# ПКШ2020\_ЛР5. Создание бинарных файлов с таблицами БД.

# Цель работы

Создание бинарных файлов с таблицами БД.

# Порядок выполнения работы

## Введение

Библиотека классов для макета СУБД включает два класса для работы с таблицами БД – *DBTableTxt* и *DBTableBin*.   
Разработку библиотеки мы начали с класса DBTableTxt, так как при работе с текстом создавать таблицы БД для тестового примера и контролировать результаты работы можно в текстовом редакторе.  
Но при обмене данными с текстовыми файлами каждый раз приходится выполнять преобразование типов.

В чем отличие текстовых файлов от бинарных?   
Текстовые файлы хранят коды символов, а бинарные – копию байтов ОП.  
 Например, если мы записываем в текстовый файл целое число 257, то в файл запишутся двоичные коды трёх символов:   
`2`, `5` и `8` -> 00110010 00110101 00110111 (32 35 38 в шестнадцатеричной системе счисления).   
Если мы записываем в бинарный файл целое число 258, то в файл запишутся 4 байта двоичного кода числа 258 (00000000 00000000 00000001 00000010) в том виде, как они хранятся в ОП, то есть младшими байтами вперёд, а в байте сначала старшая тетрада, а за ней младшая.   
При открытии бинарного файла в редакторе файлов для удобства работы с ним двоичные данные выводятся на экран в шестнадцатеричной системе счисления:  
258 -> 02 01 00 00.

В С++ можно открыть текстовый файл в бинарном режиме, но от этого он не станет бинарным. Из сказанного выше должно быть понятно, что текстовые и бинарные файлы отличаются по содержанию.

Создавать *бинарные* файлы с таблицами БД и читать их содержимое намного сложнее, чем создавать аналогичные текстовые файлы. Но это нужно сделать для разработки тестовых примеров при отладке методов класса *BTableBin*.

Зато при обмене данными с бинарными файлами не нужно выполнять преобразование типов, так как бинарные файлы хранят копию данных из ОП.  
Кроме того, для бинарных файлов можно организовать прямой доступ к данным, то есть можно считывать в память и записывать в файл не всю таблицу, а только нужные строки.  
Это очень важно при работе с таблицами с большим числом строк.

Для реализации прямого доступа к данным, хранящимся в таблице, недостаточно записать таблицу в бинарный файл. Необходимо, чтобы данные в каждом столбце таблицы были бы одинакового размера.   
В таблицах БД используются данные 4-х типов:   
*int, double, DBDate* и *string*.   
Для первых трех типов это условие выполняется автоматически:  
данные типа *int*, не зависимо от разрядности числа,занимают в ОП 4 байта;  
данные типа *double* занимают в ОП 8 байт;  
данные типа *DBDate* занимают в ОП 12 байт (три числа типа *int*: день, месяц, год).  
Чтобы обеспечить выполнение этого условия для строк, мы ввели в заголовки столбцов (структура *ColumnDesc*) поле *length*, в котором хранится максимальная длина строк в данном столбце, и все строки столбца должны иметь длину *length*.

Чтобы упростить создание бинарных файлов с таблицами БД, будем создавать их из текстовых таблиц, хранящихся в объектах класса DBTableTxt. Для этого в класс DBTableTxt добавлены две функции:   
- WriteTableBin(DBTableTxt & tab, string fileName) для записи данных из DBTableTxt в бинарный файл;   
- ReadTableBin(DBTableTxt & tab, string fileName) для чтения таблицы из бинарного файла в DBTableTxt.

С помощью функции WriteTableBin() мы будем создавать бинарные файлы с таблицами БД из текстовых файлов, с помощью функции ReadTableBin() - читать их в объект класса DBTableTxt, а с помощью DBTableTxt:: PrintTable() - контролировать содержание таблиц.

Для работы с файлами мы будем использовать потоковый ввод-вывод. Для прямого доступа к данным в файловых потоках имеются функции установки позиции чтения –записи байтов,   
и функции, непосредственно выполняющие обмен байтами.

* ifstream &seekg(long p);  
  Устанавливает указатель чтения со смещением р от начала файла.
* ifstream &seekg(long p, seek\_dir point);  
  Указывается начальная точка перемещения.  
  enum seek\_dir { beg, curr, end };  
  Положительное значение р перемещает указатель вперед (к концу файла), отрицательное значение р - назад (к началу файла).
* long tellg(void);  
  Возвращает текущее положение указателя чтения.
* ofstream &seekp(long p);  
  Перемещает указатель записи на позицию р от начала файла.
* ofstream &seekp(long p, seek\_dir point);  
  Указывается точка отсчета.
* long tellp(void);   
  Возвращает текущее положение указателя записи.

