

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

С.С. Хохлова, В.В. Юданов, С.В. Феськов

**Объектно-ориентированное
программирование на языке C++.
Лабораторный практикум**

Учебно-методическое пособие

Волгоград 2016

Рекомендовано к опубликованию Ученым советом
института математики и информационных технологий
Волгоградского государственного университета
(протокол № 10 от 21 ноября 2016 г.)

Рецензент

д.ф.-м.н., зав. каф. теоретической физики и волновых процессов
Волгоградского государственного университета *Михайлова В.А.*

Хохлова, С. С. Юданов, В. В. Феськов, С. В.

Объектно-ориентированное программирование на языке C++. Лабораторный практикум. [Текст] : учеб.-метод. пособие / С.С. Хохлова, В.В. Юданов, С.В. Феськов; Фед. гос. авт. образоват. учреждение выс. образования «Волгогр. гос. ун-т». – Волгоград: Изд. ВолГУ, 2016. – 106 с. Ил.

Пособие содержит описание лабораторных работ по дисциплине «Объектно-ориентированное программирование» по направлениям подготовки бакалавров 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 09.03.04 «Программная инженерия» и магистров 09.04.01 «Информатика и вычислительная техника», 09.04.04 «Программная инженерия». Пособие ориентировано на изучение ключевых понятий объектно-ориентированного программирования; основных инструментов языка C++ для реализации объектного подхода; процедур разработки классов, в том числе на основе композиции и наследования; стратегии использования виртуальных функций. Пособие предполагает знание студентом основ структурного программирования на языке C и навыки работы с персональным компьютером в операционной системе Microsoft Windows.

ВВЕДЕНИЕ

Настоящее учебно-методическое пособие содержит цикл лабораторных работ, направленных на формирование у студента навыков проектирования и разработки объектно-ориентированных программ на языке C++. В пособии содержится описание 7 лабораторных работ, каждая из которых связана с изучением нового понятия ООП и соответствующих языковых конструкций. В ходе выполнения лабораторной работы студент получает опыт использования этих конструкций в собственной программе.

В рамках лабораторного практикума последовательно изучаются: новые возможности C++ по отношению к своему предшественнику – языку Си (включая пространства имен, потоковый ввод/вывод, операторы new и delete, строковый тип данных string и др.); структурные типы данных и способы управления этими данными с помощью функций C++; классы и объекты, инкапсуляция данных; отношения включения – композиция и агрегация классов; перегрузка операций и дружественные функции; отношения обобщения (наследование) и иерархии классов; динамический полиморфизм и виртуальные функции. Для иллюстрации использования новых понятий и языковых конструкций в пособии приводятся фрагменты кода и полные примеры приложений.

Лабораторный практикум предполагает разработку студентом объектно-ориентированного приложения с основными элементами ООП, включая инкапсуляцию, наследование и полиморфизм. Задание на разработку студент получает в начале семестра из списка индивидуальных заданий. По ходу выполнения лабораторных работ разрабатываемое приложение последовательно усложняется за счет добавления в него новых возможностей. Вместе с тем, структура приложения остается простой и прозрачной, что демонстрирует преимущества объектно-ориентированного подхода.

Кроме изучения собственно ООП, пособие ориентировано на освоение интегрированной среды разработки Microsoft Visual Studio. Рассматриваются вопросы разработки консольных приложений, создания многофайловых проектов Visual Studio, использования встроенного отладчика Debug для идентификации проблемных участков исходного программного кода.

Навыки и умения, полученные студентом при выполнении данного лабораторного практикума, могут быть полезны для дальнейшего изучения дисциплин, связанных с алгоритмизацией, программированием и проектированием программного обеспечения. Среди таких дисциплин «Визуальное программирование», «Технологии разработки ПО», «Проектирование информационных систем», «Алгоритмы и структуры данных», «Параллельное программирование», «Программирование мобильных приложений».

ЛАБОРАТОРНАЯ РАБОТА №1

Структуры C++

Цель работы: Изучение языковых конструкций C++, предназначенных для работы с агрегатными (структурными) типами данных.

