|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_Системы обработки информации и управления \_\_\_\_\_\_

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Герасимов Андрей Денисович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа\_\_\_\_\_\_\_\_ИУ5-24Б\_\_\_\_\_\_\_\_\_\_\_

Тип практики \_\_\_\_\_\_\_\_\_\_\_Проектно-технологическая практика\_\_\_\_\_\_\_\_\_\_

Название предприятия\_\_\_\_\_\_\_\_\_\_ МГТУ им. Н. Э. Баумана\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент **\_\_\_\_\_**24.07.21**\_\_\_\_ \_\_**Герасимов А.Д.**\_\_\_**

*подпись, дата фамилия, и.о.*

Руководитель практики **\_\_\_\_\_**24.07.21**\_\_\_\_\_ \_**Правдина А. Д.**\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2021г.*

**Постановка задачи**

1. ***Цель и содержание практики***

Целями учебной практики «Проектно - технологическая практика» являются:

- закрепление и углубление теоретической подготовки студентов покурсам «Информатика», «Основы программирования» и «Программирование на основе классов и шаблонов»,

- разработка библиотеки функций или классов для указанной в задании предметной области,

- использование разработанной библиотеки при выполнении индивидуальных заданий КЛР.

.

***Содержание***

Содержание практики.

* Разработать и записать в текстовые файлы таблицы с данными тестового примера для базы данных БИБЛИОТЕКА. Каждая таблица хранится в отдельном файле. Данные хранятся в формате CSV с разделителем ‘|’. Для записи тестовых данных в файл использовать текстовый редактор MS Visual C++. Первая строка файла содержит заголовок таблицы, а остальные строки – данные.
* Разработать структуру DBTable, способную хранить данные произвольных таблиц.
* Разработать функцию readTable() для чтения данных из файла в структуру DBTable (доработать функцию readTable() из ЛР 1так, чтобы она считывала в DBTable не только заголовки, но и строки таблицы).

- Разработать функцию printTable() для печати данных из структуры DBTable

- Разработать функцию writeTable() для записи данных из структуры DBTable в файл в формате CSV;

- Разработать функцию main() для тестирования функций. Для вызова тестируемых функций из main() использовать меню.

* Создать файл DBLibrary.cpp, включающий структуру Table и функции readTable(), writeTable(), printTable(), и подключить его к проекту.
* Перейти от структуры DBTable к классу DBTable.
* Разработать класс DBTableSet, хранящий все таблицы БД.

- Добавить в классы методы, необходимые для выполнения варианта задания.

-Установить связи между таблицами.

- Разработать функции для выполнения индивидуального задания КЛР, использующие методы разработанных классов.

Для вызова функций из main() использовать меню.

***Общие указания***

Перед началом программирования разработать тестовый пример. В качестве данных для тестового примера использовать таблицы БД, указанной в задании на практику.

***База данных библиотеки*** ***DBLibrary*** содержит служебную таблицу DBTables с именами всех таблиц БД и 4 таблицы с данными:

Abonements, Students, Books и Subjects.

Поля таблиц для БД Library:

Таблица Students: StudentID, Name, Group.

Таблица Subjects: SubjectID, Subject, где:

Subject – тематика книги ( физика, математика, информатика и т.д.)

Таблица Books: BookID, Title, SubjectID, Quantity, Number, где:

Quantity – количество поступивших в библиотеку экземпляров данной книги.

Number – количество выданных экземпляров данной книги.

Таблица Abonements: StudentID, BookID, OutDate, InDate, где:

OutDate, InDate – дата выдачи, дата возврата книги.

Заполнить файлы тестовыми данными. Ввести не менее трех книг по каждому из трех предметов, не менее 5 записей в таблицу Students и не менее 10 записей в таблицу Abonements.

При заполнении таблицы данными проверять уникальность значений идентификаторов и ссылочную целостность данных, то есть наличие в основных (родительских) таблицах Students, Books и Subjects значений идентификаторов StudentID, BookID и SubjectID, использованных в дочерних таблицах Abonements и Books.

