# Лабораторная работа ПКШ2021\_ЛР3

## Разработка класса с использованием контейнеров STL *map* и *vector*

## Цель работы

Целью работы является приобретение студентами навыков групповой разработки проектов и использования контейнеров *map* и *vector* из Стандартной Библиотеки Шаблонов С++ на примере разработки классов для макета учебной СУБД.

## 2. Содержание работы

Для выполнения этой ЛР студенты объединяются в бригады по три человека.  
Каждая бригада должна разработать свой класс c именем DBTableTxtN с «базовым» интерфейсом, позволяющим читать, распечатывать и записывать таблицы базы данных (БД) в текстовый файл. Символ ‘N’ в имени класса – это номер бригады. Он будет использоваться преподавателем при оценке работы студентов.   
При выполнении ЛР студенты используют статическую библиотеку классов для макета СУБД dbmsLib\_v1\_1 и таблицы БД с тестовыми данными, созданные в ПКШ2021\_ЛР1.

Работа состоит из двух частей.   
В первой части члены бригады согласовывают «базовый» интерфейса класса, фиксируют его в заголовочном файле dbmsLibN.h, распределяют между собой функции для разработки и параллельно разрабатывают их. Файл dbmsLibN.h уже содержит описание класса DBDateN из ЛР ПКШ2021\_ЛР2. Базовый интерфейс должен обеспечивать чтение, запись и распечатку таблиц БД, а при его реализации могут потребоваться некоторые служебные функции, которые также нужно разрабатывать.   
Во второй части создаётся класс DBTableTxtN, в котором объединяются разработанные членами бригады функции.   
*!!! Для упрощения контроля выполнения ЛР соблюдайте приведённые ниже соглашения по именованию проектов и файлов.*  
ЛР разрабатывается в решении с именем ПКШ2021\_ЛР3\_N, которое включает два проекта.   
Первая часть разрабатывается в проекте с именем Lab3\_N\_m, где m - индекс члена бригады (m=1, 2 или 3). Разработка интерфейса выполняется путем перегрузки дружественных для библиотечного класса DBTableTxt функций:   
 1) friend void ReadDBTable1(DBTableTxt1& tab, string tabName),  
 2) friend void PrintTable1(DBTableTxt1& tab, string tabName),  
 3) friend void WriteDBTable1(DBTableTxt1& tab, string tabName).  
(Имена функций указаны для бригады номер 1)

Вторая часть разрабатывается в проекте с именем Lab3\_N, в котором должен быть реализован «базовый» интерфейс класса DBTableTxtN и выполнено его тестирование .  
 Во второй части при разработке функций класса DBTableTxtN библиотека учебной СУБД dbmsLib\_v1\_1 не используется. Подключаются файлы dbmsLibN.h, DBDateN.cpp из ПКШ2021\_ЛР2, а в файл DBtableTxtN.cpp помещаются все разработанные бригадой функции.  
Планируемое время выполнения работы- 8 часов занятий в компьютерном зале (2 часа на создание проекта + 6 часов на разработку функций + 20 часов самостоятельной работы студента (СРС) с учетом времени на СРС в проектно-технологической практике (ПТП ).

## 3. Порядок выполнения работы

Описание выполнения действий во всех лабораторных работах приводится применительно к MS Visual Studio 2012.

*!!! Обязательно сохраните копии таблиц тестовой БД.*

Создайте решение ПКШ\_ЛР3\_N с пустым консольным приложением по имени Lab3\_1, создайте в нём файл с именем   
 testLab3*х*.cpp, где *х* - порядковый номер разрабатываемой функции.   
Скопируйте в него содержание файла testDB из ПКШ\_ЛР1.   
 Подключите к Lab3 библиотеку dbmsLib\_v1\_1 и тестовую БД (Как это делать, смотри ПКШ\_ЛР1). Скомпилируйте и запустите проект.  
 Проверьте правильность выполненных операций путем последовательного вызова из меню библиотечных функций *ReadDBTable(), PrintTable(), WriteDBTable()* и ещё раз *ReadDBTable(), PrintTable()*.

Для выполнения ЛР вы должны адаптировать функции *Menu()* и *main(),*скопированные из testDB.cpp к своему заданию.

