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: «Паттерны проектирования»

Студент	Евкарпиев Михаил Димитриевич
Группа	М80-211Б-19
Преподаватель	Романенков Александр Михайлович
Дата	26 мая 2021 г.

Оценка: _____

Подпись преподавателя: _____

Подпись студента: _____

Москва 2021

Содержание

1 Описание задачи

Разработайте приложение для проведения лотереи формата “спортлото” (5 из 36, 6 из 49 и т. п.). Ваше приложение должно обеспечивать генерацию билетов для очередного тиража лотереи (генератор должен быть реализован посредством паттерна “фабричный метод”). Количество генерируемых билетов произвольно и может быть велико ($>$ 20’000’000 шт.). Учтите ситуацию, что не все сгенерированные билеты могут участвовать в тираже (это типичная ситуация, которая возникает при неполной реализации билетов к тиражу). Смоделируйте проведение розыгрыша: на каждом ходе проверяйте, появился ли победитель; предусмотрите систему выигрышей; предоставьте возможность поиска билетов по заданным критериям: номеру билета, величине выигрыша, и т. д.. Сохраняйте информацию о проведенных тиражах для обеспечения поиска данных в будущем. Реализуйте функционал обработки данных таким образом, чтобы тип коллекции, в которой будут храниться ваши данные, являлся параметром. Продемонстрируйте обработку данных с использованием std::vector и собственной реализации двусвязного списка.

2 Решение задачи

Для решения поставленной задачи, были описаны 4 заголовочных файлов.

2.1 Lottery_tickets.h

В данном файле описан класс, который содержит один конструктор с параметром, деструктор, методы для работы с private полями, вывод информации о билете в поток. В билете содержится следующая информация: набор чисел, количество чисел в наборе, номер билета и сумма выигрыша.

```
class Lottery_ticket
{
private:
    unsigned int                size;
    std::vector<unsigned int>    numbers;
    unsigned int                id;
    unsigned int                gain = 0;

public:
    Lottery_ticket(unsigned int&& n = 0)
    {
        size = n;
    }

    ~Lottery_ticket()
    {}

    unsigned int    get_gain() const;
    unsigned int    get_id() const;
    unsigned int    get_size() const;
    void            set_size(const unsigned int& value);
    void            set_gain(const unsigned int& value);
    void            set_id(const unsigned int& value);
    void            set_numbers(const unsigned int& value);
    std::ostream&    print_ticket(std::ostream& stream);
    std::vector<unsigned int> get_numbers() const;
};
```

2.2 Decor.h

Для реализация функционала обработки данных таким образом, чтобы тип коллекции, в которой будут храниться данные, являлся параметром, воспользуемся паттерном программирования “Декоратор”. Декоратор — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные “обёртки”. Класс Interface определяет интерфейс обёртки для всех конкретных декораторов. Конкретные декораторы вызывают обёрнутый объект и изменяют его результат некоторым образом. В данном файле описан класс Interface, который является декоратором для конкретных реализаций базы данных на основе std::vector и двусвязного списка. В каждой базе данных были реализованы функции поиска победителей, билета с указанным номером и билетов с указанным выигрышем.

2.3 Generator.h

В данном файле были реализованы генераторы билетов посредством паттерна “абстрактная фабрика": Абстрактная фабрика — порождающий шаблон проектирования, предоставляет интер-

фейс для создания объектов, шаблон реализуется путём создания абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы . Затем пишутся классы, реализующие этот интерфейс. В данном случае абстрактной фабрикой является абстрактный класс : Generator с виртульными методом Generation и Generation_win, а классы, реализующие интерфейс: Generator4_20, Generator7_49, Generator6_45 и Generator5_36. Для всех лотерей генерация билетов происходит следующим образом: с начала генерируется победный билет с помощью Generation_win и заносится в базу данных под нулевым номером. После чего этот билет передается в качестве параметра в Generation. Далее в Generation новый билет заполняется случайными цифрами, после чего выполняется проверка на уникальность полученных чисел. В случае если числа не уникальны, происходит повторная генерация. Далее билету присваивается номер и устанавливается кол-во чисел в билете, так же подсчтывается сумма выигрыша в соответсвии с количеством "угаданных"чисел и заносится в билет.

2.4 dc_list.h

В этом файле был реализован двусвязный список с итераторами. Шаблонный класс элемента двусвязного списка включает в себя следующие поля: данные типа T, указатель на предыдущий и следующий элемент, указатель на первый элемент двусвязного списка, указатель на последний элемент, и счётчик длины двусвязного списка. Для него реализованы следующие методы: взять длину списка, проверить список на пустоту, вставить элемент в начало или конец, удалить элемент из начала или конца, найти элемент по значению, очистить список, вернуть значения первого и последнего элемента, а также вернуть итератор на первый и последний элемент списка. Для перемещения по двусвязному списку был реализован вложенный класс итератора с операциями: разыменовывания("*"), логическими (“==”) и (“! =”), присваивания("="), а также инкремента("++") и декремента("−−").

2.5 main.cpp

В этом файле описано взаимодействие с пользователем. Сперва пользователь в функции main выбирает коллекцию, в которой будут храниться лотерейные билеты, затем у пользователя в цикле запрашивается тип лоттереи и количество билетов учавствующих в тираже, после чего вызывается генерация билетов. Далее вызывается метод find_winners, который ищет и выводит на экран всех победителей в текущем тираже. Далее информация о билетах сохраняется в файл "save_tickets". И пользователю предлагается выбор действий: осуществить поиск по номеру или выигрышу билета, либо вернуться к меню.