Теоретические сведения

Язык C++ предоставляет возможности для создания пользовательских типов данных, которые в дальнейшем могут использоваться наряду со встроенными типами (int, float, char и т.д.). Простейшим пользовательским типом данных в C++ является **структура**.

Структура – агрегатный тип данных, способный содержать разнотипные элементы (целочисленные, вещественные, символьные и другие в любых комбинациях). Основное назначение программных структур – упорядочение данных в оперативной памяти и обеспечение удобного доступа к этим данным.

1.1) Определение структуры.

Перед первым использованием структура должна быть определена. В общем виде синтаксис объявления структуры имеет вид

```
struct struct_name
{
    type1 field1;
    type2 field2;
    ...
};
```

Здесь ключевое слово struct начинает определение структуры. Имя структуры struct_name, которое также называют дескриптором структуры, должно быть уникальным и не совпадать с зарезервированными словами языка C++. Далее, type1 – это тип данных 1-го элемента структуры, а field1 – имя 1-го элемента структуры и т.д. В качестве типа данных может выступить любой стандартный тип (int, float, double, char, bool), также объединение (union), перечисление (enum), или другой агрегатный тип (struct, class). Кроме того, элементами структуры могут быть массивы, строки, и указатели. Элементы структуры (field1, field2, field3, ...) часто называют **полями**. Внутри одной структуры поля должны быть уникальными. Как видно из объявления структуры, определение завершается закрывающей фигурной скобкой и точкой с запятой. Определение структуры не приводит к выделению памяти компилятором.

Имена структуры и полей выбираются программистом самостоятельно. Хорошей практикой является использование «говорящих» имен, когда имя структуры и имена всех полей отражают их содержание. Такая практика повышает читаемость программного кода, улучшает возможности его понимания и упрощает поиск ошибок.

Рассмотрим фрагмент кода

```
struct book    // Определение структуры «Книга»
{
    char title[50];    // Название книги
    char authors[50];  // Список авторов книги
    char publisher[25]; // Издательство
    int year;          // Год выпуска книги
    unsigned int pages; // Количество страниц
};
```

Этот фрагмент содержит определение структуры с именем book, содержащей информацию о некоторой книге. Согласно приведенному выше листингу, структура book включает в себя пять полей.

- 1) Поле title – текстовая строка (тип данных – char [50]), содержащая до 49 текстовых символов. Значение 49 определяется тем, что последний элемент строки в языке C должен содержать 0 (нуль-терминированная строка). Поле title предназначено для хранения названия книги, например «Алиса в стране чудес».
- 2) Поле authors – также текстовая строка (тип данных – char [50]) для хранения данных об авторе/авторах. Условимся далее, что поле authors будет содержать фамилию автора и его инициалы, например «Толстой Л.Н.», «Бродский И.А.», «Цветаева М.И.» и т.д. Если авторов несколько, то они задаются через запятую. Пример: «Зельдович Я.Б., Яглом И.М.».
- 3) Поле publisher – текстовая строка (тип данных char [25]) с информацией об издательстве. Примеры: «КоЛибри», «Elsevier» и т.д.
- 4) Поле year хранит целочисленное значение (тип данных – int), означающее год издания книги.
- 5) Поле pages содержит объем книги в страницах (тип данных – unsigned int).

Фактически приведенный фрагмент кода означает создание пользователем своего собственного типа данных (в рассматриваемом случае – структурного). Обычно определения структур располагаются в начале программного модуля, сразу после директив препроцессора (#include, #define и др., см. Приложение Б).

1.2) Объявление структурной переменной.

После определения структуры в программе может быть создана одна или несколько **структурных переменных**. Каждая из этих переменных должна иметь уникальное имя. Структурная переменная содержит информация о некоторой *конкретной* книге.

