Звіт до лабораторної роботи 1 з баз даних

Завдання лаби

Завдання лабораторної –розробити програму, що працює з двома структурованими файлами з індексно-довільним доступом. Два файли відповідають двом пов'язаним сутностям із ER-моделі зі зв'язком один-до-багатьох(на основі об'єкту, до якого спрямована 1, формується master-file, а на основі іншого – slave-file).

Сутності, з якими працював я – користувач(User) та стаття(Article). Атрибутами користувача є ID, ім'я та пароль, атрибутами статті – ID статті, ID її автора(користувача), назва статті та її категорія.

На файловому рівні програма працює з 2-ма бінарними файлами: master-file(Sfl.bin) та slave-file(SP.bin). Крім цього, є ще також допоміжний файл, що відповідає індексній таблиці(Sind.bin)

Індексна таблиця містить пари - значення ключа (ID) master-сутності та адреса відповідного запису в

master-файлі S.fl. Записи індексної таблиці відсортовані, наприклад, за

зростанням значення ключа.

У master-file записані значення атрибутів для конкретного запису, а також адреса, за якою у slave-file записані значення атрибутів першої із підзаписів цього запису. Відповідно, у slave-file записані значення атрибутів для конкретного підзапису, а також адреса наступного підзапису(якщо це останній підзапис для даного запису, ця адреса буде рівна -1). Таким чином, всі статті для кожного користувача зв'язані у список.

Робота з файлами

Для реалізації цього я створив три структури:

```
struct Key //стовпчики індексної таблиці
{
   int64_t user_id = 0;
   int64_t address = -1;
};
struct User //екземпляри користувачів
{
   int64_t user_id = 0;
   char username[MaxFieldLen] = {};
   char password[MaxFieldLen] = {};
   int64_t adress = -1;
   int64_t slave_adress = -1;
};
struct Article //екземпляри статей
{
   int64_t author_code = 0;
   int64_t article_code = 0;
   char category[MaxFieldLen] = {};
   char name[MaxFieldLen] = {};
   int64_t next = -1;
};
```

Відповідно до умови лабораторної, індексну таблицю дозволяється зберігати у внутрішній пам'яті та переписувати у файл після завершення. У моєму коді їй віповідає вектор екземплярів структур Key:

```
vector<Key> index_table;
```

Функції для читання цих структур та зміни їх вмісту:

```
bool ReadMaster(User& record, std::fstream& file, const std::streampos& pos)
{
   if (!file)  return false;
   file.seekg(pos);
   file.read(reinterpret_cast<char*>(&record), sizeof(User));
```

```
    return !file.fail();
}
bool ReadSlave(Article& record, std::fstream& file, const std::streampos& pos)
{
    if (!file)  return false;
    file.seekg(pos);
    file.read(reinterpret_cast<char*>(&record), sizeof(Article));
    return !file.fail();
}
bool WriteMaster(const User& record, std::fstream& file, const std::streampos& pos)
{
    if (!file) return false;
    file.seekp(pos);
    file.write(reinterpret_cast<const char*>(&record), sizeof(User));
    file.flush();
    return !file.fail();
}
bool WriteSlave(const Article& record, std::fstream& file, const std::streampos& pos)
{
    if (!file) return false;
    file.seekp(pos);
    file.write(reinterpret_cast<const char*>(&record), sizeof(Article));
    file.flush();
    return !file.fail();
}
bool WriteKey(Key& record, std::fstream& file, std::streampos& pos)
{
    if (!file) return false;
    file.seekp(pos);
    file.write(reinterpret_cast<const char*>(&record), sizeof(Key));
    file.flush();
    pos = pos + static_cast<std::streamoff>(sizeof(Key));
    return !file.fail();
}
```

Ось ще кілька допоміжних функцій, які я зробив для роботи з файлами(їх реалізацію див. у файлі з кодом):

bool FillUserInfo(User& u, string name, string pass)
bool FillArticleInfo(Article& a, string name, string category) //встановлюють задані нами значення полів для структур Article та User(повертають false, якщо задані нами значення є занадто довгими, інакше true)

PrintNodesMaster(std::fstream& file, const std::streampos& record_pos, bool IsFirstRow) //виводить значення всіх полів запису, що знаходиться у master-файлі за заданою адресою

PrintNodesSlave(std::fstream& file, const std::streampos& record_pos) //виводить значення всіх полів для всіх підзаписів, що відповідають конкретному запису, де record_pos – адреса першого підзапису