2.6 Примеры вывода

На скриншоте №1 представлен вывод для выбора лотереии типа 4 из 20, генерации 100 билетов, поиск по номеру 55.

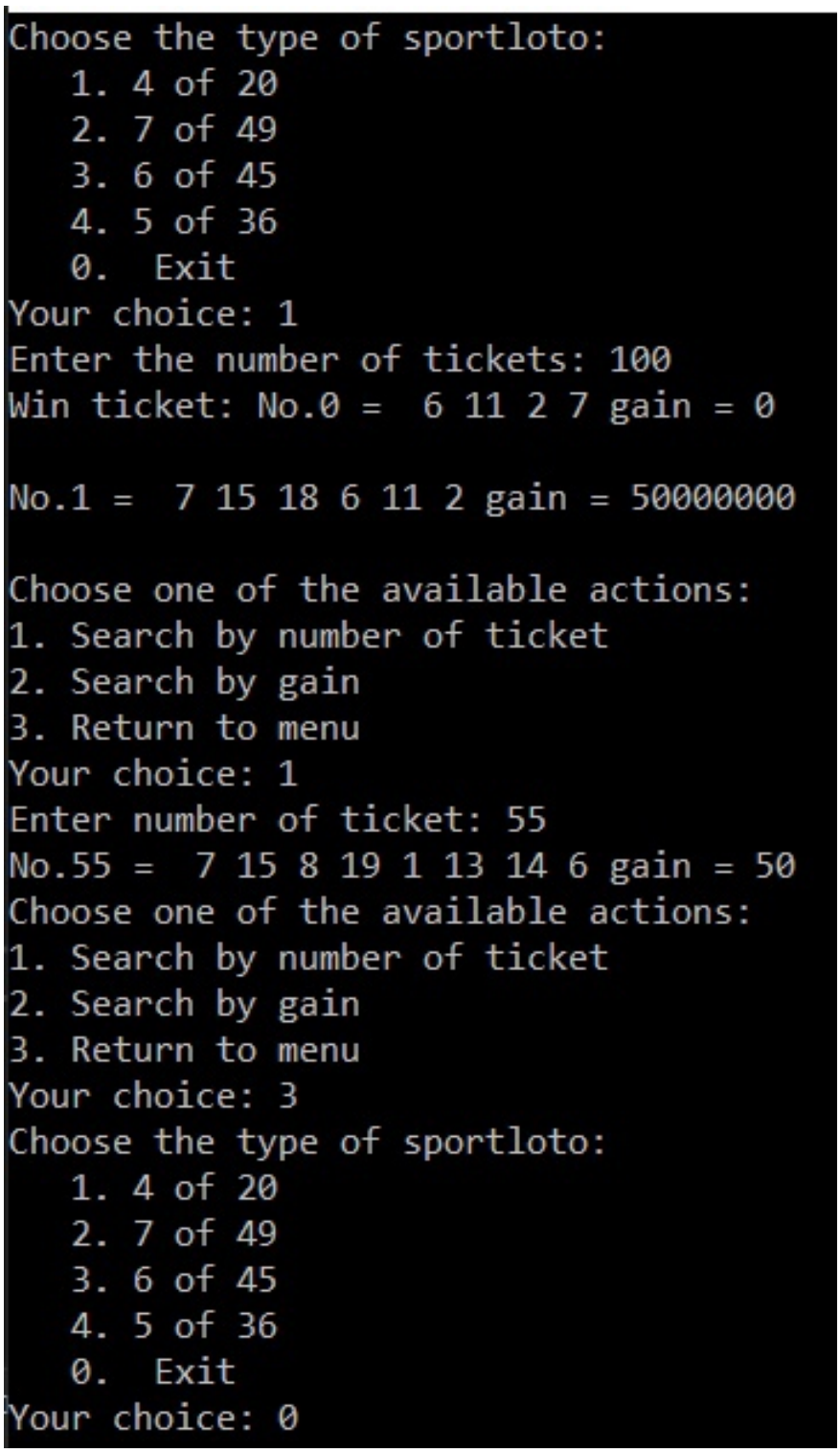

Рис. 1: скриншот №1

На скриншоте №2 представлен вывод для выбора лотереии типа 5 из 36, генерации 100 билетов, поиск по выигрышу в размере 500.

```
Choose the type of sportloto:
   1. 4 of 20
   2. 7 of 49
   3. 6 of 45
   4. 5 of 36
   0.  Exit
Your choice: 4
Enter the number of tickets: 100
Win ticket: No.0 =  16 7 10 20 8 gain = 0

No winners
Choose one of the available actions:
1. Search by number of ticket
2. Search by gain
3. Return to menu
Your choice: 2
Enter gain: 500
No.15 =  27 20 16 4 12 2 15 8 19 9 gain = 500
No.16 =  35 30 10 20 15 1 32 5 16 12 26 gain = 500
No.21 =  4 36 5 13 11 7 22 31 10 16 6 gain = 500
No.63 =  7 21 35 6 8 24 5 16 15 gain = 500
No.72 =  10 7 28 17 3 32 1 14 20 5 34 gain = 500
Choose one of the available actions:
1. Search by number of ticket
2. Search by gain
3. Return to menu
Your choice: 3
Choose the type of sportloto:
   1. 4 of 20
   2. 7 of 49
   3. 6 of 45
   4. 5 of 36
   0.  Exit
Your choice: 0
Goodbye
```

