Опишу контекст (не само тестовое задание), чтобы лучше понять цель тестового задания и используемую терминологию:
    ** У нас есть обмен по некоему протоколу s1ap между двумя узлами ENODEB и MME.
    ** Узлы ENODEB и MME обмениваются командами и ответами на команды, а также сообщениями не требующими ответа.
    ** Сценарии этого обмена называются call flow и описаны в соответствующих документах 3GPP.
    ** У нас есть точка (man in the middle) в которой происходит логирование (зеркалирование) этого обмена между ENODEB и MME.
    ** В этой точке сообщения парсятся и передаются в модуль обработки (назовем его S1apDb). Будем называть такие распарсенные сообщения событиями (events).
    ** Каждое событие содержит:
        - время (64 битное целое число (timestamp (unsigned long) в миллисекундах от начала EPOCH))
        - тип события:
```
              enum EventType {
                AttachRequest,                    // ENODEB --> MME
                IdentityResponse,                 // ENODEB --> MME
                AttachAccept,                     // MME    --> ENODEB
                Paging,                           // ENODEB --> MME
                PathSwitchRequest,                // ENODEB --> MME
                PathSwitchRequestAcknowledge,     // MME    --> ENODEB
                UEContextReleaseCommand,          // ENODEB --> MME
                UEContextReleaseResponse,         // MME    --> ENODEB
              };
```
        - enodeb_id (32 битное беззнаковое целое: unsigned int) в сообщениях: во всех кроме Paging
        - mme_id (32 битное беззнаковое целое: unsigned int) в сообщениях: во всех кроме AttachRequest, Paging
        - imsi (64 битное беззнаковое целое: unsigned long) в сообщениях: AttachRequest (опционально, но что-то одно будет или imsi или m_tmsi), IdentityResponse
        - m_tmsi (32 битное беззнаковое целое: unsigned int) в сообщениях: AttachRequest(опционально), AttachAccept, Paging
        - cgi (std::vector<unsigned char>) может содержаться только в сообщениях типа: ENODEB --> MME
    ** call flow 1:
        - AttachRequest {enodeb_id_1, imsi_1, cgi_1}   (регистрация абонента)
        - IdentityResponse {enodeb_id_1, mme_id_1, imsi_1, cgi_2}   (не более чем через 1 сек с AttachRequest) (опциональное сообщение)
        - AttachAccept {enodeb_id_1, mme_id_1, m_tmsi_1}   (не более чем через 1 сек с AttachRequest)
        - Paging {m_tmsi_1, cgi_3}   (приходят регулярно)
        - PathSwitchRequest {enodeb_id_2, mme_id_1, cgi_4}   (новый enodeb_id_2, но старый mme_id_1)
        - PathSwitchRequestAcknowledge {enodeb_id_2, mme_id_2}   (не более чем через 1 сек с PathSwitchRequest, новый mme_id_2)
            *** такие пары PathSwitchRequest и PathSwitchRequestAcknowledge могут приходить регулярно и каждый раз меняется пара (enodeb_id, mme_id)
        - UEContextReleaseCommand {enodeb_id_2, mme_id_2, cgi_4}   (дерегистрация абонента)
        - UEContextReleaseResponse {enodeb_id_2, mme_id_2}   (не более чем через 1 сек с UEContextReleaseCommand)
            *** после этого сбрасываются (enodeb_id, mme_id), но сохраняется m_tmsi_1
        - прошло менее 24 часов
        - AttachRequest {enodeb_id_3, m_tmsi_1, cgi_5}   (нет imsi !!!, но мы запомнили связку (m_tmsi_1, imsi_1) из предыдущих сообщений)
        - AttachAccept {enodeb_id_3, mme_id_3, m_tmsi_1}   (не более чем через 1 сек с AttachRequest)
        - Paging {m_tmsi_1, cgi_3}   (приходят регулярно)
        - PathSwitchRequest {enodeb_id_4, mme_id_3, cgi_6}   (новый enodeb_id_4, но старый mme_id_3)
        - PathSwitchRequestAcknowledge {enodeb_id_4, mme_id_4}   (не более чем через 1 сек с PathSwitchRequest, новый mme_id_4)
            *** такие пары PathSwitchRequest и PathSwitchRequestAcknowledge могут приходить регулярно и каждый раз меняется пара (enodeb_id, mme_id)
        - UEContextReleaseCommand {enodeb_id_4, mme_id_4, cgi_7}   (дерегистрация абонента)
        - UEContextReleaseResponse {enodeb_id_4, mme_id_4}   (не более чем через 1 сек с UEContextReleaseCommand)
            *** после этого сбрасываются (enodeb_id, mme_id), но сохраняется m_tmsi_1
        - прошло более 24 часов (связка (m_tmsi_1, imsi_1) пропала на узле MME) тогда call flow идет с самлого начала:
        -- AttachRequest {enodeb_id_5, imsi_1, cgi_8}
            ...
    ** call flow 2 (небольшая модификация):
        - AttachRequest {enodeb_id_1, m_tmsi_1, cgi_1}   (регистрация абонента)
        - IdentityResponse {enodeb_id_1, mme_id_1, imsi_1, cgi_2}   (не более чем через 1 сек с AttachRequest) (опциональное сообщение)
        - AttachAccept {enodeb_id_1, mme_id_1, m_tmsi_2}   (не более чем через 1 сек с AttachRequest, смена m_tmsi_2 !!!!)
            ...
    ** imsi - уникальный идентификатор абонента, все остальные - временные