***База данных DBCompany*** компании по оптовой торговле содержит служебную таблицу DBTables с именами всех таблиц БД и 4 таблицы с данными:

Orders, OrderDetails, Customers, Products.

Поля таблиц для БД ***DBCompany***:

Customers: CustomerID, CompanyName, ContactName

OrderDetails: ID, OrderID, ProductID, UnitPrice, Quantity

Orders: OrderID, CustomerID,|OrderDate

Products: ProductID, UnitPrice, UnitInStock

Базы данных хранятся на диске в текстовых файлах с разделителями (формат CSV).

Каждая таблица хранится в отдельном файле, <имя файла>=<имя таблицы>.txt.

Спецификация класса DBTable приведена ниже в Приложении 1.

**Вариант 2**

Выдача книги в библиотеке. На одних руках не должно быть двух книг по одному предмету. Перед выдачей сделать проверку наличия экземпляра книги в книгохранилище.

***3. Рекомендации по выполнению работы***

**3.1. Шаги разработки программы**

3.1.1. Разработка структуры таблиц для хранения данных в файле и создание в текстовом редакторе файлов с табличными данными.

3.1.2. Разработка структуры DBTable для хранения данных в ОП.

3.1.3. Разработка спецификаций функций для чтения данных из файла в структуру DBTable, печати считанных данных и записи данных из структуры DBTable в файл.

3.1.4. Кодирование и отладка функций чтения, печати и записи данных.

3.1.5. Разработка структуры DBTableSet для хранения данных всех таблиц БД в ОП.

3.1.6. Разработка спецификаций функций для чтения, печати и записи данных для структуры DBTableSet.

3.1.7. Кодирование и отладка функций чтения, печати и записи данных для структуры DBTableSet.

*Приложение 1.* Файл DBTable.h

#include <map>

#include <string>

#include "DBDate.h"

using namespace std;

typedef map<string, void\*> Row;

typedef map<string, string> Header;

class DBTable

{

private:

string tableName; //имя таблицы

Header columnHeaders; //заголовок таблицы

vector<Row> data;//вектор из строк (записей) таблицы

void\* GetValue(string value, string columnName); //возвращает

// указатель void\*, инициализированный адресом

//значения переменной value, преобразованной из типа

//string в тип typeName, имя которого указано в заголовке

//столбца таблицы в CVS-файле.

public:

DBTable(){} //конструктор по умолчанию

~DBTable(){} //деструктор

DBTable(string tabName) //конструктор с параметром

{

tableName = tabName;

}

string valueToString(Row& row,string columnName); //возвращает //значение переменной value, преобразованное из типа

//typeName, имя которого указано в заголовке columnName столбца

//таблицы, в тип string.

void ReadTable(string tabName);

void PrintTable(int numcol);

void WriteTable();

int GetSize() //возвращает число строк в таблице

{

return (int)data.size();

}

Row& operator[](int ind) //перегрузка операции индексации

{

return data[ind];

}

int GetType(string columnName); //возвращает код типа данных

//в столбце columnName таблицы

bool ValueIsEqual(string columnNameA, void\* objA, void\* objB);

//сравнивает значения данных в столбце columnNameA двух строк //таблицы

vector<Row> SelfRows(string columnName, void\* value);

//возвращае вектор из строк таблицы, в которых значение данных

//в столбце columnName равно \*value

};

Шаблонный класс map<Type1,Type2> - это ассоциативный контейнер, построенный на основе бинарного дерева, в узлах которого хранятся пары значений типа pair<Type1,Type2>.

Type1 – тип ключа, Type2 – тип значения. Значение ключа должно быть уникальным в пределах контейнера. В качестве ключа могут использоваться только те типы, для которых определены операции сравнения.