Ми «проходимо» всі підзаписи у зв'язаному списку(переходячи на наступний, отримуючи його значення з поля next), доки не натрапимо на кінцевий для даного запису(адреса наступного підзапису рівна -1).

int GetUserIndex(int UserID) – повертає індекс у інексній таблиці, що відповідає користувачу з заданим ID(перебираємо всі едементи індексної таблиці, доки не знадемо потрібний)

int GetSlaveAdress(int master_index, int slave_ID, std::fstream& masterfile, std::fstream& slavefile, std::streampos& prev_pos) – повертає адресу підзапису, якому відповідає стаття з заданим ID, якщо відомий індекс її автора(теж «проходить» усі підзаписи заданого записа, доки не знайде потрібний)

int GetMasterIndex(int slave_ID, std::fstream& masterfile, std::fstream& slavefile, std::streampos& prev_pos) – повертає індекс у інексній таблиці, що відповідає автору статті з заданим ID.

Реалізація команд

У лабораторній роботі потрібно було реалізувати наступні функції:

Функція get-m – вивести інформацію(значення атрибутів) про користувача із заданим ID.

```
void GetM(int UserID, std::fstream& masterfile)
{
```

```
    int master_index = GetUserIndex(UserID);
    if (master_index == -1)
    {
        cout << "Error: User with given ID doesn't exist" << "\n";
        return;
    }
    PrintNodesMaster(masterfile, index_table[master_index].address, true);
}
```

За допомогою ф-ції GetUserIndex знаходимо індекс, за яким шуканий користувач записаний у індексній таблиці, таким чином знаходимо, де він записаний у master-файлі і виводимо інформацію

Функція get-s – вивести інформацію(значення атрибутів) про статтю із заданим ID

```
void GetS(int ArticleID, std::fstream& masterfile, std::fstream& slavefile)
{
    std::streampos PrevSlavePos = -1;
    int master_index = GetMasterIndex(ArticleID, masterfile, slavefile, PrevSlavePos);
    if (master_index == -1)
    {
        cout << "Error: the Article with given ID doesn't exist" << "\n";
        return;
    }
    int64_t slave_adress = GetSlaveAdress(master_index, ArticleID, masterfile, slavefile, PrevSlavePos);
    Article TempArticle;
    if (slave_adress == -2)
    {
        std::cerr << "Unable to update next_ptr. Error: read failed" << "\n";
        return;
    }
    ReadSlave(TempArticle, slavefile, slave_adress);
    cout << TempArticle << "\n";
}
```

За допомогою ф-ції GetMasterIndex знаходимо індекс, за яким автор статті записаний у індексній таблиці, таким чином знаходимо, де він записаний у master-файлі. Переходимо за першим підзаписом цього користувача і «проходимо» по зв’язаному списку, шукаючи потрібний.

Функція del-m – видалити користувача із заданим ID.

```
void DelM(int UserID, std::fstream& masterfile, std::fstream& slavefile)
{
    int master_index = GetUserIndex(UserID);
    if (master_index == -1)
    {
        cout << "Error: User with given ID doesn't exist" << "\n";
        return;
    }
    std::streampos user_address = index_table[master_index].address;
    rubbish_m.push_back(user_address);
    User user;
    ReadMaster(user, masterfile, user_address);
    index_table.erase(index_table.begin() + master_index);
    --UserCnt;
    cout << "User with ID " << UserID << " deleted!" << "\n";
    if (user.slave_adress == -1) return;
    std::streampos CurPos = user.slave_adress;
    Article TempArticle;
    while (CurPos != -1)
    {
        if (!ReadSlave(TempArticle, slavefile, CurPos))
        {
            std::cerr << "Unable to update next_ptr. Error: read failed" << "\n";
            break;
        }
        rubbish_s.push_back(CurPos);
```

```
        --TotalArticleCnt;
        CurPos = TempArticle.next;
    }
}
```

Аналогічно до get-m знаходимо адресу користувача. Переходимо за його адресою першого підзапису, і, «проходячись» по зв'язаному списку підзаписів, видаляємо по черзі усі його статті. Далі видаляємо самого користувача, вилучаючи з індексної таблиці стовпчик, що йому відповідає