Объявление структурной переменной в C++ во многом аналогично объявлению переменной встроенного типа. Синтаксис объявления в общем виде

```
struct_name struct_var;
```

Здесь `struct_name` – имя структурного типа (для приведенного выше примера – `book`), `struct_var` – имя структурной переменной. Имя переменной может быть любым со стандартными ограничениями, касающимися всех имен в C++.

Отметим одно из различий языков C и C++ в части синтаксиса структур: если в C обязательным является использование ключевого слова `struct` при объявлении структурной переменной, то в C++ это слово не используется. Это еще раз подчеркивает то, что структура C++ – это полноценный новый тип данных.

Примеры объявления структурных переменных:

```
book b1, b2, b3;    // объявляем переменные b1, b2 и b3 структурного типа
book shelf[10];     // массив из 10-ти структур типа book с именем shelf
book *pointer;      // указатель на структурную переменную
```

Так же, как и в случае с переменными встроенных типов, структурная переменная может быть объявлена в любом месте программы. Важно при этом, чтобы определение структуры было расположено *до* объявления любых переменных структурного типа.

С точки зрения компилятора, объявление структурной переменной приводит к выделению оперативной памяти необходимого размера для данной переменной. Если выделение памяти произошло успешно, выделенный участок «закрепляется» за структурной переменной, в том смысле, что доступ к нему возможен только по имени переменной.

Структурная переменная может быть **инициализирована** при объявлении. Это означает, что после выделения оперативной памяти переменная заполняется заданными значениями. Синтаксис объявления с инициализацией следующий

```
struct_name struct_var = { field1_value, field2_value, field3_value, ... };
```

При выполнении данного оператора создается переменная с именем `struct_var` структурного типа `struct_name`, причем полю `field1` присваивается значение `field1_value`, полю `field2` – значение `field2_value`, полю `field3` – значение `field3_value` и т.д. Описанная языковая конструкция используется в случаях, когда значения полей создаваемой структурной переменной известны уже на этапе компиляции. Пример

```
book favorite = { «Обломов», «Гончаров И.А.», «Астрель», 2012, 608 };
```

1.3) Использование структуры. Доступ к полям.

Содержимое некоторой структурной переменной может быть прочитано и/или изменено программистом на уровне отдельных полей. Поэлементный доступ осуществляется с помощью оператора точки «.». При этом указывается имя структурной переменной, далее знак точки и имя интересующего нас поля (`struct_var.field`). Примеры

```
b1.title = «Приключения Тома Сойера и Гекльберри Финна»;
b2.title = «Мартин Иден»;
```

```
b3.year = favorite.year;
printf("%s", b1.pages);
shelf[3].authors = "Ильф И.А., Петров Е.П.";
```

Предполагается, что структурные переменные b1, b2, b3, favorite и shelf были объявлены ранее так, как указано выше. Из приведенных примеров видно, что конструкция struct_var.field может использоваться как справа от знака равенства (для чтения значения поля), так и слева от него (для изменения значения). Последний пример демонстрирует использование доступа к массиву структур shelf.

1.4) Пространства имен.

C++ во многих смыслах является наследником языка C и поддерживает большинство конструкций своего предшественника без изменений. Вместе с тем, в C++ введен ряд дополнений, призванных облегчить работу программиста. Одной из таких особенностей является поддержка пространств имен (namespaces). Конструкция namespace была введена в C++ для того, чтобы избежать конфликтов имен при использовании сторонних библиотек. Такие конфликты возникают при совпадении имени пользовательской переменной, константы или функции с именем переменной, константы или функции в подключаемом библиотечном модуле.

Язык C++ позволяет определить для каждой библиотеки свое пространство имен и таким образом избежать возможных конфликтов. К примеру, функции стандартной библиотеки времени исполнения языка C (C Run-Time library, CRT) погружены в пространстве имен std. Эти функции могут использоваться в программе C++, если в начале программного модуля имеется директива using namespace std.

В C++ введен новый формат подключения заголовочных файлов стандартной библиотеки – расширение .h не указывается, а перед названием библиотеки добавляется латинская буква c. Например, #include <cstdio>.

