МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Кафедра алгоритмических языков

Отчет о выполнении задания практикума

«*Моделирование FM-радиостанции*»

Автор: студент 4 курса гр. 424 Д.В. Горбунов

Москва, 2017

Постановка задачи

Некоторая радиостанция осуществляет круглосуточную трансляцию музыкальных произведений. В течение суток радиостанция предлагает несколько радиопрограмм ($7 \le K \le 12$), посвященных разным жанрам музыки. Существует два вида программ: в одних подбор произведений выполняется по заявкам пользователей, другие же программы составляются как хит-парады.

Длительность каждой программы – M часов ($1 \le M \le 3$). Необходимо создать компьютерную систему, составляющую программы радиостанции в течение дня на основе поступающих заявок слушателей. В фонотеке радиостанции хранятся музыкальные записи разных жанров и исполнителей. Каталог фонотеки учитывает для каждой музыкальной записи: жанр музыки (классика, джаз, рок, поп, рэп и др.), название произведения, авторы, исполнители, название и год выпуска альбома, количество минут звучания, рейтинг.

Для составления программы по заявкам слушателей система фиксирует поступающие по телефону заявки, в которых заказывается либо конкретное музыкальное произведение, либо любое произведение определенного автора, либо любое произведение из некоторого альбома, либо любая запись определенного исполнителя. Заявки выполняются по возможности последовательно, но так, чтобы не допускать однообразия исполняемых подряд произведений (например, не допускается подряд один и тот же исполнитель).

При большом количестве поступивших заявок делается попытка выбрать очередную музыкальную запись так, чтобы удовлетворить несколько заявок. При невозможности выполнить все заявки удовлетворяются те, которые позволяют составить более разнообразную программу.

В хит-парадах проигрываются произведения определенного жанра, получившие наибольший рейтинг за последние дни. Рейтинг рассчитывается по поступившим заявкам слушателей, отдельно по каждому жанру. Для тестирования построенной модели составления радиопрограмм необходимо статистически смоделировать поток заявок от слушателей. Каждая составляющая заявки (автор, произведение, альбом, исполнитель) определяется случайным образом. Период моделирования – N дней ($1 \le N \le 7$).

Спецификация классов

structures.h

```
1   #ifndef STRUCTURES
2   #define STRUCTURES
3
4   #include <vector>
5   #include <string>
6   #include <memory>
7   #include <algorithm>
8
9   struct Author_;
10  struct Song_;
11  struct Album_;
12  struct Genre_;
13  struct Play_;
14  struct Query_;
15  struct Event_;
16
17  using Author = std::shared_ptr<Author_>;
18  using Song = std::shared_ptr<Song_>;
19  using Album = std::shared_ptr<Album_>;
20  using Genre = std::shared_ptr<Genre_>;
21  using Play = std::shared_ptr<Play_>;
22  using Query = std::shared_ptr<Query_>;
23  using Event = std::shared_ptr<Event_>;
24   // Структура для хранения автора
25  struct Author_ {
26      std::string name;
27      std::vector<Song> songs;
28      std::vector<Album> albums;
29   // Вывести в stream описание автора
30      void describe(std::ostream& stream);
31   // Вернуть строку с корректным именем автора
32      std::string processedName();
33  };
34   // Структура для хранения жанра
35  struct Genre_ {
36      std::string name;
37   // Вывести в stream описание жанра
38      void describe(std::ostream& stream);
39   // Вернуть строку с корректным названием жанра
40      std::string processedName();
41  };
42   // Структура для хранения альбома
43  struct Album_ {
44      Author author;
45      std::string name;
46      Genre genre;
47      std::vector<Song> songs;
```

```
48   // Вывести в stream описание альбома
49       void describe(std::ostream& stream);
50   // Вернуть строку с корректным названием альбома
51       std::string processedName();
52   };
53   // Структура для хранения песни
54   struct Song_ {
55       Author author;
56       Album album;
57       std::string name;
58       int duration;
59       Genre genre;
60   // Вывести в stream описание песни
61       void describe(std::ostream& stream);
62   // Вернуть строку с корректным названием песни
63       std::string processedName();
64   // Вернуть строку с корректным описанием песни
65       std::string processedDescription();
66   };
67   // Структура для хранения проигрывания
68   struct Play_ {
69       Play_(Song song, int timestamp);
70
71       Song song;
72       int timestamp;
73   };
74   // Структура для хранения запроса от слушателей
75   struct Query_ {
76   public:
     // Проверка, подходит ли песня под запрос слушателя
77       virtual bool songMatches(Song song) = 0;
     // Вывести в stream описание запроса
78       virtual void describe(std::ostream& stream) = 0;
79   // Выдать список песен, подходящих под запрос
80       template<typename Catalog>
81       std::vector<Song> findPossibleSongs(Catalog catalog);
82   };
83   // Структура для хранения запроса песни от слушателей
84   struct SongQuery_ : public Query_ {
85   public:
86       SongQuery_(Song song);
87
88       bool songMatches(Song song);
89
90       void describe(std::ostream& stream);
91
92   private:
93       Song song;
94   };
95   // Структура для хранения запроса альбома от слушателей
96   struct AlbumQuery_ : public Query_ {
97   public:
```

```
98      AlbumQuery_(Album album);
99
100     bool songMatches(Song song);
101
102     void describe(std::ostream& stream);
103
104 private:
105     Album album;
106 };
107  // Структура для хранения запроса исполнителя от слушателей
108 struct AuthorQuery_ : public Query_ {
109 public:
110     AuthorQuery_(Author author);
111
112     bool songMatches(Song song);
113
114     void describe(std::ostream& stream);
115
116 private:
117     Author author;
118 };
119
120 #endif // STRUCTURES
```