** при регистрации абонента imsi может не появляться ни в одном сообщении (если телефон был выключен менее 24 часов назад). Ситуацию спасет только, если мы сохранили связку (m_tmsi, imsi) из предыдущей регистрации абонента
** текущую активность абонента: изменения cgi), а также дерегистрацию, можно отслеживать только по enodeb_id, mme_id, но не по m_tmsi
** enodeb_id, mme_id сбрасываются при дерегистации абонента
** обработчик событий должен:
- отслеживать регистрацию абонента
- отслеживать дерегистрацию абонента
- отслеживать изменения его cgi
- возвращать информацию о регистрации, дерегистрации, изменения cgi в привязке к imsi (а не к m_tmsi, enodeb_id или mme_id !!!)
- отслеживать время жизни различных объектов (например, не пришел ответ на команду в течении 1 сек), тогда надо сбрасывать инфу о команде
(Обработчик должен самостоятельно отслеживать таймауты (напр., 1 сек для ответов), используя timestamp из событий)
- быть устойчивым к нарушению call flow (мало ли там что придет) и сохранять консистентность своих внутренних структур данных
- быть оптимальным в вычислительном смысле: все изменения внутренних структур данных через ключи, ни каких сколько-нибудь длинных циклов
- при дерегистрации абонента стирать в базе всю инфу о связанных с imsi идентификаторах: enodeb_id, mme_id (связка (m_tmsi, imsi) конечно сохраняется)
- обновлять в базе время последнего event для конкретного imsi. Если прошло более 24 часов, то стирать в базе всю инфу о imsi и связанном с ним m_tmsi

Цель тестового задания написать модуль S1apDb обработки событий (еще раз подчеркну, что в тестовом задании не нужно парсить s1ap):
** Модуль обработки это класс S1apDb, инкапсулирующий доступ к набору контейнеров stl (множество различных std::unordered_map, std::multimap, ...)
через функцию handler(event) обработчик событий
** Иногда handler ничего не возвращает (например, через std::optional или std::shared_ptr == nullptr), иногда возвращает объект класса :

```
struct S1apOut {
  enum S1apOutType {
    Reg, UnReg, Cgi
  };
  S1apOutType s1ap_type;
  unsigned long imsi;
  std::vector<unsigned char> cgi;
};
```

** S1apDb::handler (примерно):

```
switch (event_type) {
    case EventType::AttachRequest:
    {
       ...
    }
    ...
}
```

** Работа со временем. Внутри класса S1apDb не использовать функции получения текущего времени. Вся информация о текущем времени содержится в event (см. выше: "время (64 битное целое число (timestamp (unsigned long) в миллисекундах от начала EPOCH))"). Считаем, что event по всем абонам идут с огромной частотой. При каждом event стираем из внутренней базы не более одной (+-) устаревшей записи.
** также нужно написать ряд тестов (Google Test)
** программу написать для x86_64, Linux на c++, не используя ничего кроме stl
** желательно под cmake, хорошо бы под Clion (если есть)
** настроить локальный гит и коммитить туда