Рис. 2: скриншот №2

3 Вывод

Таким образом было реализовано приложение, имитирующее проведение лоттереи Спортлото с использованием паттерна проектирования фабрик, а также сыметирована генерация тиража и проведение самой лоттереи, с ежеходной проверкой на появление победителя лоттереи и занесением информации о разыгранных тиражах в файл. Для хранения данных в приложении были использованы две коллекции: std::vector из стандартной библиотеки шаблонов и собственная реализация двусвязного списка.

Для двусвязного списка был сделан следующий вывод: двусвязный список подходит для решения поставленной задачи, однако в рамках этой задачи он избыточен, нет необходимости в доступе к предыдущему элементу. Для хранения данных лучше использовать односвязный список. К недостаткам двусвязного списка стоит отнести операцию поиска элемента, сложность которой составляет O(n). Сложность вставки в конец и удаления из конца списка составляет O(1). Такой результат достигается благодаря указателю на конец списка, который позволяет избежать прохождения по всему списку в поиске последнего элемента.

Почти все что было сказано про двусвязный список применимо и к вектору.

Сложность алгоритма проведения розыгрыша составляет O(n*m). Сложность алгоритма поиска победителя составляет O(m), т. к. после проведения розыгрыша для поиска победителя необходимо проверить m лотерейных билетов, которые участвуют в игре.

Обе реализации были протестированы на генерацию 100000 билетов для лотереии формата 4 из 20. По результатам тестирования время затраченное на генерацию у коллекций несущественно отличается.

4 Приложение

4.1 Заголовочные файлы

4.1.1 Decor.h

```
#pragma once
#include "dc_list.h"
#include <iostream>
#include <memory>
#include <vector>
using namespace std;

#define max_gain 50000000

template <template <typename T, class allocator = std::allocator<T>>
class Container, typename T>
    class Interface
{
private:
    Container<T>* decor_container;
public:
    Interface(Container<T>* container)
    {
        decor_container = container;
    }
    ~Interface() {}

    unsigned int    size() const
    {
        return (this->decor_container->size());
    }

    bool    empty() const
    {
        return (this->decor_container->empty());
    }

    void    insert_back(const T& value)
    {
        this->decor_container->insert_back(value);
    }

    void    remove_back()
    {
        this->decor_container->remove_back();
    }

    void    clear_memory()
    {
        this->decor_container->clear_memory();
    }

    void    clear()
    {
        this->decor_container->clear();
    }

    void    print(ostream& stream) const
    {
        this->decor_container->print(stream);
    }

    void    find_winners() const
    {
```

```
60          return (this->decor_container->find_winners());
61      }
62
63      void    search_by_id(const unsigned int& id) const
64      {
65          this->decor_container->search_by_id(id);
66      }
67
68      void    search_by_gain(const unsigned int& g) const
69      {
70          this->decor_container->search_by_gain(g);
71      }
72  };
73
74  template <typename T, class allocator = std::allocator<T>>
75  class    Vector_interface
76  {
77  private:
78      vector<T>    vec;
79  public:
80
81
82      unsigned int    size() const
83      {
84          return vec.size();
85      }
86
87      bool    empty() const
88      {
89          return vec.empty();
90      }
91
92      void    insert_back(const T& value)
93      {
94          vec.push_back(value);
95      }
96
97      void    remove_back()
98      {
99          vec.pop_back();
100     }
101
102     void    clear_memory()
103     {
104         typename vector<T>::iterator    iter;
105
106         iter = vec.begin();
107         while (iter != vec.end())
108         {
109             delete(*iter);
110             ++iter;
111         }
112     }
113
114     void    clear()
115     {
116         if (vec.empty())
117             return;
118         this->vec.clear();
119     }
120
121     void    print(ostream& stream) const
122     {
123         typename vector<T>::const_iterator    iter;
124
125         iter = vec.begin();
```

```
126         while (iter != vec.end())
127         {
128             (*iter)->print_ticket(stream);
129             ++iter;
130         }
131     }
132
133     void    search_by_id(const unsigned int& id) const
134     {
135         typename vector<T>::const_iterator    iter;
136         bool    flag = false;
137
138         iter = vec.begin();
139         ++iter;
140         while (iter != vec.end())
141         {
142             if ((*iter)->get_id() == id)
143             {
144                 (*iter)->print_ticket(std::cout);
145                 flag = true;
146                 break;
147             }
148             ++iter;
149         }
150         if (!flag)
151             cout << "\nNo tickets found for the specified \"No.\"\n";
152     }
153
154     void    search_by_gain(const unsigned int& g) const
155     {
156         typename vector<T>::const_iterator    iter;
157         bool        flag = false;
158
159         iter = vec.begin();
160         ++iter;
161         while (iter != vec.end())
162         {
163             if ((*iter)->get_gain() == g)
164             {
165                 (*iter)->print_ticket(std::cout);
166                 flag = true;
167             }
168             ++iter;
169         }
170         if (!flag)
171             cout << "\nNo tickets found for the specified prize\n";
172     }
173
174     void    find_winners() const
175     {
176         typename vector<T>::const_iterator    iter;
177         bool    flag = false;
178
179         iter = vec.begin();
180         ++iter;
181         while (iter != vec.end())
182         {
183             if ((*iter)->get_gain() == max_gain)
184             {
185                 cout << "\n";
186                 (*iter)->print_ticket(std::cout);
187                 cout << "\n";
188                 flag = true;
189             }
190             ++iter;
191         }
```

```
        if (!flag)
            cout << "\nNo winners\n";
    }
};


template <typename T, class allocator = std::allocator<T>>
class    List_interface
{
private:
    List<T>    lst;

public:

    unsigned int    size() const
    {
        return lst.size();
    }

    bool    empty() const
    {
        return (lst.empty());
    }

    void    insert_back(const T& value)
    {
        lst.push_back(value);
    }

    void    remove_back()
    {
        lst.pop_back();
    }

    void    clear_memory()
    {
        typename List<T>::iterator    iter;

        if (lst.empty())
            return;
        iter = lst.begin();
        while (iter != lst.end())
        {
            delete(*iter);
            ++iter;
        }
    }

    void    clear()
    {
        if (lst.empty())
            return;
        this->lst.clear();
    }

    void    print(ostream& stream) const
    {
        typename List<T>::iterator    iter;

        iter = lst.begin();
        while (iter != lst.end())
        {
            (*iter)->print_ticket(stream);
            ++iter;
        }
    }
```

```
258
259     void    search_by_id(const unsigned int& id) const
260     {
261         typename List<T>::iterator    iter;
262         bool    flag = false;
263
264         iter = lst.begin();
265         ++iter;
266         while (iter != lst.end())
267         {
268             if ((*iter)->get_id() == id)
269             {
270                 (*iter)->print_ticket(std::cout);
271                 flag = true;
272                 break;
273             }
274             ++iter;
275         }
276         if (!flag)
277             cout << "\nNo tickets found for the specified \"No.\"\n";
278     }
279
280     void    search_by_gain(const unsigned int& g) const
281     {
282         typename List<T>::iterator    iter;
283         bool    flag = false;
284
285         iter = lst.begin();
286         ++iter;
287         while (iter != lst.end())
288         {
289             if ((*iter)->get_gain() == g)
290             {
291                 (*iter)->print_ticket(std::cout);
292                 flag = true;
293             }
294             ++iter;
295         }
296         if (!flag)
297             cout << "\nNo tickets found for the specified gain\n";
298     }
299
300     void    find_winners() const
301     {
302         typename List<T>::iterator    iter;
303         bool    flag = false;
304
305         iter = lst.begin();
306         ++iter;
307         while (iter != lst.end())
308         {
309             if ((*iter)->get_gain() == max_gain)
310             {
311                 cout << "\n";
312                 (*iter)->print_ticket(std::cout);
313                 cout << "\n";
314                 flag = true;
315             }
316             ++iter;
317         }
318         if (!flag)
319             cout << "\nNo winners\n";
320     }
321 };
```

