# ПКШ\_ЛР4 Наследование, виртуальные функции и полиморфизм.

Разработка системы классов с абстрактным базовым классом, производным и полиморфным классами.

## 1. Цель работы

Целью работы является приобретение студентами навыков использования в программах наследования, виртуальных функций и полиморфизма.  
Для достижения этой цели в ЛР разрабатывается иерархическая система классов для макета учебной СУБД с абстрактным базовым классом DBTableN, производными классами DBTableTxtN, DBTableBinN, классом DBDateN и полиморфным классом DBTableSetN. В этой и последующих ЛР нужно использовать 2-ю версию библиотеки классов учебной СУБД (файлы dbmsLib\_v2.lib и dbmsLib\_v2.h).

## 2. Содержание работы

ЛР выполняется бригадами из трёх студентов.  
Каждая бригада должна разработать свой полиморфный класс c именем DBTableSetN с «базовым» интерфейсом, позволяющим читать, распечатывать и записывать все таблицы базы БД в текстовый файл. Символ ‘N’ в имени класса – это номер бригады.

Работа состоит из двух частей.   
 В первой части разрабатываются функции для «базового» интерфейса класса DBTableSetN (по одной каждым членом бригады). При выполнении первой части ЛР используются:  
- статическая библиотека классов для учебной СУБД (файлы dbmsLib\_v2.lib и

dbmsLib\_v2.h),

- таблицы БД LibraryTxt с тестовыми данными, созданные в ПКШ2021\_ЛР1,  
 - тестовая БД LibraryBin, предоставляемая преподавателем.

Во второй части разрабатывается своя система классов для макета учебной СУБД, в которой полиморфный класс DBTableSetN выполняет функции базового интерфейса:

- ReadDBN();  
 - WriteDBN();  
 - PrintDBN(int screenWidth).

Для этого в проект члена бригады, в котором в класс DBTableSetN включён метод ReadDBN(), добавьте методы WriteDBN() и PrintDBN(), разработанные другими членами бригады. Начинать нужно с ReadDBN(), так как метод ReadDBN() вызывается в конструкторе DBTableSetN (string dbName), который используется другими методами при создании объекта БД оператором DBTableSetN db(dbName).

*!!! Для упрощения дистанционного контроля выполнения ЛР соблюдайте приведённые ниже соглашения по именованию проектов и файлов.*

ЛР разрабатывается в решении с именем ПКШ\_ЛР4\_N, которое включает два проекта. Первая часть разрабатывается в проекте с именем Lab4\_1\_N, а вторая часть в проекте с именем Lab4\_2\_N.

Разработка базовых функций интерфейса класса DBTableSet1 выполняется путем перегрузки дружественных для библиотечного класса DBTableSet функций   
 friend void ReadDB1(DBTableSet& tab),  
 friend void PrintDB1(DBTableSet& tab, int screenWidth),  
 friend void WriteDB1(DBTableSet& tab).  
Имена функций указаны для бригады номер 1.

!!! Если в файле dbmsLib\_v2.h нужной дружественной функции нет, то добавьте её. Делать это нужно аккуратно. При добавлении не изменяйте содержание других операторов файла и не захламляйте его не используемыми дружественными функциями.

Планируемое время выполнения работы - 8 часов занятий в компьютерном зале (2 часа на создание проектов + 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

### 3.1. Разработка UML- диаграммы системы классов.

Выполнение задания ЛР начинается с *проектирования* системы классов для макета учебной СУБД, а проектирование – с поиска прототипа. ***Прототипом создаваемой системы классов является система классов учебной СУБД, которая представлена статической библиотекой классов dbmsLib\_v2***, но в разрабатываемой системе интерфейс классов будет сокращен до базового.