### 3.1. Разработка функции для чтения таблиц БД из текстового файла

Разработайте собственную функцию ReadDBTable1() для чтения таблиц БД из файла в объект типа DBTableTxt. Разработку выполняйте путем перегрузки дружественной функции   
 friend void ReadDBTable1(DBTableTxt& tab, string fileName).   
Она должна быть включена в описание класса DBTableTxt в файле dbmsLib\_v1\_1.h.

Дружественные функции – это глобальные функции, определённые в пространстве dbmsLib . Но они имеют доступ к закрытым членам классов и позволяют обращаться при их перегрузке к элементам класса из области *private.* Использование дружественных функций упростит для вас процесс разработки «базового» интерфейса для класса DBTableTxtN.  
 fileName = ..\\ dbName + tabName + " .txt" - путь к файлу из текущей папки.  
БД хранится в папке "..\\"+dbName".

Если таблицы БД хранятся в текстовых файлах, то имя БД должно оканчиваться на Txt, например, LibraryTxt.

Каждая таблица хранится в отдельном файле с именем = <имя таблицы>+".txt"

Структура файла:

Первая строка файла содержит имя таблицы и имя столбца с первичным ключом.

Вторая строка файла содержит заголовок таблицы (обязательно в одной строке).

Все последующие строки до конца файла – это записи таблицы.

#### 3.1.1. Описание алгоритма

#### 3.1.2. Кодирование

- Шаг 0. Подготовительный.  
Разработку функций будем делать по шагам, контролируя выполнение каждого шага.   
Для реализации такой технологии добавьте в testLab3\_1.cpp функцию  
void testing(string dbName, string tabName){

cout <<"dbName="<<dbName<<" tabName="<<tabName<<endl;

string path="..\\"+dbName+"\\";

dbmsLib::DBTableTxt tab;

tab.ReadDBTable(path+tabName+".txt");

tab.PrintTable(80);//контроль связи с БД  
 //Вызов разрабатываемых функций (изменяемый код)  
 //Разрабатываемые функции будут использовать объект tab1.  
 dbmsLib::DBTableTxt tab1;

…

tab1.PrintTable(80);//контроль сделанных изменений   
}  
и включите в main() вызов этой функции из меню.

- Шаг 1. Создайте «заглушку» разрабатываемой функции   
 void ReadDBTable1(DBTableTxt& tab1, string fileName){

cout<< " ReadDBTable1 "<<fileName<<endl;//отладочная печать  
 }

Добавьте в функцию testing(string dbName, string fileName) код для вызова разрабатываемой функции ReadDBTable1(). Убедитесь в правильности сделанных действий и только после этого переходите к выполнению следующего шага.

- Шаг 2. Добавьте в функцию ReadDBTable1() код для чтения в tab1 первой строки файла с именем таблицы и именем ключевого столбца. Код должен включать открытие файла и проверку совпадения имени таблицы, извлечённого из filename библиотечной функцией GetTabNameFromPath1(fileName), с именем, считанным из файла. Проверьте сделанные изменения с помощью PrintTable().

- Шаг 3. Чтение заголовка таблицы в поле *Header columnHeaders*,   
где *Header –* имя типа *map<string, ColumnDesc>*;   
Определения типов *Header* и *ColumnDesc* приведены в заголовочном файле библиотеки *dbmstLib\_v1*.  
Записи данных в *map* выполняйте в цикле. Используйте итераторы в качестве параметра цикла и для доступа к данным. Пример использования итераторов для чтения заголовка таблицы приведён ниже на рисунке 1.  
Для обращения к данным в *map* можно также использовать и перегруженную в *map* операцию индексации [].

- Шаг 4. Чтение данных таблицы в поле *vector<Row> data*,  
 где *Row –* имя типа *map<string, void\*>.*   
Для преобразования читаемых из текстового файла данных из типа string   
в тип typeName, имя которого указано в заголовке столбца таблицы в CVS-файле,  
используйте функцию из библиотеки dbmsLib\_v1

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

Для проверки правильности выполненных действий вставьте в меню пункт для вызова функции ReadDBTable1() и выполняйте его для чтения таблицы вместо вызова библиотечной функции ReadDBTable().

После выполнения каждого шага выводите таблицу на экран, используя функцию PrintTable() из библиотеки dbmsLib.  
После выполнения 2-го шага у вас должно распечататься имя таблицы и пустая таблица.  
После выполнения 3-го шага в таблице должен появиться заголовок,  
 а после выполнения 4-го шага должна распечататься заполненная таблица.

