_list.h

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: лист и splay-дерево»

Студент	Алимов Исмаил Рифатович
Группа	М80-211Б-19
Преподаватель	Романенков Александр Михайлович
Дата	24 мая 2021 г.

Оценка: _____

Подпись преподавателя: _____

Подпись студента: _____

Москва 2021

Содержание

1 Описание задачи

Вариант №48

Разработайте приложение для обработки данных по налоговой отчётности. Придумайте форму отчётности (или найдите готовую в сети). Основным требованием является достаточный объем критериев (не менее 20) с различными видами значений: в виде числа, с единичным или множественным выбором из предложенных вариантов, либо в виде развернутого текста. Ваше приложение должно обеспечить возможность хранения и обработки отчётностей. 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 отчётностей. Реализуйте возможности добавления отчётностей, удаления отчётностей. Для демонстрации работы реализуйте генератор, выдающий случайным образом заполненные формы отчётностей (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 splay-дерева (с компаратором-стратегией).

2 Описание решения

Для эффективного выполнения курсовой работы задача была разбита на несколько подзадач. Для выполнения каждой подзадачи были реаизованы следующие файлы: основной файл, файл с полями информации и функцией их заполнения соответственно, файл с компаратором - стратегией, файл с абстрактным шаблонным классом для бинарного дерева, файл с реализацией splay дерева, файлы с подсчётом статистики для splay дерева и списка, файл с декоратором для листа, файл с реализацией паттерна "фабричный метод "для генерации формы отчетностей и файл с общим интерфейсом. Далее подробнее объясняется о функциях каждого файла - заголовка.

2.1 binary_tree.h и strategy.h

binary_tree.h - это базовый абстрактный класс дерева от которого наследуется splay дерево. Класс содержит три чистых виртуальных функции: добавления элемента в дерево, удаления элемента из дерева и поиска элемента в дереве по значению.

strategy.h - это компаратор - стратегия для обеспечения возможности поиска форм по различным критериям (имени, гражданству, id номера и тд).

2.2 splay_tree.h

splay_tree.h - это полная реализация splay дерева с присущими ему поворотами и балансировкой, а также публичные функции для интерфейса пользователя и функции приватные для выпонения всей работы (добавить, найти, удалить и тд).

2.3 stat_splay.h и stat_list.h

Выпоняют вычисление статистики по выбранному параметру с помощью контейнера map, где ключ - это искомый параметр (int или string), а значение - количество этих самых параметров (int). В случае splay дерева сначала происходит получение всех его элементов, в случае листа - элементы уже есть и ими заполняется map. Потом с помощью обхода вычисляется абсолютная и относительная частота.

2.4 doc.h

doc.h - содержит структуру Document, состоящую из 20 полей, которая лежит в классе Doc_form, эмулирующую анкету. Также содержит процедуру filling(), которая заполняет случайными значениями форму.

2.5 list_realization.h

list_realization.h - содержит общий декоратор Interface, от которого наследуется класс декоратора для листа с набором функций необходимых по заданию, которые "завёртывают"функции листа в функции с нужными названиями.

2.6 factory_method.h

Задаёт метод, который используется вместо вызова оператора new для создания объектов-продуктов. Подклассы переопределяют этот метод, чтобы изменять тип создаваемых продуктов.

2.7 user_interaction.h

Создаются нужные компараторы и два заполненных интерфейса. После происходит опрос пользователя с помощью цикла и функции switch().

2.8 main.cpp

Запрашивает количество создаваемых анкет (для отладки достаточно 10). А после переходит к "диалогу"с пользователем.