1.5) Примеры консольных приложений Visual C++.

Рассмотрим пример приложения, использующего описанную выше структуру book. В приложении создается одна переменная-структура с именем my_book. Поля структуры заполняются некоторыми данными. Далее содержимое всех полей выводится на экран.

===== ЛИСТИНГ 1.1 =====

```
#include "stdafx.h"
#include <locale>
#include <cstdlib>
#include <cstring>
using namespace std;
```

```

// Определение структуры - сразу за блоком подключения библиотек
struct book
{
    char title[50];
    char authors[50];
    char publisher[25];
    int year;
    unsigned int pages;
};
book my_book; // Объявление структурной переменной
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    strcpy_s(my_book.title, "Обратные тригонометрические функции");
    strcpy_s(my_book.authors, "Новоселов С.И.");
    strcpy_s(my_book.publisher, "Учпедгиз");
    my_book.year = 1956;
    my_book.pages = 125;
    printf("    ИНФОРМАЦИЯ О КНИГЕ\n");
    printf("Название: %s\n", my_book.title);
    printf("Автор(ы): %s\n", my_book.authors);
    printf("Издат-во: %s\n", my_book.publisher);
    printf("Год      : %i\n", my_book.year);
    printf("Страниц : %i\n", my_book.pages);
    system("pause");
    return 0;
}

```

=====

Обратим внимание на следующие особенности приведенного примера:

- 1) Заголовочные файлы библиотек CRT (locale, stdlib, string) включены в формате C++; для доступа к библиотечным функциям из пространства имен std использована директива using.
- 2) Определение структуры book расположено в самом начале программного модуля, до момента объявления первой структурной переменной (в данном случае – переменной my_book).
- 3) Для изменения значений текстовых полей (title, authors, publisher) используется функция strcpy_s из библиотеки string. Это «безопасный» (safe) аналог функции strcpy. Отметим, что использование оператора присваивания, например my_book.publisher = “Учпедгиз”, в данном случае приведет к утечке оперативной памяти (объясните почему).

Следующий пример демонстрирует работу с массивами структур. Обратите внимание на то, как объявляется массив из N структур с именем collection и как в дальнейшем происходит доступ к отдельным его элементам.

ЛИСТИНГ 1.2

```

#include "stdafx.h"
#include <locale>
#include <stdlib>
#include <cstring>
#include <Windows.h>
using namespace std;
struct book
{
    char title[50];
    char authors[50];
    char publisher[25];
    int year;
    unsigned int pages;
};
const int N = 2;
book collection[N]; // создаем массив из N структур book
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    for (int i = 0; i < N; i++)
    {
        printf("\n\nВведите данные для книги №%d\n", i + 1);
        printf("    Название - ");
        SetConsoleCP(1251);
        gets_s(collection[i].title, 49);
        SetConsoleCP(866);
        printf("    Автор(ы) - ");
        SetConsoleCP(1251);
        gets_s(collection[i].authors, 49);
        SetConsoleCP(866);
        printf("    Издательство - ");
        SetConsoleCP(1251);
        gets_s(collection[i].publisher, 49);
        SetConsoleCP(866);
        printf("    Год выпуска - ");
        scanf_s("%d", &collection[i].year);
        getchar();
        printf("    Число страниц - ");
        scanf_s("%u", &collection[i].pages);
        getchar();
    }
    char find_title[50];
    printf("\n\n Введите название искомой книги - ");
    SetConsoleCP(1251);
    gets_s(find_title, 49);
    SetConsoleCP(866);
    bool found = false;
    for (int i = 0; i < N; i++)
    {
        if (strcmp(collection[i].title, find_title) == 0)
        {

```

```

        found = true;
        printf("\n===== НАЙДЕНА КНИГА =====\n");
        printf("  Название:      ");
        puts(collection[i].title);
        printf("  Автор(ы):      ");
        puts(collection[i].authors);
        printf("  Издательство: ");
        puts(collection[i].publisher);
        printf("  Год выпуска:  %d\n", collection[i].year);
        printf("  Страниц:      %u\n", collection[i].pages);
    }
}
if (!found)
    printf(" Книга с таким названием не найдена в коллекции!\n");
system("pause");
return 0;
}

