

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	Солдатов Вячеслав Алексеевич
Группа	М80-211Б-19
Преподаватель	Романенков Александр Михайлович
Дата	

Оценка: _____

Подпись преподавателя: _____

Подпись студента: _____

Содержание

1	Описание задания	3
1.1	Вариант №32	3
2	Описание решения	4
2.1	Описание алгоритма	4
2.2	Описание реализованных сущностей	5
2.2.1	main.cpp	5
2.2.2	client_code.h	5
2.2.3	functions.h	5
2.2.4	components.h	6
2.2.5	generators.h	7
2.2.6	err_check.h	7
2.2.7	list_decorator.h	7
2.2.8	Структура билета	7
2.2.9	Демонстрация работы, входные и выходные данные	8
3	Вывод	16
4	Литература	17
5	Приложение	18
5.0.1	main.cpp	18
5.0.2	client_code.h	18
5.0.3	functions.h	20
5.0.4	components.h	20
5.0.5	generators.h	29
5.0.6	error_check.h	31
5.0.7	list_decorator.h	32

1 Описание задания

1.1 Вариант №32

Разработайте приложение для проведения лотереи (например, аналога Русского лото). 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::forward_list` и собственной реализации односвязного списка.

2 Описание решения

2.1 Описание алгоритма

Для решения поставленной задачи программа выполняет следующие действия

1. Открытие файла на запись с проверкой. Если проверка не пройдена, программа завершает свою работу с кодом ошибки, переданным в функцию `errCheck_Main()`, которая также выводит информацию для пользователя в соответствии с кодом ошибки.
2. Передача управления функции `clientCode()`
3. Создание генераторов для генерации объектов типа: БИЛЕТ, ТИРАЖ, ИГРА.
4. Создание объекта класса ИГРА, ввод пользователем следующих параметров: id игры, количество тиражей, количество билетов в каждом тираже, шанс продажи билета.
5. Создание объектов класса ТИРАЖ внутри конструктора класса ИГРА с помощью генератора, помещение объектов класса ТИРАЖ в односвязный список указателей типа ТИРАЖ - приватное поле класса ИГРА.
6. Создание объектов класса БИЛЕТ, внутри конструктора класса ТИРАЖ с помощью генератора, помещение объектов класса БИЛЕТ в односвязный список указателей типа БИЛЕТ - приватное поле класса ТИРАЖ.
7. Возвращение управления функции `clientCode()`, начало взаимодействия с пользователем в формате "ввод операции - вывод данных или особого ответа."
8. При вводе пользователем операции обработки игры происходит обработка тиража, номер которого указал пользователь, или всех тиражей с помощью функции `processGame()`.
9. Передача управления функции `processLot()`.
10. В функции `processLot()` происходит "пошаговая" обработка билетов по принципу: вынимается бочонок с числом - обрабатывается весь список билетов. Если число на боченке совпадает с числом в одном или нескольких из полей билета, число в билете "зачеркивается".
11. Выбор победителей посредством проведения 3-х туров. Билет выигравший в определенном туре помечается особым образом и попадает в список указателей `<имя тура>_tour_winner_tickets` типа БИЛЕТ - приватное поле класса ИГРА.
12. Возвращение управления функции `processGame()`, внутри которой сначала происходит обработка ошибок с помощью функций заголовочного файла `error_check.h`, которые могли возникнуть при обработке тиражей в функции `processLot`.
13. В функции `processGame()` происходит вывод всех списков с названиями `<имя тура>_tour_winner_tickets` в стандартный поток вывода, таким образом показывая пользователю победившие билеты.
14. Возвращение управления функции `clientCode()` вместе с кодом ошибки, обработка ошибок с помощью функций заголовочного файла `error_check.h`, блокирование доступа к определенным

операциям в зависимости от кода ошибки.

15. Следующая итерация взаимодействия с пользователем, в зависимости от введенной пользователем операции производится вывод данных об прошедшей игре в файл, поиск билета по файлу.
16. Завершение работы с пользователем, очистка памяти.
17. Возвращение управления функции `main()` вместе с кодом ошибки, обработка ошибок с помощью функций заголовочного файла `error_check.h`, завершение работы программы с помощью функции `errCheck_Main()`.

2.2 Описание реализованных сущностей

Для решения поставленной задачи были реализованы следующие сущности:

2.2.1 `main.cpp`

Функция `main()`

Открывает файл `output_file.txt` на запись

Задаёт seed для случайной генерации чисел в объектах класса ПОЛЕ - частных объектах класса БИЛЕТ

Передаёт управление функции `clientCode()`

Обрабатывает код ошибки, полученный в качестве возвращаемого значения от функции `clientCode()`, передавая управление функции `errCheck_Main()`.

2.2.2 `client_code.h`

Функция `clientCode()`

Создаёт указатели на объекты классов `SuperGenerator` и `Game`

Осуществляет взаимодействие с пользователем с помощью классической конструкции `while(true)` и `switch(operation)`. Доступные для пользователя операции: `p` - основная операция, выявление победителей среди купивших билет, обработка тиражей; `o` - вывод данных всех по проведенной игре в файл `output_file.txt`; `s` - поиск определенного билета в файле `output_file.txt`; `c` - выход из цикла, завершение взаимодействия с пользователем.

Очистка памяти.

2.2.3 `functions.h`

Функция `floatChanceToInt(float)`

Приведение пользовательского значения типа `float` к значению типа `int`. Используется для получения шанса продажи билетов для тиража.

Функция `foundDuplicate(int8_t*, int, int)`

Поиск дубликатов в массиве, замена дубликата другим значением

Функция `getValues(int8_t*)`

Заполнение массива размером 2x5 случайными значениями, используется функция `foundDuplicate()`.

2.2.4 components.h

Все классы, кроме класса `KegBag`, описанные в этом заголовочном файле, имеют производные классы с приставкой `RusLot` в названии.

Класс `KegBag`

Задача этого класса - симуляция мешка с бочонками. Количество бочонков задается макро-ком `NUMBER_OF_KEGS` и специально сделано неизменяемым с пользовательской стороны. Класс имеет метод для вытаскивания одного бочонка или нескольких бочонков `getKegs(vector<unsigned int>, unsigned int)`, а также оператор вывода в поток.