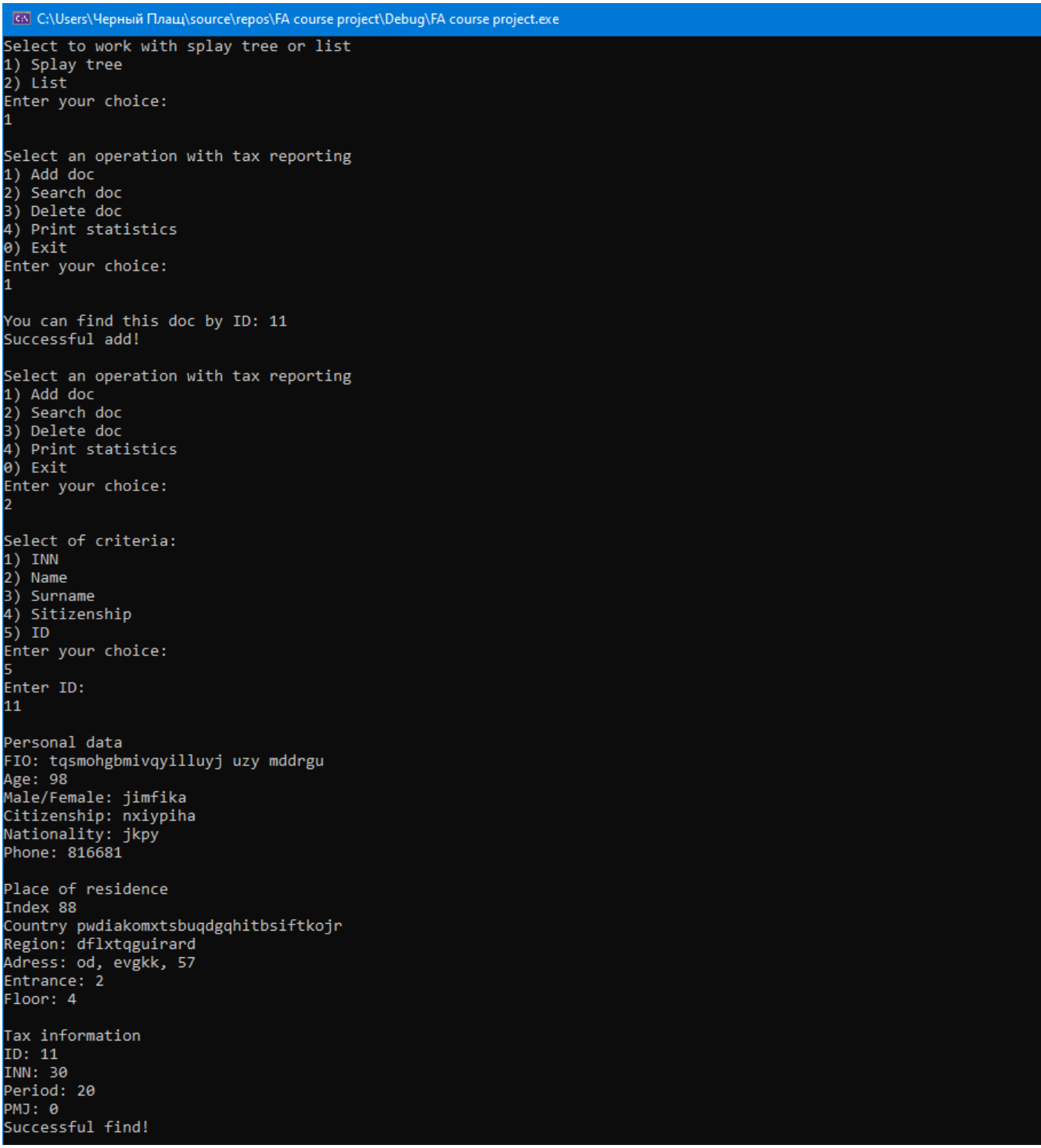

Рис 1. Успешное добавление и поиск элемента в программе

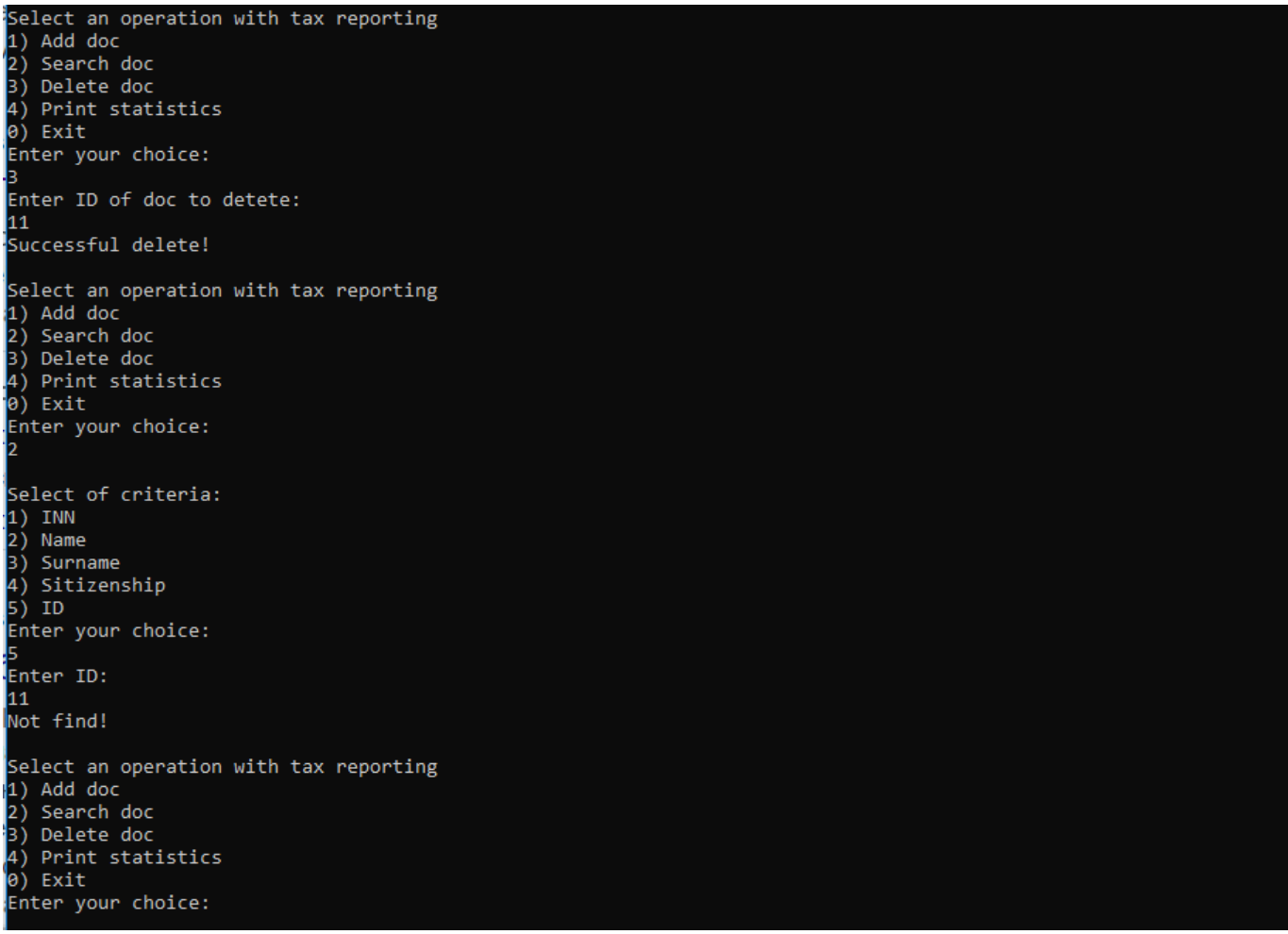

Рис 2. Успешное удаление элемента в программе

3 Вывод

В ходе выпонения данной курсовой работы было разработано приложение для обработки данных по налоговой отчётности. Была придумана форма отчётности с достаточным объемом критериев с различными видами значений. Приложение обеспечивает хранение и обработку отчётностей, обеспечивает возможность поиска форм по различным критериям, статистическую обработку результатов и добавление или удаление отчетностей. Также был реализован генератор на основе паттерна "фабричный метод". Обработка данных реализуется с использованием контейнера std::list и собственной реализации splay-дерева (с компаратором-стратегией).

Стоит отметить, что добавление и удаление элементов удобнее осуществляется на основе splay дерева, нежели на основе списка, за счет более высокой скорости: Olog(N) против O(n). Со вставкой элементов противоположная ситуация: Olog(n) у splay дерева против O(1) у списка.

4 Приложение

4.1 Файлы заголовков

4.1.1 doc.h

```
#pragma once
#include <iostream>
#include <string>
#include <ctime>

using namespace std;

struct Document
{
    string surname;
    string name;
    string patronymic;
    int age;
    string gender;
    string citizenship;
    string nationality;
    string phone;

    int index;
    string country;
    string region;
    string city;
    string street;
    int apartment;
    int entrance;
    int floor;

    int registr;
    int inn;
    int period;
    int pmj;

    void print()
    {
        cout << "\nPersonal data " << endl;
        cout << "FIO: " << surname << " " << name << " " << patronymic << endl;
        cout << "Age: " << age << endl;
        cout << "Male/Female: " << gender << endl;
        cout << "Citizenship: " << citizenship << endl;
        cout << "Nationality: " << nationality << endl;
        cout << "Phone: " << phone << endl;
        cout << endl;

        cout << "Place of residence " << endl;
        cout << "Index " << index << endl;
        cout << "Country " << country << endl;
        cout << "Region: " << region << endl;
        cout << "Adress: " << city << ", " << street << ", " << apartment << endl;
        cout << "Entrance: " << entrance << endl;
        cout << "Floor: " << floor << endl;
        cout << endl;

        cout << "Tax information " << endl;
        cout << "ID: " << registr << endl;
```

```
        cout << "INN: " << inn << endl;
        cout << "Period: " << period << endl;
        cout << "PMJ: " << pmj << endl;
    }
};

class Form
{
public:

    virtual ~Form() {}
    virtual void filling() = 0;
};

class Doc_form : public Form
{

private: static int id;

public: Document info;

public:
    Doc_form()
    {
        info.registr = 0;
        info.inn = 0;
    }

    ~Doc_form()
    {
        info.surname = "";
        info.name = "";
        info.patronymic = "";
        info.age = 0;
        info.gender = "";
        info.citizenship = "";
        info.nationality = "";
        info.phone = "";

        info.index = 0;
        info.country = "";
        info.region = "";
        info.city = "";
        info.street = "";
        info.apartment = 0;
        info.entrance = 0;
        info.floor = 0;

        info.registr = 0;
        info.inn = 0;
        info.period = 0;
        info.pmj = 0;
    }

    void filling() override
    {
        int tmp_int = 0;
        string tmp_str;

        tmp_str = string_generator(1 + rand() % 20);
        info.surname = tmp_str;
```

```
116         tmp_str = string_generator(1 + rand() % 18);
117         info.name = tmp_str;
118         info.patronymic = string_generator(1 + rand() % 12);
119         info.age = 18 + rand() % 100;
120         tmp_str = string_generator(1 + rand() % 10);
121         info.gender = tmp_str;
122         tmp_str = string_generator(1 + rand() % 9);
123         info.citizenship = tmp_str;
124         tmp_str = string_generator(1 + rand() % 8);
125         info.nationality = tmp_str;
126         info.phone = to_string(int_generator(7));
127 
128         info.index = int_generator(1 + rand() % 5);
129         info.country = string_generator(1 + rand() % 44);
130         info.region = string_generator(1 + rand() % 32);
131         info.city = string_generator(1 + rand() % 19);
132         info.street = string_generator(1 + rand() % 10);
133         info.apartment = 1 + rand() % 100;
134         info.entrance = 1 + rand() % 15;
135         info.floor = 1 + rand() % 10;
136 
137         tmp_int = id * 3;
138         info.inn = tmp_int;
139         info.period = int_generator(1 + rand() % 3);
140         info.pmj = rand() % 4;
141         info.registr = ++id;
142     }
143 
144     string string_generator(int size)
145     {
146         int j = 0;
147         string string;
148 
149         for (int i = 0; i < size; i++)
150         {
151             j = rand() % 50 + 1;
152 
153             if (j > size - i) j = size - i;
154             i += j;
155 
156             for (int k = 0; k < j; k++)
157             {
158                 string += (char)(rand() % 26 + 97);
159             }
160             break;
161             string += ' ';
162         }
163         return string;
164     }
165 
166     int int_generator(int size)
167     {
168         string tmp;
169         for (int i = 0; i < size; i++)
170         {
171             if (i == 10)
172                 break;
173             int j = (rand() % 9);
174             tmp += to_string(j);
175         }
176         return stoi(tmp);
```

```
177     }
178 };
179
180 int Doc_form::id = 0;
```

