

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: Тарабукин М.И.

Москва, 2023

## Содержание

Введение.....	3
Описание реализованного решения.....	4
Руководство пользователя.....	9
Вывод.....	14
Приложение.....	15

## Введение

В рамках курсовой работы нужно на языке программирования C++ (стандарт C++14 и выше) реализовать приложение, позволяющее выполнять операции над коллекциями данных и контекстами их хранения (коллекциями данных) типа:

Данные о доставке (id пользователя, id доставки, описание доставки, ФИО пользователя (раздельные поля), адрес электронной почты пользователя, номер телефона пользователя, адрес доставки (в виде строки), комментарий пользователя, дата/время доставки).

Ключами являются поля id пользователя и id доставки.

Коллекция данных описывается набором строковых параметров (набор параметров однозначно идентифицирует коллекцию данных):

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

Коллекция данных представляет собой ассоциативный контейнер (АВЛ-дерево), в котором каждый объект данных соответствует некоторому уникальному ключу. Для ассоциативного контейнера необходимо вынести интерфейсную часть (в виде абстрактного класса C++) и реализовать этот интерфейс. 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 пула данных.

Поток команд, выполняемых в рамках работы приложения, поступает из файла, путь к которому подаётся в качестве аргумента командной строки.

Реализовать возможность кастомизации (для заданного пула схем) аллокаторов для размещения объектов данных первый + лучший + худший подходящий + освобождение в рассортированном списке, первый + лучший + худший подходящий + освобождение с дескрипторами границ.

Реализовать функционал приложения в виде сервера, запросы на который поступают из клиентских приложений. При этом взаимодействие клиентских приложений с серверным должно быть реализовано посредством IPC: Unix message queues.

Реализовать серверное приложение, собирающее логи клиентской (и серверной) части приложения в файловые потоки вывода. Конфигурирование серверного логгера обеспечить на основе файла со структурой JSON.

## Описание реализованного решения

Реализованное решение основывается на использовании стандартных средств разработки на языке программирования C++, использовании внешней библиотеки, алгоритмов и структур данных, а также средств межпроцессного взаимодействия.

Для хранения данных были использованы STL контейнеры: `std::string`, `std::stack`, `std::vector`, которые позволяют хранить широкий спектр типов данных. В дополнение использовано динамическое выделение памяти.

Для взаимодействия с JSON-файлами использована внешняя библиотека `json.hpp`, осуществляющая чтение файлов.

Для реализации архитектуры были использованы классы, структуры, их наследование и переопределение методов, были реализованы интерфейсы взаимодействия с классами.

Взаимодействие клиента и сервера было реализовано при помощи IPC Unix queue messages.

Серверное приложение получает сообщение от клиента посредством очереди сообщений и обрабатывает запросы клиента, после чего отправляет ответ обратно пользователю.

Клиентское приложение получает на вход текстовый файл с командами и отправляет запрос серверу.

Существует отдельное приложение, которое собирает логи клиента и сервера, если запущено, и отправляет их на файловые потоки вывода.

Сервер содержит базу данных типа database, в которой можно создавать пулы pools, в которых можно создавать схемы schemes, в которых можно создавать коллекции collections, в которых можно создавать записи notes. Пулы, схемы, коллекции хранятся в виде элементов AVL-деревьев.

Основанное на шаблонах AVL-дерево представляет собой сбалансированное бинарное дерево, позволяющее осуществлять создание, хранение, перезапись, поиск, удаление, получение его элементов. Элементами дерева являются пары

ключ-значение. Дерево обладает итераторами, реализованными в виде подклассов класса дерева, для его обходов: префиксного, инфиксного, постфиксного.

Существует класс логгер для логирования хода выполнения приложения. При наличии сервера, собирающего логи, логирование происходит на нём. Иначе локально у сервера и у клиента. Потоки вывода логгера определяются конфигурационным файлом. Логгер создаётся при помощи порождающего паттерна “строитель”, позволяющий удобно кастомизировать потоки вывода логгера в зависимости от жёсткости логируемого сообщения.

Динамически выделенная память берётся из глобальной кучи и может управляться при помощи кастомных аллокаторов двух типов, каждый из которых обладает разными видами аллокации ячеек памяти. В динамической памяти хранится база данных и ее элементы.

Интерфейс ассоциативного контейнера позволяет добавлять, удалять, перезаписывать, получать, печатать элементы структуры данных наследованной от него.

Команды, полученные от клиента, проходят валидацию и, в случае успеха, выполняются сервером.

Клиентское приложение получает на вход файл в виде аргумента командной строки. Осуществляется попытка открыть файл и в случае успеха его содержимое переписывается в буфер типа `char[]` структуры, содержащей помимо него тип сообщения.

Создается псевдорандомный ключ-токен для идентификации двух определенных очередей сообщений при помощи функции `ftok`, осуществляется попытка подключения к очереди сообщения для отправки структуры с командами посредством функции `msgget`. Если очередь не существует, она создаётся. Сообщение добавляется в очередь сообщений, используя `msgsnd`. После клиентское приложение ждёт ответного сообщения от сервера по другой очереди сообщений. Когда ответ приходит, сообщение выводится в консоль.

Сервер в начале своей работы также подключается к двум очередям сообщений, создаёт в динамической памяти базу данных, после чего ждёт сообщений, содержащих команды.

Когда сообщение появляется в очереди сообщения, команды из буфера копируются в объект типа `std::stringstream in_stream`, передаваемый в функцию для обработки запросов `process_file`. Второй объект типа `std::stringstream out_stream` также передаётся по ссылке в функцию для получения вывода.

В функции `process_file` происходит пословная обработка потока команд. В случае несоответствия команд формату, в поток `out_stream` записывается сообщение об ошибке и производится попытка интерпретации последующих команд.

Список возможных действий с базой данных:

1. Создание пула/схемы/коллекции/записи.
2. Удаление пула/схемы/коллекции/записи.
3. Перезапись пула/схемы/коллекции/записи.
4. Чтение записи.
5. Чтение записей из заданного диапазона.

Выполнение команд делегируется последовательно в следующем порядке: база данных->пул->схема->коллекция. Глубина делегирования зависит от структуры, над которой производится действие.

В случае несуществования структуры, некорректности названия структуры в поток вывода добавляется сообщение о невозможности выполнения команды.

Во время создания каждой структуры создаётся ассоциативный контейнер структур, находящихся ниже по иерархии база\_данных->пул->схема->коллекция->запись.

Создание пула требует определения типа аллокатора, который будет его хранить, как и все структуры ниже его по иерархии.

Выделение динамической памяти производится при помощи метода `safe_allocate()`, который проверяет наличие кастомного аллокатора и выделяет память в нем, при его отсутствии – в глобальной куче. Деаллокация динамической памяти производится методом `safe_deallocate()` который работает по этому же принципу, возвращая память либо в аллокатор, либо в кучу.

При создании записи происходит валидация не только ключа, но и значения. Значение ключа записи представляет собой пару значений беззнакового целого числа { `id` пользователя, `id` доставки }. Значение представляет собой структуру состоящую из объектов строк, за исключением телефонного номера, являющегося беззнаковым целым числом:

1. Описание доставки
2. Имя
3. Фамилия
4. Отчество
5. Адрес электронной почты
6. Номер телефона
7. Адрес доставки
8. Комментарий
9. Дата и время доставки

При вставке элемента в ассоциативный контейнер АВЛ-дерево осуществляется двоичный поиск по дереву, используя определенный для типа ключа функтор компаратор. В случае нахождения места для вставки выделяется память для элемента. Новый элемент инициализируется стандартными значениями и имеет структуру:

1. Ключ
2. Значение
3. Указатель на левое поддерево
4. Указатель на правое поддерево

Само дерево содержит указатель только на его корень `_root`.

Во время поиска места для вставки в дерево в стек сохраняются указатели на указатели на элементы дерева для быстрого доступа к родительским элементам и обратного прохода к корню.

При создании элемента указатель на дочерний элемент родительского элемента обновляется. Заметим, что при вставке нового элемента в дерево, этот элемент всегда является листом, то есть элементом, не имеющим дочерних элементов.

После вставки элемента в дерево начинается прохождение по обратному пути от элемента до корня дерева. Для каждого элемента на пути производится переоценка его высоты методом `fix_height()`. Высота определяется путём нахождения максимума из дочерних узлов и добавлением единицы. Все листья имеют высоту равную единице.

Кроме переоценки высоты производится проверка сбалансированности дерева методом `balance_factor()`, который вычитает из высоты левого поддерева высоту правого поддерева. Если фактор сбалансированности равен двойке по модулю, то дерево несбалансированно, поскольку высота одного из поддеревьев на две единицы выше.

Для возвращения сбалансированности дереву выполняется поворот поддерева. Существует четыре типа поворотов: большой левый, малый левый, большой правый, малый правый. Поворот осуществляется в левую сторону, если высота правого поддерева больше, иначе в правую. Большой поворот состоит из двух малых поворотов в разные стороны. Если высота правого поддерева больше, а также высота левого поддерева правого поддерева больше, чем высота правого поддерева правого поддерева, то АВЛ-дереву необходим большой левый поворот, а если высота левого поддерева правого поддерева меньше или равна высоте правого поддерева правого поддерева, то нужен малый левый поворот.

Во время поворота происходит переопределение указателей и для удобства замена текущего элемента на элемент, который окажется на его месте по завершении поворота.

При перезаписи элемента выполняется тот же алгоритм, за исключением того, что двоичный поиск происходит до нужного элемента. Выделяется память под новый элемент, указатели на дочерние элементы инициализируются указателями старого элемента, указатель родительского элемента также перенаправляется на новый элемент.

На этом этапе старый элемент перестаёт быть нужным и поэтому вызывается его деструктор, также происходит освобождение занимаемой им памяти. Старый элемент успешно заменён новым.

Для удаления элемента мы его находим двоичным поиском. На этом моменте может возникнуть три типа ситуаций.

Если удаляемый элемент имеет два дочерних элемента, находим максимальный элемент его левого поддерева при помощи цикла `while`, после чего копируем ключ на место старого ключа и переносим значение в старый элемент. После этого максимальный элемент левого поддерева старого элемента полностью симитирован в удаляемый элемент, и мы можем его удалить.

Теперь необходимо пройти по элементам, от родителя бывшего максимального элемента левого поддерева до корня дерева обновляя значения высот встречающихся элементов и выполняя балансировку дерева.

Если элемент является листом, то достаточно просто его удалить, также пройдясь до корня после.

Если имеется один дочерний элемент, то достаточно указатель родительского элемента на удаляемый заменить на единственный дочерний элемент удаляемого, и пройти до корня после.

Чтобы вывести элемент, нужно всего лишь найти его, после чего передать значение наверх, как в методе `read`, либо сохранив указатель на него в переменную, как в методе `find`.

Для вывода записей из диапазона создадим инфиксный итератор, указывающий на первый элемент, больший или равный левой границе диапазона. Выводим элементы в поток вывода до тех пор, пока ключи не больше правой границы диапазона, увеличивая значение итератора после каждого вывода элемента.



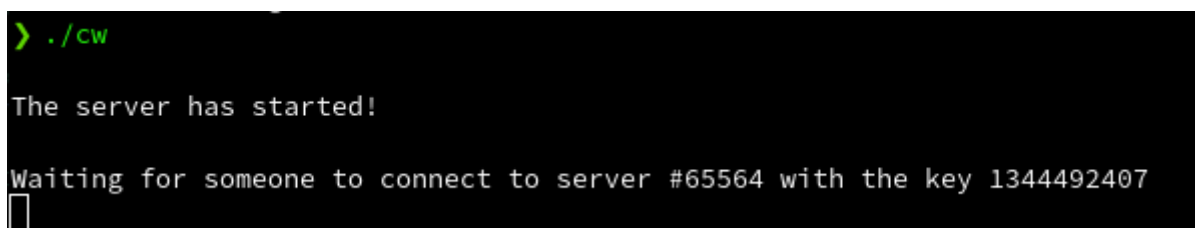
Команды от клиента выполняются, пока поток ввода не завершится, после чего сервер отправляет буфер, полученный из `out_stream` клиенту. Приложение клиента, получив сообщение, выводит его на экран консоли.

Во время выполнения программ сервер и клиент создают логи. При помощи метода `safe_log()` выполняется проверка активности сервера по сбору логов. Если он работает, то сообщение с логом и его строгостью отправляется на сервер-логгер при помощи функции `msgsnd` с флагом `NO_WAIT`, чтобы не ждать отчета об успешности отправки во избежание замедления работы приложения.

Сервер-логгер создает объект логгер при помощи объекта `builder`, получая конфигурацию из файла `JSON`.

## Руководство пользователя

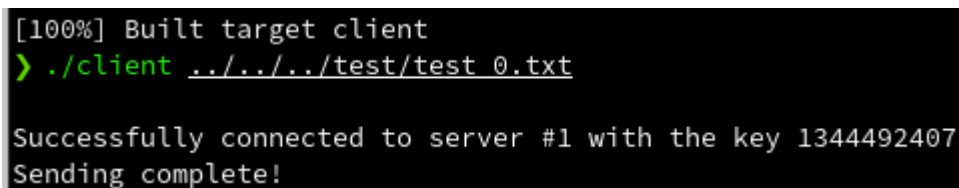
Для запуска приложения нужно собрать исполняемый файл, выполнив команды `“cmake –init .”` и `“cmake –build .”`, находясь в директории `build`. Теперь можно запустить исполняемый файл командой `“./cw”`



```
> ./cw  
The server has started!  
Waiting for someone to connect to server #65564 with the key 1344492407  
█
```

Рис.1 Запуск сервера

Сервер перешёл в режим ожидания сообщений. Теперь откроем директорию `source/client_app/build` в другой консоли, скомпилируем и запустим клиентское приложение командами `“cmake init .”`, `“cmake –build .”` и `“../.././test/test_0.txt”`. В аргумент нужно передать файл с командами, некоторые тестовые файлы хранятся в директории `test`.



```
[100%] Built target client  
> ./client ../.././test/test_0.txt  
Successfully connected to server #1 with the key 1344492407  
Sending complete!
```

Рис.2 Запуск клиентского приложения

Приложение отправило файл серверу и ждёт ответа.

Синтаксис файлов с командами содержит команды:

1. create        –        создание
2. delete       –        удаление
3. read         –        чтение одной записи
4. read\_range –        чтение записей из диапазона

Структура note состоит из ключа и значения. Ключ состоит из двух целых беззнаковых чисел, обрамлённых фигурными скобками и разделённые запятой. Фигурные скобки должны быть разделены пробелом со значениями, запятая должна стоять вплотную к значению id пользователя, например:

{ 2, 2 } или { 03, 13 }

Значения записи передаются разделёнными знаком сравнения “==”, при этом описание и комментарий могут содержать от одного слова, дата и время состоят из двух слов, тогда как остальные поля должны содержать исключительно одно слово. Дата и время записываются в формате “dd/mm/yy hh:mm”, либо “dd.mm.yy hh:mm”. Адрес электронной почты должен быть допустимым.

Пример значения записи:

== here is a description== Mark == Tarabukin == Ivanovich ==  
[tamarkus@gmail.com](mailto:tamarkus@gmail.com) == +79990969615 == Moscow Orshanskaya street 3 room 304B ==  
it is a comment == 27.06.2023 09:00 ==

Команды read и read\_range доступны только для записей (note)

.

Команда create имеет вид:

create [ container ] [ container\_name ] [ path ],

где container – тип контейнера из списка: pool, scheme, collection, note.

container\_name – название контейнера, path – путь к контейнеру, состоящий из имен контейнеров по иерархии pool->scheme->collection.

Для пула необходимо выбрать тип аллокатора для хранения:

create pool [ allocator\_type ] [ pool\_name ]

Тип алокатора может быть:

- |        |   |   |
|--------|---|---|
| GH     | – | глобальная куча                                     |
| L(FBW) | – | аллокатор с освобождением в рассортированном списке |
| D(FBW) | – | аллокатор с освобождением с дескрипторами границ.   |

Из скобки нужно выбрать метод размещения:

- |   |   |                   |
|---|---|-------------------|
| F | – | первый подходящий |
|---|---|-------------------|

B — лучший подходящий  
W — худший подходящий

Пример команды для создания пула с аллокатором с освобождением в рассортированном списке методом худшего подходящего с именем my\_pool:

```
create pool LW my_pool
```

Пример команды для создания коллекции в пуле best\_pool в схеме best\_scheme с именем best\_collection:

```
create collection best_pool best_scheme best_collection
```

Пример команды для создания записи в пуле best\_pool в схеме best\_scheme в коллекции best\_collection записи выше:

```
create note best_pool best_scheme best_collection { 2, 2 } == here is a  
description== Mark == Tarabukin == Ivanovich == tamarkus@gmail.com ==  
+79990969615 == Moscow Orshanskaya street 3 room 304B == it is a comment ==  
27.06.2023 09:00 ==
```

Команда delete имеет вид

```
delete [ container ] [ container_name ] [ path ]
```

Чтобы удалить запись выше нужно ввести команду:

```
delete note best_pool best_scheme best_collection { 2, 2 }
```

Чтобы прочитать записи необходимо написать путь до коллекции и ключ:

```
read Pool Scheme Collection { 1, 13 }
```

Чтобы вывести записи из диапазона используем соответствующую команду и указываем два ключа поряд: левую и правую границы:

```
read_range Pool Scheme Collection { 0, 0 } { 2, 0 }
```

Для ведения логов на сервере-логгере нужно перейти в директорию source/logger/source/server\_logger/builder, выполнить команду “сmake init .” и “сmake –build .”. Для запуска сервера нужно передать ему путь к конфигурационному файлу логгера, например: “./server\_logger ../../test\_files/conf.json”. Логи будут сохраняться в текущей директории.