Абстрактный класс `Field`

Этот класс содержит указатель на указатель типа `unsigned int8_t`, количество строк и столбцов поля билета, имеет перегруженный деструктор для очистки памяти, операторы выгрузки и вставки в поток. Класс `Ticket` является дружественным к этому классу.

Шаблонный абстрактный класс `Ticket`

Этот класс содержит шаблонный вектор полей производного класса от класса `Ticket`, а также поля атомарных типов для вывода информации о билете. Имеет метод `getStatus()` нужный для получения информации о статусе билета (продан, куплен, победил в n-ом туре и т.д.), `processTicket()`, который "зачеркивает" совпавшие числа в полях билета, имеет операторы вставки и выгрузки из потока. Абстрактный класс `Lot` и шаблонная функция `Ticket<> *searchTickets(Game<, >, Ticket<>*, int)` являются дружественными к этому классу.

Шаблонный абстрактный класс `Lot`

Этот класс хранит список указателей на объекты класса `Ticket`, объект класса `KegBag`, оператор выгрузки из потока. Абстрактный шаблонный класс `Game` является дружественным к этому классу.

Шаблонный абстрактный класс `Game`

Этот класс хранит список указателей на объекты класса `Lot`, а также три списка указателей на объекты класса `Ticket`. Класс `Game` в самом верху иерархии лотереи, имея список класса `Lot` (тираж). Конструктор класса, используя методы класса, создает лотерию, иерархично создавая объекты классов типа `Lot` и `Ticket`. `Game` имеет методы вывода в стандартный поток списков билетов-победителей `printWinners(ListBasicInterface<, Ticket<>*>)`, задания пользователем параметров генерации тиражей и билетов `generationMenu()`, генерации тиражей `generationProceed(float, GeneratorLot<T, L>, GeneratorTicket<T>)`, обработки тиража `processLot(long int, auto lot)` и всей

игры processGame(int, unsigned int, int), проведения розыгрыша processTour(unsigned int, auto lot, ListBasicInterface< , Ticket<>*>, int, vector<unsigned int>), вставки в поток.

2.2.5 generators.h

В этом файле хранятся абстрактные шаблонные и производные генераторы для всех классов, кроме Field и KegBag. При генерации билетов используется паттерн "Фабричный метод". Также здесь определен абстрактный класс SuperGenerator, хранящий указатели на абстрактные шаблонные генераторы.

2.2.6 err_check.h

В этом файле хранятся функции int errCheck_Components(int, unsigned int), int errCheck_ClientCode(int, long int), int errCheck_Main(int, long int) для обработки ошибок на разных уровнях выполнения программы. Реализация одной из функций:

```
1 int errCheck_Components(unsigned int additional_information)
2 {
3     switch(err_code)
4     {
5         case 1:
6             cerr << NON_CRITICAL"Unable to find Lot with id " <<
              ↳ additional_information << " in Game!\nProcessing of this lot will not
              ↳ begin.\n\n";
7             break;
8         case 2:
9             cerr << NON_CRITICAL"Lot with id " << additional_information << " has been
              ↳ already processed.\nReprocessing is prohibited\n\n";
10            break;
11     }
12 }
```

2.2.7 list_decorator.h

В этом файле хранится реализация собственного односвязного списка в виде класса List со стандартными для односвязного списка методами и общий для List и forward_list шаблонный интерфейс в виде класса ListBasicInterface, хранящий контейнер односвязного списка.

2.2.8 Структура билета

Билет содержит поля:

1. Номер тиража
2. Номер билета
3. Статус билета
4. Вектор, содержащий игровые поля.

2.2.9 Демонстрация работы, входные и выходные данные

```
Available generation parameters:
Game id;
Number of lots;
Number of tickets in lot. Set this parameter to 0 if number of tickets in each lot should vary;
Ticket sale chance (float). Set this parameter to number less than 0 if sale chance for tickets in each lot should vary.
..... Set this parameter to number greater than 1 to get random chance of sale for tickets in each lot.

Enter parameters:
0 3 20000 0.8

Tickets were successfully generated.
```

Рис. 1: Пользователь вводит собственные параметры для генерации тиражей.

```
Available operations:
'p' - enter process mode and id of lot to process. In zero mode id is useless;
'o' - print winners;
'f' - write processed data to output_file;
's' - search specific tickets in output_file;
'c' - finish work and exit program.

Enter operation:
p 1 0

Processing lot with id 0
Processing first tour...
Processing second tour...
Processing third tour...
Remaining kegs in bag: 87 18 55 72 41 84 23 81 89 60 73 22 1 79 57 75 31 65 6 49 9 86 56 83 38 44 27 8 76

Enter operation:
p 1 1

Processing lot with id 1
Processing first tour...
Processing second tour...
Processing third tour...
Remaining kegs in bag: 86 67 77 56 79 82 63 43 62 75 20 44 25 23 41 28 5 35 1 26 30 88 15 37 42 24 57 54 76

Enter operation:
o

First tour winners:
Lot series: 1
Ticket: 3385
```

Рис. 2: Пользователь вводит операцию обработки первого и второго тиража билетов, и операцию вывода победителей в стандартный поток. Программа обрабатывает тиражи без ошибок.


```

First tour winners:
Lot series: 1
Ticket: 3385
Field 0
9 16 -29 0 44 0 0 0 82
0 10 0 35 40 53 0 -73 0
0 0 28 0 41 55 0 71 88
Field 1
0 -12 -29 -31 0 0 -69 -73 0
0 18 29 0 40 0 0 71 80
10 18 0 -33 0 0 0 79 86
1-st TOUR WINNER, SOLD

Lot series: 1
Ticket: 6695
Field 0
-3 0 21 -33 0 0 0 77 87
5 0 0 30 0 58 -69 0 80
1 -13 21 0 0 54 0 76 0
Field 1
0 -12 -29 -38 0 -50 0 -73 0
0 0 -27 -38 45 0 63 70 0
0 0 24 34 0 57 0 72 85
1-st TOUR WINNER, SOLD

Lot series: 1
Ticket: 10038
Field 0
9 0 25 0 47 -50 60 0 0
0 0 28 32 40 57 0 0 84
8 0 0 39 44 0 63 0 86
Field 1
8 11 25 -38 43 0 0 0 0