В процессе *проектирования* объектно-ориентированных программ предметная область задачи, для решения которой разрабатывается программа, представляется в виде совокупности взаимосвязанных классов и их интерфейсов. Результат проектирования описывается с помощью UML-диаграмм взаимодействия классов. Она создаётся совместно всеми членами бригады и используется ими для согласования действий при групповой разработке программ и для связи с заказчиком (преподавателем).   
 UML-диаграмма классов библиотеки dbmsLib\_v2 показана на рисунке 2.   
Класс DBDate на рисунке не показан.   
 В программе описанная UML-диаграммой система классов ***представляется заголовочным файлом***. Это описание не является окончательным, оно может уточняться в процессе разработки программы, но уже позволяет перейти к программированию.

*Каждое изменение должно согласовываться с бригадиром и доводиться до всех членов бригады.*

В проекте Lab4\_1\_N используется заголовочный файл библиотеки dbmsLib\_v2.

В проекте Lab4\_2\_N используется заголовочный файл с именем dbmsLibN.h, описывающий разрабатываемую бригадой № N в проекте Lab4\_2\_N иерархическую систему классов.  
Для бригады №1 этот файл показан на рисунке 1. *Закомментированные методы и виртуальные функции следует оставить, так как в процессе разработки базового интерфейса некоторые из них нужно будет реализовать*.

#pragma once

#ifndef \_dbmsLib1\_

#define \_dbmsLib1\_

#include <iostream>

#include <fstream>

#include <strstream>

#include <iomanip>

#include <string>

#include <vector>

#include <map>

using namespace std;

namespace dbmsLib1

{

//-----------------класс DBDate1----------------------------

class DBDate1

{

static const int arrDays[13];

friend string DateToStr(DBDate1& date);

friend ostream& operator<<(ostream& out,DBDate1& 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:

DBDate1(string date);//формат строки: dd.mm.yyyy

DBDate1(int d,int m,int y); DBDate1():day(0),month(0),year(0){};//???конструктор по умолчанию

DBDate1(DBDate1& dat):day(dat.day),month(dat.month),year(dat.year){} int GetDay();

int GetMonth();  
 int GetYear();

bool operator==(DBDate1& date);

bool operator<(DBDate1& date);

bool operator>(DBDate1& date);

bool operator<= (DBDate1& date);

bool operator>= (DBDate1& date);

bool operator!= (DBDate1& date);

DBDate1& operator+=(int days);//Прибавляет к текущей дате days дней

DBDate1& operator-=(int days);//Вычитает из текущей даты days дней

int operator-(DBDate1& date);//Количество дней между текущей датой и date

//Отсчитывается от текущей даты. Если текущая дата > date, результат < 0.

};

static const char\* typeNames[]={

"NoType",

"Int32",

"Double",

"String",

"DBDate"

};

//DBType-перечисление типов полей таблиц БД:

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

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

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

const int LENGTH1 = 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[LENGTH1];//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, ColumnDesc1> Header;

//----------------class DBTable1--------------------------------------

class DBTable1{//Абстрактный базовый класс

public:

DBType GetType1(char\* columnName);

const char\* TypeName1(DBType type);

virtual ~DBTable1(){};//виртуальный деструктор

virtual Header GetHeader1()=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 operator[](int index)=0;

virtual string valueToString1(Row& row,string columnName)=0;

/\*virtual int GetSize()=0;

virtual void SetHeader(Header& hdr)=0;\*/

virtual void ReadDBTable1(string fileName)=0; //fileName="..\\"+dbName+"\\"+tableName+".bin"|".txt" virtual void WriteDBTable1(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 PrintTable1(int screenWidth)=0;

//virtual Row CreateRow()=0;

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

// virtual DBTable1\* SelfRows(string colName,Condition cond, void\* value)=0;

};

//=============== class DBTableTxt1 =============================

class DBTableTxt1:public DBTable1{

private:

Header columnHeaders;

string tableName;

string primaryKey;

vector<Row> data;

string fileName;

public:

DBTableTxt1(){}

DBTableTxt1(string tabName);/\*: DBTable(tabName){}\*/

DBTableTxt1(string tabName,Header hdr,string primKey); /\*DBTable(tabName,hdr,primKey){}\*/

~DBTableTxt1(){};//!!!деструктор нужно будет перегружать, чтобы память, // выделенная void\*, не утекала

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

string valueToString1(Row& row,string columnName);

void ReadDBTable1(string fileName);//tabName=path+tableName

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

void PrintTable1(int screenWidth);

void WriteDBTable1(string fileName);//tabName=path+tableName

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

int GetSize1();

// 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 GetHeader1();

// void SetHeader(Header& hdr);

// Row CreateRow();

// Row GetRow(int index);

// void AddRow(Row row,int index);

// DBTable1\* 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(Strip1\* &strips,int &nStrips,int screenWidth);

};

class DBTableBin1: public DBTable1{

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

//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:

DBTableBin1(){}

DBTableBin1(string tableName);

DBTableBin1(string tableName,Header hdr,string primKey);

~DBTableBin1(){}

Header GetHeader();

void DBTableClear();

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

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 ReadDBTable1(string fileName);

void WriteDBTable1(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 PrintTable1(int screenWidth);

//Row CreateRow();

//void AddRow(Row row,int index);

//DBTable1\* SelfRows(string colName,Condition cond, void\* value);

};

//===============class DBTableSet1=============================

class DBTableSet1{

private:

string dbName;

map<string, DBTable1\*> db;

public:

DBTableSet1(){};

DBTableSet1(string name);

int ReadDB1();

void PrintDB1(int screenWidth);

void WriteDB1();

string GetDBName(){return dbName;}

DBTable1\* operator[](string tableName);

/\*Relation GetRelation(string relationName);

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

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

};

}

#endif //конец \_dbmsLib1\_  
*Рисунок 1. Заголовочный файл dbmsLib1.h системы классов макета учебной СУБД.*

Система состоит из абстрактного базового класса DBTable1, производного класса DBTableTxt1, полиморфного класса DBTableSet1, и класса DBDate1.   
Класс DBTableBin1 должен быть включён в систему, но исходные коды его методов в данной ЛР не разрабатываются, а заменяются «заглушками». Объекты этого класса создаваться не будут, но его присутствие нужно для создания полиморфного класса DBTableSet1. Класс DBTableBin1 будет разрабатываться в ЛР5.  
 Класс DBDate1 был разработан вами в ПКШ\_ЛР2. Методы класса DBDate1 определены в файле DBDate1.cpp.

Класс DBTableTxt1 с базовым интерфейсом разрабатывался в ПКШ\_ЛР3. Методы класса DBTableTxt1 определены в файле DBTableTxt1.cpp. Для включения его в иерархическую систему классов в его описание нужно добавить базовый класс DBTable1и согласовать имена методов класса с именами виртуальных функций базового класса.

DBTableBin1.cpp

Полиморфный класс DBTableSet1 разрабатывается в данной ЛР. DBTableSet1.cpp  
Базовый интерфейс класса DBTableSet1 должен включать методы:   
 1. ReadDB1 (..) для чтения всех таблиц БД из текстовых файлов,  
 2. PrintDB 1(..) для распечатки всех таблиц БД,  
 3. WriteDB 1(..) для записи всех таблиц БД в файл.

UML-диаграмма класса DBTableSet1 представлена на рисунке 2.  
Множество таблиц БД в классе DBTableSet1 хранится в контейнере   
 map<string, DBTable1\*> db (смотри dbmsLib\_v2.h или UML-диаграмму).  
 Здесь string – имя таблицы БД.   
Поэтому методы класса DBTableSet1 могут обращаться к таблицам только через указатель на DBTable1.  
DBTable1\* может инициализироваться адресом объекта класса DBTableTxt1 или адресом объекта класса DBTableBin1. В С++ механизм виртуальных функций в первом случае будет вызывать методы класса DBTableTxt1, а во втором – DBTableBin1.