Перед внесением изменений в разрабатываемую программу обязательно сохраняйте её текущую *работающую* версию. Чтобы не запутаться в версиях, сохраняйте их в папках с указанием даты создания и включайте номер версии в имя файла. Так как система классов для макета учебной СУБД будет разрабатываться в течение всего семестра, это поможет вам избежать путаницы в именах файлов и связанных с ней потерь времени.

Ниже приведены *фрагменты* метода *ReadDBTable ()*, выполняющего чтение таблиц БД из текстовых файлов формата CSV в объект типа DBTableTxt. Они должны помочь вам в выполнении задания, но требуют некоторого уточнения и дополнения.  
 В файлах в качестве разделителя используется символ ' | '.

-----------------------------------------------------------------------------------------------------------------------------

Чтение заголовка в columnHeaders:  
columnHeaders.clear();  
char \*token, \*next\_token;  
char line[200];  
fin.getline(line, 200); //чтение строки заголовка в line  
next\_token=line;  
 //цикл по словам (лексемам) в считанной строке с заголовком  
ColumnDesc colHdr;  
pair<string,ColumnDesc> parHdr;  
while( (token = strtok\_s( next\_token, delims, &next\_token)) != NULL ) {   
 strcpy\_s(colHdr.colName,token);  
 token = strtok\_s( next\_token, delims, &next\_token);

…

parHdr.first= colHdr.colName;  
 parHdr.second=colHdr;

columnHeaders.insert(parHdr);  
}

Чтение строк данных.  
К этому моменту заголовок уже прочитан.  
//читаем строки в line (до EOF) и записываем их в tab.data в цикле по столбцам заголовка  
data.clear();  
Header::iterator hdrIter;

pair<string,void\*> parRow;  
while (fin.getline(line, 200)) {

Row row=\*(new Row());//строка таблицы

next\_token=line;

//цикл по столбцам (словам) в строке с использованием итератора

//итератор позволяет определять тип данных в столбцах

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

{

//добавление поля в строку с преобразованием типа

token = strtok\_s( next\_token, delims, &next\_token);

parRow.first=hdrIter->first;

parRow.second=GetValue1(token,hdrIter->first,tab.columnHeaders);

row.insert(parRow);

}

tab.data.push\_back(row);//добавить строку данных в таблицу  
}

--------------------------------------------------------------------------------------------------------------------

Рис. 1. Фрагменты кода для чтения таблицы

### 3.2. Разработка функции для печати таблиц БД

Разрабатываемая функция PrintTable1(DBTableTxt&,screenWidth), где screenWidth - ширина экрана.  
 Функция должна печатать таблицу в том же виде, как это делает PrintTable(screenWidth) из библиотеки dbmsLib.

#### 3.2.1.Описание алгоритма

Если ширина таблицы больше ширины экрана, то таблица печатается в  
несколько полос, размещаемых друг под другом. В каждой полосе печатается последовательность столбцов, по ширине не превышающая ширину экрана.  
Столбцы, не поместившиеся на экране, печатаются в последующих полосах.

Количество полос, число и ширина полей в полосе будут определяться в отдельной функции (метод класса), которая вызывается из функции печати PrintTable(..) перед выводом на экран данных. Использование функции для определения макета распечатки таблицы делает алгоритм PrintTable(..) более читаемым и удобным для отладки.  
Число столбцов, их наименование, тип и размер данных в столбцах определяются из заголовка таблицы.

Печать данных выполняется с использованием 3-х вложенных циклов for:  
- внешний цикл по полосам,   
- средний – по строкам таблицы (в полосе выводятся все строки размещенных в ней столбцов),  
- внутренний – по столбцам в строке (столбцы в строке совпадают со столбцами в полосе).

Доступ к данным выполняется с помощью итераторов.

При перегрузке метода DBTableTxt:: PrintTable(..) используйте функции CreateTableMaket(…), TypeName(DBType) и valueToString(…) из библиотеки dbmsLib.

Функция string dbmsLib::valueToString(DBTableTxt& tab,Row& row,string columnName) возвращает значение переменной value, преобразованное из типа typeName, имя которого указано в заголовке columnName столбца таблицы, в тип string. В функции выполняется проверка наличия columnName в DBTableTxt.

const char\* TypeName(DBType) используется при печати 2-й строки заголовка. Она возвращает имя типа headerIter->second.colType.

