# ПКШ2018\_ЛР2.Создание бинарных файлов с таблицами БД.

# Цель работы

Создание бинарных файлов с таблицами БД.

# Порядок выполнения работы

## Введение

Библиотека классов для макета СУБД будет включать 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 придется сделать при разработке тестовых примеров для отладки 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 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 разработаем две дружественные классу DBTableTxt функции:   
- WriteTableBin1(DBTableTxt & tab, string fileName) для записи данных из DBTableTxt в бинарный файл;   
- ReadTableBin1(DBTableTxt & tab, string fileName) для чтения таблицы из бинарного файла в DBTableTxt.

С помощью функции WriteTableBin1() мы будем создавать бинарные файлы с таблицами БД из текстовых файлов, с помощью функции ReadTableBin1() мы будем читать их в объект класса DBTableTxt, а с помощью DBTableTxt:: PrintTable() мы будем контролировать разработку этих функций.

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().* Поэтому посмотреть измененное содержание файла можно только после его закрытия.

## 2.1 Создание и работа с бинарными файлами

Для выполнения этого пункта задания не нужно создавать отдельный проект. Выполните его в функции, которая вызывается из пункта меню «Тестирование» проекта, в котором вы разрабатываете интерфейс макета СУБД, предварительно очистив (или закоментировав) старый текст функции. Эта функция предназначена для различных экспериментов, необходимость проведения которых возникает в процессе разработки макета СУБД.  
Цель этого вспомогательного задания – научится создавать бинарные файлы, выполняя эксперименты с файлом date.bin.

2.1.1 Создайте бинарный файл date.bin и запишите в него массив из трех структур типа Date.  
 struct Date{  
 int day;  
 int month;  
 int year;  
 };  
2.1.2 Откройте файл date.bin в окне редактора Visual Studio и измените в редакторе значения полей второго элемента массива. Запишите измененный файл в массив структур.

2.1.3 Напишите код, который, 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.

При открытии потока установите флаги так, как указано в операторе fio.open():  
fstream fio;   
fio.open("date.bin",ios::out|ios::in|ios::binary|ios::ate|ios::trunc);

2.2 Разработка функций WriteTableBin1() и ReadTableBin1()

Функции разрабатываются как дружественные функции класса DBTableTxt.

Разработку функций удобно вести параллельно и по шагам, используя для контроля функцию PrintTable().

Шаг 1.   
 - Прочитать в *DBTableTxt* тестовую таблицу БД.   
 - С помощью функции WriteTableBin1() записать в бинарный файл данные первой строки таблицы из *DBTableTxt* .   
 - С помощью функции ReadTableBin1() считать эту строку в *DBTableTxt*.  
 - Используя функцию *PrintTable(),* распечатать *DBTableTxt*.  
Шаг 2.  
 Повторить действия шага 1 для заголовка таблицы.

Шаг 3.  
 Повторить действия шага 1 для данных таблицы.

Ниже приведены фрагменты программ, которые должны помочь вам в разработке функций WriteTableBin1() и ReadTableBin1().

//------Запись данных таблицы в бинарный файл tabName+".bin"--------------

void DBTableTxt::WriteTableBin(string fileName){

ofstream fout;

fout.open((fileName).c\_str(),ios::binary|ios::out);

if (!fout.is\_open())

{

cout<<"Ошибка открытия файла "<<fileName<< endl;

system("pause");

return;

}

Header::iterator iter;

Row ::iterator dataIter;

//Запись в бинарный файл первой строки .txt-файла:

//имени таблицы и имени столбца primaryKey

//(длины имён таблиц и столбцов фиксированы и равны 24 байтам),

char buf[80];

strcpy\_s(buf,80,tableName.c\_str());

int len=LENGTH;//tableName.size();//+1 - символ конца строки '\0'

fout.write((char\*)&len,4);

fout.write(buf,len);

strcpy\_s(buf,80,primaryKey.c\_str());

// len=primaryKey.size();

fout.write((char\*)&len,4);

fout.write(buf,len);

//Запись в бинарный файл заголовка таблицы (вторая строка.txt-файла)

int size = columnHeaders.size();

fout.write((char\*)&size,4);

size=sizeof(ColumnDesc);

for(iter=columnHeaders.begin();iter!=columnHeaders.end();iter++)

fout.write((char\*)&(iter->second),size);

//Запись строк таблицы.

int nRows=(int)data.size();

fout.write((char\*)&nRows,4);

for (int i = 0; i < nRows; i++) {

dataIter = data[i].begin();

//============================================================================

//-------Чтение данных из бинарного файла tabName+".bin" в таблицу DBTable-------

void DBTableTxt::ReadTableBin(string tabName)//!!! tabName - путь из текущей папки

{

ifstream fin;

fin.open((tabName).c\_str(),ios::binary|ios::in);

if (!fin.is\_open())

{

cout<<"Ошибка открытия файла\n"<< endl;

system("pause");

return;

}

//Прочитать имя таблицы и имя столбца с primaryKey из первой строки файла

int len;

char buf[80];

fin.read((char\*)&len,4);

fin.read((char\*)buf,len);

if(len>79){

cout<<"Ошибка: длина имени таблицы "<<tabName<<endl;

return;

}

buf[len]='\0';

tableName=buf;

fin.read((char\*)&len,4);

fin.read((char\*)buf,len);

if(len>79){

cout<<"Ошибка: длина имени primaryKey таблицы "<<tabName <<endl;

return;

}

buf[len]='\0';

primaryKey=buf;

//Прочитать из файла заголовок таблицы и записать его в table.columnHeaders

int size=0; //число столбцов в заголовке таблицы

fin.read((char\*)&size,4);

===============================================================================

## 2.3 Создание и тестирование БД LibraryBin

Используя разработанные в п.2.2 функции WriteTableBin1() и ReadTableBin1() и БД LibraryTxt, создать и проверить БД LibraryBin.