### 3.2. Разработка функций базового интерфейса класса DBTableSet

Создайте решение ПКШ\_ЛР4\_N с пустым консольным приложением по имени Lab4\_1, создайте в нём файл с именем   
 testLab4\_1\_N*х*.cpp, где N – индекс бригады, *х* – индекс студента в бригаде.

Файл содержит функции main(), menu(), testing(string dbName), которые приведены ниже, и в него нужно добавить разрабатываемые дружественные функции ReadDB1(..), WriteDB1(..) и PrintDB1(..).

int menu(){

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

cout<<"\t1 - Чтение БД из файла\n";

cout<<"\t2 - Печать БД\n";

cout<<"\t3 - Запись БД в файл\n";

cout<<"\t8 – Тестирование разрабатываемых функций\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;

}

Пример тестирующей программы для проверки таблиц БД и работоспособности проекта :  
 void testing(string dbName){

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

dbmsLib::DBTableSet db(dbName);

db.PrintDB(78);

cout<<"Измените \"в ручную\" одну из таблиц БД"<<endl;

system("pause");

ReadDB1(db);

db.PrintDB(78);

dbmsLib::PrintDB1(db,78);

dbmsLib::WriteDB1(db);

dbmsLib::ReadDB1(db);

dbmsLib::PrintDB1(db,78);

}

Функция main().

int main(){

system("chcp 1251>nul");

//cout<<"Введите имя БД\n"; //"LibraryTxt"

//для удобства тестирования при отладке выполняем присваивание

string dbName("LibraryBin");

//cin>>dbName;

cout<<dbName<<endl;

//string tabName("Students");

//cin>>tabName;

// cout<<"Введите ширину экрана в символах\n";

int screenWidth=78;

// cin>>screenWidth;

dbmsLib::DBTableSet db(dbName);

while(true){

switch(menu()){

case 1: db.ReadDB();

break;

case 2: db.PrintDB(screenWidth);

break;

case 3: db.WriteDB();

break;

case 8:testing(dbName);

break;

case 10: return 0;//завершение работы

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

break;

}

}

return 0;

}   
  
 Подключите к Lab4\_1 библиотеку dbmsLib\_v2 и тестовую БД.  
Скомпилируйте и запустите проект.

Путь к таблицам БД определяется по имени БД и имени таблицы на основании соглашений по именованию и расположению папок и файлов.  
Описание класса DBTableSet1 выполнено в п. 3.1 путем коррекции прототипа - файла dbmsLib\_v2.h и представлено на UML-диаграмме.

Если имя БД LibraryTxt (оканчивается на “Txt”), то реализованный в классе DBTableSet полиморфный вызов виртуальных функций инициализирует DBTable\* адресом DBTableTxt и в результате вызываются переопределения виртуальных функций из класса DBTableTxt.

Если имя БД LibraryBin (оканчивается на “Bin”), то вызываются переопределения виртуальных функций из класса DBTableBin.

#### 3.2.1. Разработка функции для чтения БД

Эта функция должна быть разработана первой.   
Разработку функции ReadDB1() будем выполнять путем перегрузки дружественной для библиотечного класса DBTablSet функции friend void ReadDB1(DBTableSet& tab).  
Для пошагового контроля разработки добавьте в функцию testing() вызов ReadDB1() и печать результатов её выполнения.

- Шаг 1. Чтение имен таблиц БД из файла DBTables.txt в буфер типа vector<string>.  
По принятому в макете СУБД соглашению, в каждой БД имеется файл с именем DBTables.txt, который хранит список имён таблиц БД. Этот файл содержит единственную строку с именами таблиц, разделёнными символом '|’.  
Описание алгоритма:  
 - очистка таблиц БД в ОП для того, чтобы в случае повторного чтения данные не добавлялись бы в таблицу, а записывались заново;  
 - открытие файла DBTables.txt;  
 - чтение имен таблиц в vector<string> с использованием getline().  
 Шаг 2. Для реализации полиморфного вызова функций чтения таблиц БД добавим в

Файл testLab4\_1\_N*х*.cpp функцию string fileType = FileType(string dbName);

Функция возвращает расширение файлов в БД, получаемое из окончания имени БД :  
 - “.Txt” для текстовой БД,  
 - “.Bin” для бинарной БД.