4.1.2 strategy.h

```
1  #pragma once
2  #include "doc.h"
3
4  template <typename T, typename _allocator = allocator<T>>
5  class Strategy
6  {
7  public:
8      virtual ~Strategy() {}
9      virtual int compare(const T& obj1, const T& obj2) const = 0;
10 };
11
12 class Documents_inn_strategy : public Strategy<Doc_form*> {
13 public:
14     int compare(Doc_form* const& obj1, Doc_form* const& obj2) const override {
15         if (obj1->info.inn != obj2->info.inn) {
16             return obj1->info.inn < obj2->info.inn ? -1 : 1;
17         }
18         else {
19             return 0;
20         }
21     }
22 };
23
24 class Documents_name_strategy : public Strategy<Doc_form*> {
25 public:
26     int compare(Doc_form* const& obj1, Doc_form* const& obj2) const override {
27         if (obj1->info.name != obj2->info.name) {
28             return obj1->info.name < obj2->info.name ? -1 : 1;
29         }
30         else {
31             return 0;
32         }
33     }
34 };
35
36 class Documents_surname_strategy : public Strategy<Doc_form*> {
37 public:
38     int compare(Doc_form* const& obj1, Doc_form* const& obj2) const override {
39         if (obj1->info.surname != obj2->info.surname) {
40             return obj1->info.surname < obj2->info.surname ? -1 : 1;
41         }
42         else {
43             return 0;
44         }
45     }
46 };
47
48 class Documents_citizenship_strategy : public Strategy<Doc_form*> {
49 public:
50     int compare(Doc_form* const& obj1, Doc_form* const& obj2) const override {
51         if (obj1->info.citizenship != obj2->info.citizenship) {
52             return obj1->info.citizenship < obj2->info.citizenship ? -1 : 1;
```

```
        }
        else {
            return 0;
        }
    }
};

class Documents_registr_strategy : public Strategy<Doc_form*> {
public:
    int compare(Doc_form* const& obj1, Doc_form* const& obj2) const override {
        if (obj1->info.registr != obj2->info.registr) {
            return obj1->info.registr < obj2->info.registr ? -1 : 1;
        }
        else {
            return 0;
        }
    }
};
```

4.1.3 binary_tree.h

```
#pragma once
#include "strategy.h"

template <typename T>
class Binary_tree
{
protected:
        Strategy<T>* comparator;

public:
        Binary_tree(Strategy <T>* compare = nullptr) : comparator(compare)
        {}
        virtual ~Binary_tree()
        {}

        virtual bool find_node(T&& val) = 0;
        virtual void add_node(T&& val) = 0;
        virtual bool delete_node(T&& val) = 0;
};
```