В функции void dbmsLib::CreateTableMaket(DBTableTxt &tab,Strip \*&strips,int &nStrips,  
int screenWidth) strips и nStrips – выходные параметры.  
 strips - массив размеров столбцов strips[i].fieldWidth в полосах таблицы.   
Количество столбцов в полосах таблицы может быть различным.  
Для создания массива используется структура Strip - сущность более высокого, по сравнению со строкой, уровня, которая скрывает различия в числе столбцов в строках разных полос в макете таблицы. В результате из описаний полос, содержащих различное число столбцов, можно создать массив из полос.

struct Strip{//полоса распечатки таблицы

int nField;//размерность массива fieldWidth (число полей в полосе)

int\* fieldWidth;//указатель на массив из длин полей в полосе

};

Динамическая память, выделяемая под массив strips, должна освобождаться в функции PrintTable(…), которая владеет объектом strips.

Перед использованием библиотечных функций выполняйте их тестирование, чтобы понять, как они работают. Для тестирования функции CreateTableMaket() выполните в testing(..) задание 1 из раздела 5 «Контрольные вопросы и задания».

Так как функция печати таблиц БД относительно сложная для программирования, для неё имеет смысл разработать схему алгоритма. Она приведена на рисунке 2.



Рис.2 Алгоритм вывода на экран таблиц БД

#### 3.2.2. Кодирование

Выполните шаги «Шаг 0» и «Шаг 1» , описанные в пункте 3.1, применительно к функции PrintTable1(). Не забудьте изменить функцию testing().  
Последующие шаги будем выполнять руководствуясь разработанной схемой алгоритма.

- Шаг 2. Печать имени таблицы.   
Напечатайте после имени таблицы верхнюю границу таблицы с помощью оператора  
cout<<setfill('=')<<setw(screenWidth-1)<<'='<<setfill(' ')<<endl;

Если таблица печатается в несколько полос, то перед каждой из них печатайте «Продолжение таблицы».  
Вызовите из меню функцию testing() и убедитесь в правильности добавленного кода.

- Шаг 3. Печать заголовка таблицы.  
Заголовок печатается в две строки. Напечатайте нижнюю границу заголовка.  
Вызовите из меню функцию testing() и убедитесь в правильности добавленного кода.

- Шаг 4. Печать строк данных.  
При выводе данных используйте функцию valueToString().  
Напечатайте нижнюю границу таблицы.  
Вызовите из меню функцию testing() и убедитесь в правильности добавленного кода.

### 3.3. Разработка функции для записи таблиц БД в текстовый файл

Функция WriteDBTable1(DBTableTxt&,string fileName) должна записывать таблицу в текстовый файл в формате CSV с разделителем '|' в том же виде, как это делает WriteDBTable (string fileName) из библиотеки dbmsLib.   
fileName = ..\\ dbName + tabName + " .txt" - путь к файлу из текущей папки.

БД хранится в папке "..\\"+dbName+".

Имя БД оканчивается на Txt, если таблицы БД хранятся в текстовых файлах, например,  
LibraryTxt.

Каждая таблица хранится в отдельном файле с именем: <имя таблицы>+".txt"

Структура файла:

Первая строка файла содержит имя таблицы и имя столбца с первичным ключем.

Вторая строка файла - заголовок таблицы

Все последующие строки - записи таблицы (тип Row).

#### 3.3.1. Описание алгоритма

- Проверка соответствия имени таблицы в fileName и имени таблицы, указанному в tab. Это нужно сделать, чтобы избежать возможных нарушений принятых соглашений по именованию файлов и таблиц.  
 - Открытие файлового потока на вывод.   
 - Запись имени таблицы и имени столбца primaryKey (первая строка CSV - файла).  
- Запись заголовка таблицы (вторая строка файла).  
 - Запись в файл строк с данными.

#### 3.3.2. Кодирование

- Шаг 0. Подготовительный.  
Разработку функций будем делать по шагам, контролируя выполнение каждого шага.   
Для реализации такой технологии добавьте в testLab3\_1.cpp функцию  
void testing(string dbName, string tabName){

cout <<"dbName="<<dbName<<" tabName="<<tabName<<endl;

string path="..\\"+dbName+"\\";

dbmsLib::DBTableTxt tab;

tab.ReadDBTable(path+tabName+".txt");

tab.PrintTable(80);//контроль связи с БД  
 //Вызов разрабатываемой функции (изменяемый код)  
 …  
 tab.PrintTable(80);//контроль сделанных изменений   
}  
и включите в main() вызов testing() из меню.