Для чтения данных из файла используется функция   
 ifstream &read(char \*buffer, int size);

Она просто копирует size байтов из файла, начиная с позиции, определяемой указателем чтения, в буфер buffer в ОП, не выполняя никаких преобразований. Не поддерживает разделителей, и считанные в буфер символы не завершаются нулевым символом.  
Для записи данных в файл используется функция   
 ofstream &write(const char \*buffer, int size);  
Данная функция осуществляет передачу необработанных данных (бинарных или текстовых) в файл. Она записывает в файл содержимое буфера. Символы копируются до тех пор, пока не возникнет ошибка или не будет скопировано *size* символов. Данные не форматируются. Обработка нулевых символов ничем не отличается от обработки других.

На самом деле функции *read()* и *write()* обмениваются данными с буфером потока, который, в свою очередь, связан с файлом. Непосредственный обмен данными между буфером и файлом выполняется с помощью функций операционной системы.   
Перепись данных из буфера в файл полностью завершается при закрытии файла. Закрытие файла происходит при уничтожении потокового объекта или при вызове метода *close().*  
Посмотреть в Редакторе Visual Studio измененное содержание файла можно только после его закрытия в программе. Две программы одновременно не могут работать с файлом.

## 2.1 Работа с бинарными файлами

Шаг\_0. Создайте решение ПКШ\_ЛР5\_N с пустым консольным приложением по имени Lab5\_1, создайте в нём файл с именем   
 testLab51\_N*х*.cpp, где N – индекс бригады, *х* – индекс студента в бригаде.   
Подключите к Lab5\_1 библиотеку dbmsLib\_v2 и тестовую БД.  
Путь к таблицам БД определяется в программе по имени БД и имени таблицы на основании соглашений по именованию и расположению папок и файлов.

Добавьте в файл testLab51\_N*х*.cpp функцию main(), которая должна выполнять действия, указанные в меню «СОЗДАНИЕ БИНАРНОЙ БД»:

int menu(){

cout<<"============ СОЗДАНИЕ БИНАРНОЙ БД ============\n";

cout<<"\t1 - Чтение БД из файла\n";

cout<<"\t2 - Печать БД\n";

cout<<"\t3 - Запись БД в файл\n";

cout<<"\t4 - Работа с бинарными файлами\n";

cout<<"\t5 - Создание бинарной версии текстовой БД\n";

cout<<"\t8 - Тестирование\n";

cout<<"\t10 - 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;

}  
Скомпилируйте и запустите проект.   
Проверьте правильность выполнения п. Шаг-0, выполнив пункты 1, 2 и 3 меню.

Функция testing(), которая вызывается из п. 8 меню «Тестирование», предназначается для различных экспериментов.

2.1.1. Работа с бинарными файлами.   
 Цель этого задания – научиться создавать бинарные файлы, выполняя эксперименты с файлом date.bin.   
Все указанные ниже действия задания должны выполняться в функции BinaryFiles(), которая будет вызываться из п. 4 меню «Работа с бинарными файлами».

В функции BinaryFiles() создайте и инициализируйте массив Date dates1[] из трех структур типа Date с тестовыми данными и «рабочий» массив Date\* dates2=new Date[3].

struct Date{  
 int day;  
 int month;  
 int year;  
 };

Создайте поток и свяжите его с файлом date.bin.   
При открытии потока установите флаги так, как указано в операторе :  
fstream fio("date.bin",ios::out|ios::in|ios::binary);  
Для реализации отдельных действий функция должна использовать собственное меню.

int menu1(){

cout<<"=============== БИНАРНЫЕ ФАЙЛЫ ===============\n";

cout<<"\t1 - Запись массива структур в бинарный файл\n";

cout<<"\t2 - Чтение бинарного файла в массив структур\n";

cout<<"\t3 - Ручное редактирование бинарного файла\n";

cout<<"\t4 - Прямой доступ к данным\n";

//cout<<"\t4 - Работа с бинарными файлами\n";

cout<<"\t10 - 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;

}

2.1.1 Запись данных из массива структур в бинарный файл.  
 Напишите функцию, которая записывает в бинарный файл date.bin массив   
Date dates1[].  
Проверьте результат, открыв файл в редакторе Visual Studio.   
2.1.2. Чтение бинарного файла в массив структур.  
Напишите функцию, которая читает данные из бинарного файла в массив Date dates2[].  
Для проверки результата распечатайте массив.  
2.1.3 Ручное редактирование бинарного файла.  
Порядок действий для выполнения этого пункта задания.  
- Закройте поток.  
- Выполните оператор system("pause");  
- Откройте файл date.bin в окне редактора Visual Studio и измените в редакторе значения полей второго элемента массива.  
- Откройте поток.   
- Запишите измененный файл в массив структур.