4.1.4 splay_tree.h

```
#pragma once
#include "binary_tree.h"
#include <vector>
#define _CRT_SECURE_NO_WARNINGS

using namespace std;

template <typename T, typename _allocator = allocator<T>>
class Splay_tree : public Binary_tree<T>
{
private:
        struct tree_node
        {
                T data;
                tree_node* left;
```

```
16                  tree_node* right;
17          };
18
19          tree_node* root;
20          vector <tree_node*> way;
21
22          bool find_node(tree_node* node, T val);
23          void delete_tree(tree_node* node);
24          void pretty_print(tree_node* curr, int depth);
25          void tree_go(tree_node* node);
26          tree_node* create_tree(T val);
27          tree_node* return_find(tree_node* node, T val);
28          tree_node* split_megre(tree_node* node, T val);
29          tree_node* find_max(tree_node* node);
30          tree_node* splay(tree_node* node, T val);
31          tree_node* rotate_left(tree_node* y);
32          tree_node* rotate_right(tree_node* x);
33          tree_node* zigzig_right(tree_node* x);
34          tree_node* zigzig_left(tree_node* x);
35          tree_node* zigzag_right(tree_node* x);
36          tree_node* zigzag_left(tree_node* x);
37          tree_node* add_node(tree_node* node, T val);
38          tree_node* delete_node(tree_node* node, T val);
39
40  public:
41          Splay_tree(Strategy<T>* compare);
42          ~Splay_tree();
43
44          void add_node(T&& val);
45          void add_node(T& val);
46          bool find_node(T&& val);
47          bool delete_node(T&& val);
48          bool is_empty();
49          void print_tree();
50          void clear_tree();
51          bool delete_with_comp(T& val, Strategy<T>* compare);
52          bool delete_with_comp(T&& val, Strategy<T>* compare);
53          bool find_with_comp(T& val, Strategy<T>* compare);
54          bool find_with_comp(T&& val, Strategy<T>* compare);
55          void tree_go(vector <T>& fil);
56          void statistics();
57  };
58
59  // конструктор
60  template <typename T, typename _allocator>
61  Splay_tree<T, _allocator>::Splay_tree(Strategy<T>* compare) :
    ↪  Binary_tree<T>::Binary_tree(compare)
62  {
63          root = nullptr;
64          way.clear();
65  }
66
67  // деструктор
68  template <typename T, typename _allocator>
69  Splay_tree<T, _allocator>::~Splay_tree()
70  {
71          if (root != nullptr)
72                  delete_tree(this->root);
73          way.clear();
74  }
75
```

```
76 // приватные методы
77 template <typename T, typename _allocator>
78 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
   ↪  _allocator>::create_tree(T val)
79 {
80         tree_node* new_node = new tree_node;
81 
82         new_node->data = val;
83         new_node->left = nullptr;
84         new_node->right = nullptr;
85         return new_node;
86 }
87 
88 template <typename T, typename _allocator>
89 void Splay_tree<T, _allocator>::pretty_print(tree_node* curr, int depth)
90 {
91         int i;
92         int rec[100];
93 
94         if (curr == nullptr)
95                 return;
96         for (i = 0; i < depth; i++)
97         {
98                 if (i == depth - 1)
99                 {
100                         if (rec[depth - 1] == 0)
101                                 cout << " + ";
102                         else
103                                 cout << " L ";
104                         cout << "---";
105                 }
106                 else
107                         cout << "       ";
108         }
109         cout << " " << *(curr->data) << endl;
110         rec[depth] = 1;
111         pretty_print(curr->left, depth + 1);
112         rec[depth] = 0;
113         pretty_print(curr->right, depth + 1);
114 }
115 
116 template <typename T, typename _allocator>
117 void Splay_tree<T, _allocator>::delete_tree(tree_node* node)
118 {
119         if (node != nullptr)
120         {
121                 delete_tree(node->left);
122                 delete_tree(node->right);
123                 delete node;
124         }
125 }
126 
127 template <typename T, typename _allocator>
128 bool Splay_tree<T, _allocator>::find_node(tree_node* node, T val)
129 {
130         if (node != nullptr)
131         {
132                 if (this->comparator->compare(val, (*node).data) == 0)
133                         return true;
134                 if (this->comparator->compare(val, (*node).data) < 0)
135                         return find_node(node->left, val);
```

```
136                 else
137                         return find_node(node->right, val);
138         }
139         return false;
140 }
141
142 template <typename T, typename _allocator>
143 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::add_node(tree_node* node, T val)
144 {
145         if (node == nullptr)
146                 node = create_tree(val);
147         else if (this->comparator->compare(val, (*node).data) < 0)
148         {
149                 way.push_back(node);
150                 add_node(node->left, val);
151         }
152         else
153         {
154                 way.push_back(node);
155                 add_node(node->right, val);
156         }
157         return splay(node, val);
158 }
159
160 template <typename T, typename _allocator>
161 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::return_find(tree_node* node, T val)
162 {
163         if (this->comparator->compare(val, (*node).data) == 0)
164         {
165                 return splay(node, val);
166         }
167
168         if (this->comparator->compare(val, (*node).data) < 0)
169         {
170                 way.push_back(node);
171                 return (splay(return_find(node->left, val), val));
172         }
173         else
174         {
175                 way.push_back(node);
176                 return (splay(return_find(node->right, val), val));
177         }
178         return (splay(node, val));
179 }
180
181 template <typename T, typename _allocator>
182 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::delete_node(tree_node* node, T val)
183 {
184         return_find(root, val);
185         split_megre(node, val);
186         return root;
187 }
188
189 template <typename T, typename _allocator>
190 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::split_megre(tree_node* node, T val)
191 {
192         tree_node* right_del;
```

```
193         tree_node* to_del = find_max(root->left);
194 
195         if (to_del == root)
196         {
197                 if (root->right == nullptr)
198                 {
199                         tree_node* tmp = root->left;
200                         root->left = root->right = nullptr;
201                         root = tmp;
202                         return root;
203                 }
204                 else if (root->left == nullptr)
205                 {
206                         tree_node* tmp = root->right;
207                         root->left = root->right = nullptr;
208                         root = tmp;
209                         return root;
210                 }
211         }
212         else
213                 return_find(root->left, to_del->data);
214         to_del = root->left;
215         right_del = root->right;
216         root->left = root->right = nullptr;
217         to_del->right = right_del;
218         root = to_del;
219         return root;
220 }
221 
222 template <typename T, typename _allocator>
223 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::find_max(tree_node* node)
224 {
225         if (node == nullptr)
226                 return root;
227         if (node->right != nullptr)
228                 return (find_max(node->right));
229         return node;
230 }
231 
232 template <typename T, typename _allocator>
233 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::splay(tree_node* node, T orig)
234 {
235         int flag = 0;
236         int len = way.size();
237 
238         if (root == nullptr)
239         {
240                 root = node;
241                 return root;
242         }
243         if (!way.empty())
244         {
245                 if (this->comparator->compare(way[way.size() - 1]->data, node->data) >
                    ↪  0 && way[way.size() - 1]->left == nullptr &&
                    ↪  this->comparator->compare(orig, node->data) == 0)
246                         way[way.size() - 1]->left = node;
247                 else if (way[way.size() - 1]->right == nullptr &&
                    ↪  this->comparator->compare(orig, node->data) == 0 && way[way.size()
                    ↪  - 1]->left != node)
```

```
248                         way[way.size() - 1]->right = node;
249
250                 if (way.size() == 1)
251                 {
252                         if (way[0]->right != nullptr &&
                        ↪  this->comparator->compare(orig, way[0]->right->data) == 0)
253                         {
254                                 root = rotate_left(way[way.size() - 1]);
255                                 way.clear();
256                         }
257                         else if (way[0]->left != nullptr &&
                        ↪  this->comparator->compare(orig, way[0]->left->data) == 0)
258                         {
259                                 root = rotate_right(way[way.size() - 1]);
260                                 way.clear();
261                         }
262                 }
263                 else
264                 {
265                         if (way.size() == 2)
266                         {
267                                 flag = 1;
268                                 tree_node* empt = new tree_node;
269                                 empt->left = way[0];
270                                 empt->right = nullptr;
271                                 way.insert(way.begin(), empt);
272                         }
273                         if (way[way.size() - 2]->right == way[way.size() - 1] &&
                        ↪  way[way.size() - 1]->right != nullptr &&
                        ↪  this->comparator->compare(orig, way[way.size() -
                        ↪  1]->right->data) == 0)
274                         {
275                                 if (way[way.size() - 3]->left != nullptr &&
                                ↪  way[way.size() - 3]->left == way[way.size() - 2])
276                                         way[way.size() - 3]->left =
                                        ↪  zigzig_left(way[way.size() - 3]);
277                                 else if (way[way.size() - 3]->right != nullptr &&
                                ↪  way[way.size() - 3]->right == way[way.size() - 2])
278                                         way[way.size() - 3]->right =
                                        ↪  zigzig_left(way[way.size() - 3]);
279                                 way.resize(way.size() - 2);
280                         }
281                         else if (way[way.size() - 2]->right == way[way.size() - 1] &&
                        ↪  this->comparator->compare(orig, way[way.size() -
                        ↪  1]->left->data) == 0)
282                         {
283                                 if (way[way.size() - 3]->left != nullptr &&
                                ↪  way[way.size() - 3]->left == way[way.size() - 2])
284                                         way[way.size() - 3]->left =
                                        ↪  zigzag_right(way[way.size() - 3]);
285                                 else if (way[way.size() - 3]->right != nullptr &&
                                ↪  way[way.size() - 3]->right == way[way.size() - 2])
286                                         way[way.size() - 3]->right =
                                        ↪  zigzag_right(way[way.size() - 3]);
287                                 way.resize(way.size() - 2);
288                         }
289                         else if (way[way.size() - 2]->left == way[way.size() - 1] &&
                        ↪  way[way.size() - 1]->left != nullptr &&
                        ↪  this->comparator->compare(orig, way[way.size() -
                        ↪  1]->left->data) == 0)
290                         {
```

```
291                                 if (way[way.size() - 3]->left != nullptr &&
                                    ↪  way[way.size() - 3]->left == way[way.size() - 2])
292                                         way[way.size() - 3]->left =
                                            ↪  zigzig_right(way[way.size() - 3]);
293                                 else if (way[way.size() - 3]->right != nullptr &&
                                    ↪  way[way.size() - 3]->right == way[way.size() - 2])
294                                         way[way.size() - 3]->right =
                                            ↪  zigzig_right(way[way.size() - 3]);
295                                 way.resize(way.size() - 2);
296                         }
297                         else if (way[way.size() - 2]->left == way[way.size() - 1] &&
                            ↪  this->comparator->compare(orig, way[way.size() -
                            ↪  1]->right->data) == 0)
298                         {
299                                 if (way[way.size() - 3]->left != nullptr &&
                                    ↪  way[way.size() - 3]->left == way[way.size() - 2])
300                                         way[way.size() - 3]->left =
                                            ↪  zigzag_left(way[way.size() - 3]);
301                                 else if (way[way.size() - 3]->right != nullptr &&
                                    ↪  way[way.size() - 3]->right == way[way.size() - 2])
302                                         way[way.size() - 3]->right =
                                            ↪  zigzag_left(way[way.size() - 3]);
303                                 way.resize(way.size() - 2);
304                         }
305                         if (flag == 1)
306                         {
307                                 root = way[0]->left;
308                                 delete way[0];
309                                 way.erase(way.begin());
310                                 flag = 0;
311                         }
312                 }
313         }
314         if (way.empty() && len && this->comparator->compare(orig, (*node).data) == 0)
315                 root = node;
316         return root;
317 }
318
319 template <typename T, typename _allocator>
320 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::rotate_right(tree_node* x)
321 {
322         tree_node* y;
323
324         y = x->left;
325         x->left = y->right;
326         y->right = x;
327         return y;
328 }
329
330 template <typename T, typename _allocator>
331 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::rotate_left(tree_node* y)
332 {
333         tree_node* x;
334
335         x = y->right;
336         y->right = x->left;
337         x->left = y;
338         return x;
339 }
```

```
340
341 template <typename T, typename _allocator>
342 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::zigzig_right(tree_node* x)
343 {
344         tree_node* y;
345
346         if (x->left != nullptr && way[way.size() - 2] == x->left)
347                 y = rotate_right(x->left);
348         else
349                 y = rotate_right(x->right);
350         (x->left != nullptr && way[way.size() - 2] == x->left) ? x->left = y :
          ↪  x->right = y;
351         tree_node* z = rotate_right(y);
352         return z;
353 }
354
355 template <typename T, typename _allocator>
356 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::zigzig_left(tree_node* x)
357 {
358         tree_node* y;
359
360         if (x->left == nullptr || way[way.size() - 2] == x->right)
361                 y = rotate_left(x->right);
362         else
363                 y = rotate_left(x->left);
364         (x->left == nullptr || way[way.size() - 2] == x->right) ? x->right = y :
          ↪  x->left = y;
365         tree_node* z = rotate_left(y);
366         return z;
367 }
368
369 template <typename T, typename _allocator>
370 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::zigzag_right(tree_node* x)
371 {
372         tree_node* y;
373         if (x->right == nullptr || way[way.size() - 2] == x->left)
374                 y = rotate_right(x->left->right);
375         else
376                 y = rotate_right(x->right->right);
377         (x->right == nullptr || way[way.size() - 2] == x->left) ? x->left->right = y :
          ↪  x->right->right = y;
378         tree_node* z;
379         if (x->right == nullptr || way[way.size() - 2] == x->left)
380                 z = rotate_left(x->left);
381         else
382                 z = rotate_left(x->right);
383         (x->right == nullptr || way[way.size() - 2] == x->left) ? x->left = z :
          ↪  x->right = z;
384         return z;
385 }
386
387 template <typename T, typename _allocator>
388 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
    ↪  _allocator>::zigzag_left(tree_node* x)
389 {
390         tree_node* y;
391         if (x->left == nullptr || way[way.size() - 2] == x->right)
392                 y = rotate_left(x->right->left);
```

```
393         else
394                 y = rotate_left(x->left->left);
395         (x->left == nullptr || way[way.size() - 2] == x->right) ? x->right->left = y :
            ↪  x->left->left = y;
396         tree_node* z;
397         if (x->left == nullptr || way[way.size() - 2] == x->right)
398                 z = rotate_right(x->right);
399         else
400                 z = rotate_right(x->left);
401         (x->left == nullptr || way[way.size() - 2] == x->right) ? x->right = z :
            ↪  x->left = z;
402         return z;
403 }
404
405 template <typename T, typename _allocator>
406 void Splay_tree<T, _allocator>::tree_go(tree_node* node)
407 {
408         if (node != nullptr)
409         {
410                 way.push_back(node);
411                 tree_go(node->left);
412                 tree_go(node->right);
413         }
414 }
415
416 // публичные методы
417
418 template <typename T, typename _allocator>
419 void Splay_tree<T, _allocator>::add_node(T& val)
420 {
421         root = add_node(root, val);
422 }
423
424 template <typename T, typename _allocator>
425 void Splay_tree<T, _allocator>::add_node(T&& val)
426 {
427         root = add_node(root, val);
428 }
429
430 template <typename T, typename _allocator>
431 bool Splay_tree<T, _allocator>::find_node(T&& val)
432 {
433         return find_node(root, val);
434 }
435
436 template <typename T, typename _allocator>
437 bool Splay_tree<T, _allocator>::is_empty()
438 {
439         return (root == nullptr);
440 }
441
442 template <typename T, typename _allocator>
443 bool Splay_tree<T, _allocator>::delete_node(T&& val)
444 {
445         if (find_node(static_cast<T&&>(val)))
446         {
447                 delete_node(root, val);
448                 return true;
449         }
450         else
451         {
```

```
452                 return false;
453         }
454 }
455
456 template <typename T, typename _allocator>
457 bool Splay_tree<T, _allocator>::delete_with_comp(T& val, Strategy<T>* compare)
458 {
459         if (root->left == nullptr && root->right == nullptr)
460         {
461                 delete root;
462                 root = nullptr;
463         }
464         else
465         {
466                 if (compare != this->comparator)
467                         return false;
468                 if (find_node(static_cast<T&&>(val)))
469                 {
470                         delete_node(root, val);
471                         return true;
472                 }
473                 else
474                 {
475                         return false;
476                 }
477         }
478 }
479
480 template <typename T, typename _allocator>
481 bool Splay_tree<T, _allocator>::delete_with_comp(T&& val, Strategy<T>* compare)
482 {
483         if (root->left == nullptr && root->right == nullptr)
484         {
485                 delete root;
486                 root = nullptr;
487         }
488         else
489         {
490                 if (compare != this->comparator)
491                         return false;
492                 if (find_node(static_cast<T&&>(val)))
493                 {
494                         delete_node(root, val);
495                         return true;
496                 }
497                 else
498                 {
499                         return false;
500                 }
501         }
502 }
503
504 template <typename T, typename _allocator>
505 void Splay_tree<T, _allocator>::print_tree()
506 {
507         if (root != nullptr)
508                 pretty_print(this->root, 0);
509 }
510
511 template <typename T, typename _allocator>
512 void Splay_tree<T, _allocator>::clear_tree()
```

```
513 {
514         if (root != nullptr)
515                 delete_tree(root);
516         root = nullptr;
517 }
518
519 template <typename T, typename _allocator>
520 bool Splay_tree<T, _allocator>::find_with_comp(T& val, Strategy<T>* compare)
521 {
522         if (compare != this->comparator)
523         {
524                 Splay_tree <T>* for_search = new Splay_tree<T>(compare);
525                 vector <T> tmp;
526                 tree_go(tmp);
527                 for (int i = 0; i < tmp.size(); i++)
528                 {
529                         for_search->add_node(tmp[i]);
530                 }
531                 return for_search->find_with_comp(static_cast<T&&>(val), compare);
532         }
533         else
534         {
535                 if (find_node(root, val))
536                 {
537                         (return_find(root, val))->data->info.print();
538                         return true;
539                 }
540                 else
541                 {
542                         return false;
543                 }
544         }
545 }
546
547 template <typename T, typename _allocator>
548 bool Splay_tree<T, _allocator>::find_with_comp(T&& val, Strategy<T>* compare)
549 {
550         if (compare != this->comparator)
551         {
552                 Splay_tree <T>* for_search = new Splay_tree<T>(compare);
553                 vector <T> tmp;
554                 tree_go(tmp);
555                 for (int i = 0; i < tmp.size(); i++)
556                 {
557                         for_search->add_node(tmp[i]);
558                 }
559                 return for_search->find_with_comp(static_cast<T&&>(val), compare);
560         }
561         else
562         {
563                 if (find_node(root, val))
564                 {
565                         (return_find(root, val))->data->info.print();
566                         return true;
567                 }
568                 else
569                 {
570                         return false;
571                 }
572         }
573 }
```

```
574
575 template <typename T, typename _allocator>
576 void Splay_tree<T, _allocator>::tree_go(vector <T>& fil)
577 {
578         tree_go(root);
579         for (auto it = way.begin(); it != way.end(); it++)
580                 fil.push_back((*it)->data);
581         way.clear();
582 }
```