( Во второй части ЛР – проект Lab4\_2\_N - сделаем её методом класса DBTableSet1:   
 string fileType = FileType();  
Его использование потребует уточнения заголовочного файла Lab4\_2\_N.h и UML- диаграммы, так как его не было в библиотеке dbmsLib\_v2).  
 В ОС Windows имена файлов приводятся к нижнему регистру, и .Txt заменится на .txt. Эта функция содержит всего один оператор return …;. Для его написания используйте интерфейс класса string и соглашения по именованию БД.

Шаг 3. Чтение таблиц БД в контейнер db в цикле for:

string fileType=FileType(dbName);//

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

if(fileType==".Txt")

//инициализация DBTable\* адресом DBTableTxt

db[strArray[i]]=new DBTableTxt(strArray[i]);

else if(fileType==".Bin")

//инициализация DBTable\* адресом DBTableBin1

db[strArray[i]]=new DBTableBin (strArray[i]); else{

cout<<"ReadDB1:Ошибка имени БД"<<endl;

return -1;

}

//полиморфный вызов виртуальной функции

db[strArray[i]]->ReadDBTable1(path+strArray[i]+fileType);

}

Для инициализации DBTable\* адресом DBTableBin необходимо подключить к проекту БД LibraryBin. Папка с готовой БД LibraryBin для ЛР4 должна быть включена в папку с описанием ПКШ2021\_Лаб4. (Создание бинарной БД LibraryBin будет выполняться в ЛР5.)

#### 3.2.2. Разработка функции для записи таблиц БД

В цикле for(iter=db.begin();iter!=db.end();iter++)

выполняется полиморфный вызов через указатель на базовый класс DBTable  
переопределённой в классах DBTableTxt1 или DBTableBin1 виртуальной функции WriteDBTable1:  
 iter->second->WriteDBTable1(…);  
Размер кода функции WriteDBTable1 – всего 5 операторов С++.

#### 3.2.3. Разработка функции для печати таблиц БД

В цикле for(iter=db.begin();iter!=db.end();iter++)

выполняется полиморфный вызов переопределённой в классах DBTableTxt1 или DBTableBin1 виртуальной функции PrintTable1(…) через указатель на базовый класс DBTable1.  
Размер кода функции – всего 4 оператора С++.

### 3.3. Разработка системы классов макета СУБД с полиморфным классом DBTableSet.

- Шаг 1. Создание «скелета» проекта.

Добавьте в решение ПКШ\_ЛР4 с пустое консольное приложение по имени Lab4\_2, создайте в нём пустые файлы Lab42\_N.h, testLab42\_Nx.cpp , DBTableN.cpp, DBTableTxtN, DBTableBinN.cpp , DBTableSetN.cpp и DBDateN.cpp (для бригады 1 имена файлов будут DBTableTxt1.cpp, DBTableBin1.cpp и т.д.).   
Библиотеку dbmsLib подключать не нужно, а тестовая БД в решении уже есть.

Замените оператор подключения библиотеки оператором   
 #include "Lab42\_N.h".  
Файл Lab42\_N.h должен быть включен во все файлы с расширением .cpp.  
 Файл testLab42\_Nx.cpp – запускаемый файл. Скопируйте в него содержание файла testLab41\_Nx.cpp из проекта Lab4\_1.

Все .cpp файлы, кроме запускаемого, должны быть определены в едином пространстве имён dbmsLibN (для первой бригады имя пространства - dbmsLib1).  
  
