

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: _____

Содержание

1	Описание задачи	3
2	Описание решения	4
2.1	binary_tree.h и strategy.h	4
2.2	splay_tree.h	4
2.3	stat_splay.h и stat_list.h	4
2.4	doc.h	4
2.5	list_realization.h	4
2.6	factory_method.h	5
2.7	user_interaction.h	5
2.8	main.cpp	5
3	Вывод	8
4	Приложение	9
4.1	Файлы заголовков	9
4.1.1	doc.h	9
4.1.2	strategy.h	12
4.1.3	binary_tree.h	13
4.1.4	splay_tree.h	13
4.1.5	stat_splay.h	24
4.1.6	list_realization.h	26
4.1.7	stat_list.h	29
4.1.8	factory_method.h	31
4.1.9	user_interaction.h	32
4.2	main.cpp	38

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 реализованы 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 выполнения всей работы (добавить, найти, удалить и тд).

2.3 `stat_splay.h` и `stat_list.h`

Выполняют 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.

```

Choose the amount of documents:
1) 10
2) 50000
Enter your choice:
1
...Waiting...

Select to work with splay tree or list
1) Splay tree
2) List
Enter your choice:
1

Select an operation with tax reporting
1) Add doc
2) Search doc
3) Delete doc
4) Print statistics
0) Exit
Enter your choice:
1

You can find this doc by ID: 11
Successful add!

Select an operation with tax reporting
1) Add doc
2) Search doc
3) Delete doc
4) Print statistics
0) Exit
Enter your choice:
2

Select of criteria:
1) INN
2) Name
3) Surname
4) Citizenship
5) ID
Enter your choice:
5
Enter ID:
11

Personal data
FIO: tqsmohgbmivqyilluyj uzy mddrgu
Age: 98
Male/Female: jmfika
Citizenship: nxiypiha
Nationality: jkpy
Phone: 816681

Place of residence
Index 88
Country pwdiakomxtsbuqdgqhitbsiftkojr
Region: dflxtqguirard
Adress: od, evgkk, 57
Entrance: 2
Floor: 4

```

Рис 1. Успешное добавление и поиск элемента в программе

```
Select an operation with tax reporting
1) Add doc
2) Search doc
3) Delete doc
4) Print statistics
0) Exit
Enter your choice:
3
Enter ID of doc to detete:
11
Successful delete!

Select an operation with tax reporting
1) Add doc
2) Search doc
3) Delete doc
4) Print statistics
0) Exit
Enter your choice:
2

Select of criteria:
1) INN
2) Name
3) Surname
4) Sitizenship
5) ID
Enter your choice:
5
Enter ID:
11
Not find!

Select an operation with tax reporting
1) Add doc
2) Search doc
3) Delete doc
4) Print statistics
0) Exit
Enter your choice:
```

Рис 2. Успешное удаление элемента в программе

3 Вывод

В ходе выполнения данной курсовой работы было разработано приложение для обработки данных по налоговой отчетности. Была придумана форма отчетности с достаточным объемом критериев с различными видами значений. Приложение обеспечивает хранение и обработку отчетностей, 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 поиск и удаление элементов удобнее осуществляется на основе `splay` дерева, нежели на основе списка, за счет более высокой скорости: $O(\log(n))$ против $O(n)$. Со вставкой элементов противоположная ситуация: $O(\log(n))$ у `splay` дерева против $O(1)$ у списка.

