру, похожую на бинарное дерево поиска, с одним дополнительным условием: дерево должно оставаться сбалансированным по высоте после каждой операции вставки или удаления. Поскольку АВЛ - дерево является расширенным бинарным деревом поиска, класс AVLTree строится на базе класса BinaryTree и является его наследником.

Вставка нового ключа в АВЛ-дерево выполняется, по большому счету, так же, как это делается в бинарных деревьях поиска: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что выполняется балансировка текущего узла.

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда фактор сбалансированности некоторых узлов оказывается равными 2 или -2, т.е. возникает расбалансировка поддерева. Для исправления ситуации применяются повороты вокруг тех или иных узлов дерева.

Поворот дерева - это операция, которая позволяет изменить структуру дерева не меняя порядка элементов. Особенность этих операций в том, что они могут уменьшить высоту дерева. Используя эти операции для соответствующих узлов можно минимизировать высоту дерева. Левый поворот позволяет укорачивать ветвь, "растянувшуюся"вправо, правый, наоборот, длинную левую ветвь.

Во-первых, мы "расплетаем"дерево, используя левый и правый поворот, превращаем его в левоассоциативную лозу (*vine*). У этой лозы будет m узлов. Во-вторых, мы сводим эту лозу, используя правый пово-

рот, к дереву, у которого главная левая лоза состоит из n-1 узла, где n - это ближайшее целое число, меньше данного, равное степени двойки (целочисленный логарифм по основанию 2). В конце мы проводим следующую операцию: отмечаем, начиная с вершины, через один все узлы дерева, после чего производим относительно них поворот. Эту операцию повторяем *log(n)* раз. Получается сбалансированное двоичное дерево поиска.

Для реализации **удаления** будем исходить из того же принципа, что и при вставке: найдём вершину, удаление из которой не приведёт к изменению её высоты. найденная удаляемая вершина заменяется значением из левой подветви.

1. Ищем удаляемый элемент и попутно находим нашу замечательную вершину
2. Производим изменение балансов, в случае необходимости делаем ребалансировку
3. Удаляем наш элемент

Реализация функции добавлении заключается в следующем: элемент добавляется в односвязный список, после чего элемент добавляется в каждое дерево.

Реализация функции удаления: пользователь выбирает удаляются ли все вхождения или только первое, далее выбирается поле для удаление, создается форма с заполненным полем, по которому будет происходить поиск, если элемент был найден, он сохраняется и вызывается приватный метод класса *deleteData*, в которой циклично находится удаляемый объект, после чего он удаляется из односвязного списка и из всех АВЛ-деревьев.

Функция поиска работает по следующему алгоритму: пользователь выбирает поле для поиска, создается форма с заполненным полем и происходит поиск в конкретном дереве, найденные значения записываются в двусвязный список и возвращаются из функции, иначе выводится сообщение о том, что поиск не дал результатов.

```
1 void add(std::shared_ptr<PopulationCensus> data) override
2 {
3     dataList.emplace_front(data);
```

```
    for (std::pair<int,
     ↪  BinaryTree<std::shared_ptr<PopulationCensus>>*> it :
     ↪  indexMap)
        it.second->addNode(data);
}
```

2.8 menu.h

В данный заголовочный файл вынесены меню для работы с пользователем: меню для выбора критерия подсчета статистики и общее меню, которые увидит пользователь при запуске программы.

Также присутствует возможность добавления анкеты с помощью перегруженного оператора вставки в поток.

2.9 countingStatistics.h

В файле реализованы функции для подсчета абсолютных и относительных частот для каждого пункта. Результатом ее работы является вывод конкретных значений, их количество в коллекции (абсолютная частота), и относительная частота рассчитываемая по формуле "абсолютная частота / размер коллекции".

```
std::map<marriageAnswer, uint16_t> marriageInfoStatistics()
{
    std::map<marriageAnswer, uint16_t> statistic;
    for (const auto& iter : statisticList)
    {
        auto requiredItem =
         ↪  statistic.find(iter->getMarriageInfo());
        if (requiredItem != statistic.end())
            requiredItem->second++;
        else
            statistic.insert({iter->getMarriageInfo(), 1});
    }
    return statistic;
}
```

2.10 main.cpp

1. Вызывается функция *menu*
2. Пользователю предлагается список дальнейших действий:
    - Сгенерировать и обработать данные
    - Выход
3. Создается выбранный тип коллекции
4. Вводится количество генерируемых анкет
5. Пользователю предлагается список дальнейших действий:
    - Найти форму по одному из полей
    - Удалить форму по одному из полей
    - Добавить форму
    - Вывести статистическую обработку результатов
    - Завершить работу с анкетами
6. Пока пользователь не выберет "Завершить работу с анкетами"его взаимодействие с коллекцией будет продолжаться
7. Происходит удаление использовавшейся коллекции
8. Пока не выбран пункт "Выход пункт №2

2.11 Демонстрация работы программы

```
Приложение для обработки данных переписи населения
1. Сгенерировать и обработать данные
2. Выход
>1
Обработать данные с помощью коллекции:
1. АВЛ Дерева
2. Двунаправленного связанного списка элементов (стандартная библиотека шаблонов)
>1
Какое количество анкет вы хотите сгенерировать?
>2000
Что вы хотите сделать?
1. Найти форму по одному из полей
2. Удалить форму по одному из полей
3. Добавить форму
4. Вывести статистическую обработку результатов
5. Завершить работу с анкетами
>1
Выберете поле для поиска:
1. Номер переписного участка
2. Имя
3. Место рождения
4. Гражданство
5. Номер бланка
>
```

Рис. 2: Поиск в коллекции

```
2. Выход
>1
Обработать данные с помощью коллекции:
1. АВЛ Дерева
2. Двунаправленного связанного списка элементов (стандартная библиотека шаблонов)
>1
Какое количество анкет вы хотите сгенерировать?
>200
Что вы хотите сделать?
1. Найти форму по одному из полей
2. Удалить форму по одному из полей
3. Добавить форму
4. Вывести статистическую обработку результатов
5. Завершить работу с анкетами
>2
Удалить все совпадения? (если выбрано 'нет', то удалится первый найденный элемент):
1. Да
2. Нет
>2
Выберете поле для удаления:
1. Номер переписного участка
2. Имя
3. Место рождения
4. Гражданство
5. Номер бланка
>1
Введите номер участка: 4345
```

Рис. 3: Удаление в коллекции

```
>4
Выберите поле для сбора статистики:
1. номер переписного участка
2. номер помещения в пределах счетного участка
3. номер бланка
4. имя
5. пол
6. дата рождения
7. число полных лет
8. ваше состояние в браке
9. супруг(а) этого лица проживает в домохозяйстве
10. сколько детей вы родили
11. год рождения первого ребенка
12. место вашего рождения
13. с какого года вы непрерывно проживаете в этом населенном пункте
14. прежднее место жительства
15. проживали ли выболее 12 месяцев в других странах
16. где вы проживали до прибытия в Россию (если выбрано проживание в других странах)
17. год прибытия (возвращения) в Россию
18. владаете ли вы русским языком
19. используете ли вы его в повседневной жизни
20. какими иными языками вы владеете
21. родной язык
22. гражданство
23. национальная принадлежность
24. умеете ли вы читать и писать
25. имеете ли вы ученую степень
```

Рис. 4: Меню выбора подсчета статистики

```
>5
женский Статистика:
Абсолютная: 963
Относительная: 481.5
мужской Статистика:
Абсолютная: 1037
Относительная: 518.5
```

Рис. 5: Вывод статистики

3 Вывод

При написании курсовой работы по теме: "Исследование и реализация структур данных: список и АВЛ-дерево" была продемонстрирована работа с использованием stl-коллекции, а также была реализована собственная коллекция. Также в работе использовались различные паттерны: "Декоратор", "Фабричный метод" и "Стратегия".

Проведя сравнительный анализ можно сказать, что для решения поставленной проблемы АВЛ-дерево подходит лучше, так как работа с анкетами подразумевает постоянное обращение к ним, поиск или удаление. Сложность алгоритма поиска в двусвязном списке $O(n)$, где n — количество элементов, а в АВЛ-дереве сложность алгоритма поиска $O(log(n))$, что позволяет гарантировать более быстрый доступ к элементам, однако более продолжительную вставку, по сравнению с двусвязным списком, где вставка элемента будет происходить за O(1), а в АВЛ-дереве за O(log(n)).

При использовании полученного приложения пользователь может добавлять новые анкеты, искать нужные по определенным полям, а также удалять анкеты и узнавать статистику по каждому из полей.