Функція del-s: видалити статтю із заданим ID.

```
void DelS(int ArticleID, std::fstream& masterfile, std::fstream& slavefile)
{
    std::streampos PrevSlavePos = -1;
    int master_index = GetMasterIndex(ArticleID, masterfile, slavefile, PrevSlavePos);
    if (master_index == -1)
    {
        cout << "Error: the Article with given ID doesn't exist" << "\n";
        return;
    }
    std::streampos user_address = index_table[master_index].address;
    std::streampos slave_adress = GetSlaveAdress(master_index, ArticleID, masterfile, slavefile, PrevSlavePos);
    if (slave_adress == -2)
    {
        std::cerr << "Unable to update next_ptr. Error: read failed" << "\n";
        return;
    }
    Article TempArticle;
    ReadSlave(TempArticle, slavefile, slave_adress);
    rubbish_s.push_back(slave_adress);
    User user;
    if (PrevSlavePos == -1)
    {
        ReadMaster(user, masterfile, user_address);
        user.slave_adress = TempArticle.next;
        WriteMaster(user, masterfile, user_address);
    }
    else
    {
        slave_adress = TempArticle.next;
        ReadSlave(TempArticle, slavefile, PrevSlavePos);
        TempArticle.next = slave_adress;
        WriteSlave(TempArticle, slavefile, PrevSlavePos);
    }
    cout << "Article with ID " << ArticleID << " deleted!" << "\n";
    --TotalArticleCnt;
}
```

Аналогічно до get-s знаходимо адресу відповідного підзапису. Також, знаходимо адреси наступного і попереднього підзаписів(якщо вони існують). Наступний знаходиться посто як значення поля next, попередній – як запис, адреса наступного для якого збігається з адресою нашого підзапису.

Якщо попередній підзапис існує, встановлюємо значення поля next на наступний запис після шуканого(це значення може бути рівним -1, якщо запис, що видаляється, був кінцевим)

Якщо його не існує, то видалений підзапис був першим для свого записа. Тому, встановлюємо значення поля slave_adress для master-записа на наступний запис після шуканого(це значення може бути рівним -1, якщо запис, що видаляється, був кінцевим)

Функція update-m – оновити значення якогось із полів для користувача із заданим ID. Спочатку знаходимо постачальника аналогічно до get-m, потім змінюємо значення у заданого поля і заносимо запис на теж саме місце.

```
void UpdateM(int UserID, string field, string val, std::fstream& masterfile)
{
    int master_index = GetUserIndex(UserID);
```

```
    if (master_index == -1)
    {
        cout << "Error: User with given ID doesn't exist" << "\n";
        return;
    }
    std::streampos user_address = index_table[master_index].address;
    User user;
    ReadMaster(user, masterfile, user_address);
    if (field != "username" && field != "password")
    {
        cout << "Error: the User doesn't have a field named " << field << "\n";
        return;
    }
    bool FilledCorrectly;
    if (field == "username") FilledCorrectly = FillUserInfo(user, val, user.password);
    if (field == "password") FilledCorrectly = FillUserInfo(user, user.username, val);
    if (!FilledCorrectly) return;
    WriteMaster(user, masterfile, user_address);
    cout << field << " is updated! New value: " << val << "\n";
}
```

Функція update-s – оновити значення якогось із полів для статті із заданим ID. Працює повністю аналогічно.

Функція insert-m – наповнюємо структуру, яка відповідає запису файлу S.fl, і записуємо в цей файл; необхідно зафіксувати адресу (чи номер) запису і внести її разом із значенням ключа до індексної таблиці; при необхідності індексну таблицю необхідно відсортувати за зростанням значення ключа і теж записати у свій файл.