```

Рис. 3: Программа выводит в стандартный поток вывода билеты, победившие в первом туре.

```
Ticket: 10038
Field 0
9 0 25 0 47 -50 60 0 0
0 0 28 32 40 57 0 0 84
8 0 0 39 44 0 63 0 86
Field 1
8 11 25 -38 43 0 0 0 0
0 16 0 0 41 0 65 70 83
-3 -12 -29 0 0 -50 0 -73 0
1-st TOUR WINNER, SOLD

Lot series: 1
Ticket: 1303
Field 0
0 18 29 0 0 53 67 -73 0
-3 -13 0 -33 0 -50 0 -73 0
0 0 25 35 48 0 63 78 0
Field 1
7 18 0 0 46 0 67 0 85
7 17 24 32 0 0 0 0 86
0 14 25 -38 43 0 0 0 87
1-st TOUR WINNER, SOLD

Lot series: 1
Ticket: 16381
Field 0
0 0 23 0 0 55 68 74 85
0 11 23 32 0 52 0 0 86
2 0 0 0 47 51 64 0 89
Field 1
0 -13 0 -38 0 -50 -69 -73 0
4 19 0 -38 0 0 0 -73 83
9 16 21 36 0 57 0 0 0
```

Рис. 4: Программа выводит в стандартный поток вывода билеты, победившие в первом туре.

```

Second tour winners:
Lot series: 1
Ticket: 5342
Field 0
0 -13 0 -31 -48 -50 -64 0 0
0 0 -29 -33 0 -58 -65 0 -87
0 0 -27 -33 -48 0 0 -74 -89
Field 1
0 0 23 0 42 59 63 75 0
-9 0 23 -33 0 -53 0 71 0
-3 0 -22 0 0 56 63 0 86
2-nd TOUR WINNER, SOLD

Lot series: 1
Ticket: 8580
Field 0
0 18 25 0 44 0 67 -70 0
0 0 28 -36 41 0 -69 0 86
0 10 23 35 44 57 0 0 0
Field 1
-8 0 -27 -36 0 0 -64 0 -80
-9 -12 0 -33 0 0 -64 0 -87
-2 0 0 -33 0 -58 -68 0 -80
2-nd TOUR WINNER, SOLD

Lot series: 0
Ticket: 12809
Field 0
0 -10 -25 -35 0 55 0 0 -80
-10 -13 28 -39 0 0 0 0 86
6 -10 0 0 0 0 -63 -77 89
Field 1
-7 -13 -25 -30 0 -59 0 0 0

```

Рис. 5: Программа выводит в стандартный поток вывода билеты, победившие во втором туре.

```

Lot series: 0
Ticket: 12809
Field 0
0 -10 -25 -35 0 55 0 0 -80
-10 -13 28 -39 0 0 0 0 86
6 -10 0 0 0 0 -63 -77 89
Field 1
-7 -13 -25 -30 0 -59 0 0 0
-10 -12 0 0 -42 0 0 -78 -85
0 -17 0 0 0 -52 -63 -70 -80
2-nd TOUR WINNER, SOLD

Third tour winners:
No winners!

Enter operation:
p 1 2

Processing lot with id 2
Processing first tour...
Processing second tour...
Processing third tour...
Remaining kegs in bag: 14 5 49 86 29 64 52 62 40 88 10 8 47 66 58 68 45 50 20 70 35 1 39 37 46 23 77 33 16

Enter operation:
f
Done.

```

Рис. 6: Программа закончила вывод победивших во втором туре билетов и сообщила, что победителей в третьем туре нет. Пользователь вводит операцию обработки третьего, последнего тиража и операцию вывода данных в файл.

```
Enter operation:  
  
c  
  
Process finished with exit code 0
```

Рис. 8: Пользователь завершает свою работу с программой, которая отработала без ошибок.

```
Enter operation:  
s  
  
Available searching parameters:  
Lot number;  
Ticket number;  
Ticket status.  
If you enter a negative number for criteria, this criteria will not be taken into consideration when searching. In that  
case first matching ticket will be returned.  
1 1917 -1  
  
Lot series: 1  
Ticket: 1917  
Field 0  
-4 12 -22 -33 45 0 0 0 0  
0 -11 -22 31 0 0 0 76 85  
0 0 -21 0 -49 55 0 -74 -80  
Field 1  
3 -17 0 0 -42 59 -63 0 0  
3 -10 0 0 -48 -56 -67 0 0  
-10 18 0 0 -46 50 0 -78 0  
SOLD
```

Рис. 7: Пользователь вводит операцию поиска билета и программа находит билет согласно введенным параметрам.

```

Game ID: 0

Lot series: 2
Ticket: 19999
Field 0
0 0 0 0 -44 58 66 -73 -84
-2 10 0 35 0 -53 0 0 -85
0 0 -24 37 -42 -54 0 77 0
Field 1
0 10 0 0 0 -59 66 -72 -84
10 0 0 -31 45 0 -63 0 -84
0 10 -21 -38 0 52 -61 0 0
SOLD

Lot series: 2
Ticket: 19998
Field 0
0 0 0 -34 46 58 -60 0 -83
1 -12 0 33 0 0 62 0 -87
0 0 -27 0 47 -56 0 -71 -80
Field 1
-3 10 -26 -31 0 0 0 0 -80
0 0 23 0 0 -57 64 -78 -85
10 0 -24 0 -44 -59 0 -74 0
SOLD

Lot series: 2
Ticket: 19997
Field 0
0 -13 0 0 46 0 -67 -79 -87
10 0 0 37 0 50 0 -78 -82
-2 0 0 -38 0 58 -61 -73 0
Field 1
10 -15 0 0 0 -56 0 -79 -85
0 0 -26 0 40 0 -65 -78 -83
-7 14 29 0 -41 0 0 0 88
SOLD

```

Рис. 9: Часть содержания файла output_file.txt

```
Enter operation:  
p 0 0  
  
NON CRITICAL: Lot with id 2 has been already processed.  
Reprocessing is prohibited  
  
NON CRITICAL: Lot with id 1 has been already processed.  
Reprocessing is prohibited  
  
NON CRITICAL: Lot with id 0 has been already processed.  
Reprocessing is prohibited
```

Рис. 10: Пример появления ошибок в процессе выполнения программы.