4 Приложение

4.1 PopulationCensus.h

```
//
// Created by ayttekao on 5/14/21.
//

#ifndef FUNDI_COURSEWORK_4_SEMESTER__POPULATIONCENSUS_H
#define FUNDI_COURSEWORK_4_SEMESTER__POPULATIONCENSUS_H
#include "customTypes.h"
#include <random>
#include <set>
#include <utility>
#include <vector>
#include <list>
#include <memory>

#define MAX_NUM_OF_SECONDS (5*365*24*60*60) // number of seconds in 5 years

struct tm* GetTimeAndDate()
{
    unsigned int now_seconds  = (unsigned int)time(nullptr);
    unsigned int rand_seconds = (rand()*rand())%(MAX_NUM_OF_SECONDS+1);
    time_t       rand_time    = (time_t)(now_seconds-rand_seconds);
    return localtime(&rand_time);
}

std::string randomString(size_t length)
{
    auto randChar = []() -> char
    {
        static std::random_device rd;
        std::default_random_engine generator(rd());
        std::uniform_int_distribution<unsigned int> distribution(0, 25);
        const char charset[] =
                "abcdefghijklmnopqrstuvwxyz";
        return charset[ distribution(generator) ];
    };
    std::string str(length,0);
    std::generate_n(str.begin(), length, randChar );
    return str;
}

class PopulationCensus
{
protected:
    /*1*/uint16_t        enumerationAreaNumber; /* номер переписного участка */
```

```
45      /*2*/uint16_t         householdNumberWithinTheEnumerationArea; /*
        ↪  номер помещения в пределах счетного участка */
46      /*3*/uint16_t         formNumber; /* номер бланка */
47      /*4*/std::string      fullName; /* ваше имя */
48      /*5*/gender           selectedGender; /* ваш пол */
49      /*6*/struct tm*       birthDay; /* дата вашего рождения */
50      /*7*/uint8_t          numberOfYears; /* число полных лет */
51      /*8*/marriageAnswer   marriageInfo; /* ваше состояние в браке */
52      /*9*/yesNoAnswer      thatPersonSpouseLivesHousehold; /* супруг(а)
        ↪  этого лица проживает в домохозяйстве */
53      /*10*/uint8_t          amountOfChildren; /* сколько детей вы родили */
54      /*11*/struct tm*       birthDayFirstChild; /* год рождения первого
        ↪  ребенка */
55      /*12*/std::string      placeOfBirth; /* место вашего рождения */
56      /*13*/uint8_t          yearOfContinuousResidence; /* с какого года вы
        ↪  непрерывно проживаете в этом населенном пункте */
57      /*14*/std::string      previousPlaceOfResidence; /* ваше прежднее место
        ↪  жительства */
58      /*15*/yesNoAnswer      livedInOtherCountries; /* проживали ли выболее
        ↪  12 месяцев в других странах */
59      /*16*/std::string      placeOfResidenceToArrivalInRussia; /* где вы
        ↪  проживали до прибытия в Россию (если выбрано проживание
60                                                  * в других странах)
    ↪  */
61      /*17*/uint8_t          yearsOfComebackToRussia; /* год прибытия
        ↪  (возвращения) в Россию */
62      /*18*/yesNoAnswer      proficiencyInRussian; /* владаете ли вы русским
        ↪  языком */
63      /*19*/yesNoAnswer      useOfRussianLanguageInEverydayLife; /*
        ↪  используете ли вы его в повседневной жизни */
64      /*20*/std::set<std::string> languages; /* какими иными языками вы
        ↪  владеете */
65      /*21*/std::string      nativeLanguage; /* ваш родной язык */
66      /*22*/std::string      citizenship; /* ваше гражданство */
67      /*23*/std::string      nationality; /* ваша национальная
        ↪  принадлежность */
68      /*24*/yesNoAnswer      abilityReadAndWrite; /* умеете ли вы читать и
        ↪  писать (если выбрано не имею образования) */
69      /*25*/typeOfDegree     academicDegree; /* имеете ли вы ученую степень
        ↪  (для лиц с высшим образованием и кадров высшей квалификации) */
70      /*26*/yesNoAnswer      currentEducation; /* получаете ли вы
        ↪  образование в настоящее время */
71      /*27*/std::set<studyingPrograms> currentEducationPrograms; /* отметье
        ↪  все программы по которым обучаетесь (если есть текущее
        ↪  образование) */
72      /*28*/std::set<typesOfLivelihoods> livelihoods; /* укажите все
        ↪  имеющиеся у вас источники средств к существованию */
73      /*29*/typesOfLivelihoods mainLivelihood; /* какой из отмеченных
        ↪  источников вы считаете для себя основным */
74      /*30*/yesNoAnswer      workForCertainPeriod; /* имели ли вы какую-либо
        ↪  оплачиваемую работу или доходное занятие с 25 по 31 марта 2021
        ↪  года */
```

```
75      /*31*/positionAtWork  positionMainJob; /* кем вы являлись на основной
        ↪  работе */
76      /*32*/yesNoAnswer     mainWorkWasInTheSameSettlement; /* ваша
        ↪  основная работа находилась в том же населенном пункте, где вы
        ↪  проживаете постоянно */
77      /*33*/typeInformationAboutWork informationAboutWork; /* где
        ↪  находилась ваша основная работа */
78      /*34*/workScheduleOptions schedule; /* вы выезжали(выходили) на
        ↪  работу */
79      /*35*/jobOptionsMarch     suitableJobInMarch; /* если бы вам
        ↪  предложили подходящую работу в последнюю неделю марта,
80                                          * то когда вы смогли бы
    ↪  приступить к ней */
81      /*36*/yesNoAnswer                   jobSearchInMarch; /* вы искали
        ↪  работу в течении марта */
82      /*37*/mainReasonWorkMarch           jobSearchDuringMarch; /* вы искали
        ↪  работу в течении марта */
83      /*38*/registrationInfoInHousehold registrationInThisHousehold; /*
        ↪  зарегистрированы ли вы в этом помещении */
84      /*39*/residenceRegistrationInfo   residenceRegistration; /* где вы
        ↪  зарегистрировваны по месту жительства */
85
86      static yesNoAnswer generateYesNo()
87      {
88          static std::random_device rd;
89          std::default_random_engine generator(rd());
90          std::uniform_int_distribution<unsigned int> distribution(0, 1);
91          auto answer = static_cast<yesNoAnswer>(distribution(generator));
92          return answer;
93      };
94
95      static gender generateGender()
96      {
97          static std::random_device rd;
98          std::default_random_engine generator(rd());
99          std::uniform_int_distribution<unsigned int> distribution(0, 1);
100         auto answer = static_cast<gender>(distribution(generator));
101         return answer;
102     }
103
104     static marriageAnswer generateMarriageAnswer()
105     {
106         static std::random_device rd;
107         std::default_random_engine generator(rd());
108         std::uniform_int_distribution<unsigned int> distribution(0, 5);
109         auto answer =
            ↪  static_cast<marriageAnswer>(distribution(generator));
110         return answer;
111     }
112
113     static typeOfDegree generateTypeOfDegree()
114     {
```

```
115         static std::random_device rd;
116         std::default_random_engine generator(rd());
117         std::uniform_int_distribution<unsigned int> distribution(0, 2);
118         auto answer = static_cast<typeOfDegree>(distribution(generator));
119         return answer;
120     }
121 
122     static typesOfLivelihoods generateTypesOfLivelihoods()
123     {
124         static std::random_device rd;
125         std::default_random_engine generator(rd());
126         std::uniform_int_distribution<unsigned int> distribution(0, 13);
127         auto answer =
            ↪  static_cast<typesOfLivelihoods>(distribution(generator));
128         return answer;
129     }
130 
131     static positionAtWork generatePositionAtWork()
132     {
133         static std::random_device rd;
134         std::default_random_engine generator(rd());
135         std::uniform_int_distribution<unsigned int> distribution(0, 5);
136         auto answer =
            ↪  static_cast<positionAtWork>(distribution(generator));
137         return answer;
138     }
139 
140     static workScheduleOptions generateWorkScheduleOptions()
141     {
142         static std::random_device rd;
143         std::default_random_engine generator(rd());
144         std::uniform_int_distribution<unsigned int> distribution(0, 4);
145         auto answer =
            ↪  static_cast<workScheduleOptions>(distribution(generator));
146         return answer;
147     }
148 
149     static jobOptionsMarch generateJobOptionsMarch()
150     {
151         static std::random_device rd;
152         std::default_random_engine generator(rd());
153         std::uniform_int_distribution<unsigned int> distribution(0, 2);
154         auto answer =
            ↪  static_cast<jobOptionsMarch>(distribution(generator));
155         return answer;
156     }
157 
158     static mainReasonWorkMarch generateMainReasonWorkMarch()
159     {
160         static std::random_device rd;
161         std::default_random_engine generator(rd());
162         std::uniform_int_distribution<unsigned int> distribution(0, 10);
```

```
163         auto answer =
            ↪ static_cast<mainReasonWorkMarch>(distribution(generator));
164         return answer;
165     }
166
167     static registrationInfoInHousehold
        ↪ generateRegistrationInfoInHousehold()
168     {
169         static std::random_device rd;
170         std::default_random_engine generator(rd());
171         std::uniform_int_distribution<unsigned int> distribution(0, 2);
172         auto answer =
            ↪ static_cast<registrationInfoInHousehold>(distribution(generator));
173         return answer;
174     }
175
176     static residenceRegistrationInfo generateResidenceRegistrationInfo()
177     {
178         static std::random_device rd;
179         std::default_random_engine generator(rd());
180         std::uniform_int_distribution<unsigned int> distribution(0, 3);
181         auto answer =
            ↪ static_cast<residenceRegistrationInfo>(distribution(generator));
182         return answer;
183     }
184
185     static std::set<studyingPrograms>
        ↪ generateSetCurrentEducationPrograms()
186     {
187         static std::random_device rd;
188         std::default_random_engine generator(rd());
189         std::uniform_int_distribution<unsigned int>
            ↪ distributionNumElements(1, 2);
190         size_t numTotalOfAllElements =
            ↪ distributionNumElements(generator);
191         std::set<studyingPrograms> set;
192         std::uniform_int_distribution<unsigned int>
            ↪ distributionNumEnums(0, 3);
193         for (int i = 0; i < numTotalOfAllElements; i++)
194
                ↪ set.insert(static_cast<studyingPrograms>(distributionNumEnums(gener
195         return set;
196     }
197
198     static std::set<typesOfLivelihoods> generateSetTypesOfLivelihoods()
199     {
200         static std::random_device rd;
201         std::default_random_engine generator(rd());
202         std::uniform_int_distribution<unsigned int>
            ↪ distributionNumElements(1, 2);
203         size_t numTotalOfAllElements =
            ↪ distributionNumElements(generator);
```

```
204         std::set<typesOfLivelihoods> set;
205         std::uniform_int_distribution<unsigned int>
            ↪  distributionNumEnums(0, 13);
206         for (int i = 0; i < numTotalOfAllElements; i++)
207 
                ↪  set.insert(static_cast<typesOfLivelihoods>(distributionNumEnums(gen
208         return set;
209     }
210 
211     static std::set<std::string> generateSetLanguages()
212     {
213         static std::random_device rd;
214         std::default_random_engine generator(rd());
215         std::uniform_int_distribution<unsigned int>
            ↪  distributionNumElements(0, 4);
216         size_t numTotalOfAllElements =
            ↪  distributionNumElements(generator);
217         std::set<std::string> set;
218         std::uniform_int_distribution<unsigned int>
            ↪  distributionLenString(5, 10);
219         for (int i = 0; i < numTotalOfAllElements; i++)
220             set.insert(randomString(distributionLenString(generator)));
221         return set;
222     }
223 public:
224     friend class StrategyEnumerationAreaNumber;
225     friend class StrategyFullName;
226     friend class StrategyPlaceOfBirth;
227     friend class StrategyCitizenship;
228     friend class StrategyFormNumber;
229 
230     ~PopulationCensus() = default;
231 
232     static std::shared_ptr<PopulationCensus>
        ↪  randomGenerateObjectPopulationCensus()
233     {
234         PopulationCensus *object = new PopulationCensus;
235         std::random_device rd;
236         std::mt19937 gen(rd());
237         std::uniform_int_distribution<> distrib(1, 65535);
238         std::uniform_int_distribution<> distribLengthForStrings(4, 13);
239         std::uniform_int_distribution<unsigned int> distribution(0, 1);
240         object->enumerationAreaNumber = distrib(gen);
241         object->householdNumberWithinTheEnumerationArea = distrib(gen);
242         object->formNumber = distrib(gen);
243         object->fullName = randomString(distribLengthForStrings(gen));
244         object->selectedGender = generateGender();
245         object->birthDay = GetTimeAndDate();
246         distrib = std::uniform_int_distribution<>(1, 89);
247         object->numberOfYears = distrib(gen);
248         object->marriageInfo = generateMarriageAnswer();
249         object->thatPersonSpouseLivesHousehold = generateYesNo();
```

```
250         distrib = std::uniform_int_distribution<>(1, 6);
251         object->amountOfChildren = distrib(gen);
252         object->birthDayFirstChild = GetTimeAndDate();
253         object->placeOfBirth =
            ↪  randomString(distribLengthForStrings(gen));
254         distrib = std::uniform_int_distribution<>(1993, 2020);
255         object->yearOfContinuousResidence = distrib(gen);
256         object->previousPlaceOfResidence =
            ↪  randomString(distribLengthForStrings(gen));
257         object->livedInOtherCountries =
            ↪  object->thatPersonSpouseLivesHousehold = generateYesNo();
258         object->placeOfResidenceToArrivalInRussia =
            ↪  randomString(distribLengthForStrings(gen));
259         object->yearsOfComebackToRussia = distrib(gen);
260         object->proficiencyInRussian = generateYesNo();
261         object->useOfRussianLanguageInEverydayLife = generateYesNo();
262         object->languages = generateSetLanguages();
263         object->nativeLanguage =
            ↪  randomString(distribLengthForStrings(gen));
264         object->citizenship = randomString(distribLengthForStrings(gen));
265         object->nationality = randomString(distribLengthForStrings(gen));
266         object->abilityReadAndWrite = generateYesNo();
267         object->academicDegree = generateTypeOfDegree();
268         object->currentEducation = generateYesNo();
269         object->currentEducationPrograms =
            ↪  generateSetCurrentEducationPrograms();
270         object->livelihoods = generateSetTypesOfLivelihoods();
271         object->mainLivelihood = generateTypesOfLivelihoods();
272         object->workForCertainPeriod = generateYesNo();
273         object->positionMainJob = generatePositionAtWork();
274         object->mainWorkWasInTheSameSettlement = generateYesNo();
275         object->informationAboutWork.anotherState =
            ↪  randomString(distribLengthForStrings(gen));
276         object->informationAboutWork.district =
            ↪  randomString(distribLengthForStrings(gen));
277         object->informationAboutWork.subject =
            ↪  randomString(distribLengthForStrings(gen));
278         object->informationAboutWork.urbanSettlement =
            ↪  randomString(distribLengthForStrings(gen));
279         object->schedule = generateWorkScheduleOptions();
280         object->suitableJobInMarch = generateJobOptionsMarch();
281         object->jobSearchInMarch = generateYesNo();
282         object->jobSearchDuringMarch = generateMainReasonWorkMarch();
283         object->registrationInThisHousehold =
            ↪  generateRegistrationInfoInHousehold();
284         object->residenceRegistration =
            ↪  generateResidenceRegistrationInfo();
285         return std::shared_ptr<PopulationCensus>(object);
286     }
287     bool operator==(const PopulationCensus& rhs)
288     {
```

```
289         if (enumerationAreaNumber == rhs.enumerationAreaNumber and
            ↪  householdNumberWithinTheEnumerationArea
290                                                                  ==
                                                                     ↪  rhs.household
                                                                     ↪  and
                                                                     ↪  formNumber
                                                                     ↪  ==
                                                                     ↪  rhs.formNumb
                                                                     ↪  and
                                                                     ↪  selectedGend
                                                                     ↪  ==
                                                                     ↪  rhs.selected
291
292             thatPersonSpouseLivesHousehold ==
                ↪  rhs.thatPersonSpouseLivesHousehold and amountOfChildren
                ↪  == rhs.amountOfChildren
293
294
```

```
295
296             placeOfResidenceToArrivalInRussia ==
                ↪  rhs.placeOfResidenceToArrivalInRussia and
                ↪  yearsOfComebackToRussia ==
297
298
299             citizenship == rhs.citizenship and nationality ==
                ↪  rhs.nationality and abilityReadAndWrite ==
                ↪  rhs.abilityReadAndWrite
300
```

```
301
302             workForCertainPeriod == rhs.workForCertainPeriod and
                ↪ positionMainJob ==
303                                                                 ↪ rhs.positionMa
                                                                    ↪ and
                                                                    ↪ mainWorkWasInT
                                                                    ↪ ==
                                                                    ↪ rhs.mainWorkWa
                                                                    ↪ and
                                                                    ↪ informationAbo
                                                                    ↪ ==
304
305             informationAboutWork.district ==
                ↪ rhs.informationAboutWork.district and
                ↪ informationAboutWork.anotherState ==
                ↪ rhs.informationAboutWork.urbanSettlement
306             and schedule == rhs.schedule and suitableJobInMarch ==
                ↪ rhs.suitableJobInMarch and jobSearchInMarch ==
                ↪ rhs.jobSearchInMarch and
307             jobSearchDuringMarch == rhs.jobSearchDuringMarch and
                ↪ registrationInThisHousehold ==
                ↪ rhs.registrationInThisHousehold and
308             residenceRegistration == rhs.residenceRegistration)
309             return true;
310         else
311             return false;
312     }
313     friend std::istream& operator>>(std::istream &in, PopulationCensus&
        ↪ census)
314     {
315         uint16_t    tmpNum;
316         std::string tmpString;
317         int year, month, day;
318
319         std::cout << "Номер переписного участка: ";
```

```
320         in >> tmpNum;
321         tmpNum = census.enumerationAreaNumber;
322         std::cout << "Номер помещения в пределах счетного участка: ";
323         in >> census.householdNumberWithinTheEnumerationArea;
324         std::cout << "Номер бланка: ";
325         in >> census.formNumber;
326         std::cout << "Ваше имя: ";
327         in >> census.fullName;
328         std::cout << "Ваш пол (1 - мужской, 2 - женский): ";
329         in >> tmpNum;
330         if (tmpNum == 1)
331             census.selectedGender = gender::male;
332         else if (tmpNum == 2)
333             census.selectedGender = gender::female;
334         else
335             throw std::invalid_argument("Incorrect input!");
336         std::cout << "число полных лет: ";
337         in >> census.numberOfYears;
338         std::cout << "ваше состояние в браке:\n"
339                      "1. состою в зарегистрированном браке\n"
340                      "2.состою в незарегистрированном супружеском
                         ↪  союзе\n"
341                      "3. разведен(а) официально\n"
342                      "4. разошелся(лась)\n"
343                      "5. вдовец(вдова)\n"
344                      "6. никогда не состоял(а) в браке, супружеском
                         ↪  союзе\n>";
345
346         in >> tmpNum;
347         if (tmpNum == 1)
348             census.marriageInfo = marriageAnswer::registeredMarriage;
349         switch (tmpNum) {
350             case 1:
351                 census.marriageInfo = marriageAnswer::registeredMarriage;
352                 break;
353             case 2:
354                 census.marriageInfo =
                     ↪  marriageAnswer::unregisteredMaritalUnion;
355                 break;
356             case 3:
357                 census.marriageInfo = marriageAnswer::officiallyDivorced;
358                 break;
359             case 4:
360                 census.marriageInfo = marriageAnswer::breakUp;
361                 break;
362             case 5:
363                 census.marriageInfo = marriageAnswer::widow;
364                 break;
365             case 6:
366                 census.marriageInfo = marriageAnswer::neverMarried;
367                 break;
368             default:
```

```
369                 throw std::invalid_argument("Incorrect input!");
370             }
371             std::cout << "супруг(а) этого лица проживает в домохозяйстве (1 -
                ↪  да, 2 - нет)\n>";
372
373             in >> tmpNum;
374             if (tmpNum == 1)
375                 census.thatPersonSpouseLivesHousehold = yesNoAnswer::yes;
376             else if (tmpNum == 2)
377                 census.thatPersonSpouseLivesHousehold = yesNoAnswer::no;
378             else
379                 throw std::invalid_argument("Incorrect input!");
380             std::cout << "сколько детей вы родили: ";
381             in >> census.amountOfChildren;
382             std::cout << "место вашего рождения: ";
383             in >> census.placeOfBirth;
384             std::cout << "с какого года вы непрерывно проживаете в этом
                ↪  населенном пункте: ";
385             in >> census.yearOfContinuousResidence;
386             std::cout << "ваше прждее место жительства: ";
387             in >> census.previousPlaceOfResidence;
388             std::cout << "проживали ли выболее 12 месяцев в других странах(1
                ↪  - да, 2 - нет): ";
389             in >> tmpNum;
390             if (tmpNum == 1)
391                 census.livedInOtherCountries = yesNoAnswer::yes;
392             else if (tmpNum == 2)
393                 census.livedInOtherCountries = yesNoAnswer::no;
394             else
395                 throw std::invalid_argument("Incorrect input!");
396             std::cout << "где вы проживали до прибытия в Россию: ";