Контейнер ***map*** можно рассматривать как массив, упорядоченный по значению ключа. Упорядочивание производится при записи данных в контейнер. Например, для добавления значения столбца value в строку заголовка columnName можно использовать оператор:

columnHeaders [columnName]=value;

**Текст программы**

***main.cpp(Главный файл)***

#include <iostream>

#include <iomanip>

#include <Windows.h>

#include"Header.h"

using namespace std;

int menu() {

cout << "================= МАКЕТ СУБД ===================\n";

cout << "\t1 - Чтение таблиц из файла\n";

cout << "\t2 - Печать таблиц\n";

cout << "\t3 - Выдача книги\n";

cout << "\t4 - Выход\n";

int choice;

cout << "Выберите действие\n";

cin >> choice;

while (cin.fail()) {

cout << "Ошибка ввода. Повторите ввод\n";

cin.clear();

cin.ignore(10, '\n');

cin >> choice;

}

return choice;

}

void Read(dbmsLib::DBTable\* students, dbmsLib::DBTable\* subjects, dbmsLib::DBTable\* books, dbmsLib::DBTable\* abonements)

{

students->ReadDBTable("Students.txt");

students->PrintTable(80);

abonements->ReadDBTable("Abonements.txt");

abonements->PrintTable(80);

books->ReadDBTable("Books.txt");

books->PrintTable(80);

subjects->ReadDBTable("Subjects.txt");

subjects->PrintTable(80);

}

void Print(dbmsLib::DBTable\* students, dbmsLib::DBTable\* subjects, dbmsLib::DBTable\* books, dbmsLib::DBTable\* abonements)

{

students->PrintTable(80);

subjects->PrintTable(80);

books->PrintTable(80);

abonements->PrintTable(80);

}

void allow(dbmsLib::DBTable\* students, dbmsLib::DBTable\* subjects, dbmsLib::DBTable\* books, dbmsLib::DBTable\* abonements) {

string name;

cout << "Введите фамилию студента\n";

cin >> name;

string group;

cout << "Введите группу студента\n";

cin >> group;

bool flag = false,

available = true;

string bookSubject;

for (int i = 0; i < students->GetSize(); i++) {

dbmsLib::Row studRow = students->GetRow(i);

string studName = \*((string\*)studRow["Name"]);

string studGroup = \*((string\*)studRow["Group"]);

if (name == studName && group == studGroup) {

flag = true;

int studID = \*((int\*)studRow["StudentID"]);

cout << "Студент найден!\n";

string bookTitle;

cout << "Введите название книги: ";

cin >> bookTitle;

if (!bookAvailable(abonements, books, subjects, bookTitle, bookSubject)) {

cout << "Книги нет в наличии!\n";

return;

}

if (AbonementChecker(abonements, subjects, books, studID, bookSubject)) {

cout << "Книга может быть выдана\n";

//добавить в данные книги +1

}

else {

cout << "У студента есть книга по тому же предмету! Книга не может быть выдана!\n";

}

}

}

if (!flag) {

cout << "Студент не найден, вы будете возвращены в меню\n";

}

}

bool bookAvailable(dbmsLib::DBTable\* abonements, dbmsLib::DBTable\* books, dbmsLib::DBTable\* subjects, string bookTitle, string& bookSubject) {

for (int j = 0; j < books->GetSize(); j++) {

dbmsLib::Row booksRow = books->GetRow(j);

string curBookName = \*((string\*)booksRow["Title"]);

if (curBookName == bookTitle) {

int quantity = \*((int\*)booksRow["Quantity"]);

int num = \*((int\*)booksRow["Number"]);

if (quantity == 0 || num == quantity) {

return false;

}

int subjectID = \*((int\*)booksRow["SubjectID"]);

for (int j = 0; j < subjects->GetSize(); j++) {

dbmsLib::Row subjectsRow = subjects->GetRow(j);

int curSubjID = \*((int\*)subjectsRow["SubjectID"]);

if (subjectID == curSubjID) {

bookSubject = \*((string\*)subjectsRow["Subject"]);

}

}

}

}

}

bool AbonementChecker(dbmsLib::DBTable\* abonements, dbmsLib::DBTable\* subjects, dbmsLib::DBTable\* books, int studID, string bookSubject) {

for (int i = 0; i < abonements->GetSize(); i++) {

dbmsLib::Row abonRow = abonements->GetRow(i);

int curStudentID = \*((int\*)abonRow["StudentID"]);

if (curStudentID == studID) {

int bookID = \*((int\*)abonRow["BookID"]);

if (!bookChecker(books, subjects, bookID, bookSubject)) {

return false;