В файл Lab42\_N.h из библиотечного файла dbmsLib\_v2.h вставляется всё, что на ваш взгляд необходимо для описания разрабатываемой вами системы классов для макета СУБД в соответствии с разработанной в п. 3.1. UML-диаграммой взаимодействия классов, которая показана на рисунке 2.

В файл DBTableTxtN.cpp из файла testLab32\_Nx.cpp переместите всё , что на ваш взгляд относится к определению перегруженной вами функции, например, ReadDBTableTxt( ) и используемых в ней глобальных функций и переменных.

В файл DBDateN.cpp вставьте код класса DBDateN из ПКШ\_ЛР2.  
В файл DBTableSetN вставьте из файла testLab41\_Nx.cpp код разработанной вами функции, например, ReadDBN.  
Файлы DBTableN.cpp и DBTableBinN.cpp будут заполняться, по мере необходимости, в процессе отладки.   
Выполните коррекцию кодов в .cpp файлах, связанную с наследованием, переопределением виртуальных функций и заменой перегруженных дружественных функций на методы класса.   
- Шаг 2. Скомпилируйте проект. Компиляция может 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. Новые ошибки будут связаны с добавленным кодом и их будет легче исправлять. Программа всегда должна «дышать», быть работоспособной.   
На этом шаге ничего не нужно программировать.   
В результате должны определиться недостающие функции, которые вы добавите на следующем шаге.   
 - Шаг 3. Для выполнения этого шага вспомните основы программирования на С++: конструкторы, 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 объектов должны различаться, а в разных пространствах могут быть объекты с одинаковыми именами.

Пример программы для тестирования полиморфного класса DBTableSet1:

void testing(string dbName, string tabName){

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

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

dbmsLib1::DBTableSet1 db(dbName);

db.PrintDB1(78);

db.WriteDB1();

db.ReadDB1();

db.PrintDB1(78);

}

Примечание. Метод ReadDB1() должна быть разработан первым, так как он вызывается из конструктора в операторе DBTableSet1 db(dbName) для инициализации таблиц БД.

 Рисунок 2. UML- диаграмма классов библиотеки dbmsLib\_v2



Рисунок 3. UML-диаграмма класса DBTableSet

## 4. Контрольные вопросы и задания

1. Что такое виртуальная функция?   
 2. Что такое абстрактный базовый класс?  
 3. Как можно убедиться в вызове деструктора?  
 4. Для чего нужен виртуальный деструктор?  
 5. Каков порядок вызова конструкторов при создании объекта класса DBTableTxt?  
 6. Каков порядок вызова деструкторов при уничтожении объекта класса DBTableTxt?  
 7. Какие методы класса создаются компилятором по умолчанию?  
 8. В каких случаях и как выполняется перегрузка конструктора копирования?  
 9. В каких случаях и как выполняется перегрузка оператора присваивания?  
10. Что такое итератор? Какие операции с итератором доступны в контейнере *map*?   
11. Напишите фрагмент программы для распечатки содержимого контейнера map<string,int>, в котором итератор используется в качестве параметра цикла и для доступа к данным.   
12. Где и как используется полиморфизм в классе DBTableSet?   
13. В чём отличие в использовании для обобщённого программирования в С++ указателя void\* и указателя на абстрактный базовый класс?

## 5. Рекомендуемые источники информации.

1. Эккель Б. Философия С++. Введение в стандартный С++. 2-е изд.- СПб.: Питер, 2004.- 572с.: ил.
2. MSDN Library for Visual Studio 2012 (<http://msdn.microsoft.com/ru-ru/library/> , раздел: Справочник по С++.)
3. Р. Лафоре. Объектно-ориентированное программирование в С++. Издательство ПИТЕР, 2004 г. – 532с.
4. Брюсс Эккель, Чак Эллисон. Философия С++. Практическое программирование. Издательство ПИТЕР, 2004 г.