```

Этот пример также демонстрирует использование:

- 1) Операторов scanf, printf, gets_s, puts для ввода/вывода данных (подробное описание – см. доп. литературу и справочные ресурсы Интернет).
- 2) Функции SetConsoleCP из библиотеки Windows.h> для изменения кодовой страницы. Установки кодовых страниц позволяют корректно вводить текстовые данные на русском языке (кодировка 1251) и отображать после ввода (кодировка 866).
- 3) Функции strcmp из библиотеки <cstring> для сравнения текстовых строк.
- 4) Логической переменной (флага) found для индикации успешного поиска.

Задание на лабораторную работу

- 1) Спроектируйте программную структуру для своего варианта. Обоснуйте выбор типа данных для каждого поля структуры.
- 2) Реализуйте консольное приложение, использующее спроектированную структуру. Внутри приложения: а) определите новый структурный тип; б) создайте массив из нескольких (5-10) структур; в) заполните структуры поэлементно, используя ввод с клавиатуры или текстового файла (Приложение Д); г) выведите на экран содержимое всего массива; д) реализуйте в программе дополнительные функции, указанные в задании для лабораторной работы (см. свой вариант).

Процедура сдачи лабораторных работ

- 1) Получите свой вариант задания на лабораторную работу у преподавателя (Приложение Г).
- 2) Ознакомьтесь с описанием лабораторной работы. Изучите необходимые языковые конструкции, разберите примеры. При необходимости обратитесь к лекционным ма-

териалам, дополнительной литературе и справочным ресурсам в локальной сети кафедры ИСКМ и в сети Интернет.

- 3) Спроектируйте программу в соответствии с полученным заданием. В качестве основы разрешается использовать примеры, приведенные в данном методическом пособии. Вместе с тем, творческая инициатива студента (например, дополнительная функциональность программы, улучшенная структура, более удобный интерфейс и т.д.) также всячески приветствуется.
- 4) Наберите код приложения в выбранной среде программирования, например Microsoft Visual Studio (Приложение А). Выполните трансляцию приложения, при необходимости исправьте синтаксические ошибки в коде (ошибки времени компиляции).
- 5) Проверьте работу приложения для различных входных данных. В случае некорректной работы используйте пошаговый режим выполнения программы и встроенный отладчик (меню debug в среде Visual C++) для поиска ошибок времени исполнения (Приложение В).
- 6) При возникновении существенных затруднений на любом из описанных выше этапов обратитесь за помощью к преподавателю.
- 7) Сдайте проверенную программу преподавателю, предварительно оформив в тетради шапку отчета. Во время сдачи программы студенту необходимо
 - показать на примерах корректность работы программы, т.е. их соответствие заданию;
 - при необходимости кратко сформулировать алгоритм работы программы и показать, как этот алгоритм реализован в конкретной программе;
 - знать и при необходимости объяснить назначение и принцип работы всех используемых языковых конструкций;
 - понимать назначение и смысл всех использованных в программе переменных, констант, функций, классов и т.д.

При успешной сдаче этой части лабораторной работы преподаватель выставляет свою подпись и дату в поле «Программа».

- 8) Полностью оформите отчет в тетради (см. пример). Подготовьтесь к ответу на контрольные вопросы.
- 9) Сдайте отчет по лабораторной работе преподавателю. Преподаватель проверяет: полноту и правильность оформления отчета, корректность записи кода и результатов работы программы. Теоретические знания студента проверяются с помощью контрольных вопросов (выборочно).

- 10) Получите подпись преподавателя в поле «Отчет». Лабораторная работа считается сданной.