397             in >> census.placeOfResidenceToArrivalInRussia;
398             std::cout << "год прибытия (возвращения) в Россию: ";
399             in >> census.yearsOfComebackToRussia;
400             std::cout << "владаете ли вы русским языком(1 - да, 2 - нет): ";
401             in >> tmpNum;
402             if (tmpNum == 1)
403                 census.proficiencyInRussian = yesNoAnswer::yes;
404             else if (tmpNum == 2)
405                 census.proficiencyInRussian = yesNoAnswer::no;
406             else
407                 throw std::invalid_argument("Incorrect input!");
408             std::cout << "используете ли вы его в повседневной жизни(1 - да,
                ↪  2 - нет): ";
409             in >> tmpNum;
410             if (tmpNum == 1)
411                 census.useOfRussianLanguageInEverydayLife = yesNoAnswer::yes;
412             else if (tmpNum == 2)
413                 census.useOfRussianLanguageInEverydayLife = yesNoAnswer::no;
414             else
415                 throw std::invalid_argument("Incorrect input!");
```

```
416         std::cout << "какими иными языками вы владеете(количество языков,
            ↪  языки)";
417         in >> tmpNum;
418         for (int i = 0; i < tmpNum; i++) {
419             in >> tmpString;
420             census.languages.insert(tmpString);
421         }
422         std::cout << "ваш родной язык: ";
423         in >> census.nativeLanguage;
424         std::cout << "ваше гражданство: ";
425         in >> census.citizenship;
426         std::cout << "ваша национальная принадлежность: ";
427         in >> census.nationality;
428         std::cout << "умеете ли вы читать и писать(1 - да, 2 - нет): ";
429         in >> tmpNum;
430         if (tmpNum == 1)
431             census.useOfRussianLanguageInEverydayLife = yesNoAnswer::yes;
432         else if (tmpNum == 2)
433             census.useOfRussianLanguageInEverydayLife = yesNoAnswer::no;
434         else
435             throw std::invalid_argument("Incorrect input!");
436         std::cout << "имеете ли вы ученую степень (для лиц с высшим
            ↪  образованием и кадров высшей квалификации):\n"
437                      "1. кандидат наук\n"
438                      "2. доктор наук\n"
439                      "3. не имею\n>";
440         in >> tmpNum;
441         if (tmpNum == 1)
442             census.academicDegree = typeOfDegree::PhD;
443         else if (tmpNum == 2)
444             census.academicDegree = typeOfDegree::DoctorOfScience;
445         else if (tmpNum == 3)
446             census.academicDegree = typeOfDegree::dontHave;
447         else
448             throw std::invalid_argument("Incorrect input!");
449         std::cout << "получаете ли вы образование в настоящее время(1 -
            ↪  да, 2 - нет): ";
450         in >> tmpNum;
451         if (tmpNum == 1)
452             census.currentEducation = yesNoAnswer::yes;
453         else if (tmpNum == 2)
454             census.currentEducation = yesNoAnswer::no;
455         else
456             throw std::invalid_argument("Incorrect input!");
457         std::cout << "отметье все программы по которым обучаетесь\n"
458                      "1. программы дошкольного образования\n"
459                      "2. основные профессиональные программы\n"
460                      "3. дополнительные образовательные программы\n"
461                      "4. отмечены все программы\n>";
462         in >> tmpNum;
463         while (tmpNum != 4)
464         {
```

```
465             switch (tmpNum) {
466                 case 1:
467 
                        ↪  census.currentEducationPrograms.insert(studyingPrograms::pr
468                     break;
469                 case 2:
470 
                        ↪  census.currentEducationPrograms.insert(studyingPrograms::ge
471                     break;
472                 case 3:
473 
                        ↪  census.currentEducationPrograms.insert(studyingPrograms::ba
474                     break;
475                 case 4:
476 
                        ↪  census.currentEducationPrograms.insert(studyingPrograms::ad
477                     break;
478                 default:
479                     throw std::invalid_argument("Incorrect input!");
480             }
481             in >> tmpNum;
482             std::cout << ">";
483         }
484         std::cout << "укажите все имеющиеся у вас источники средств к
        ↪  существованию:\n"
485                   "1. заработная плата\n"
486                   "2. предпринимательский доход, самозанятость\n"
487                   "3. производство товаров для собственного
                       ↪  использования\n"
488                   "4. сдача в аренду имущества\n"
489                   "5. доход от патентов, авторских прав\n"
490                   "6. сбережения, дивиденды, проценты, ссуды,
                       ↪  реализация капитала\n"
491                   "7. пенсия(кроме пенсии по инвалидности)\n"
492                   "8. пенсия по инвалидности\n"
493                   "9. стипендия\n"
494                   "10. пособие по безработице\n"
495                   "11. другие пособия и выплаты от организаций,
                       ↪  государства\n"
496                   "12. льготы, компенсации, субсидии, выигрыши\n"
497                   "13. обеспечение со стороны других лиц, иждивение\n"
498                   "14. иной источник\n"
499                   "15. указаны все источники\n>";
500         in >> tmpNum;
501         while (tmpNum != 15)
502         {
503             switch (tmpNum) {
504                 case 1:
505                     census.livelihoods.insert(typesOfLivelihoods::wage);
506                     break;
507                 case 2:
```

```
508
                    ↪ census.livelihoods.insert(typesOfLivelihoods::entrepreneuri
509                 break;
510             case 3:
511
                    ↪ census.livelihoods.insert(typesOfLivelihoods::productionOfG
512                 break;
513             case 4:
514
                    ↪ census.livelihoods.insert(typesOfLivelihoods::rentalOfPrope
515                 break;
516             case 5:
517
                    ↪ census.livelihoods.insert(typesOfLivelihoods::incomeFromPat
518                 break;
519             case 6:
520
                    ↪ census.livelihoods.insert(typesOfLivelihoods::capital);
521                 break;
522             case 7:
523
                    ↪ census.livelihoods.insert(typesOfLivelihoods::pension);
524                 break;
525             case 8:
526
                    ↪ census.livelihoods.insert(typesOfLivelihoods::disabilityPen
527                 break;
528             case 9:
529
                    ↪ census.livelihoods.insert(typesOfLivelihoods::scholarship);
530                 break;
531             case 10:
532
                    ↪ census.livelihoods.insert(typesOfLivelihoods::unemploymentB
533                 break;
534             case 11:
535
                    ↪ census.livelihoods.insert(typesOfLivelihoods::otherBenefits
536                 break;
537             case 12:
538
                    ↪ census.livelihoods.insert(typesOfLivelihoods::compensation)
539                 break;
540             case 13:
541
                    ↪ census.livelihoods.insert(typesOfLivelihoods::dependency);
542                 break;
543             case 14:
544
                    ↪ census.livelihoods.insert(typesOfLivelihoods::anotherSource
545                 break;
546             case 15:
```

```
547                     break;
548                 default:
549                     throw std::invalid_argument("Incorrect input!");
550             }
551         }
552         std::cout << "какой из отмеченных источников вы считаете для себя
        ↪  основным\n"
553                   "1. заработная плата\n"
554                   "2. предпринимательский доход, самозанятость\n"
555                   "3. производство товаров для собственного
                       ↪  использования\n"
556                   "4. сдача в аренду имущества\n"
557                   "5. доход от патентов, авторских прав\n"
558                   "6. сбережения, дивиденды, проценты, ссуды,
                       ↪  реализация капитала\n"
559                   "7. пенсия(кроме пенсии по инвалидности)\n"
560                   "8. пенсия по инвалидности\n"
561                   "9. стипендия\n"
562                   "10. пособие по безработице\n"
563                   "11. другие пособия и выплаты от организаций,
                       ↪  государства\n"
564                   "12. льготы, компенсации, субсидии, выигрыши\n"
565                   "13. обеспечение со стороны других лиц,
                       ↪  иждивение\n";
566         in >> tmpNum;
567         switch (tmpNum) {
568             case 1:
569                 census.livelihoods.insert(typesOfLivelihoods::wage);
570                 break;
571             case 2:
572 
                    ↪  census.livelihoods.insert(typesOfLivelihoods::entrepreneurialIn
573                 break;
574             case 3:
575 
                    ↪  census.livelihoods.insert(typesOfLivelihoods::productionOfGoods
576                 break;
577             case 4:
578 
                    ↪  census.livelihoods.insert(typesOfLivelihoods::rentalOfProperty)
579                 break;
580             case 5:
581 
                    ↪  census.livelihoods.insert(typesOfLivelihoods::incomeFromPatents
582                 break;
583             case 6:
584                 census.livelihoods.insert(typesOfLivelihoods::capital);
585                 break;
586             case 7:
587                 census.livelihoods.insert(typesOfLivelihoods::pension);
588                 break;
589             case 8:
```

```
590
                ↪ census.livelihoods.insert(typesOfLivelihoods::disabilityPension
591             break;
592         case 9:
593
                ↪ census.livelihoods.insert(typesOfLivelihoods::scholarship);
594             break;
595         case 10:
596
                ↪ census.livelihoods.insert(typesOfLivelihoods::unemploymentBenef
597             break;
598         case 11:
599
                ↪ census.livelihoods.insert(typesOfLivelihoods::otherBenefits);
600             break;
601         case 12:
602
                ↪ census.livelihoods.insert(typesOfLivelihoods::compensation);
603             break;
604         case 13:
605
                ↪ census.livelihoods.insert(typesOfLivelihoods::dependency);
606             break;
607         case 14:
608
                ↪ census.livelihoods.insert(typesOfLivelihoods::anotherSource);
609             break;
610         case 15:
611             break;
612         default:
613             throw std::invalid_argument("Incorrect input!");
614     }
615     std::cout << "кем вы являлись на основной работе\n"
616                  "1. работающий по найму\n"
617                  "2. владелец(совладелец) собственного
                      ↪ предприятия(дела)\n"
618                  "3. индивидуальный предприниматель\n"
619                  "4. самозанятый\n"
620                  "5. помогающий на семейном предприятии\n"
621                  "6. иное\n>";
622     in >> tmpNum;
623     switch (tmpNum){
624         case 1:
625             census.positionMainJob = positionAtWork::employed;
626             break;
627         case 2:
628             census.positionMainJob = positionAtWork::businessOwner;
629             break;
630         case 3:
631             census.positionMainJob =
                ↪ positionAtWork::individualEntrepreneur;
632             break;
```

```
633             case 4:
634                 census.positionMainJob = positionAtWork::selfEmployed;
635                 break;
636             case 5:
637                 census.positionMainJob = positionAtWork::familyBusiness;
638                 break;
639             case 6:
640                 census.positionMainJob = positionAtWork::otherwise;
641                 break;
642             default:
643                 throw std::invalid_argument("Incorrect input!");
644         }
645         std::cout << "ваша основная работа находилась в том же населенном
            ↪  пункте, где вы проживаете постоянно\n"
646                   "1. Да\n"
647                   "2. Нет\n>";
648         in >> tmpNum;
649         if (tmpNum == 1)
650             census.currentEducation = yesNoAnswer::yes;
651         else if (tmpNum == 2)
652             census.currentEducation = yesNoAnswer::no;
653         else
654             throw std::invalid_argument("Incorrect input!");
655         std::cout << "где находилась ваша основная работа\n"
656                   "1. субъект РФ\n"
657                   "2. городской населенный пункт\n"
658                   "3. муниципальный район/округ, городской округ\n"
659                   "4. иное государство\n>";
660         in >> census.informationAboutWork.subject >>
            ↪  census.informationAboutWork.urbanSettlement
661            >> census.informationAboutWork.district >>
               ↪  census.informationAboutWork.anotherState;
662         std::cout << "вы выезжали(выходили) на работу\n"
663                   "1. ежедневно\n"
664                   "2. несколько раз в неделю\n"
665                   "3. несколько раз в месяц\n"
666                   "4. один раз в месяц и реже\n"
667                   "5. работаю дистанционно\n>";
668         in >> tmpNum;
669         switch (tmpNum){
670             case 1:
671                 census.schedule = workScheduleOptions::daily;
672                 break;
673             case 2:
674                 census.schedule = workScheduleOptions::fewTimesAWeek;
675                 break;
676             case 3:
677                 census.schedule = workScheduleOptions::fewTimesAMonth;
678                 break;
679             case 4:
680                 census.schedule = workScheduleOptions::onceAMonthOrLess;
681                 break;
```

```
682             case 5:
683                 census.schedule = workScheduleOptions::remotely;
684                 break;
685             default:
686                 throw std::invalid_argument("Incorrect input!");
687         }
688         std::cout << "если бы вам предложили подходящую работу в
        ↪  последнюю неделю марта,\n"
689                   "то когда вы смогли бы приступить к ней:\n"
690                   "1. с 25 по 31 марта\n"
691                   "2. с 1 по 14 апреля\n"
692                   "3. не смог бы приступить в эти периоды\n>";
693         in >> tmpNum;
694         switch (tmpNum){
695             case 1:
696                 census.suitableJobInMarch =
                    ↪  jobOptionsMarch::from25To31March;
697                 break;
698             case 2:
699                 census.suitableJobInMarch =
                    ↪  jobOptionsMarch::from1To14April;
700                 break;
701             case 3:
702                 census.suitableJobInMarch =
                    ↪  jobOptionsMarch::couldNotStartDuringThesePeriods;
703                 break;
704             default:
705                 throw std::invalid_argument("Incorrect input!");
706         }
707         std::cout << "вы искали работу в течении марта:\n"
708                   "1. Да\n"
709                   "2. Нет\n>";
710         in >> tmpNum;
711         if (tmpNum == 1)
712             census.jobSearchInMarch = yesNoAnswer::yes;
713         else if (tmpNum == 2)
714             census.jobSearchInMarch = yesNoAnswer::no;
715         else
716             throw std::invalid_argument("Incorrect input!");
717         std::cout << "зарегистрированы ли вы в этом помещении\n"
718                   "1. по месту жительства(постоянно)\n"
719                   "2. по месту пребывания(временно)\n"
720                   "3. нет\n";
721         in >> tmpNum;
722         switch (tmpNum) {
723             case 1:
724                 census.registrationInThisHousehold =
                    ↪  registrationInfoInHousehold::atThePlaceOfResidence;
725                 break;
726             case 2:
727                 census.registrationInThisHousehold =
                    ↪  registrationInfoInHousehold::atThePlaceOfStay;
```

```
728                 break;
729             case 3:
730                 census.registrationInThisHousehold =
                    ↪ registrationInfoInHousehold::notRegistered;
731                 break;
732             default:
733                 throw std::invalid_argument("Incorrect input!");
734         }
735         std::cout << "где вы зарегистрировваны по месту жительства:\n"
736                     "1. том же населенном пункте, где проживаю
                        ↪ постоянно\n"
737                     "2. в другом населенном пункте того же субъекта РФ,
                        ↪ где проживаю постоянно\n"
738                     "3. в другом субъекте РФ\n"
739                     "4. нет регистрации по месту жительства в
                        ↪ России\n>";
740         in >> tmpNum;
741         switch (tmpNum) {
742             case 1:
743                 census.residenceRegistration =
                    ↪ residenceRegistrationInfo::inTheSameSettlementWhereLivePermanen
744                 break;
745             case 2:
746                 census.residenceRegistration =
                    ↪ residenceRegistrationInfo::inAnotherLocalityOfTheSameSubjectWhe
747                 break;
748             case 3:
749                 census.residenceRegistration =
                    ↪ residenceRegistrationInfo::inAnotherSubject;
750                 break;
751             case 4:
752                 census.residenceRegistration =
                    ↪ residenceRegistrationInfo::noRegistrationInTheRF;
753                 break;
754             default:
755                 throw std::invalid_argument("Incorrect input!");
756         }
757
758         return in;
759     }
760
761     void setEnumerationAreaNumber(uint16_t inputEnumerationArea)
762     {
763         this->enumerationAreaNumber = inputEnumerationArea;
764     }
765
766     void setFullName(const std::string& name)
767     {
768         this->fullName = name;
769     }
770
771     void setPlaceOfBirth(const std::string& input)
```

```
772     {
773         this->placeOfBirth = input;
774     }
775
776     void setCitizenship(std::string inputCitizenship)
777     {
778         this->citizenship = std::move(inputCitizenship);
779     }
780
781     void setFormNumber(uint16_t input)
782     {
783         this->formNumber = input;
784     }
785
786     uint16_t getEnumerationAreaNumber() const
787     {
788         return this->enumerationAreaNumber;
789     }
790
791     uint16_t getHouseholdNumberWithinTheEnumerationArea() const
792     {
793         return this->householdNumberWithinTheEnumerationArea;
794     }
795
796     uint16_t getFormNumber() const
797     {
798         return this->formNumber;
799     }
800
801     gender getSelectedGender()
802     {
803         return this->selectedGender;
804     }
805
806     struct tm* getBirthDay()
807     {
808         return this->birthDay;
809     }
810
811     uint8_t getNumberOfYears() const
812     {
813         return this->numberOfYears;
814     }
815
816     marriageAnswer getMarriageInfo()
817     {
818         return this->marriageInfo;
819     }
820
821     yesNoAnswer getThatPersonSpouseLivesHousehold()
822     {
823         return this->thatPersonSpouseLivesHousehold;
```

```
824     }
825
826     uint8_t getAmountOfChildren() const
827     {
828         return this->amountOfChildren;
829     }
830
831     struct tm* getBirthDayFirstChild()
832     {
833         return this->birthDayFirstChild;
834     }
835
836     std::string getFullName()
837     {
838         return this->fullName;
839     }
840
841     std::string getPlaceOfBirth()
842     {
843         return this->placeOfBirth;
844     }
845
846     uint8_t getYearOfContinuousResidence() const
847     {
848         return this->yearOfContinuousResidence;
849     }
850
851     std::string getPreviousPlaceOfResidence()
852     {
853         return this->previousPlaceOfResidence;
854     }
855
856     yesNoAnswer getLivedInOtherCountries()
857     {
858         return this->livedInOtherCountries;
859     }
860
861     std::string getPlaceOfResidenceToArrivalInRussia()
862     {
863         return this->placeOfResidenceToArrivalInRussia;
864     }
865
866     uint8_t getYearsOfComebackToRussia() const
867     {
868         return this->yearsOfComebackToRussia;
869     }
870
871     yesNoAnswer getProficiencyInRussian()
872     {
873         return this->proficiencyInRussian;
874     }
875
```

```
876     yesNoAnswer getUseOfRussianLanguageInEverydayLife()
877     {
878         return this->useOfRussianLanguageInEverydayLife;
879     }
880 
881     std::set<std::string> getLanguages()
882     {
883         return this->languages;
884     }
885 
886     std::string getNativeLanguage()
887     {
888         return this->nativeLanguage;
889     }
890 
891     std::string getCitizenship()
892     {
893         return this->citizenship;
894     }
895 
896     std::string getNationality()
897     {
898         return this->nationality;
899     }
900 
901     yesNoAnswer getAbilityReadAndWrite()
902     {
903         return this->abilityReadAndWrite;
904     }
905 
906     typeOfDegree getAcademicDegree()
907     {
908         return this->academicDegree;
909     }
910 
911     yesNoAnswer getCurrentEducation()
912     {
913         return this->currentEducation;
914     }
915 
916     std::set<studyingPrograms> getCurrentEducationPrograms()
917     {
918         return this->currentEducationPrograms;
919     }
920 
921     std::set<typesOfLivelihoods> getLivelihoods()
922     {
923         return this->livelihoods;
924     }
925 
926     typesOfLivelihoods getMainLivelihood()
927     {
```

```
928         return this->mainLivelihood;
929     }
930 
931     yesNoAnswer getWorkForCertainPeriod()
932     {
933         return this->workForCertainPeriod;
934     }
935 
936     positionAtWork getPositionMainJob()
937     {
938         return this->positionMainJob;
939     }
940 
941     yesNoAnswer getMainWorkWasInTheSameSettlement()
942     {
943         return this->mainWorkWasInTheSameSettlement;
944     }
945 
946     typeInformationAboutWork getInformationAboutWork()
947     {
948         return this->informationAboutWork;
949     }
950 
951     workScheduleOptions getSchedule()
952     {
953         return this->schedule;
954     }
955 
956     jobOptionsMarch getSuitableJobInMarch()
957     {
958         return this->suitableJobInMarch;
959     }
960 
961     yesNoAnswer getJobSearchInMarch()
962     {
963         return this->jobSearchInMarch;
964     }
965 
966     mainReasonWorkMarch getJobSearchDuringMarch()
967     {
968         return this->jobSearchDuringMarch;
969     }
970 
971     registrationInfoInHousehold getRegistrationInThisHousehold()
972     {
973         return this->registrationInThisHousehold;
974     }
975 
976     residenceRegistrationInfo getResidenceRegistration()
977     {
978         return this->residenceRegistration;
979     }
```

```
980
981 };
982
983 #endif //FUNDI_COURSEWORK_4_SEMESTER__POPULATIONCENSUS_H
```