3 Вывод

Было разработано приложение для проведения лотереи и обработки соответствующих данных с использованием паттерна проектирования "Фабричный метод". Также в этом приложении был реализован интерфейс для односвязных списков `forwardList`, `forwardList(> 20000000)`. — $O(n)$.

4 Литература

Список литературы

- [1] Donald Knuth. *Knuth: Computers and Typesetting*. URL: <http://www-cs-faculty.stanford.edu/~uno/abcde.html>.
- [2] Donald E. Knuth. “Fundamental Algorithms”. В: Addison-Wesley, 1973. гл. 1.2.
- [3] Donald E. Knuth. *The Art of Computer Programming*. Four volumes. Seven volumes planned. Addison-Wesley, 1968.

5 Приложение

5.0.1 main.cpp

```
1 #include <iostream>
2 #include <fstream>
3 #include <ctime>
4 #include "client_code.h"
5
6 using namespace std;
7
8 int main()
9 {
10     ofstream ofp("output_file.txt");
11     if(!ofp.is_open())
12     {
13         err_code=1;
14         errCheck_Main();
15     }
16
17     srand(static_cast <unsigned>(time(nullptr)));
18
19     if(clientCode(ofp))
20         err_code=2;
21     ofp.close();
22
23     errCheck_Main();
24 }
```

5.0.2 client_code.h

```
1 int clientCode(ofstream &fp)
2 {
3     SuperGenerator<FieldRusLot, LIST_IMPLEMENTATION> *super_gen = new
4     ↪ SuperGeneratorRusLot<LIST_IMPLEMENTATION>();
5     Game<FieldRusLot, LIST_IMPLEMENTATION>
6     ↪ *game=super_gen->gen_game->getGame(*super_gen->gen_lot, *super_gen->gen_tic);
7     ↪ // Пример ввода: 0, 2, 2000000, 0.8
8
9     cout << "Tickets were successfully generated.\n\n"
10     ↪ "Available operations:\n"
11     ↪ "'p' - enter process mode and id of lot to process. In zero mode id is
12     ↪ ↪ useless;\n"
13     ↪ "'o' - print winners;\n"
14     ↪ "'f' - write processed data to output_file;\n"
15     ↪ "'s' - search specific tickets in output_file;\n"
16     ↪ "'c' - finish work and exit program.";
17     char operation;
18     int process_mode, number;
19     bool proceed=true, p_not_blocked=true, s_not_blocked=false, o_f_not_blocked=false;
20     while(proceed)
21     {
22         cout << "\n\nEnter operation:\n";
23         ↪ // Пример
24         ↪ ввода: p 0 0
25         cin >> operation;
26         switch(operation)
27         {
28             case 'p':
```

```

23         if(p_not_blocked)
24         {
25             cin >> process_mode >> number;
26             while((err_code = game->processGame(process_mode, number)) == -1)
27                 cin >> process_mode;
28             if(err_code)
29             {
30                 err_code=1;
31                 p_not_blocked=false;
32                 break;
33             }
34             o_f_not_blocked=true;
35         }
36         else
37             cout << BAD_ACCESS"Last calculation ended with errors.\n";
38         break;
39
40     case 'o':
41         if(o_f_not_blocked)
42             game->printAllWinners();
43         else
44             cout << BAD_ACCESS"Game has not been processed or processing has
45             ↪ ended with errors.\n";
46         break;
47
48     case 'f':
49         if(o_f_not_blocked)
50         {
51             fp << *game;
52             s_not_blocked=true;
53             cout << "Processed data has been written to file.";
54         }
55         else
56             cout << BAD_ACCESS"Game has not been processed or processing has
57             ↪ ended with errors.\n";
58         break;
59
60     case 's':
61         if(s_not_blocked)
62         {
63             if(!searchTickets(*game, super_gen->gen_tic->getTicket(0, 0, -1)))
64                 ↪ // Пример ввода: 0 432 -1
65                 err_code=1;
66         }
67         else
68             cout << BAD_ACCESS"Processed data has not been written to
69             ↪ file.\n";
70         break;
71
72     case 'c':
73         proceed=false;
74         break;
75
76     default:
77         cout << "No such operation, try again.\n";
78     }
79 }
80
81 delete super_gen; delete game;
82
83 return errCheck_ClientCode();
84 }

```

5.0.3 functions.h

```
1 int floatChanceToInt(float chance)
2 {
3     chance*=100;
4     if(chance>100)
5         chance=rand() %100 +1;
6     return (int)chance;
7 }
8
9 int foundDuplicate(int8_t *arr, int i, int number)
10 {
11     i++;
12     for(int j=0; j<i; j++)
13         if(number==arr[j])
14             return 1;
15     arr[i-1]=number;
16     return 0;
17 }
18
19 void getValues(int8_t arr[2][5])
20 {
21     int i, min;
22     for(i=0; i<5; i++)
23         while(foundDuplicate(arr[0], i, rand() % 9)) {}
24     for(i=0; i<5; i++)
25     {
26         if((min=(arr[0][i]+1)*10-10)==0)
27             min=1;
28         while(foundDuplicate(arr[1], i, rand() % 10+min)) {}
29     }
30 }
```

