

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	24 мая 2021 г.

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	6
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	14
4	Литература	15
5	Приложение	16
5.0.1	main.cpp	16
5.0.2	client_code.h	16
5.0.3	functions.h	18
5.0.4	components.h	18
5.0.5	generators.h	27
5.0.6	error_check.h	28
5.0.7	list_decorator.h	29

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, передача в конструктор класса 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 типа>_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 типа>_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 - приватных 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 `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 макросом NUMBER_OF_KEGS, специально сделано неизменяемым с пользовательской стороны. Класс имеет метод для доставания 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 тире 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 верхе иерархии лотереи, имея список класса Lot (тираж). Конструктор класса создает лотерию, иерархично создавая объекты классов типа Lot и Ticket. Game имеет методы вывода в стандартный поток списков билетов-победителей printWinners(ListBasicInterface<, Ticket<>*>), 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. 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.

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 обобщий 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 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. 1: Пользователь вводит операцию обработки первого и второго тиража билетов, 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. 2: 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. 3: 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

```

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. 5: 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.

```

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

```

```
Enter operation:
s

Enter lot number, ticket number, status to find ticket.
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
0 -16 -21 0 41 54 -66 0 0
0 -18 0 0 0 -58 63 -71 -81
-6 0 0 30 -45 0 0 -71 88
Field 1
0 -11 20 0 0 -51 0 -73 82
1 0 23 -34 -40 0 -66 0 0
0 -14 24 0 0 -52 0 75 -84
SOLD
```

Рис. 6: Пользователь вводит операцию поиска билета и находит билет согласно введенным параметрам.

```
Enter operation:

C

Process finished with exit code 0
```

Рис. 7: Пользователь завершает свою работу с программой, которая отработала без ошибок.

3 Вывод

В этой курсовой работе мною были продемонстрированы навыки работы с шаблонными и абстрактными классами, пример проектирования приложения с помощью паттерна "Шаблонный метод". Также был реализован интерфейс для односвязных списков с использованием паттерна "Декоратор" и продемонстрирована работа спроектированного приложения.

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 "generators.h"
5 #include "client_code.h"
6
7 using namespace std;
8
9 int main()
10 {
11     int err_code; // Замена errno.
12     ofstream ofp("output_file.txt");
13     if(!ofp.is_open())
14     {
15         err_code=1;
16         errCheck_Main(err_code, 0);
17     }
18
19     srand(static_cast <unsigned>(time(nullptr)));
20
21     if(clientCode(ofp, err_code))
22         err_code=2;
23     ofp.close();
24
25     errCheck_Main(err_code, 0);
26 }
```

5.0.2 client_code.h

```
1 #ifndef FA_KURS_CLIENT_CODE_H
2 #define FA_KURS_CLIENT_CODE_H
3
4 #include <forward_list>
5
6 #define LIST_IMPLEMENTATION forward_list
7 #define BAD_ACCESS "You cannot access this operation: "
8
9 int clientCode(ofstream &fp, int err_code)
10 {
11     SuperGenerator<FieldRusLot, LIST_IMPLEMENTATION> *super_gen = new
12     ↪ SuperGeneratorRusLot<LIST_IMPLEMENTATION>();
13     Game<FieldRusLot, LIST_IMPLEMENTATION> *game=super_gen->gen_game->getGame(0, 3,
14     ↪ 20000, 0.8, *super_gen->gen_lot,
15     ↪ *super_gen->gen_
16
17     cout << "Available operations:\n"
18     ↪ "'p' - enter process mode and id of lot to process. In zero mode id is
19     ↪ useless;\n"
20     ↪ "'o' - print winners;\n"
21     ↪ "'f' - write processed data to output_file;\n"
22     ↪ "'s' - search specific tickets in output_file;\n"
23     ↪ "'c' - finish work and exit program.";
24
25     char operation;
26     int process_mode, number;
```



```

22     bool proceed=true, p_not_blocked=true, s_not_blocked=false, o_f_not_blocked=false;
23     while(proceed)
24     {
25         cout << "\n\nEnter operation:\n";
26         cin >> operation;
27         switch(operation)
28         {
29             case 'p':
30                 if(p_not_blocked)
31                 {
32                     cin >> process_mode >> number;
33                     while((err_code = game->processGame(process_mode, number,
34                     ↪ err_code)) == -1)
35                         cin >> process_mode;
36                     if(err_code)
37                     {
38                         err_code=1;
39                         p_not_blocked=false;
40                         break;
41                     }
42                     o_f_not_blocked=true;
43                 }
44             else
45                 cout << BAD_ACCESS"Last calculation ended with errors.\n";
46                 break;
47             case 'o':
48                 if(o_f_not_blocked)
49                     game->printAllWinners();
50                 else
51                 cout << BAD_ACCESS"Game has not been processed or processing has
52                 ↪ ended with errors.\n";
53                 break;
54             case 'f':
55                 if(o_f_not_blocked)
56                 {
57                     fp << *game;
58                     s_not_blocked=true;
59                     cout << "Done.\n";
60                 }
61                 else
62                 cout << BAD_ACCESS"Game has not been processed or processing has
63                 ↪ ended with errors.\n";
64                 break;
65             case 's':
66                 if(s_not_blocked)
67                 {
68                     if(!searchTickets(*game, super_gen->gen_tic->getTicket(0, 0, -1),
69                     ↪ err_code))
70                         err_code=1;
71                 }
72                 else
73                 cout << BAD_ACCESS"Processed data has not been written to
74                 ↪ file.\n";
75                 break;
76             case 'c':
77                 proceed=false;
78                 break;