}

return true;

}

}

//добавить абонемент

return true;

}

bool bookChecker(dbmsLib::DBTable\* books, dbmsLib::DBTable\* subjects, int bookID, string bookSubject) {

for (int j = 0; j < books->GetSize(); j++) {

dbmsLib::Row booksRow = books->GetRow(j);

int curBookID = \*((int\*)booksRow["BookID"]);

if (bookID == curBookID) {

int subjectID = \*((int\*)booksRow["SubjectID"]);

return (subjectChecker(subjects, subjectID, bookSubject));

}

}

return true;

}

bool subjectChecker(dbmsLib::DBTable\* subjects, int subjectID, string bookSubject) {

for (int j = 0; j < subjects->GetSize(); j++) {

dbmsLib::Row subjectsRow = subjects->GetRow(j);

int curSubjID = \*((int\*)subjectsRow["SubjectID"]);

if (subjectID == curSubjID) {

string subjectName = \*((string\*)subjectsRow["Subject"]);

if (bookSubject == subjectName) {

return false;

}

return true;

}

}

}

int main() {

cout << left;

system("chcp 1251>nul");

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

dbmsLib::DBTableTxt students("Students.txt");

dbmsLib::DBTableTxt subjects("Subjects.txt");

dbmsLib::DBTableTxt books("Books.txt");

dbmsLib::DBTableTxt abonements("Abonements.txt");

while (true) {

switch (menu()) {

case 1:

Read(&students, &subjects, &books, &abonements);

break;

case 2:

Print(&students, &subjects, &books, &abonements);

break;

case 3:

allow(&students, &subjects, &books, &abonements);

break;

case 4: return 0;

default:cout << "Недопустимое действие. Повторите выбор\n";

break;

}

}

return 0;

}

***dbmsLib.h***

#pragma once

//файл dbmsLib.h

#pragma once

#ifndef \_dbmsLib\_

#define \_dbmsLib\_

#include <iostream>

#include <fstream>

#include <strstream>

#include <iomanip>

#include <string>

#include <vector>

#include <map>

using namespace std;

namespace dbmsLib {

//====================класс DBDate======================

class DBDate {

friend string DateToStr(DBDate& date);

friend ostream& operator<<(ostream& out, DBDate& date);

int day, month, year;

bool IsLeapYear(int year); //год високосный?

int GetDaysInMonth(int month, int year);//Количество дней в месяце

int DaysInCurYear();//Количество дней от начала года до текущей даты

public:

DBDate(string date);//формат строки: dd.mm.yyyy

DBDate(int d, int m, int y);

DBDate() :day(0), month(0), year(0) {};//???конструктор по умолчанию

DBDate(DBDate& dat) :day(dat.day), month(dat.month), year(dat.year) {}

int GetDay();

int GetMonth();

int GetYear();

bool operator==(DBDate& date);

bool operator<(DBDate& date);

bool operator>(DBDate& date);

bool operator<= (DBDate& date);

bool operator>= (DBDate& date);

bool operator!= (DBDate& date);

DBDate& operator+=(int days);//Прибавляет к текущей дате days дней

DBDate& operator-=(int days);//Вычитает из текущей даты days дней

int operator-(DBDate& date); //Количество дней между текущей датой и date

//Если текущая дата > date, результат < 0.

};

//DBType-перечисление типов полей таблиц БД:

enum DBType { NoType, Int32, Double, String, Date };

//typeNames - массив имен типов.

//Используется для преобразования имени типа (char\*) в объект DBType

static const char\* typeNames[] = {

"NoType",

"Int32",

"Double",

"String",

"DBDate"

};

//Condition-перечисление условий, по которым могут сравниваються поля таблиц БД:

enum Condition { Undefined, Equal, NotEqual, Less, Greater, LessOrEqual, GreaterOrEqual };

const int LENGTH = 24;//длина имени таблицы и имени столбца.