4.2 customTypes.h

```
1  //
2  // Created by aytteкao on 5/10/21.
3  //
4  #ifndef FUNDI_COURSEWORK_4_SEMESTER__CUSTOMTYPES_H
5  #define FUNDI_COURSEWORK_4_SEMESTER__CUSTOMTYPES_H
6  enum gender
7  {
8      male,
9      female
10 };
11 enum marriageAnswer
12 {
13     registeredMarriage, /* состою в зарегистрированном браке */
14     unregisteredMaritalUnion, /*состою в незарегистрированном супружеском
       ↪  союзе*/
15     officiallyDivorced, /* разведен(а) официально */
16     breakUp, /* рашелся(лась) */
17     widow, /* вдвовец(вдова) */
18     neverMarried /* никогда не состоял(а) в браке, супружеском союзе */
19 };
20 std::string putMarriageAnswer(marriageAnswer answer)
21 {
22     switch (answer) {
23         case marriageAnswer::registeredMarriage:
24             return "состою в зарегистрированном браке";
25         case marriageAnswer::unregisteredMaritalUnion:
26             return "состою в незарегистрированном супружеском союзе";
27         case marriageAnswer::officiallyDivorced:
28             return "разведен(а) официально";
29         case marriageAnswer::breakUp:
30             return "рашелся(лась)";
31         case marriageAnswer::widow:
32             return "вдвовец(вдова)";
33         case marriageAnswer::neverMarried:
34             return "никогда не состоял(а) в браке, супружеском союзе";
35         default:
36             return "invalid";
37     }
38 }
39 enum yesNoAnswer
40 {
41     yes,
42     no
```

```
43 };
44 enum typeOfDegree
45 {
46     PhD, /* кандидат наук */
47     DoctorOfScience, /* доктор наук */
48     dontHave /* не имею */
49 };
50 std::string putTypeOfDegree(typeOfDegree type)
51 {
52     if (type == typeOfDegree::PhD)
53         return "кандидат наук";
54     else if (type == typeOfDegree::DoctorOfScience)
55         return "доктор наук";
56     else if (type == typeOfDegree::dontHave)
57         return "не имею";
58     return "invalid";
59 }
60 enum studyingPrograms
61 {
62     preschoolEducationPrograms, /* программы дошкольного образования */
63     generalEducationPrograms, /* программы общего образования */
64     basicProfessionalPrograms, /* основные профессиональные программы */
65     additionalEducationalPrograms /* дополнительные образовательные
       ↪  программы */
66 };
67 std::string putStudyingPrograms(studyingPrograms type)
68 {
69     if (type == studyingPrograms::preschoolEducationPrograms)
70         return "программы дошкольного образования";
71     else if (type == studyingPrograms::generalEducationPrograms)
72         return "программы общего образования";
73     else if (type == studyingPrograms::basicProfessionalPrograms)
74         return "основные профессиональные программы";
75     else if (type == studyingPrograms::additionalEducationalPrograms)
76         return "дополнительные образовательные программы";
77     else
78         return "ivalid";
79 }
80 enum typesOfLivelihoods
81 {
82     wage, /* заработная плата */
83     entrepreneurialIncome, /* предпринимательский доход, самозанятость */
84     productionOfGoodsForOwnUse, /* производство товаров для собственного
       ↪  использования */
85     rentalOfProperty, /* сдача в аренду имущества */
86     incomeFromPatents, /* доход от патентов, авторских прав */
87     capital, /* сбережения, дивиденды, проценты, ссуды, реализация
       ↪  капитала */
88     pension, /* пенсия(кроме пенсии по инвалидности) */
89     disabilityPension, /* пенсия по инвалидности */
90     scholarship, /* стипендия */
91     unemploymentBenefits, /* пособие по безработице */
```

```
92      otherBenefits, /* другие пособия и выплаты от организаций,
        ↪  государства */
93      compensation, /* льготы, компенсации, субсидии, выигрыши */
94      dependency, /* обеспечение со стороны других лиц, иждивение */
95      anotherSource /* иной источник */
96  };
97  std::string putTypesOfLivelihoods(typesOfLivelihoods type)
98  {
99      switch (type) {
100         case typesOfLivelihoods::wage:
101             return "заработная плата";
102         case typesOfLivelihoods::entrepreneurialIncome:
103             return "предпринимательский доход, самозанятость";
104         case typesOfLivelihoods::productionOfGoodsForOwnUse:
105             return "производство товаров для собственного использования";
106         case typesOfLivelihoods::rentalOfProperty:
107             return "сдача в аренду имущества";
108         case typesOfLivelihoods::incomeFromPatents:
109             return "доход от патентов, авторских прав";
110         case typesOfLivelihoods::capital:
111             return "сбережения, дивиденды, проценты, ссуды, реализация
                ↪  капитала";
112         case typesOfLivelihoods::pension:
113             return "пенсия(кроме пенсии по инвалидности)";
114         case typesOfLivelihoods::disabilityPension:
115             return "пенсия по инвалидности";
116         case typesOfLivelihoods::scholarship:
117             return "стипендия";
118         case typesOfLivelihoods::unemploymentBenefits:
119             return "пособие по безработице";
120         case typesOfLivelihoods::otherBenefits:
121             return "другие пособия и выплаты от организаций,
                ↪  государства";
122         case typesOfLivelihoods::compensation:
123             return "льготы, компенсации, субсидии, выигрыши";
124         case typesOfLivelihoods::dependency:
125             return "обеспечение со стороны других лиц, иждивение";
126         case typesOfLivelihoods::anotherSource:
127             return "иной источник";
128         default:
129             return "invalid";
130     }
131 }
132 enum positionAtWork
133 {
134     employed, /* работающий по найму */
135     businessOwner, /* владелец(совладелец) собственного предприятия(дела)
        ↪  */
136     individualEntrepreneur, /* индивидуальный предприниматель */
137     selfEmployed, /* самозанятый */
138     familyBusiness, /* помогающий на семейном предприятии */
139     otherwise /* иное */
```

```
};
std::string putPositionAtWork(positionAtWork type)
{
    switch (type) {
        case positionAtWork::employed:
            return "работающий по найму";
        case positionAtWork::businessOwner:
            return "владелец(совладелец) собственного предприятия(дела)";
        case positionAtWork::individualEntrepreneur:
            return "индивидуальный предприниматель";
        case positionAtWork::selfEmployed:
            return "самозанятый";
        case positionAtWork::familyBusiness:
            return "помогающий на семейном предприятии";
        case positionAtWork::otherwise:
            return "иное";
        default:
            return "invalid";
    }
}
enum workScheduleOptions
{
    daily, /* ежедневно */
    fewTimesAWeek, /* несколько раз в неделю */
    fewTimesAMonth, /* несколько раз в месяц */
    onceAMonthOrLess, /* один раз в месяц и реже */
    remotely, /* работаю дистанционно */
};
std::string putWorkScheduleOptions(workScheduleOptions option)
{
    switch (option) {
        case workScheduleOptions::daily:
            return "ежедневно";
        case workScheduleOptions::fewTimesAWeek:
            return "несколько раз в неделю";
        case workScheduleOptions::fewTimesAMonth:
            return "несколько раз в месяц";
        case workScheduleOptions::onceAMonthOrLess:
            return "один раз в месяц и реже";
        case workScheduleOptions::remotely:
            return "работаю дистанционно";
        default:
            return "invalid";
    }
}
enum jobOptionsMarch
{
    from25To31March, /* с 25 по 31 марта */
    from1To14April, /* с 1 по 14 апреля */
    couldNotStartDuringThesePeriods /* не смог бы приступить в эти
    ↪  периоды */
};
```

```
191 std::string putJobOptionsMarch(jobOptionsMarch type)
192 {
193     switch (type) {
194         case jobOptionsMarch::from25To31March:
195             return "с 25 по 31 марта";
196         case jobOptionsMarch::from1To14April:
197             return "с 1 по 14 апреля";
198         case jobOptionsMarch::couldNotStartDuringThesePeriods:
199             return "не смог бы приступить в эти периоды";
200         default:
201             return "invalid";
202     }
203 }
204 enum mainReasonWorkMarch
205 {
206     wasOrganizingHisOwnBusiness, /* занимался(лась) организацией
      ↪  собственного дела */
207     foundJobAndWasWaitingForAnAnswer, /* нашел(ла) работу и ждал(а)
      ↪  ответа */
208     lookingForwardToTheStartOfTheSeason, /* ожидаю начало сезона */
209     learning, /* учусь */
210     retired, /* нахожусь на пенсии */
211     forHealthReasons, /* по состоянию здоровья */
212     caringForTheSick, /* ухаживал(а) за больным */
213     ranHouseholdOrRaisedChildren, /* вел(а) домашнее хозяйство и/или
      ↪  воспитывал(а) детей */
214     canFindWork, /* не могу найти работу */
215     noNeedToWork, /* нет необходимости работать */
216     otherReason /* иная причина */
217 };
218 std::string putMainReasonWorkMarch(mainReasonWorkMarch type)
219 {
220     switch (type) {
221         case mainReasonWorkMarch::wasOrganizingHisOwnBusiness:
222             return "занимался(лась) организацией собственного дела";
223         case mainReasonWorkMarch::foundJobAndWasWaitingForAnAnswer:
224             return "нашел(ла) работу и ждал(а) ответа";
225         case mainReasonWorkMarch::lookingForwardToTheStartOfTheSeason:
226             return "ожидаю начало сезона";
227         case mainReasonWorkMarch::learning:
228             return "учусь";
229         case mainReasonWorkMarch::retired:
230             return "нахожусь на пенсии";
231         case mainReasonWorkMarch::forHealthReasons:
232             return "по состоянию здоровья";
233         case mainReasonWorkMarch::caringForTheSick:
234             return "ухаживал(а) за больным";
235         case mainReasonWorkMarch::ranHouseholdOrRaisedChildren:
236             return "вел(а) домашнее хозяйство и/или воспитывал(а) детей";
237         case mainReasonWorkMarch::canFindWork:
238             return "не могу найти работу";
239         case mainReasonWorkMarch::noNeedToWork:
```

```
240             return "нет необходимости работать";
241         case mainReasonWorkMarch::otherReason:
242             return "иная причина";
243         default:
244             return "invalid";
245     }
246 }
247 enum registrationInfoInHousehold
248 {
249     atThePlaceOfResidence, /* по месту жительства(постоянно) */
250     atThePlaceOfStay, /* по месту пребывания(временно) */
251     notRegistered /* нет */
252 };
253
254 std::string putRegistrationInfoInHousehold(registrationInfoInHousehold
    ↪ type)
255 {
256     switch (type) {
257         case registrationInfoInHousehold::atThePlaceOfResidence:
258             return "по месту жительства(постоянно)";
259         case registrationInfoInHousehold::atThePlaceOfStay:
260             return "по месту пребывания(временно)";
261         case registrationInfoInHousehold::notRegistered:
262             return "нет";
263         default:
264             return "invalid";
265     }
266 }
267 enum residenceRegistrationInfo
268 {
269     inTheSameSettlementWhereLivePermanently, /* том же населенном пункте,
        ↪ где проживаю постоянно */
270     inAnotherLocalityOfTheSameSubjectWhereLivePermanently, /* в другом
        ↪ населенном пункте того же субъекта РФ,
271                                                             * где
    ↪ проживаю постоянно */
272     inAnotherSubject, /* в другом субъекте РФ */
273     noRegistrationInTheRF, /* нет регистрации по месту жительства в
        ↪ России */
274 };
275 std::string putResidenceRegistrationInfo(residenceRegistrationInfo type)
276 {
277     switch (type) {
278         case
            ↪ residenceRegistrationInfo::inTheSameSettlementWhereLivePermanently:
279             return "том же населенном пункте, где проживаю постоянно";
280         case
            ↪ residenceRegistrationInfo::inAnotherLocalityOfTheSameSubjectWhereLivePe
281             return "в другом населенном пункте того же субъекта РФ, где
                ↪ проживаю постоянно";
282         case residenceRegistrationInfo::inAnotherSubject:
283             return "в другом субъекте РФ";
```

```
284         case residenceRegistrationInfo::noRegistrationInTheRF:
285             return "нет регистрации по месту жительства в России";
286         default:
287             return "invalid";
288     }
289 }
290 struct typeInformationAboutWork
291 {
292     std::string subject; /* субъект РФ */
293     std::string urbanSettlement; /* городской населенный пункт */
294     std::string district; /* муниципальный район/округ, городской округ
      ↪  */
295     std::string anotherState; /* иное государство */
296     bool operator==(const typeInformationAboutWork& rhs) const
297     {
298         if (subject == rhs.subject and urbanSettlement ==
          ↪  rhs.urbanSettlement and
299         district == rhs.district and anotherState == rhs.anotherState)
300             return true;
301         return false;
302     }
303     bool operator<(const typeInformationAboutWork& rhs) const
304     {
305         if (subject < rhs.subject or urbanSettlement <
          ↪  rhs.urbanSettlement or
306             district < rhs.district or anotherState < rhs.anotherState)
307             return true;
308         return false;
309     }
310     bool operator>(const typeInformationAboutWork& rhs) const
311     {
312         if (subject > rhs.subject or urbanSettlement >
          ↪  rhs.urbanSettlement or
313             district > rhs.district or anotherState > rhs.anotherState)
314             return true;
315         return false;
316     }
317 };
318 #endif //FUNDI_COURSEWORK_4_SEMESTER__CUSTOMTYPES_H
```