2.1.4 Прямой доступ к данным.   
 Напишите функцию, которая, используя прямой доступ к данным в файле, меняет местами значение месяца в 1-ом и 2-ом элементах массива.   
Запишите измененный файл в массив структур.

## 2.2 Создание и тестирование бинарной БД LibraryBin

Шаг\_0. Вручную создать папку с именем бинарной версии текстовой БД и скопировать в неё файл DBTables.txt из текстовой БД

Шаг\_1. Разработать функцию CreateBinaryDB(string dbName), реализующую п. 5 «Создание бинарной версии текстовой БД» главного меню.  
Для создания таблиц БД LibraryBin использовать функции WriteTableBin() и ReadTableBin() класса DBTableTxt и БД LibraryTxt.  
Шаг\_2. Распечатать и сравнить таблицы БД LibraryBin и БД LibraryTxt.

## 2.3 Разработка «базового» интерфейса класса DBTableBin.

Задание является продолжением ПКШ\_ЛР4.  
 В ПКШ\_ЛР5 разрабатывается базовый интерфейс производного класса DBTableBin и этот класс добавляется в иерархическую систему классов макета учебной СУБД, которая уже имеет в своём составе классы DBTable, DBDate, DBTableTxt и DBTableSet.

- Шаг 0. Установка «начального состояния» проекта Lab52\_N.   
Добавьте в решение ПКШ\_ЛР5\_ N пустое консольное приложение по имени Lab52\_N.  
Создайте в нём пустой файл testLab52\_Nx.cpp. Скопируйте в текущую папку проекта из проекта Lab4\_2 файлы testLab42\_Nx.cpp, Lab42\_N.h, DBTableN.cpp, DBTableTxtN, DBTableBinN.cpp (заглушки) , DBTableSetN.cpp и DBDateN.cpp.   
Переименуйте файл testLab42\_Nx.cpp в testLab52\_Nx.cpp и файл Lab42\_N.h в Lab52\_N.h.  
Библиотеку dbmsLib\_v2 подключать не надо.   
Выполните и протестируйте проект. Проверьте результаты выполнения для LibraryTxt и LibraryBin. Для LibraryBin вместо таблиц должны распечататься тексты заглушек.

### 2.3.1 Переопределение виртуальной функции PrintTableN(..).

Укрупнённый алгоритм и результат выполнения функции PrintTable() в классах DBTableBin и DBTableTxt полностью совпадают. Отличия в деталях реализации связаны с тем, что заголовок и данные в объектах типа DBTableBin хранятся в *динамических массивах* ColumnDesc\* header и char\*\* data,   
а в объектах типа DBTableTxt – *в контейнерах*   
 Header columnHeaders и vector<Row> data.  
Для устранения отличий вместо непосредственного обращения к данным типа Header и Row будем использовать переопределённые в каждом классе виртуальные функции:  
Header GetHeader();  
Row GetRow();  
int GetSize();  
string valueToString(Row& row,string columnName);  
Чтобы дважды (в классах DBTableBin и DBTableTxt) не повторять практически один и тот же код, метод  
 void CreateTableMaket(Strip\* &strips,int &nStrip,int screenWidth)  
заменим глобальной функцией   
 void TableMaket(Header hdr, Strip\* &strips,int &nStrip,int screenWidth)   
и в обоих классах будем вызывать её из метода PrintTable(..), используя в каждом классе для получения заголовка таблицы переопределённую виртуальную функцию GetHeader().  
С учётом выше сказанного, разработку метода PrintTable(..) будем выполнять следующим образом. Сначала сделаем указанные преобразования в уже работающей функции PrintTable1(..) в классе DBTableTxt. После того, как она будет отлажена, скопируем её в файл DBTableBin.cpp и завершим отладку.   
Причина возникающих в процессе отладки ошибок, заключается в том, что где-то вы непосредственно обращались к свойствам класса, вместо того, чтобы использовать переопределённые виртуальные функции интерфейса.

Шаг 1. Создание глобальной функции  
 void TableMaket(Header hdr, Strip\* &strips,int &nStrip,int screenWidth).  
Для реализации этой функции требуется только заголовок таблицы. Если его включить в список параметров, то функцию можно сделать глобальной и исключить её из методов классов. Новое имя используется для исключения путаницы в именах при программировании.   
- 1.1. В функции void CreateTableMaket(..), которая определена в файле DBTableTxt1.cpp, сделаем следующие изменения:  
 - добавим параметр Header hdr ;   
 - переименуем в TableMaket(..);  
 - переведём из методов класса в пространство dbmsLib1.   
 - внесём связанные с этим изменения в функции PrintTable1(..), CreateTableMaket(..) в классе DBTableTxt1, в описание класса DBTableBin1 и в заголовочный файл Lab52\_1.h.