4.1.2 Lottery_ticket

```cpp
#pragma once

#include <iostream>
#include <algorithm>
#include <vector>
#include <fstream>

class Lottery_ticket
{
private:
    unsigned int                size;
    std::vector<unsigned int>    numbers;
    unsigned int                id;
    unsigned int                gain = 0;

public:
    Lottery_ticket(unsigned int&& n = 0)
    {
        size = n;
    }

    ~Lottery_ticket()
    {}

    unsigned int    get_gain() const;
    unsigned int    get_id() const;
    unsigned int    get_size() const;
    void            set_size(const unsigned int& value);
    void            set_gain(const unsigned int& value);
    void            set_id(const unsigned int& value);
    void            set_numbers(const unsigned int& value);
    std::ostream&    print_ticket(std::ostream& stream);
    std::vector<unsigned int> get_numbers() const;
};


unsigned int Lottery_ticket::get_gain() const
{
    return gain;
}

unsigned int Lottery_ticket::get_id() const
{
    return id;
}

unsigned int Lottery_ticket::get_size() const
{
    return size;
}

std::vector<unsigned int> Lottery_ticket::get_numbers() const
{
    return numbers;
}

void Lottery_ticket::set_size(const unsigned int& value)
{
    this->size = value;
}


void Lottery_ticket::set_gain(const unsigned int& value)
{
```

```
    this->gain = value;
}

void Lottery_ticket::set_id(const unsigned int& value)
{
    this->id = value;
}

void Lottery_ticket::set_numbers(const unsigned int& value)
{
    if (this->numbers.size() == this->size)
        return;
    this->numbers.push_back(value);
}

std::ostream& Lottery_ticket::print_ticket(std::ostream& stream)
{
    stream << "No." << id << " = ";
    for (int i = 0; i < size; i++) {
        stream << " " << numbers[i];
    }
    stream << " gain = " << gain << std::endl;
    return stream;
}
```