5.0.4 components.h

```
1 #include <algorithm>
2 #include <vector>
3 #include <cstdlib>
4 #include <random>
5 #include <chrono>
6 #include "list_decorator.h"
7 #include "functions.h"
8
9 #define NUMBER_OF_KEGS 90
10
11 using namespace std;
12
13 template<class T>
14 class GeneratorTicket;
15 template <class T, template<class> class L>
16 class GeneratorLot;
17
18 class KegBag
19 {
20 private:
21     vector<unsigned int> bag;
```

```

22 public:
23     KegBag(unsigned &seed)
24     {
25         seed+=rand(); // rand() прибавлен из-за того, что
                ↪ seed практически не меняется за время, достаточное для генерации
                ↪ нескольких билетов.
26         for(int i=1; i<NUMBER_OF_KEGS; i++)
27             bag.push_back(i);
28         shuffle(bag.begin(), bag.end(), default_random_engine(seed));
29     }
30     void getKegs(vector<unsigned int> &vec, unsigned int amount)
31     {
32         vec.clear();
33         for(int i=0; i<amount; i++)
34         {
35             vec.push_back(bag.back());
36             bag.pop_back();
37         }
38     }
39
40     friend const ostream& operator<<(ostream &out, const KegBag &keg_bag)
41     {
42         out << "Remaining kegs in bag:";
43         for(auto it : keg_bag.bag)
44             out << ' ' << it;
45         return out;
46     }
47 };
48
49 class Field
50 {
51 protected:
52     unsigned long long row_count;
53     unsigned long long col_count;
54     int8_t **field;
55 public:
56     Field(unsigned long long rows, unsigned long long cols) : row_count(rows),
                ↪ col_count(cols)
57     {
58         unsigned long long j;
59         int8_t field_values_in_row[2][5];
60
61         field=new int8_t *[row_count];
62         for(unsigned long long i=0; i<row_count; i++)
63         {
64             field[i]=new int8_t[col_count];
65             for(j=0; j<col_count; j++)
66                 field[i][j] = 0;
67
68             getValues(field_values_in_row);
69             for (j = 0; j < 5; j++)
70                 field[i][field_values_in_row[0][j]] = field_values_in_row[1][j];
71         }
72     }
73     ~Field()
74     {
75         for(int i=0; i<row_count; i++)
76             delete[]field[i];
77         delete[]field;
78     }
79     friend const ostream& operator<< (ostream &out, const Field &fld);

```

```

80     friend istream& operator>> (istream &in, Field &fld);
81     template<class> friend class Ticket;
82 };
83
84 class FieldRusLot : public Field
85 {
86 public:
87     FieldRusLot() : Field(3, 9) {}
88 };
89
90 const ostream& operator<< (ostream &out, const Field &fld)
91 {
92     for(int i=0; i<fld.row_count; i++)
93     {
94         for(int j=0; j<fld.col_count; j++)
95             out << (int)fld.field[i][j] << ' ';
96         out << '\n';
97     }
98     return out;
99 }
100
101 istream& operator>> (istream &in, Field &fld)
102 {
103     int tmp;
104     for(int i=0; i<fld.row_count; i++)
105     {
106         for(int j=0; j<fld.col_count; j++)
107         {
108             in >> tmp;
109             fld.field[i][j]=tmp;
110         }
111         in.get(); in.get();
112     }
113 }
114
115 template <class T, template<class> class L> class Game;
116
117 template<class T>
118 class Ticket
119 {
120 protected:
121     unsigned long long lot_number, ticket_number, status;
122     vector<T> fields;
123 public:
124     Ticket(unsigned long long lot_num, unsigned long long ticket_num, unsigned int
        ↪ field_count, unsigned int status) :
125         lot_number(lot_num), ticket_number(ticket_num),
126         fields(field_count), status(status) {}
127     ~Ticket()
128     {
129         fields.clear();
130     }
131     int getStatus()
132     {
133         return status;
134     }
135     int processTicket(vector<unsigned int> &numbers, int vic_type)
136     {
137         int i, j;
138         int counters[3] = {0};
139         // Проверяют
140         ↪ строку, поле и билет на зачеркивание всех чисел в строке/поле/всех полях.

```

```

139     for(auto &fld : fields)
140     {
141         counters[1]=0;
142         for(i=0; i<fld.row_count; i++)
143         {
144             counters[2]=0;
145             for(j=0; j<fld.col_count; j++)
146             {
147                 for(auto &num : numbers)
148                 {
149                     if(fld.field[i][j] < 0)
150                         counters[2]++;
151                     else if (num == fld.field[i][j])
152                     {
153                         fld.field[i][j]*=-1;
154                         counters[2]++;
155                         break;
156                     }
157                 }
158             }
159             if(counters[2]==5)
160             {
161                 if(vic_type==1)
162                 {
163                     status = 2;
164                     return 1;
165                 }
166                 counters[1]++;
167             }
168         }
169         if(counters[1]==3)
170         {
171             counters[0]++;
172             if(vic_type==2)
173             {
174                 status = 3;
175                 return 1;
176             }
177         }
178     }
179     if(counters[0]==2 && vic_type==3)
180     {
181         status = 4;
182         return 1;
183     }
184     return 0;
185 }
186 friend ostream& operator<< (ostream &out, const Ticket<T> &tic)
187 {
188     out << "Lot series: " << tic.lot_number << '\n';
189     out << "Ticket: " << tic.ticket_number << '\n';
190
191     unsigned int field_count=tic.fields.size();
192     for(int i=0; i<field_count; i++)
193         out << "Field " << i << '\n' << tic.fields[i];
194
195     switch(tic.status)
196     {
197         case 0:
198             out << "UNSOLD";
199             break;

```

```

200         case 1:
201             out << "SOLD";
202             break;
203         case 2:
204             out << "1-st TOUR WINNER, SOLD";
205             break;
206         case 3:
207             out << "2-nd TOUR WINNER, SOLD";
208             break;
209         case 4:
210             out << "3-rd TOUR WINNER, SOLD";
211             break;
212     }
213     out << '\n';
214     return out;
215 }
216 friend istream& operator>> (istream &in, Ticket<T> &tic)
217 {
218     unsigned int size=tic.fields.size();
219     char c;
220
221     in.ignore(12);
222     in >> tic.lot_number;
223     in.ignore(8, ' ');
224     in >> tic.ticket_number;
225     in.get(c);
226     for(int i=0; i<size; i++)
227     {
228         in.ignore(20, '\n');
229         in>>tic.fields[i];
230     }
231     in.get(c);
232     switch(c)
233     {
234         case 'U':
235             tic.status = 0;
236             break;
237         case 'S':
238             tic.status = 1;
239             break;
240         case '1':
241             tic.status = 2;
242             break;
243         case '2':
244             tic.status = 3;
245             break;
246         case '3':
247             tic.status = 4;
248             break;
249     }
250     in.ignore(30, '\n');
251 }
252 template <class C, template<class> class L> friend Ticket<C>
253     ⇨ *searchTickets(Game<C, L> &game, Ticket<C> *tic);
254 template <class, template<class> class> friend class Lot;
255 };
256 class TicketRusLot : public Ticket<FieldRusLot>
257 {
258 public:

```

// Пропуск


```

259     TicketRusLot(unsigned long long lot_num, unsigned long long ticket_num, unsigned
    ↪ int status) : Ticket(lot_num, ticket_num, 2, status) {}
260 };
261
262 template <class T, template<class> class L>
263 class Lot
264 {
265 private:
266     bool was_processed;
267     unsigned long long lot_number;
268     KegBag keg_bag;
269     ListBasicInterface<L, Ticket<T>*> lot_tickets;
270 public:
271     Lot(unsigned long long num, unsigned long long num_of_tickets, unsigned int
    ↪ sale_chance, unsigned &seed, GeneratorTicket<T> &gen) : was_processed(false),
    ↪ lot_number(num),
272
273     {
274         for(unsigned long long i=0; i<num_of_tickets; i++)
275             lot_tickets.push_front(gen.getTicket(lot_number,
    ↪ i,(rand()%100)<sale_chance));
276     }
277     ~Lot()
278     {
279         lot_tickets.clear();
280     }
281     template <class, template<class> class> friend class Game;
282     friend const ostream& operator<< (ostream &out, Lot<T, L> &lot)
283     {
284         for(auto it = lot.lot_tickets.begin(); it!=lot.lot_tickets.end(); it++)
285             out << *it << '\n';
286         return out;
287     }
288 };
289
290
291 template <class T, template<class> class L>
292 class Game
293 {
294 private:
295     unsigned seed;
296     unsigned long long id;
297     unsigned long long count_of_lots;
298     unsigned long long count_of_tickets;
299     ListBasicInterface<L, Lot<T, L>*> lot_tickets;
300     ListBasicInterface<L, Ticket<T>*> first_tour_winner_tickets;
301     ListBasicInterface<L, Ticket<T>*> second_tour_winner_tickets;
302     ListBasicInterface<L, Ticket<T>*> third_tour_winner_tickets;
303
304     Game(GeneratorLot<T, L> &gen_lot, GeneratorTicket<T> &gen_tic) :
    ↪ seed(chrono::system_clock::now().time_since_epoch().count())
305     {
306         generationProceed(generationMenu(), gen_lot, gen_tic);    // Вызов меню для
    ↪ задания параметров генерации в качестве параметра для основной функции
    ↪ генерации
307     }
308
309     float generationMenu()
310     {
311         float sale_chance;

```

```

312     cout << "Available generation parameters:\n"
313           "Game id;\n"
314           "Number of lots;\n"
315           "Number of tickets in lot. Set this parameter to 0 if number of
    ↳ tickets in each lot should vary;\n"
316           "Ticket sale chance (float). Set this parameter to number less than 0
    ↳ if sale chance for tickets in each lot should vary.\t\t "
317           "Set this parameter to number greater than
    ↳ 1 to get random chance of sale for
    ↳ tickets in each lot.\n\n"

318     "Enter parameters:\n";
319     cin >> id >> count_of_lots >> count_of_tickets >> sale_chance;
320     return sale_chance;
321 }
322 void generationProceed(float sale_chance, GeneratorLot<T, L> &gen_lot,
    ↳ GeneratorTicket<T> &gen_tic)
323 {
324     if(!count_of_tickets && (sale_chance<0)) // He
    ↳ хочется еще больше плодить сущностей и делать отдельную функцию.
325     {
326         unsigned long long num;
327         for(int i=0; i<count_of_lots; i++)
328         {
329             cout << "Enter number of tickets and/or sale chance:\n";
330             cin >> num >> sale_chance;
331             lot_tickets.push_front(gen_lot.getLot(i, num,
    ↳ floatChanceToInt(sale_chance), seed, gen_tic));
332         }
333     }
334     else if(!count_of_tickets)
335     {
336         unsigned long long num;
337         for(int i=0; i<count_of_lots; i++)
338         {
339             cout << "Enter number of tickets and/or sale chance:\n";
340             cin >> num;
341             lot_tickets.push_front(gen_lot.getLot(i, count_of_tickets,
    ↳ sale_chance, seed, gen_tic));
342         }
343     }
344     else if(sale_chance<0)
345     {
346         for(int i=0; i<count_of_lots; i++)
347         {
348             cout << "Enter number of tickets and/or sale chance:\n";
349             cin >> sale_chance;
350             lot_tickets.push_front(gen_lot.getLot(i, count_of_tickets,
    ↳ sale_chance, seed, gen_tic));
351         }
352     }
353     else
354     {
355         sale_chance = floatChanceToInt(sale_chance);
356         for (int i = 0; i < count_of_lots; i++)
357             lot_tickets.push_front(gen_lot.getLot(i, count_of_tickets,
    ↳ sale_chance, seed, gen_tic));
358     }
359 }
360 void printWinners(ListBasicInterface<L, Ticket<T>*> &winner_tickets) const
361 {
362     if(winner_tickets.empty())