4.1.5 stat_splay.h

```
1  #pragma once
2  #include <iostream>
3  #include "binary_tree.h"
4  #include "splay_tree.h"
5  #include <vector>
6  #include <map>
7
8  using namespace std;
9
10
11 int stat_choice()
12 {
13         cout << "\nAt what parameter you want to show statistics?" << endl;
14         cout << "1) Inn" << endl;
15         cout << "2) Name" << endl;
16         cout << "3) Surname" << endl;
17         cout << "4) Citizenship" << endl;
18         cout << "5) ID" << endl;
19         cout << "0) Exit" << endl;
20         cout << "Enter choice:" << endl;
21         int choice;
22
23         cin >> choice;
24         cin.ignore();
25         if (choice < 0 || choice > 5)
26         {
27                 cout << "Wrong choice!" << endl;
28                 return 0;
29         }
30
31         return choice;
32 }
33
34 template <typename T, typename _allocator>
35 void Splay_tree<T, _allocator>::statistics()
36 {
37         int data_size;
38         map< int, int > stat_int;
39         map< string, int > stat_string;
40         vector <T> tmp_per;
41         tree_go(tmp_per);
42         int choice = stat_choice();
43         switch (choice)
44         {
45         case (0):
46                 return;
47         case (1):
```

```
48
49                  for (int i = 0; i < tmp_per.size(); i++)
50                          stat_int[tmp_per[i]->info.inn]++;
51                  data_size = tmp_per.size();
52
53                  for (map< int, int >::iterator it = stat_int.begin(); it !=
                    ↪  stat_int.end(); ++it)
54                  {
55                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
56                  }
57                  break;
58          case (2):
59
60                  for (int i = 0; i < tmp_per.size(); i++)
61                          stat_string[tmp_per[i]->info.name]++;
62                  data_size = tmp_per.size();
63
64                  for (map< string, int >::iterator it = stat_string.begin(); it !=
                    ↪  stat_string.end(); ++it)
65                  {
66                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
67                  }
68                  break;
69          case (3):
70
71                  for (int i = 0; i < tmp_per.size(); i++)
72                          stat_string[tmp_per[i]->info.surname]++;
73                  data_size = tmp_per.size();
74
75                  for (map< string, int >::iterator it = stat_string.begin(); it !=
                    ↪  stat_string.end(); ++it)
76                  {
77                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
78                  }
79                  break;
80          case (4):
81
82                  for (int i = 0; i < tmp_per.size(); i++)
83                          stat_string[tmp_per[i]->info.citizenship]++;
84                  data_size = tmp_per.size();
85
86                  for (map< string, int >::iterator it = stat_string.begin(); it !=
                    ↪  stat_string.end(); ++it)
87                  {
88                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
89                  }
90                  break;
91          case (5):
92
93                  for (int i = 0; i < tmp_per.size(); i++)
94                          stat_int[tmp_per[i]->info.registr]++;
95                  data_size = tmp_per.size();
96
```

```
97                  for (map< int, int >::iterator it = stat_int.begin(); it !=
                    ↪  stat_int.end(); ++it)
98                  {
99                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
100                 }
101                 break;
102         }
103 }
```