4.1.3 Generator.h

```
#pragma once

#include <iostream>
#include <string>
#include <vector>
#include <ctime>
#include <random>
#include <set>
#include <unordered_set>
#include "Lottery_ticket.h"
using namespace std;

#define min_gain 50
#define max_gain 50000000

class Generator {
public:
    virtual ~Generator() {}
    virtual Lottery_ticket* Generation(int i, Lottery_ticket* lot_win) const = 0;
    virtual Lottery_ticket* Generation_win() const = 0;
};

class Generator4_20 : public Generator {  // 4-20
public:
    Lottery_ticket* Generation_win() const override
    {
        Lottery_ticket* ticket = new Lottery_ticket(4);
        vector<int> vec;
        set<int> st;
        unordered_set<int> ust;
        int num;


        static mt19937 gen(time(NULL));
        uniform_int_distribution<int> uid1(1, 20);

        while (true)
        {
            for (int i = 0; i < 4; i++) {
```

```
                num = uid1(gen);
                st.insert(num);
                vec.push_back(num);
            }
            if (st.size() == vec.size()) {
                break;
            }
            else
            {
                st.clear();
                vec.clear();
            }
        }

        for (const auto& it : vec) {
            ticket->set_numbers(it);
        }

        ticket->set_id(0);
        return ticket;
    }

    Lottery_ticket* Generation(int i, Lottery_ticket* lot_win) const override
    {
        Lottery_ticket* ticket = new Lottery_ticket ;
        vector<int> vec;
        vector<unsigned int> tmp;
        set<int> st;
        unordered_set<int> ust;
        int num;
        int counter;
        size_t size_counter = 4;

        static mt19937 gen(time(NULL));
        uniform_int_distribution<int> uid1(1, 20);
        uniform_int_distribution<int> number1(4, 8);

        tmp = lot_win->get_numbers();
        for (const auto& it : tmp)
            ust.insert(it);

        while (true)
        {
            counter = number1(gen);
            for (int i = 0; i < counter; i++) {
                num = uid1(gen);
                st.insert(num);
                vec.push_back(num);
            }
            if (st.size() == vec.size()) {
                break;
            }
            else
            {
                st.clear();
                vec.clear();
            }
        }

        for (const auto& elem : vec) {
            if (ust.count(elem)) {
                size_counter--;
            }
        }

        ticket->set_size(counter);
```

```
106
107         if (size_counter > 0 && size_counter < ust.size() - 1) {
108             ticket->set_gain(min_gain * pow(10, ust.size() - size_counter - 1) / 10);
109         }
110         else if (size_counter == 0) {
111             ticket->set_gain(max_gain);
112         }
113
114         for (const auto& it : vec) {
115             ticket->set_numbers(it);
116         }
117
118         ticket->set_id(i);
119         return ticket;
120     }
121 };
122
123 class Generator7_49 : public Generator { // 7-49
124 public:
125     Lottery_ticket* Generation_win() const override
126     {
127         Lottery_ticket* ticket = new Lottery_ticket(7);
128         vector<int> vec;
129         set<int> st;
130         unordered_set<int> ust;
131         int num;
132
133
134         static mt19937 gen(time(NULL));
135         uniform_int_distribution<int> uid2(1, 49);
136
137         while (true)
138         {
139             for (int i = 0; i < 7; i++) {
140                 num = uid2(gen);
141                 st.insert(num);
142                 vec.push_back(num);
143             }
144             if (st.size() == vec.size()) {
145                 break;
146             }
147             else
148             {
149                 st.clear();
150                 vec.clear();
151             }
152         }
153
154         for (const auto& it : vec) {
155             ticket->set_numbers(it);
156         }
157
158         ticket->set_id(0);
159         return ticket;
160     }
161
162     Lottery_ticket* Generation(int i, Lottery_ticket* lot_win) const override
163     {
164         Lottery_ticket* ticket = new Lottery_ticket;
165         vector<int> vec;
166         vector<unsigned int> tmp;
167         set<int> st;
168         unordered_set<int> ust;
169         int num;
170         int counter;
171         size_t size_counter = 7;
```

```
172
173         static mt19937 gen(time(NULL));
174         uniform_int_distribution<int> uid2(1, 49);
175         uniform_int_distribution<int> number2(7, 14);
176
177         tmp = lot_win->get_numbers();
178         for (const auto& it : tmp)
179             ust.insert(it);
180
181         while (true)
182         {
183             counter = number2(gen);
184             for (int i = 0; i < counter; i++) {
185                 num = uid2(gen);
186                 st.insert(num); // log(n)
187                 vec.push_back(num);
188             }
189             if (st.size() == vec.size()) {
190                 break;
191             }
192             else
193             {
194                 st.clear();
195                 vec.clear();
196             }
197         }
198
199         for (const auto& elem : vec) {
200             if (ust.count(elem)) {
201                 size_counter--;
202             }
203         }
204
205         ticket->set_size(counter);
206
207         if (size_counter > 0 && size_counter < ust.size() - 1) {
208             ticket->set_gain(min_gain * pow(10, ust.size() - size_counter - 1) / 10);
209         }
210         else if (size_counter == 0) {
211             ticket->set_gain(max_gain);
212         }
213
214         for (const auto& it : vec) {
215             ticket->set_numbers(it);
216         }
217
218         ticket->set_id(i);
219         return ticket;
