# ПКШ2021\_ЛР5. Создание бинарных файлов с таблицами БД.

# Цель работы

Освоение методов работы с бинарными файлами на примере создания бинарных файлов с таблицами для учебной СУБД.

## Введение

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 258, 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.

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.

Работа состоит из двух частей.   
В первой части вы будете учиться обрабатывать бинарные файлы, выполняя эксперименты с файлом date.bin, в котором хранится массив из произвольного числа элементов типа Date.

Во второй части вы будете создавать бинарные файлы с таблицами БД.

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.

# Порядок выполнения работы

## 2.1 Работа с бинарными файлами

Цель этого задания – научиться создавать бинарные файлы, выполняя эксперименты с файлом date.bin.

Создайте решение ПКШ\_ЛР5\_N с пустым консольным приложением по имени Lab51\_N, создайте в нём файл с именем   
 testLab51\_N\_*х*.cpp, где N – индекс бригады, *х* – индекс студента в бригаде.

Функция main() этого файла должна выполнять действия, указанные в меню «БИНАРНЫЕ ФАЙЛЫ»

int menu (){

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<<"\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;

}

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

В функции main() создайте поток fstream fio, создайте и инициализируйте массив dates1[] из структур типа Date с тестовыми данными и «рабочий» массив Date\* dates2.  
Открытие потока, установку режимов работы с файлом date.bin и закрытие потока выполняйте в теле функций, вызываемых из меню для работы с бинарными файлами.

Фрагмент функции main():

int main(){

system("chcp 1251 >nul");

fstream fio;

//тестовый пример

Date dates[]={{4,4,2020},{5,5,2020},{6,6,2020},{20,12,2020},{20,11,2020}};

int size1=sizeof(dates)/sizeof(Date);

Date\* dates1=new Date[size1];

for(int i=0;i<size1;i++){

dates1[i]=dates[i];

cout<<dates1[i].day<<'.'<<dates1[i].month<<'.'<<dates1[i].year<<endl;

}

Date\* dates2;//указатель на динамический массив для хранения считываемых данных

int size2;

while(true){…

…

2.1.1 Запись данных из массива структур в бинарный файл.  
 Напишите функцию WriteDate(fstream& fio, ...), которая создаёт бинарный файл date.bin, если его нет, и записывает в date.bin массив dates1[]. Размерность массива может быть произвольной. Для того, чтобы этот массив можно было бы правильно прочитать, непосредственно перед элементами массива запишите в файл его размер.  
Откройте поток и свяжите его с файлом date.bin  
При открытии потока установите флаги так, как указано в операторе :  
fio.open("date.bin", ios::out |ios::binary);  
Перед выходом из функции закройте поток. Это необходимо для переписи содержимого буфера потока в файл.  
  
2.1.2. Чтение бинарного файла в массив структур.  
Напишите функцию ReadDate(fstream& fio, ...), которая читает данные из бинарного файла в динамический массив dates2.

Для проверки результата распечатайте массив после выхода из функции.  
  
2.1.3 Прямой доступ к данным.   
 Напишите функцию DirectAcces(fstream& fio, ...), которая, используя прямой доступ к данным в файле, меняет местами значение месяца в 1-ом и 2-ом элементах массива и читает измененный файл в динамический массив dates2.   
Для проверки результата распечатайте массив после выхода из функции.

2.1.4 Ручное редактирование бинарного файла.  
Так как файл не может одновременно использоваться двумя программами, то для выполнения этого требования нужно предусмотреть определённый порядок действий.  
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.  
- Сохраните измененный файл.  
Для контроля выполнения операций используйте меню.  
!!! Функции меню после выполнения п.1 должны выполняться многократно в произвольном порядке. Для выполнения этого требования все функции выполняются с одним и тем же потоком, который передаётся им по ссылке в качестве параметра.

## 2.2 Создание и тестирование бинарной БД LibraryBin

Библиотека классов учебной СУБД включает два класса для работы с таблицами БД – *DBTableTxt* и *DBTableBin*.   
Чтобы упростить создание бинарных файлов с таблицами БД, будем сначала создавать их текстовые версии, хранящихся в объектах класса DBTableTxt1, а затем переписывать их из объектов DBTableTxt1 в бинарные файлы. Для этого в ЛР разрабатываются две функции, которые являются друзьями класса DBTableTxt1:   
- WriteTableBin1(DBTableTxt1 & tab, string fileName) для записи данных из DBTableTxt1 в бинарный файл;   
- ReadTableBin1(DBTableTxt1 & tab, string fileName) для чтения таблицы из бинарного файла в объекты типа DBTableTxt1.

Определим структуру бинарных файлов таким образом, чтобы можно было выполнять прямой доступ к полям таблиц БД.  
Для реализации прямого доступа к данным, хранящимся в таблице, недостаточно записать таблицу в бинарный файл. Необходимо, чтобы данные в одном и том же столбце таблицы были бы одинакового размера. Только в этом случае можно будет вычислять позицию отдельных элементов таблицы в файле.

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);  
данные типа *string* в разных столбцах таблицы могут отличаться по длине.  
Для определения типа и размера данных в столбцах таблиц БД в заголовках столбцов учебной СУБД используется структура *ColumnDesc*, поле *length* которой хранит длину строки данных в данном столбце.