!!! Отладка функций записи в файл усложняется тем, что при наличии ошибок они могут испортить тестовые данные. Поэтому при отладке следует использовать копию тестовых данных, которую нужно восстанавливать при каждом вызове testing().  
Для сохранения неизменности тестовых данных перед вызовом WriteDBTable1() вставим в файл testing() следующий код:

//---Контроль разработки функции WriteDBTable1(fileName)----

dbmsLib::DBTableTxt tab1;

tab1.ReadDBTable(path+"Students.txt");

tab1.SetTableName("Students1");

tab1.WriteDBTable(path+"Students1.txt");

tab1.ReadDBTable(path+"Students1.txt");

tab1.PrintTable(80);

dbmsLib::WriteDBTable1(path+"Students1.txt");

tab1.ReadDBTable(path+"Students1.txt");

tab1.PrintTable(80);

Вызовите из меню функцию testing() и убедитесь в правильности разработанного кода.  
В случае ошибки понять её причину может помочь просмотр записанного файла Students1.txt в редакторе файлов Visual Studio.

- Шаг 1. Создайте «заглушку» разрабатываемой функции   
 void WriteDBTable1(DBTableTxt& tab, string fileName){

cout<< " ReadDBTable1 "<<fileName<<endl;//отладочная печать  
 }  
Вызовите из меню функцию testing() и убедитесь в правильности разработанного кода.

Последующие шаги будем выполнять руководствуясь разработанным алгоритмом.

- Шаг 2. Извлечение имени таблицы из fileName (путь к файлу из текущей папки).  
Проверка соответствия имени таблицы в fileName и имени таблицы, указанному в tab. Это нужно сделать, чтобы избежать возможных нарушений принятых соглашений по именованию файлов и таблиц.  
- Шаг 3. Открытие файлового потока на вывод.  
 Запись имени таблицы и имени столбца primaryKey (первая строка CSV - файла).  
Вызовите из меню функцию testing() и убедитесь в правильности разработанного кода.

- Шаг 4. Запись заголовка таблицы (вторая строка файла).  
В соответствии с форматом файла, после последнего элемента строки с заголовком таблицы вместо '|' нужно вставить перевод строки.   
Элемент заголовка – это структура типа ColumnDesc.   
Поле colType (тип данных в столбце таблицы) имеет целочисленный тип DBType (перечисление), а в заголовке должна быть строка с именем типа (Int32, Double, String или DBDate).   
Вызовите из меню функцию testing() и убедитесь в правильности разработанного кода.  
- Шаг 5. Запись в файл строк с данными. Для записи используем два вложенных цикла:  
- внешний – по строкам таблицы;  
- внутренний – по столбцам.  
Для удобства чтения файла данные форматируются.  
В ОП компьютера данные хранятся в двоичном коде. При записи в файл их нужно преобразовать в текстовый формат. Для преобразования используйте функцию   
string valueToString(Row& row,string columnName) из библиотеки dbmsLib.  
Вызовите из меню функцию testing() и убедитесь в правильности разработанного кода.

Ниже приведены фрагменты метода WriteDBTableTxt(), которые могут помочь вам при разработке программы. Использование итераторов позволяет разработать универсальный алгоритм, не независящий от числа и наименования столбцов в таблицах.

//Запись в файл заголовка  
iter=columnHeaders.begin();  
iterWhile=iter;  
Чтобы не выводить в цикле последнюю строку, т.к. для итераторов в *map* не перегружены операции iter+1 и iter<, для вывода в файл используем два итератора:  
iterWhile - для управления циклом,  
 iter - для выбора столбца.   
 iter в цикле указывает на предыдущий столбец, так как он изменяется в конце цикла,  
после вывода заголовка столбца в fout.

while(++iterWhile!=columnHeaders.end())  
 {  
 fout<< iter->second.colName<<"|"<< TypeName(iter->second.colType)<< "|"<<iter->second.length<<"|";  
 iter++;  
 }  
fout<<iter->first<<"|"<< TypeName(iter->second.colType)<<"|"<<iter->second.length <<endl;

-------------------------------------------------------------------------------------------------

### 3.4 Разработка класса DBTableTxtN с «базовым» интерфейсом

Каждый член бригады сначала разрабатывает свой класс, который содержит только одну, разработанную им функцию «базового» интерфейса (ReadDBTable1(), PrintTable1() или WriteDBTable1()) и используемые ей вспомогательные функции.  
Затем в DBTableTxtN добавляются функции, разработанные другими членами бригады. Работа считается выполненной, если в проекте студента полностью реализован «базовый» интерфейс класса DBTableTxtN.  
Необходимые вспомогательные функции, которые вы в проекте Lab3\_1 вызывали из библиотеки dbmsLib\_v1, приведены в разделе 3.5 «Исходные коды вспомогательных функций».