```

```

78
79         default:
80             cout << "No such operation, try again.\n";
81     }
82 }
83 delete super_gen; delete game;
84
85 return errCheck_ClientCode(err_code, 0);
86 }
87
88 #endif

```

5.0.3 functions.h

```

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

```

5.0.4 components.h

```

1  #ifndef FA_KURS_COMPONENTS_H
2  #define FA_KURS_COMPONENTS_H
3
4  #include <algorithm>
5  #include <vector>
6  #include <cstdlib>
7  #include <random>
8  #include <chrono>
9  #include "list_decorator.h"
10 #include "generators.h"
11 #include "functions.h"
12
13 #define NUMBER_OF_KEGS 90
14

```

```

15 using namespace std;
16
17 template<class T>
18 class GeneratorTicket;
19 template <class T, template<class> class L>
20 class GeneratorLot;
21
22 class KegBag
23 {
24 private:
25     vector<unsigned int> bag;
26 public:
27     KegBag()
28     {
29         unsigned seed = chrono::system_clock::now().time_since_epoch().count() +
            → rand();           // rand() прибавлен из-за того, что сид практически не
            → меняется в единицу времени.
30         for(int i=1; i<NUMBER_OF_KEGS; i++)
31             bag.push_back(i);
32         shuffle(bag.begin(), bag.end(), default_random_engine(seed));
33     }
34     void getKegs(vector<unsigned int> &vec, unsigned int amount)
35     {
36         vec.clear();
37         for(int i=0; i<amount; i++)
38         {
39             vec.push_back(bag.back());
40             bag.pop_back();
41         }
42     }
43
44     friend const ostream& operator<<(ostream &out, const KegBag &keg_bag)
45     {
46         out << "Remaining kegs in bag:";
47         for(auto it : keg_bag.bag)
48             out << ' ' << it;
49         return out;
50     }
51 };
52
53 class Field
54 {
55 protected:
56     unsigned int row_count;
57     unsigned int col_count;
58     int8_t **field;
59 public:
60     Field(unsigned int rows, unsigned int cols) : row_count(rows), col_count(cols)
61     {
62         unsigned int j;
63         int8_t field_values_in_row[2][5];
64
65         field=new int8_t *[row_count];
66         for(unsigned int i=0; i<row_count; i++)
67         {
68             field[i]=new int8_t[col_count];
69             for(j=0; j<col_count; j++)
70                 field[i][j] = 0;
71
72             getValues(field_values_in_row);
73             for (j = 0; j < 5; j++)