Отправим файл со следующим содержимым:

```

create pool LF fiji

create scheme fiji giant
create collection fiji giant my_new_collection

create note fiji giant my_new_collection { 1, 1 } == hi, == Mark == Tarabukin == Ivanovich
== tamarkus@gma.com == +79994617 ==
Dub 5 k 6 == No == 17.01.2023 15:43 ==

create note fiji giant my_new_collection { 1, 4 } == desc == Vaa == Cas == Ivanovich ==
tamardcdkus@gmail.com == +7999496000000 ==
Dub 5 k 6 == No == 17.06.2023 15:40 ==

create note fiji giant my_new_collection { 2, 1 } == rerer == Las == PPPAS == Ivanovich ==
tamaaaaakus@gmail.com == +7999496963 ==
Dub 22 == No == 17.06.2023 15:40 ==

create note fiji giant my_new_collection { 4, 5 } == desction == Vil == Vol == Ila ==
tus@mail.ru == +79994961543 ==
Dub 44 k 6 == No == 17.06.2023 15:40 ==

create note fiji giant my_new_collection { 10, 1 } == Description here == Mark == Tarabukin
== Ivanovich == tamarkus@gmail.com == +7999 ==
Dub 12 k 6 == comm == 17.06.2023 15:40 ==

read fiji giant my_new_collection { 2, 1 }

read_range fiji giant my_new_collection { 1, 100 } { 10, 1 }

delete note fiji giant my_new_collection { 4, 5 }

read_range fiji giant my_new_collection { 2, 1 } { 5, 1 }

```

Мы создаём пул fiji, в нём создаём схему giant, в которой создаём коллекцию my\_new\_collection. После этого создадим пять записей с ключами

```

{ 1, 1 },
{ 1, 4 },
{ 2, 1 },
{ 4, 5 }
{ 10, 1 }.

```

Прочитаем запись с ключом { 2, 1 }, прочитаем диапазон записей от { 1, 100 } до { 10, 1 }, удалим запись { 4, 5 }, прочитаем диапазон от { 2, 1 } до { 5, 1 }.

При чтении диапазона записей от { 1, 100 } до { 10, 1 } мы должны получить записи с ключами { 2, 1 } { 4, 5 } { 10, 1 }.

Получим следующий вывод, который ожидался:

```

Sending complete!
Recieved message:

-----

{ 2, 1 }
{
    rerer,
    Las,
    PPPAS,
    Ivanovich,
    tamaaaaakus@gmail.com,
    7999496963,
    Dub 22,
    No,
    17.06.2023 15:40
}

-----

{ 2, 1 }
{
    rerer,
    Las,
    PPPAS,
    Ivanovich,
    tamaaaaakus@gmail.com,
    7999496963,
    Dub 22,
    No,
    17.06.2023 15:40
}

{ 4, 5 }
{
    desction,
    Vil,
    Vol,
    Ila,
    tus@mail.ru,
    79994961543,
    Dub 44 k 6,
    No,
    17.06.2023 15:40
}

{ 10, 1 }
{
    Description here,
    Mark,
    Tarabukin,
    Ivanovich,
    tamarkus@gmail.com,
    7999,
    Dub 12 k 6,
    comm,
    17.06.2023 15:40
}

-----

{ 2, 1 }
{
    rerer,
    Las,
    PPPAS,
    Ivanovich,
    tamaaaaakus@gmail.com,
    7999496963,
    Dub 22,
    No,
    17.06.2023 15:40
}

```

Рис.3 Вывод программы

## Вывод

Я написал работающую программу, способную хранить записи и коллекции коллекций в АВЛ-дереве. Помимо этого я реализовал клиент-серверную архитектуру, при помощи Unix messages queue, а также логгирование на отдельном сервере.

В процессе разработки данного приложения на языке программирования C++ были использованы стандартные средства ввода/вывода для работы с файлами и операционной системой. Для упрощения реализации функционала приложения были использованы классы и функции, предоставляемые библиотеками стандартной библиотеки C++.

Разработка этого приложения позволила нам погрузиться в разработку на языке C++ и получить опыт работы с коллекциями данных, аллокаторами и файловым вводом/выводом. В результате выполнения работы было создано функциональное приложение на языке C++, способное выполнять операции над коллекциями данных заданных типов.

Реализованный функционал позволяет добавлять, читать, обновлять и удалять записи в коллекциях данных, а также выполнять операции с пулами данных, схемами данных и коллекциями данных. Особое внимание было уделено возможности настройки аллокаторов для эффективного управления памятью объектов данных. Были реализованы различные методы выделения и освобождения памяти, такие как первый подходящий, лучший подходящий, худший подходящий, освобождение в рассортированном списке, освобождение с дескрипторами границ и система двойников. Это позволяет эффективно использовать память при работе с коллекциями данных.

Также в процессе разработки приложения были использованы стандартные средства ввода/вывода C++ для работы с файлами и взаимодействия с операционной системой. Благодаря классам и функциям, предоставляемым стандартной библиотекой C++, была облегчена реализация функционала приложения.

Итак, выполнение данной работы позволило нам погрузиться в разработку приложений на языке C++ и получить опыт работы с коллекциями данных, аллокаторами и файловым вводом/выводом. Результатом работы является функциональное приложение, способное эффективно управлять коллекциями данных и памятью.

## Приложение

main.cpp:

```
#include <iostream>
#include <cstdlib>
#include <ostream>
#include <sstream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "processing/processing.h"
#include "logger/source/logger/concrete/logger_builder_concrete.h"
#define MSG_SIZE 5000
struct MsgQueue {
    long messageType;
    char buff[MSG_SIZE];
};
const int MSG_Q_KEY_FLAG_SERVER = 0664;
const int MSG_Q_KEY_FLAG_CLIENT = 0700;
const int MSG_Q_CHANNEL_RECEIVE = 26;
const int MSG_Q_CHANNEL_SEND = 16;
int main(int argc, char *argv[]) {
    logging::logger *logger;
    if (argc == 2) {
        std::string filename(argv[1]);
        logging::logger_builder *builder = new logger_builder_concrete();
        if (nullptr == builder) {
            std::cout << "Cannot allocate memory for builder" << std::endl;
            return -1;
        }
        logger = builder->construct_configuration(filename);
    }
    database *db = new database(logger);
    if (nullptr == db) {
        std::cout << "Cannot allocate memory for database" << std::endl;
        return -4;
    }
    // declare variables
    key_t key_receive = -1;
    key_t key_send = -1;
    int msqid_receive = -1;
    int msqid_send = -1;
    MsgQueue msg_receive;
    MsgQueue msg_send;
    key_receive = ftok("/bin/ls", 'P');
    if (key_receive < 0) {
        perror("ftok error");
        exit(1);
    }
    msqid_receive = msgget(key_receive, MSG_Q_KEY_FLAG_SERVER | IPC_CREAT);
    if (msqid_receive < 0) {
```

```

        perror("msgget");
        exit(1);
    }
    std::cout << "\nThe server has started!\n" << "\nWaiting for someone to connect to
server #" << msqid_receive
        << " with the key " << key_receive << std::endl;
    // while (1) {
    for (int i = 0; i < 1; ++i) {
        if (msgrcv(msqid_receive, &msg_receive, sizeof(msg_receive) - sizeof(long),
MSG_Q_CHANNEL_RECEIVE, 0) < 0)
        {
            perror("msgrcv");
            return -3;
        }
        std::stringstream ss;
        std::stringstream out;
        ss << msg_receive.buff;
        process_file(db, ss, out, logger);
        std::cout << "processed\n";
        std::string str = out.str();
        char *ptr = const_cast<char *>(str.c_str());
        int count = 0;
        while (*ptr != '\0' && count < MSG_SIZE) {
            msg_send.buff[count++] = *(ptr++);
        }
        msg_send.buff[count] = '\0';
        msg_send.messageType = MSG_Q_CHANNEL_SEND;
        key_send = ftok("/bin/ls", 'X');
        if (key_send < 0) {
            perror("ftok");
            return -1;
        }
        msqid_send = -1;
        msqid_send = msgget(key_send, MSG_Q_KEY_FLAG_CLIENT | IPC_CREAT);
        if (msqid_send < 0) {
            perror("msgget");
            return -2;
        }
        if (msgsnd(msqid_send, &msg_send, sizeof(msg_send) - sizeof(long), 0) < 0) {
            perror("msgsnd");
            return -4;
        }
        if (logger) logger->log("Message processed",
logging::logger::severity::information);
        sleep(1);
    }
    if (msgctl(msqid_receive, IPC_RMID, NULL) < 0) {
        perror("msgctl");
        return -5;
    }
    delete db;
    std::cout << "\nServer is now shutting down!\n";
    return 0;
}

```



## client\_main.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <ostream>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdexcept>
#include <fstream>
#include <stdexcept>
#include <string>
#define MSG_SIZE 5000
std::ifstream open_file(std::string &filename) {
    std::ifstream file;
    file.open(filename);
    if (!file.is_open() || !file) {
        throw std::runtime_error("File doesn't exist!");
    }
    return file;
}
struct MsgQueue {
    long messageType;
    char buff[MSG_SIZE];
};
const int MSG_Q_KEY_FLAG_SERVER = 0664;
const int MSG_Q_KEY_FLAG_CLIENT = 0700;
const int MSG_Q_CHANNEL_SEND = 26;
const int MSG_Q_CHANNEL_RECEIVE = 16;
int main(int argc, char *argv[]) {
    std::string filename = argv[1];
    std::ifstream file;
    try {
        file = open_file(filename);
    } catch (std::runtime_error &ex) {
        std::cout << ex.what() << std::endl;
        return 3;
    }
    if (!file.is_open()) {
        std::cout << "File was not opened" << std::endl;
        return -3;
    }
    // declare variables
    key_t key_send = -1;
    key_t key_receive = -1;
    int msqid_send = -1;
    int msqid_receive = -1;
    MsgQueue msg_send;
    MsgQueue msg_receive;
    // message send
```

```

// read file to buffer
std::string str((std::istreambuf_iterator<char>(file)),
                (std::istreambuf_iterator<char>()));
char *ptr = const_cast<char *>(str.c_str());
int count = 0;
while (*ptr != '\0' && count < MSG_SIZE) {
    msg_send.buff[count++] = *(ptr++);
}
msg_send.buff[count] = '\0';
msg_send.messageType = MSG_Q_CHANNEL_SEND;
key_send = ftok("/bin/ls", 'P');
if (key_send < 0) {
    perror("ftok");
    return -1;
}
msqid_send = msgget(key_send, MSG_Q_KEY_FLAG_SERVER | IPC_CREAT);
if (msqid_send < 0) {
    perror("msgget");
    return -2;
}
std::cout << "\nSuccessfully connected to server #" << msqid_send << " with the key " <<
key_send << std::endl;
// message receive
msg_receive.messageType = MSG_Q_CHANNEL_RECEIVE;
key_receive = ftok("/bin/ls", 'X');
if (key_receive < 0) {
    perror("ftok");
    return -1;
}
msqid_receive = msgget(key_receive, MSG_Q_KEY_FLAG_CLIENT | IPC_CREAT);
if (msqid_receive < 0) {
    perror("msgget");
    return -3;
}
// send message
if (msgsnd(msqid_send, &msg_send, sizeof(msg_send) - sizeof(long), 0) < 0) {
    perror("msgsnd");
    exit(1);
}
std::cout << "Sending complete!" << std::endl;
sleep(1);
// receive message
if (msgrcv(msqid_receive, &msg_send, sizeof(msg_send) - sizeof(long),
MSG_Q_CHANNEL_RECEIVE, 0) < 0)
{
    perror("msgrcv");
    exit(1);
}
std::cout << "Recieved message:" << std::endl;
std::cout << msg_send.buff;
// delete queue
if (msgctl(msqid_receive, IPC_RMID, NULL) < 0) {
    perror("msgctl");
    exit(1);
}

```

```

    }
    return 0;
}

```

## processing.h

```

#ifndef PROCESSING_H
#define PROCESSING_H
#include <iostream>
#include <sstream>
#include <stdexcept>
#include <vector>
#include <string>
#include "../file_opening/file_opening.h"
#include "../data/database/database.h"
#include "../logger/source/logger/complete/complete_logger.h"
#include "../memory/2/source/memory/memory.h"
int process_file(database *&db, std::stringstream &in_stream, std::stringstream &out_stream,
logging::logger *logger);
int validate(std::string &string);
#endif

```

## processing.cpp

```

#include <sstream>
#include <stdexcept>
#include <string>
#include "processing.h"
// int process_file(database *&db, std::string &filename, logging::logger *logger) {
int process_file(database *&db, std::stringstream &in_stream, std::stringstream &out_stream,
logging::logger *logger) {
    std::string word;
    std::vector<std::string> query(4);
    while (get_word(in_stream, word)) {
        if (word == "create") {
            if (!get_word(in_stream, word)) {
                out_stream << "Expected container type" << std::endl; break;
            }
            if (word == "pool") {
                if (!get_word(in_stream, word)) {
                    out_stream << "Expected allocator type [ GH | DF | DB | DW | LF | LB |
LW ]" << std::endl; break;
                }
                // allocator
                query[1] = word;
                if (!get_word(in_stream, word)) {
                    out_stream << "Expected pool name" << std::endl; break;
                }
                // pool
                query[0] = word;
                if (!validate(word)) {
                    out_stream << "Pool name contains denied symbols" << std::endl; break;
                }
            }
        }
    }
}

```

```

    }
    try {
        db->create_pool(query);
        if (logger) logger->log("created pool outside",
logging::logger::severity::debug);
    } catch (std::runtime_error &ex) {
        out_stream << ex.what() << std::endl;
    }
} else if (word == "scheme") {
    if (!get_word(in_stream, word)) {
        out_stream << "Expected pool name" << std::endl; break;
    }
    // pool
    query[0] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected scheme name" << std::endl; break;
    }
    // scheme
    query[1] = word;
    if (!validate(word)) {
        out_stream << "Pool name contains denied symbols" << std::endl; break;
    }
    try {
        db->create_scheme(query);
        if (logger) logger->log("created scheme outside",
logging::logger::severity::debug);
    } catch (std::runtime_error &ex) {
        out_stream << ex.what() << std::endl;
    }
} else if (word == "collection") {
    if (!get_word(in_stream, word)) {
        out_stream << "Expected pool name" << std::endl; break;
    }
    // pool
    query[0] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected scheme name" << std::endl; break;
    }
    // scheme
    query[1] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected collection name" << std::endl; break;
    }
    // collecion
    query[2] = word;
    if (!validate(word)) {
        out_stream << "Pool name contains denied symbols" << std::endl; break;
    }
    try {
        db->create_collection(query);
        if (logger) logger->log("created collection outside",
logging::logger::severity::debug);
    } catch (std::runtime_error &ex) {
        out_stream << ex.what() << std::endl;
    }
}

```

```

    }
} else if (word == "note") {
    if (!get_word(in_stream, word)) {
        out_stream << "Expected pool name" << std::endl; break;
    }
    // pool
    query[0] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected scheme name" << std::endl; break;
    }
    // scheme
    query[1] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected collection name" << std::endl; break;
    }
    // collection
    query[2] = word;
    // note
    try {
        db->create_note(in_stream, query);
        if (logger) logger->log("created note outside",
logging::logger::severity::debug);
    } catch (std::runtime_error &ex) {
        out_stream << ex.what() << std::endl;
    }
} else {
    out_stream << "No such container: " << word << std::endl;
}
} else if (word == "delete") {
    if (!get_word(in_stream, word)) {
        out_stream << "Expected container type" << std::endl; break;
    }
    if (word == "pool") {
        if (!get_word(in_stream, word)) {
            out_stream << "Expected pool name";
        }
        // pool
        query[0] = word;
        try {
            db->delete_pool(query);
            if (logger) logger->log("deleted pool outside",
logging::logger::severity::debug);
        } catch (std::runtime_error &ex) {
            out_stream << ex.what() << std::endl;
        }
    } else if (word == "scheme") {
        if (!get_word(in_stream, word)) {
            out_stream << "Expected pool name" << std::endl; break;
        }
        // pool
        query[0] = word;
        if (!get_word(in_stream, word)) {
            out_stream << "Expected scheme name" << std::endl; break;
        }
    }
}

```

```

        // scheme
        query[1] = word;
        try {
            db->delete_scheme(query);
            if (logger) logger->log("deleted scheme outside",
logging::logger::severity::debug);
        } catch (std::runtime_error &ex) {
            out_stream << ex.what() << std::endl;
        }
    } else if (word == "collection") {
        if (!get_word(in_stream, word)) {
            out_stream << "Expected pool name" << std::endl; break;
        }
        // pool
        query[0] = word;
        if (!get_word(in_stream, word)) {
            out_stream << "Expected scheme name" << std::endl; break;
        }
        // scheme
        query[1] = word;
        if (!get_word(in_stream, word)) {
            out_stream << "Expected collection name" << std::endl; break;
        }
        // collection
        query[2] = word;
        try {
            db->delete_collection(query);
            if (logger) logger->log("deleted collection outside",
logging::logger::severity::debug);
        } catch (std::runtime_error &ex) {
            out_stream << ex.what() << std::endl;
        }
    } else if (word == "note") {
        if (!get_word(in_stream, word)) {
            out_stream << "Expected pool name" << std::endl; break;
        }
        // pool
        query[0] = word;
        if (!get_word(in_stream, word)) {
            out_stream << "Expected scheme name" << std::endl; break;
        }
        // scheme
        query[1] = word;
        if (!get_word(in_stream, word)) {
            out_stream << "Expected collection name" << std::endl; break;
        }
        // collecion
        query[2] = word;
        // note
        try {
            db->delete_note(in_stream, query);
            if (logger) logger->log("deleted note outside",
logging::logger::severity::debug);
        } catch (std::runtime_error &ex) {

```

```

        out_stream << ex.what() << std::endl;
    }
} else {
    out_stream << "No such container: " << word << std::endl;
}
} else if (word == "read") {
    if (!get_word(in_stream, word)) {
        out_stream << "Expected pool name" << std::endl; break;
    }
    // pool
    query[0] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected scheme name" << std::endl; break;
    }
    // scheme
    query[1] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected collection name" << std::endl; break;
    }
    // collection
    query[2] = word;
    // note
    try {
        out_stream << "\n-----\n\n";
        db->read_note(in_stream, out_stream, query);
        if (logger) logger->log("read note outside",
logging::logger::severity::debug);
    } catch (std::runtime_error &ex) {
        out_stream << ex.what() << std::endl;
    }
} else if (word == "read_range") {
    if (!get_word(in_stream, word)) {
        out_stream << "Expected pool name" << std::endl; break;
    }
    // pool
    query[0] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected scheme name" << std::endl; break;
    }
    // scheme
    query[1] = word;
    if (!get_word(in_stream, word)) {
        out_stream << "Expected collection name" << std::endl; break;
    }
    // collection
    query[2] = word;
    // note
    try {
        out_stream << "\n-----\n\n";
        db->read_note_range(in_stream, out_stream, query);
        if (logger) logger->log("read note range outside",
logging::logger::severity::debug);
    } catch (std::runtime_error &ex) {
        out_stream << ex.what() << std::endl;
    }
}

```

```

        }
    } else {
        out_stream << "No such command: " << word << std::endl;
    }
}
if (logger) logger->log("EnD", logging::logger::severity::debug);
return 0;
}
int validate(std::string &string) {
    for (char c : string) {
        if (c == '\n' || c == '\t' || c == ' ') {
            return 0;
        }
    }
    return 1;
}
}

```

## binary\_search\_tree.h

```

#ifndef SANDBOX_CPP_BINARY_SEARCH_TREE_H
#define SANDBOX_CPP_BINARY_SEARCH_TREE_H
#include <cstdint>
#include <exception>
#include <functional>
#include <iostream>
#include <stack>
#include <sstream>
#include <stdexcept>
#include <utility>
#include <vector>
#define UNUSED(expr) do { (void)(expr); } while (0)
#include "../../../../../data/type_data/type_data.h"
#include "../../../../../memory/3/source/memory_with_list/memory_with_list.h"
#include "associative_container.h"
#include "../../../../../logger/source/logger/complete/complete_logger.h"
#include "../../../../../11/source/allocator/safe_allocator.h"
template <class T>
std::string cast_to_str(const T& object) {
    std::stringstream ss;
    ss << object;
    return ss.str();
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
class binary_search_tree:
    public associative_container<tkey, tvalue>, protected logging::complete_logger,
    protected allocating::safe_allocator
{
protected:
    struct tree_node
    {

```



```

        tkey key;
        tvalue value;
        tree_node *left_subtree_address;
        tree_node *right_subtree_address;
    };
public:
    class prefix_iterator final
    {
    private:
        tree_node *_tree_root;
        std::stack<tree_node *> _way;
        tree_node *_current;
        tree_node *_begin;
        tree_node *_end;
    public:
        explicit prefix_iterator(
            tree_node *tree_root,
            tree_node *begin_or_end_node);
        tree_node *get_node_pointer() const;
        void set_iterator(tree_node *);
    public:
        bool operator==(
            prefix_iterator const &other) const;
        bool operator!=(
            prefix_iterator const &other) const;
        prefix_iterator& operator++();
        prefix_iterator operator++(
            int not_used);
        std::tuple<unsigned int, tkey const&, tvalue const&> operator*() const;
    };
    class infix_iterator final
    {
    private:
        tree_node *_tree_root;
        std::stack<tree_node *> _way;
        tree_node *_current;
        tree_node *_begin;
        tree_node *_end;
    public:
        explicit infix_iterator(
            tree_node *tree_root,
            tree_node *begin_or_end_node);
        tree_node *get_node_pointer() const;
        void set_iterator(tree_node *, std::stack<binary_search_tree::tree_node *>
&path_to_subtree_root_exclusive);
    public:
        bool operator==(
            infix_iterator const &other) const;
        bool operator!=(
            infix_iterator const &other) const;
        infix_iterator& operator++();
        infix_iterator operator++(
            int not_used);
        std::tuple<unsigned int, tkey const&, tvalue const&> operator*() const;

```

```

};
class postfix_iterator final
{
private:
    tree_node *_tree_root;
    std::stack<tree_node *> _way;
    tree_node *_current;
    tree_node *_begin;
    tree_node *_end;
public:
    explicit postfix_iterator(
        tree_node *tree_root,
        tree_node *begin_or_end_node);
    tree_node *get_node_pointer() const;
public:
    bool operator==(
        postfix_iterator const &other) const;
    bool operator!=(
        postfix_iterator const &other) const;
    postfix_iterator &operator++();
    postfix_iterator operator++(
        int not_used);
    std::tuple<unsigned int, tkey const&, tvalue const&> operator*() const;
};
protected:
    class insertion_template_method : private allocating::safe_allocator, private
logging::complete_logger
    {
    public:
        virtual ~insertion_template_method() = default;
        insertion_template_method(binary_search_tree<tkey, tvalue, tkey_comparer> *tree)
            : _tree(tree) {}
    public:
        void insert(
            tkey const &key,
            tvalue &&value,
            tree_node *&tree_root_address);
    private:
        void insert_inner(
            tkey const &key,
            tvalue &&value,
            tree_node *&subtree_root_address,
            std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive);
    protected:
        virtual void after_insert_inner(
            tkey const &key,
            tree_node *&subtree_root_address,
            std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive);
    private:
        binary_search_tree<tkey, tvalue, tkey_comparer> *_tree;
        logging::logger *get_logger() const override;
        allocating::memory *get_allocator() const override;
    protected:
        virtual size_t get_size_node() const;

```

```

        virtual void initialize_new_node(
            tree_node *&new_node, tkey const &key, tvalue &&value) const;
};
class reading_template_method : private allocating::safe_allocator, private
logging::complete_logger
{
public:
    virtual ~reading_template_method() = default;
    reading_template_method(binary_search_tree<tkey, tvalue, tkey_comparer> *tree)
        : _tree(tree) {}
public:
    tvalue const &read(
        tkey const &key,
        tree_node *&tree_root_address);
    bool find(
        tkey const &key,
        tree_node *&tree_root_address,
        std::pair<tkey, tvalue *> *found);
    void find_left_bound(
        tkey const &key,
        tree_node *&tree_root_address,
        std::stack<binary_search_tree::tree_node *> &path_to_subtree_root_exclusive,
        tree_node *&node_need);
private:
    tvalue const &read_inner(
        tkey const &key,
        tree_node *&subtree_root_address,
        std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive);
protected:
    virtual void after_read_inner(
        tkey const &key,
        tree_node *&subtree_root_address,
        std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive);
private:
    binary_search_tree<tkey, tvalue, tkey_comparer> *_tree;
    logging::logger *get_logger() const override;
    allocating::memory *get_allocator() const override;
};
class removing_template_method : private allocating::safe_allocator, private
logging::complete_logger
{
public:
    virtual ~removing_template_method() = default;
    removing_template_method(binary_search_tree<tkey, tvalue, tkey_comparer> *tree)
        : _tree(tree) {}
public:
    tvalue &&remove(
        tkey const &key,
        tree_node *&tree_root_address);
private:
    tvalue &&remove_inner(
        tkey const &key,
        tree_node *&subtree_root_address,
        std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive);

```

```

protected:
    virtual void after_remove_inner(
        tkey const &key,
        tree_node *&subtree_root_address,
        std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive);
private:
    binary_search_tree<tkey, tvalue, tkey_comparer> *_tree;
    logging::logger *get_logger() const override;
    allocating::memory *get_allocator() const override;
};
protected:
    insertion_template_method *_insertion;
    reading_template_method *_reading;
    removing_template_method *_removing;
    allocating::memory *_allocator;
    logging::logger *_logger;
    tree_node *_root;
protected:
    explicit binary_search_tree(
        insertion_template_method *insertion,
        reading_template_method *reading,
        removing_template_method *removing,
        allocating::memory *allocator = nullptr,
        logging::logger *logger = nullptr,
        tree_node *root = nullptr);
public:
    explicit binary_search_tree(
        allocating::memory *allocator = nullptr,
        logging::logger *logger = nullptr);
    binary_search_tree(
        binary_search_tree const &other);
    binary_search_tree(
        binary_search_tree &&other) noexcept;
    binary_search_tree &operator=(
        binary_search_tree const &other);
    binary_search_tree &operator=(
        binary_search_tree &&other) noexcept;
    ~binary_search_tree() override;
public:
    void insert(
        tkey const &key,
        tvalue &&value) final override;
    tvalue const &get(
        tkey const &key) final override;
    tvalue &&remove(
        tkey const &key) final override;
    bool find(
        tkey const &key, std::pair<tkey, tvalue *> *found) final override;
private:
    void find_left_bound(tkey const &key,
        std::stack<tree_node *> &path_to_subtree_root_exclusive,
        tree_node *&node_need);
private:
    logging::logger *get_logger() const override;

```

```

    allocating::memory *get_allocator() const override;
public:
    prefix_iterator begin_prefix() const noexcept;
    prefix_iterator end_prefix() const noexcept;
    infix_iterator begin_infix() const noexcept;
    infix_iterator end_infix() const noexcept;
    postfix_iterator begin_postfix() const noexcept;
    postfix_iterator end_postfix() const noexcept;
protected:
    virtual void left_rotation(tree_node **subtree_root, tree_node **parent) const;
    virtual void right_rotation(tree_node **subtree_root, tree_node **parent) const;
public:
    void print_prefix() const;
    void print_infix() const;
    void print_postfix() const;
    virtual void print_container() const override;
    virtual void print_container_logger() const override;
    virtual void print_notes_between(std::stringstream &out_stream, tkey left_bound, tkey
right_bound) override;
};
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::print_notes_between(std::stringstream
&out_stream, tkey const left_bound, tkey const right_bound) {
    safe_log("START", logging::logger::severity::information);
    auto it = end_infix();
    tree_node *node = nullptr;
    std::stack<binary_search_tree::tree_node *> path_to_subtree_root_exclusive;
    find_left_bound(left_bound, path_to_subtree_root_exclusive, node);
    if (nullptr == node) {
        return;
    }
    it.set_iterator(node, path_to_subtree_root_exclusive);
    tkey_comparer comparer;
    auto end = end_infix();
    print_container();
    safe_log("left_bound " + cast_to_str(left_bound) + " right bound " +
cast_to_str(right_bound), logging::logger::severity::information);
    auto iii = end;
    while (it != end && comparer(std::get<1>(*it), right_bound) >= 0) {
        if (iii != end ? comparer(std::get<1>(*it), std::get<1>(*iii)) : 1) {
            out_stream << std::get<1>(*it) << std::endl;
            out_stream << std::get<2>(*it) << std::endl << std::endl;
        }
        iii = it;
        ++it;
    }
    safe_log("ENDED", logging::logger::severity::information);
}
template<
    typename tkey,
    typename tvalue,

```

```

    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method::find_left_bound(
    tkey const &key,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&subtree_root_address,
    std::stack<binary_search_tree::tree_node *> &path_to_subtree_root_exclusive,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&node_need) {
    if (node_need) _tree->safe_log("Current node " + cast_to_str(node_need->key),
logging::logger::severity::information);
    if (nullptr == subtree_root_address) {
        // next node
        _tree->safe_log("Leaf", logging::logger::severity::debug);
        if (path_to_subtree_root_exclusive.empty()) {
            return;
        }
        while (path_to_subtree_root_exclusive.top() != node_need) {
            path_to_subtree_root_exclusive.pop();
            if (path_to_subtree_root_exclusive.empty()) {
                break;
            }
        }
        return;
    }
    tkey_comparer comparer;
    if (comparer(subtree_root_address->key, key) == 0) {
        node_need = subtree_root_address;
        path_to_subtree_root_exclusive.push(subtree_root_address);
        _tree->safe_log(cast_to_str(subtree_root_address->key) + " == " +
cast_to_str(node_need->key), logging::logger::severity::information);
        return;
    }
    tree_node *next_node;
    // _tree->safe_log("Pair {" + cast_to_str(subtree_root_address->key) + ", " +
cast_to_str(subtree_root_address->value) + "}", logging::logger::severity::debug);
    if (comparer(subtree_root_address->key, key) > 0) {
        next_node = subtree_root_address->right_subtree_address;
    } else {
        node_need = subtree_root_address;
        next_node = subtree_root_address->left_subtree_address;
    }
    path_to_subtree_root_exclusive.push(subtree_root_address);
    find_left_bound(key, next_node, path_to_subtree_root_exclusive, node_need);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::print_container_logger() const {
    std::function<void(tree_node *, size_t)> print_tree;
    print_tree = [&](tree_node *subtree_root, size_t deep) {
        if (deep == 0) {
            safe_log("Tree", logging::logger::severity::information);
        }
        if (nullptr != subtree_root) {

```

```

        print_tree(subtree_root->left_subtree_address, deep + 1);
    }
    if (subtree_root == nullptr) {
        return;
    }
    // safe_log(cast_to_str(subtree_root->key), logging::logger::severity::information);
    print_tree(subtree_root->right_subtree_address, deep + 1);
};
print_tree(_root, 0);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::print_container() const {
    std::function<void(tree_node *, size_t)> print_tree;
    print_tree = [&](tree_node *subtree_root, size_t deep) {
        if (deep == 0) {
            std::cout << "\nTree:\n";
        }
        if (nullptr != subtree_root) {
            print_tree(subtree_root->right_subtree_address, deep + 1);
        }
        std::string s;
        for (size_t i = 0; i < deep; ++i) {
            s += "\t";
        }
        s += "_____";
        if (subtree_root == nullptr) {
            s += "NULL\n";
            std::cout << s;
            return;
        }
        std::cout << s << cast_to_str(subtree_root->key) << std::endl;
        print_tree(subtree_root->left_subtree_address, deep + 1);
    };
    print_tree(_root, 0);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::print_prefix() const {
    auto it_end = end_prefix();
    for (auto it = begin_prefix(); it != it_end; ++it) {
        std::cout << std::get<2>(*it) << " ";
    }
    std::cout << std::endl;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::print_infix() const {

```

```

    auto it_end = end_infix();
    for (auto it = begin_infix(); it != it_end; ++it) {
        std::cout << std::get<2>(*it) << " ";
    }
    std::cout << std::endl;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::print_postfix() const {
    auto it_end = end_postfix();
    for (auto it = begin_postfix(); it != it_end; ++it) {
        std::cout << std::get<2>(*it) << " ";
    }
    std::cout << std::endl;
}
// region iterators implementation
// region prefix_iterator implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator::prefix_iterator(
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *tree_root,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *begin_or_end_node) {
    _way = std::stack<tree_node *>();
    _begin = tree_root;
    _end = tree_root;
    if (nullptr != _end) {
        while (_end->right_subtree_address) {
            _end = _end->right_subtree_address;
        }
    }
    _current = nullptr == begin_or_end_node ? nullptr : _begin;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator::operator==(
    binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator const &other) const {
    if (_current == other._current && _way == other._way) {
        return true;
    }
    return false;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator::operator!=(
    binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator const &other) const {
    return !(*this == other);
}

```



```

}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator
&binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator::operator++() {
    if (nullptr == _current || _current == _end) {
        _current = nullptr;
        _way = std::stack<tree_node*>();
        return *this;
    }
    if (nullptr != _current->left_subtree_address) {
        _way.push(_current);
        _current = _current->left_subtree_address;
        return *this;
    }
    if (nullptr != _current->right_subtree_address) {
        _way.push(_current);
        _current = _current->right_subtree_address;
        return *this;
    }
    if (_current == _way.top()->left_subtree_address) {
        _current = _way.top();
        _way.pop();
        while (nullptr == _current->right_subtree_address && !_way.empty()) {
            _current = _way.top();
            _way.pop();
        }
        _current = _current->right_subtree_address;
    }
    return *this;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator::operator++(
    int not_used) {
    UNUSED(not_used);
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator previous =
*this;
    ++(*this);
    return previous;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
std::tuple<unsigned int, tkey const&, tvalue const&> binary_search_tree<tkey, tvalue,
tkey_comparer>::prefix_iterator::operator*() const {
    return std::tuple<unsigned int, tkey const&, tvalue const&>(_way.size(), _current->key,
_current->value);
}

```

```

}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator::get_node_pointer() const
{
    return _current;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator::get_node_pointer() const {
    return _current;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator::get_node_pointer() const
{
    return _current;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::prefix_iterator::set_iterator(binary_search_tree<tkey, tvalue,
tkey_comparer>::tree_node *node) {
    _current = node;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::infix_iterator::set_iterator(binary_search_tree<tkey, tvalue,
tkey_comparer>::tree_node *node,
        std::stack<binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *>
&path_to_subtree_root_exclusive) {
    _way = path_to_subtree_root_exclusive;
    _current = node;
}
//endregion prefix_iterator implementation
//region infix_iterator implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>

```

```

binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator::infix_iterator(
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *tree_root,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *begin_or_end_node) {
    _way = std::stack<tree_node *>();
    _begin = tree_root;
    if (nullptr == begin_or_end_node) {
        if (nullptr != _begin) {
            while (nullptr != _begin->left_subtree_address) {
                _begin = _begin->left_subtree_address;
            }
        }
        _current = nullptr;
    } else {
        if (nullptr != _begin) {
            while (nullptr != _begin->left_subtree_address) {
                _way.push(_begin);
                _begin = _begin->left_subtree_address;
            }
        }
        _current = _begin;
    }
    _end = tree_root;
    if (nullptr != _end) {
        while (nullptr != _end->right_subtree_address) {
            _end = _end->right_subtree_address;
        }
    }
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator::operator==(
    binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator const &other) const {
    if (_current == other._current && _way == other._way) {
        return true;
    }
    return false;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator::operator!=(
    binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator const &other) const {
    return !(*this == other);
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator
&binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator::operator++() {
    if (nullptr == _current || _current == _end) {

```

```

        _current = nullptr;
        _way = std::stack<tree_node*>();
        return *this;
    }
    if (nullptr != _current->right_subtree_address) {
        _way.push(_current);
        _current = _current->right_subtree_address;
        while (nullptr != _current->left_subtree_address) {
            _way.push(_current);
            _current = _current->left_subtree_address;
        }
        return *this;
    }
    if (!_way.empty()) {
        if (_way.top()->left_subtree_address == _current) {
            _current = _way.top();
            _way.pop();
            return *this;
        }
        while (_way.top()->right_subtree_address == _current) {
            _current = _way.top();
            _way.pop();
        }
        _current = _way.top();
        _way.pop();
        return *this;
    }
    // if (_way.empty()) {
    //     _way.push(_tree_root);
    // }
    return *this;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator::operator++(
    int not_used) {
    UNUSED(not_used);
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator previous =
*this;
    ++(*this);
    return previous;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
std::tuple<unsigned int, tkey const&, tvalue const&> binary_search_tree<tkey, tvalue,
tkey_comparer>::infix_iterator::operator*() const {
    return std::tuple<unsigned int, tkey const&, tvalue const&>(_way.size(), _current->key,
_current->value);
}

```

```

//endregion infix_iterator implementation
//region postfix_iterator implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator::postfix_iterator(
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *tree_root,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *begin_or_end_node) {
    _way = std::stack<tree_node *>();
    _begin = tree_root;
    if (nullptr == begin_or_end_node) {
        if (nullptr != _begin) {
            while (nullptr != _begin->left_subtree_address) {
                _begin = _begin->left_subtree_address;
            }
        }
        _current = nullptr;
    } else {
        while (nullptr != _begin->left_subtree_address) {
            _way.push(_begin);
            _begin = _begin->left_subtree_address;
        }
        _current = _begin;
    }
    _end = tree_root;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator::operator==(
    binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator const &other) const {
    if (_current == other._current && _way == other._way) {
        return true;
    }
    return false;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator::operator!=(
    binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator const &other) const {
    return !(*this == other);
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator
&binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator::operator++() {
    if (nullptr == _current || _current == _end) {
        _current = nullptr;
    }
}

```

```

        _way = std::stack<tree_node *>();
        return *this;
    }
    if (_way.top()->right_subtree_address == _current) {
        _current = _way.top();
        _way.pop();
        return *this;
    }
    if (_way.top()->left_subtree_address == _current) {
        _current = _way.top();
        _way.pop();
        if (nullptr != _current->right_subtree_address) {
            _way.push(_current);
            _current = _current->right_subtree_address;
            while (nullptr != _current->left_subtree_address || nullptr !=
                _current->right_subtree_address) {
                _way.push(_current);
                if (nullptr != _current->left_subtree_address) {
                    _current = _current->left_subtree_address;
                } else {
                    _current = _current->right_subtree_address;
                }
            }
        }
    }
    return *this;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator::operator++(
    int not_used) {
    UNUSED(not_used);
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator previous =
        *this;
    ++(*this);
    return previous;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
std::tuple<unsigned int, tkey const&, tvalue const&> binary_search_tree<tkey, tvalue,
tkey_comparer>::postfix_iterator::operator*() const {
    return std::tuple<unsigned int, tkey const&, tvalue const&>(_way.size(), _current->key,
        _current->value);
}
// endregion prefix_iterator implementation
// endregion iterators implementation
// region template methods implementation
// region rotation implementation
template<

```

```

    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::left_rotation(
    tree_node **subtree_root, tree_node **parent) const {
    tree_node *node = (*subtree_root)->right_subtree_address;
    (*subtree_root)->right_subtree_address = node->left_subtree_address;
    node->left_subtree_address = *subtree_root;
    if (nullptr != parent) {
        if ((*parent)->right_subtree_address == *subtree_root) {
            (*parent)->right_subtree_address = node;
            // } else if ((*parent)->left_subtree_address == *subtree_root) {
        } else {
            (*parent)->left_subtree_address = node;
        }
    }
    *subtree_root = node;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::right_rotation(
    tree_node **subtree_root, tree_node **parent) const {
    safe_log("" + cast_to_str(*subtree_root), logging::logger::severity::debug);
    if (parent) {
        if ((*parent)->right_subtree_address) {
            safe_log("" + cast_to_str((*parent)->right_subtree_address),
logging::logger::severity::debug);
        }
    }
    tree_node *node = (*subtree_root)->left_subtree_address;
    (*subtree_root)->left_subtree_address = node->right_subtree_address;
    node->right_subtree_address = *subtree_root;
    if (nullptr != parent) {
        if ((*parent)->right_subtree_address == *subtree_root) {
            (*parent)->right_subtree_address = node;
        } else {
            (*parent)->left_subtree_address = node;
        }
    }
    *subtree_root = node;
}

// endregion rotation implementation
// region insertion implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::insertion_template_method::insert(
    tkey const &key,
    tvalue &&value,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&tree_root_address) {
    UNUSED(tree_root_address);

```

```

        std::stack<binary_search_tree::tree_node **> path_to_subtree_root_exclusive;
        return insert_inner(key, std::move(value), _tree->_root,
                           path_to_subtree_root_exclusive);
    }
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method::insert_inner(
    tkey const &key,
    tvalue &&value,
    binary_search_tree::tree_node *&subtree_root_address,
    std::stack<binary_search_tree::tree_node **> &path_to_subtree_root_exclusive) {
    tkey_comparer comparer;
    _tree->safe_log("Insert inner", logging::logger::severity::trace);
    if (nullptr == subtree_root_address) {
        // creating new node here
        _tree->safe_log("subtree_root_address is nullptr. Start creating a new node",
logging::logger::severity::debug);
        tree_node *new_node = reinterpret_cast<tree_node *>(
            _tree->safe_allocate(get_size_node())
        );
        initialize_new_node(new_node, key, std::move(value));
        if (path_to_subtree_root_exclusive.empty()) {
            subtree_root_address = new_node;
        } else {
            if (comparer((*path_to_subtree_root_exclusive.top())->key, key) > 0) {
                (*path_to_subtree_root_exclusive.top())->right_subtree_address = new_node;
            } else {
                (*path_to_subtree_root_exclusive.top())->left_subtree_address = new_node;
            }
        }
        _tree->safe_log("Created node is " + cast_to_str(new_node),
logging::logger::severity::debug);
        after_insert_inner(key, new_node, path_to_subtree_root_exclusive);
        return;
    }
    int comparation = comparer(subtree_root_address->key, key);
    if (comparation == 0) {
        // replacing node by new one
        _tree->safe_log("node with existing key. Start creating a replacement node key: " +
cast_to_str(key), logging::logger::severity::debug);
        tree_node *new_node = reinterpret_cast<tree_node *>(
            _tree->safe_allocate(get_size_node())
        );
        initialize_new_node(new_node, key, std::move(value));
        new_node->left_subtree_address = subtree_root_address->left_subtree_address;
        new_node->right_subtree_address = subtree_root_address->right_subtree_address;
        if (!path_to_subtree_root_exclusive.empty()) {
            if (comparer((*path_to_subtree_root_exclusive.top())->key, key) > 0) {
                (*path_to_subtree_root_exclusive.top())->right_subtree_address = new_node;
            } else {
                (*path_to_subtree_root_exclusive.top())->left_subtree_address = new_node;
            }
        }
    }
}

```



```

    }
    }
    subtree_root_address->~tree_node();
    _tree->safe_deallocate(subtree_root_address);
    subtree_root_address = new_node;
    _tree->safe_log("Replaced node is " + cast_to_str(new_node),
logging::logger::severity::debug);
    after_insert_inner(key, new_node, path_to_subtree_root_exclusive);
    return;
}
path_to_subtree_root_exclusive.push(&subtree_root_address);
tree_node *next_node;
if (comparation > 0) {
    _tree->safe_log("Move right", logging::logger::severity::debug);
    next_node = subtree_root_address->right_subtree_address;
} else {
    _tree->safe_log("Move left", logging::logger::severity::debug);
    next_node = subtree_root_address->left_subtree_address;
}
insert_inner(key, std::move(value), next_node, path_to_subtree_root_exclusive);
path_to_subtree_root_exclusive.pop();
// tree_node **nodenode = &next_node;
// after_insert_inner(key, *nodenode, path_to_subtree_root_exclusive);
_tree->safe_log("after inner #3", logging::logger::severity::debug);
after_insert_inner(key, subtree_root_address, path_to_subtree_root_exclusive);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method::after_insert_inner(
    tkey const &key,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&subtree_root_address,
    std::stack<binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
&path_to_subtree_root_exclusive) {
    UNUSED(key);
    UNUSED(subtree_root_address);
    UNUSED(path_to_subtree_root_exclusive);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
size_t binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method::get_size_node() const {
    return sizeof(tree_node);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method::initialize_new_node(

```

```

        tree_node *&new_node, tkey const &key, tvalue &&value) const {
            new (new_node) tree_node{key, std::move(value), nullptr, nullptr };
        }
// endregion insertion implementation
// region reading implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
tvalue const &binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method::read(
    tkey const &key,
    tree_node *&tree_root_address) {
    UNUSED(tree_root_address);
    std::stack<tree_node **> path_to_subtree_root_exclusive;
    return read_inner(key, _tree->_root, path_to_subtree_root_exclusive);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::reading_template_method::find(
    tkey const &key,
    tree_node *&subtree_root_address,
    std::pair<tkey, tvalue *> *found) {
    if (nullptr == subtree_root_address) {
        _tree->safe_log("A value was not found", logging::logger::severity::debug);
        throw std::runtime_error("A value was not found");
        return false;
    }
    tkey_comparer comparer;
    if (comparer(subtree_root_address->key, key) == 0) {
        // _tree->safe_log("Found a value " + cast_to_str(subtree_root_address->value),
logging::logger::severity::debug);
        found->first = subtree_root_address->key;
        found->second = &(subtree_root_address->value);
        return true;
    }
    tree_node *next_node;
    // _tree->safe_log("Pair {" + cast_to_str(subtree_root_address->key) + ", " +
cast_to_str(subtree_root_address->value) + "}", logging::logger::severity::debug);
    if (comparer(subtree_root_address->key, key) > 0) {
        next_node = subtree_root_address->right_subtree_address;
    } else {
        next_node = subtree_root_address->left_subtree_address;
    }
    return find(key, next_node, found);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
tvalue const &binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method::read_inner(

```

```

    tkey const &key,
    tree_node *&subtree_root_address,
    std::stack<tree_node **> &path_to_subtree_root_exclusive) {
    if (nullptr == subtree_root_address) {
        _tree->safe_log("A value was not found", logging::logger::severity::debug);
        throw std::runtime_error("No such key in a tree!");
    }
    tkey_comparer comparer;
    if (comparer(subtree_root_address->key, key) == 0) {
        // _tree->safe_log("Found a value " + cast_to_str(subtree_root_address->value),
logging::logger::severity::debug);
        after_read_inner(key, subtree_root_address, path_to_subtree_root_exclusive);
        return subtree_root_address->value;
    }
    path_to_subtree_root_exclusive.push(&subtree_root_address);
    tree_node *next_node;
    // _tree->safe_log("Pair {" + cast_to_str(subtree_root_address->key) + ", " +
cast_to_str(subtree_root_address->value) + "}", logging::logger::severity::debug);
    if (comparer(subtree_root_address->key, key) > 0) {
        next_node = subtree_root_address->right_subtree_address;
    } else {
        next_node = subtree_root_address->left_subtree_address;
    }
    tvalue const &value = read_inner(key, next_node, path_to_subtree_root_exclusive);
    after_read_inner(key, subtree_root_address, path_to_subtree_root_exclusive);
    return value;
}

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method::after_read_inner(
    tkey const &key,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&subtree_root_address,
    std::stack<binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
&path_to_subtree_root_exclusive) {
    UNUSED(key);
    UNUSED(subtree_root_address);
    UNUSED(path_to_subtree_root_exclusive);
}

// endregion reading implementation
// region removing implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
tvalue &&binary_search_tree<tkey, tvalue, tkey_comparer>::removing_template_method::remove(
    tkey const &key,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&tree_root_address) {
    UNUSED(tree_root_address);
    std::stack<binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
path_to_subtree_root_exclusive;
    return std::move(remove_inner(key, _tree->_root,

```

```

        path_to_subtree_root_exclusive));
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
tvalue &&binary_search_tree<tkey, tvalue,
tkey_comparer>::removing_template_method::remove_inner(
    tkey const &key,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&subtree_root_address,
    std::stack<binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
&path_to_subtree_root_exclusive) {
    if (nullptr == subtree_root_address) {
        throw std::runtime_error("No such key in a tree!");
    }
    tkey_comparer comparer;
    int comparison = comparer(subtree_root_address->key, key);
    if (comparison == 0) {
        tvalue &&value_to_remove = std::move(subtree_root_address->value);
        if (subtree_root_address->left_subtree_address != nullptr && nullptr !=
subtree_root_address->right_subtree_address) {
            tree_node *left_max = subtree_root_address->left_subtree_address;
            tree_node *left_max_parent = subtree_root_address;
            path_to_subtree_root_exclusive.push(&subtree_root_address);
            std::stack<tree_node *> references;
            int ii = 0;
            while (left_max->right_subtree_address) {
                ++ii;
                references.push(left_max);
                path_to_subtree_root_exclusive.push(&(references.top()));
                left_max_parent = left_max;
                left_max = left_max->right_subtree_address;
                _tree->safe_log("=== node " + cast_to_str(&(references.top()))),
logging::logger::severity::debug);
            }
            _tree->safe_log("iterations " + cast_to_str(ii),
logging::logger::severity::debug);
            subtree_root_address->key = left_max->key;
            subtree_root_address->value = std::move(left_max->value);
            if (left_max_parent->right_subtree_address == left_max) {
                left_max_parent->right_subtree_address = left_max->left_subtree_address;
            } else {
                left_max_parent->left_subtree_address = left_max->left_subtree_address;
            }
            left_max->~tree_node();
            _tree->safe_deallocate(left_max);
            if (!path_to_subtree_root_exclusive.empty() && ii != 0) {
                tree_node *touched_node = *(path_to_subtree_root_exclusive.top());
                _tree->safe_log("After remove inner removing
=====",
logging::logger::severity::debug);
                while (comparer(touched_node->key, subtree_root_address->key)) {
                    path_to_subtree_root_exclusive.pop();
                    if (comparer(touched_node->key, key)) {

```

```

        after_remove_inner(key, touched_node,
path_to_subtree_root_exclusive);
    }
    if (!path_to_subtree_root_exclusive.empty()) {
        touched_node = *(path_to_subtree_root_exclusive.top());
    } else {
        break;
    }
}
_tree->safe_log("After remove inner removing completed! =====",
logging::logger::severity::debug);
}
return std::move(value_to_remove);
} else if (nullptr != subtree_root_address->left_subtree_address) {
    tree_node *left_node = subtree_root_address->left_subtree_address;
    subtree_root_address->~tree_node();
    _tree->safe_deallocate(subtree_root_address);
    subtree_root_address = left_node;
} else if (nullptr != subtree_root_address->right_subtree_address) {
    tree_node *right_node = subtree_root_address->right_subtree_address;
    subtree_root_address->~tree_node();
    _tree->safe_deallocate(subtree_root_address);
    subtree_root_address = right_node;
} else {
    subtree_root_address->~tree_node();
    _tree->safe_deallocate(subtree_root_address);
    subtree_root_address = nullptr;
}
after_remove_inner(key, subtree_root_address, path_to_subtree_root_exclusive);
return std::move(value_to_remove);
}
binary_search_tree::tree_node **next_node = &(comparation > 0 ?
subtree_root_address->right_subtree_address : subtree_root_address->left_subtree_address);
path_to_subtree_root_exclusive.push(&subtree_root_address);
tvalue &&value_to_remove = std::move(remove_inner(key, *next_node,
path_to_subtree_root_exclusive));
path_to_subtree_root_exclusive.pop();
after_remove_inner(key, subtree_root_address, path_to_subtree_root_exclusive);
return std::move(value_to_remove);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue,
tkey_comparer>::removing_template_method::after_remove_inner(
    tkey const &key,
    binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&subtree_root_address,
    std::stack<binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
&path_to_subtree_root_exclusive) {
    UNUSED(key);
    UNUSED(subtree_root_address);
    UNUSED(path_to_subtree_root_exclusive);
}

```

```

// endregion implementation
// endregion template methods
// region construction, assignment, destruction implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::binary_search_tree(
    binary_search_tree::insertion_template_method *insertion,
    binary_search_tree::reading_template_method *reading,
    binary_search_tree::removing_template_method *removing,
    allocating::memory *allocator,
    logging::logger *logger,
    tree_node *root)
    : _insertion(insertion),
      _reading(reading),
      _removing(removing),
      _allocator(allocator),
      _logger(logger),
      _root(root) {
    safe_log("Tree is created", logging::logger::severity::debug);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::binary_search_tree(
    allocating::memory *allocator,
    logging::logger *logger):
    _insertion(new insertion_template_method(this)),
    _reading(new reading_template_method(this)),
    _removing(new removing_template_method(this)),
    _allocator(allocator),
    _logger(logger),
    _root(nullptr) {
    safe_log("Tree is created", logging::logger::severity::debug);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::binary_search_tree(
    const binary_search_tree &other)
{
    UNUSED(other);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::binary_search_tree(
    binary_search_tree &&other) noexcept
{
    UNUSED(other);
}

```

```

}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer> &binary_search_tree<tkey, tvalue,
tkey_comparer>::operator=(
    const binary_search_tree &other)
{
    UNUSED(other);
    // if (this == &other)
    // {
    //     return *this;
    // }
    // return *this;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer> &binary_search_tree<tkey, tvalue,
tkey_comparer>::operator=(
    binary_search_tree &&other) noexcept
{
    UNUSED(other);
    // if (this == &other)
    // {
    //     return *this;
    // }
    // return *this;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
binary_search_tree<tkey, tvalue, tkey_comparer>::~~binary_search_tree() {
    safe_log("[BST] Deallocating tree", logging::logger::severity::debug);
    auto it_end = end_postfix();
    for (auto it = begin_postfix(); it != it_end; ++it) {
        (it.get_node_pointer()->~tree_node());
        safe_deallocate(it.get_node_pointer());
    }
    safe_log("Deallocation complited", logging::logger::severity::debug);
    delete _insertion;
    delete _reading;
    delete _removing;
}
// endregion construction, assignment, destruction implementation
// region associative_container contract implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::insert(

```

```

    tkey const &key,
    tvalue &&value) {
    // safe_log("Start inserting {" + cast_to_str(key) + ", " + cast_to_str(value) + "}",
logging::logger::severity::debug);
    return _insertion->insert(key, std::move(value), _root);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
tvalue const &binary_search_tree<tkey, tvalue, tkey_comparer>::get(
    tkey const &key) {
    // safe_log("Start getting the node with key " + cast_to_str(key),
logging::logger::severity::debug);
    return _reading->read(key, _root);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
tvalue &&binary_search_tree<tkey, tvalue, tkey_comparer>::remove(
    tkey const &key) {
    // safe_log("Start removing the node with key " + cast_to_str(key),
logging::logger::severity::debug);
    return std::move(_removing->remove(key, _root));
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
bool binary_search_tree<tkey, tvalue, tkey_comparer>::find(
    tkey const &key,
    std::pair<tkey, tvalue *> *found) {
    // safe_log("Start finding the node with key " + cast_to_str(key),
logging::logger::severity::debug);
    return _reading->find(key, _root, found);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void binary_search_tree<tkey, tvalue, tkey_comparer>::find_left_bound(
    tkey const &key,
    std::stack<tree_node *> &path_to_subtree_root_exclusive,
    binary_search_tree::tree_node *&node_need) {
    // safe_log("Start finding the node with key " + cast_to_str(key),
logging::logger::severity::debug);
    return _reading->find_left_bound(key, _root, path_to_subtree_root_exclusive, node_need);
}
// endregion associative_container contract implementation
// region logger_holder contract implementation
template<
    typename tkey,
    typename tvalue,

```



```

    typename tkey_comparer>
logging::logger *binary_search_tree<tkey, tvalue, tkey_comparer>::get_logger() const {
    return _logger;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
logging::logger *binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method::get_logger() const {
    return _tree->_logger;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
logging::logger *binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method::get_logger() const {
    return _tree->_logger;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
logging::logger *binary_search_tree<tkey, tvalue,
tkey_comparer>::removing_template_method::get_logger() const {
    return _tree->_logger;
}
// endregion logger_holder contract implementation
// region allocator_holder contract implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
allocating::memory *binary_search_tree<tkey, tvalue, tkey_comparer>::get_allocator() const {
    return _allocator;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
allocating::memory *binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method::get_allocator() const {
    return _tree->_allocator;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
allocating::memory *binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method::get_allocator() const {
    return _tree->_allocator;
}
template<

```

```

        typename tkey,
        typename tvalue,
        typename tkey_comparer>
allocating::memory *binary_search_tree<tkey, tvalue,
tkey_comparer>::removing_template_method::get_allocator() const {
    return _tree->_allocator;
}
// endregion allocator_holder contract implementation
// region iterators requesting implementation
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::begin_prefix() const noexcept {
    return prefix_iterator(_root, _root);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::prefix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::end_prefix() const noexcept {
    return prefix_iterator(_root, nullptr);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::begin_infix() const noexcept {
    return infix_iterator(_root, _root);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::infix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::end_infix() const noexcept {
    return infix_iterator(_root, nullptr);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
typename binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::begin_postfix() const noexcept {
    return postfix_iterator(_root, _root);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>

```

```

typename binary_search_tree<tkey, tvalue, tkey_comparer>::postfix_iterator
binary_search_tree<tkey, tvalue, tkey_comparer>::end_postfix() const noexcept {
    return postfix_iterator(_root, nullptr);
}
// // endregion iterators requesting implementation
#endif //SANDBOX_CPP_BINARY_SEARCH_TREE_H

```

## associative\_container.h

```

#ifndef SANDBOX_CPP_ASSOCIATIVE_CONTAINER_H
#define SANDBOX_CPP_ASSOCIATIVE_CONTAINER_H
#include <sstream>
#include <utility>
template<
    typename tkey,
    typename tvalue>
class associative_container
{
public:
    virtual ~associative_container() = default;
public:
    virtual void insert(
        tkey const &key,
        tvalue &&value) = 0;
    virtual tvalue const &get(
        tkey const &key) = 0;
    virtual tvalue&& remove(
        tkey const &key) = 0;
    virtual bool find(
        tkey const &key,
        std::pair<tkey, tvalue *> *) = 0;
    virtual void print_notes_between(std::stringstream &out_stream, tkey left_bound, tkey
right_bound) = 0;
    virtual void print_container() const = 0;
    virtual void print_container_logger() const = 0;
};
#endif //SANDBOX_CPP_ASSOCIATIVE_CONTAINER_H

```

## avl.h

```

#ifndef AVL_TREE_H
#define AVL_TREE_H
#include "../11/source/tree/binary_search_tree.h"
#include <cstdint>
#include <exception>
#include <iostream>
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
class avl_tree:
    public binary_search_tree<tkey, tvalue, tkey_comparer> {
public:

```

```

explicit avl_tree(
    allocating::memory *allocator = nullptr,
    logging::logger *logger = nullptr);
avl_tree(
    avl_tree const &other);
avl_tree(
    avl_tree &&other) noexcept;
avl_tree &operator=(
    avl_tree const &other);
avl_tree &operator=(
    avl_tree &&other) noexcept;
~avl_tree();
protected:
    struct avl_tree_node : public binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
    {
        size_t height = 0;
        ~avl_tree_node() {
            std::cout << "avl_tree_node destructor\n";
        }
    };
private:
    size_t get_height(typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
    *node) const;
    int balance_factor(typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
    *node) const;
    void fix_height(typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
    **node);
    void print_container() const override;
protected:
    class insertion_template_method_avl : public binary_search_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method {
    private:
        avl_tree<tkey, tvalue, tkey_comparer> *_tree;
    public:
        explicit insertion_template_method_avl(avl_tree<tkey, tvalue, tkey_comparer> *tree);
    protected:
        void after_insert_inner(
            tkey const &key,
            typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root_address,
            std::stack<typename binary_search_tree<tkey, tvalue,
tkey_comparer>::tree_node **> &path_to_subtree_root_exclusive) override;
    private:
        size_t get_size_node() const override;
        void initialize_new_node(typename binary_search_tree<tkey, tvalue,
tkey_comparer>::tree_node *&new_node, tkey const &key, tvalue &&value) const override;
    };
    class reading_template_method_avl final : public binary_search_tree<tkey, tvalue,
tkey_comparer>::reading_template_method {
    public:
        explicit reading_template_method_avl(avl_tree<tkey, tvalue, tkey_comparer> *tree);
    private:
        avl_tree<tkey, tvalue, tkey_comparer> *_tree;
    protected:

```

```

        void after_read_inner(
            tkey const &key,
            typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root_address,
            std::stack<typename binary_search_tree<tkey, tvalue,
tkey_comparer>::tree_node **> &path_to_subtree_root_exclusive) override;
    };
    class removing_template_method_avl final : public binary_search_tree<tkey, tvalue,
tkey_comparer>::removing_template_method {
    public:
        explicit removing_template_method_avl(avl_tree<tkey, tvalue, tkey_comparer> *tree);
    private:
        avl_tree<tkey, tvalue, tkey_comparer> *_tree;
    protected:
        void after_remove_inner(
            tkey const &key,
            typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root_address,
            std::stack<typename binary_search_tree<tkey, tvalue,
tkey_comparer>::tree_node **> &path_to_subtree_root_exclusive) override;
    };
};
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void avl_tree<tkey, tvalue, tkey_comparer>::print_container() const {
    std::function<void(typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*, size_t)> print_tree;
    print_tree = [&](typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root, size_t deep) {
        if (deep == 0) {
            std::cout << "\nTree =====\n\n";
        }
        if (nullptr != subtree_root) {
            print_tree(subtree_root->right_subtree_address, deep + 1);
        }
        std::string s;
        for (size_t i = 0; i < deep; ++i) {
            s += "\t";
        }
        s += "_____";
        if (subtree_root == nullptr) {
            s += "NULL (" + cast_to_str(get_height(subtree_root)) + ")\n";
            std::cout << s;
            return;
        }
        std::cout << s << subtree_root->key << " (" << get_height(subtree_root) << ")" <<
std::endl;
        print_tree(subtree_root->left_subtree_address, deep + 1);
    };
    print_tree(this->_root, 0);
    std::cout << "\nTree end =====\n";
}

```

```

template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
size_t avl_tree<tkey, tvalue, tkey_comparer>::insertion_template_method_avl::get_size_node()
const {
    return sizeof(avl_tree_node);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void avl_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method_avl::initialize_new_node(
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *&new_node, tkey
const &key, tvalue &&value) const {
    new (new_node) typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node {key,
std::move(value), nullptr, nullptr};
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
size_t avl_tree<tkey, tvalue, tkey_comparer>::get_height(
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *node) const {
    this->safe_log("Get height " + cast_to_str(node ? reinterpret_cast<avl_tree_node
*>(node)->height : 0), logging::logger::severity::trace);
    return nullptr != node ? reinterpret_cast<avl_tree_node *>(node)->height : 0;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
int avl_tree<tkey, tvalue, tkey_comparer>::balance_factor(
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node *node) const {
    if (node) {
        return get_height(node->left_subtree_address) -
get_height(node->right_subtree_address);
    }
    return 0;
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void avl_tree<tkey, tvalue, tkey_comparer>::fix_height(
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **node) {
    this->safe_log("_____Fix height_____", logging::logger::severity::debug);
    size_t left;
    size_t right;
    if (nullptr == *node) {
        this->safe_log("nullptr in fix height", logging::logger::severity::debug);
        left = right = 0;
    } else {

```

```

        left = get_height((*node)->left_subtree_address);
        right = get_height((*node)->right_subtree_address);
    }
    this->safe_log("Fix height left, right {" + cast_to_str(left) + ", " +
cast_to_str(right) + "}", logging::logger::severity::debug);
    if (nullptr != *node) {
        reinterpret_cast<avl_tree_node *>(*node)->height = std::max(left, right) + 1;
        this->safe_log("Now height is " + cast_to_str(reinterpret_cast<avl_tree_node
*>(*node)->height),
            logging::logger::severity::debug);
    }
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
avl_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method_avl::insertion_template_method_avl(
    avl_tree<tkey, tvalue, tkey_comparer> *tree) :
    binary_search_tree<tkey, tvalue, tkey_comparer>::insertion_template_method(tree),
    _tree(tree)
{
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
avl_tree<tkey, tvalue,
tkey_comparer>::reading_template_method_avl::reading_template_method_avl(
    avl_tree<tkey, tvalue, tkey_comparer> *tree) :
    binary_search_tree<tkey, tvalue, tkey_comparer>::reading_template_method(tree),
    _tree(tree)
{
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
avl_tree<tkey, tvalue,
tkey_comparer>::removing_template_method_avl::removing_template_method_avl(
    avl_tree<tkey, tvalue, tkey_comparer> *tree) :
    binary_search_tree<tkey, tvalue, tkey_comparer>::removing_template_method(tree),
    _tree(tree)
{
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void avl_tree<tkey, tvalue,
tkey_comparer>::insertion_template_method_avl::after_insert_inner(
    tkey const &key,
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root_address,

```

```

    std::stack<typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
    &path_to_subtree_root_exclusive) {
        UNUSED(key);
        UNUSED(path_to_subtree_root_exclusive);
        _tree->safe_log(cast_to_str(subtree_root_address), logging::logger::severity::debug);
        typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **parent = nullptr;
        if (!path_to_subtree_root_exclusive.empty()) {
            parent = path_to_subtree_root_exclusive.top();
        }
        _tree->fix_height(&subtree_root_address);
        int balance = _tree->balance_factor(subtree_root_address);
        _tree->safe_log("balance factor has ended", logging::logger::severity::debug);
        _tree->safe_log("Balance factor " + cast_to_str(balance),
logging::logger::severity::debug);
        if (balance == 2) {
            if (_tree->balance_factor(subtree_root_address->left_subtree_address) == -1)
            {
                _tree->safe_log("BIG rotation right rotation",
logging::logger::severity::debug);
                _tree->left_rotation(&(subtree_root_address->left_subtree_address),
&subtree_root_address);

                _tree->fix_height(&(subtree_root_address->left_subtree_address->left_subtree_address));
                _tree->fix_height(&(subtree_root_address->left_subtree_address));
            }
            _tree->safe_log("right rotation", logging::logger::severity::debug);
            _tree->right_rotation(&subtree_root_address, parent);
            _tree->fix_height(&(subtree_root_address->right_subtree_address));
            _tree->fix_height(&subtree_root_address);
        } else if (balance == -2) {
            if (_tree->balance_factor(subtree_root_address->right_subtree_address) == 1)
            {
                _tree->safe_log("BIG rotation right rotation",
logging::logger::severity::debug);
                _tree->right_rotation(&(subtree_root_address->right_subtree_address),
&subtree_root_address);

                _tree->fix_height(&(subtree_root_address->right_subtree_address->right_subtree_address));
                _tree->fix_height(&(subtree_root_address->right_subtree_address));
            }
            _tree->safe_log("left rotation", logging::logger::severity::debug);
            _tree->left_rotation(&subtree_root_address, parent);
            _tree->fix_height(&(subtree_root_address->left_subtree_address));
            _tree->fix_height(&subtree_root_address);
        }
        _tree->safe_log("After insert inner ended", logging::logger::severity::debug);
    }
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void avl_tree<tkey, tvalue,
tkey_comparer>::removing_template_method_avl::after_remove_inner(
    tkey const &key,

```



```

    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root_address,
    std::stack<typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
&path_to_subtree_root_exclusive) {
    UNUSED(key);
    UNUSED(path_to_subtree_root_exclusive);
    if (nullptr == subtree_root_address) {
        return;
    }
    // _tree->safe_log("____After remove inner____ key removing " + cast_to_str(key),
logging::logger::severity::debug);
    _tree->safe_log(cast_to_str(subtree_root_address), logging::logger::severity::debug);
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **parent = nullptr;
    if (!path_to_subtree_root_exclusive.empty()) {
        parent = path_to_subtree_root_exclusive.top();
    }
    // _tree->safe_log("key: " + cast_to_str(subtree_root_address->key),
logging::logger::severity::debug);
    _tree->fix_height(&subtree_root_address);
    int balance = _tree->balance_factor(subtree_root_address);
    _tree->safe_log("Balance factor " + cast_to_str(balance),
logging::logger::severity::debug);
    if (balance == 2) {
        // std::cout << "before:\n";
        _tree->print_container();
        if (_tree->balance_factor(subtree_root_address->left_subtree_address) == -1)
        {
            _tree->safe_log("BIG left rotation", logging::logger::severity::debug);
            _tree->left_rotation(&(subtree_root_address->left_subtree_address),
&subtree_root_address);

            _tree->fix_height(&(subtree_root_address->left_subtree_address->left_subtree_address));
            _tree->fix_height(&(subtree_root_address->left_subtree_address));
            // std::cout << "half way:\n";
            _tree->print_container();
        }
        _tree->safe_log("right rotation", logging::logger::severity::debug);
        _tree->right_rotation(&subtree_root_address, parent);
        _tree->fix_height(&(subtree_root_address->right_subtree_address));
        _tree->fix_height(&subtree_root_address);
        // std::cout << "after:\n";
        _tree->print_container();
    } else if (balance == -2) {
        // std::cout << "before:\n";
        _tree->print_container();
        if (_tree->balance_factor(subtree_root_address->right_subtree_address) == 1)
        {
            _tree->safe_log("BIG right rotation", logging::logger::severity::debug);
            _tree->right_rotation(&(subtree_root_address->right_subtree_address),
&subtree_root_address);

            _tree->fix_height(&(subtree_root_address->right_subtree_address->right_subtree_address));
            _tree->fix_height(&(subtree_root_address->right_subtree_address));
            // std::cout << "half way:\n";

```

```

        _tree->print_container();
    }
    _tree->safe_log("left rotation", logging::logger::severity::debug);
    _tree->left_rotation(&subtree_root_address, parent);
    _tree->fix_height(&(subtree_root_address->left_subtree_address));
    _tree->fix_height(&subtree_root_address);
    // std::cout << "after:\n";
    _tree->print_container();
}
_tree->safe_log("After remove inner ended", logging::logger::severity::debug);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
void avl_tree<tkey, tvalue, tkey_comparer>::reading_template_method_avl::after_read_inner(
    tkey const &key,
    typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node
*&subtree_root_address,
    std::stack<typename binary_search_tree<tkey, tvalue, tkey_comparer>::tree_node **>
&path_to_subtree_root_exclusive) {
    UNUSED(key);
    UNUSED(subtree_root_address);
    UNUSED(path_to_subtree_root_exclusive);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
avl_tree<tkey, tvalue, tkey_comparer>::avl_tree(
    allocating::memory *allocator,
    logging::logger *logger) :
    binary_search_tree<tkey, tvalue, tkey_comparer>(
        new insertion_template_method_avl(this),
        new reading_template_method_avl(this),
        new removing_template_method_avl(this),
        allocator,
        logger,
        nullptr) {
    this->safe_log("AVL tree is created", logging::logger::severity::debug);
}
template<
    typename tkey,
    typename tvalue,
    typename tkey_comparer>
avl_tree<tkey, tvalue, tkey_comparer>::~~avl_tree() {
}
#endif

```

## safe\_allocator.h

```

#ifndef SAFE_ALLOCATOR_H
#define SAFE_ALLOCATOR_H

```

```

#include "../../../../../memory/2/source/memory/memory.h"
#include <cstdint>
namespace allocating {
class safe_allocator
{
public:
    virtual ~safe_allocator() = default;
    void *safe_allocate(size_t) const;
    void safe_deallocate(void *) const;
    virtual allocating::memory *get_allocator() const = 0;
};
}
#endif

```

## safe\_allocator.cpp

```

#include "safe_allocator.h"
#include <iostream>
void *allocating::safe_allocator::safe_allocate(size_t size) const {
    allocating::memory *allocator = get_allocator();
    if (nullptr != allocator) {
        return allocator->allocate(size);
    }
    return ::operator new(size);
}
void allocating::safe_allocator::safe_deallocate(void *object) const {
    allocating::memory *allocator = get_allocator();
    if (nullptr != allocator) {
        allocator->deallocate(object);
    } else {
        ::operator delete(object);
    }
}

```

## memory\_with\_descriptors.cpp

```

#include "memory_with_descriptors.h"
#include <cstdint>
#include <stdexcept>
#include <string>
#include <sstream>
template <class T>
std::string cast_to_str(const T& object) {
    std::stringstream ss;
    ss << object;
    return ss.str();
}
void allocating::memory_with_descriptors::deallocate(void * const target_to_dealloc) const {
    safe_log("Deallocating.....", logging::logger::severity::trace);
    if (target_to_dealloc == trusted_memory_to_block()) {
        memory *outer_allocator = get_outer_allocator();
        outer_allocator->deallocate(target_to_dealloc);
        return;
    }
}

```

```

    }
    safe_log("Prev " + cast_to_str(get_pointer_previous(target_to_dealloc)),
logging::logger::severity::trace);
    void *next = *get_pointer_next(target_to_dealloc);
    void *prev = get_pointer_previous(target_to_dealloc);
    safe_log("Prev " + cast_to_str(prev), logging::logger::severity::trace);
    safe_log("Next " + cast_to_str(next), logging::logger::severity::trace);
    safe_log("Outer next " + cast_to_str(*get_pointer_next(trusted_memory_to_block())),
logging::logger::severity::trace);
    safe_log("Outer end " + cast_to_str(get_end_allocator()),
logging::logger::severity::trace);
    *(reinterpret_cast<void **>(prev) - 1) = next;
    safe_log("Prev " + cast_to_str(prev), logging::logger::severity::trace);
    safe_log("Curr " + cast_to_str(target_to_dealloc), logging::logger::severity::trace);
    safe_log("Next " + cast_to_str(next), logging::logger::severity::trace);
    safe_log("Outer next " + cast_to_str(*get_pointer_next(trusted_memory_to_block())),
logging::logger::severity::trace);
    safe_log("Outer end " + cast_to_str(get_end_allocator()),
logging::logger::severity::trace);
    safe_log("Deallocation completed!", logging::logger::severity::trace);
}
logging::logger *allocating::memory_with_descriptors::get_logger() const {
    return *reinterpret_cast<logging::logger **>(_trusted_memory);
}
void allocating::memory_with_descriptors::clear_logger() const {
    *reinterpret_cast<logging::logger **>(_trusted_memory) = nullptr;
}
allocating::memory_with_descriptors::~memory_with_descriptors() {
    safe_log("Allocator is destroyed", logging::logger::severity::debug);
    clear_logger();
    if (get_outer_allocator() == nullptr) {
        ::operator delete(_trusted_memory);
    } else {
        deallocate(_trusted_memory);
    }
}
std::string allocating::memory_with_descriptors::get_bytes(const void * const memory) {
    size_t size = get_size_block(memory) + get_size_service_block_block();
    unsigned char *ptr = reinterpret_cast<unsigned char *>(const_cast<void *>(memory)) - 3;
    std::stringstream ss;
    while (size--) {
        ss << static_cast<int>(*(ptr++)) << ' ';
    }
    return ss.str();
}
void *allocating::memory_with_descriptors::get_memory_pointer() {
    return _trusted_memory;
}
void *allocating::memory_with_descriptors::allocate(const size_t target_size) const {
    safe_log("New memory allocating.....",
logging::logger::severity::trace);
    void *new_memory = allocate_fit(target_size + get_size_service_block_block(), _fit);
    if (new_memory == nullptr) {
        throw std::runtime_error("There is not enough space for memory allocation!");
    }
}

```

```

    }
    safe_log("Allocated block " + std::to_string(target_size) + " bytes of memory at "
            + cast_to_str(new_memory), logging::logger::severity::debug);
    safe_log("ALLOCATOR " + cast_to_str(_trusted_memory), logging::logger::severity::trace);
    safe_log("Starts at      " + cast_to_str(_trusted_memory),
logging::logger::severity::trace);
    safe_log("New memory at " + cast_to_str(new_memory), logging::logger::severity::trace);
    safe_log("Ends at      " + cast_to_str(get_end_allocator()),
logging::logger::severity::trace);
    return new_memory;
}
allocating::memory_with_descriptors::memory_with_descriptors(
    size_t size, memory *outer_allocator, logging::logger *memory_logger,
    allocating::memory_with_descriptors::fit_type fit =
allocating::memory_with_descriptors::fit_type::first
) : _fit { fit } {
    _trusted_memory = outer_allocator == nullptr ?
        ::operator new(size + get_size_service_block_allocator()) :
        outer_allocator->allocate(size + get_size_service_block_allocator());
    if (_trusted_memory == nullptr) {
        safe_log("Cannot allocate " + cast_to_str(size) + " bytes of memory",
logging::logger::severity::error);
        throw std::runtime_error("There is not enough space for allocator!");
    }
    logging::logger **logger_ptr = reinterpret_cast<logging::logger **>(_trusted_memory);
    *logger_ptr = memory_logger;
    safe_log("Allocated allocator with " + cast_to_str(size) + " bytes of memory at " +
cast_to_str(_trusted_memory),
        logging::logger::severity::debug);
    size_t *pointer_size = reinterpret_cast<size_t *>(_trusted_memory) + 2;
    *pointer_size = size;
    void **previous = reinterpret_cast<void **>(_trusted_memory) + 3;
    *previous = nullptr;
    void **first_block = reinterpret_cast<void **>(_trusted_memory) + 4;
    *first_block = reinterpret_cast<unsigned char *>(_trusted_memory) + size +
get_size_service_block_allocator();
    safe_log("Memory of new Allocator", logging::logger::severity::trace);
    safe_log("Start at " + cast_to_str(_trusted_memory), logging::logger::severity::trace);
    safe_log("New a at " + cast_to_str(_trusted_memory), logging::logger::severity::trace);
    safe_log(" End at " + cast_to_str(*first_block), logging::logger::severity::trace);
    if (outer_allocator == nullptr) {
        set_outer_allocator(nullptr);
        unsigned char *ptr = reinterpret_cast<unsigned char *>(reinterpret_cast<size_t
*>(_trusted_memory) + 5);
        for (size_t i = 0; i < size; ++i) {
            *(ptr++) = 0;
        }
    } else {
        set_outer_allocator(outer_allocator);
    }
}
void *allocating::memory_with_descriptors::allocate_fit(size_t size,
allocating::memory_with_descriptors::fit_type fit =
allocating::memory_with_descriptors::fit_type::first) const {

```

```

    void *fit_memory_block;
    safe_log("Start finding memory block for allocator with size " + cast_to_str(size) + "
bytes",
            logging::logger::severity::debug);
    if (fit == allocating::memory_with_descriptors::fit_type::first) {
        fit_memory_block = allocating::memory_with_descriptors::find_first_fit(size);
    } else if (fit == allocating::memory_with_descriptors::fit_type::best) {
        fit_memory_block = allocating::memory_with_descriptors::find_best_fit(size);
    } else {
        fit_memory_block = allocating::memory_with_descriptors::find_worst_fit(size);
    }
    if (fit_memory_block == nullptr) {
        safe_log("There is no memory for allocator with size " + cast_to_str(size),
logging::logger::severity::error);
        return nullptr;
    } else {
        safe_log(cast_to_str(size) + " bytes of memory is allocated",
logging::logger::severity::trace);
    }
    set_size_block(fit_memory_block, size);
    insert_block_to_pointer_list(fit_memory_block);
    return fit_memory_block;
}

void allocating::memory_with_descriptors::insert_block_to_pointer_list(void *block) const {
    void *ptr_next = trusted_memory_to_block();
    void *ptr_prev = ptr_next;
    safe_log("cur " + cast_to_str(ptr_next) + " prev " + cast_to_str(ptr_prev) + " block " +
cast_to_str(block),
            logging::logger::severity::trace);
    while (block > ptr_next) {
        ptr_prev = ptr_next;
        ptr_next = *get_pointer_next(ptr_next);
        safe_log("cur " + cast_to_str(ptr_next) + " prev " + cast_to_str(ptr_prev) + " block
" + cast_to_str(block),
            logging::logger::severity::trace);
    }
    *(reinterpret_cast<void **>(block) - 1) = ptr_next;
    *(reinterpret_cast<void **>(ptr_prev) - 1) = block;
    if (ptr_next != get_end_allocator()) {
        *(reinterpret_cast<void **>(ptr_next) - 2) = block;
    }
    *(reinterpret_cast<void **>(block) - 2) = ptr_prev;
}

void allocating::memory_with_descriptors::set_pointer_to_next_block(void *block, void
*pointer) {
    *(reinterpret_cast<void **>(block) - 1) = pointer;
}

void *allocating::memory_with_descriptors::find_first_fit(size_t size) const {
    void **ptr_next = get_pointer_next(trusted_memory_to_block());
    void *current_memory = reinterpret_cast<void *>(reinterpret_cast<void
**>(_trusted_memory) + 5);
    void **ptr_current = &current_memory;
    void *ptr_end = get_end_allocator();
    void **fit = nullptr;

```

```

size_t free_space;
safe_log("Start jumping between memory blocks", logging::logger::severity::trace);
safe_log(">>>>>ptr_current: " + cast_to_str(*ptr_current),
logging::logger::severity::trace);
safe_log("ptr_next: " + cast_to_str(*ptr_next), logging::logger::severity::trace);
safe_log("ptr_end: " + cast_to_str(ptr_end), logging::logger::severity::trace);
if (*ptr_next == ptr_end) {
    free_space = get_space_between(*ptr_current, *ptr_next) +
get_size_block(*ptr_current);
    safe_log("First block! Space for data between " + cast_to_str(*ptr_current) + " and
" + cast_to_str(ptr_end) +
        " is " + cast_to_str(free_space), logging::logger::severity::trace);
    if (free_space >= size) {
        current_memory = reinterpret_cast<void *>(reinterpret_cast<size_t
*>(*ptr_current) + 3);
        fit = &current_memory;
    } else {
        ptr_current = get_pointer_next(*ptr_current);
    }
} else {
    while (*ptr_current < ptr_end) {
        free_space = get_space_between(*ptr_current, *ptr_next);
        safe_log("Space for data between " + cast_to_str(*ptr_current) + " and " +
cast_to_str(*ptr_next) +
            " is " + cast_to_str(free_space),
                logging::logger::severity::trace);
        safe_log("Size need " + cast_to_str(size), logging::logger::severity::trace);
        if (free_space >= size) {
            current_memory = reinterpret_cast<void *>(reinterpret_cast<unsigned char
*>(*ptr_current) +
                get_size_block(*ptr_current) + sizeof(size_t *) + sizeof(void **) * 2);
            fit = &current_memory;
            break;
        }
        ptr_current = get_pointer_next(*ptr_current);
        ptr_next = get_pointer_next(*ptr_current);
        safe_log("ptr_next: " + cast_to_str(*ptr_next),
logging::logger::severity::trace);
        safe_log("ptr_current: " + cast_to_str(*ptr_current),
logging::logger::severity::trace);
        safe_log("ptr_end: " + cast_to_str(ptr_end), logging::logger::severity::trace);
    }
}
if (fit == nullptr) {
    safe_log("Cannot find memory block with " + cast_to_str(size) + " bytes of memory
(first fit)",
        logging::logger::severity::error);
    return nullptr;
} else {
    safe_log("Found first fit block with size " + cast_to_str(size) + " at " +
cast_to_str(*fit), logging::logger::severity::debug);
}
safe_log("_allocated_memory " + cast_to_str(_trusted_memory),
logging::logger::severity::trace);

```

```

    safe_log("block fit " + cast_to_str(*fit), logging::logger::severity::trace);
    return *fit;
}
void *allocating::memory_with_descriptors::find_best_fit(size_t size) const {
    void **ptr_next = get_pointer_next(trusted_memory_to_block());
    void *current_memory = reinterpret_cast<void *>(reinterpret_cast<void
**>(_trusted_memory) + 5);
    void **ptr_current = &current_memory;
    void **fit = nullptr;
    size_t free_space;
    safe_log("Start jumping beetween memory blocks", logging::logger::severity::trace);
    void *ptr_end = get_end_allocator();
    if (*ptr_next == ptr_end) {
        free_space = get_space_beetween(*ptr_current, *ptr_next) +
get_size_block(*ptr_current);
        safe_log("First block! Space for data beetween " + cast_to_str(*ptr_current) + " and
" + cast_to_str(ptr_end) +
            " is " + cast_to_str(free_space), logging::logger::severity::trace);
        if (free_space >= size) {
            current_memory = reinterpret_cast<void *>(reinterpret_cast<size_t
**>(*ptr_current) + 3);
            fit = &current_memory;
        } else {
            ptr_current = get_pointer_next(*ptr_current);
        }
    } else {
        size_t best_size = -1;
        while (*ptr_current < ptr_end) {
            free_space = get_space_beetween(*ptr_current, *ptr_next);
            safe_log("Space for data beetween " + cast_to_str(*ptr_current) + " and " +
cast_to_str(*ptr_next) +
                " is " + cast_to_str(free_space),
                logging::logger::severity::trace);
            safe_log("Size need " + cast_to_str(size), logging::logger::severity::trace);
            if (free_space >= size) {
                if (best_size > free_space) {
                    current_memory = reinterpret_cast<void *>(reinterpret_cast<unsigned char
**>(*ptr_current) +
                        get_size_block(*ptr_current) + sizeof(size_t *) + sizeof(void **) *
2);
                    fit = &current_memory;
                    best_size = free_space;
                }
            }
            ptr_current = get_pointer_next(*ptr_current);
            ptr_next = get_pointer_next(*ptr_current);
            safe_log("ptr_current: " + cast_to_str(*ptr_current),
logging::logger::severity::trace);
            safe_log("prt_end: " + cast_to_str(ptr_end), logging::logger::severity::trace);
        }
    }
    if (fit == nullptr) {
        safe_log("Cannot find memory block with " + cast_to_str(size) + " bytes of memory
(best fit)",

```



```

        logging::logger::severity::error);
    return nullptr;
} else {
    safe_log("Found first fit block with size " + cast_to_str(size) + " at " +
cast_to_str(*fit), logging::logger::severity::debug);
}
    safe_log("_allocated_memory " + cast_to_str(_trusted_memory),
logging::logger::severity::trace);
    safe_log("block fit " + cast_to_str(*fit), logging::logger::severity::trace);
    return *fit;
}

void *allocating::memory_with_descriptors::find_worst_fit(size_t size) const {
    void **ptr_next = get_pointer_next(trusted_memory_to_block());
    void *current_memory = reinterpret_cast<void *>(reinterpret_cast<void
**>(_trusted_memory) + 5);
    void **ptr_current = &current_memory;
    void **fit = nullptr;
    size_t free_space;
    safe_log("Start jumping beetween memory blocks", logging::logger::severity::trace);
    void *ptr_end = get_end_allocator();
    if (*ptr_next == ptr_end) {
        free_space = get_space_beetween(*ptr_current, *ptr_next) +
get_size_block(*ptr_current);
        safe_log("First block! Space for data beetween " + cast_to_str(*ptr_current) + " and
" + cast_to_str(ptr_end) +
            " is " + cast_to_str(free_space), logging::logger::severity::trace);
        if (free_space >= size) {
            current_memory = reinterpret_cast<void *>(reinterpret_cast<size_t
**>(*ptr_current) + 3);
            fit = &current_memory;
        } else {
            ptr_current = get_pointer_next(*ptr_current);
        }
    } else {
        size_t worst_size = 0;
        while (*ptr_current < ptr_end) {
            free_space = get_space_beetween(*ptr_current, *ptr_next);
            safe_log("Space for data beetween " + cast_to_str(*ptr_current) + " and " +
cast_to_str(*ptr_next) +
                " is " + cast_to_str(free_space),
                logging::logger::severity::trace);
            safe_log("Size need " + cast_to_str(size), logging::logger::severity::trace);
            if (free_space >= size) {
                if (worst_size < free_space) {
                    current_memory = reinterpret_cast<void *>(reinterpret_cast<unsigned char
**>(*ptr_current) +
                        get_size_block(*ptr_current) + sizeof(size_t *) + sizeof(void **) *
2);
                    fit = &current_memory;
                    worst_size = free_space;
                }
            }
            ptr_current = get_pointer_next(*ptr_current);
            ptr_next = get_pointer_next(*ptr_current);

```

```

        safe_log("ptr_current: " + cast_to_str(*ptr_current),
logging::logger::severity::trace);
        safe_log("prt_end: " + cast_to_str(ptr_end), logging::logger::severity::trace);
    }
}
if (fit == nullptr) {
    safe_log("Cannot find memory block with " + cast_to_str(size) + " bytes of memory
(worst fit)",
        logging::logger::severity::error);
    return nullptr;
} else {
    safe_log("Found first fit block with size " + cast_to_str(size) + " at " +
cast_to_str(*fit), logging::logger::severity::debug);
}
    safe_log("_allocated_memory " + cast_to_str(_trusted_memory),
logging::logger::severity::trace);
    safe_log("block fit " + cast_to_str(*fit), logging::logger::severity::trace);
    return *fit;
}
size_t allocating::memory_with_descriptors::get_space_beetween(void *ptr_current, void
*ptr_next) {
    unsigned char *ptr_1_casted = reinterpret_cast<unsigned char *>(ptr_current);
    unsigned char *ptr_2_casted = reinterpret_cast<unsigned char *>(ptr_next);
    if (ptr_2_casted - ptr_1_casted - (long) get_size_block(ptr_current) < (long)
get_size_service_block_block()) {
        return 0;
    }
    return ptr_2_casted - ptr_1_casted - get_size_service_block_block() -
get_size_block(ptr_current);
}
size_t allocating::memory_with_descriptors::get_size_block(const void * const block) {
    return *(reinterpret_cast<size_t *>(const_cast<void *>(block)) - 3);
}
void allocating::memory_with_descriptors::set_size_block(void *block, size_t size) {
    *(reinterpret_cast<size_t *>(block) - 3) = size;
}
void **allocating::memory_with_descriptors::get_pointer_next(const void * const block) {
    return reinterpret_cast<void **>(const_cast<void *>(block)) - 1;
}
void *allocating::memory_with_descriptors::get_pointer_previous(const void * const block)
const {
    void *current = trusted_memory_to_block();
    void *previous = current;
    while (current < block) {
        previous = current;
        current = *get_pointer_next(current);
    }
    return previous;
}
size_t allocating::memory_with_descriptors::get_size_service_block_allocator() {
    return sizeof(size_t) + sizeof(void *) + sizeof(void *) + sizeof(logging::logger **);
}
size_t allocating::memory_with_descriptors::get_size_service_block_block() {
    return sizeof(size_t) + sizeof(void **);
}

```