catalog.h

```
1   #ifndef CATALOG_H
2   #define CATALOG_H
3
4   #include <fstream>
5   #include <algorithm>
6   #include <map>
7   #include <iostream>
8   #include <cassert>
9   #include <memory>
10
11  class Catalog_;
12
13  using Catalog = std::shared_ptr<Catalog_>;
14
15  #include "structures.h"
16   // Структура для хранения каталога песен
17  class Catalog_
18  {
19  public:
20      Catalog_(std::string fileName);
21   // Выдать список всех песен
22      const std::vector<Song>& getSongs();
23   // Выдать список всех исполнителей
24      const std::vector<Author>& getAuthors();
25   // Выдать список всех альбомов
26      const std::vector<Album>& getAlbums();
27   // Выдать список всех жанров
28      const std::vector<Genre>& getGenres();
29   // Выдать список всех песен с данным жанром
30      const std::vector<Song>& getSongsByGenre(Genre genre);
31   // Найти исполнителя по имени
32      Author findAuthor(std::string name);
33   // Найти жанр по имени
34      Genre findGenre(std::string name);
35   // Найти альбом по имени и имени исполнителя
36      Album findAlbum(std::string name, std::string author);
37   // Найти песню по имени, имени альбома и имени исполнителя
38      Song findSong(std::string name, std::string album, std::string
    author);
39   // Структура для хранения считанной из каталога записи о жанре
40      struct GenreEntity {
41        std::string name;
42
43        friend std::istream& operator >> (std::istream &stream,
    GenreEntity &entity);
44      };
45   // Структура для хранения считанной из каталога записи об
    исполнителе
```

```
46      struct AuthorEntity {
47        std::string name;
48
49        friend std::istream& operator >> (std::istream &stream,
    AuthorEntity &entity);
50      };
51   // Структура для хранения считанной из каталога записи об альбоме
52      struct AlbumEntity {
53        std::string name;
54        std::string author;
55        std::string genre;
56
57        friend std::istream& operator >> (std::istream &stream,
    AlbumEntity &entity);
58      };
59   // Структура для хранения считанной из каталога записи о песне
60      struct SongEntity {
61        std::string name;
62        std::string album;
63        std::string author;
64        int duration;
65
66        friend std::istream& operator >> (std::istream &stream,
    SongEntity &entity);
67      };
68
69  private:
70      void postprocessSongs();
71
72      template<typename Entity>
73      std::vector<Entity> readEntities(std::string fileName);
74
75      void prepareGenres(std::vector<GenreEntity> entities);
76
77      void prepareAuthors(std::vector<AuthorEntity> entities);
78
79      void prepareAlbums(std::vector<AlbumEntity> entities);
80
81      void prepareSongs(std::vector<SongEntity> entities);
82
83      std::vector<Author> authors;
84      std::vector<Song> songs;
85      std::vector<Genre> genres;
86      std::vector<Album> albums;
87      std::map<Genre, std::vector<Song>> songsByGenre;
88  };
89
90  #endif // CATALOG_H
```

events.h

```
1   #ifndef EVENTS
2   #define EVENTS
3
4   #include <memory>
5   #include <iostream>
6
7   struct Event_;
8   struct QueryEvent_;
9   struct EndOfSongEvent_;
10  struct StatisticsQueryEvent_;
11  struct PlayEvent_;
12  struct StatisticsEvent_;
13
14  using Event = std::shared_ptr<Event_>;
15  using QueryEvent = std::shared_ptr<QueryEvent_>;
16  using EndOfSongEvent = std::shared_ptr<EndOfSongEvent_>;
17  using StatisticsQueryEvent =
    std::shared_ptr<StatisticsQueryEvent_>;
18  using PlayEvent = std::shared_ptr<PlayEvent_>;
19  using StatisticsEvent = std::shared_ptr<StatisticsEvent_>;
20
21  #include "radioprogram.h"
22   // Структура для хранения события на радиостанции
23  struct Event_ {
24      Event_(int timestamp);
25
26      int timestamp;
27
28      virtual void polymorphic() = 0;
29  };
30   // Структура для хранения события о запросе от слушателе на
    радиостанции
31  struct QueryEvent_ : public Event_ {
32      QueryEvent_(int timestamp, Query query);
33
34      Query query;
35
36      void polymorphic() { }
37  };
38   // Структура для хранения события о конце песни на радиостанции
39  struct EndOfSongEvent_ : public Event_ {
40      EndOfSongEvent_(int timestamp);
41
42      void polymorphic() { }
43  };
44   // Структура для хранения события для статистики на радиостанции
45  struct StatisticsEvent_ {
```

```
46  public:
47      StatisticsEvent_(int timestamp);
48
49      virtual void describe(std::ostream& stream) = 0;
50
51      int timestamp;
52  };
53   // Структура для хранения события о проигранном произведении на
    радиостанции
54  struct PlayEvent_ : public StatisticsEvent_ {
55  public:
56      PlayEvent_(int timestamp, Song song, RadioProgram program);
57
58      void describe(std::ostream& stream);
59
60  protected:
61      Song song;
62      RadioProgram program;
63  };
64   // Структура для хранения события о запросе на радиостанции
65  struct StatisticsQueryEvent_ : public StatisticsEvent_ {
66  public:
67      StatisticsQueryEvent_(int timestamp, Query query, RadioProgram
    program);
68
69      void describe(std::ostream& stream);
70
71  protected:
72      Query query;
73      RadioProgram program;
74  };
75
76  #endif // EVENTS
```

statistics.h

```
1   #ifndef STATISTICS_H
2   #define STATISTICS_H
3
4   #include <map>
5   #include <iostream>
6   #include <memory>
7
8   class Statistics_;
9
10  using Statistics = std::shared_ptr<Statistics_>;
11
12  #include "structures.h"
13  #include "types.h"
14  #include "events.h"
15   // Структура для хранения статистики о событиях на радиостанции
16  class Statistics_ {
17  public:
        // Выдать список проигранных произведений
18      const std::vector<Play>& getPlays();
19   // Выдать список запросов от слушателей по конкретной песне
20      int getSongRequestsCount(Song song);
21   // Добавить программное произведение
22      void addPlay(RadioProgram program, Song song, int timestamp);
23   // Добавить запрос от слушателей
24      void addQuery(RadioProgram program, Query query, int
    timestamp);
25   // Напечатать статистику в std::cout
26      void printStats();
27   // Выдать список всех событий
28      const std::vector<StatisticsEvent>& getEvents() {
29          return events;
30      }
31
32  private:
33      std::vector<Play> plays;
34      std::vector<StatisticsEvent> events;
35      std::map<Song, int> songRequestsCount;
36  };
37
38  #endif // STATISTICS_H
```

requestsgenerator.h

```
1   #ifndef REQUESTSGENERATOR_H
2   #define REQUESTSGENERATOR_H
3
4   #include "types.h"
5   #include "structures.h"
6   #include "catalog.h"
7   #include "events.h"
8   #include <random>
9    // Структура для генерации запросов от слушателей
10  class RequestsGenerator {
11  public:
12      RequestsGenerator(int timestamp, int duration) : mt(timestamp),
    timestamp(timestamp), duration(duration) {}
13   // Сгенерировать список запросов от слушателей
14      std::vector<Event> generate(Catalog catalog);
15
16  private:
17      Event generate_query(Catalog catalog);
18
19      int randInt(int leftBound, int rightBound);
20
21      std::mt19937 mt;
22      int timestamp;
23      int duration;
24  };
25
26  #endif // REQUESTSGENERATOR_H
```

radioprogram.h

```
1   #ifndef RADIOPROGRAM_H
2   #define RADIOPROGRAM_H
3
4   #include <memory>
5   #include <map>
6
7   class RadioProgram_;
8   class RequestsProgram_;
9   class HitParadProgram_;
10
11  using RadioProgram = std::shared_ptr<RadioProgram_>;
12  using RequestsProgram = std::shared_ptr<RequestsProgram_>;
13  using HitParadProgram = std::shared_ptr<HitParadProgram_>;
14
15  #include "structures.h"
16  #include "statistics.h"
17  #include "catalog.h"
18
19  const std::string REQUESTS_PROGRAM = "REQUESTS_PROGRAM";
20  const std::string HITPARAD_PROGRAM = "HITPARAD_PROGRAM";
21   // Структура для симуляции радиопрограммы (внутренние события)
22  class RadioProgram_ {
23  public:
24      RadioProgram_(std::string name, Genre genre, int start, int
    duration, Statistics statistics);
25   // Выбрать следующую песню
26      Song pickNextSong(Catalog catalog, int timestamp);
27   // Выдать имя программы
28      std::string getName();
29
30  protected:
31      virtual std::map<Song, double> getQualities(Catalog catalog,
    int timestamp) = 0;
32      virtual void postprocessSong(Song song, int timestamp) = 0;
33
34      std::string name;
35      Genre genre;
36      int start;
37      int duration;
38      Statistics statistics;
39  };
40
41  class RequestsProgram_ : public RadioProgram_ {
42  public:
43      RequestsProgram_(std::string name, Genre genre, int start, int
    duration, Statistics statistics);
44  // Добавить запрос от слушателя
```

```
45      void makeQuery(int timestamp, Query query) {
46          queries.push_back(query);
47      }
48
49  private: // Оценить песни по качеству
50      std::map<Song, double> getQualities(Catalog catalog, int
    timestamp);
51
52      double evaluateQuality(int timestamp, Song song);
53
54      void postprocessSong(Song song, int timestamp);
55
56      std::vector<Query> queries;
57  };
58
59  class HitParadProgram_ : public RadioProgram_ {
60  public:
61      HitParadProgram_(std::string name, Genre genre, int start, int
    duration, Statistics statistics);
62
63  private:
64      std::map<Song, double> getQualities(Catalog catalog, int
    timestamp);
65
66      void postprocessSong(Song song, int timestamp) { }
67
68      double evaluateQuality(int timestamp, Song song);
69
70      void postprocessSong() { }
71  };
72
73  #endif // RADIOPROGRAM_H
```

simulator.h

```
1   #ifndef SIMULATOR_H
2   #define SIMULATOR_H
3
4   #include <memory>
5   #include <set>
6
7   class RadioProgramSimulator_;
8   class HitParadProgramSimulator_;
9   class RequestsProgramSimulator_;
10  class Simulator_;
11
12  using RadioProgramSimulator =
    std::shared_ptr<RadioProgramSimulator_>;
13  using HitParadProgramSimulator =
    std::shared_ptr<HitParadProgramSimulator_>;
14  using RequestsProgramSimulator =
    std::shared_ptr<RequestsProgramSimulator_>;
15  using Simulator = std::shared_ptr<Simulator_>;
16
17  #include "statistics.h"
18  #include "radioprogram.h"
19  #include "requestsgenerator.h"
20  #include "events.h"
21   // Структура для симуляции радиопрограммы (внешняя часть)
22  class RadioProgramSimulator_ {
23  public: // Структура для сравнения событий по времени
24      struct EventComparator {
25          bool operator () (const Event &a, const Event &b);
26      };
27
28      using Queue = std::set<Event, EventComparator>;
29
30      RadioProgramSimulator_(std::string name, Genre genre, int
    duration, int timestamp, Statistics statistics);
31   // Симулировать радиопрограмму
32      int simulate(Catalog catalog);
33   // Создать изначальные события (окончание песни в начале и запросы
    (?) от пользователей)
34      virtual std::vector<Event> buildInitialEvents(Catalog catalog)
    = 0; // Обрабовать событие на радиопрограмме
35      virtual void process(Queue &events, Event event, Catalog
    catalog) = 0;
36
37  protected:
38      std::string name;
39      Genre genre;
40      int duration;
41      int timestamp;
42      Statistics statistics;
```

```
43  };
44   // Симулятор хитпарадов
45  class HitParadProgramSimulator_ : public RadioProgramSimulator_ {
46  public:
47      HitParadProgramSimulator_(std::string name, Genre genre, int
    duration, int timestamp, Statistics statistics);
48
49      std::vector<Event> buildInitialEvents(Catalog catalog);
50
51      void process(Queue &events, Event event, Catalog catalog);
52   // Обработать событие окончания песни
53      void endOfSongEvent(Queue &events, EndOfSongEvent event,
    Catalog catalog);
54  private:
55      HitParadProgram program;
56  };
57   // Симулятор программы по заявкам
58  class RequestsProgramSimulator_ : public RadioProgramSimulator_ {
59  public:
60      RequestsProgramSimulator_(std::string name, Genre genre, int
    duration, int timestamp, Statistics statistics);
61
62      std::vector<Event> buildInitialEvents(Catalog catalog);
63
64      void process(Queue &events, Event event, Catalog catalog);
65   // Обработать событие запроса от слушателя
66      void query(Queue &events, QueryEvent event, Catalog catalog);
67   // Обработать событие окончания песни
68      void endOfSongEvent(Queue &events, EndOfSongEvent event,
    Catalog catalog);
69
70  private:
71      RequestsProgram program;
72  };
73   // Симулятор радиостанции
74  class Simulator_ {
75  public:
76      Simulator_(std::string fileName);
77   // Симулировать расписание
78      Statistics simulate(Catalog catalog);
79
80      std::vector<std::tuple<std::string, std::string, std::string,
    int>> entities;
81  };
82
83  #endif // SIMULATOR_H
```

Инструментальные средства

Для выполнения работы использовался компилятор g++-6 (Homebrew GCC 6.3.0_1 --without-multilib) 6.3.0 и библиотека Qt-5.3.0.

Файловая структура системы

- albumdescriber.{h,cpp,ui}: форма для показа расширенной информации об альбоме
- authordescriber.{h,cpp,ui}: форма для показа расширенной информации об исполнителе
- songdescriber.{h,cpp,ui}: форма для показа расширенной информации о песне
- genredescriber.{h,cpp,ui}: форма для показа расширенной информации о жанре
- catalog.{h,cpp}: класс Catalog_ для хранения информации о каталоге
- structures.{h,cpp}: классы для хранения информации об альбомах, исполнителях, событиях и т.д.
- events.{h,cpp}: классы для хранения информации о статистических событиях
- requestgenerator.{h,cpp}: класс для генерации запросов от пользователя
- statistics.{h,cpp}: класс для хранения информации о статистике
- radioprogram.{h,cpp}: класс для симуляции внутренних событий на радиостанции
- simulator.{h,cpp}: класс для симуляции внешних событий на радиопрограммах
- mainwindow.{h,cpp,ui}: форма главного окна
- timetableeditorform.{h,cpp,ui}: форма для редактирования расписания
- main.cpp: запуск программы

Диаграмма наследований