```

```

74         field[i][field_values_in_row[0][j]] = field_values_in_row[1][j];
75     }
76 }
77 ~Field()
78 {
79     for(int i=0; i<row_count; i++)
80         delete[] field[i];
81     delete[] field;
82 }
83 friend const ostream& operator<< (ostream &out, const Field &fld);
84 friend istream& operator>> (istream &in, Field &fld);
85 template<class> friend class Ticket;
86 };
87
88 class FieldRusLot : public Field
89 {
90 public:
91     FieldRusLot() : Field(3, 9) {}
92 };
93
94 const ostream& operator<< (ostream &out, const Field &fld)
95 {
96     for(int i=0; i<fld.row_count; i++)
97     {
98         for(int j=0; j<fld.col_count; j++)
99             out << (int)fld.field[i][j] << ' ';
100         out << '\n';
101     }
102     return out;
103 }
104
105 istream& operator>> (istream &in, Field &fld)
106 {
107     int tmp;
108     for(int i=0; i<fld.row_count; i++)
109     {
110         for(int j=0; j<fld.col_count; j++)
111         {
112             in >> tmp;
113             fld.field[i][j]=tmp;
114         }
115         in.get(); in.get();
116     }
117 }
118
119 template <class T, template<class> class L> class Game;
120
121 template<class T>
122 class Ticket
123 {
124 protected:
125     unsigned int lot_number, ticket_number, status;
126     vector<T> fields;
127 public:
128     Ticket(unsigned int lot_num, unsigned int ticket_num, unsigned int field_count,
129         ↪ unsigned int status) :
130         lot_number(lot_num), ticket_number(ticket_num),
131         fields(field_count), status(status) {}
132     ~Ticket()
133     {
134         fields.clear();

```

```

134     }
135     int getStatus()
136     {
137         return status;
138     }
139     int processTicket(vector<unsigned int> &numbers, int vic_type)
140     {
141         int i, j;
142         int counters[3] = {0}; // Проверяют
143         → строку, поле и билет на зачеркивание всех чисел в строке/поле/всех полях.
144         for(auto &fld : fields)
145         {
146             counters[1]=0;
147             for(i=0; i<fld.row_count; i++)
148             {
149                 counters[2]=0;
150                 for(j=0; j<fld.col_count; j++)
151                 {
152                     for(auto &num : numbers)
153                     {
154                         if(fld.field[i][j] < 0)
155                             counters[2]++;
156                         else if (num == fld.field[i][j])
157                         {
158                             fld.field[i][j]*=-1;
159                             counters[2]++;
160                             break;
161                         }
162                     }
163                 }
164                 if(counters[2]==5)
165                 {
166                     if(vic_type==1)
167                     {
168                         status = 2;
169                         return 1;
170                     }
171                     counters[1]++;
172                 }
173             }
174             if(counters[1]==3)
175             {
176                 counters[0]++;
177                 if(vic_type==2)
178                 {
179                     status = 3;
180                     return 1;
181                 }
182             }
183             if(counters[0]==2 && vic_type==3)
184             {
185                 status = 4;
186                 return 1;
187             }
188             return 0;
189         }
190         friend ostream& operator<< (ostream &out, Ticket<T> &tic)
191         {
192             out << "Lot series: " << tic.lot_number << '\n';
193             out << "Ticket: " << tic.ticket_number << '\n';

```

```

194
195     unsigned int field_count=tic.fields.size();
196     for(int i=0; i<field_count; i++)
197         out << "Field " << i << '\n' << tic.fields[i];
198
199     switch(tic.status)
200     {
201         case 0:
202             out << "UNSOLD";
203             break;
204         case 1:
205             out << "SOLD";
206             break;
207         case 2:
208             out << "1-st TOUR WINNER, SOLD";
209             break;
210         case 3:
211             out << "2-nd TOUR WINNER, SOLD";
212             break;
213         case 4:
214             out << "3-rd TOUR WINNER, SOLD";
215             break;
216     }
217     out << '\n';
218     return out;
219 }
220 friend istream& operator>> (istream &in, Ticket<T> &tic)
221 {
222     unsigned int size=tic.fields.size();
223     char c;
224
225     in.ignore(12);
226     in >> tic.lot_number;
227     in.ignore(8, ' ');
228     in >> tic.ticket_number;
229     in.get(c);
230     for(int i=0; i<size; i++)
231     {
232         in.ignore(20, '\n');
233         in>>tic.fields[i];
234     }
235     in.get(c);
236     switch(c)
237     {
238         case 'U':
239             tic.status = 0;
240             break;
241         case 'S':
242             tic.status = 1;
243             break;
244         case '1':
245             tic.status = 2;
246             break;
247         case '2':
248             tic.status = 3;
249             break;
250         case '3':
251             tic.status = 4;
252             break;
253     }