Функція insert-s - наповнюємо структуру, яка відповідає запису файлу SP.fl; виконуємо get-m по ключу постачальника і заносимо значення поля посилання (з нього) на ланцюг підлеглих записів поставок в структуру запису поставок (тим самим ставимо її на початок ланцюга); заносимо запис поставок в файл SP.fl (в кінець) і запам’ятовуємо її адресу (чи номер), потім цю адресу переписуємо в прочитаний запис постачальника і заносимо цей запис до файлу S.fl.

```
void UpdateS(int ArticleID, string field, string val, std::fstream& masterfile, std::fstream& slavefile)
{
    std::streampos PrevSlavePos = -1;
    int master_index = GetMasterIndex(ArticleID, masterfile, slavefile, PrevSlavePos);
    if (master_index == -1)
    {
        cout << "Error: Article with given ID doesn't exist" << "\n";
        return;
    }
    std::streampos slave_adress = GetSlaveAdress(master_index, ArticleID, masterfile, slavefile, PrevSlavePos);
    if (slave_adress == -2)
    {
        std::cerr << "Unable to update next_ptr. Error: read failed" << "\n";
        return;
    }
    Article TempArticle;
    ReadSlave(TempArticle, slavefile, slave_adress);
    if (field != "name" && field != "category")
    {
        cout << "Error: the Article doesn't have a field named " << field << "\n";
        return;
    }
    bool FilledCorrectly;
    if (field == "name") FilledCorrectly = FillArticleInfo(TempArticle, val, TempArticle.category);
```

```
    if (field == "category") FilledCorrectly = FillArticleInfo(TempArticle, TempArticle.name, val);
    if (!FilledCorrectly) return;
    WriteSlave(TempArticle, slavefile, slave_adress);
    cout << field << " is updated! New value: " << val << "\n";
}
```

Функція insert-m – вставити новий запис.

```
void InsertM(string name, string pass, std::fstream& masterfile)
{
    User user;
    user.user_id = LastUserID;
    bool FilledCorrectly = FillUserInfo(user, name, pass);
    bool RightPosUsed = false;
    std::streampos InsertPos;
    if (rubbish_m.empty())
    {
        InsertPos = RightMasterPos;
        RightPosUsed = true;
    }
    else
    {
        InsertPos = rubbish_m[rubbish_m.size() - 1];
        rubbish_m.pop_back();
    }
    if (!FilledCorrectly) return;
    WriteMaster(user, masterfile, InsertPos);
    Key k{ LastUserID,InsertPos };
    index_table.push_back(k);
    if (RightPosUsed) RightMasterPos = RightMasterPos + static_cast<std::streamoff>(sizeof(User));
    cout << "new User added! ID: " << LastUserID << ", username: " << name << ", password: " << pass << ", adress: " <<
InsertPos << "\n";
    ++LastUserID;
    ++UserCnt;
}
```

Вводимо ім’я та пароль нашого нового користувача. ID призначаємо за допомогою змінної LastUserID, що автоматично збільшується на 1 при кожному створенні нового користувача. Далі обираємо місце у файлі, куди ми запишемо запис. Якщо ОЗС(див. область збору сміття) порожня, то це – місце після найправішої використаної позиції(тоді цю позицію нам треба пересунути), інакше обираємо останнє із місць з ОЗС.

Функція insert-s – додати новий підзапис до заданого підзапису.

```
void InsertS(int UserID, string name, string category, std::fstream& masterfile, std::fstream& slavefile)
{
    int master_index = GetUserIndex(UserID);
    if (master_index == -1)
    {
        cout << "Error: User with given ID doesn't exist" << "\n";
        return;
    }
    std::streampos user_address = index_table[master_index].address;
    User user;
    ReadMaster(user, masterfile, user_address);
    bool RightPosUsed = false;
    std::streampos PrevSlavePos = -1;
    std::streampos InsertPos,CurPos;
    Article TempArticle;
    if (rubbish_s.empty())
    {
        InsertPos = RightSlavePos;
        RightPosUsed = true;
    }
    else
    {
```

```
        InsertPos = rubbish_s[0];
        rubbish_s.erase(rubbish_s.begin());
    }
    if (user.slave_adress == -1)
    {
        user.slave_adress = InsertPos;
        WriteMaster(user, masterfile, user_address);
    }
    else
    {
        CurPos = user.slave_adress;
        while (CurPos != -1)
        {
            if (!ReadSlave(TempArticle, slavefile, CurPos))
            {
                std::cerr << "Unable to update next_ptr. Error: read failed" << "\n";
                break;
            }
            PrevSlavePos = CurPos;
            CurPos = TempArticle.next;
        }
    }
    TempArticle.article_code = LastArticleID;
    TempArticle.author_code = UserID;
    TempArticle.next = -1;
    bool FilledCorrectly = FillArticleInfo(TempArticle, name, category);
    if (!FilledCorrectly) return;
    WriteSlave(TempArticle, slavefile, InsertPos);
    if (PrevSlavePos != -1)
    {
        ReadSlave(TempArticle, slavefile, PrevSlavePos);
        TempArticle.next = InsertPos;
        WriteSlave(TempArticle, slavefile, PrevSlavePos);
    }
    cout << "new Article added! ID: " << LastArticleID << ", name: " << name << ", category: " << category << ", address: "
<< InsertPos << "\n";
    if (RightPosUsed) RightSlavePos = RightSlavePos + static_cast<std::streamoff>(sizeof(Article));
    ++LastArticleID;
    ++TotalArticleCnt;
}
```