Иванов И. И. гр. ИВТ – 000	Лабораторная работа № 1 Структуры C++	Программа
		Отчет

Содержание отчета по лабораторной работе

- 1) Стандартная «шапку» отчета (см. выше).
- 2) Цель: формулировка цели работы.
- 3) Теория: краткие сведения о конструкциях C++, изученных в работе (объем 1-2 стр.).
- 4) Программа:
 - код некоторых компонентов разработанного приложения;
 - индивидуальное задание;
 - определение используемой структуры;
 - основные функции, использованные в этой работе с указанием библиотек.

Контрольные вопросы

- 1) Подключение заголовочного файла стандартной библиотеки языка C++. Пространства имен и их использование. Примеры.
- 2) Синтаксис определения структуры данных в языке C++. Примеры.
- 3) Синтаксис объявления новой структурной переменной? Примеры.
- 4) Синтаксис доступа к полям структуры. Примеры.
- 5) Инициализация полей структурной переменной при ее объявлении. Примеры.
- 6) Создание массива структур и работа с ним. Примеры.
- 7) Почему в листинге 1.1 использование конструкции типа

```
my_book.publisher = "Учпедгиз";
```

ведет к утечке памяти? Обоснуйте ответ.

- 8) В чем заключается разница между массивом и структурой? Дайте определение массива и структуры. Приведите примеры.
- 9) Поясните параметры функции:

```
int _tmain(int argc, _TCHAR* argv[]) {}.
```
- 10) Для чего нужны следующие функции в коде лабораторной работы 1: printf(), puts(), scanf_s(), gets_s(), getchar(), SetConsoleCP(), system()?

ЛАБОРАТОРНАЯ РАБОТА №2

Строки, потоки, функции

Цель работы: Изучение новых возможностей C++ (текстовые строки, потоковый ввод/вывод, динамическое выделение памяти) и языковых конструкций для обработки структурных данных с помощью пользовательских функций.

Теоретические сведения

2.1) Текстовые строки в C++. Класс **string**.

В языке C для хранения текстовой информации традиционно используются символьные массивы (`char []`). Признаком окончания текстовой строки в таком массиве является нулевой символ `'\0'` (*нуль-терминированные строки*). При написании программ размер символьного массива часто задается на этапе компиляции, тогда как содержание текстовой строки определяется на этапе выполнения программы (например, вводится с клавиатуры). В результате одной из частых ошибок при работе со строками Си является выход за пределы массива (сообщение `Segmentation fault`). Кроме того, нуль-терминированные строки оказываются неудобными для копирования текста из одной переменной в другую, при конкатенации (слиянии, объединении) строк, при сравнении двух строковых переменных и др.

Одним из новых компонентов языка C++ является стандартный класс **string** для работы с текстовыми строками. Класс содержит большой набор функций, облегчающий работу с текстом и исключающий большую часть описанных выше ошибок.

Для использования класса `string` в программе C++ необходимо подключить библиотеку `<string>` и сделать класс видимым в пространстве имен `std`.

```
#include <string>
using namespace std;
```

Следующий фрагмент кода демонстрирует основные приемы работы с текстовыми строками:

```
string s1;                // создание пустой строки
string s2 = "Иван";       // создание и инициализация
string s3("Петрович");    // другой вид инициализации
s1 = "Сидоров";           // операция присваивания
string s4 = s1 + s2 + s3; // объединение строк
s1[2] = 'Д';              // работа с отдельными символами строки
if (s1 == s2)             // сравнение строк между собой
    printf("Строки равны!");
```

Переопределим далее структуру `book`, используя новый тип данных

```
struct book
{
    string title;
```

```

string authors;
string publisher;
int year;
unsigned int pages;
};

```

При таком определении пропадает необходимость явно указывать максимальное количество символов в строке, так как размер строки всегда будет соответствовать ее содержимому.

2.2) Поточковый ввод/вывод с консоли. Объекты `cin` и `cout`.

Ввод/вывод в C++ основан на концепции потоков. Входные и выходные данные представлены интуитивно как входной и выходной потоки, которые видит программа вне зависимости от их источника и приемника. Под потоком понимается последовательность символов, для которой определены операции вставки элемента (символа, байта) в поток и извлечения элемента из потока.