```

// Пропуск
↪ Field.

```

254     in.ignore(30, '\n');
255 }
256 template <class C, template<class> class L> friend Ticket<C>
    ↪ *searchTickets(Game<C, L> &game, Ticket<C> *tic, int &err_code);
257 template <class, template<class> class> friend class Lot;
258 };
259
260 class TicketRusLot : public Ticket<FieldRusLot>
261 {
262 public:
263     TicketRusLot(unsigned int lot_num, unsigned int ticket_num, unsigned int status) :
        ↪ Ticket(lot_num, ticket_num, 2, status) {}
264 };
265
266 template <class T, template<class> class L>
267 class Lot
268 {
269 private:
270     bool was_processed;
271     unsigned int lot_number;
272     KegBag keg_bag;
273     ListBasicInterface<L, Ticket<T>*> lot_tickets;
274 public:
275     Lot(unsigned int num, unsigned int num_of_tickets, unsigned int sale_chance,
        ↪ GeneratorTicket<T> &gen) : was_processed(false), lot_number(num)
276     {
277         if(rand()%2 || sale_chance==100)
278             ↪ // Полная или неполная реализация тиража.
                for(unsigned int i=0; i<num_of_tickets; i++)
279                 lot_tickets.push_front(gen.getTicket(lot_number, i, 1));
280         else
281             for(unsigned int i = 0; i < num_of_tickets; i++)
282                 lot_tickets.push_front(gen.getTicket(lot_number,
                    ↪ i, (rand()%100)<sale_chance));
283     }
284     ~Lot()
285     {
286         lot_tickets.clear();
287     }
288     template <class, template<class> class> friend class Game;
289     friend ostream& operator<< (ostream &out, Lot<T, L> &lot)
290     {
291         for(auto it = lot.lot_tickets.begin(); it!=lot.lot_tickets.end(); it++)
292             out << *it << '\n';
293         return out;
294     }
295 };
296
297
298 template <class T, template<class> class L>
299 class Game
300 {
301 private:
302     unsigned int game_id;
303     unsigned int game_size;
304     ListBasicInterface<L, Lot<T, L>*> lot_tickets;
305     ListBasicInterface<L, Ticket<T>*> first_tour_winner_tickets;
306     ListBasicInterface<L, Ticket<T>*> second_tour_winner_tickets;
307     ListBasicInterface<L, Ticket<T>*> third_tour_winner_tickets;
308

```

```

309 Game(unsigned int id, unsigned int num_of_lot, unsigned int num_of_tic, float
    ↪ sale_chance, GeneratorLot<T, L> &gen_lot, GeneratorTicket<T> &gen_tic)
310     : game_id(id), game_size(num_of_lot)
311 {
312     sale_chance*=100;
313     if(sale_chance<1 || sale_chance>100)
314         sale_chance=rand() %100 +1;
315
316     for(int i=0; i<game_size; i++)
317         lot_tickets.push_front(gen_lot.getLot(i, num_of_tic, sale_chance,
    ↪ gen_tic));
318 }
319
320 void printWinners(ListBasicInterface<L, Ticket<T>*> &winner_tickets)
321 {
322     if(winner_tickets.empty())
323     {
324         cout << "No winners!";
325         return;
326     }
327     for(auto it=winner_tickets.begin(); it!=winner_tickets.end(); it++)
328         cout << **it << '\n';
329 }
330 int processTour(unsigned int n, auto &lot, ListBasicInterface<L, Ticket<T>*>
    ↪ &tour_winner_tickets, int victory_type, vector<unsigned int> &kegs)
331 {
332     for(int i=0; i<n; i++)
333     {
334         (*lot)->keg_bag.getKegs(kegs, 1);
335         for (auto tic : (*lot)->lot_tickets)
336             if (tic->getStatus() == 1)
337                 if (tic->processTicket(kegs, victory_type))
338                     tour_winner_tickets.push_front(tic);
339     }
340 }
341 public:
342 ~Game()
343 {
344     lot_tickets.clear();
345     first_tour_winner_tickets.clear();
346     second_tour_winner_tickets.clear();
347     third_tour_winner_tickets.clear();
348 };
349 int processLot(long int lot_to_process, auto lot)
350 {
351     if(lot==lot_tickets.end()) //
    ↪ Проверка на нахождение номера тиража в списке тиражей.
352     {
353         auto lt=lot_tickets.begin();
354         for (; lt!=lot_tickets.end(); lt++)
355         {
356             if (lot_to_process == (*lt)->lot_number)
357             {
358                 lot_to_process = -1;
359                 break;
360             }
361         }
362         if(lot_to_process != -1)
363             return 1;
364         lot=lt;
365     }