Аналогічно до get-m, маючи ID запису, знаходимо його адресу у master-файлі, переходимо за адресою першого його підзапису та “проходимо” в кінець зв’язаного списка(коли адреса наступного підзапису рівна -1). Аналогічним чином до insert-m, вводимо в консоль значення полів статті(вводимо назву та категорію, ID встановлюємо автоматично), знаходимо місце у slave-файлі, куди запишемо новий підзапис, встановлюємо значення поля next останнього(тепер уже передостаннього) підзапису на цю адресу, за цією адресою записуємо наш підзапис(для нього значення next буде -1, бо ми вставили його в кінець списку).

calc-m – вивести кількість записів.

Для реалізації команди підтримуємо змінну, що спочатку рівна 0, та збільшується/зменшується на 1 при створенні/видаленнні нового запису. При введенні команди просто виводимо її

calc-s – вивести кількість підзаписів заданого запису

«Проходимось» по всіх підзаписах нашого запису, рахуємо їх кількість і виводимо

ut-m – вивести інформацію про всі записи

Перебираємо всі записи за допомогою індексної таблиці і для кожного виконуємо дії, аналогічні до get-m

ut-s – вивести інформацію про всі підзаписи

Перебираємо всі підзаписи даного записа, «проходячись» по ним за допомогою зв’язаного списку і для кожного виконуємо дії, аналогічні до get-s.

Область збору сміття (ОЗС).

При вилученні запису/підзапису його не слід вилучати фізично (великі витрати часу при великих

об’ємах файлів), а лише - логічно. У моїй програмі при їх вилученні запис у файлі за цією ж адресою залишається, але стає недоступним(адреса запису вилучається з індексної таблиці, а при видаленні підзапису уже нема вказівників, які б на нього вказали). Щоб мати можливість

ввести на це місце інший запис при виконанні функції insert адресу

вилученого запису програма зберігає у спеціальній області, яку називають сміттям. В моїй програмі це – два вектори(для записів і підзаписів). При вилученні запису/підзапису його адреса записується у відповідний вектор. При додаванні нового запису/підзапису, якщо відповідний вектор не порожній, то запис додається за однією із записаних туди адрес(і ця адреса вилучається)

Реорганізація

При багатократному видаленні записів та додаванні нових може виникнути ситуація, коли у файлі є лише невелика кількість записів і дуже багато «видалених» комірок, які, проте, досі займають багато місця у файлі(бо їх видалення не було фізичним). Для цього необхідно реалізувати реорганізацію даних – коли розмір ОЗС перевищує деяку константу(у мене це – 3 записи або підзаписи), значення комірок переписуються більш «компактно» на початку файлу, після чого файл обрізається(і всі зайві значення стираються)

У моїй програмі реорганізація відбувається таким чином:

Усі адреси записів(у парі з індексами, що їм відповідають записуються у вектор, який потім сортується за зростанням адрес(зростанням 1го елемента пари {адреса;індекс}). Якщо і-тий запис у цьому векторі знаходиться за адресою (i-1)*sizeof(User), пропускаємо його. Інакше – переносимо на цю адресу. Оскільки ми йдемо по записам у порядку зростання адрес, то втрати даних не відбудеться – ми не запишемо черговий запис на місце ще не перенесеного.

Для цього я реалізував функцію, що переписує підзапис із однієї адреси в іншу:

```
void CopyMaster(std::fstream& masterfile, int MasterIndex, int new_address)
{
   int prev_address = index_table[MasterIndex].address;
   User user;
   ReadMaster(user, masterfile, prev_address);
   index_table[MasterIndex].address = new_address;
   WriteMaster(user, masterfile, new_address);
}
```

З підзаписами принцип аналогічний, проте, крім перенесення самого підзапису, потрібно розшукати попередній підзапис і змінити йому значення поля next, встановивши на нову адресу(або, якщо попереднього підзапису нема і наш підзапис перший, встановити на нову адресу значення поля slave_adress для записа, якому належить підзапис)