```

```

363     {
364         cout << "No winners!";
365         return;
366     }
367     for(auto it=winner_tickets.begin(); it!=winner_tickets.end(); it++)
368         cout << **it << '\n';
369 }
370 int processTour(unsigned int n, auto &lot, ListBasicInterface<L, Ticket<T*>
→ &tour_winner_tickets, int victory_type, vector<unsigned int> &kegs)
371 {
372     for(int i=0; i<n; i++) // Определение
→ победителей происходит как по условию - после каждого хода.
373     {
374         (*lot)->keg_bag.getKegs(kegs, 1);
375         for(auto tic : (*lot)->lot_tickets)
376             if(tic->getStatus() == 1)
377                 if(tic->processTicket(kegs, victory_type))
378                     tour_winner_tickets.push_front(tic);
379     }
380 }
381 public:
382 ~Game()
383 {
384     lot_tickets.clear();
385     first_tour_winner_tickets.clear();
386     second_tour_winner_tickets.clear();
387     third_tour_winner_tickets.clear();
388 };
389 int processLot(long int lot_to_process, auto lot)
390 {
391     if(lot==lot_tickets.end()) //
→ Проверка на нахождение номера тиража в списке тиражей.
392     {
393         auto lt=lot_tickets.begin();
394         for (; lt!=lot_tickets.end(); lt++)
395         {
396             if (lot_to_process == (*lt)->lot_number)
397             {
398                 lot_to_process = -1;
399                 break;
400             }
401         }
402         if(lot_to_process != -1)
403             return 1;
404         lot=lt;
405     }
406     if((*lot)->was_processed)
407         return 2;
408
409     vector<unsigned int> kegs;
410     (*lot)->keg_bag.getKegs(kegs, 5);
411     cout << "\nProcessing lot with id " << (*lot)->lot_number << "\nProcessing
→ first tour...\n";
412     for(auto tic : (*lot)->lot_tickets)
413         if(tic->getStatus()==1)
414             tic->processTicket(kegs, 1);
415     processTour(10, lot, first_tour_winner_tickets, 1, kegs); //
→ Первый параметр - количество вынимаемых бочонков в туре.
416     cout << "Processing second tour...\n";
417     processTour(30, lot, second_tour_winner_tickets, 2, kegs);
418     cout << "Processing third tour...\n";

```

```

419     processTour(15, lot, third_tour_winner_tickets, 3, kegs);
420
421     (*lot)->was_processed=true;
422     cout << (*lot)->keg_bag;
423
424     return 0;
425 }
426 int processGame(int process_mode, unsigned long long number)
427 {
428     switch(process_mode)
429     {
430         case 0:
431             for(auto lot=lot_tickets.begin(); lot!=lot_tickets.end(); lot++)
432                 if((err_code=processLot((*lot)->lot_number, lot)))
433                     errCheck_Components((*lot)->lot_number);
434             break;
435         case 1:
436             if((err_code=processLot(number, lot_tickets.end())))
437                 errCheck_Components(number);
438             break;
439         default:
440             cout << "\nNo such mode, try again.";
441             return -1;
442     }
443     return err_code;
444 }
445 void printAllWinners()
446 {
447     cout << "\n\nFirst tour winners:\n";
448     printWinners(first_tour_winner_tickets);
449     cout << "\n\nSecond tour winners:\n";
450     printWinners(second_tour_winner_tickets);
451     cout << "\n\nThird tour winners:\n";
452     printWinners(third_tour_winner_tickets);
453 }
454
455 template <class, template<class> class> friend class GeneratorGameRusLot;
456 friend const ostream& operator<< (ostream &out, Game<T, L> &game)
457 {
458     out << "Game ID: " << game.id << "\n\n";
459     for(auto it=game.lot_tickets.begin(); it!=game.lot_tickets.end(); it++)
460         out << *it;
461     return out;
462 }
463 };
464
465 template <class C, template<class> class L>
466 Ticket<C> *searchTickets(Game<C, L> &game, Ticket<C> *tic) // Функция может
    ↳ быть также использована при работе с компонентами, так как возвращает указатель
    ↳ на Ticket.
467 {
468     ifstream fp("output_file.txt");
469     if(!fp.is_open())
470     {
471         err_code=1;
472         errCheck_Main();
473     }
474     fp.ignore(10, '\n'); // Пропуск
    ↳ Game id.
475     fp.get();
476

```

```

477     long long criteria[3], initial_value=0, count_of_matching;
478     cout << "\nAvailable searching parameters:\n"
479           "Lot number;\n"
480           "Ticket number;\n"
481           "Ticket status.\n"
482           "If you enter a negative number for criteria, this criteria will not be
         ↳ taken into consideration when searching. "
483           "In that case first matching ticket will be returned.\n";
484     cin >> criteria[0] >> criteria[1] >> criteria[2];
485     for(long long i : criteria)
486         if(i<0)
487             initial_value++;
488     do
489     {
490         count_of_matching=initial_value;
491         fp >> *tic;
492         if(tic->lot_number==criteria[0] && criteria[0]>-1)
493             count_of_matching++;
494         if(tic->ticket_number==criteria[1] && criteria[1]>-1)
495             count_of_matching++;
496         if(tic->status==criteria[2] && criteria[2]>-1)
497             count_of_matching++;
498         if(count_of_matching==3)
499         {
500             cout << '\n' << *tic;
501             fp.close();
502             return tic;
503         }
504     }
505     while(fp.get()!=EOF);
506
507     fp.close();
508     cout << "Ticket not found.\n";
509     return nullptr;
510 }

```

5.0.5 generators.h

```

1  template<class T>
2  class GeneratorTicket
3  {
4  public:
5      virtual Ticket<T> *getTicket(unsigned long long lot_num, unsigned long long
         ↳ ticket_num, unsigned int status) const = 0;
6  };
7
8  class GeneratorTicketRusLot : public GeneratorTicket<FieldRusLot>
9  {
10 public:
11     Ticket<FieldRusLot> *getTicket(unsigned long long lot_num, unsigned long long
         ↳ ticket_num, unsigned int status) const override
12     {
13         return new TicketRusLot(lot_num, ticket_num, status);
14     }
15 };
16
17 template <class T, template<class> class L>
18 class GeneratorLot
19 {