```



```

366         if((*lot)->was_processed)
367             return 2;
368
369         vector<unsigned int> kegs;
370         (*lot)->keg_bag.getKegs(kegs, 5);
371         cout << "\nProcessing lot with id " << (*lot)->lot_number << "\nProcessing
    ↳ first tour...\n";
372         for(auto tic : (*lot)->lot_tickets)
373             if(tic->getStatus()==1)
374                 tic->processTicket(kegs, 1);
375         processTour(10, lot, first_tour_winner_tickets, 1, kegs);           // Первый
    ↳ параметр - количество вынимаемых бочонков в туре.
376         cout << "Processing second tour...\n";
377         processTour(30, lot, second_tour_winner_tickets, 2, kegs);
378         cout << "Processing third tour...\n";
379         processTour(15, lot, third_tour_winner_tickets, 3, kegs);
380
381         (*lot)->was_processed=true;
382         cout << (*lot)->keg_bag;
383
384         return 0;
385     }
386     int processGame(int process_mode, unsigned int number, int &err_code)
387     {
388         switch(process_mode)
389         {
390             case 0:
391                 for(auto lot = lot_tickets.begin(); lot != lot_tickets.end(); lot++)
392                     if((err_code=processLot((*lot)->lot_number, lot)))
393                         errCheck_Components(err_code, (*lot)->lot_number);
394                 break;
395             case 1:
396                 if((err_code=processLot(number, lot_tickets.end())))
397                     errCheck_Components(err_code, number);
398                 break;
399             default:
400                 cout << "\nNo such mode, try again.";
401                 return -1;
402         }
403         return err_code;
404     }
405     void printAllWinners()
406     {
407         cout << "\n\nFirst tour winners:\n";
408         printWinners(first_tour_winner_tickets);
409         cout << "\nSecond tour winners:\n";
410         printWinners(second_tour_winner_tickets);
411         cout << "\nThird tour winners:\n";
412         printWinners(third_tour_winner_tickets);
413     }
414
415     template <class, template<class> class> friend class GeneratorGameRusLot;
416     friend ostream& operator<< (ostream &out, Game<T, L> &game)
417     {
418         out << "Game ID: " << game.game_id << "\n\n";
419         for(auto it=game.lot_tickets.begin(); it!=game.lot_tickets.end(); it++)
420             out << **it;
421         return out;
422     }
423 };
424

```

```

425 template <class C, template<class> class L>
426 Ticket<C> *searchTickets(Game<C, L> &game, Ticket<C> *tic, int &err_code) //
    ↳ По обработке ошибок и принципу работы (не в бесконечном цикле) предполагается, что
    ↳ функция
427
//
    ↳ должна
    ↳ использоваться
    ↳ для
    ↳ работы
    ↳ с
    ↳ компонентой
    ↳ не
    ↳ для
    ↳ пользователя

428 {
429     ifstream fp("output_file.txt");
430     if(!fp.is_open())
431     {
432         err_code=1;
433         errCheck_Main(err_code, 0);
434     }
435     fp.ignore(10, '\n'); // Пропуск
    ↳ Game id.
436     fp.get();
437
438     int criteria[3], initial_value=0, count_of_matching;
439     cout << "\nEnter lot number, ticket number, status to find ticket.\n"
440           "If you enter a negative number for criteria, this criteria will not be
    ↳ taken into consideration when searching."
441           "\nIn that case first matching ticket will be returned.\n";
442     cin >> criteria[0] >> criteria[1] >> criteria[2];
443     for(int i : criteria)
444         if(i<0)
445             initial_value++;
446     do
447     {
448         count_of_matching=initial_value;
449         fp >> *tic;
450         if(tic->lot_number==criteria[0] && criteria[0]>-1)
451             count_of_matching++;
452         if(tic->ticket_number==criteria[1] && criteria[1]>-1)
453             count_of_matching++;
454         if(tic->status==criteria[2] && criteria[2]>-1)
455             count_of_matching++;
456         if(count_of_matching==3)
457         {
458             cout << '\n' << *tic;
459             fp.close();
460             return tic;
461         }
462     }
463     while(fp.get() != EOF);
464
465     fp.close();
466     cout << "Ticket not found.\n";
467     return nullptr;
468 }
469 #endif