//ColumnDesc - описание данных в столбце таблицы

//данные типа string имеют одинаковую длину (length) внутри одного столбца,

//но в разных столбцах их длина может отличаться

struct ColumnDesc {

char colName[LENGTH];//имя столбца

DBType colType;//тип данных в столбце таблицы

int length; //максимальное число символов, допустимое

//для представления данных в столбце

};

struct Strip {//полоса распечатки таблицы

int nField;//число полей

int\* fieldWidth;//ширина полей в полосе (массив)

};

typedef map<string, void\*> Row;

typedef map<string, ColumnDesc> Header;

//-----------------класс DBDate----------------------------

class DBTable {

public:

DBType GetType(char\* columnName);

const char\* TypeName(DBType type);

virtual ~DBTable() {};//виртуальный деструктор

virtual Header GetHeader() = 0;

virtual vector<int> IndexOfRecord(void\* keyValue, string keyColumnName) = 0;

//по значению ключа определяет

//порядковый номер (индекс) строки (записи) в data.

// keyValue-значение ключа (указатель на string, int или DBDate)

// keyColumnName-имя ключевого столбца (первичный или вторичный ключ)

//в таблице может быть несколько записей с одинаковым значением вторичного ключа.

virtual Row& GetRow(int index) = 0;

virtual Row operator[](int index) = 0;

virtual string valueToString(Row& row, string columnName) = 0;

virtual int GetSize() = 0;

virtual void SetHeader(Header& hdr) = 0;

virtual void ReadDBTable(string fileName) = 0;//fileName="..\\"+dbName+"\\"+tableName+".bin"|".txt"

virtual void WriteDBTable(string fileName) = 0;

virtual void SetTableName(string tName) = 0;

virtual void SetPrimaryKey(string key) = 0;

virtual string GetTableName() = 0;

virtual string GetPrimaryKey() = 0;

virtual void SetFileName(string path) = 0;

virtual string GetFileName() = 0;

virtual void PrintTable(int screenWidth) = 0;

virtual Row CreateRow() = 0;

virtual void AddRow(Row row, int index) = 0;

virtual DBTable\* SelfRows(string colName, Condition cond, void\* value) = 0;

};

string GetTabNameFromPath(string path);

string ignoreBlanc(const string str);

void\* GetValue(string value, string columnName, Header hdr);

void\* SetValue(string value, string columnName, Header hdr);

bool comparator(/\*TableDataType\*/DBType type, void\* obj1, Condition condition, void\* obj);

int GetLength(ColumnDesc colDesc);

//===============class DBTableTxt=============================

class DBTableTxt :public DBTable {

private:

Header columnHeaders;

string tableName;

string primaryKey;

vector<Row> data;

string fileName;

public:

DBTableTxt() {}

DBTableTxt(string tabName);/\*: DBTable(tabName){}\*/

DBTableTxt(string tabName, Header hdr, string primKey);/\*:DBTable(tabName,hdr,primKey){}\*/

~DBTableTxt();//!!!деструктор перегружен,чтобы память,выделенная void\*,не утекала

vector<int> IndexOfRecord(void\* keyValue, string keyColumnName);

string valueToString(Row& row, string columnName);

void ReadDBTable(string tabName);//tabName=path+tableName

void ReadTableBin(string tabName);//чтение таблицы из бинарного файла

void PrintTable(int screenWidth);

void WriteDBTable(string tabName);//tabName=path+tableName

void WriteTableBin(string tabName);//запись таблицы в бинарный файл

int GetSize();

Row operator[](int ind);

void SetFileName(string path);

void SetTableName(string tName);

void SetPrimaryKey(string key);

string GetFileName();

string GetTableName();

string GetPrimaryKey() { return primaryKey; }

Header GetHeader();

void SetHeader(Header& hdr);

Row CreateRow();

Row& GetRow(int index);

void AddRow(Row row, int index);

DBTable\* SelfRows(string columnName, Condition cond, void\* value);

//Макет распечатки таблицы:

//screenWidth - ширина экрана (входной параметр)

//nStrips - число полос в распечатке (выходной параметр)

//strips[nStrips] - описание полос таблицы:

//число столбцов и ширина каждого столбца в полосе (выходной параметр)

void CreateTableMaket(Strip\*& strips, int& nStrips, int screenWidth);

friend void ReadDBTable1(DBTableTxt& tab, string tabName);//tabName=path+tableName

friend void WriteDBTable1(DBTableTxt& tab, string tabName);

friend void PrintTable1(DBTableTxt& tab, int screenWidth);

//запись таблицы в бинарный файл:

friend void WriteTableBin1(DBTableTxt& tab, string tabName);

//чтение таблицы из бинарного файла:

friend void ReadTableBin1(DBTableTxt& tab, string tabName);

friend void CreateTableMaket1(Strip\*& strips, int& nStrips, int screenWidth);

};

//=====================Класс DBTableBin=====================================

class DBTableBin : public DBTable {

char tabName[LENGTH];

char primaryKey[LENGTH];//имя столбца со значениями первичного ключа

int nColumn;

ColumnDesc\* header;

int nRows;

int RowLength();

int DataBegin();

int FieldPosition(string colName);//смещение поля colName от начала строки в байтах

int FieldLength(string colName);//длина поля colName в байтах

void ResizeData(int deltaRows);

char\*\* data;

int maxRows;//число строк в data с учетом DELTA (при считывании таблицы//????int maxRows()

//из файла устанавливается равным nRows+DELTA).В файл не записывается

char fileName[80];//В файл не записывается. Используется в методах для

//обращения к файлу

void CreateTableMaket(Strip\*& strips, int& nStrips, int screenWidth);

public:

DBTableBin() {}

DBTableBin(string tableName);

DBTableBin(string tableName, Header hdr, string primKey);

~DBTableBin() {/\*cout<<"~DBTableBin()"<<endl;\*/ }

Header GetHeader();

vector<int> IndexOfRecord(void\* keyPtr, string keyName);

//по значению ключа определяет

//порядковый номер (индекс) строки (записи) в data.

// keyPtr-значение ключа (указатель на string, int или DBDate)

// keyName-имя ключевого столбца (первичный или вторичный ключ)

//в таблице может быть несколько записей с одинаковым значением вторичнего ключа.

int FieldPosInFile(int rowIndex, string columnName);

Row& GetRow(int index);

Row operator[](int index);

string valueToString(Row& row, string columnName);

int GetSize() { return nRows; }

void SetHeader(Header& hdr);

void ReadDBTable(string fileName);//fileName="..\\"+dbName+"\\"+tableName+".bin"

void WriteDBTable(string fileName);

void SetTableName(string tName);

void SetPrimaryKey(string key);

string GetTableName() { return tabName; }

string GetPrimaryKey() { return primaryKey; }

void SetFileName(string path) {

strcpy\_s(fileName, 80, (path + "//" + tabName + ".bin").c\_str());

}

string GetFileName() { return fileName; }

void PrintTable(int screenWidth);

Row CreateRow();

void AddRow(Row row, int index);

DBTable\* SelfRows(string colName, Condition cond, void\* value);

};

//==============Класс DBTableSet============================

struct Relation {//описывает связь двух таблиц

string relationName;

string parentTable;

string parentPrimaryKey;

string childTable;

string childSecondaryKey;

};

void PrintRow(Row row, DBTable\* tab);

//-------------------------класс DBTableSet---------------------

class DBTableSet

{

private:

string dbName;

map<string, DBTable\*> db;

public:

DBTableSet() {};

DBTableSet(string name);

string FileType();//".Txt" или ".Bin"

int ReadDB();

void PrintDB(int numcol);

void WriteDB();

string GetDBName() { return dbName; }

DBTable\* operator[](string tableName);

Relation GetRelation(string relationName);

Row ParentRow(Relation& relation, Row& childRow);

DBTable\* ChildRows(Relation& relation, Row& parentRow);

};

const int DELTA = 10;//запас в 10 строк для новых записей в таблицу.

//Если запас израсходован, то для его восстановления нужно увеличить

//на DELTA размер динамического массива char\* data[].