4.1.6 list_realization.h

```
1  #pragma once
2  #include "binary_tree.h"
3  #include "splay_tree.h"
4  #include "stat_splay.h"
5  #include "stat_list.h"
6  #include "factory_method.h"
7  #include <list>
8  #include <map>
9  #include <vector>
10
11 using namespace std;
12
13 int stat_choice();
14
15 template <template<typename TargetType, typename _allocator = allocator<TargetType>>
   ↪  typename Container, typename TargetType>
16 class Interface
17 {
18 private:
19
20         Container<TargetType>* _decorated_container;
21
22 public:
23
24         Interface(Container<TargetType>* container)
25         {
26                 _decorated_container = container;
27         }
28
29         ~Interface() {}
30
31         void add_elem(TargetType& data_to_add)
32         {
33                 _decorated_container->add_node(data_to_add);
34         }
35
36         void add_elem(TargetType&& data_to_add)
37         {
38                 _decorated_container->add_node(static_cast<TargetType&&>(data_to_add));
39         }
40
41         bool del_elem(const TargetType& data_to_del, Strategy<TargetType>* compare)
42         {
43                 return _decorated_container->delete_with_comp(data_to_del, compare);
44         }
45
```

```
46          bool del_elem(TargetType&& data_to_del, Strategy<TargetType>* compare)
47          {
48                  return
                    ↪  _decorated_container->delete_with_comp(static_cast<TargetType&&>(data_to_del),
                    ↪  compare);
49          }
50
51          bool find_elem(TargetType&& data_to_find, Strategy<TargetType>* compare)
52          {
53                  return
                    ↪  _decorated_container->find_with_comp(static_cast<TargetType&&>(data_to_find),
                    ↪  compare);
54          }
55          bool find_elem(const TargetType& data_to_find, Strategy<TargetType>* compare)
56          {
57                  return _decorated_container->find_with_comp(data_to_find, compare);
58          }
59          void statistic()
60          {
61                  _decorated_container->statistics();
62          }
63
64  };
65
66  template <typename T, typename _allocator = allocator<T>>
67  class List_interface
68  {
69  private:
70          list <T> data;
71  public:
72          List_interface() {}
73          ~List_interface()
74          {
75                  data.clear();
76          }
77
78          void add_node_one(list <T>* tmp)
79          {
80                  data = *tmp;
81          }
82
83          void add_node(const T& to_add)
84          {
85                  data.push_front(to_add);
86          }
87
88          void add_node(T&& to_add)
89          {
90                  data.push_front(data.begin(), static_cast<T&&>(to_add));
91
92          }
93          bool delete_with_comp(const T& to_del, Strategy<T>* comp)
94          {
95                  typename list<T>::iterator it = data.begin();
96                  it = find_if(data.begin(), data.end(), [comp, to_del](T cur)
97                          {return (comp->compare(cur, to_del) == 0); });
98                  if (it == data.end())
99                  {
100                         return false;
101                 }
102                 else
```

```
103                 {
104                         data.erase(it);
105                 }
106                 return true;
107         }
108         bool delete_with_comp(T&& to_del, Strategy<T>* comp)
109         {
110                 typename list<T>::iterator it = data.begin();
111                 it = find_if(data.begin(), data.end(), [comp, to_del](T cur)
112                         {return (comp->compare(cur, to_del) == 0); });
113                 if (it == data.end())
114                 {
115                         return false;
116                 }
117                 else
118                 {
119                         data.erase(it);
120                 }
121                 return true;
122         }
123
124         bool find_with_comp(T&& to_f, Strategy<T>* comp)
125         {
126                 typename list<T>::iterator it = data.begin();
127                 it = find_if(data.begin(), data.end(), [comp, to_f](T& tmp)
128                         {return (comp->compare(tmp, to_f) == 0); });
129                 if (it == data.end())
130                 {
131                         return false;
132                 }
133                 else
134                 {
135                         (*it)->info.print();
136                         to_f = *it;
137                         return true;
138                 }
139         }
140
141         bool find_with_comp(T& to_f, Strategy<T>* comp)
142         {
143                 typename list<T>::iterator it = data.begin();
144                 it = find_if(data.begin(), data.end(), [comp, to_f](T& tmp)
145                         {return (comp->compare(tmp, to_f) == 0); });
146                 if (it == data.end())
147                 {
148                         return false;
149                 }
150                 else
151                 {
152                         it->info.print();
153                         to_f = *it;
154                         return true;
155                 }
156         }
157
158         void statistics();
159
160 };
```