4.3 decorator.h

```
1 //
2 // Created by ayttekao on 5/13/21.
3 //
4 #ifndef FUNDI_COURSEWORK_4_SEMESTER__DECORATOR_H
5 #define FUNDI_COURSEWORK_4_SEMESTER__DECORATOR_H
6 template <class TypeData>
7 class Decorator
8 {
```

```
9  public:
10     Decorator() = default;
11     virtual void add(std::shared_ptr<TypeData> data) = 0;
12     virtual void remove() = 0;
13     virtual void countingStatistics() = 0;
14     virtual std::list<std::shared_ptr<TypeData>> find() = 0;
15     virtual ~Decorator() = default;
16 };
17 void generateOver(Decorator<PopulationCensus>* container, size_t count)
18 {
19     std::shared_ptr<PopulationCensus> populationCensus;
20     for (int i = 0; i < count; i++)
21     {
22         populationCensus =
           ↪ PopulationCensus::randomGenerateObjectPopulationCensus();
23         try{
24
               ↪ container->add(std::shared_ptr<PopulationCensus>(populationCensus))
25         } catch (const std::exception& e) {
26 //             std::cout << e.what();
27         }
28     }
29 }
30 #endif //FUNDI_COURSEWORK_4_SEMESTER__DECORATOR_H
```