//const int LENGTH = 24; длина имени таблицы и имени столбца.

string GetTabNameFromPath(string path);

string ignoreBlanc(const string str);

void\* GetValue(string value, string columnName, Header hdr);

void\* SetValue(string value, string columnName, Header hdr);

bool comparator(/\*TableDataType\*/DBType type, void\* obj1, Condition condition, void\* obj);

int GetLength(ColumnDesc colDesc);

}

void allow(dbmsLib::DBTable\* students, dbmsLib::DBTable\* subjects, dbmsLib::DBTable\* books, dbmsLib::DBTable\* abonements);

bool bookAvailable(dbmsLib::DBTable\* abonements, dbmsLib::DBTable\* books, dbmsLib::DBTable\* subjects, string bookTitle, string& bookSubject);

bool AbonementChecker(dbmsLib::DBTable\* aboniments, dbmsLib::DBTable\* subjects, dbmsLib::DBTable\* books, int studID, string bookSubject);

bool bookChecker(dbmsLib::DBTable\* books, dbmsLib::DBTable\* subjects, int bookID, string bookSubject);

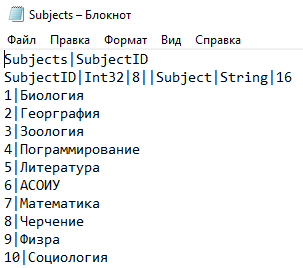
bool subjectChecker(dbmsLib::DBTable\* subjects, int subjectID, string bookSubject);

#endif //конец

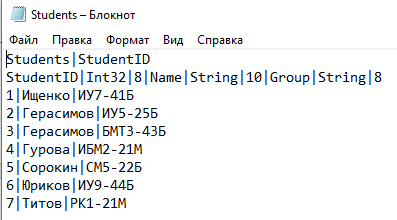
**Тестирование**

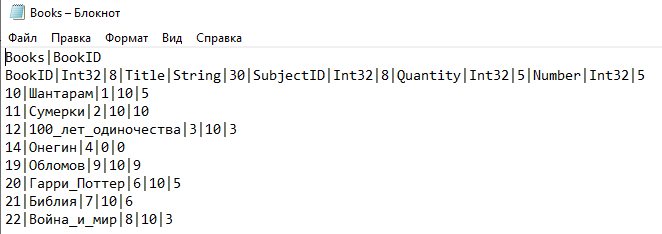
Исходные файлы:

Subjects.txt:

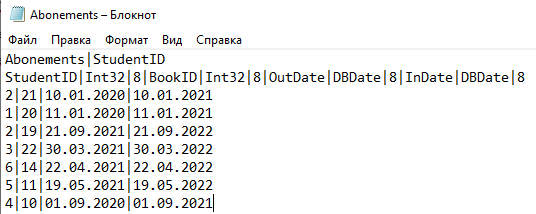


Students.txt:

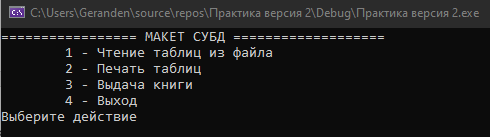


Books.txt: 