```

5.0.5 generators.h

```
1  #ifndef FA_KURS_GENERATORS_H
2  #define FA_KURS_GENERATORS_H
3
4  #include "error_check.h"
5  #include "components.h"
6
7  template<class T>
8  class GeneratorTicket
9  {
10 public:
11     virtual Ticket<T> *getTicket(unsigned int lot_num, unsigned int ticket_num,
12     ↪ unsigned int status) const = 0;
13     template <class, template<class> class> friend class Lot;
14 };
15
16 class GeneratorTicketRusLot : public GeneratorTicket<FieldRusLot>
17 {
18 public:
19     Ticket<FieldRusLot> *getTicket(unsigned int lot_num, unsigned int ticket_num,
20     ↪ unsigned int status) const override
21     {
22         return new TicketRusLot(lot_num, ticket_num, status);
23     }
24 };
25
26 template <class T, template<class> class L>
27 class GeneratorLot
28 {
29 public:
30     virtual Lot<T, L> *getLot(unsigned int num, unsigned int num_of_tickets, unsigned
31     ↪ int sale_chance, GeneratorTicket<T> &gen) const=0;
32 };
33
34 template <class T, template<class> class L>
35 class GeneratorLotRusLot : public GeneratorLot<FieldRusLot, L>
36 {
37 public:
38     Lot<FieldRusLot, L> *getLot(unsigned int num, unsigned int num_of_tickets,
39     ↪ unsigned int sale_chance, GeneratorTicket<T> &gen) const override
40     {
41         return new Lot<FieldRusLot, L>(num, num_of_tickets, sale_chance, gen);
42     }
43 };
44
45 template <class T, template<class> class L>
46 class GeneratorGame
47 {
48 public:
49     virtual Game<T, L> *getGame(unsigned int id, unsigned int num_of_lot, unsigned int
50     ↪ num_of_tic, float sale_chance,
51     ↪ GeneratorLot<T, L> &gen_lot, GeneratorTicket<T>
52     ↪ &gen_tic) const=0;
53 };
54
55 template <class T, template<class> class L>
56 class GeneratorGameRusLot : public GeneratorGame<FieldRusLot, L>
57 {
58 public:
```

```

53     Game<FieldRusLot, L> *getGame(unsigned int id, unsigned int num_of_lot, unsigned
    ↪ int num_of_tic, float sale_chance,
54                                     GeneratorLot<T, L> &gen_lot, GeneratorTicket<T>
    ↪ &gen_tic) const override
55     {
56         return new Game<FieldRusLot, L>(id, num_of_lot, num_of_tic, sale_chance,
    ↪ gen_lot, gen_tic);
57     };
58 };
59
60 template <class T, template<class> class L>
61 class SuperGenerator
62 {
63 public:
64     GeneratorGame<T, L> *gen_game;
65     GeneratorLot<T, L> *gen_lot;
66     GeneratorTicket<T> *gen_tic;
67     SuperGenerator(GeneratorGame<T, L> *gen_one, GeneratorLot<T, L> *gen_two,
    ↪ GeneratorTicket<T> *gen_three) : gen_game(gen_one), gen_lot(gen_two),
68
69     ~SuperGenerator()
70     {
71         delete gen_game;
72         delete gen_lot;
73         delete gen_tic;
74     }
75 };
76
77 template <template<class> class L>
78 class SuperGeneratorRusLot : public SuperGenerator<FieldRusLot, L>
79 {
80 public:
81     SuperGeneratorRusLot() : SuperGenerator<FieldRusLot, L>::SuperGenerator(new
    ↪ GeneratorGameRusLot<FieldRusLot, L>(), new GeneratorLotRusLot<FieldRusLot,
    ↪ L>(),
82                                     new
    ↪ GeneratorTicketRusLot<FieldRusLot, L>(),
    ↪ {}
83 };
84
85 #endif

```

5.0.6 error_check.h

```

1  #ifndef FA_KURS_ERROR_CHECK_H
2  #define FA_KURS_ERROR_CHECK_H
3
4  #define NON_CRITICAL "\nNON CRITICAL: "
5  #define CRITICAL    "\nCRITICAL: "
6
7  using namespace std;
8
9  int errCheck_Components(int err_code, unsigned int additional_information) //
    ↪ Можно было упаковать в классы, но, по-моему, лучше так.
10 {
11     switch(err_code)
12     {
13         case 1:

```

```

14         cerr << NON_CRITICAL"Unable to find Lot with id " <<
           ↳ additional_information << " in Game!\nProcessing of this lot will not
           ↳ begin.\n\n";
15         break;
16     case 2:
17         cerr << NON_CRITICAL"Lot with id " << additional_information << " has been
           ↳ already processed.\nReprocessing is prohibited\n\n";
18         break;
19     }
20 }
21
22 int errCheck_ClientCode(int err_code, long int additional_information)
23 {
24     switch(err_code)
25     {
26     case 1:
27         cerr << NON_CRITICAL"Game processing operation ended with errors.\n\n";
28         break;
29     case 2:
30         cerr << NON_CRITICAL"Search operation has returned nullptr.\n\n";
31         break;
32     }
33     if(err_code)
34         return 2;
35     return 0;
36 }
37
38 int errCheck_Main(int err_code, long int additional_information)
39 {
40     switch(err_code)
41     {
42     case 1:
43         cerr << CRITICAL"Failed to open output file.";           // Объекты,
           ↳ наследуемые от класса исключений, не используются.
44         exit(1);
45     case 2:
46         cerr << NON_CRITICAL"Program has ended with some errors.";
47         exit(2);
48     default:
49         exit(0);
50     }
51 }
52
53 #endif

```

5.0.7 list_decorator.h

```

1  #ifndef FA_KURS_LIST_DECORATOR_H
2  #define FA_KURS_LIST_DECORATOR_H
3
4  template <class T>
5  struct Node
6  {
7      T data;
8      Node *next;
9  };
10
11 template <class T>
12 class List

```

```

13 {
14 private:
15     Node<T> *head;
16     Node<T> *temp;
17 public:
18     List()
19     {
20         head=nullptr;
21     }
22
23     bool empty() const
24     {
25         return (head==nullptr);
26     }
27     void push_front(T data)
28     {
29         temp = new Node<T>;
30         temp->data = data;
31         if(empty())
32             temp->next = nullptr;
33         else
34             temp->next = head;
35         head = temp;
36     }
37     bool pop_front(T data)
38     {
39         temp = head;
40         Node<T> *prev;
41         while(temp->next != nullptr && temp->data != data)
42         {
43             prev = temp;
44             temp = temp->next;
45         }
46         if(temp->data == data)
47         {
48             prev->next = temp->next;
49             delete temp;
50             return true;
51         }
52         return false;
53     }
54     void clear()
55     {
56         if(empty())
57             return;
58         for(typename List<T>::iterator it=begin(); it!=end(); it++)
59             delete *it;
60     }
61
62     class iterator
63     {
64     friend class List;
65     private:
66         const Node<T>* current;
67     public:
68         iterator() : current(nullptr) {}
69
70         iterator(const Node<T>* pNode) noexcept : current(pNode) {}
71         iterator& operator=(Node<T>* node)
72         {
73             this->current = node;

```

```

74         return *this;
75     }
76     iterator& operator++()
77     {
78         if(current)
79             current = current->next;
80         return *this;
81     }
82     iterator operator++(int)
83     {
84         iterator iterator=*this;
85         ++*this;
86         return iterator;
87     }
88     bool operator==(const iterator& it)
89     {
90         return current == it.current;
91     }
92     bool operator!=(const iterator& it)
93     {
94         return current != it.current;
95     }
96     T operator*()
97     {
98         return current->data;
99     }
100 };
101 iterator begin() const
102 {
103     return iterator(head);
104 }
105 iterator end() const
106 {
107     return iterator(nullptr);
108 }
109 };
110
111 template<template<typename> class Container, typename T>
112 class ListBasicInterface
113 {
114 private:
115     Container<T> lst;
116 public:
117     ListBasicInterface() = default;
118     typename Container<T>::iterator begin()
119     {
120         return lst.begin();
121     }
122     typename Container<T>::iterator end()
123     {
124         return lst.end();
125     }
126     bool empty() const
127     {
128         return lst.empty();
129     }
130     void push_front(const T &data)
131     {
132         lst.push_front(data);
133     }
134     void pop_front()

```

```
135     {
136         lst.pop_front();
137     }
138     void clear()
139     {
140         lst.clear();
141     }
142 };
143
144 #endif
```

[2] [3] [1]