4.4 decoratorList.h

```
1  //
2  // Created by ayttekao on 5/19/21.
3  //
4  #ifndef FUNDI_COURSEWORK_4_SEMESTER__DECORATORLIST_H
5  #define FUNDI_COURSEWORK_4_SEMESTER__DECORATORLIST_H
6  #include <memory>
7  #include <cstring>
8  #include "strategy.h"
9  #include "decorator.h"
10 #include "countingStatistics.h"
11 #include <map>
12 class DecoratorList : public Decorator<PopulationCensus>
13 {
14 private:
15     std::list<std::shared_ptr<PopulationCensus>> list;
16     std::shared_ptr<PopulationCensus> innerFind(const
       ↪ std::shared_ptr<PopulationCensus>& censusPtr,
17                                                 const
                                                    ↪ Strategy<std::shared_ptr<Popula
                                                    ↪ currentStrategy)
18     {
19         if (censusPtr == nullptr)
20             throw std::invalid_argument("");
```

```
21          int comparisonResult;
22          for (const auto& iter : list)
23          {
24              comparisonResult = currentStrategy->compare(iter,
                ↪  std::shared_ptr<PopulationCensus>(censusPtr));
25              if (comparisonResult == 0)
26                  return std::shared_ptr<PopulationCensus>(iter);
27          }
28          return nullptr;
29      }
30      static void fieldSelectionMenu()
31      {
32          std::cout << "1. Номер переписного участка\n"
33                       "2. Имя\n"
34                       "3. Место рождения\n"
35                       "4. Гражданство\n"
36                       "5. Номер бланка\n>";
37      }
38      static void createComparerAndCensus(PopulationCensus** census,
        ↪  Strategy<std::shared_ptr<PopulationCensus>>** strategy, int
        ↪  choice)
39      {
40          int inputValue;
41          std::string tmpString;
42
43          *census = new PopulationCensus();
44
45          switch (choice) {
46              case 1:
47                  std::cout << "Введите номер участка: ";
48                  std::cin >> inputValue;
49                  (*census)->setEnumerationAreaNumber(inputValue);
50                  *strategy = new StrategyEnumerationAreaNumber();
51                  break;
52              case 2:
53                  std::cout << "Введите имя: ";
54                  std::cin >> tmpString;
55                  (*census)->setFullName(tmpString);
56                  *strategy = new StrategyFullName();
57                  break;
58              case 3:
59                  std::cout << "Введите место: ";
60                  std::cin >> tmpString;
61                  (*census)->setPlaceOfBirth(tmpString);
62                  *strategy = new StrategyPlaceOfBirth();
63                  break;
64              case 4:
65                  std::cout << "Введите гражданство: ";
66                  (*census)->setCitizenship(tmpString);
67                  *strategy = new StrategyCitizenship();
68                  break;
69              case 5:
```

```
70                  std::cout << "Введите номер бланка: ";
71                  std::cin >> inputValue;
72                  (*census)->setFormNumber(inputValue);;
73                  *strategy = new StrategyFormNumber();
74                  break;
75              default:
76                  throw std::invalid_argument("");
77          }
78      }
79  public:
80      void add(std::shared_ptr<PopulationCensus> data) override
81      {
82          this->list.emplace_back(data);
83      }
84      void remove() override
85      {
86          int comparisonResult;
87          bool removeAllOccurrences;
88          std::shared_ptr<PopulationCensus> censusPtr;
89
90          PopulationCensus* census = nullptr;
91          PopulationCensus tmp;
92          Strategy<std::shared_ptr<PopulationCensus>> *strategy = nullptr;
93          std::cout << "Удалить все совпадения? (если выбрано 'нет', то
            ↪  удалится первый найденный элемент):\n"
94                    "1. Да\n"
95                    "2. Нет\n>";
96          std::cin >> comparisonResult;
97          if (comparisonResult == 1)
98              removeAllOccurrences = true;
99          else if (comparisonResult == 2)
100             removeAllOccurrences = false;
101         else
102             throw std::invalid_argument("Invalid input!");
103
104         std::cout << "Выберете поле для удаления:\n";
105         fieldSelectionMenu();
106         std::cin >> comparisonResult;
107         createComparerAndCensus(&census, &strategy, comparisonResult);
108         tmp = *census;
109         if (removeAllOccurrences) {
110             while (true) {
111                 censusPtr =
                    ↪  innerFind(std::shared_ptr<PopulationCensus>(std::shared_ptr<Pop
112                                  strategy);
113                 census = new PopulationCensus(tmp);
114                 if (censusPtr == nullptr)
115                     break;
116                 list.remove(censusPtr);
117             }
118             delete census;
119         }
```

```
120         else
121         {
122             censusPtr =
                ↪  innerFind(std::shared_ptr<PopulationCensus>(census),
                ↪  strategy);
123             if (censusPtr == nullptr)
124             {
125                 std::cout << "Данные не найдены\n";
126                 return;
127             }
128             list.remove(censusPtr);
129             std::cout << "Данные удалены\n";
130         }
131         delete strategy;
132     }
133     std::list<std::shared_ptr<PopulationCensus>> find() override
134     {
135         int comparisonResult;
136         PopulationCensus* census;
137         PopulationCensus tmp;
138         Strategy<std::shared_ptr<PopulationCensus>> *strategy = nullptr;
139         std::list<std::shared_ptr<PopulationCensus>> foundItems;
140         std::cout << "Выберете поле для поиска:\n";
141         fieldSelectionMenu();
142         std::cin >> comparisonResult;
143         createComparerAndCensus(&census, &strategy, comparisonResult);
144         tmp = *census;
145         for (const auto& iter : list)
146         {
147             comparisonResult = strategy->compare(iter,
                ↪  std::shared_ptr<PopulationCensus>(census));
148             if (comparisonResult == 0)
149                 foundItems.emplace_back(iter);
150             census = new PopulationCensus(tmp);
151         }
152         delete census;
153         delete strategy;
154         return foundItems;
155     }
156     void countingStatistics() override
157     {
158         CountingStatistics calculate(list);
159         calculate.countingStatistics();
160     }
161     DecoratorList()= default;
162     ~DecoratorList() override = default;
163
164 };
165 #endif //FUNDI_COURSEWORK_4_SEMESTER__DECORATORLIST_H
```

4.5 strategy.h

```
//
// Created by ayttekao on 4/17/21.
//
#ifndef FUNDI_7_STRATEGY_H
#define FUNDI_7_STRATEGY_H
#include <ctime>
#include "PopulationCensus.h"
template <typename T>
class Strategy{
public:
    virtual ~Strategy() = default;
    virtual int compare(const T& left, const T& right) const = 0;
};
class StrategyEnumerationAreaNumber : public
↪ Strategy<std::shared_ptr<PopulationCensus>>
{
    int compare(const std::shared_ptr<PopulationCensus>& left, const
    ↪ std::shared_ptr<PopulationCensus>& right) const override
    {
        if (left->enumerationAreaNumber == right->enumerationAreaNumber)
            return 0;
        else if (left->enumerationAreaNumber <
        ↪ right->enumerationAreaNumber)
            return -1;
        return 1;
    }
};
class StrategyFullName : public
↪ Strategy<std::shared_ptr<PopulationCensus>>
{
    int compare(const std::shared_ptr<PopulationCensus>& left, const
    ↪ std::shared_ptr<PopulationCensus>& right) const override
    {
        int comparisonResult = left->fullName.compare(right->fullName);
        if (comparisonResult > 0)
            return 1;
        else if (comparisonResult < 0)
            return -1;
        return 0;
    }
};
class StrategyPlaceOfBirth : public
↪ Strategy<std::shared_ptr<PopulationCensus>>
{
    int compare(const std::shared_ptr<PopulationCensus>& left, const
    ↪ std::shared_ptr<PopulationCensus>& right) const override
    {
        int comparisonResult =
        ↪ left->placeOfBirth.compare(right->placeOfBirth);
```

```
42          if (comparisonResult > 0)
43              return 1;
44          else if (comparisonResult < 0)
45              return -1;
46          return 0;
47      }
48  };
49  class StrategyCitizenship : public
    ↪  Strategy<std::shared_ptr<PopulationCensus>>
50  {
51      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪  std::shared_ptr<PopulationCensus>& right) const override
52      {
53          int comparisonResult = strcmp(left->citizenship.c_str(),
            ↪  right->citizenship.c_str());
54          if (comparisonResult > 0)
55              return 1;
56          else if (comparisonResult < 0)
57              return -1;
58          return 0;
59      }
60  };
61  class StrategyFormNumber : public
    ↪  Strategy<std::shared_ptr<PopulationCensus>>
62  {
63      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪  std::shared_ptr<PopulationCensus>& right) const override
64      {
65          if (left->formNumber == right->formNumber)
66              return 0;
67          else if (left->formNumber < right->formNumber)
68              return -1;
69          return 1;
70      }
71  };
72  #endif //FUNDI_7_STRATEGY_H
```

4.6 binaryTree.h

```
1  //
2  // Created by ayttekao on 4/17/21.
3  //
4  #ifndef FUNDI_7_STRATEGY_H
5  #define FUNDI_7_STRATEGY_H
6  #include <ctime>
7  #include "PopulationCensus.h"
8  template <typename T>
9  class Strategy{
10 public:
11     virtual ~Strategy() = default;
```

```
12      virtual int compare(const T& left, const T& right) const = 0;
13  };
14  class StrategyEnumerationAreaNumber : public
    ↪   Strategy<std::shared_ptr<PopulationCensus>>
15  {
16      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪   std::shared_ptr<PopulationCensus>& right) const override
17      {
18          if (left->enumerationAreaNumber == right->enumerationAreaNumber)
19              return 0;
20          else if (left->enumerationAreaNumber <
            ↪   right->enumerationAreaNumber)
21              return -1;
22          return 1;
23      }
24  };
25  class StrategyFullName : public
    ↪   Strategy<std::shared_ptr<PopulationCensus>>
26  {
27      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪   std::shared_ptr<PopulationCensus>& right) const override
28      {
29          int comparisonResult = left->fullName.compare(right->fullName);
30          if (comparisonResult > 0)
31              return 1;
32          else if (comparisonResult < 0)
33              return -1;
34          return 0;
35      }
36  };
37  class StrategyPlaceOfBirth : public
    ↪   Strategy<std::shared_ptr<PopulationCensus>>
38  {
39      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪   std::shared_ptr<PopulationCensus>& right) const override
40      {
41          int comparisonResult =
            ↪   left->placeOfBirth.compare(right->placeOfBirth);
42          if (comparisonResult > 0)
43              return 1;
44          else if (comparisonResult < 0)
45              return -1;
46          return 0;
47      }
48  };
49  class StrategyCitizenship : public
    ↪   Strategy<std::shared_ptr<PopulationCensus>>
50  {
51      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪   std::shared_ptr<PopulationCensus>& right) const override
52      {
```

```
53          int comparisonResult = strcmp(left->citizenship.c_str(),
            ↪  right->citizenship.c_str());
54          if (comparisonResult > 0)
55              return 1;
56          else if (comparisonResult < 0)
57              return -1;
58          return 0;
59      }
60  };
61  class StrategyFormNumber : public
    ↪  Strategy<std::shared_ptr<PopulationCensus>>
62  {
63      int compare(const std::shared_ptr<PopulationCensus>& left, const
        ↪  std::shared_ptr<PopulationCensus>& right) const override
64      {
65          if (left->formNumber == right->formNumber)
66              return 0;
67          else if (left->formNumber < right->formNumber)
68              return -1;
69          return 1;
70      }
71  };
72  #endif //FUNDI_7_STRATEGY_H
```