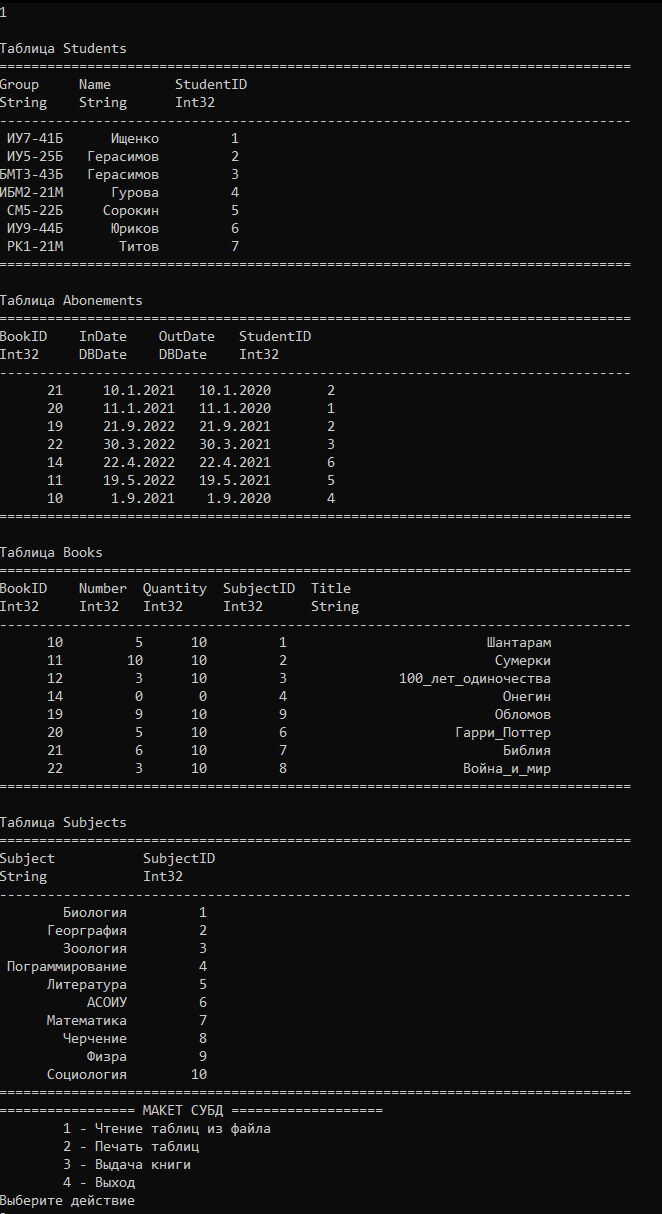
Abonements.txt:

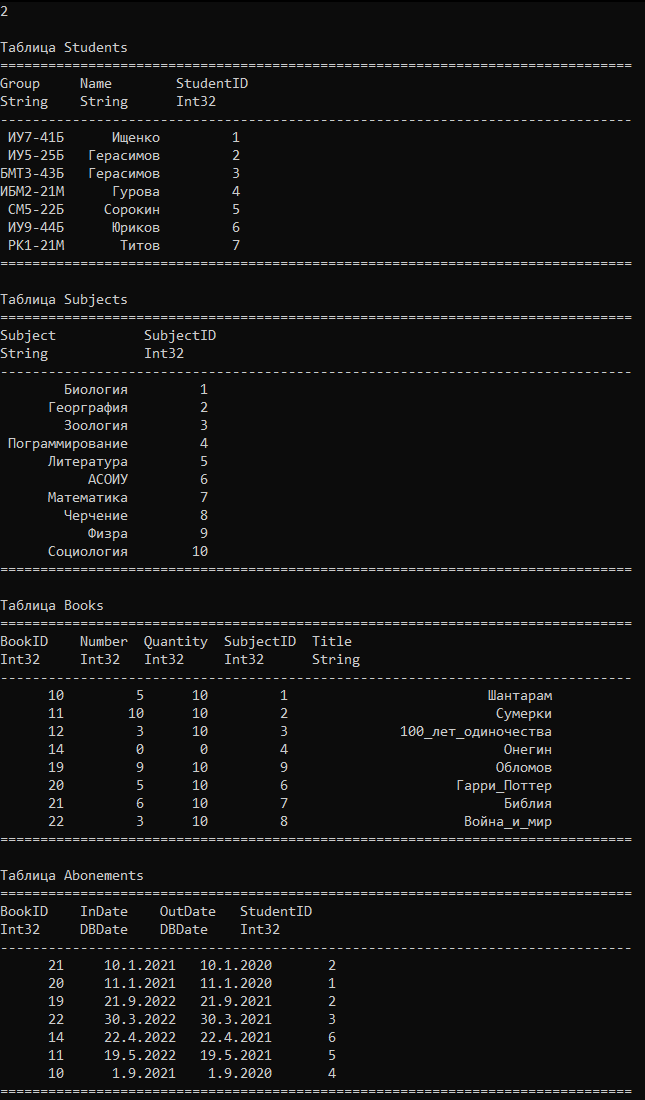


Главное меню:



Чтение:



Печать таблицы: 

Запись в файл:

