МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный исследовательский университет)

Институт №8

«Компьютерные науки и прикладная математика»

Кафедра 806

«Вычислительная математика и программирование»

Курсовой проект по дисциплине «Фундаментальные алгоритмы»

Тема: «Разработка алгоритмов системы хранения и управления данными на основе динамических структур данных»

Студент: Мингазова Д.И.

Группа: М8О-211Б-21

Преподаватель: Ирбитский И.С.

Оценка:

Дата:

Москва, 2023

Содержание

Задание……..……..…….3

Аллокаторы…….…..……4

Логгер…….…….……5

Структура данных……..…….……6

База данных……..……..…….……7

Документация……..……...…..……..8

Руководство пользователя……..…….…….13

Вывод……...…….…..17

Приложение……...……....18

Источники……..22

Задание

0. На языке программирования C++ (стандарт C++14 и выше) реализовать приложение, позволяющее выполнять операции над коллекциями данных заданных типов и контекстами их хранения (коллекциями данных).

Коллекция данных описывается набором строковых параметров:

- название пула схем данных, хранящего схемы данных;
- название схемы данных, хранящей коллекции данных;
- название коллекции данных.

Коллекция данных представляет собой ассоциативный контейнер (красно-черное дерево), в котором каждый объект данных соответствует некоторому уникальному ключу. Взаимодействие с коллекцией объектов происходит посредством выполнения одной из операций над ней:

- добавление новой записи по ключу;
- чтение записи по её ключу;
- чтение набора записей с ключами из диапазона [$minbound$... $maxbound$];
- обновление данных для записи по ключу;
- удаление существующей записи по ключу.

Во время работы приложения возможно выполнение также следующих операций:

- добавление/удаление пулов данных;
- добавление/удаление схем данных для заданного пула данных;
- добавление/удаление коллекций данных для заданной схемы данных заданного пуладанных.

1. Реализовать интерактивный диалог с пользователем. Пользователь при этом может вводить конкретные команды и подавать на вход файлы с потоком команд.
2. Реализовать возможность кастомизации аллокатора для размещения объектов данных: первый + лучший + худший подходящий освобождение с дескрипторами границ.
3. Тип данных: данные игрока MMO (id пользователя, никнейм, игровая зона (строка), статус (обычный игрок/премиум игрок/модератор чата/администратор), значение внутриигровой валюты, значение внутриигровой премиальной валюты, количество очков опыта, дата регистрации, время проведенное в игре (в минутах)).

Аллокаторы

Аллокатор или распределитель памяти в языке программирования C++ — специализированный класс, реализующий и инкапсулирующий малозначимые (с прикладной точки зрения) детали распределения и освобождения ресурсов компьютерной памяти.

Способ выделения зависит от аллокатора, но методы поиска подходящего для выделения блока памяти одинаковы:

- Метод первого подходящего - поиск первого блока памяти, размер которого удовлетворяет размеру запрашиваемой памяти
- Метод лучшего подходящего - поиск блока памяти, размер которого наибольший среди всех блоков, удовлетворяющих размеру запрашиваемой памяти
- Метод худшего подходящего - поиск блока памяти, размер которого наименьший среди всех блоков, удовлетворяющих размеру запрашиваемой памяти

Каждый класс аллокатора представляет собой реализацию интерфейса класса memory:

Листинг 1. Класс memory.

```
class memory{
public:
    virtual void* allocate(size_t target_size) const = 0;

    virtual void deallocate(void* result) const = 0;

    virtual ~memory(){

    }
};
```

Помимо выделения из глобальной кучи («allocator_2.h» в разделе Приложение) для реализации приложения был использован аллокатор с освобождением с дескрипторами границ, который реализован по определенному алгоритму, описанному в [1].

Логгер

В реализации логгера был использован порождающий паттерн проектирования Builder. Паттерн Builder отделяет алгоритм поэтапного конструирования сложного продукта (объекта) от его внешнего представления так, что с помощью одного и того же алгоритма можно получать разные представления этого продукта.

Поэтапное создание продукта означает его построение по частям. После того как построена последняя часть, продукт можно использовать.

Логгер конфигурируется двумя командами класса logger_builder:

- Add_stream - принимает на вход название файлового потока вывода и значение уровня логирования

- Construct - отдает указатель на сконфигурировнный логгер на внешний уровень

Класс логгера logger содержит коллекцию потоков вывода, ключом которой является название потока, а значением - пара из указателя на открытый поток и количества логгеров, владеющих потоком, перечисление уровней логирования, а также метод log, принимающий на вход 2 аргумента - название потока и уровень логирования.

Листинг 2. Класс logger_builder.

```
class logger_builder{

public:

    virtual ~logger_builder(){}


    virtual logger_builder* add_stream(std::string const & , logger::severity)=0;


    virtual logger* construct() const = 0;


    virtual logger_builder* add_in_file(const std::string& way) = 0;

};
```

Структура данных

Хранение коллекций данных в приложении сделано на базе структуры данных “красно-чёрное дерево”, которая является производной класса от класса бинарного дерева поиска, которое в свою очередь наследуется от абстрактного класса associative_container (associative_container.h представлен в разделе Приложение)

Двоичное дерево поиска — двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- оба поддерева — левое и правое — являются двоичными деревьями поиска;
- у всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X;
- у всех узлов правого поддерева произвольного узла X значения ключей данных больше, нежели значение ключа данных самого узла X.

Красно-чёрное дерево — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный».

Принцип организации:

1. Узел может быть либо красным, либо чёрным и имеет двух потомков;
2. Корень — как правило чёрный.
3. Все листья, не содержащие данных — чёрные.
4. Оба потомка каждого красного узла — чёрные.
5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.

Случаи удаления узлов из красно-черного дерева были реализованы на основе [2] и [3].

База данных

База данных реализована как красно-черное дерево с указателями на другие красно-черные деревья (пулы), которые в свою очередь хранят на красно-черные деревья (схемы), в которых хранятся указатели на абстрактный класс associative_container (для дальнейшей конфигурации типа дерева для хранения информации об игроке MMO):

Листинг 3. Объявление основного красно-черного дерева _database.

```
      red_black_tree<std::string, red_black_tree<std::string,
red_black_tree<std::string, associative_container<player_key, player_value>*,
comparer_string>*, comparer_string>*, comparer_string>* _database;
```

У класса database два публичных метода:

- run_file_commands(file_path) - считывание команд из файла и выполнение их последовательно.
- dialog() - открывает интерактивный диалог с пользователем. В диалоге предусмотрены все невалидные действия со стороны пользователя, а также обрабатываются все ошибки при работе с приложением.

Документация

В файле instruction.txt приведены описания команд и формат их ввода:

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
----------------------INSERT-----------------------------


insert: [pool_name] [scheme_name] [collection_name]
    press enter
key: [id] [game_zone]
    ~press enter
value: [nickname] [status] [valute] [premium_valute] [experience] [date_of_registration] [time_in_game]
    ~press enter
----------------------INFO-------------------------------
Before inserting add pool, scheme and collection.
Types:
    id -> int
    status -> can onle be: simple/premium/moderator/admin

-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------------------READ_KEY-----------------------------

read key: [pool_name] [scheme_name] [collection_name]
    ~press enter
key: [id] [game_zone]
    ~press enter

----------------------INFO-------------------------------

Before reading key, there must be pool, scheme and collection with key/keys.

-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

----------------------READ_RANGE-----------------------------

read range: [pool_name] [scheme_name] [collection_name]

~press enter

[first_player_id]

~press enter

[second_player_id]

~press enter

----------------------INFO-------------------------------

Reading value from first key until second one.

If you don't enter first key, range will begin with first key in scheme.

If you enter second key, range will go until lastest key in scheme.

>>

----------------------UPDATE_KEY-----------------------------

update key: [pool_name] [scheme_name] [collection_name]

~press enter

key: [id] [game_zone]

~press enter

value: [nickname] [status] [valute] [premium_valute] [experience] [date_of_registration] [time_in_game]

~press enter

----------------------INFO-------------------------------

Make sure there is a key which you want to update.

>>

----------------------REMOVE-----------------------------

```
insert: [pool_name] [scheme_name] [collection_name]
    ~press enter
key: [id] [game_zone]
    ~press enter

----------------------INFO-------------------------------

Make sure there is a key which you want to remove.

-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------------------ADD_POOL-----------------------------

add pool: [pool_name] { [allocator_name] [size(optional)] [allocator_mode(optional)] }
    ~press enter

----------------------INFO-------------------------------

Accessible allocator names:
                -simple -> doesn't need size and allocator_mode
                -border_descriptor -> need size and one of allocator_mode: first, best, worst, none(nullptr)

-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------------------DELETE_POOL-----------------------------

delete pool: [pool_name]
    ~press enter

----------------------INFO-------------------------------

Make sure there is a pool which you want to delete.
```

```
-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------------------ADD_SCHEME-----------------------------

add scheme: [pool_name] [scheme_name]
    ~press enter

----------------------INFO-------------------------------

Make sure there is a pool which you want to add scheme in.

-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------------------DELETE_SCHEME-----------------------------

delete scheme: [pool_name] [scheme_name]
    ~press enter

----------------------INFO-------------------------------

Make sure there are pool scheme and  which you want to delete.

-----------------------------------------------------------

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

----------------------ADD_COLLECTION-----------------------------

add collection: [pool_name] [scheme_name] [collection_name] {tree_type}
    ~press enter

----------------------INFO-------------------------------
```

Make sure there are pool and scheme which you want to add collection in.

Types of tree: red black tree.

>>

----------------------DELETE_COLLECTION-----------------------------

delete collection: [pool_name] [scheme_name] [collection_name]

    ~press enter

----------------------INFO-------------------------------

Make sure there are pool, scheme and collection which you want to delete.

>>

----------------------RESET_DATABASE-----------------------------

reset database:

    ~press enter

----------------------INFO-------------------------------

Deleting all pools, schemes and collections.

Руководство пользователя

Продемонстрируем работу и взаимодействие с приложением. Для начала выполним список команд из файла todo.txt:

database > todo.txt

```
add pool: [firstpool] { [border_descriptor] [200000] [best] }
add scheme: [firstpool] [firstscheme]
add collection: [firstpool] [firstscheme] [firstcollection] {red black tree}
insert: [firstpool] [firstscheme] [firstcollection]
key: [123] [europe]
value: [dinka] [moderator] [2345] [4567] [34567] [20.03.19] [45]
read key: [firstpool] [firstscheme] [firstcollection]
key: [123] [europe]
update key: [firstpool] [firstscheme] [firstcollection]
key: [123] [europe]
value: [diners] [moderator] [2345] [34] [4567] [20.03.19] [300]
insert: [firstpool] [firstscheme] [firstcollection]
key: [125] [europe]
value: [roman] [admin] [2345] [4567] [34567] [20.03.19] [45]
read key: [firstpool] [firstscheme] [firstcollection]
key: [125] [europe]
read range: [firstpool] [firstscheme] [firstcollection]
{key: [123] [europe]}
{key: [125] [europe]}
delete pool: [firstpool]
```

В main.cpp создается указатель на объект logger типа logger, а также база данных my_db. У my_db вызвывается диалог dialog(), который является интерфейсом для общения с пользователем. После работы программы чистится память: удаляется база данных, логгер и билдер, который создавался для создания логгера.

Листинг 4. Main.cpp

```
int main(int argc, char* argv[]){

    auto* builder = new logger_builder_concrete();

    logger* logger = builder->add_stream("console", logger::severity::information)-
>construct();


    auto* my_db = new database(logger);


    try{
```

```
        my_db->dialog();
    } catch(const std::logic_error& ex){
        std::cout << ex.what() << std::endl;
    }

    delete my_db;
    delete logger;
    delete builder;
}
```

Вывод приложения:

```
[warning][02.09.2023 04:26:26]

-------- Welcome! I'm doing my best! --------


[warning][02.09.2023 04:26:26]
----------------------------------------

 ~~~~~Choose an action ~~~~~

1) Run commands in a file
2) Enter a command manually
3) Reset database
4) Exit

~~~~~ Enter:
1
[warning][02.09.2023 04:27:03]
----------------------------------------

~~~~~ Enter a file's path ~~~~~

~~~~~ Path:
C:\dev\COURSEWORK\database\todo.txt
[warning][02.09.2023 04:27:05] ~~~~Parsing file << todo.txt>> ...
[information][02.09.2023 04:27:05] Pool was added
[warning][02.09.2023 04:27:05] ***** Pool firstpool was added succesfully!
[information][02.09.2023 04:27:05] Scheme was added.
[warning][02.09.2023 04:27:05] ***** Scheme 'firstschemewas added in firstpool!
[information][02.09.2023 04:27:05] Collection was added.
[warning][02.09.2023 04:27:05] ***** Collection firstcollection was added in pool firstpool in scheme firstscheme!
[information][02.09.2023 04:27:05] Done inserting.
[warning][02.09.2023 04:27:05]  ~~Insert finished!
--key: [id: 123, game zone: europe]
```

Рисунок 1. Логи выполнения команд из файла. 1 часть.

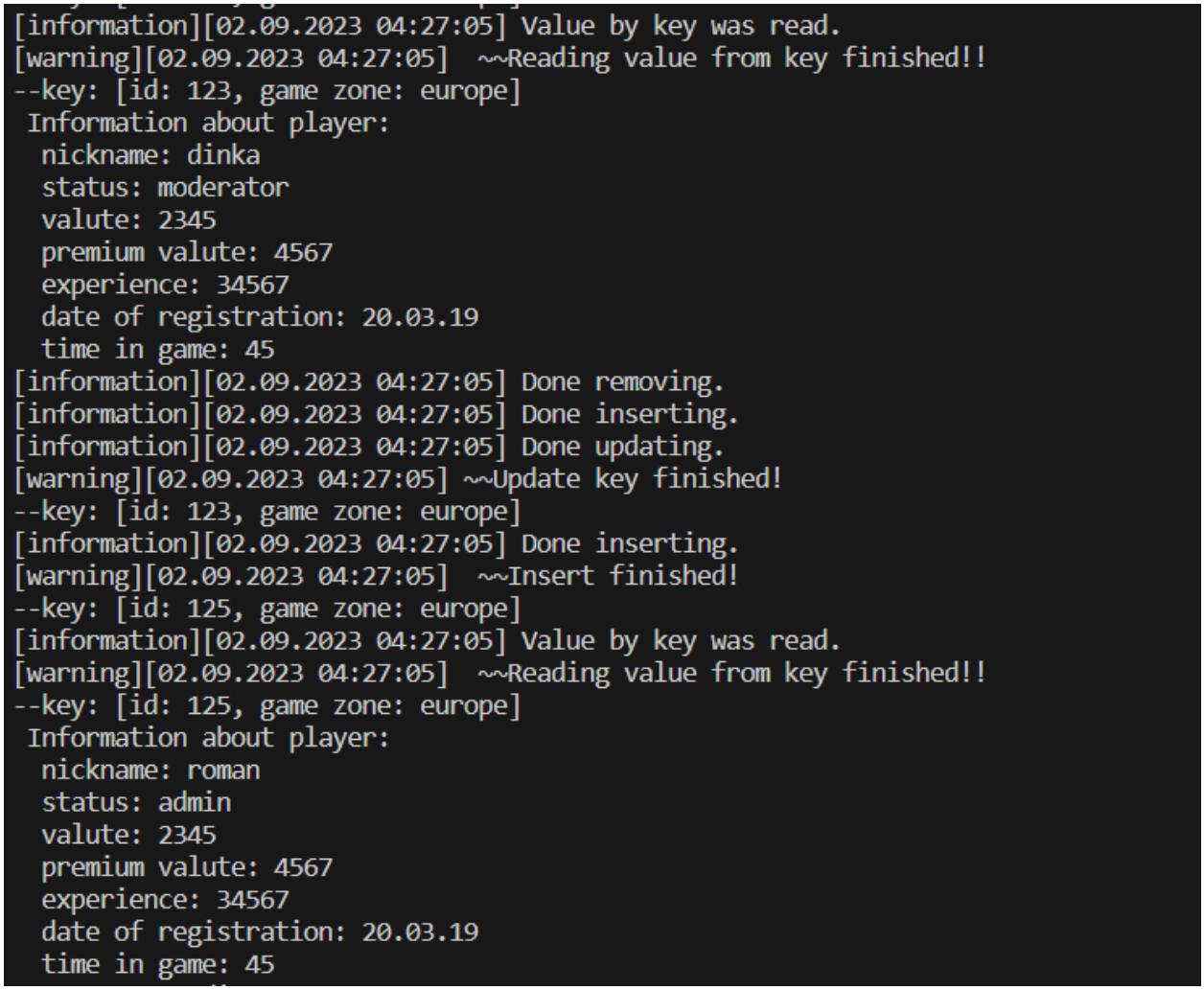

Рисунок 2. Логи выполнения команд из файла. 2 часть.

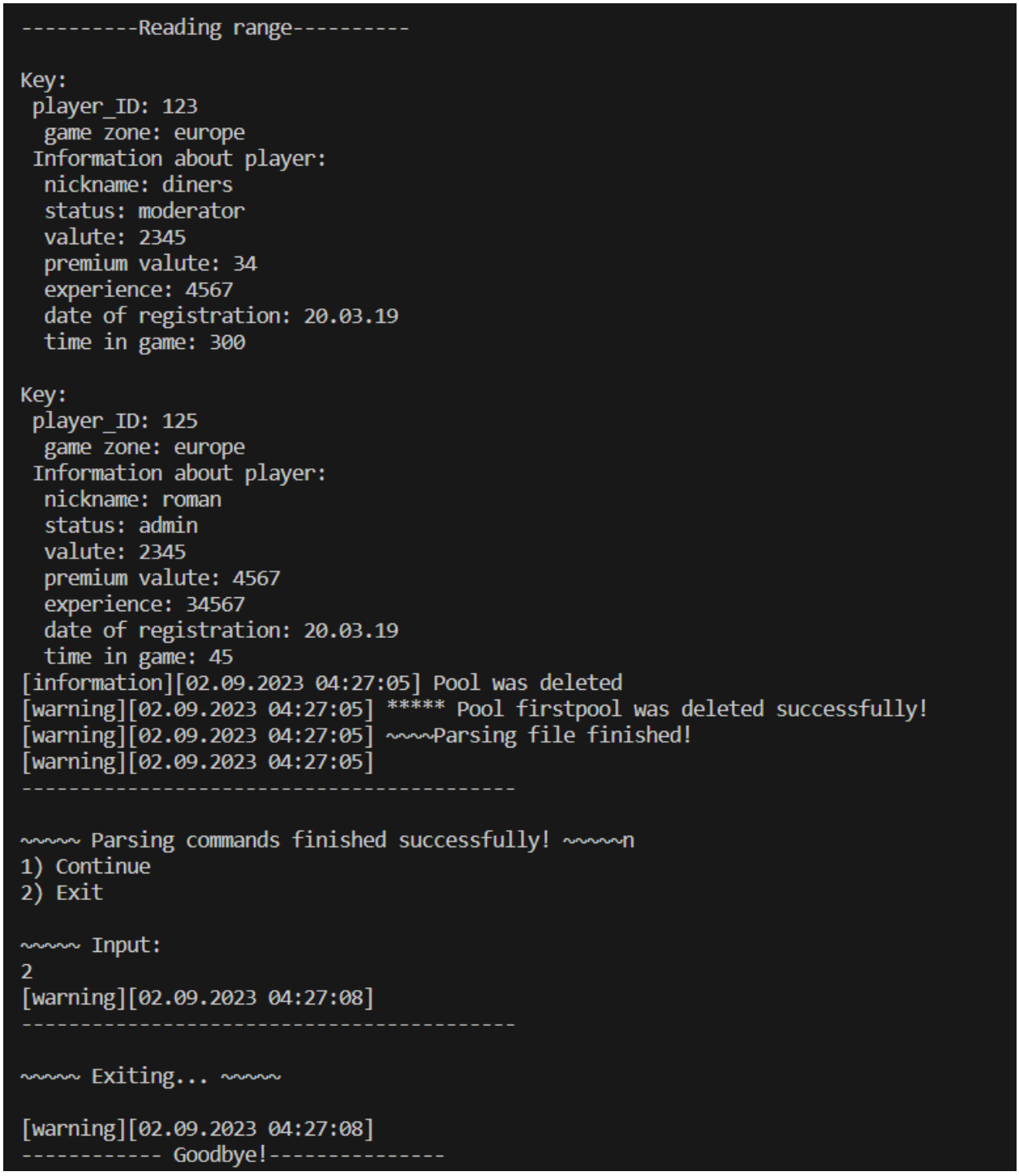

Рисунок 3. Логи выполнения команд из файла. 3 часть.

Вывод

В ходе курсового проекта было разработано приложение с алгоритмами систем управления данными на основе динамических структур данных. В процессе реализации также написаны компоненты приложения, такие как ассоциативные контейнеры, логгер, аллокаторы. Создание данного приложения требует знаний об устройстве баз данных, управлении памяти и ресурсами, а также знаний о распределении памяти, стандартах языка C++ и сбалансированных деревьях.

Приложение

Весь код приложения и его компоненты доступны в репозитории на GitHub:

https://github.com/midinka/Coursework

Контакты для обращений касаемо курсового проекта: dinam03777@gmail.com

«allocator_2.h»

```
class allocator final:public memory
{
private:
    logger *_log;
public:
    allocator(logger *log = nullptr): _log(log) {}
    void* allocate(size_t target_size) const override{
        auto * result =::operator new(sizeof(size_t) + target_size);
        *reinterpret_cast<size_t *>(result) = target_size;
        std::ostringstream oss;
        oss<<result;
        if (_log != nullptr)
        {
            _log->log(oss.str(), logger::severity::debug);
        }
        return reinterpret_cast<void *>(reinterpret_cast<size_t *>(result) + 1);
    }
    void deallocate(void* result) const override {
        result = reinterpret_cast<size_t *>(result) - 1;
        auto memory_size = *reinterpret_cast<size_t *>(result);
        :: operator delete(result);
    }
};
```

«associative_container.h»

```
template<
        typename tkey,
        typename tvalue>
class associative_container
{
public:
    enum class bypass_detour{
        prefix,
        postfix,
        infix
    };
    struct key_value_pair
    {