- Шаг 1. Добавьте в решение ПКШ\_ЛР3 с пустое консольное приложение по имени Lab3\_2, создайте в нём файлы testLab3\_2.cpp , DBTableTxtN.cpp и DBTableTxtN.h (для бригады 1 имена файлов будут DBTableTxt1.cpp и DBTableTxt1.h).  
В файле DBTableTxt1.h по шагам будем формировать описание класса, а в файле DBTableTxt1.cpp – определение класса.  
 Библиотеку dbmsLib подключать не нужно, а тестовая БД в решении уже есть.  
Файл testLab3\_2.cpp – запускаемый файл. Скопируйте в него содержание файла testLab3\_1.cpp из проекта Lab3\_1. Замените оператор подключения библиотеки оператором   
 #include "DBTableTxt1.h".  
Вставьте в файл DBTableTxt1.h из библиотечного файла dbmsLib\_v1\_1.h всё, что на ваш взгляд необходимо для описания разрабатываемого вами класса DBTableTxt1.

В файл DBTableTxt1.cpp из файла testLab3\_2.cpp переместите всё , что на ваш взгляд относится к определению перегруженной вами функции ReadDBTable1( ) и используемых в ней глобальных функций и переменных.

- Шаг 2. Скомпилируйте проект. Компиляция завершиться успешно, даже если некоторые функции недоступны. В этом случае при попытке выполнить программу ошибки будет выдавать редактор связей.   
Добейтесь того, чтобы программа компилировалась без ошибок. Для этого исключайте (для начала комментируйте) из файлов лишнее и добавляйте то, чего по вашему мнению не хватает для того, чтобы ваш класс выполнял перегруженный вами метод, например, только читал или только печатал или только записывал таблицу. Каждый член группы делает этот шаг отдельно, при необходимости согласовывая решения с остальными членами группы.

Начинайте с самого простого проекта (закомментируйте почти всё). Только после того, как вы добьётесь успешной компиляции, расширяйте проект. Программа всегда должна «дышать», быть работоспособной. На этом шаге ничего не нужно программировать.   
В результате определятся недостающие функции, которые вы добавите на следующем шаге.   
 - Шаг 3. Для выполнения этого шага вспомните основы программирования (процедурного и ООП): пространства имен, прототипы функций-друзей и методов класса и синтаксис их вызова.   
Для того, чтобы объекты программы, размещенные в разных файлах, видели друг друга, в С++ используется элемент «Пространство имён». В одном пространстве имена должны различаться, а в разных пространствах могут быть объекты с одинаковыми именами. Полное имя включает имя пространства (или перечисление имён всех пространств, в которые вложено пространство имён, где определён объект ) перед именем объекта программы, например,  
 void dbmsLib1:: DBTableTxt1::ReadDBTable1(string fileName),   
в пространстве dbmsLib1 имя этой функции можно определить короче:  
 void DBTableTxt1::ReadDBTable1(string fileName).

Пространства имён ограничиваются фигурными скобками {…}

- Шаг 4. В заключение создайте класс DBTableTxt1с «базовым» интерфейсом, включающим методы ReadDBTable1(), PrintTable1() и WriteDBTable1().  
Для проверки его работоспособности вставьте в функцию testing() код:  
 string path="..\\"+dbName+"\\";

dbmsLib1::DBTableTxt1 tab;

tab.ReadDBTable1(path+tabName+".txt");

tab.PrintTable1(78);//контроль связи с БД

tab.WriteDBTable1(path+tabName+".txt");

tab.ReadDBTable1(path+tabName+".txt");

tab.PrintTable1(78);

Вызовите из меню функцию testing() и убедитесь в правильности разработанного кода.

--------------------------------------------------------------------------------------

### 3.5. Исходные коды вспомогательных функций

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

//возвращает значение переменной value, преобразованное из типа

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

//столбца таблицы в C-файле.