4.7 decoratorAVL.h

```
1  //
2  // Created by ayttekao on 5/19/21.
3  //
4  #ifndef FUNDI_COURSEWORK_4_SEMESTER__DECORATORAVL_H
5  #define FUNDI_COURSEWORK_4_SEMESTER__DECORATORAVL_H
6  #include "AVLTree.h"
7  #include "strategy.h"
8  #include "countingStatistics.h"
9  #include <forward_list>
10 #include <map>
11 class DecoratorAVL : public Decorator<PopulationCensus>
12 {
13 private:
14     std::forward_list<std::shared_ptr<PopulationCensus>> dataList;
15     std::map<int, BinaryTree<std::shared_ptr<PopulationCensus>>*>
       ↪  indexMap;
16     void addTreeToMap(int indexGuid,
       ↪  BinaryTree<std::shared_ptr<PopulationCensus>>* tree)
17     {
18         indexMap.insert(std::pair<int,
           ↪  BinaryTree<std::shared_ptr<PopulationCensus>>*>(indexGuid,
           ↪  tree));
19         for (const auto& iter : dataList)
20             (*tree).addNode(iter);
```

```
21      }
22      static void fieldSelectionMenu()
23      {
24          std::cout << "1. Номер переписного участка\n"
25                       "2. Имя\n"
26                       "3. Место рождения\n"
27                       "4. Гражданство\n"
28                       "5. Номер бланка\n>";
29      }
30      static void createCensus(PopulationCensus** census, int choice)
31      {
32          int inputValue;
33          std::string tmpString;
34
35          *census = new PopulationCensus();
36
37          switch (choice) {
38              case 1:
39                  std::cout << "Введите номер участка: ";
40                  std::cin >> inputValue;
41                  (*census)->setEnumerationAreaNumber(inputValue);
42                  break;
43              case 2:
44                  std::cout << "Введите имя: ";
45                  std::cin >> tmpString;
46                  (*census)->setFullName(tmpString);
47                  break;
48              case 3:
49                  std::cout << "Введите место: ";
50                  std::cin >> tmpString;
51                  (*census)->setPlaceOfBirth(tmpString);
52                  break;
53              case 4:
54                  std::cout << "Введите гражданство: ";
55                  (*census)->setCitizenship(tmpString);
56                  break;
57              case 5:
58                  std::cout << "Введите номер бланка: ";
59                  std::cin >> inputValue;
60                  (*census)->setFormNumber(inputValue);
61                  break;
62              default:
63                  throw std::invalid_argument("");
64          }
65      }
66      void deleteData(const std::shared_ptr<PopulationCensus>& data)
67      {
68          std::shared_ptr<PopulationCensus> tmpData = data;
69          for (const auto& iter : dataList)
70              if (iter == data)
71              {
72                  tmpData = data;
```

```
73                  dataList.remove(tmpData);
74                  for (std::pair<int,
                    ↪  BinaryTree<std::shared_ptr<PopulationCensus>>*> it :
                    ↪  indexMap)
75
                        ↪  it.second->deleteNode(std::shared_ptr<PopulationCensus>(tmp
76                  return;
77              }
78          throw std::invalid_argument("Value doesn't exist!");
79      }
80  public:
81      void add(std::shared_ptr<PopulationCensus> data) override
82      {
83          dataList.emplace_front(data);
84          for (std::pair<int,
            ↪  BinaryTree<std::shared_ptr<PopulationCensus>>*> it :
            ↪  indexMap)
85              it.second->addNode(data);
86      }
87      std::list<std::shared_ptr<PopulationCensus>> find() override
88      {
89          int choice;
90          std::list<std::shared_ptr<PopulationCensus>> list;
91          AVLTreeNode<std::shared_ptr<PopulationCensus>>* tmpNode;
92          PopulationCensus* census = nullptr;
93          PopulationCensus tmp;
94          std::cout << "Выберете поле для поиска:\n";
95          fieldSelectionMenu();
96          std::cin >> choice;
97          createCensus(&census, choice);
98          tmp = *census;
99          while (true)
100         {
101             tmpNode =
                ↪  static_cast<AVLTreeNode<std::shared_ptr<PopulationCensus>>
                ↪  *>(indexMap.find(
102
                        ↪  choice)->second->searchByValue(std::shared_ptr<PopulationCe
103
104             census = new PopulationCensus(tmp);
105             if (tmpNode == nullptr)
106                 break;
107
108             list.emplace_back(tmpNode->value);
109         }
110         delete census;
111
112         return list;
113     }
114     void remove() override
115     {
116         bool removeAllOccurrences;
```

```
117         int choice;
118         std::shared_ptr<PopulationCensus> censusPtr;
119         AVLTreeNode<std::shared_ptr<PopulationCensus>>* tmpNode;
120         PopulationCensus* census = nullptr;
121         PopulationCensus tmp;
122 
123         std::cout << "Удалить все совпадения? (если выбрано 'нет', то
            ↪  удалится первый найденный элемент):\n"
124                   "1. Да\n"
125                   "2. Нет\n>";
126         std::cin >> choice;
127         if (choice == 1)
128             removeAllOccurrences = true;
129         else if (choice == 2)
130             removeAllOccurrences = false;
131         else
132             throw std::invalid_argument("Invalid input!");
133 
134         std::cout << "Выберете поле для удаления:\n";
135         fieldSelectionMenu();
136         std::cin >> choice;
137         createCensus(&census, choice);
138         tmp = *census;
139         if (removeAllOccurrences)
140         {
141             while (true)
142             {
143                 tmpNode =
                    ↪  static_cast<AVLTreeNode<std::shared_ptr<PopulationCensus>>
                    ↪  *>(indexMap.find(
144 
                            ↪  choice)->second->searchByValue(std::shared_ptr<Populati
145 
146                 census = new PopulationCensus(tmp);
147                 if (tmpNode == nullptr)
148                     break;
149 
150                 deleteData(tmpNode->value);
151             }
152             delete census;
153         }
154         else
155         {
156             tmpNode =
                ↪  static_cast<AVLTreeNode<std::shared_ptr<PopulationCensus>>
                ↪  *>(indexMap.find(
157 
                        ↪  choice)->second->searchByValue(std::shared_ptr<PopulationCe
158 
159             if (tmpNode == nullptr)
160             {
161                 std::cout << "Данные не найдены\n";
```

```
162                 return;
163             }
164             deleteData(tmpNode->value);
165             std::cout << "Данные удалены\n";
166         }
167     }
168     void countingStatistics() override
169     {
170         CountingStatistics calculate(dataList);
171         calculate.countingStatistics();
172     }
173     DecoratorAVL()
174     {
175         BinaryTree<std::shared_ptr<PopulationCensus>>*
            ↪ EnumerationAreaNumberTree = new
            ↪ AVLTree<std::shared_ptr<PopulationCensus>>(new
            ↪ StrategyEnumerationAreaNumber);
176         BinaryTree<std::shared_ptr<PopulationCensus>>* FullNameTree = new
            ↪ AVLTree<std::shared_ptr<PopulationCensus>>(new
            ↪ StrategyFullName);
177         BinaryTree<std::shared_ptr<PopulationCensus>>* placeOfBirthTree =
            ↪ new AVLTree<std::shared_ptr<PopulationCensus>>(new
            ↪ StrategyPlaceOfBirth);
178         BinaryTree<std::shared_ptr<PopulationCensus>>* CitizenshipTree =
            ↪ new AVLTree<std::shared_ptr<PopulationCensus>>(new
            ↪ StrategyCitizenship);
179         BinaryTree<std::shared_ptr<PopulationCensus>>* FormNumberTree =
            ↪ new AVLTree<std::shared_ptr<PopulationCensus>>(new
            ↪ StrategyFormNumber);
180         addTreeToMap(1, EnumerationAreaNumberTree);
181         addTreeToMap(2, FullNameTree);
182         addTreeToMap(3, placeOfBirthTree);
183         addTreeToMap(4, CitizenshipTree);
184         addTreeToMap(5, FormNumberTree);
185     }
186     ~DecoratorAVL() override
187     {
188         for (std::pair<int,
            ↪ BinaryTree<std::shared_ptr<PopulationCensus>>*> it :
            ↪ indexMap)
189             delete it.second;
190     }
191 };
192 #endif //FUNDI_COURSEWORK_4_SEMESTER__DECORATORAVL_H
```

4.8 menu.h

```
1 //
2 // Created by aytteкao on 5/20/21.
3 //
```

```
4
5  #ifndef FUNDI_COURSEWORK_4_SEMESTER__MENU_H
6  #define FUNDI_COURSEWORK_4_SEMESTER__MENU_H
7
8  #include "PopulationCensus.h"
9  #include "decoratorList.h"
10 #include "decoratorAVL.h"
11
12 void menu()
13 {
14     uint16_t choice = 0;
15     uint32_t count;
16     Decorator<PopulationCensus> *decorator;
17     while (choice != 255)
18     {
19         std::cout << "Приложение для обработки данных переписи
           ↪  населения\n"
20                      "1. Сгенерировать и обработать данные\n"
21                      "2. Выход\n>";
22         std::cin >> choice;
23         switch (choice) {
24             case 1:
25                 std::cout << "Обработать данные с помощью коллекции:\n"
26                              "1. АВЛ Дерева\n"
27                              "2. Двунаправленного связанного списка
                               ↪  элементов (стандартная библиотека
                               ↪  шаблонов)\n>";
28                 std::cin >> choice;
29                 if (choice == 1)
30                     decorator = new DecoratorAVL();
31                 else if (choice == 2)
32                     decorator = new DecoratorList();
33                 else
34                 {
35                     std::cout << "Неккоректный ввод!\n";
36                     break;
37                 }
38                 std::cout << "Какое количество анкет вы хотите
                   ↪  сгенерировать?\n>";
39                 std::cin >> count;
40                 generateOver(decorator, count);
41                 while (choice != 254)
42                 {
43                     std::cout << "Что вы хотите сделать?\n"
44                                  "1. Найти форму по одному из полей\n"
45                                  "2. Удалить форму по одному из полей\n"
46                                  "3. Добавить форму\n"
47                                  "4. Вывести статистическую обработку
                                   ↪  результатов\n"
48                                  "5. Завершить работу с анкетами\n>";
49                     std::cin >> choice;
50                     switch (choice) {
```

```
51                          case 1:
52                          {
53                              std::list<std::shared_ptr<PopulationCensus>>
                                ↪  foundList;
54                              foundList = decorator->find();
55                              if (foundList.empty())
56                                  std::cout << "Поиск не дал
                                    ↪  результатов\n";
57                              break;
58                          }
59                          case 2:
60                              try {
61                                  decorator->remove();
62                              } catch (const std::exception& e){
63                                  std::cout << e.what();
64                              }
65                              break;
66                          case 3:
67                          {
68                              PopulationCensus tmpCensus;
69                              std::cin >> tmpCensus;
70
                                ↪  decorator->add(std::shared_ptr<PopulationCensus>(&t
71                              break;
72                          }
73                          case 4:
74                              decorator->countingStatistics();
75                              break;
76                          case 5:
77                              std::cout << "Завершение работы с
                                ↪  анкетами\n";
78                              choice = 254;
79                              delete decorator;
80                              break;
81                          default:
82                              std::cout << "Не удалось обработать выбор,
                                ↪  попробуйте ещё раз\n";
83                      }
84                  }
85                  break;
86              case 2:
87                  std::cout << "Завершение работы\n";
88                  choice = 255;
89                  break;
90              default:
91                  std::cout << "Не удалось обработать выбор, попробуйте ещё
                    ↪  раз\n";
92          }
93      }
94  }
95  #endif //FUNDI_COURSEWORK_4_SEMESTER__MENU_H
```