4.1.7 stat_list.h

```
#pragma once
#include <iostream>
#include <map>
#include <list>
#include "list_realization.h"

using namespace std;

template <typename T, typename _allocator>
void List_interface<T, _allocator>::statistics()
{
        int data_size = data.size();
        map< int, int > stat_int;
        map< string, int > stat_string;
        int choice = stat_choice();
        switch (choice)
        {
        case (0):
                return;
        case (1):
                for (const auto& it : data)
                {
                        stat_int[(int)it->info.inn]++;
                }
                for (map< int, int >::iterator it = stat_int.begin(); it !=
                ↪  stat_int.end(); ++it)
                {
                        cout << it->first << ":  Absolute frequency: " << it->second
                        ↪  << "\tRelative frequency: " << (double)it->second /
                        ↪  (double)data_size << endl;
                }
                break;
        case (2):
                for (const auto& it : data)
                {
                        stat_string[it->info.name]++;
                }
                for (map< string, int >::iterator it = stat_string.begin(); it !=
                ↪  stat_string.end(); ++it)
                {
                        cout << it->first << ":  Absolute frequency: " << it->second
                        ↪  << "\tRelative frequency: " << (double)it->second /
                        ↪  (double)data_size << endl;
                }
                break;
        case (3):
                for (const auto& it : data)
                {
                        stat_string[it->info.surname]++;
                }
                for (map< string, int >::iterator it = stat_string.begin(); it !=
                ↪  stat_string.end(); ++it)
                {
                        cout << it->first << ":  Absolute frequency: " << it->second
                        ↪  << "\tRelative frequency: " << (double)it->second /
                        ↪  (double)data_size << endl;
                }

```

```
50                  break;
51          case (4):
52                  for (const auto& it : data)
53                  {
54                          stat_string[it->info.citizenship]++;
55                  }
56                  for (map< string, int >::iterator it = stat_string.begin(); it !=
                    ↪  stat_string.end(); ++it)
57                  {
58                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
59                  }
60                  break;
61          case (5):
62                  for (const auto& it : data)
63                  {
64                          stat_int[(int)it->info.inn]++;
65                  }
66                  for (map< int, int >::iterator it = stat_int.begin(); it !=
                    ↪  stat_int.end(); ++it)
67                  {
68                          cout << it->first << ":  Absolute frequency: " << it->second
                            ↪  << "\tRelative frequency: " << (double)it->second /
                            ↪  (double)data_size << endl;
69                  }
70                  break;
71          }
72  }
```

4.1.8 factory_method.h

```
#pragma once
#include <iostream>
#include "doc.h"

using namespace std;

class Creator
{
public:
    virtual ~Creator() {};
    virtual Form* factory_method() const = 0;
    Form* Generate() const
    {
        Form* product = this->factory_method();
        product->filling();
        return product;
    }
};

class Create_Tax : public Creator
{
public:

      Create_Tax() : Creator() {}

public:

    Form* factory_method() const override
    {
        return new Doc_form();
    }
};
```

4.1.9 user_interaction.h

```
#pragma once
#include <iostream>
#include <algorithm>
#include <string>
#include <iostream>
#include <list>
#include "stat_splay.h"
#include "stat_list.h"

using namespace std;

int get_doc()
{
    int choice;

    cout << "Choose the amount of documents:" << endl;
    cout << "1) 100" << endl;
    cout << "2) 50000" << endl;
    cout << "Enter your choice:" << endl;
    cin >> choice;

    if (choice < 0 || choice > 2)
    {
        cout << "Wrong choice!" << endl;
        exit(-1);
    }
    else if (choice == 1)
    {
        return 100;
    }
    else
        return 50000;
}

void inter_choice(int choice)
{
    switch (choice)
    {
    case(0):
        cout << "\nSelect to work with splay tree or list" << endl;
        cout << "1) Splay tree" << endl;
        cout << "2) List" << endl;
        cout << "Enter your choice:" << endl;
        break;
    case(1):
        cout << "\nSelect an operation with tax reporting" << endl;
        cout << "1) Add doc" << endl;
        cout << "2) Search doc" << endl;
        cout << "3) Delete doc" << endl;
        cout << "4) Print statistics" << endl;
        cout << "0) Exit" << endl;
        cout << "Enter your choice:" << endl;
        break;
    case(2):
        cout << "\nSelect of criteria:" << endl;
        cout << "1) INN" << endl;
        cout << "2) Name" << endl;
        cout << "3) Surname" << endl;
```

```
59          cout << "4) Sitizenship" << endl;
60          cout << "5) ID" << endl;
61          cout << "Enter your choice:" << endl;
62          break;
63      }
64  }
65
66  void user_interaction(vector <Doc_form*> database, Creator* make, int doc)
67  {
68          int cont_ch = 0, what_do = 0, what_find, rand_tax = 0;
69          Doc_form ad, del, fnd;
70      Strategy <Doc_form*>* compreg = new Documents_registr_strategy;
71          Strategy <Doc_form*>* compn = new Documents_name_strategy;
72          Strategy <Doc_form*>* compcs = new Documents_citizenship_strategy;
73          Strategy <Doc_form*>* compsn = new Documents_surname_strategy;
74          Strategy <Doc_form*>* compin = new Documents_inn_strategy;
75
76          List_interface <Doc_form*>* lst = new List_interface<Doc_form*>;
77          Interface<List_interface, Doc_form*> for_list(lst);
78          Splay_tree<Doc_form*>* spl = new Splay_tree<Doc_form*>(compreg);
79          Interface<Splay_tree, Doc_form*> for_splay(spl);
80
81          for (int i = 0; i < doc; i++)
82          {
83                  for_splay.add_elem(database[i]);
84                  for_list.add_elem(database[i]);
85          }
86
87      inter_choice(0);
88      cin >> cont_ch;
89      cin.ignore();
90      switch (cont_ch)
91      {
92      case(1):
93          while (1)
94          {
95              inter_choice(1);
96              cin >> what_do;
97              cin.ignore();
98              switch (what_do)
99              {
100             case(0):
101                 return;
102             case(1):
103                 if (1)
104                 {
105                     Doc_form* chance = dynamic_cast<Doc_form*>(make->Generate());
106                     for_splay.add_elem(chance);
107
108                     cout << "\nYou can find this doc by ID: " << chance->info.registr
                        ↪  << endl;
109                     cout << "Successful add!" << endl;
110                 }
111                 break;
112
113             case(2):
114                 inter_choice(2);
115                 cin >> what_find;
116                 cin.ignore();
117                 switch (what_find)
118                 {
```

```
119                 case(1):
120                     cout << "Enter INN:" << endl;
121                     cin >> fnd.info.inn;
122                     cin.ignore();
123                     if (for_splay.find_elem(&fnd, compin))
124                         cout << "Successful find!" << endl;
125                     else
126                         cout << "Not find!" << endl;
127                     break;
128                 case(2):
129                     cout << "Enter name:" << endl;
130                     cin >> fnd.info.name;
131                     cin.ignore();
132                     if (for_splay.find_elem(&fnd, compn))
133                         cout << "Successful find!" << endl;
134                     else
135                         cout << "Not find!" << endl;
136                     break;
137                 case(3):
138                     cout << "Enter surname:" << endl;
139                     cin >> fnd.info.surname;
140                     cin.ignore();
141                     if (for_splay.find_elem(&fnd, compsn))
142                         cout << "Successful find!" << endl;
143                     else
144                         cout << "Not find!" << endl;
145                     break;
146                 case(4):
147                     cout << "Enter citizenship:" << endl;
148                     cin >> fnd.info.citizenship;
149                     cin.ignore();
150                     if (for_splay.find_elem(&fnd, compcs))
151                         cout << "Successful find!" << endl;
152                     else
153                         cout << "Not find!" << endl;
154                     break;
155                 case(5):
156                     cout << "Enter ID:" << endl;
157                     cin >> fnd.info.registr;
158                     cin.ignore();
159                     if (for_splay.find_elem(&fnd, compreg))
160                         cout << "Successful find!" << endl;
161                     else
162                         cout << "Not find!" << endl;
163                     break;
164                 default:
165                     cout << "Wrong parameter" << endl;
166                     break;
167                 }
168                 break;
169             case(3):
170                 cout << "Enter ID of doc to detete:" << endl;
171                 cin >> del.info.registr;
172                 cin.ignore();
173                 if (for_splay.del_elem(&del, compreg))
174                     cout << "Successful delete!" << endl;
175                 else
176                     cout << "Wrong parameter" << endl;
177                 break;
178             case(4):
179                 for_splay.statistic();
```

```
180                 break;
181
182             default:
183                 cout << "Invalid choise" << endl;
184                 break;
185             }
186         }
187         break;
188     case (2):
189     {
190         while (1)
191         {
192             inter_choice(1);
193             cin >> what_do;
194             cin.ignore();
195             switch (what_do)
196             {
197             case(0):
198                 return;
199             case(1):
200                 if (1)
201                 {
202                     Doc_form* chance = dynamic_cast<Doc_form*>(make->Generate());
203                     for_list.add_elem(chance);
204
205                     cout << "\nYou can find this doc by ID: " << chance->info.registr
                        ↪  << endl;
206                     cout << "Successful add!" << endl;
207                     break;
208                 }
209
210             case(2):
211                 inter_choice(2);
212                 cin >> what_find;
213                 cin.ignore();
214                 switch (what_find)
215                 {
216                 case(1):
217                     cout << "Enter INN:" << endl;
218                     cin >> fnd.info.inn;
219                     cin.ignore();
220                     if (for_list.find_elem(&fnd, compin))
221                         cout << "Successful find!" << endl;
222                     else
223                         cout << "Not find!" << endl;
224                     break;
225                 case(2):
226                     cout << "Enter name:" << endl;
227                     cin >> fnd.info.name;
228                     cin.ignore();
229                     if (for_list.find_elem(&fnd, compn))
230                         cout << "Successful find!" << endl;
231                     else
232                         cout << "Not find!" << endl;
233                     break;
234                 case(3):
235                     cout << "Enter surname:" << endl;
236                     cin >> fnd.info.surname;
237                     cin.ignore();
238                     if (for_list.find_elem(&fnd, compsn))
239                         cout << "Successful find!" << endl;
```

```
240                     else
241                         cout << "Not find!" << endl;
242                     break;
243                 case(4):
244                     cout << "Enter citizenship:" << endl;
245                     cin >> fnd.info.citizenship;
246                     cin.ignore();
247                     if (for_list.find_elem(&fnd, compcs))
248                         cout << "Successful find!" << endl;
249                     else
250                         cout << "Not find!" << endl;
251                     break;
252                 case(5):
253                     cout << "Enter ID:" << endl;
254                     cin >> fnd.info.registr;
255                     cin.ignore();
256                     if (for_list.find_elem(&fnd, compreg))
257                         cout << "Successful find!" << endl;
258                     else
259                         cout << "Not find!" << endl;
260                     break;
261                 default:
262                     cout << "Wrong parameter" << endl;
263                     break;
264                 }
265                 break;
266 
267             case(3):
268                 cout << "Enter ID of doc:" << endl;
269                 cin >> del.info.registr;
270                 cin.ignore();
271                 if (for_list.del_elem(&del, compreg))
272                     cout << "Successful delete!" << endl;
273                 else
274                     cout << "Wrong parameter" << endl;
275                 break;
276             case(4):
277                 for_list.statistic();
278                 break;
279 
280             default:
281                 cout << "Invalid choise!" << endl;
282                 break;
283             }
284         }
285         break;
286     }
287     break;
288     default:
289         cout << "Invalid container choice" << endl;
290         exit(-2);
291     }
292 
293     for (int i = 0; i < doc; i++)
294         delete database[i];
295 
296     delete compin;
297     delete compn;
298     delete compsn;
299     delete compcs;
300     delete compreg;
```

```
301     delete spl;
302     delete lst;
303 }
```

4.2 main.cpp

```cpp
#include <iostream>
#include "user_interaction.h"

using namespace std;

int main()
{
        vector <Doc_form*> database;
        Creator* make = new Create_Tax;
        int doc;
        doc = get_doc();
        srand(time(NULL));

        for (int i = 0; i < doc; i++)
        {
                Doc_form* chance = dynamic_cast<Doc_form*>(make->Generate());
                database.insert(database.begin() + (((1 + rand()) % (database.size() +
                ↪  1))), chance);
                if (i % 2000 == 0)
                        cout << "...Waiting..." << endl;
        }

        user_interaction(database, make, doc);

        return 0;
}
```