{

switch (hdr[columnName].colType)

{

case Int32: return new int(stoi(value)); break;

case Double: return new double(stod(value)); break;

case String: return new string(value); break;

case Date: return new DBDate(value); break;

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

columnName<<endl;

}

return 0;

}   
//-------------------------------------------------------------------------------------------

string DBTableTxt::valueToString(Row& row,string columnName)

//возвращает значение переменной value, преобразованное из типа

//typeName, имя которого указано в заголовке columnName столбца

//таблицы, в тип string.

//!!! Вставить проверку наличия columnName в DBTableTxt

{

char buf[80]={0};

if (this->columnHeaders.find(columnName)!= this->columnHeaders.end())

{

void\* value=row[columnName];//

ostrstream os(buf,80);

switch(columnHeaders[columnName].colType){

case Int32: os<<setw(8)<<\*(int\*) value<<'\0'; break;

case Double: os<<setw(12)<<\*(double\*) value<<'\0'; break;

case String:os<<setw(columnHeaders[columnName].length)<<

ignoreBlanc(\*(string\*)value)<<'\0'; break;

case Date: os<<setw(12)<<\*(DBDate\*) value<<'\0'; break;

default: cout<<setw(8)<<"NoType"<<'\0';

system("pause");break;

}

}else

cout << "column not found " << endl;

return buf;

}

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

//убирает из строки ведущие пробелы  
//в таблицах БД строки выравнены по правому краю

string ignoreBlanc(const string str){

string bufStr=str;

int begStr=bufStr.find\_first\_not\_of(' ');

return bufStr.substr(begStr);

}

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

//извлечение имени таблицы из полного имени файла (путь из текущей папки)

GetTabNameFromPath(string path){ //fileName=..\\dbName\\tableName+.txt  
//копируем строку в буфер, т.к. c\_str() возвращает const char\* и её нельзя изменять

// char tName[100];

int pos1=path.find\_last\_of('\\');

int pos2=path.find\_last\_of('.');

// strcpy\_s(tName,100,path.c\_str());

// tName[pos2]='\0';

// strcpy\_s(tName,80,tName+pos1+1);

return path.substr(pos1,pos2-pos1-1)// tName;

}

//===================================================  
//Сравнивает по условиям Condition объекты типов DBType   
// При разработке использован приём «множественная диспетчеризация» (двойная)

bool comparator(DBType type, void \*obj1, Condition condition, void \*obj2){

switch (type){

case Int32: switch (condition){

case Equal: return \*(int\*)obj1 == \*(int\*)obj2;

case NotEqual: return \*(int\*)obj1 != \*(int\*)obj2;

case Less: return \*(int\*)obj1 < \*(int\*)obj2;

case Greater: return \*(int\*)obj1 > \*(int\*)obj2;

case LessOrEqual: return \*(int\*)obj1 <= \*(int\*)obj2;

case GreaterOrEqual: return \*(int\*)obj1 >= \*(int\*)obj2;

default: cout<<"Недопустимая операция сравнения\n"<<endl;

return false;

}

case Double: switch (condition){

case Equal: return \*(double\*)obj1 == \*(double\*)obj2;

case NotEqual: return \*(double\*)obj1 != \*(double\*)obj2;

case Less: return \*(double\*)obj1 < \*(double\*)obj2;

case Greater: return \*(double\*)obj1 > \*(double\*)obj2;

case LessOrEqual: return \*(double\*)obj1 <= \*(double\*)obj2;

case GreaterOrEqual: return \*(double\*)obj1 >= \*(double\*)obj2;

default: cout<<"Недопустимая операция сравнения\n"<<endl;

return false;

}

case String: //пропуск ведущих пробелов в string

switch (condition){

case Equal: return ignoreBlanc(\*(string\*)obj1) == ignoreBlanc(\*(string\*)obj2);

case NotEqual: return ignoreBlanc(\*(string\*)obj1) != ignoreBlanc(\*(string\*)obj2);

case Less: return ignoreBlanc(\*(string\*)obj1) < ignoreBlanc(\*(string\*)obj2);

case Greater: return ignoreBlanc(\*(string\*)obj1) > ignoreBlanc(\*(string\*)obj2);

case LessOrEqual: return ignoreBlanc(\*(string\*)obj1) <= ignoreBlanc(\*(string\*)obj2);

case GreaterOrEqual: return ignoreBlanc(\*(string\*)obj1)>= ignoreBlanc(\*(string\*)obj2);

default: cout<<"Недопустимая операция сравнения\n"<<endl;

return false;