Для использования стандартных потоков, связанных с консолью (клавиатура и текстовый экран), в программе C++ требуется подключение заголовочного файла библиотеки `<iostream>` и использования пространства имен `std`. В библиотеке `<iostream>`, в частности, определены потоковые объекты `cin` и `cout`, связанные с клавиатурой и дисплеем, соответственно. Объект `cin` использует операцию `>>` для занесения (записи) значения в поток, объект `cout` – операцию `<<` для извлечения (чтения) элемента из потока. Рассмотрим в качестве примера следующий фрагмент.

```

cout << "\nВВЕДИТЕ ДАННЫЕ О КНИГЕ:\n";
cout << "    Название    : ";
getline(cin, collection[i].title);
cout << "    Автор(ы)     : ";
getline(cin, collection[i].authors);
cout << "    Издатель      : ";
getline(cin, collection[i].publisher);
cout << "    Год выпуска  : ";
cin  >> collection[i].year;
cout << "    Страниц      : ";
cin  >> collection[i].pages;

```

Обратите внимание на то, что поток `cout` допускает использование ESC-последовательностей для управления выводом (символы перевода строки `'\n'` и др.). Для ввода текстовых строк, содержащих пробелы, мы используем функцию `getline` вместо оператора `>>`. Поскольку названия книги, список авторов и название авторов в общем случае содержит более одного слова, то при попытке использования `cin>>` будет прочитано только одно слово (до первого пробела), в то время как `getline` позволяет прочесть строку целиком до перевода каретки на новую строку.

2.3) Форматированный вывод на экран чисел с плавающей точкой.

При работе с числами типа `float` и `double` иногда необходимо использовать определенный формат, позволяющий представить числа в удобной форме. Для этого в программе необходимо использовать специальные инструкции, которые подскажут компилятору форму вывода числа. Например, для вывода вещественного числа с двумя знаками после запятой нужно использовать следующие команды:

```
cout.setf(ios::fixed);      // не использовать число e для представления
cout.setf(ios::showpoint);  // показывать нули после точки
cout.precision(2);          // точность - 2 знака после запятой
```

После выполнения этих инструкций все вещественные числа будут выводиться на экран в формате с двумя знаками после запятой. Обозначение `ios` – это сокращение от `input and output stream` (входной и выходной потоки). Имя `setf` является сокращением от `set flag`, т.е. установить флаги (инструкции, предписывающие установить одно из двух возможных действий). В таблице 1 приведены форматирующие флаги, используемые функцией `setf`.

Таблица 1 – форматирующие флаги `setf`

Флаг	Значение	Значение по умолчанию
<code>ios::fixed</code>	Позволяет не использовать число <code>e</code> при записи числа с плавающей точкой. При его использовании <code>ios::scientific</code> автоматически сбрасывается.	false
<code>ios::scientific</code>	Использует научный формат при записи числа с плавающей точкой. Установка данного флага сбрасывает <code>ios::fixed</code> .	false
<code>ios::showpoint</code>	Выводит плавающую точку и замыкающие нули. Если флаг сброшен, числа, у которых после точки идут нули, могут выводиться без десятичной точки и нулей (как тип <code>int</code>).	false
<code>ios::showpos</code>	Выводит плюс перед положительными числами.	false
<code>ios::right</code>	Используется для выравнивания текста по правому краю (совместно с функцией <code>width</code> , которая устанавливает ширину поля).	true
<code>ios::left</code>	Используется для выравнивания текста по левому краю (совместно с функцией <code>width</code> , которая устанавливает ширину поля). Сбрасывает флаг <code>ios::right</code> .	false

Функция `width` также является форматирующей при потоковом выводе и задает размер поля, отводимого под число (количество символов).

```
cout<< «Вывод числа с заданной шириной поля»<<endl;
cout.width(4);
cout<<7<<endl;
```

Вызов функции `width` применяется только к следующему выводимому элементу. Для установки ширины поля вывода и количества значащих цифр после запятой можно также воспользоваться функциями-манипуляторами: `setw()` и `setprecision()`:

```
cout<<«Цена продукта:»<<setw(8)<<setprecision(2)<<100.5<<«руб.»<<endl;
```

Для использования функций-манипуляторов необходимо подключить библиотеку `<iomanip>`.