Например, в PrintTable1(..):  
 - заменим оператор вызова функции CreateTableMaket(strips, nStrip, screenWidth)  
 на TableMaket(GetHeader(), strips, nStrip, screenWidth);  
 - в тексте функции заменим columnHeaders на GetHeader().

Выполните и протестируйте проект. Проверьте результаты выполнения для LibraryTxt и LibraryBin. Для LibraryBin вместо таблиц должны распечататься тексты заглушек.

Шаг 2. Внесение изменений в функцию PrintTable1(..) в классе DBTableTxt.  
 Цель изменений – получить функцию PrintTable1(..), которая может использоваться для печати таблиц обоих классов.

Изменения потребуются только для печати строк данных. Они связаны с тем, что все строки таблицы в классе DBTableBin последовательно загружаются в единственную строку row=GetRow(i).

После выполнения присваивания запомненное значение итератора для предыдущей строки становится недействительным, так оно связано с другой областью памяти. Поэтому вместо запоминания значения итератора, как это делалось в классе DBTableTxt1, начальные значения итератора будут вычисляться.

2.1. Установка итераторов на начало строки row в r-полосе (r>0).  
Вместо использования итератора rowIterCont перед печатью i-ой строки в r-полосе таблицы выполним следующий код, который установит итератор на начальный столбец строки в полосе:

Row row=GetRow(i);

rowIter=row.begin();

if(r>0)  
 for( int k=0; k< strips[r-1].nField ; k++,rowIter++);

2.2. Добавим в классы DBTableTxt и DBTableBin переопределенные виртуальные функции int GetRow(int ind) и Header GetHeader(). Для DBTableTxt их код очевиден, а для DBTableBin код приведён ниже.

Row DBTableBin1:: GetRow(int ind){

void\* val=0;

Row row;

for(int j=0;j<nColumn;j++){

char\* posPtr=data[ind]+FieldPosition(header[j].colName);

switch (header[j].colType){

case Int32: val=(int\*)posPtr; break;

case Double: val=(double\*)posPtr; break;

case String: val=new (string)(posPtr);

//без new не получается, т.к. string добавляет 1 байт перед char\*.

//Для изменений нужно будет использовать связку

//GetRow(int ind) - SetRow(Row row,int ind);

break;

case Date: val=(DBDate1\*)posPtr; break;

default: cout<<"Недопустимый тип данных в столбце\n"<<

header[j].colName<<endl; break;

}

row[header[j].colName]=val; }

return row;

}

//---------------------------------------------------

Header DBTableBin1::GetHeader(){

Header hdr;

for(int i=0; i<nColumn; i++)

hdr[header[i].colName]= (header[i]);

return hdr;

}

//-----------------метод int FieldPosition(string colName)---------------------

int DBTableBin1:: FieldPosition(string colName){

int pos=0;

int j=0;

for (j = 0; j <nColumn; j++){

if(header[j].colName==colName)

break;

pos+=GetLength(header[j]);

}

if(j==nColumn && colName!="NoPrimaryKey"){

cout<<"Нет поля "<<colName<<" в таблице "<<tableName<<endl;

system("pause");

return -1;

}

return pos;

}

### 2.3.2 Переопределение виртуальной функции WriteDBTableN(..). При записи данных из объекта типа DBTableBin в двоичный файла в преобразование типов не требуется.

Шаг 1. Открыть файловый поток на вывод данных в бинарный файл.  
Записать в файл имя таблицы и имя столбца с первичным ключом.  
Проверить результат, открыв файл в редакторе MS Visual Studio.  
Шаг 2. Добавить код для записи в файл заголовка таблицы и строк данных.  
Длина строк вычисляется методом RowLength().

### 2.3.3 Переопределение виртуальной функции ReadDBTableN(..).

### При чтении из двоичного файла в объект DBTableBin преобразование типов не требуется.

Шаг 1. Открыть файловый поток на ввод данных из бинарного файла.  
Считать из файла имя таблицы и имя столбца с первичным ключом.  
Проверить результат, распечатав таблицу.  
Шаг 2. Считывание заголовка в ColumnDesc\* header.   
Шаг 3. Считывание данных.  
Память выделяется под maxRows, а читается nRows строк.   
maxRows=nRows+DELTA;  
DELTA - запас на случай расширения таблицы. DELTA определяется в заголовочном файле.