Задания к работе №1 по теории сложных систем.

Все задания реализуются на языке программирования C++ (стандарт C++14 и выше).
Реализованные в заданиях приложения не должны завершаться аварийно; все возникающие исключительные ситуации должны быть перехвачены и обработаны.
Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения.
Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.
Во всех заданиях все вводимые (с консоли, файла, командной строки) пользователем данные должны (если не сказано обратное) быть подвергнуты валидации в соответствии с типом валидируемых данных.
Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.
Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.
Реализованные компоненты должны зависеть от абстракций, а не от конкретных реализаций абстракций.
Для реализованных компонентов должны быть переопределены (либо перекрыты - для классов-сервисов) следующие механизмы классов C++: конструктор копирования, деструктор, оператор присваивания, конструктор перемещения, присваивание перемещением.

0. Опишите контракты взаимодействия и связанные типы для следующих компонент:

- Логгер:
  - *severity* - ICS (in-class scope) перечисление с возможными значениями (в порядке увеличения жёсткости): *trace*, *debug*, *information*, *warning*, *error*, *critical*; nested по отношению к интерфейсу *logger*; описывает уровень жёсткости логгирования;
  - *logger* - интерфейс, предоставляющий методы для
    - записи лога с заданным уровнем жёсткости:

    *virtual logger *logger::log(const std::string& target, severity level) const = 0;*
  - *logger_builder* - интерфейс, предоставляющий fluent-функционал для последовательного конструирования объекта, реализующего интерфейс *logger*, на основе порождающего паттерна проектирования “Строитель”:
    - добавления в строитель файлового потока вывода, в который будут писаться только логи, уровень жёсткости которых >= заданного уровня жёсткости:

      *virtual logger_builder *add_file_stream(std::string const &stream_file_path, logger::severity severity) = 0;*
    - добавления в строитель консольного потока вывода, в который будут писаться только логи, уровень жёсткости которых >= заданного уровня жёсткости:

      *virtual logger_builder *add_console_stream(logger::severity severity) = 0;*
    - добавления в строитель потоков вывода на основе конфигурационного файла (в качестве параметров подаются путь к файлу и path уровня конфигурации):

      *virtual logger_builder *transform_with_configuration(std::string const &configuration_file_path, std::string const &configuration_path) = 0;*
    - Очистки в объекте строителя всей ранее заполненной информации о потоках вывода:

      *virtual logger_builder *clear() = 0;*
    - Создания объекта логгера на основе ранее заполненной в строителе информации (внутренняя информация в объекте строителя при этом не очищается):

      *virtual logger *build() const = 0;*

- Аллокатор:
  - *allocator* - интерфейс, предоставляющий методы для
    - выделения динамической памяти размера *target_size* байтов из объекта аллокатора (задекорирован оператором +=)

      *virtual void *allocator::allocate(size_t target_size) = 0;*
    - освобождения ранее выделенной из объекта аллокатора динамической памяти (задекорирован оператором -=):

      *virtual void allocator::deallocate(void *target_to_dealloc) = 0;*
    - перевыделения ранее выделенной из объекта аллокатора динамической памяти:

  *virtual void *allocator::reallocate(void *target_to_realloc, size_t new_size) = 0;*

1. Реализуйте логгер на основе контракта *logger* из задания 0. Ваша реализация логгера должна конфигурироваться двумя способами:

- на основе конфигурационного файла в формате JSON
- на основе порождающего паттерна проектирования “Строитель”

Конфигурирование объекта логгера позволяет задавать потоки вывода (файловые, консольный) и минимальные *severity* заданных потоков вывода для конкретного объекта логгера (пример: если минимальный *severity* == *warning*, то через данный объект логгера в данный поток вывода будут выводиться логи только с *severity* == *warning*, *error*, *critical*). При повторной настройке уже открытого для объекта логгера потока вывода необходимо обновить уже настроенный *severity* для этого потока вывода.

Учтите, что один и тот же поток вывода может использоваться единовременно разными объектами логгеров (и *severity* этого потока вывода для разных логгеров также может различаться). При разрушении последнего объекта логгера, связанного с заданным файловым потоком вывода, этот файловый поток вывода должен быть закрыт.

Структура лога:

[timestamp][severity] message

где

timestamp - текущие <дата по григорианскому календарю>/<время> в формате

dd/MM/yyyy hh:mm:ss

severity - строковое представление значения жёсткости лога

message - передаваемое логгеру сообщение.

Реализуйте и продемонстрируйте работу приложения, в рамках которого

- различными способами конфигурируются несколько объектов логгера (при этом хотя бы два объекта логгера связаны с одним и тем же файловым потоком вывода);
- для записи логов в файл/на консоль используются сконфигурированные объекты логгеров;
- при разрушении объектов логгеров закрываются связанные с ними потоки вывода.