```

}
std::string allocating::memory_with_descriptors::print_memory(const void *block) {
    std::string s;
    s = "[" + get_bytes(block) + "]";
    return s;
}
std::string allocating::memory_with_descriptors::print_allocator(const memory * const
allocator) {
    memory_with_descriptors *allocator_memory = reinterpret_cast<memory_with_descriptors
*>(const_cast<memory *>(allocator));
    std::stringstream ss;
    ss << "[";
    unsigned char *ptr = reinterpret_cast<unsigned char
*>(allocator_memory->_trusted_memory);
    void *ptr_end = allocator_memory->get_end_allocator();
    while (ptr < ptr_end) {
        ss << static_cast<int>(*(ptr++)) << " ";
    }
    ss << "]";
    return ss.str();
}
std::string allocating::memory_with_descriptors::print_allocator_data(const memory * const
allocator) {
    memory_with_descriptors *allocator_memory = reinterpret_cast<memory_with_descriptors
*>(const_cast<memory *>(allocator));
    std::stringstream ss;
    ss << "[";
    unsigned char *ptr = reinterpret_cast<unsigned char *>(reinterpret_cast<void
**>(allocator_memory->_trusted_memory) + 5);
    void *ptr_end = allocator_memory->get_end_allocator();
    while (ptr < ptr_end) {
        ss << static_cast<int>(*(ptr++)) << " ";
    }
    ss << "]";
    return ss.str();
}
void *allocating::memory_with_descriptors::trusted_memory_to_block() const {
    return reinterpret_cast<void *>(reinterpret_cast<size_t *>(_trusted_memory) + 5);
}
void *allocating::memory_with_descriptors::get_end_allocator() const {
    return reinterpret_cast<void *>(reinterpret_cast<unsigned char *>(_trusted_memory) +
*(reinterpret_cast<size_t *>(_trusted_memory) + 2) +
get_size_service_block_allocator());
}
allocating::memory *allocating::memory_with_descriptors::get_outer_allocator() const {
    return *(reinterpret_cast<allocating::memory **>(_trusted_memory) + 1);
}
void allocating::memory_with_descriptors::set_outer_allocator(memory *allocator) {
    *(reinterpret_cast<memory **>(_trusted_memory) + 1) = allocator;
}

```

## memory\_with\_descriptors.h

```
#ifndef MEMORY_WITH_descriptors_H
```

```

#define MEMORY_WITH_descriptors_H
#include "memory.h"
#include "../../../../../logger/source/logger/complete/complete_logger.h"
#include <cstdint>
namespace allocating {
class memory_with_descriptors final : public allocating::memory, protected
logging::complete_logger {
public:
    memory_with_descriptors(memory_with_descriptors const&) = delete;
    void operator=(memory_with_descriptors const&) = delete;
    memory_with_descriptors() = default;
    memory_with_descriptors(size_t, memory *, logging::logger *, fit_type);
    ~memory_with_descriptors() override;
    void *allocate(size_t) const override;
    void *allocate_fit(size_t, fit_type) const;
    void deallocate(void *) const override;
    static size_t get_size_block(const void *);
    void *get_memory_pointer();
    void *find_first_fit(size_t) const;
    void *find_best_fit(size_t) const;
    void *find_worst_fit(size_t) const;
    static void **get_pointer_next(const void *);
    void *get_pointer_previous(const void *) const;
    static size_t get_size_service_block_allocator();
    static size_t get_size_service_block_block();
    void insert_block_to_pointer_list(void *) const;
    logging::logger *get_logger() const override;
    static std::string print_memory(const void *);
    static std::string print_allocator(const memory *);
    static std::string print_allocator_data(const memory *);
private:
    void clear_logger() const;
    void *trusted_memory_to_block() const;
    static void set_pointer_to_next_block(void *, void *);
    void *get_end_allocator() const;
    static size_t get_space_beetween(void *, void *);
    static void set_size_block(void *, size_t);
    void *_trusted_memory{};
    fit_type _fit {};
    memory *get_outer_allocator() const;
    void set_outer_allocator(memory *);
    static std::string get_bytes(const void *);
};
}
#endif

```

conf.json

```

{
  "streams":
    {
      "info.log": "info",
      "trace.log": "trace",

```

```

        "debug.log": "debug",
        "console": "error"
    }
}

```

## comparer.h

```

#ifndef COMPARER_H
#define COMPARER_H
#include "../data/type_data/type_data.h"
class comparers {
public:
    int operator()(type_key lhs, type_key rhs) {
        if (lhs._user_id < rhs._user_id) {
            return 1;
        } else if (lhs._user_id > rhs._user_id) {
            return -1;
        }
        if (lhs._delivery_id < rhs._delivery_id) {
            return 1;
        } else if (lhs._delivery_id > rhs._delivery_id) {
            return -1;
        }
        return 0;
    }
    int operator()(std::string lhs, std::string rhs) {
        if (lhs < rhs) {
            return 1;
        } else if (lhs > rhs) {
            return -1;
        }
        return 0;
    }
};
#endif

```

## collection.h

```

#include <iostream>
#include <fstream>
#include <string>
#include <regex>
#include "../logger/source/logger/complete/complete_logger.h"
#include "../memory/2/source/memory/memory.h"
#include "../tree/11/source/tree/associative_container.h"
#include "../tree/11/source/allocator/safe_allocator.h"
#include "../comparer/comparer.h"
#include "../tree/12/source/avl/avl.h"
#include "../memory/3/source/memory_with_list/memory_with_list.h"
#include "../memory/4/source/memory_with_descriptors/memory_with_descriptors.h"
class collection : protected allocating::safe_allocator, protected logging::complete_logger
{
    friend class database;

```

```

public:
    explicit collection(allocating::memory *allocator = nullptr, logging::logger *logger =
    nullptr);
    ~collection();
public:
    int create_note(std::stringstream &file, std::vector<std::string> &query);
public:
    int read_note(std::stringstream &file, std::stringstream &out_stream,
    std::vector<std::string> &query);
    int read_note_range(std::stringstream &file, std::stringstream &out_stream,
    std::vector<std::string> &query);
public:
    int delete_note(std::stringstream &file, std::vector<std::string> &query);
private:
    void key_filling(std::stringstream &file, type_key &key) const;
    void value_filling(std::stringstream &file, type_value &value) const;
private:
    associative_container<type_key, type_value> *_notes;
private:
    allocating::memory *get_allocator() const override;
    logging::logger *get_logger() const override;
    allocating::memory *_allocator;
    logging::logger *_logger;
};
inline std::ostream &operator<<(std::ostream &stream, collection const &key) {
    return stream << "[ object collection ]";
}
int get_word(std::stringstream &stream, std::string &word);
int is_date(std::string &date);
int is_time(std::string &time);
int is_digit(char c);
bool is_email_valid(const std::string& email);

```

## collection.cpp

```

#include "collection.h"
allocating::memory *collection::get_allocator() const {
    return _allocator;
}
logging::logger *collection::get_logger() const {
    return _logger;
}
collection::~collection() {
    safe_log("Collection destructor", logging::logger::severity::warning);
    _notes->print_container();
    _notes->~associative_container();
    safe_deallocate(_notes);
}
collection::collection(allocating::memory *allocator, logging::logger *logger)
    : _allocator(allocator), _logger(logger) {
    _notes = reinterpret_cast<avl_tree<type_key, type_value, comparers>
*>(safe_allocate(sizeof(avl_tree<type_key, type_value, comparers>)));
    new (_notes) avl_tree<type_key, type_value, comparers>(allocator, logger);
    safe_log("Collection constructor", logging::logger::severity::warning);
}

```

```

}
int collection::create_note(std::stringstream &file, std::vector<std::string> &query) {
    type_key key;
    key_filling(file, key);
    type_value value;
    value_filling(file, value);
    _notes->insert(key, std::move(value));
    safe_log("Note created", logging::logger::severity::information);
    return 0;
}
// reading
int collection::read_note(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query) {
    type_key key;
    key_filling(file, key);
    type_value value;
    try {
        value = _notes->get(key);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error(cast_to_str(ex.what()) + " " +
            "{ " + cast_to_str(key._user_id) + ", " + cast_to_str(key._delivery_id) + " }");
    }
    out_stream << key << std::endl << value << std::endl;
    return 0;
}
int collection::read_note_range(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query) {
    type_key key_left;
    type_key key_right;
    key_filling(file, key_left);
    key_filling(file, key_right);
    _notes->print_notes_between(out_stream, key_left, key_right);
    return 0;
}
// deleting
int collection::delete_note(std::stringstream &file, std::vector<std::string> &query) {
    type_key key;
    key_filling(file, key);
    try {
        _notes->remove(key);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error(cast_to_str(ex.what()) + " " +
            "{ " + cast_to_str(key._user_id) + ", " + cast_to_str(key._delivery_id) + " }");
    }
    safe_log("Note removed", logging::logger::severity::information);
    return 0;
}
// functions
int is_time(std::string &time) {
    if (time.length() != 5) {
        throw std::runtime_error("Wrong time length! Should be 'mm:hh'");
    }
    int digits[] = { 0, 1, 3, 4 };
    for (int i : digits) {

```

```

        if (!is_digit(time[i])) {
            throw std::runtime_error("It does not a digit '" + cast_to_str(time[i]) + "'");
        }
    }
    if (time[2] != ':' && time[2] != '-') {
        throw std::runtime_error("It does not a separator '" + cast_to_str(time[2]) + "'");
    }
    int hours = 10 * (time[0] - '0') + time[1] - '0';
    int minutes = 10 * (time[3] - '0') + time[4] - '0';
    if (hours < 0 || 23 < hours) {
        throw std::runtime_error("Wrong count of hours '" + cast_to_str(hours) + "'");
    }
    if (minutes < 0 || 59 < minutes) {
        throw std::runtime_error("Wrong count of minutes '" + cast_to_str(hours) + "'");
    }
    return 1;
}

int is_date(std::string &date) {
    if (date.length() != 10) {
        throw std::runtime_error("Wrong date length! Should be 'dd/mm/yy'");
    }
    int digits[] = { 0, 1, 3, 4, 6, 7, 8, 9 };
    for (int i : digits) {
        if (!is_digit(date[i])) {
            throw std::runtime_error("It does not a digit '" + cast_to_str(date[i]) + "'");
        }
    }
    int separators[] = { 2, 5 };
    for (int i : separators) {
        if (date[i] != '/' && date[i] != '.') {
            throw std::runtime_error("Wrong separator '" + cast_to_str(date[i]) + "'");
        }
    }
    int day = 10 * (date[0] - '0') + date[1] - '0';
    int month = 10 * (date[3] - '0') + date[4] - '0';
    int year = 1000 * (date[6] - '0') + 100 * (date[7] - '0') + 10 * (date[8] - '0') +
(date[9] - '0');
    if (day < 1 || day > 31) {
        throw std::runtime_error("Wrong day '" + cast_to_str(day) + "'");
    } else if (month < 1 || month > 12) {
        throw std::runtime_error("Wrong day '" + cast_to_str(month) + "'");
    }
    if ((month == 4 || month == 6 || month == 9 || month == 11) && day == 31) {
        throw std::runtime_error("Wrong count of days in month");
    } else if ((month == 2) && (year % 4 == 0) && day > 29) {
        throw std::runtime_error("Wrong count of days in month");
    } else if ((month == 2) && (year % 4 != 0) && day > 28) {
        throw std::runtime_error("Wrong count of days in month");
    }
    return 1;
}

int is_digit(char c) {
    return '0' <= c && c <= '9';
}

```



```

int get_word(std::stringstream &stream, std::string &word) {
    if (stream >> word) {
        return 1;
    }
    return 0;
}

bool is_email_valid(const std::string& email)
{
    const std::regex pattern("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\. (\\w+))+");
    return std::regex_match(email, pattern);
}

// validation
void collection::key_filling(std::stringstream &file, type_key &key) const {
    std::string word;
    if (!get_word(file, word)) {
        throw std::runtime_error("Expected key");
    }
    if (word != "{") {
        throw std::runtime_error("Expected '{' after delivery_id" + word);
    }
    if (!get_word(file, word)) {
        throw std::runtime_error("Expected user_id");
    }
    key._user_id = 0;
    for (int i = 0; i < word.size() - 1; ++i) {
        if (word[i] < '0' || '9' < word[i]) {
            throw std::runtime_error("Invalid user_id");
        }
        key._user_id = key._user_id * 10 + word[i] - '0';
    }
    if (word[word.size() - 1] != ',') {
        throw std::runtime_error("Expected ',' after user_id");
    }
    if (!get_word(file, word)) {
        throw std::runtime_error("Expected delivery_id");
    }
    key._delivery_id = 0;
    for (int i = 0; i < word.size(); ++i) {
        if (word[i] < '0' || '9' < word[i]) {
            throw std::runtime_error("Invalid delivery_id");
        }
        key._delivery_id = key._delivery_id * 10 + word[i] - '0';
    }
    if (!get_word(file, word)) {
        throw std::runtime_error("Unexpected end of file");
    }
    if (word != "}") {
        throw std::runtime_error("Expected '}' after delivery_id");
    }
}

void collection::value_filling(std::stringstream &file, type_value &value) const {
    std::string word;
    // description
    if (!get_word(file, word)) {

```

```

        throw std::runtime_error("Expected value");
    }
    if (word != "==") {
        throw std::runtime_error("Expected '=' before value");
    }
    while (get_word(file, word)) {
        if (word == "==") {
            break;
        }
        value.description += word + " ";
    }
    value.description.pop_back();
    if (word != "==") {
        throw std::runtime_error("Expected '=' after description");
    }
    if (value.description.size() == 0) {
        throw std::runtime_error("Description must not be empty");
    }
    // user's name
    if (!get_word(file, word)) {
        throw std::runtime_error("Expected user's name");
    }
    for (char c : word) {
        if ((c < 'a' || 'z' < c) && (c < 'A' || 'Z' < c)) {
            throw std::runtime_error("Invalid user's name");
        }
    }
    value.name = word;
    if (!get_word(file, word)) {
        throw std::runtime_error("Unexpected end of file");
    }
    if (word != "==") {
        throw std::runtime_error("Expected '=' after user's name");
    }
    //user's second_name
    if (!get_word(file, word)) {
        throw std::runtime_error("Expected user's second_name");
    }
    for (char c : word) {
        if ((c < 'a' || 'z' < c) && (c < 'A' || 'Z' < c)) {
            throw std::runtime_error("Invalid user's second_name");
        }
    }
    value.second_name = word;
    if (!get_word(file, word)) {
        throw std::runtime_error("Unexpected end of file");
    }
    if (word != "==") {
        throw std::runtime_error("Expected '=' after user's second_name");
    }
    // user's last_name
    if (!get_word(file, word)) {
        throw std::runtime_error("Expected user's last_name");
    }
}

```

```

for (char c : word) {
    if ((c < 'a' || 'z' < c) && (c < 'A' || 'Z' < c)) {
        throw std::runtime_error("Invalid user's last_name");
    }
}
value.last_name = word;
if (!get_word(file, word)) {
    throw std::runtime_error("Unexpected end of file");
}
if (word != "==") {
    throw std::runtime_error("Expected '==' after user's last_name");
}
// user's email
if (!get_word(file, word)) {
    throw std::runtime_error("Expected user's email");
}
if (!is_email_valid(word)) {
    throw std::runtime_error("Invalid user's email");
}
value.email = word;
if (!get_word(file, word)) {
    throw std::runtime_error("Unexpected end of file");
}
if (word != "==") {
    throw std::runtime_error("Expected '==' after user's last_name");
}
// user's phone number
if (!get_word(file, word)) {
    throw std::runtime_error("Expected user's phone_number");
}
if (word.size() > MAX_SIZE_T_LEN) {
    throw std::runtime_error("Too long user's phone_number");
}
value.phone_number = 0;
for (int i = 0; i < word.size(); ++i) {
    if (word[i] < '0' && '9' < word[i] && word[i] != '+') {
        throw std::runtime_error("Invalid user's phone_number");
    }
    if (word[i] == '+' && i != 0) {
        throw std::runtime_error("Invalid user's phone_number");
    }
    if (word[i] != '+') {
        value.phone_number = value.phone_number * 10 + (word[i] - '0');
    }
}
if (!get_word(file, word)) {
    throw std::runtime_error("Unexpected end of file");
}
if (word != "==") {
    throw std::runtime_error("Expected '==' after user's phone_number");
}
// address
while (get_word(file, word)) {
    if (word == "==") {

```

```

        break;
    }
    value.address += word + " ";
}
value.address.pop_back();
if (word != "==") {
    throw std::runtime_error("Expected '=' after address");
}
if (value.address.size() == 0) {
    throw std::runtime_error("Address must not be empty");
}
// comment
while (get_word(file, word)) {
    if (word == "==") {
        break;
    }
    value.comment += word + " ";
}
value.comment.pop_back();
if (word != "==") {
    throw std::runtime_error("Expected '=' after address");
}
if (value.comment.size() == 0) {
    throw std::runtime_error("Comment must not be empty");
}
// date_time
if (!get_word(file, word)) {
    throw std::runtime_error("Expected date_time of delivery");
}
try {
    is_date(word);
} catch (std::runtime_error &ex) {
    throw std::runtime_error("Invalid date '" + word + "': " + ex.what());
}
value.date_time += word;
if (!get_word(file, word)) {
    throw std::runtime_error("Expected time of delivery");
}
if (!is_time(word)) {
    throw std::runtime_error("Invalid time");
}
value.date_time += " " + word;
if (!get_word(file, word)) {
    throw std::runtime_error("Unexpected end of file");
}
if (word != "==") {
    throw std::runtime_error("Expected '=' after date_time of delivery");
}
}
}

```

## database.h