220     }
221 };
222
223 class Generator6_45 : public Generator { // 6-45
224 public:
225     Lottery_ticket* Generation_win() const override
226     {
227         Lottery_ticket* ticket = new Lottery_ticket(6);
228         vector<int> vec;
229         set<int> st;
230         unordered_set<int> ust;
231         int num;
232
233
234         static mt19937 gen(time(NULL));
235         uniform_int_distribution<int> uid3(1, 45);
236
237         while (true)
```

```
238         {
239             for (int i = 0; i < 6; i++) {
240                 num = uid3(gen);
241                 st.insert(num);
242                 vec.push_back(num);
243             }
244             if (st.size() == vec.size()) {
245                 break;
246             }
247             else
248             {
249                 st.clear();
250                 vec.clear();
251             }
252         }
253 
254         for (const auto& it : vec) {
255             ticket->set_numbers(it);
256         }
257 
258         ticket->set_id(0);
259         return ticket;
260     }
261     Lottery_ticket* Generation(int i, Lottery_ticket* lot_win) const override
262     {
263         Lottery_ticket* ticket = new Lottery_ticket;
264         vector<int> vec;
265         vector<unsigned int> tmp;
266         set<int> st;
267         unordered_set<int> ust;
268         int num;
269         int counter;
270         size_t size_counter = 6;
271 
272         static mt19937 gen(time(NULL));
273         uniform_int_distribution<int> uid3(1, 45);
274         uniform_int_distribution<int> number3(6, 13);
275 
276         tmp = lot_win->get_numbers();
277         for (const auto& it : tmp)
278             ust.insert(it);
279 
280         while (true)
281         {
282             counter = number3(gen);
283             for (int i = 0; i < counter; i++) {
284                 num = uid3(gen);
285                 st.insert(num); // log(n)
286                 vec.push_back(num);
287             }
288             if (st.size() == vec.size()) {
289                 break;
290             }
291             else
292             {
293                 st.clear();
294                 vec.clear();
295             }
296         }
297 
298         for (const auto& elem : vec) {
299             if (ust.count(elem)) {
300                 size_counter--;
301             }
302         }
303 
```

```
304         ticket->set_size(counter);
305 
306         if (size_counter > 0 && size_counter < ust.size() - 1) {
307             ticket->set_gain(min_gain * pow(10, ust.size() - size_counter - 1) / 10);
308         }
309         else if (size_counter == 0) {
310             ticket->set_gain(max_gain);
311         }
312 
313         for (const auto& it : vec) {
314             ticket->set_numbers(it);
315         }
316 
317         ticket->set_id(i);
318         return ticket;
319     }
320 };
321 
322 class Generator5_36 : public Generator { // 5-36
323 public:
324     Lottery_ticket* Generation_win() const override
325     {
326         Lottery_ticket* ticket = new Lottery_ticket(5);
327         vector<int> vec;
328         set<int> st;
329         unordered_set<int> ust;
330         int num;
331 
332 
333         static mt19937 gen(time(NULL));
334         uniform_int_distribution<int> uid4(1, 36);
335 
336         while (true)
337         {
338             for (int i = 0; i < 5; i++) {
339                 num = uid4(gen);
340                 st.insert(num);
341                 vec.push_back(num);
342             }
343             if (st.size() == vec.size()) {
344                 break;
345             }
346             else
347             {
348                 st.clear();
349                 vec.clear();
350             }
351         }
352 
353         for (const auto& it : vec) {
354             ticket->set_numbers(it);
355         }
356 
357         ticket->set_id(0);
358         return ticket;
359     }
360 
361     Lottery_ticket* Generation(int i, Lottery_ticket* lot_win) const override
362     {
363         Lottery_ticket* ticket = new Lottery_ticket;
364         vector<int> vec;
365         vector<unsigned int> tmp;
366         set<int> st;
367         unordered_set<int> ust;
368         int num;
369         int counter;
```

```
        size_t size_counter = 5;

        static mt19937 gen(time(NULL));
        uniform_int_distribution<int> uid4(1, 36);
        uniform_int_distribution<int> number4(5, 11);

        tmp = lot_win->get_numbers();
        for (const auto& it : tmp)
            ust.insert(it);

        while (true)
        {
            counter = number4(gen);
            for (int i = 0; i < counter; i++) {
                num = uid4(gen);
                st.insert(num); // log(n)
                vec.push_back(num);
            }
            if (st.size() == vec.size()) {
                break;
            }
            else
            {
                st.clear();
                vec.clear();
            }
        }

        for (const auto& elem : vec) {
            if (ust.count(elem)) {
                size_counter--;
            }
        }

        ticket->set_size(counter);

        if (size_counter > 0 && size_counter < ust.size() - 1) {
            ticket->set_gain(min_gain * pow(10, ust.size() - size_counter - 1) / 10);
        }
        else if (size_counter == 0) {
            ticket->set_gain(max_gain);
        }

        for (const auto& it : vec) {
            ticket->set_numbers(it);
        }

        ticket->set_id(i);
        return ticket;
    }
};

class Creator {
public:
    virtual ~Creator() {};
    virtual Generator* FactoryMethod() const = 0;
};


class ConcreteCreator1 : public Creator {
public:
    Generator* FactoryMethod() const override {
        return new Generator4_20();
    }
};