- Шаг 0. Добавьте в решение ПКШ\_ЛР5\_ N консольное приложение по имени Lab52\_N,  
создайте в нём файл с именем   
 testLab52\_N\_*х*.cpp, где N – индекс бригады, *х* – индекс студента в бригаде.   
Вставьте в него содержимое файла testLab31\_1 из ПКШ\_ЛР3, добавив в меню п.4:   
«Создание бинарной версии текстовой БД».  
Функция main() этого файла должна выполнять действия, указанные в меню «СОЗДАНИЕ БИНАРНОЙ БД»

int menu (){

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

cout<<"\t1 - Чтение таблицы из текстовой БД\n";

cout<<"\t2 - Печать таблицы\n";

cout<<"\t3 - Запись таблицы в текстовую БД\n";

cout<<"\t4 - 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;

}

Скопируйте в текущую папку проекта из проекта Lab3\_1 файлы DBTableTxt1.cpp, DBDate1.cpp и DBTable31.h. Выполните и протестируйте проект. На этом шаге он практически повторяет проект Lab3\_1.

### 2.2.1 Разработка функции CreateBinaryDB(string dbName).

Функция CreateBinaryDB() предназначается для создания бинарной версии текстовой БД и вызывается из п.4 меню «СОЗДАНИЕ БИНАРНОЙ БД».

Предварительно вручную создаём папку с именем бинарной версии текстовой БД,   
копируем в неё из текстовой БД файл DBTables.txt с именами таблиц БД, а затем в цикле создаём бинарные файлы для всех таблиц БД.

Алгоритм создания бинарных файлов с таблицами учебной СУБД.   
1) чтение текстовой таблицы в объект *tab1*типа *DBTableTxt*.  
2) запись данных из объекта *tab1* в бинарный файл с помощью функции WriteTableBin().  
3) контроль бинарной таблицы путём считывания её с помощью функции ReadTableBin() в объект *tab2 DBTableTxt* и распечатки функцией *PrintTable().*

Разработку функций WriteTableBin1() и ReadTableBin1() удобно вести параллельно и по шагам.

Шаг 1.   
 - Прочитать в объект типа *DBTableTxt1* тестовую таблицу БД.   
 - С помощью функции WriteTableBin1() записать в бинарный файл данные ***первой строки таблицы*** из *tab1*.   
 - С помощью функции ReadTableBin1() считать ***эту строку*** в *tab2*.  
 - Используя функцию *PrintTable1(),* распечатать *tab2*.  
Шаг 2.  
 Повторить действия шага 1, добавив в разрабатываемые функции чтение и запись ***заголовка таблицы***.

Шаг 3.  
 Повторить действия шага 1, добавив в разрабатываемые функции чтение и запись ***данных таблицы***.

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

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

void dbmsLib1:: WriteTableBin1(DBTableTxt1& tab,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

//(длины имён таблиц и столбцов фиксированы и равны 24 байтам, включая \0)

char buf[80];

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

int len=LENGTH;

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

fout.write(buf,len);

//Запись имени столбца primaryKey

. . .

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

int size = tab.columnHeaders.size();

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

size=sizeof(ColumnDesc);

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

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

//Запись строк таблицы.

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

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

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

dataIter = tab.data[i].begin();

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

{//iter->second указывает на ColumnDesc, dataIter->second указывает на void\*.

switch((iter->second).colType) {

case Int32: fout.write((char\*)((dataIter->second)),sizeof(int));  
 break;

case Double: . . .

case Date: . . .

case String: strcpy\_s(buf,80,(\*(string\*)(dataIter->second)).c\_str());

fout.write(buf+1,iter->second.length);

//buf+1 - для того, чтобы length не обрубал конец строки '\0'

break;

default: cout<<"Недопустимый тип данных в БД\n";

return;

}

}

}

}

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

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

void dbmsLib1::ReadTableBin1(DBTableTxt1& tab,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';

tab.tableName=buf;

//Прочитать имя столбца с primaryKey

. . .

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

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

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

len=sizeof(ColumnDesc);

ColumnDesc colDesc;

tab.columnHeaders.clear();

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

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

tab.columnHeaders[colDesc.colName]=colDesc;

}

}

//Чтение строк таблицы

Header::iterator hdrIter;

data.clear();

int nRows;

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

len=sizeof(colDesc.colName);//длина имени столбца (=24 байта)

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

{

Row row;//буфер для формирования строки таблицы

pair<string,void\*> pr;

for(hdrIter=columnHeaders.begin();hdrIter!=columnHeaders.end();hdrIter++){

pr.first=hdrIter->first;

switch((hdrIter->second).colType) {//чтение данных

case Int32: pr.second=new int;

fin.read((char\*)pr.second,sizeof(int));

row.insert(pr); break;

case Double:

. . .

case Date:

. . .

case String:if(hdrIter->second.length>79){

cout<<"Ошибка: длина поля "<<hdrIter->second.colName<<

" таблицы "<<tabName<<endl;

return;

}

fin.read(buf,hdrIter->second.length);

buf[hdrIter->second.length]='\0';

pr.second=new string(ignoreBlanc(buf));

row.insert(pr);

//cout<<\*(string\*)row[pr.first]<<endl;

break;

default: cout<<"Недопустимый тип данных в БД\n";

return;

}

}

data.push\_back(row);

}

fin.close();

}