        tkey _key;
        tvalue _value;
    };
public:
    virtual void insert(
            const tkey &key,
            const tvalue &value) = 0;
    void operator+=(
            key_value_pair pair);
    virtual bool find(
            key_value_pair *target_key_and_result_value) = 0;
    bool operator[](
            key_value_pair *target_key_and_result_value);
    virtual tvalue remove(
            const tkey &key) = 0;

    virtual const tvalue& get(
```

```
            const tkey &key) const = 0;
    tvalue operator-=(
            key_value_pair pair);
    tvalue operator-=(
            tkey const &key);
    virtual void bypass_tree(bypass_detour detour) const = 0;
    virtual ~associative_container() = default;
};

template<
        typename tkey,
        typename tvalue>
void associative_container<tkey, tvalue>::operator+=(
        key_value_pair pair)
{
    return insert(pair._key, std::move(pair._value));
}

template<
        typename tkey,
        typename tvalue>
bool associative_container<tkey, tvalue>::operator[](
        key_value_pair *target_key_and_result_value)
{
    return find(target_key_and_result_value);
}

template<
        typename tkey,
        typename tvalue>
tvalue associative_container<tkey, tvalue>::operator-=(
        key_value_pair pair)
```

```
{
    return remove(pair._key);
}

template<
        typename tkey,
        typename tvalue>
tvalue associative_container<tkey, tvalue>::operator-=(
        tkey const &key)
{
    return remove(key);
}

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Источники

1. Д. Э. Кнут. “Искусство программирования”. Том 1.

2. Томас Кормен, Чарльз Эрик Лейзерсон, Рональд Линн Ривест, Клиффорд Штайн “Алгоритмы: построение и анализ”

3. https://habr.com/ru/companies/otus/articles/521034/