```

```
class ConcreteCreator2 : public Creator {
public:
    Generator* FactoryMethod() const override {
        return new Generator7_49();
    }
};

class ConcreteCreator3 : public Creator {
public:
    Generator* FactoryMethod() const override {
        return new Generator6_45();
    }
};

class ConcreteCreator4 : public Creator {
public:
    Generator* FactoryMethod() const override {
        return new Generator5_36();
    }
};
```

4.1.4 dc_list.h

```
#pragma once

#include <iostream>
#include <list>

template <typename T>
class    List
{
    struct Node
    {
        T        ticket;
        Node*    prev;
        Node*    next;

        Node(T value)
        {
            this->ticket = value;
            this->prev = nullptr;
            this->next = nullptr;
        }

        ~Node() {}
    };

private:
    Node*        Head;
    Node*        Tail;
    unsigned int    lst_size;

    Node* create_new_node(const T& value);

public:
    class    iterator
    {
    private:
        Node* node_ptr;

    public:
        iterator()
        {
            node_ptr = nullptr;
        }
        iterator(Node* ptr)
```

```
44          {
45              node_ptr = ptr;
46          }
47          ~iterator() {}
48
49          bool operator==(const iterator& rhs) const
50          {
51              return (this->node_ptr == rhs.node_ptr);
52          }
53          bool operator!=(const iterator& rhs) const
54          {
55              return (this->node_ptr != rhs.node_ptr);
56          }
57          T& operator*()
58          {
59              return (node_ptr->ticket);
60          }
61          iterator& operator++()
62          {
63              this->node_ptr = this->node_ptr->next;
64              return (*this);
65          }
66          iterator& operator--()
67          {
68              this->node_ptr = this->node_ptr->prev;
69              return (*this);
70          }
71          iterator operator++(int)
72          {
73              iterator copy = *this;
74              this->node_ptr = this->node_ptr->next;
75              return copy;
76          }
77          iterator operator--(int)
78          {
79              iterator copy = *this;
80              this->node_ptr = this->node_ptr->prev;
81              return copy;
82          }
83          iterator& operator=(const iterator& rhs)
84          {
85              this->node_ptr = rhs.node_ptr;
86              return (*this);
87          }
88      };
89
90      List()
91      {
92          Head = nullptr;
93          Tail = nullptr;
94          lst_size = 0;
95      }
96      ~List()
97      {
98          clear();
99      }
100
101     unsigned int    size() const;
102     bool            empty() const;
103     T&                front();
104     T&                back();
105     iterator        end() const;
106     iterator        begin() const;
107     iterator        find(const T& value);
108     void            clear();
109     void            push_back(const T& value);
```

```
110     void            push_front(const T& value);
111     void            pop_back();
112     void            pop_front();
113 };
114 
115 template <typename T>
116 typename List<T>::iterator    List<T>::begin() const
117 {
118     return iterator(Head);
119 }
120 
121 template <typename T>
122 typename List<T>::iterator    List<T>::end() const
123 {
124     return iterator(Tail->next);
125 }
126 
127 template <typename T>
128 T& List<T>::front()
129 {
130     return (Head->ticket);
131 }
132 
133 template <typename T>
134 T& List<T>::back()
135 {
136     return (Tail->ticket);
137 }
138 
139 template <typename T>
140 typename List<T>::Node* List<T>::create_new_node(const T& value)
141 {
142     Node* new_node = nullptr;
143 
144     new_node = new Node(value);
145     if (!new_node)
146         return nullptr;
147     return new_node;
148 }
149 
150 template <typename T>
151 unsigned int List<T>::size() const
152 {
153     return lst_size;
154 }
155 
156 template <typename T>
157 bool    List<T>::empty() const
158 {
159     return (lst_size == 0);
160 }
161 
162 template <typename T>
163 void    List<T>::clear()
164 {
165     Node* temp_ptr;
166 
167         while (Head != nullptr)
168         {
169             temp_ptr = Head;
170             Head = Head->next;
171             delete temp_ptr;
172         }
173         if(Head != nullptr)
174         Head = nullptr;
175         Tail = nullptr;
```

```
176         lst_size = 0;
177 }
178
179 template <typename T>
180 void    List<T>::push_front(const T& value)
181 {
182     Node* new_node;
183
184     new_node = create_new_node(value);
185     if (new_node != nullptr)
186     {
187         if (!Head)
188             Tail = new_node;
189         else
190         {
191             new_node->next = Head;
192             Head->prev = new_node;
193         }
194         Head = new_node;
195         lst_size++;
196     }
197 }
198
199 template <typename T>
200 void    List<T>::push_back(const T& value)
201 {
202     Node* new_node = nullptr;
203
204     new_node = create_new_node(value);
205     if (new_node != nullptr)
206     {
207         if (!Head)
208             Head = new_node;
209         else
210         {
211             new_node->prev = Tail;
212             Tail->next = new_node;
213         }
214         Tail = new_node;
215         lst_size++;
216     }
217 }
218
219 template <typename T>
220 void    List<T>::pop_back()
221 {
222     Node* prev_ptr;
223     Node* temp_ptr;
224
225     if (!Head)
226         return;
227     else if (Head && !Head->next)
228     {
229         lst_size--;
230         delete Head;
231         Head = nullptr;
232         Tail = nullptr;
233         return;
234     }
235     prev_ptr = Tail->prev;
236     prev_ptr->next = nullptr;
237     temp_ptr = Tail;
238     Tail = prev_ptr;
239     lst_size--;
240     delete temp_ptr;
241 }
```

```cpp

template <typename T>
void    List<T>::pop_front()
{
    Node* next_ptr;
    Node* tmp_ptr;

    if (Head && !Head->next)
    {
        Head = nullptr;
        Tail = nullptr;
        lst_size--;
        delete Head;
        return;
    }

    next_ptr = Head->next;
    next_ptr->prev = nullptr;
    tmp_ptr = Head;
    Head = next_ptr;
    lst_size--;
    delete tmp_ptr;
}

template <typename T>
typename List<T>::iterator    List<T>::find(const T& value)
{
    List<T>::iterator    iter;

    iter = begin();
    while (iter != end())
    {
        if (*iter == value)
            return (iter);
        iter++;
    }
    return end();
}
```