Так виглядає функція перенесення за новою адресою підзапису, якщо нам уже відома адреса попереднього підзапису:

```
void CopySlave(std::fstream& masterfile, std::fstream& slavefile, int MasterIndex, int prev_adress, int cur_adress, int
next_adress)
{
   int user_adress = index_table[MasterIndex].address;
   User user;
   Article tmp;
   ReadMaster(user, masterfile, user_adress);
   ReadSlave(tmp, slavefile, cur_adress);
```

```
    if (prev_adress == -1)
    {
        user.slave_adress = next_adress;
        WriteMaster(user, masterfile, user_adress);
        WriteSlave(tmp, slavefile, next_adress);
    }
    else
    {
        WriteSlave(tmp, slavefile, next_adress);
        ReadSlave(tmp, slavefile, prev_adress);
        tmp.next = next_adress;
        WriteSlave(tmp, slavefile, prev_adress);
    }
}
```

А ось функція виконання всієї реорганізації. Спочатку відбувається реорганізація записів, потім підзаписів(для кожного підзапису ми шукаємо попередній, «проходячись» по зв’язаному списку, що відповідає запису):

```
void Rewrite(std::fstream& masterfile, std::fstream& slavefile)
{
    User user;
    Article tmp;
    std::streampos NewMasterPos = 0;
    std::streampos NewSlavePos = 0;
    std::streampos NextPos, PrevPos;
    int UserNum;
    vector<pair<int, int> >vm;
    vector<pair<int, int> >vs;
    int UserCnt = index_table.size();
    for (int i = 0; i < UserCnt; ++i) vm.push_back({ index_table[i].address,i });
    sort(vm.begin(), vm.end());
    for (int i = 0; i < UserCnt; ++i)
    {
        UserNum = vm[i].second;
        if (NewMasterPos != vm[i].first) CopyMaster(masterfile, UserNum, NewMasterPos);
        NewMasterPos = NewMasterPos + static_cast<std::streamoff>(sizeof(User));
    }
    for (int i = 0; i < UserCnt; ++i)
    {
        ReadMaster(user, masterfile, index_table[i].address);
        if (user.slave_adress != -1)
        {
            NextPos = user.slave_adress;
            ReadSlave(tmp, slavefile, NextPos);
            vs.push_back({ NextPos,i });
            NextPos = tmp.next;
            while (NextPos != -1)
            {
                ReadSlave(tmp, slavefile, NextPos);
                vs.push_back({ NextPos,i });
                NextPos = tmp.next;
            }
        }
    }
    sort(vs.begin(), vs.end());
    for (int i = 0; i < vs.size(); ++i)
    {
        UserNum = vs[i].second;
        if (NewSlavePos != vs[i].first)
        {
            ReadMaster(user, masterfile, index_table[UserNum].address);
            PrevPos = user.slave_adress;
            ReadSlave(tmp, slavefile, PrevPos);
            while (tmp.next != -1 && tmp.next != vs[i].first)
            {
                PrevPos = tmp.next;
                ReadSlave(tmp, slavefile, PrevPos);
```

```
            }
            if (tmp.next == -1) PrevPos = -1;
            CopySlave(masterfile, slavefile, UserNum, PrevPos, vs[i].first, NewSlavePos);
        }
        NewSlavePos = NewSlavePos + static_cast<std::streamoff>(sizeof(Article));
    }
}
```

Обробка вхідних даних

За допомогою усіх цих функцій можна обробляти вхідні дані(які, по суті, є командами). Для правильного виконання, команда має чітко відповідати синтаксису

```
"get-m [userID]: finds and writes info about a User with given ID" << "\n";
"get-s [articleID]: finds and writes info about the Article with a given ID" << "\n";
"del-m [userID]: finds and deletes info about the User with given ID and all of his articles" << "\n";
"del-s [articleID]: finds and deletes info about the Article with given ID" << "\n";
"update-m [userID] [field] [value]: updates the value of the field of a certain User" << "\n";
"update-s [articleID] [field] [value]: updates the value of a certain Article" << "\n";
"insert-m [username] [password]: inserts the new User with given parameteres" << "\n";
"insert-s [userID] [name] [category]: inserts the new Article with given parameteres to the User with given ID" << "\n";
"calc-m: calculates the number of Users" << "\n";
"calc-s [userID]: calculates the number of Articles written by a User with given ID" << "\n";
"ut-m: writes info about all Users" << "\n";
"ut-s [userID]: writes info about all Articles written by a User with given ID. Enter `ut-s all` to get info about ALL Articles"
<< "\n";
"exit: ends the program" << "\n";
```