2. Реализуйте аллокатор на основе контракта *allocator* из задания 0. Выделение и освобождение динамической памяти реализуйте посредством глобальных операторов *::operator new* и *::operator delete* соответственно.

Продемонстрируйте работу аллокатора, разместив в нём объекты различных типов (числа, строки, объекты собственных типов данных, etc.). Предусмотрите логгирование (на основе реализации логгера из задания 1) вызовов интерфейсных методов (на уровне Вашей реализации аллокатора) с указанием:

- текстового описания действия;
- адреса места в памяти (относительно адреса со значением *0x0* на уровне глобальной кучи), для которого происходит выделение/освобождение;
- неслужебного содержимого освобождаемого участка перед освобождением в виде коллекции значений байт (в диапазоне [0..255]).

3. Реализуйте аллокатор на основе контракта *allocator* из задания 0. Выделение памяти реализуйте при помощи методов (с возможностью конфигурации конкретной реализации через конструктор объекта аллокатора и через метод для уже созданного объекта аллокатора) первого подходящего, лучшего подходящего, худшего подходящего; а освобождение памяти - при помощи метода освобождения в рассортированном списке. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из аллокатора, передаваемого как параметр по умолчанию конструктору (если аллокатор не передан, память запрашивается из глобальной кучи)). В объекте аллокатора допускается размещение единственного поля типа *void* * (репрезентирующего указатель на доверенную объекту аллокатора области памяти).
Продемонстрируйте работу аллокатора, разместив в нём объекты различных типов (числа, строки, объекты собственных типов данных). Предусмотрите логгирование (на основе реализации логгера из задания 1) вызовов интерфейсных методов (на уровне Вашей реализации аллокатора) с указанием:

- текстового описания действия;
- адреса места в памяти (относительно адреса *0x0* начала доверенной аллокатору области памяти), для которого происходит выделение/освобождение;
- неслужебного содержимого освобождаемого участка перед освобождением в виде коллекции значений байт (в диапазоне [0..255]).

4. Реализуйте аллокатор на основе контракта *allocator* из задания 0. Выделение памяти реализуйте при помощи методов (с возможностью конфигурации конкретной реализации через конструктор объекта аллокатора и через метод для уже созданного объекта аллокатора) первого подходящего, лучшего подходящего, худшего подходящего; а освобождение памяти - при помощи метода освобождения с дескрипторами границ. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из аллокатора, передаваемого как параметр по умолчанию конструктору (если аллокатор не передан, память запрашивается из глобальной кучи)). В объекте аллокатора допускается размещение единственного поля типа *void* * (репрезентирующего указатель на доверенную объекту аллокатора области памяти).
Продемонстрируйте работу аллокатора, разместив в нём объекты различных типов данных (числа, строки, объекты собственных типов данных, etc.); память под аллокатор запросите из реализации аллокатора из задания 3. Предусмотрите логгирование (на основе реализации логгера из задания 1) вызовов интерфейсных методов (на уровне Вашей реализации аллокатора) с указанием:

- текстового описания действия;
- адреса места в памяти (относительно адреса начала доверенной аллокатору области памяти), для которого происходит выделение/освобождение;
- неслужебного содержимого освобождаемого участка перед освобождением в виде коллекции значений байт (в диапазоне [0..255]).

5. Реализуйте аллокатор на основе контракта *allocator* из задания 0. Выделение (с возможностью конфигурации конкретной реализации через конструктор объекта аллокатора и через метод для уже созданного объекта аллокатора) и освобождение памяти реализуйте при помощи алгоритмов системы двойников. Служебные данные для работы аллокатора размещайте в доверенной ему области памяти (размер доверенной области памяти задаётся на уровне конструктора объекта аллокатора и доверенная память при этом запрашивается из аллокатора, передаваемого как параметр по умолчанию конструктору (если аллокатор не передан, память запрашивается из глобальной кучи)). В объекте аллокатора допускается размещение единственного поля типа *void ** (репрезентирующего указатель на доверенную объекту аллокатора области памяти).

Продемонстрируйте работу аллокатора, разместив в нём объекты различных типов данных (числа, строки, объекты собственных типов данных, etc.), а также объекты логгеров из задания 1. Предусмотрите логгирование (на основе реализации логгера из задания 1) вызовов интерфейсных методов (на уровне Вашей реализации аллокатора) с указанием:

- текстового описания действия;
- адреса места в памяти (относительно адреса начала доверенной аллокатору области памяти), для которого происходит выделение/освобождение;
- неслужебного содержимого освобождаемого участка перед освобождением в виде коллекции значений байт (в диапазоне [0..255]).