```

#include "../pool/pool.h"
#include <vector>

```

```

class database : protected allocating::safe_allocator, protected logging::complete_logger {
public:
    explicit database(logging::logger *logger = nullptr);
    ~database() override;
public:
    int create_pool(std::vector<std::string> &query);
    int create_scheme(std::vector<std::string> &query);
    int create_collection(std::vector<std::string> &query);
    int create_note(std::stringstream &file, std::vector<std::string> &query);
public:
    int delete_pool(std::vector<std::string> &query);
    int delete_scheme(std::vector<std::string> &query);
    int delete_collection(std::vector<std::string> &query);
    int delete_note(std::stringstream &file, std::vector<std::string> &query);
public:
    int read_note(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query);
    int read_note_range(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query);
private:
    void key_filling(std::stringstream &file, type_key &key) const;
    void value_filling(std::stringstream &file, type_value &value) const;
private:
    associative_container<std::string, pool> *_pools;
private:
    allocating::memory *_allocator;
    logging::logger *_logger;
    allocating::memory *get_allocator() const override;
    logging::logger *get_logger() const override;
};

```

## database.cpp

```

#include "database.h"
#include <fstream>
#include <stdexcept>
#include <string>
#include <utility>
#include <vector>
// constructor destructor
database::database(logging::logger *logger)
    : _logger(logger) {
    // _pools = reinterpret_cast<avl_tree<std::string, pool, comparers>
*>(safe_allocate(sizeof(avl_tree<std::string, pool, comparers>)));
    // new (_pools) avl_tree<std::string, pool, comparers>(nullptr, logger);
    _pools = new avl_tree<std::string, pool, comparers>(nullptr, logger);
    safe_log("Database constructor", logging::logger::severity::warning);
}
database::~database() {
    safe_log("Database destructor", logging::logger::severity::warning);
    _pools->print_container();
    delete _pools;
}
// creating

```

```

int database::create_pool(std::vector<std::string> &query) {
    bool fail = true;
    allocating::memory *allocator = nullptr;
    allocating::memory::fit_type fit;
    if (query[1] == "GH") {
        fail = false;
    } else if (query[1][0] == 'D' || query[1][0] == 'L') {
        fail = false;
        if (query[1][1] == 'B') {
            fit = allocating::memory::fit_type::best;
        } else if (query[1][1] == 'W') {
            fit = allocating::memory::fit_type::worst;
        } else if (query[1][1] == 'F') {
            fit = allocating::memory::fit_type::first;
        } else {
            fail = true;
        }
        if (query[1][0] == 'D') {
            allocator = new allocating::memory_with_descriptors(2048000, nullptr, _logger,
fit);
        } else if (query[1][0] == 'L') {
            allocator = new allocating::memory_with_list(2048000, nullptr, _logger, fit);
        }
    }
    if (fail) {
        throw std::runtime_error("Wrong allocator type!");
    }
    pool *new_pool;
    if (nullptr == allocator) {
        new_pool = new pool(allocator, _logger);
    } else {
        new_pool = reinterpret_cast<pool *>(allocator->allocate(sizeof(pool)));
        new (new_pool) pool(allocator, _logger);
    }
    safe_log("Memory for pool is allocated", logging::logger::severity::information);
    _pools->insert(query[0], std::move(*new_pool));
    safe_log("Pool created", logging::logger::severity::information);
    return 0;
}
int database::create_scheme(std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).create_scheme(query);
    return 0;
}
int database::create_collection(std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {

```

```

        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).create_collection(query);
    return 0;
}

int database::create_note(std::stringstream &file, std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).create_note(file, query);
    return 0;
}

//reading
int database::read_note(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).read_note(file, out_stream, query);
    return 0;
}

int database::read_note_range(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).read_note_range(file, out_stream, query);
    return 0;
}

// deleting
int database::delete_pool(std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    allocating::memory *allocator = (*pool_found.second)._allocator;
    _pools->remove(query[0]);
    if (nullptr != allocator) {
        safe_deallocate(allocator);
    }
    safe_log("Pool removed", logging::logger::severity::information);
    return 0;
}

```

```

int database::delete_scheme(std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).delete_scheme(query);
    return 0;
}

int database::delete_collection(std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).delete_collection(query);
    return 0;
}

int database::delete_note(std::stringstream &file, std::vector<std::string> &query) {
    std::pair<std::string, pool *> pool_found;
    try {
        _pools->find(query[0], &pool_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Pool " + cast_to_str(query[0]) + " doesn't exists!\n");
    }
    (*pool_found.second).delete_note(file, query);
    return 0;
}

logging::logger *database::get_logger() const {
    return _logger;
}

allocating::memory *database::get_allocator() const {
    return _allocator;
}

```

## type\_data.h

```

#ifndef DATA_TYPE_H
#define DATA_TYPE_H
#define MAX_SIZE_T_LEN 15
#include <iostream>
#include <sstream>
struct type_key {
    size_t _user_id;
    size_t _delivery_id;
};
struct type_value {
    std::string description;
    std::string name;
    std::string second_name;

```



```

    std::string last_name;
    std::string email;
    size_t phone_number;
    std::string address;
    std::string comment;
    std::string date_time;
};
inline std::ostream &operator<<(std::ostream &stream, type_key const &key) {
    return stream << "{ " << key._user_id << ", " << key._delivery_id << " }";
}
inline std::ostream &operator<<(std::ostream &stream, type_value const &value) {
    return stream << "{\n\t" << value.description << ",\n\t"
        << value.name << ",\n\t"
        << value.second_name << ",\n\t"
        << value.last_name << ",\n\t"
        << value.email << ",\n\t"
        << value.phone_number << ",\n\t"
        << value.address << ",\n\t"
        << value.comment << ",\n\t"
        << value.date_time << "\n}";
}
#endif

```

## pool.cpp

```

#include "pool.h"
#include <sstream>
allocating::memory *pool::get_allocator() const {
    return _allocator;
}
logging::logger *pool::get_logger() const {
    return _logger;
}
pool::~~pool() {
    safe_log("Pool destructor", logging::logger::severity::warning);
    _schemes->print_container();
    _schemes->~associative_container();
    safe_deallocate(_schemes);
}
pool::pool(allocating::memory *allocator, logging::logger *logger)
    : _allocator(allocator), _logger(logger) {
    _schemes = reinterpret_cast<avl_tree<std::string, scheme, comparers>
*>(safe_allocate(sizeof(avl_tree<std::string, scheme, comparers>)));
    new (_schemes) avl_tree<std::string, scheme, comparers>(allocator, logger);
    safe_log("Pool constructor", logging::logger::severity::warning);
}
// creating
int pool::create_scheme(std::vector<std::string> &query) {
    scheme *new_scheme = reinterpret_cast<scheme *>(safe_allocate(sizeof(scheme)));
    new (new_scheme) scheme(_allocator, _logger);
    safe_log("Memory for scheme is allocated", logging::logger::severity::information);
    _schemes->insert(query[1], std::move(*new_scheme));
    safe_log("Scheme created", logging::logger::severity::information);
}

```

```

        return 0;
    }
    int pool::create_collection(std::vector<std::string> &query) {
        std::pair<std::string, scheme*> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        (*scheme_found.second).create_collection(query);
        return 0;
    }
    int pool::create_note(std::stringstream &file, std::vector<std::string> &query) {
        std::pair<std::string, scheme*> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        (*scheme_found.second).create_note(file, query);
        return 0;
    }
    // reading
    int pool::read_note(std::stringstream &file, std::stringstream &out_stream,
                        std::vector<std::string> &query) {
        std::pair<std::string, scheme*> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        (*scheme_found.second).read_note(file, out_stream, query);
        return 0;
    }
    int pool::read_note_range(std::stringstream &file, std::stringstream &out_stream,
                              std::vector<std::string> &query) {
        std::pair<std::string, scheme*> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        (*scheme_found.second).read_note_range(file, out_stream, query);
        return 0;
    }
    // deleting
    int pool::delete_scheme(std::vector<std::string> &query) {
        std::pair<std::string, scheme*> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);

```

```

    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    _schemes->remove(query[1]);
    safe_log("Scheme removed", logging::logger::severity::information);
    return 0;
}

int pool::delete_collection(std::vector<std::string> &query) {
    std::pair<std::string, scheme *> scheme_found;
    try {
        _schemes->find(query[1], &scheme_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    (*scheme_found.second).delete_collection(query);
    return 0;
}

int pool::delete_note(std::stringstream &file, std::vector<std::string> &query) {
    std::pair<std::string, scheme *> scheme_found;
    try {
        _schemes->find(query[1], &scheme_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    (*scheme_found.second).delete_note(file, query);
    return 0;
}

```

## pool.h

```

#include "pool.h"
#include <sstream>
allocating::memory *pool::get_allocator() const {
    return _allocator;
}
logging::logger *pool::get_logger() const {
    return _logger;
}
pool::~pool() {
    safe_log("Pool destructor", logging::logger::severity::warning);
    _schemes->print_container();
    _schemes->~associative_container();
    safe_deallocate(_schemes);
}
pool::pool(allocating::memory *allocator, logging::logger *logger)
    : _allocator(allocator), _logger(logger) {
    _schemes = reinterpret_cast<avl_tree<std::string, scheme, comparers>
*>(safe_allocate(sizeof(avl_tree<std::string, scheme, comparers>)));
    new (_schemes) avl_tree<std::string, scheme, comparers>(allocator, logger);
    safe_log("Pool constructor", logging::logger::severity::warning);
}

```

```

// creating
int pool::create_scheme(std::vector<std::string> &query) {
    scheme *new_scheme = reinterpret_cast<scheme *>(safe_allocate(sizeof(scheme)));
    new (new_scheme) scheme(_allocator, _logger);
    safe_log("Memory for scheme is allocated", logging::logger::severity::information);
    _schemes->insert(query[1], std::move(*new_scheme));
    safe_log("Scheme created", logging::logger::severity::information);
    return 0;
}

int pool::create_collection(std::vector<std::string> &query) {
    std::pair<std::string, scheme *> scheme_found;
    try {
        _schemes->find(query[1], &scheme_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    (*scheme_found.second).create_collection(query);
    return 0;
}

int pool::create_note(std::stringstream &file, std::vector<std::string> &query) {
    std::pair<std::string, scheme *> scheme_found;
    try {
        _schemes->find(query[1], &scheme_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    (*scheme_found.second).create_note(file, query);
    return 0;
}

// reading
int pool::read_note(std::stringstream &file, std::stringstream &out_stream,
    std::vector<std::string> &query) {
    std::pair<std::string, scheme *> scheme_found;
    try {
        _schemes->find(query[1], &scheme_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    (*scheme_found.second).read_note(file, out_stream, query);
    return 0;
}

int pool::read_note_range(std::stringstream &file, std::stringstream &out_stream,
    std::vector<std::string> &query) {
    std::pair<std::string, scheme *> scheme_found;
    try {
        _schemes->find(query[1], &scheme_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + " doesn't exists!\n");
    }
    (*scheme_found.second).read_note_range(file, out_stream, query);
}

```

```

        return 0;
    }
    // deleting
    int pool::delete_scheme(std::vector<std::string> &query) {
        std::pair<std::string, scheme *> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        _schemes->remove(query[1]);
        safe_log("Scheme removed", logging::logger::severity::information);
        return 0;
    }
    int pool::delete_collection(std::vector<std::string> &query) {
        std::pair<std::string, scheme *> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        (*scheme_found.second).delete_collection(query);
        return 0;
    }
    int pool::delete_note(std::stringstream &file, std::vector<std::string> &query) {
        std::pair<std::string, scheme *> scheme_found;
        try {
            _schemes->find(query[1], &scheme_found);
        } catch (std::runtime_error &ex) {
            throw std::runtime_error("Scheme " + cast_to_str(query[0]) + "/"
                                     + cast_to_str(query[1]) + " doesn't exists!\n");
        }
        (*scheme_found.second).delete_note(file, query);
        return 0;
    }
}

```

## sheme.cpp

```

#include "scheme.h"
allocating::memory *scheme::get_allocator() const {
    return _allocator;
}
logging::logger *scheme::get_logger() const {
    return _logger;
}
scheme::~scheme() {
    safe_log("Scheme destructor", logging::logger::severity::warning);
    _collections->print_container();
    _collections->~associative_container();
    safe_deallocate(_collections);
}
scheme::scheme(allocating::memory *allocator, logging::logger *logger)
    : _allocator(allocator), _logger(logger) {

```

```

    _collections = reinterpret_cast<avl_tree<std::string, collection, comparers>
*>(safe_allocate(sizeof(avl_tree<std::string, collection, comparers>)));
    new (_collections) avl_tree<std::string, collection, comparers>(allocator, logger);
    safe_log("Scheme constructor", logging::logger::severity::warning);
}
// creating
int scheme::create_collection(std::vector<std::string> &query) {
    collection *new_collection = reinterpret_cast<collection
*>(safe_allocate(sizeof(collection)));
    new (new_collection) collection(_allocator, _logger);
    safe_log("Memory for collection is allocated", logging::logger::severity::information);
    _collections->insert(query[2], std::move(*new_collection));
    safe_log("Collection created", logging::logger::severity::information);
    return 0;
}
int scheme::create_note(std::stringstream &file, std::vector<std::string> &query) {
    std::pair<std::string, collection *> collection_found;
    try {
        _collections->find(query[2], &collection_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Collection " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + "/" + cast_to_str(query[2]) + " doesn't exists!\n");
    }
    (*collection_found.second).create_note(file, query);
    return 0;
}
// reading
int scheme::read_note(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query) {
    std::pair<std::string, collection *> collection_found;
    try {
        _collections->find(query[2], &collection_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Collection " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + "/" + cast_to_str(query[2]) + " doesn't exists!\n");
    }
    (*collection_found.second).read_note(file, out_stream, query);
    return 0;
}
int scheme::read_note_range(std::stringstream &file, std::stringstream &out_stream,
std::vector<std::string> &query) {
    std::pair<std::string, collection *> collection_found;
    try {
        _collections->find(query[2], &collection_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Collection " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + "/" + cast_to_str(query[2]) + " doesn't exists!\n");
    }
    (*collection_found.second).read_note_range(file, out_stream, query);
    return 0;
}
// deleting
int scheme::delete_collection(std::vector<std::string> &query) {
    std::pair<std::string, collection *> collection_found;

```

```

    try {
        _collections->find(query[2], &collection_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Collection " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + "/" + cast_to_str(query[2]) + " doesn't exists!\n");
    }
    _collections->remove(query[2]);
    safe_log("Collection removed", logging::logger::severity::information);
    return 0;
}

int scheme::delete_note(std::stringstream &file, std::vector<std::string> &query) {
    std::pair<std::string, collection *> collection_found;
    try {
        _collections->find(query[2], &collection_found);
    } catch (std::runtime_error &ex) {
        throw std::runtime_error("Collection " + cast_to_str(query[0]) + "/"
            + cast_to_str(query[1]) + "/" + cast_to_str(query[2]) + " doesn't exists!\n");
    }
    (*collection_found.second).delete_note(file, query);
    return 0;
}

```

## scheme.h

```

#include "../collection/collection.h"
class scheme : protected allocating::safe_allocator, protected logging::complete_logger {
    friend class database;
public:
    explicit scheme(allocating::memory *allocator = nullptr, logging::logger *logger =
        nullptr);
    ~scheme();
public:
    int create_collection(std::vector<std::string> &query);
    int create_note(std::stringstream &file, std::vector<std::string> &query);
public:
    int read_note(std::stringstream &file, std::stringstream &out_stream,
        std::vector<std::string> &query);
    int read_note_range(std::stringstream &file, std::stringstream &out_stream,
        std::vector<std::string> &query);
public:
    int delete_collection(std::vector<std::string> &query);
    int delete_note(std::stringstream &file, std::vector<std::string> &query);
private:
    associative_container<std::string, collection> *_collections;
private:
    allocating::memory *get_allocator() const override;
    logging::logger *get_logger() const override;
    allocating::memory *_allocator;
    logging::logger *_logger;
};
inline std::ostream &operator<<(std::ostream &stream, scheme const &key) {
    return stream << "[ object scheme ]";
}

```

## main.cpp

```
#include "logger/server_logger.h"
const int MSG_Q_KEY_FLAG_LOGGER = 0600;
const int MSG_Q_CHANNEL_RECEIVE_LOG = 36;
int main(int argc, char *argv[]) {
    if (argc != 2) {
        std::cout << "You should enter the configuration file name" << std::endl;
        return 1;
    }
    std::string filename(argv[1]);
    logging::logger_builder *builder = new logger_server_builder_concrete();
    if (nullptr == builder) {
        std::cout << "Cannot allocate memory for builder" << std::endl;
        return -1;
    }
    logging::logger *logger;
    logger = builder->construct_configuration(filename);
    if (nullptr == logger) {
        std::cout << "Logger falled" << std::endl;
        return -3;
    }
    delete builder;
    key_t key = -1;
    int msqid = -1;
    MsgQueue msg;
    key = ftok("/bin/ls", 'E');
    if (key < 0) {
        perror("ftok");
        return -1;
    }
    msqid = msgget(key, MSG_Q_KEY_FLAG_LOGGER | IPC_CREAT);
    if (msqid < 0) {
        perror("msgget");
        return -2;
    }
    while (1) {
        if (msgrcv(msqid, &msg, sizeof(msg) - sizeof(long), MSG_Q_CHANNEL_RECEIVE_LOG, 0) <
0)
        {
            perror("msgrcv");
            return -3;
        }
        std::string message(msg.buff);
        logger->log(message, msg.severity);
    }
    delete logger;
    if (msgctl(msqid, IPC_RMID, NULL) < 0) {
        perror("msgctl");
        return -4;
    }
    std::cout << "\nServer is now shutting down!\n";
    return 0;
}
```



## memory.h

```
#ifndef MEMORY_H
#define MEMORY_H
#include <cstdint>
#include "../..../logger/source/logger/prototypes/logger.h"
namespace allocating {
class memory {
public:
    enum fit_type {
        first,
        best,
        worst
    };
    memory() = default;
    virtual ~memory() = default;
    memory(memory const&) = delete;
    void operator=(memory const&) = delete;
    void *operator+=(size_t const &);
    void operator-=(void *);
    virtual void *allocate(size_t) const = 0;
    virtual void deallocate(void *) const = 0;
protected:
    logging::logger *_logger_allocator;
    void *_trusted_memory;
};
}
#endif
```