Якщо ж введена команда йому не відповідає, програма має продовжити роботу, вивівши повідомлення про те, що введена команда неправильна. Оскільки кожна команда вводиться у новому рядку, то вхідні дані я зчитую по рядках за допомогою функції cin.getline(див. Основна програма). Зчитавши черговий рядок, я виокремлюю з нього «слова»(що розділені пробілами) за допомогою наступної функції:

```
std::vector<std::string> DivideLine(const std::string& line)
{
    std::vector<std::string> parts;
    std::istringstream iss(line);
    std::string part;
    while (iss >> part) parts.push_back(part);
    return parts;
}
```

Після цього можна уже зрозуміти, що робити з командою далі, оскільки її тип визначається першим «словом». Якщо воно відповідає якомусь із допустимих(get-m,get-s,del-m,del-s,update-m,update-s,insert-m,insert-s,ut-m,ut-s,calc-m,calc-s), ми перевіряємо, чи є кількість «слів» у команді такою ж, як у синтаксисі(наприклад, для insert-m їх має бути 3 – власне insert-m, ім’я та пароль користувача, які ми хочемо вставити). Якщо їх кількість не така, як треба, команда неправильна. Якщо якесь із цих «слів» не відповідає певним критеріям(наприклад, ID має бути числом, тобто містити лише цифри), команда неправильна. Якщо ж команда правильна, то можна викликати відповідну функцію для виконання команди, передавши туди уведені дані.

До речі, для конвертації рядка у число я використовував наступну функцію(повертає -1, якщо рядок містить не лише цифри):

```
int StrToInt(string& s)
{
    int num = 0;
    for (int i = s.size() - 1; i >= 0; --i)
    {
        if (s[i] < 48 || s[i]>57)
        {
            s = "error";
            return -1;
        }
```

```
        num = 10 * num + (s[i] - 48);
    }
    return num;
}
```

Якщо ж перше «слово» не відповідає жодній з команд, то команда теж неправильна

Це все опрацьовується у функції

```
void ProcessCommand(const std::string& line, std::fstream& masterfile, std::fstream& slavefile, bool& NotFinished)
```

(у неї передаємо черговий уведений рядок)

Основна програма

Нарешті, опишемо, як виглядає основна програма – int main()

Створюємо та відкриваємо бінарні файли:

```
const std::string slavename = "SP.bin";
const std::string mastername = "Sfl.bin";
const std::string tablename = "Sind.bin";
string command;
bool NotFinished=true;
std::fstream slavefile(slavename, std::ios::binary | std::ios::in | std::ios::out | std::ios::trunc);
std::fstream masterfile(mastername, std::ios::binary | std::ios::in | std::ios::out | std::ios::trunc);
std::fstream indextable(tablename, std::ios::binary | std::ios::in | std::ios::out | std::ios::trunc);
auto err = errno;
if (err == ENOENT)
{
    slavefile = std::fstream(slavename, std::ios::binary | std::ios::in | std::ios::out | std::ios::trunc);
    masterfile = std::fstream(mastername, std::ios::binary | std::ios::in | std::ios::out | std::ios::trunc);
    indextable = std::fstream(tablename, std::ios::binary | std::ios::in | std::ios::out | std::ios::trunc);
}
if (!slavefile)
{
    std::cerr << "Unable to open file=" << slavename << "\n";
    return -1;
}
if (!masterfile)
{
    std::cerr << "Unable to open file=" << mastername << "\n";
    return -1;
}
if (!indextable)
{
    std::cerr << "Unable to open file=" << tablename << "\n";
    return -1;
}
```

Після цього, запускаємо цикл while, що на кожній ітерації зчитує новий рядок. Якщо у одній із ОЗС набралось більше 3 елементів, запускаємо реорганізацію(а також обрізаємо файли та оочищуємо обидві ОЗС)

Далі за допомогою ProcessCommand опрацьовуємо введену команду. Окрім 12, які необхідно реалізувати для лаби, є ще дві: help – виводить на екран список усіх доступних команд та exit – перериває цикл і завершує програму.

Перед завершенням програми індексна таблиця переписується з масиву у файл:

```
std::streampos CurPos = 0;
for (int i = 0; i < index_table.size(); ++i) WriteKey(index_table[i], indextable, CurPos);
```

Дякую за увагу ☺