4.2 main.cpp

```cpp
#include <iostream>
#include <string>
#include <ctime>
#include <vector>
#include <random>
#include <fstream>
#include "Generator.h"
#include "dc_list.h"
#include "Lottery_ticket.h"
#include "Decor.h"
using namespace std;

// виды спортлото: 4-20, 7-49, 6-45, 5-36.

int main() {
    clock_t         start;
    clock_t         end;
    double          seconds;
    int             choice = -1;
    int             choice2;
    int             number_of_tickets;
    unsigned int    gn;
    unsigned int    number_of_ticket;
    Lottery_ticket* lot_win;
```

```
    Creator*        creator4_20 = new ConcreteCreator1();
    Creator*        creator7_49 = new ConcreteCreator2();
    Creator*        creator6_45 = new ConcreteCreator3();
    Creator*        creator5_36 = new ConcreteCreator4();
    Generator*      generator4_20 = creator4_20->FactoryMethod();
    Generator*      generator7_49 = creator7_49->FactoryMethod();
    Generator*      generator6_45 = creator6_45->FactoryMethod();
    Generator*      generator5_36 = creator5_36->FactoryMethod();

    /*List_interface<Lottery_ticket*> lst_tickets;
    Interface<List_interface, Lottery_ticket*> tickets(&lst_tickets);*/
    Vector_interface<Lottery_ticket*> vec_tickets;
    Interface<Vector_interface, Lottery_ticket*> tickets(&vec_tickets);


    ofstream save_tickets("save_tickets.txt", ios::trunc);
    if (save_tickets.fail()) {
        cerr << "Can't open file \"save_tickets\"\n";
        exit(-1);
    }

    mt19937 gen(time(NULL));


    while (choice != 0)
    {
        cout << "Choose the type of sportloto:\n";
        cout << "   1. 4 of 20\n";
        cout << "   2. 7 of 49\n";
        cout << "   3. 6 of 45\n";
        cout << "   4. 5 of 36\n";
        cout << "   0.  Exit\n";
        cout << "Your choice: ";
        cin >> choice;

        if (choice == 0) break;

        cout << "Enter the number of tickets: ";
        cin >> number_of_tickets;

        uniform_int_distribution<int> num(number_of_tickets / 2, number_of_tickets);
        number_of_tickets = num(gen);
        number_of_tickets += 1;

        switch (choice)
        {
        case 1:
            start = clock();
            lot_win = generator4_20->Generation_win();
            tickets.insert_back(lot_win);
            for (int i = 1; i < number_of_tickets; i++) {
                tickets.insert_back(generator4_20->Generation(i, lot_win));
            }
            end = clock();
            seconds = (double)(end - start) / 1000.0;
            std::cout << "generated" << std::endl;
            std::cout << "adding time(in seconds) - " << seconds / 1000.0 << std::endl;
            cout << seconds;
            break;
        case 2:
            lot_win = generator7_49->Generation_win();
            tickets.insert_back(lot_win);
            for (int i = 1; i < number_of_tickets; i++) {
                tickets.insert_back(generator7_49->Generation(i, lot_win));
            }
            break;
```

```
        case 3:

            lot_win = generator6_45->Generation_win();
            tickets.insert_back(lot_win);
            for (int i = 1; i < number_of_tickets; i++) {
                tickets.insert_back(generator6_45->Generation(i, lot_win));
            }
            break;
        case 4:

            lot_win = generator5_36->Generation_win();
            tickets.insert_back(lot_win);
            for (int i = 1; i < number_of_tickets; i++) {

                tickets.insert_back(generator5_36->Generation(i, lot_win));
            }
            break;
        default:
            cout << "Wrong choice!" << endl << "Try again\n";
            continue;
        }

        tickets.find_winners();

        save_tickets << "Win ticket: ";
        tickets.print(save_tickets);
        save_tickets << endl;
        choice2 = -1;

        while (choice2 != 3)
        {
            cout << "Choose one of the available actions:\n";
            cout << "1. Search by number of ticket\n";
            cout << "2. Search by gain\n";
            cout << "3. Return to menu\n";
            cout << "Your choice: ";
            cin >> choice2;

            switch (choice2)
            {
            case 1:
                cout << "Enter number of ticket: ";
                cin >> number_of_ticket;
                tickets.search_by_id(number_of_ticket);
                break;
            case 2:
                cout << "Enter gain: ";
                cin >> gn;
                tickets.search_by_gain(gn);
                break;
            default:
                tickets.clear();
                break;
            }
        }
        tickets.clear();
        tickets.clear_memory();
    }
    cout << "Goodbye";
    delete creator4_20;
    delete creator5_36;
    delete creator6_45;
    delete creator7_49;
    return 0;
}
```