```

```

20 public:
21     virtual Lot<T, L> *getLot(unsigned long long num, unsigned long long
        ↳ num_of_tickets, unsigned int sale_chance, unsigned &seed, GeneratorTicket<T>
        ↳ &gen) const = 0;
22 };
23
24 template <class T, template<class> class L>
25 class GeneratorLotRusLot : public GeneratorLot<FieldRusLot, L>
26 {
27 public:
28     Lot<FieldRusLot, L> *getLot(unsigned long long num, unsigned long long
        ↳ num_of_tickets, unsigned int sale_chance, unsigned &seed, GeneratorTicket<T>
        ↳ &gen) const override
29     {
30         return new Lot<FieldRusLot, L>(num, num_of_tickets, sale_chance, seed, gen);
31     }
32 };
33
34 template <class T, template<class> class L>
35 class GeneratorGame
36 {
37 public:
38     virtual Game<T, L> *getGame(GeneratorLot<T, L> &gen_lot, GeneratorTicket<T>
        ↳ &gen_tic) const = 0;
39 };
40
41 template <class T, template<class> class L>
42 class GeneratorGameRusLot : public GeneratorGame<FieldRusLot, L>
43 {
44 public:
45     Game<FieldRusLot, L> *getGame(GeneratorLot<T, L> &gen_lot, GeneratorTicket<T>
        ↳ &gen_tic) const override
46     {
47         return new Game<FieldRusLot, L>(gen_lot, gen_tic);
48     };
49 };
50
51 template <class T, template<class> class L>
52 class SuperGenerator
53 {
54 public:
55     GeneratorGame<T, L> *gen_game;
56     GeneratorLot<T, L> *gen_lot;
57     GeneratorTicket<T> *gen_tic;
58     SuperGenerator(GeneratorGame<T, L> *gen_one, GeneratorLot<T, L> *gen_two,
        ↳ GeneratorTicket<T> *gen_three) : gen_game(gen_one), gen_lot(gen_two),
59
60     ~SuperGenerator()
61     {
62         delete gen_game;
63         delete gen_lot;
64         delete gen_tic;
65     }
66 };
67
68 template <template<class> class L>
69 class SuperGeneratorRusLot : public SuperGenerator<FieldRusLot, L>
70 {
71 public:

```

```

72 SuperGeneratorRusLot() : SuperGenerator<FieldRusLot, L>::SuperGenerator(new
    ↳ GeneratorGameRusLot<FieldRusLot, L>(), new GeneratorLotRusLot<FieldRusLot,
    ↳ L>(),
73
                                                                    new
                                                                    ↳ GeneratorTicketRus
                                                                    ↳ {}
74 };

```

5.0.6 error_check.h

```

1  #define NON_CRITICAL "\nNON CRITICAL: "
2  #define CRITICAL     "\nCRITICAL: "
3
4  using namespace std;
5
6  static int err_code;           // Замена errno.
7
8  int errCheck_Components(unsigned int additional_information)           // Можно было
    ↳ упаковать в классы, но, по-моему, лучше так.
9  {
10     switch(err_code)
11     {
12         case 1:
13             cerr << NON_CRITICAL"Unable to find Lot with id " <<
                ↳ additional_information << " in Game!\nProcessing of this lot will not
                ↳ begin.\n\n";
14             break;
15         case 2:
16             cerr << NON_CRITICAL"Lot with id " << additional_information << " has been
                ↳ already processed.\nReprocessing is prohibited\n\n";
17             break;
18     }
19 }
20
21 int errCheck_ClientCode()
22 {
23     switch(err_code)
24     {
25         case 1:
26             cerr << NON_CRITICAL"Game processing operation ended with errors.\n\n";
27             break;
28         case 2:
29             cerr << NON_CRITICAL"Search operation has returned nullptr.\n\n";
30             break;
31     }
32     if(err_code)
33         return 2;
34     return 0;
35 }
36
37 void errCheck_Main()
38 {
39     switch(err_code)
40     {
41         case 0:
42             exit(0);
43         case 1:

```

```

44         cerr << CRITICAL"Failed to open output file.";           // Объекты,
        ↪ наследуемые от класса исключений, при критических ошибках не
        ↪ используются.
45         exit(1);
46     case 2:
47         cerr << NON_CRITICAL"Program has ended with some errors.";
48         exit(2);
49     }
50 }

```

5.0.7 list_decorator.h

```

1  template <class T>
2  struct Node
3  {
4      T data;
5      Node *next;
6  };
7
8  template <class T>
9  class List
10 {
11 private:
12     Node<T> *head;
13     Node<T> *temp;
14 public:
15     List()
16     {
17         head=nullptr;
18     }
19
20     bool empty() const
21     {
22         return (head==nullptr);
23     }
24     void push_front(const T &data)
25     {
26         temp = new Node<T>;
27         temp->data = data;
28         if(empty())
29             temp->next = nullptr;
30         else
31             temp->next = head;
32         head = temp;
33     }
34     void pop_front()
35     {
36         if(!empty())
37         {
38             Node<T> *tmp=head;
39             head=head->next;
40             delete tmp;
41         }
42     }
43     void clear()
44     {
45         if(empty())
46             return;
47         for(typename List<T>::iterator it=begin(); it!=end(); it++)

```



```

48         delete *it;
49     }
50
51     class iterator
52     {
53     friend class List;
54     private:
55         const Node<T>* current;
56     public:
57         iterator() : current(nullptr) {}
58         iterator(const Node<T>* p_node) : current(p_node) {}
59
60         iterator& operator=(Node<T>* node)
61         {
62             this->current = node;
63             return *this;
64         }
65         iterator& operator++()
66         {
67             if(current)
68                 current = current->next;
69             return *this;
70         }
71         iterator operator++(int)
72         {
73             iterator iterator=*this;
74             ++*this;
75             return iterator;
76         }
77         bool operator==(const iterator& it) const
78         {
79             return current == it.current;
80         }
81         bool operator!=(const iterator& it) const
82         {
83             return current != it.current;
84         }
85         T operator*() const
86         {
87             return current->data;
88         }
89     };
90     iterator begin() const
91     {
92         return iterator(head);
93     }
94     iterator end() const
95     {
96         return iterator(nullptr);
97     }
98 };
99
100 template<template<typename> class Container, typename T>
101 class ListBasicInterface
102 {
103 private:
104     Container<T> lst;
105 public:
106     ListBasicInterface() = default;
107     typename Container<T>::iterator begin()
108     {

```

```

109         return lst.begin();
110     }
111     typename Container<T>::iterator end()
112     {
113         return lst.end();
114     }
115     bool empty() const
116     {
117         return lst.empty();
118     }
119     void push_front(const T &data)
120     {
121         lst.push_front(data);
122     }
123     void pop_front()
124     {
125         lst.pop_front();
126     }
127     void clear()
128     {
129         lst.clear();
130     }
131 };

```

[2] [3] [1]