# 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 СУБД с абстрактным базовым классом DBTable, производными классами DBTableTxt, DBTableBin и полиморфным классом DBTableSet.

## 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.

 При выполнении ЛР студенты используют статическую библиотеку классов для макета СУБД (файлы dbmsLib\_v2.lib и dbmsLib\_v2.h) и таблицы БД с тестовыми данными, созданные в ПКШ2020\_ЛР1.
Работа состоит из двух частей.
 В первой части разрабатываются функции для «базового» интерфейса класса DBTableSetN (по одной каждым членом бригады).
 Во второй части каждым членом бригады создаётся система классов с абстрактным базовым классом, в которой полиморфный класс DBTableSetN выполняет одну, разработанную им функцию базового интерфейса, а затем в DBTableSetN добавляются методы, разработанные остальными членами бригады. Исключением является метод ReadDBN(), который должен быть разработан первым, так как он вызывается в конструкторе DBTableSetN (string dbName), который используется другими методами при создании объекта БД в операторе:
 DBTableSetN db(dbName);
*!!! Для упрощения дистанционного контроля выполнения ЛР соблюдайте приведённые ниже соглашения по именованию проектов и файлов.*

 ЛР разрабатывается в решении с именем ПКШ2020\_ЛР4\_N, которое включает два проекта. Первая часть разрабатывается в проекте с именем Lab4\_1, а вторая часть в проекте с именем Lab4\_2.

Разработка интерфейса 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 нужными дружественными функциями.

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

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

Выполнение задания ЛР начинается с *проектирования* системы классов для макета 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 объектов 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.
 UML-диаграмма классов библиотеки dbmsLib\_v2 показана на рисунке 1.
Класс DBDate на рисунке не показан.
 В программе описание системы классов UML-диаграммой 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.*

 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.
 Класс DBDate1 был разработан вами в ПКШ\_ЛР2.

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

 Полиморфный класс DBTableSet1 разрабатывается в данной ЛР.
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 DBTableBin. 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
 testLab41\_N*х*.cpp, где N – индекс бригады, *х* – индекс студента в бригаде.
Скопируйте в него содержание файла testLab3*х*.cpp из ПКШ\_ЛР3.
 Подключите к Lab4\_1 библиотеку dbmsLib\_v2 и тестовую БД.
Скомпилируйте и запустите проект.

Путь к таблицам БД определяется в программе 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 файла dbmsLib\_v2.h и представлено на UML-диаграмме.

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

 string path="..\\"+dbName+"\\";
//Для проверки приложения Lab4\_1 используем библиотечные функции

 dbmsLib::DBTableSet db(dbName);

 db.PrintDB(78);

 db[tabName]->ReadDBTable(path+tabName+".txt");

 db[tabName]->PrintTable(78);

}
 В приведенном примере db[tabName] возвращает DBTable\*, который инициализирован адресом таблицы DBTableTxt.
Инициализация выполнена в функции ReadDB(), которая вызывается из конструктора DBTableSet(dbName) в операторе dbmsLib::DBTableSet db(dbName);

Так как имя БД LibraryTxt оканчивается на “Txt”, то реализованный в С++ механизм виртуальных функций заменяет вызов виртуальной функции ReadDBTable() вызовом её переопределения в классе DBTableTxt.

#### 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() и strtok\_s().
 Шаг 2. Для реализации полиморфного вызова функций чтения таблиц БД добавим в интерфейс класса метод
 string fileType = FileType();
Его использование потребует уточнения заголовочного файла и UML- диаграммы
 (его не было в библиотеке dbmsLib\_v2).
 Метод возвращает расширение файлов в БД, получаемое из окончания имени БД :
 - “.Txt” для текстовой БД,
 - “.Bin” для бинарной БД.
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 DBTableBin

 //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);

 }

Инициализацию DBTable\* адресом DBTableBin закомментируем до создания заглушек для методов класса DBTableBin во второй части ЛР.

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

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

выполняется полиморфный вызов переопределённой в классах DBTableTxt1 или DBTableBin1 виртуальной функции WriteDBTable1 через указатель на базовый класс DBTable1:
 iter->second->WriteDBTable1(…);
Размер кода функции - 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 – 4 оператора С++.

## 4. Разработка системы классов с полиморфным классом DBTableSet1.

- 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 исходным кодом.
 Файл testLab42\_Nx.cpp – запускаемый файл. Скопируйте в него содержание файла testLab41\_Nx.cpp из проекта Lab4\_1.

 Все файлы, кроме запускаемого, должны быть определены в едином пространстве имён с именем dbmsLibN (в первой бригаде используйте имя dbmsLib1).

Вставьте в файл Lab42\_N.h из библиотечного файла dbmsLib\_v2.h всё, что на ваш взгляд необходимо для описания разрабатываемой вами системы классов для макета СУБД в соответствии с разработанной в п. 3.1. диаграммой взаимодействия классов.

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 DBDate 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, 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).

 Рис. 1 UML- диаграмма классов библиотеки dbmsLib\_v2



Рисунок 2. UML-диаграмма класса DBTableSet

## 5. 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 оператора присваивания?
 9. В каких случаях и как выполняется перегрузка оператора присваивания?
10. Что такое итератор? Какие операции с итератором доступны в контейнере *map*?
11. Что хранится в узлах контейнера *map*?
12. Напишите фрагмент программы для распечатки содержимого контейнера map<string,int>, в котором итератор используется в качестве параметра цикла и для доступа к данным.
13. Где и как используется полиморфизм в макете СУБД?
14. Как получить доступ к данным в контейнере *map*?
15. Посмотрите в отладчике интерфейс контейнера *map.*

## 6. 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. Вандервуд, Джосаттис. Шаблоны С++: справочник разработчика. Пер. с англ. –М.: Издательский дом “Вильямс”, 2003. – 544 c.