4.9 countingStatistics.h

```
//
// Created by ayttekao on 5/20/21.
//
#ifndef FUNDI_COURSEWORK_4_SEMESTER__COUNTINGSTATISTICS_H
#define FUNDI_COURSEWORK_4_SEMESTER__COUNTINGSTATISTICS_H
#include <map>
#include <forward_list>
class CountingStatistics
{
private:
    std::list<std::shared_ptr<PopulationCensus>> statisticList;

    std::map<uint16_t, uint16_t> uint16Statistics(int choice)
    {
        std::map<uint16_t, uint16_t> statistic;
        switch (choice) {
            case 1:
                for (const auto& iter : statisticList)
                {
                    auto requiredItem =
                    ↪  statistic.find(iter->getEnumerationAreaNumber());
                    if (requiredItem != statistic.end())
                        requiredItem->second++;
                    else

                        ↪  statistic.insert({iter->getEnumerationAreaNumber(),
                        ↪  1});
                }
                break;
            case 2:
                for (const auto& iter : statisticList)
                {
                    auto requiredItem =
                    ↪  statistic.find(iter->getHouseholdNumberWithinTheEnumeration
                    if (requiredItem != statistic.end())
                        requiredItem->second++;
                    else

                        ↪  statistic.insert({iter->getHouseholdNumberWithinTheEnum
                        ↪  1});
                }
                break;
            case 3:
                for (const auto& iter : statisticList)
                {
                    auto requiredItem =
                    ↪  statistic.find(iter->getFormNumber());
                    if (requiredItem != statistic.end())
                        requiredItem->second++;
```

```
43                      else
44                          statistic.insert({iter->getFormNumber(), 1});
45                  }
46                  break;
47              case 4:
48                  for (const auto& iter : statisticList)
49                  {
50                      auto requiredItem =
                        ↪  statistic.find(iter->getNumberOfYears());
51                      if (requiredItem != statistic.end())
52                          requiredItem->second++;
53                      else
54                          statistic.insert({iter->getNumberOfYears(), 1});
55                  }
56                  break;
57              case 5:
58                  for (const auto& iter : statisticList)
59                  {
60                      auto requiredItem =
                        ↪  statistic.find(iter->getAmountOfChildren());
61                      if (requiredItem != statistic.end())
62                          requiredItem->second++;
63                      else
64                          statistic.insert({iter->getAmountOfChildren(),
                            ↪  1});
65                  }
66                  break;
67              case 6:
68                  for (const auto& iter : statisticList)
69                  {
70                      auto requiredItem =
                        ↪  statistic.find(iter->getYearOfContinuousResidence());
71                      if (requiredItem != statistic.end())
72                          requiredItem->second++;
73                      else
74
                            ↪  statistic.insert({iter->getYearOfContinuousResidence(),
                            ↪  1});
75                  }
76                  break;
77              case 7:
78                  for (const auto& iter : statisticList)
79                  {
80                      auto requiredItem =
                        ↪  statistic.find(iter->getYearsOfComebackToRussia());
81                      if (requiredItem != statistic.end())
82                          requiredItem->second++;
83                      else
84
                            ↪  statistic.insert({iter->getYearsOfComebackToRussia(),
                            ↪  1});
85                  }
```

```
86                  break;
87              default:
88                  throw std::invalid_argument("Invalid input!");
89          }
90          return statistic;
91      }
92      std::map<gender, uint16_t> genderStatistics()
93      {
94          std::map<gender, uint16_t> statistic;
95          for (const auto& iter : statisticList)
96          {
97              auto requiredItem =
                ↪  statistic.find(iter->getSelectedGender());
98              if (requiredItem != statistic.end())
99                  requiredItem->second++;
100             else
101                 statistic.insert({iter->getSelectedGender(), 1});
102         }
103         return statistic;
104     }
105     std::map<struct tm*, uint16_t> dateStatistics(int choice)
106     {
107         std::map<struct tm*, uint16_t> statistic;
108         switch (choice) {
109             case 1:
110                 for (const auto& iter : statisticList)
111                 {
112                     auto requiredItem =
                        ↪  statistic.find(iter->getBirthDay());
113                     if (requiredItem != statistic.end())
114                         requiredItem->second++;
115                     else
116                         statistic.insert({iter->getBirthDay(), 1});
117                 }
118                 break;
119             case 2:
120                 for (const auto& iter : statisticList)
121                 {
122                     auto requiredItem =
                        ↪  statistic.find(iter->getBirthDayFirstChild());
123                     if (requiredItem != statistic.end())
124                         requiredItem->second++;
125                     else
126                         statistic.insert({iter->getBirthDayFirstChild(),
                            ↪  1});
127                 }
128                 break;
129             default:
130                 throw std::invalid_argument("Invalid input!");
131         }
132         return statistic;
133     }
```

```
134     std::map<marriageAnswer, uint16_t> marriageInfoStatistics()
135     {
136         std::map<marriageAnswer, uint16_t> statistic;
137         for (const auto& iter : statisticList)
138         {
139             auto requiredItem = statistic.find(iter->getMarriageInfo());
140             if (requiredItem != statistic.end())
141                 requiredItem->second++;
142             else
143                 statistic.insert({iter->getMarriageInfo(), 1});
144         }
145         return statistic;
146     }
147     std::map<yesNoAnswer, uint16_t> yesNoAnswerStatistics(int choice)
148     {
149         std::map<yesNoAnswer, uint16_t> statistic;
150         switch (choice) {
151             case 1:
152                 for (const auto& iter : statisticList)
153                 {
154                     auto requiredItem =
                        ↪  statistic.find(iter->getThatPersonSpouseLivesHousehold());
155                     if (requiredItem != statistic.end())
156                         requiredItem->second++;
157                     else
158
                            ↪  statistic.insert({iter->getThatPersonSpouseLivesHouseho
                            ↪  1});
159                 }
160                 break;
161             case 2:
162                 for (const auto& iter : statisticList)
163                 {
164                     auto requiredItem =
                        ↪  statistic.find(iter->getLivedInOtherCountries());
165                     if (requiredItem != statistic.end())
166                         requiredItem->second++;
167                     else
168
                            ↪  statistic.insert({iter->getLivedInOtherCountries(),
                            ↪  1});
169                 }
170                 break;
171             case 3:
172                 for (const auto& iter : statisticList)
173                 {
174                     auto requiredItem =
                        ↪  statistic.find(iter->getProficiencyInRussian());
175                     if (requiredItem != statistic.end())
176                         requiredItem->second++;
177                     else
```

```
178
                        ↪  statistic.insert({iter->getProficiencyInRussian(),
                        ↪  1});
179                 }
180                 break;
181             case 4:
182                 for (const auto& iter : statisticList)
183                 {
184                     auto requiredItem =
                        ↪  statistic.find(iter->getUseOfRussianLanguageInEverydayLife(
185                     if (requiredItem != statistic.end())
186                         requiredItem->second++;
187                     else
188
                        ↪  statistic.insert({iter->getUseOfRussianLanguageInEveryd
                        ↪  1});
189                 }
190                 break;
191             case 5:
192                 for (const auto& iter : statisticList)
193                 {
194                     auto requiredItem =
                        ↪  statistic.find(iter->getAbilityReadAndWrite());
195                     if (requiredItem != statistic.end())
196                         requiredItem->second++;
197                     else
198                         statistic.insert({iter->getAbilityReadAndWrite(),
                            ↪  1});
199                 }
200                 break;
201             case 6:
202                 for (const auto& iter : statisticList)
203                 {
204                     auto requiredItem =
                        ↪  statistic.find(iter->getCurrentEducation());
205                     if (requiredItem != statistic.end())
206                         requiredItem->second++;
207                     else
208                         statistic.insert({iter->getCurrentEducation(),
                            ↪  1});
209                 }
210                 break;
211             case 7:
212                 for (const auto& iter : statisticList)
213                 {
214                     auto requiredItem =
                        ↪  statistic.find(iter->getWorkForCertainPeriod());
215                     if (requiredItem != statistic.end())
216                         requiredItem->second++;
217                     else
```

```
218
                            ↪  statistic.insert({iter->getWorkForCertainPeriod(),
                            ↪  1});
219                 }
220                 break;
221             case 8:
222                 for (const auto& iter : statisticList)
223                 {
224                     auto requiredItem =
                        ↪  statistic.find(iter->getMainWorkWasInTheSameSettlement());
225                     if (requiredItem != statistic.end())
226                         requiredItem->second++;
227                     else
228
                            ↪  statistic.insert({iter->getMainWorkWasInTheSameSettleme
                            ↪  1});
229                 }
230                 break;
231             case 9:
232                 for (const auto& iter : statisticList)
233                 {
234                     auto requiredItem =
                        ↪  statistic.find(iter->getJobSearchInMarch());
235                     if (requiredItem != statistic.end())
236                         requiredItem->second++;
237                     else
238                         statistic.insert({iter->getJobSearchInMarch(),
                            ↪  1});
239                 }
240                 break;
241             default:
242                 throw std::invalid_argument("Invalid input!");
243         }
244         return statistic;
245     }
246     std::map<std::string, uint16_t> stringStatistics(int choice)
247     {
248         std::map<std::string, uint16_t> statistic;
249         switch (choice) {
250             case 0:
251                 for (const auto& iter : statisticList)
252                 {
253                     auto requiredItem =
                        ↪  statistic.find(iter->getFullName());
254                     if (requiredItem != statistic.end())
255                         requiredItem->second++;
256                     else
257                         statistic.insert({iter->getFullName(), 1});
258                 }
259                 break;
260             case 1:
261                 for (const auto& iter : statisticList)
```

```
262                 {
263                     auto requiredItem =
                    ↪  statistic.find(iter->getPlaceOfBirth());
264                     if (requiredItem != statistic.end())
265                         requiredItem->second++;
266                     else
267                         statistic.insert({iter->getPlaceOfBirth(), 1});
268                 }
269                 break;
270             case 2:
271                 for (const auto& iter : statisticList)
272                 {
273                     auto requiredItem =
                    ↪  statistic.find(iter->getPreviousPlaceOfResidence());
274                     if (requiredItem != statistic.end())
275                         requiredItem->second++;
276                     else
277
                        ↪  statistic.insert({iter->getPreviousPlaceOfResidence(),
                        ↪  1});
278                 }
279                 break;
280             case 3:
281                 for (const auto& iter : statisticList)
282                 {
283                     auto requiredItem =
                    ↪  statistic.find(iter->getPlaceOfResidenceToArrivalInRussia()
284                     if (requiredItem != statistic.end())
285                         requiredItem->second++;
286                     else
287
                        ↪  statistic.insert({iter->getPlaceOfResidenceToArrivalInR
                        ↪  1});
288                 }
289                 break;
290             case 4:
291                 for (const auto& iter : statisticList)
292                 {
293                     auto requiredItem =
                    ↪  statistic.find(iter->getNativeLanguage());
294                     if (requiredItem != statistic.end())
295                         requiredItem->second++;
296                     else
297                         statistic.insert({iter->getNativeLanguage(), 1});
298                 }
299                 break;
300             case 5:
301                 for (const auto& iter : statisticList)
302                 {
303                     auto requiredItem =
                    ↪  statistic.find(iter->getCitizenship());
304                     if (requiredItem != statistic.end())
```

```
                    requiredItem->second++;
                else
                    statistic.insert({iter->getCitizenship(), 1});
            }
            break;
        case 6:
            for (const auto& iter : statisticList)
            {
                auto requiredItem =
                ↪  statistic.find(iter->getNationality());
                if (requiredItem != statistic.end())
                    requiredItem->second++;
                else
                    statistic.insert({iter->getNationality(), 1});
            }
            break;
        default:
            throw std::invalid_argument("Invalid input!");
    }
    return statistic;
}
std::map<typeOfDegree, uint16_t> typeOfDegreeStatistics()
{
    std::map<typeOfDegree, uint16_t> statistic;
    for (const auto& iter : statisticList)
    {
        auto requiredItem =
        ↪  statistic.find(iter->getAcademicDegree());
        if (requiredItem != statistic.end())
            requiredItem->second++;
        else
            statistic.insert({iter->getAcademicDegree(), 1});
    }
    return statistic;
}
std::map<std::set<studyingPrograms>, uint16_t>
↪  studyingProgramsStatistics()
{
    std::map<std::set<studyingPrograms>, uint16_t> statistic;
    for (const auto& iter : statisticList)
    {
        auto requiredItem =
        ↪  statistic.find(iter->getCurrentEducationPrograms());
        if (requiredItem != statistic.end())
            requiredItem->second++;
        else
            statistic.insert({iter->getCurrentEducationPrograms(),
            ↪  1});
    }
    return statistic;
}
```

```
351     std::map<std::set<typesOfLivelihoods>, uint16_t>
        ↪ livelihoodsStatistics()
352     {
353         std::map<std::set<typesOfLivelihoods>, uint16_t> statistic;
354         for (const auto& iter : statisticList)
355         {
356             auto requiredItem = statistic.find(iter->getLivelihoods());
357             if (requiredItem != statistic.end())
358                 requiredItem->second++;
359             else
360                 statistic.insert({iter->getLivelihoods(), 1});
361         }
362         return statistic;
363     }
364     std::map<typesOfLivelihoods, uint16_t> mainLivelihoodStatistics()
365     {
366         std::map<typesOfLivelihoods, uint16_t> statistic;
367         for (const auto& iter : statisticList)
368         {
369             auto requiredItem =
                ↪ statistic.find(iter->getMainLivelihood());
370             if (requiredItem != statistic.end())
371                 requiredItem->second++;
372             else
373                 statistic.insert({iter->getMainLivelihood(), 1});
374         }
375         return statistic;
376     }
377     std::map<std::string, uint16_t> positionAtWorkStatistics()
378     {
379         std::map<std::string, uint16_t> statistic;
380         for (const auto& iter : statisticList)
381         {
382             auto requiredItem =
                ↪ statistic.find(putPositionAtWork(iter->getPositionMainJob()));
383             if (requiredItem != statistic.end())
384                 requiredItem->second++;
385             else
386
                    ↪ statistic.insert({putPositionAtWork(iter->getPositionMainJob())
                    ↪ 1});
387         }
388         return statistic;
389     }
390     std::map<typeInformationAboutWork, uint16_t>
        ↪ informationAboutWorkStatistics()
391     {
392         std::map<typeInformationAboutWork, uint16_t> statistic;
393         for (const auto& iter : statisticList)
394         {
395             auto requiredItem =
                ↪ statistic.find(iter->getInformationAboutWork());
```

```
396             if (requiredItem != statistic.end())
397                 requiredItem->second++;
398             else
399                 statistic.insert({iter->getInformationAboutWork(), 1});
400         }
401         return statistic;
402     }
403     std::map<workScheduleOptions, uint16_t> scheduleStatistics()
404     {
405         std::map<workScheduleOptions, uint16_t> statistic;
406         for (const auto& iter : statisticList)
407         {
408             auto requiredItem = statistic.find(iter->getSchedule());
409             if (requiredItem != statistic.end())
410                 requiredItem->second++;
411             else
412                 statistic.insert({iter->getSchedule(), 1});
413         }
414         return statistic;
415     }
416     std::map<jobOptionsMarch, uint16_t> suitableJobInMarchStatistics()
417     {
418         std::map<jobOptionsMarch, uint16_t> statistic;
419         for (const auto& iter : statisticList)
420         {
421             auto requiredItem =
                ↪  statistic.find(iter->getSuitableJobInMarch());
422             if (requiredItem != statistic.end())
423                 requiredItem->second++;
424             else
425                 statistic.insert({iter->getSuitableJobInMarch(), 1});
426         }
427         return statistic;
428     }
429     std::map<mainReasonWorkMarch, uint16_t>
        ↪  jobSearchDuringMarchStatistics()
430     {
431         std::map<mainReasonWorkMarch, uint16_t> statistic;
432         for (const auto& iter : statisticList)
433         {
434             auto requiredItem =
                ↪  statistic.find(iter->getJobSearchDuringMarch());
435             if (requiredItem != statistic.end())
436                 requiredItem->second++;
437             else
438                 statistic.insert({iter->getJobSearchDuringMarch(), 1});
439         }
440         return statistic;
441     }
442     std::map<registrationInfoInHousehold, uint16_t>
        ↪  registrationInThisHouseholdStatistics()
443     {
```

```
444         std::map<registrationInfoInHousehold, uint16_t> statistic;
445         for (const auto& iter : statisticList)
446         {
447             auto requiredItem =
                ↪  statistic.find(iter->getRegistrationInThisHousehold());
448             if (requiredItem != statistic.end())
449                 requiredItem->second++;
450             else
451                 statistic.insert({iter->getRegistrationInThisHousehold(),
                    ↪  1});
452         }
453         return statistic;
454     }
455     std::map<residenceRegistrationInfo, uint16_t>
        ↪  residenceRegistrationStatistics()
456     {
457         std::map<residenceRegistrationInfo, uint16_t> statistic;
458         for (const auto& iter : statisticList)
459         {
460             auto requiredItem =
                ↪  statistic.find(iter->getResidenceRegistration());
461             if (requiredItem != statistic.end())
462                 requiredItem->second++;
463             else
464                 statistic.insert({iter->getResidenceRegistration(), 1});
465         }
466         return statistic;
467     }
468     std::map<std::set<std::string>, uint16_t> languagesStatistics()
469     {
470         std::map<std::set<std::string>, uint16_t> statistic;
471         for (const auto& iter : statisticList)
472         {
473             auto requiredItem = statistic.find(iter->getLanguages());
474             if (requiredItem != statistic.end())
475                 requiredItem->second++;
476             else
477                 statistic.insert({iter->getLanguages(), 1});
478         }
479         return statistic;
480     }
481 public:
482     explicit CountingStatistics(const
        ↪  std::list<std::shared_ptr<PopulationCensus>>& list)
483     {
484         statisticList = list;
485     }
486     explicit CountingStatistics(const
        ↪  std::forward_list<std::shared_ptr<PopulationCensus>>&
        ↪  forwardList)
487     {
488         for (const auto& iter : forwardList)
```

```
489             statisticList.emplace_back(iter);
490     }
491     void countingStatistics()
492     {
493         int choice;
494         std::cout << "Выберите поле для сбора статистики:\n"
495                      "1. номер переписного участка\n"
496                      "2. номер помещения в пределах счетного участка\n"
497                      "3. номер бланка\n"
498                      "4. имя\n"
499                      "5. пол\n"
500                      "6. дата рождения\n"
501                      "7. число полных лет\n"
502                      "8. ваше состояние в браке\n"
503                      "9. супруг(а) этого лица проживает в
                         ↪  домохозяйстве\n"
504                      "10. сколько детей вы родили\n"
505                      "11. год рождения первого ребенка\n"
506                      "12. место вашего рождения\n"
507                      "13. с какого года вы непрерывно проживаете в этом
                         ↪  населенном пункте\n"
508                      "14. прежднее место жительства\n"
509                      "15. проживали ли выболее 12 месяцев в других
                         ↪  странах\n"
510                      "16. где вы проживали до прибытия в Россию (если
                         ↪  выбрано проживание в других странах)\n"
511                      "17. год прибытия (возвращения) в Россию\n"
512                      "18. владаете ли вы русским языком\n"
513                      "19. используете ли вы его в повседневной жизни\n"
514                      "20. какими иными языками вы владеете\n"
515                      "21. родной язык\n"
516                      "22. гражданство\n"
517                      "23. национальная принадлежность\n"
518                      "24. умеете ли вы читать и писать\n"
519                      "25. имеете ли вы ученую степень\n"
520                      "26. получаете ли вы образование в настоящее
                         ↪  время\n"
521                      "27. все программы по которым обучаетесь\n"
522                      "28. имеющиеся источники средств к существованию\n"
523                      "29. какой из отмеченных источников вы считаете для
                         ↪  себя основным\n"
524                      "30. имели ли вы какую-либо оплачиваемую работу или
                         ↪  доходное занятие с 25 по 31 марта 2021 года\n"
525                      "31. кем вы являлись на основной работе\n"
526                      "32. ваша основная работа находилась в том же
                         ↪  населенном пункте, где вы проживаете
                         ↪  постоянно\n"
527                      "33. где находилась ваша основная работа\n"
528                      "34. вы выезжали(выходили) на работу\n"
529                      "35. если бы вам предложили подходящую работу в
                         ↪  последнюю неделю марта, то когда вы смогли бы
                         ↪  приступить к ней\n"
```

```
530                     "36. вы искали работу в течении марта\n"
531                     "37. вы искали работу в течении марта\n"
532                     "38. зарегистрированы ли вы в этом помещении\n"
533                     "39. где вы зарегистрировваны по месту
                         ↪  жительства\n>";
534             std::cin >> choice;
535             switch (choice) {
536                 case 1:
537                 {
538                     std::map<uint16_t, uint16_t> statistic =
                        ↪  uint16Statistics(1);
539                     for (auto iter : statistic)
540                         std::cout << iter.first << " Статистика:\nАбсолютная:
                            ↪  " << iter.second << "\nОтносительная: " <<
                            ↪  float(iter.second) / float(statistic.size())
541                                 << std::endl;
542                     break;
543                 }
544                 case 2:
545                 {
546                     std::map<uint16_t, uint16_t> statistic =
                        ↪  uint16Statistics(2);
547                     for (auto iter : statistic)
548                         std::cout << iter.first << " Статистика:\nАбсолютная:
                            ↪  " << iter.second << "\nОтносительная: " <<
                            ↪  float(iter.second) / float(statistic.size())
549                                 << std::endl;
550                     break;
551                 }
552                 case 3:
553                 {
554                     std::map<uint16_t, uint16_t> statistic =
                        ↪  uint16Statistics(3);
555                     for (auto iter : statistic)
556                         std::cout << iter.first << " Статистика:\nАбсолютная:
                            ↪  " << iter.second << "\nОтносительная: " <<
                            ↪  float(iter.second) / float(statistic.size())
557                                 << std::endl;
558                     break;
559                 }
560                 case 4:
561                 {
562                     std::map<std::string, uint16_t> statistic =
                        ↪  stringStatistics(0);
563                     for (const auto& iter : statistic)
564                         std::cout << iter.first << " Статистика:\nАбсолютная:
                            ↪  " << iter.second << "\nОтносительная: " <<
                            ↪  float(iter.second) / float(statistic.size())
565                                 << std::endl;
566                     break;
567                 }
568                 case 5:
```

```
569             {
570                 std::map<gender, uint16_t> statistic =
                    ↪ genderStatistics();
571                 for (auto iter : statistic)
572                     std::cout << (iter.first ? "мужской" : "женский") <<
                        ↪ " Статистика:\nАбсолютная: " << iter.second <<
                        ↪ "\nОтносительная: " << float(iter.second) /
                        ↪ float(statistic.size())
573                               << std::endl;
574                 break;
575             }
576             case 6:
577             {
578                 std::map<struct tm*, uint16_t> statistic =
                    ↪ dateStatistics(1);
579                 for (auto iter : statistic)
580                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪ " << iter.second << "\nОтносительная: " <<
                        ↪ float(iter.second) / float(statistic.size())
581                               << std::endl;
582                 break;
583             }
584             case 7:
585             {
586                 std::map<uint16_t, uint16_t> statistic =
                    ↪ uint16Statistics(4);
587                 for (auto iter : statistic)
588                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪ " << iter.second << "\nОтносительная: " <<
                        ↪ float(iter.second) / float(statistic.size())
589                               << std::endl;
590                 break;
591             }
592             case 8:
593             {
594                 std::map<marriageAnswer, uint16_t> statistic =
                    ↪ marriageInfoStatistics();
595                 for (auto iter : statistic)
596                     std::cout << putMarriageAnswer(iter.first) << "
                        ↪ Статистика:\nАбсолютная: " << iter.second <<
                        ↪ "\nОтносительная: "
597                               << float(iter.second) /
                                  ↪ float(statistic.size()) << std::endl;
598                 break;
599             }
600             case 9:
601             {
602                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪ yesNoAnswerStatistics(1);
603                 for (auto iter : statistic)
```

```
604                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
605                           << std::endl;
606             break;
607         }
608         case 10:
609         {
610             std::map<uint16_t, uint16_t> statistic =
                ↪  uint16Statistics(5);
611             for (auto iter : statistic)
612                 std::cout << iter.first << " Статистика:\nАбсолютная:
                    ↪  " << iter.second << "\nОтносительная: " <<
                    ↪  float(iter.second) / float(statistic.size())
613                           << std::endl;
614             break;
615         }
616         case 11:
617         {
618             std::map<struct tm*, uint16_t> statistic =
                ↪  dateStatistics(2);
619             for (auto iter : statistic)
620                 std::cout << iter.first << " Статистика:\nАбсолютная:
                    ↪  " << iter.second << "\nОтносительная: " <<
                    ↪  float(iter.second) / float(statistic.size())
621                           << std::endl;
622             break;
623         }
624         case 12:
625         {
626             std::map<std::string, uint16_t> statistic =
                ↪  stringStatistics(1);
627             for (const auto& iter : statistic)
628                 std::cout << iter.first << " Статистика:\nАбсолютная:
                    ↪  " << iter.second << "\nОтносительная: " <<
                    ↪  float(iter.second) / float(statistic.size())
629                           << std::endl;
630             break;
631         }
632         case 13:
633         {
634             std::map<uint16_t, uint16_t> statistic =
                ↪  uint16Statistics(6);
635             for (auto iter : statistic)
636                 std::cout << iter.first << " Статистика:\nАбсолютная:
                    ↪  " << iter.second << "\nОтносительная: " <<
                    ↪  float(iter.second) / float(statistic.size())
637                           << std::endl;
638             break;
639         }
640         case 14:
```

```
641             {
642                 std::map<std::string, uint16_t> statistic =
                    ↪ stringStatistics(2);
643                 for (const auto& iter : statistic)
644                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪ " << iter.second << "\nОтносительная: " <<
                        ↪ float(iter.second) / float(statistic.size())
645                               << std::endl;
646                 break;
647             }
648             case 15:
649             {
650                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪ yesNoAnswerStatistics(2);
651                 for (auto iter : statistic)
652                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪ Статистика:\nАбсолютная: " << iter.second <<
                        ↪ "\nОтносительная: " << float(iter.second) /
                        ↪ float(statistic.size())
653                               << std::endl;
654                 break;
655             }
656             case 16:
657             {
658                 std::map<std::string, uint16_t> statistic =
                    ↪ stringStatistics(3);
659                 for (const auto& iter : statistic)
660                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪ " << iter.second << "\nОтносительная: " <<
                        ↪ float(iter.second) / float(statistic.size())
661                               << std::endl;
662                 break;
663             }
664             case 17:
665             {
666                 std::map<uint16_t, uint16_t> statistic =
                    ↪ uint16Statistics(7);
667                 for (auto iter : statistic)
668                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪ " << iter.second << "\nОтносительная: " <<
                        ↪ float(iter.second) / float(statistic.size())
669                               << std::endl;
670                 break;
671             }
672             case 18:
673             {
674                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪ yesNoAnswerStatistics(3);
675                 for (auto iter : statistic)
```

```
676                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
677                               << std::endl;
678                 break;
679             }
680             case 19:
681             {
682                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪  yesNoAnswerStatistics(4);
683                 for (auto iter : statistic)
684                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
685                               << std::endl;
686                 break;
687             }
688             case 20:
689             {
690                 std::map<std::set<std::string>, uint16_t> statistic =
                    ↪  languagesStatistics();
691                 for (const auto& iter : statistic)
692                 {
693                     for (const auto &secondIter : iter.first)
694                         std::cout << "'" << secondIter << "'";
695                     std::cout << " Статистика:\nАбсолютная: " <<
                        ↪  iter.second << "\nОтносительная: " <<
                        ↪  float(iter.second) / float(statistic.size())
696                               << std::endl;
697                 }
698                 break;
699             }
700             case 21:
701             {
702                 std::map<std::string, uint16_t> statistic =
                    ↪  stringStatistics(4);
703                 for (const auto& iter : statistic)
704                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪  " << iter.second << "\nОтносительная: " <<
                        ↪  float(iter.second) / float(statistic.size())
705                               << std::endl;
706                 break;
707             }
708             case 22:
709             {
710                 std::map<std::string, uint16_t> statistic =
                    ↪  stringStatistics(5);
711                 for (const auto& iter : statistic)
```

```
712                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪  " << iter.second << "\nОтносительная: " <<
                        ↪  float(iter.second) / float(statistic.size())
713                           << std::endl;
714                 break;
715             }
716             case 23:
717             {
718                 std::map<std::string, uint16_t> statistic =
                    ↪  stringStatistics(6);
719                 for (const auto& iter : statistic)
720                     std::cout << iter.first << " Статистика:\nАбсолютная:
                        ↪  " << iter.second << "\nОтносительная: " <<
                        ↪  float(iter.second) / float(statistic.size())
721                           << std::endl;
722                 break;
723             }
724             case 24:
725             {
726                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪  yesNoAnswerStatistics(5);
727                 for (auto iter : statistic)
728                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
729                           << std::endl;
730                 break;
731             }
732             case 25:
733             {
734                 std::map<typeOfDegree, uint16_t> statistic =
                    ↪  typeOfDegreeStatistics();
735                 for (auto iter : statistic)
736                     std::cout << putTypeOfDegree(iter.first) << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
737                           << std::endl;
738                 break;
739             }
740             case 26:
741             {
742                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪  yesNoAnswerStatistics(6);
743                 for (auto iter : statistic)
744                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
745                           << std::endl;
746                 break;
```

```
747             }
748             case 27:
749             {
750                 std::map<std::set<studyingPrograms>, uint16_t> statistic
                    ↪  = studyingProgramsStatistics();
751                 for (const auto& iter : statistic)
752                 {
753                     for (const auto &secondIter : iter.first)
754                         std::cout << "'" <<
                            ↪  putStudyingPrograms(secondIter) << "'";
755                     std::cout << " Статистика:\nАбсолютная: " <<
                        ↪  iter.second << "\nОтносительная: " <<
                        ↪  float(iter.second) / float(statistic.size())
756                               << std::endl;
757                 }
758                 break;
759             }
760             case 28:
761             {
762                 std::map<std::set<typesOfLivelihoods>, uint16_t>
                    ↪  statistic = livelihoodsStatistics();
763                 for (const auto& iter : statistic)
764                 {
765                     for (const auto &secondIter : iter.first)
766                         std::cout << "'" <<
                            ↪  putTypesOfLivelihoods(secondIter) << "'";
767                     std::cout << " Статистика:\nАбсолютная: " <<
                        ↪  iter.second << "\nОтносительная: " <<
                        ↪  float(iter.second) / float(statistic.size())
768                               << std::endl;
769                 }
770                 break;
771             }
772             case 29:
773             {
774                 std::map<typesOfLivelihoods, uint16_t> statistic =
                    ↪  mainLivelihoodStatistics();
775                 for (auto iter : statistic)
776                     std::cout << putTypesOfLivelihoods(iter.first) << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
777                               << std::endl;
778                 break;
779             }
780             case 30:
781             {
782                 std::map<yesNoAnswer, uint16_t> statistic =
                    ↪  yesNoAnswerStatistics(7);
783                 for (auto iter : statistic)
```

```
784                     std::cout << (iter.first ? "да" : "нет") << "
                        ↪  Статистика:\nАбсолютная: " << iter.second <<
                        ↪  "\nОтносительная: " << float(iter.second) /
                        ↪  float(statistic.size())
785                               << std::endl;
786             break;
787         }
788         case 31:
789         {
790             std::map<std::string, uint16_t> statistic =
                ↪  positionAtWorkStatistics();
791             for (const auto& iter : statistic)
792                 std::cout << iter.first << " Статистика:\nАбсолютная:
                    ↪  " << iter.second << "\nОтносительная: " <<
                    ↪  float(iter.second) / float(statistic.size())
793                               << std::endl;
794             break;
795         }
796         case 32:
797         {
798             std::map<yesNoAnswer, uint16_t> statistic =
                ↪  yesNoAnswerStatistics(8);
799             for (auto iter : statistic)
800                 std::cout << (iter.first ? "да" : "нет") << "
                    ↪  Статистика:\nАбсолютная: " << iter.second <<
                    ↪  "\nОтносительная: " << float(iter.second) /
                    ↪  float(statistic.size())
801                               << std::endl;
802             break;
803         }
804         case 33:
805         {
806             std::map<typeInformationAboutWork, uint16_t> statistic =
                ↪  informationAboutWorkStatistics();
807             for (const auto& iter : statistic)
808                 std::cout << "'" << iter.first.subject << "'" << "'"
                    ↪  << iter.first.urbanSettlement << "'" << "'"
809                               << iter.first.district << "'" << "'" <<
                                ↪  iter.first.anotherState << "'" << "
                                ↪  Статистика:\nАбсолютная: "
810                               << iter.second << "\nОтносительная: " <<
                                ↪  float(iter.second) /
                                ↪  float(statistic.size()) << std::endl;
811             break;
812         }
813         case 34:
814         {
815             std::map<workScheduleOptions, uint16_t> statistic =
                ↪  scheduleStatistics();
816             for (auto iter : statistic)
817                 std::cout << putWorkScheduleOptions(iter.first) << "
                    ↪  Статистика:\nАбсолютная: " << iter.second
```

```
818                                 << "\nОтносительная: " <<
                                    ↪  float(iter.second) /
                                    ↪  float(statistic.size()) << std::endl;
819             break;
820         }
821         case 35:
822         {
823             std::map<jobOptionsMarch, uint16_t> statistic =
                ↪  suitableJobInMarchStatistics();
824             for (auto iter : statistic)
825                 std::cout << putJobOptionsMarch(iter.first) << "
                    ↪  Статистика:\nАбсолютная: " << iter.second
826                             << "\nОтносительная: " <<
                                ↪  float(iter.second) /
                                ↪  float(statistic.size()) << std::endl;
827             break;
828         }
829         case 36:
830         {
831             std::map<yesNoAnswer, uint16_t> statistic =
                ↪  yesNoAnswerStatistics(9);
832             for (auto iter : statistic)
833                 std::cout << (iter.first ? "да" : "нет") << "
                    ↪  Статистика:\nАбсолютная: " << iter.second <<
                    ↪  "\nОтносительная: " << float(iter.second) /
                    ↪  float(statistic.size())
834                             << std::endl;
835             break;
836         }
837         case 37:
838         {
839             std::map<mainReasonWorkMarch, uint16_t> statistic =
                ↪  jobSearchDuringMarchStatistics();
840             for (auto iter : statistic)
841                 std::cout << putMainReasonWorkMarch(iter.first) << "
                    ↪  Статистика:\nАбсолютная: " << iter.second
842                             << "\nОтносительная: " <<
                                ↪  float(iter.second) /
                                ↪  float(statistic.size()) << std::endl;
843             break;
844         }
845         case 38:
846         {
847             std::map<registrationInfoInHousehold, uint16_t> statistic
                ↪  = registrationInThisHouseholdStatistics();
848             for (auto iter : statistic)
849                 std::cout <<
                    ↪  putRegistrationInfoInHousehold(iter.first) <<
                    ↪  "\n->Статистика:\nАбсолютная: " << iter.second
850                             << "\nОтносительная: " <<
                                ↪  float(iter.second) /
                                ↪  float(statistic.size()) << std::endl;
```

```
851                 break;
852             }
853             case 39:
854             {
855                 std::map<residenceRegistrationInfo, uint16_t> statistic =
                    ↪ residenceRegistrationStatistics();
856                 for (auto iter : statistic)
857                     std::cout << putResidenceRegistrationInfo(iter.first)
                        ↪ << " Статистика:\nАбсолютная: " << iter.second
858                               << "\nОтносительная: " <<
                                  ↪ float(iter.second) /
                                  ↪ float(statistic.size()) << std::endl;
859                 break;
860             }
861             default:
862                 throw std::invalid_argument("Invalid input!");
863         }
864     }
865 };
866 #endif //FUNDI_COURSEWORK_4_SEMESTER__COUNTINGSTATISTICS_H
```

4.10 main.cpp

```
1 #include "menu.h"
2 int main()
3 {
4     menu();
5 }
```