4 Приложение

4.1 Файлы заголовков

4.1.1 doc.h

```
1 #pragma once
2 #include <iostream>
3 #include <string>
4 #include <ctime>
5
6 using namespace std;
7
8 struct Document
9 {
10     string surname;
11     string name;
12     string patronymic;
13     int age;
14     string gender;
15     string citizenship;
16     string nationality;
17     string phone;
18
19     int index;
20     string country;
21     string region;
22     string city;
23     string street;
24     int apartment;
25     int entrance;
26     int floor;
27
28     int registr;
29     int inn;
30     int period;
31     int pmj;
32
33     void print()
34     {
35         cout << "\nPersonal data " << endl;
36         cout << "FIO: " << surname << " " << name << " " << patronymic << endl;
37         cout << "Age: " << age << endl;
38         cout << "Male/Female: " << gender << endl;
39         cout << "Citizenship: " << citizenship << endl;
40         cout << "Nationality: " << nationality << endl;
41         cout << "Phone: " << phone << endl;
42         cout << endl;
43
44         cout << "Place of residence " << endl;
45         cout << "Index " << index << endl;
46         cout << "Country " << country << endl;
47         cout << "Region: " << region << endl;
48         cout << "Adress: " << city << ", " << street << ", " << apartment << endl;
49         cout << "Entrance: " << entrance << endl;
50         cout << "Floor: " << floor << endl;
51         cout << endl;
52
53         cout << "Tax information " << endl;
54         cout << "ID: " << registr << endl;
```

```

55         cout << "INN: " << inn << endl;
56         cout << "Period: " << period << endl;
57         cout << "PMJ: " << pmj << endl;
58     }
59 };
60
61 class Form
62 {
63 public:
64
65     virtual ~Form() {}
66     virtual void filling() = 0;
67 };
68
69 class Doc_form : public Form
70 {
71
72 private: static int id;
73
74 public: Document info;
75
76 public:
77     Doc_form()
78     {
79         info.registr = 0;
80         info.inn = 0;
81     }
82
83     ~Doc_form()
84     {
85         info.surname = "";
86         info.name = "";
87         info.patronymic = "";
88         info.age = 0;
89         info.gender = "";
90         info.citizenship = "";
91         info.nationality = "";
92         info.phone = "";
93
94         info.index = 0;
95         info.country = "";
96         info.region = "";
97         info.city = "";
98         info.street = "";
99         info.apartment = 0;
100        info.entrance = 0;
101        info.floor = 0;
102
103        info.registr = 0;
104        info.inn = 0;
105        info.period = 0;
106        info.pmj = 0;
107    }
108
109     void filling() override
110     {
111         int tmp_int = 0;
112         string tmp_str;
113
114         tmp_str = string_generator(1 + rand() % 20);
115         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;

```

```

53     }
54     else {
55         return 0;
56     }
57 }
58 };
59
60 class Documents_registr_strategy : public Strategy<Doc_form*> {
61 public:
62     int compare(Doc_form* const& obj1, Doc_form* const& obj2) const override {
63         if (obj1->info.registr != obj2->info.registr) {
64             return obj1->info.registr < obj2->info.registr ? -1 : 1;
65         }
66         else {
67             return 0;
68         }
69     }
70 };

```

4.1.3 binary_tree.h

```

1  #pragma once
2  #include "strategy.h"
3
4  template <typename T>
5  class Binary_tree
6  {
7  protected:
8      Strategy<T>* comparator;
9
10 public:
11     Binary_tree(Strategy <T>* compare = nullptr) : comparator(compare)
12     {}
13     virtual ~Binary_tree()
14     {}
15
16     virtual bool find_node(T&& val) = 0;
17     virtual void add_node(T&& val) = 0;
18     virtual bool delete_node(T&& val) = 0;
19 };

```

4.1.4 splay_tree.h

```

1  #pragma once
2  #include "binary_tree.h"
3  #include <vector>
4  #define _CRT_SECURE_NO_WARNINGS
5
6  using namespace std;
7
8  template <typename T, typename _allocator = allocator<T>>
9  class Splay_tree : public Binary_tree<T>
10 {
11 private:
12     struct tree_node
13     {
14         T data;
15         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) :
62     ↪ Binary_tree<T>::Binary_tree(compare)
63 {
64     root = nullptr;
65     way.clear();
66 }
67
68 // деструктор
69 template <typename T, typename _allocator>
70 Splay_tree<T, _allocator>::~Splay_tree()
71 {
72     if (root != nullptr)
73         delete_tree(this->root);
74     way.clear();
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)
                way[way.size() - 1]->left = node;
246         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 &&
253             ↪ this->comparator->compare(orig, way[0]->right->data) == 0)
254         {
255             root = rotate_left(way[way.size() - 1]);
256             way.clear();
257         }
258         else if (way[0]->left != nullptr &&
259             ↪ this->comparator->compare(orig, way[0]->left->data) == 0)
260         {
261             root = rotate_right(way[way.size() - 1]);
262             way.clear();
263         }
264     }
265     else
266     {
267         if (way.size() == 2)
268         {
269             flag = 1;
270             tree_node* empty = new tree_node;
271             empty->left = way[0];
272             empty->right = nullptr;
273             way.insert(way.begin(), empty);
274         }
275         if (way[way.size() - 2]->right == way[way.size() - 1] &&
276             ↪ way[way.size() - 1]->right != nullptr &&
277             ↪ this->comparator->compare(orig, way[way.size() -
278             ↪ 1]->right->data) == 0)
279         {
280             if (way[way.size() - 3]->left != nullptr &&
281                 ↪ way[way.size() - 3]->left == way[way.size() - 2])
282                 way[way.size() - 3]->left =
283                 ↪ zigzig_left(way[way.size() - 3]);
284             else if (way[way.size() - 3]->right != nullptr &&
285                 ↪ way[way.size() - 3]->right == way[way.size() - 2])
286                 way[way.size() - 3]->right =
287                 ↪ zigzig_left(way[way.size() - 3]);
288             way.resize(way.size() - 2);
289         }
290         else if (way[way.size() - 2]->right == way[way.size() - 1] &&
291             ↪ this->comparator->compare(orig, way[way.size() -
292             ↪ 1]->left->data) == 0)
293         {
294             if (way[way.size() - 3]->left != nullptr &&
295                 ↪ way[way.size() - 3]->left == way[way.size() - 2])
296                 way[way.size() - 3]->left =
297                 ↪ zigzag_right(way[way.size() - 3]);
298             else if (way[way.size() - 3]->right != nullptr &&
299                 ↪ way[way.size() - 3]->right == way[way.size() - 2])
300                 way[way.size() - 3]->right =
301                 ↪ zigzag_right(way[way.size() - 3]);
302             way.resize(way.size() - 2);
303         }
304         else if (way[way.size() - 2]->left == way[way.size() - 1] &&
305             ↪ way[way.size() - 1]->left != nullptr &&
306             ↪ this->comparator->compare(orig, way[way.size() -
307             ↪ 1]->left->data) == 0)
308         {