}

case Date: switch (condition){

case Equal: return \*(DBDate\*)obj1 == \*(DBDate\*)obj2;

case NotEqual: return \*(DBDate\*)obj1 != \*(DBDate\*)obj2;

case Less: return \*(DBDate\*)obj1 < \*(DBDate\*)obj2;

case Greater: return \*(DBDate\*)obj1 > \*(DBDate\*)obj2;

case LessOrEqual: return \*(DBDate\*)obj1 <= \*(DBDate\*)obj2;

case GreaterOrEqual: return \*(DBDate\*)obj1 >= \*(DBDate\*)obj2;

default: cout<<"Недопустимая операция сравнения\n"<<endl;

return false;

}

default: cout<<"Недопустимый тип данных\n"<<endl;

return false;

}

}  
//-----------------Получить имя типа DBType---------------------------------   
const char\* typeNames[]={"NoType","Int32","Double","String","DBDate"};

const char\* DBTableTxt1::TypeName(DBType type){

return typeNames[type];

}

//----------Создать макет распечатки таблицы---------------------

void DBTableTxt::CreateTableMaket(Strip \*&strips,int &nStrips,int screenWidth)  
{

Header::iterator headerIter,contHeaderIter;

int nColumn = (int)columnHeaders.size();

//Заполнение массива ширины полей для печати таблицы

int\* fieldW=new int[nColumn];

headerIter=columnHeaders.begin();

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

//что шире: заголовок или данные?

fieldW[k]=headerIter->second.length > headerIter->first.size() ?

headerIter->second.length+2 : headerIter->first.size()+2;

headerIter++;

}

int currCol=0;//порядковый номер столбца в таблице

nStrips=1;//число полос в распечатке таблицы

int sumWidth=0;//суммарная ширина столбцов в полосе

int n=0;//число столбцов в полосе

int buff[40]={0};//объявление и обнуление буфера для временного

//хранения числа столбцов в полосе (n<40)

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

if(fieldW[currCol]>=screenWidth-1){

cout<<"Ширина столбца "<<currCol<<" больше ширины экрана\n";

cout<<"Таблицу нельзя распечатать"<<endl;

nStrips=0;

return;

}

sumWidth+=fieldW[currCol];

if((sumWidth<screenWidth-1)&&(currCol<nColumn-1)){

n++;

continue;

}

if((sumWidth>=screenWidth-1)){//выход за границу экрана

currCol--;

buff[nStrips-1]=n;

nStrips++;

n=0;

sumWidth=0;

continue;

}

if(currCol==nColumn-1){//последняя полоса

n++;

buff[nStrips-1]=n;

}

}

strips=new Strip[nStrips];

int col = 0;

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

strips[i].nField=buff[i];

strips[i].fieldWidth=new int[strips[i].nField];

for(int j=0;j<strips[i].nField;j++)

strips[i].fieldWidth[j]=fieldW[col++];

}

}

## 4. UML-диаграмма класса DBTableTxt



## 5. Контрольные вопросы и задания

1. Распечатайте результаты выполнения CreateTableMaket(…) для таблицы Books для разных значений screenWidth, например 38 и 78.  
2. Напишите свою функцию string GetTabNameFromPath1 (string path), используя в ней метод substr(…) класса string. Это может сократить приведённую выше функцию   
string GetTabNameFromPath (string path) до 3-х операторов и сделает её более понятной.   
3. Для приведённой на рис.1 схемы алгоритма напишите оператор для вывода поля заголовка, в котором печатается имя типов данных в столбце.  
4. Что такое последовательный контейнер?   
5. Что такое ассоциативный контейнер?   
6. Что хранится в узлах контейнера *map*?   
7. Что такое итератор?   
8. Как получить доступ к данным в контейнере *vector*?   
9. Посмотрите в отладчике интерфейс контейнера *vector*.  
9. Как получить доступ к данным в контейнере *map*?  
10. Посмотрите в отладчике интерфейс контейнера *map.*

## 6. Рекомендуемые источники информации.

1. Р. Лафоре. Объектно-ориентированное программирование в С++. Издательство ПИТЕР, 2004 г. – 932с.
2. MSDN Library for Visual Studio 2012 (<http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.)
3. Эккель Б. Философия С++. Введение в стандартный С++. 2-е изд.- СПб.: Питер, 2004.- 572с.: ил.