```

```

291         if (way[way.size() - 3]->left != nullptr &&
292             ↪ way[way.size() - 3]->left == way[way.size() - 2])
293             way[way.size() - 3]->left =
294                 ↪ zigzig_right(way[way.size() - 3]);
295         else if (way[way.size() - 3]->right != nullptr &&
296             ↪ way[way.size() - 3]->right == way[way.size() - 2])
297             way[way.size() - 3]->right =
298                 ↪ zigzig_right(way[way.size() - 3]);
299         way.resize(way.size() - 2);
300     }
301     else if (way[way.size() - 2]->left == way[way.size() - 1] &&
302             ↪ this->comparator->compare(orig, way[way.size() -
303             ↪ 1]->right->data) == 0)
304     {
305         if (way[way.size() - 3]->left != nullptr &&
306             ↪ way[way.size() - 3]->left == way[way.size() - 2])
307             way[way.size() - 3]->left =
308                 ↪ zigzag_left(way[way.size() - 3]);
309         else if (way[way.size() - 3]->right != nullptr &&
310             ↪ way[way.size() - 3]->right == way[way.size() - 2])
311             way[way.size() - 3]->right =
312                 ↪ zigzag_left(way[way.size() - 3]);
313         way.resize(way.size() - 2);
314     }
315     if (flag == 1)
316     {
317         root = way[0]->left;
318         delete way[0];
319         way.erase(way.begin());
320         flag = 0;
321     }
322 }
323
324 if (way.empty() && len && this->comparator->compare(orig, (*node).data) == 0)
325     root = node;
326 return root;
327 }
328
329 template <typename T, typename _allocator>
330 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
331     ↪ _allocator>::rotate_right(tree_node* x)
332 {
333     tree_node* y;
334
335     y = x->left;
336     x->left = y->right;
337     y->right = x;
338     return y;
339 }
340
341 template <typename T, typename _allocator>
342 typename Splay_tree<T, _allocator>::tree_node* Splay_tree<T,
343     ↪ _allocator>::rotate_left(tree_node* y)
344 {
345     tree_node* x;
346
347     x = y->right;
348     y->right = x->left;
349     x->left = y;
350     return x;
351 }

```

```

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 !=
54         ↪ stat_int.end(); ++it)
55     {
56         cout << it->first << ": Absolute frequency: " << it->second
57         ↪ << "\tRelative frequency: " << (double)it->second /
58         ↪ (double)data_size << endl;
59     }
60     break;
61 case (2):
62
63     for (int i = 0; i < tmp_per.size(); i++)
64         stat_string[tmp_per[i]->info.name]++;
65     data_size = tmp_per.size();
66
67     for (map< string, int >::iterator it = stat_string.begin(); it !=
68         ↪ stat_string.end(); ++it)
69     {
70         cout << it->first << ": Absolute frequency: " << it->second
71         ↪ << "\tRelative frequency: " << (double)it->second /
72         ↪ (double)data_size << endl;
73     }
74     break;
75 case (3):
76
77     for (int i = 0; i < tmp_per.size(); i++)
78         stat_string[tmp_per[i]->info.surname]++;
79     data_size = tmp_per.size();
80
81     for (map< string, int >::iterator it = stat_string.begin(); it !=
82         ↪ stat_string.end(); ++it)
83     {
84         cout << it->first << ": Absolute frequency: " << it->second
85         ↪ << "\tRelative frequency: " << (double)it->second /
86         ↪ (double)data_size << endl;
87     }
88     break;
89 case (4):
90
91     for (int i = 0; i < tmp_per.size(); i++)
92         stat_string[tmp_per[i]->info.citizenship]++;
93     data_size = tmp_per.size();
94
95     for (map< string, int >::iterator it = stat_string.begin(); it !=
96         ↪ stat_string.end(); ++it)
97     {
98         cout << it->first << ": Absolute frequency: " << it->second
99         ↪ << "\tRelative frequency: " << (double)it->second /
100        ↪ (double)data_size << endl;
101     }
102     break;
103 case (5):
104
105     for (int i = 0; i < tmp_per.size(); i++)
106         stat_int[tmp_per[i]->info.registr]++;
107     data_size = tmp_per.size();

```

```

97         for (map< int, int >::iterator it = stat_int.begin(); it !=
98             ↪ stat_int.end(); ++it)
99         {
100             cout << it->first << ": Absolute frequency: " << it->second
101             ↪ << "\tRelative frequency: " << (double)it->second /
102             ↪ (double)data_size << endl;
103         }
104     }
105     break;
106 }
107 }

```

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>>
16     ↪ typename Container, typename TargetType>
17 class Interface
18 {
19 private:
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
49         ↪     _decorated_container->delete_with_comp(static_cast<TargetType&&>(data_to_del),
50         ↪     compare);
51     }
52
53     bool find_elem(TargetType&& data_to_find, Strategy<TargetType>* compare)
54     {
55         return
56         ↪     _decorated_container->find_with_comp(static_cast<TargetType&&>(data_to_find),
57         ↪     compare);
58     }
59     bool find_elem(const TargetType& data_to_find, Strategy<TargetType>* compare)
60     {
61         return _decorated_container->find_with_comp(data_to_find, compare);
62     }
63
64     void statistic()
65     {
66         _decorated_container->statistics();
67     }
68 };
69
70 template <typename T, typename _allocator = allocator<T>>
71 class List_interface
72 {
73 private:
74     list<T> data;
75 public:
76     List_interface() {}
77     ~List_interface()
78     {
79         data.clear();
80     }
81
82     void add_node_one(list<T>* tmp)
83     {
84         data = *tmp;
85     }
86
87     void add_node(const T& to_add)
88     {
89         data.push_front(to_add);
90     }
91
92     void add_node(T&& to_add)
93     {
94         data.push_front(data.begin(), static_cast<T&&>(to_add));
95     }
96
97     bool delete_with_comp(const T& to_del, Strategy<T>* comp)
98     {
99         typename list<T>::iterator it = data.begin();
100         it = find_if(data.begin(), data.end(), [comp, to_del](T cur)
101             {return (comp->compare(cur, to_del) == 0); });
102         if (it == data.end())
103         {
104             return false;
105         }
106         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

```
1  #pragma once
2  #include <iostream>
3  #include <map>
4  #include <list>
5  #include "list_realization.h"
6
7  using namespace std;
8
9  template <typename T, typename _allocator>
10 void List_interface<T, _allocator>::statistics()
11 {
12     int data_size = data.size();
13     map< int, int > stat_int;
14     map< string, int > stat_string;
15     int choice = stat_choice();
16     switch (choice)
17     {
18     case (0):
19         return;
20     case (1):
21         for (const auto& it : data)
22         {
23             stat_int[(int)it->info.inn]++;
24         }
25         for (map< int, int >::iterator it = stat_int.begin(); it !=
26             ↪ stat_int.end(); ++it)
27         {
28             cout << it->first << ": Absolute frequency: " << it->second
29             ↪ << "\tRelative frequency: " << (double)it->second /
30             ↪ (double)data_size << endl;
31         }
32         break;
33     case (2):
34         for (const auto& it : data)
35         {
36             stat_string[it->info.name]++;
37         }
38         for (map< string, int >::iterator it = stat_string.begin(); it !=
39             ↪ stat_string.end(); ++it)
40         {
41             cout << it->first << ": Absolute frequency: " << it->second
42             ↪ << "\tRelative frequency: " << (double)it->second /
43             ↪ (double)data_size << endl;
44         }
45         break;
46     case (3):
47         for (const auto& it : data)
48         {
49             stat_string[it->info.surname]++;
50         }
51         for (map< string, int >::iterator it = stat_string.begin(); it !=
52             ↪ stat_string.end(); ++it)
53         {
54             cout << it->first << ": Absolute frequency: " << it->second
55             ↪ << "\tRelative frequency: " << (double)it->second /
56             ↪ (double)data_size << endl;
57         }
58     }
```

```

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 !=
57             ↪ stat_string.end(); ++it)
58         {
59             cout << it->first << ": Absolute frequency: " << it->second
60             ↪ << "\tRelative frequency: " << (double)it->second /
61             ↪ (double)data_size << endl;
62         }
63         break;
64     case (5):
65         for (const auto& it : data)
66         {
67             stat_int[(int)it->info.inn]++;
68         }
69         for (map< int, int >::iterator it = stat_int.begin(); it !=
70             ↪ stat_int.end(); ++it)
71         {
72             cout << it->first << ": Absolute frequency: " << it->second
73             ↪ << "\tRelative frequency: " << (double)it->second /
74             ↪ (double)data_size << endl;
75         }
76         break;
77     }
78 }

```

4.1.8 factory_method.h

```
1  #pragma once
2  #include <iostream>
3  #include "doc.h"
4
5  using namespace std;
6
7  class Creator
8  {
9  public:
10     virtual ~Creator() {};
11     virtual Form* factory_method() const = 0;
12     Form* Generate() const
13     {
14         Form* product = this->factory_method();
15         product->filling();
16         return product;
17     }
18 };
19
20 class Create_Tax : public Creator
21 {
22 public:
23
24     Create_Tax() : Creator() {}
25
26 public:
27
28     Form* factory_method() const override
29     {
30         return new Doc_form();
31     }
32 };
```

4.1.9 user_interaction.h

```
1  #pragma once
2  #include <iostream>
3  #include <algorithm>
4  #include <string>
5  #include <iostream>
6  #include <list>
7  #include "stat_splay.h"
8  #include "stat_list.h"
9
10 using namespace std;
11
12 int get_doc()
13 {
14     int choice;
15
16     cout << "Choose the amount of documents:" << endl;
17     cout << "1) 100" << endl;
18     cout << "2) 50000" << endl;
19     cout << "Enter your choice:" << endl;
20     cin >> choice;
21
22     if (choice < 0 || choice > 2)
23     {
24         cout << "Wrong choice!" << endl;
25         exit(-1);
26     }
27     else if (choice == 1)
28     {
29         return 100;
30     }
31     else
32         return 50000;
33 }
34
35 void inter_choice(int choice)
36 {
37     switch (choice)
38     {
39     case(0):
40         cout << "\nSelect to work with splay tree or list" << endl;
41         cout << "1) Splay tree" << endl;
42         cout << "2) List" << endl;
43         cout << "Enter your choice:" << endl;
44         break;
45     case(1):
46         cout << "\nSelect an operation with tax reporting" << endl;
47         cout << "1) Add doc" << endl;
48         cout << "2) Search doc" << endl;
49         cout << "3) Delete doc" << endl;
50         cout << "4) Print statistics" << endl;
51         cout << "0) Exit" << endl;
52         cout << "Enter your choice:" << endl;
53         break;
54     case(2):
55         cout << "\nSelect of criteria:" << endl;
56         cout << "1) INN" << endl;
57         cout << "2) Name" << endl;
58         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
109                     ↵ << endl;
110                     cout << "Successful add!" << endl;
111                 }
112                 break;
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
206                             ↵ << endl;
207                     cout << "Successful add!" << endl;
208                     break;
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

```
1 #include <iostream>
2 #include "user_interaction.h"
3
4 using namespace std;
5
6 int main()
7 {
8     vector <Doc_form*> database;
9     Creator* make = new Create_Tax;
10    int doc;
11    doc = get_doc();
12    srand(time(NULL));
13
14    for (int i = 0; i < doc; i++)
15    {
16        Doc_form* chance = dynamic_cast<Doc_form*>(make->Generate());
17        database.insert(database.begin() + (((1 + rand()) % (database.size() +
18        ↪ 1))), chance);
19        if (i % 2000 == 0)
20            cout << "...Waiting..." << endl;
21    }
22
23    user_interaction(database, make, doc);
24
25    return 0;
26 }
```