2.4) Ввод/вывод данных из текстовых файлов. Файловые потоки.

В C++ ввод/вывод данных из текстовых файлов возможен как с помощью функций стандартной библиотеки языка C, так и с использованием файловых потоков. Файловый поток рассматривается как последовательность байтов, причем по умолчанию процедуры чтения выполняются с начала файла, а процедуры записи – в конец файла.

Библиотека `<fstream>` предоставляет пользователю класс `ofstream` для операций файлового вывода (Output File stream) и класс `ifstream` для операций ввода данных из файла (Input File stream). Сам ввод и вывод выполняется так же, как и для консоли – с помощью операторов `>>` и `<<`, соответственно.

Рассмотрим простейший пример организации ввода данных из текстового файла

```
ifstream infile;           // Создаем потоковый объект infile ...
infile.open("my_books.txt"); // Открываем поток, связывая его с файлом
infile >> N;                // Считываем данные в переменную N
infile.close();             // Закрываем файловый поток
```

Предполагается, что файл с именем `my_books.txt` существует и находится на диске в той же папке, что и исполняемая программа (структура файла `my_books.txt` приведена в Приложении Д). В приведенном фрагменте производится одиночное считывание из файла. Значение помещается в переменную `N`.

Аналогичным образом может быть организована запись данных в файл. Для этого необходимо сначала создать объект класса `ofstream`, далее связать поток с дисковым файлом и открыть его с помощью операции `open`, произвести чтение данных и закрыть поток командой `close`

```
ofstream outfile;
outfile.open("my_collection.dat");
outfile << N;
outfile.close();
```


Отметим, что приведенные выше примеры не дают полного представления о возможностях файловых потоков `ifstream` и `ofstream`. Подробное описание методов работы с файлами, различных операций чтения и записи и т.д. можно найти в справочной литературе. Далее приведем режимы открытия файлов в языке C++.

Таблица 2 – Режимы открытия файлов

<i>Режим доступа</i>	<i>Описание</i>
<code>ios_base::in</code>	Открывает файл для чтения.
<code>ios_base::out</code>	Открывает файл для записи.
<code>ios_base::ate</code>	Перемещает указатель в конец файла при открытии.
<code>ios_base::app</code>	Открыть файл для записи в конец файла (добавление).
<code>ios_base::trunc</code>	Удаляет содержимое файла, если оно существует.
<code>ios_base::binary</code>	Открытие файла в двоичном режиме.

Пример использования файла в режиме для записи с предварительным удалением содержимого файла, если оно существовало:

```
ofstream outfile;
outfile.open("my_collection.dat", ios_base::out | ios_base::trunc);
```

2.5) Структура как аргумент и возвращаемое значение функции.

Структуры C++ позволяют упорядочить используемые в программе данные и упростить их обработку, что особенно важно в ситуациях, когда объем обрабатываемых данных велик, а они сами имеют сложную внутреннюю организацию. Вместе с тем, в больших программных проектах часто бывает необходимо структурировать не только данные, но и код программы. Одним из традиционных способов повышения эффективности кода является использование подпрограмм. В C/C++ подпрограммы реализуются в виде функций.

Важной особенностью C/C++ является возможность передачи в функцию целой структуры в качестве аргумента (вместо отдельных полей). Запись вызова функции при этом становится более лаконичной, а вероятность появления синтаксических ошибок уменьшается. Кроме того, передача внутрь функции целой структуры позволяет за один вызов обработать содержимое всех полей.

Рассмотрим функцию, выводящую на экран монитора информацию о некоторой книге. Данные передаются в функцию через структуру `book`, определенную выше.

```
void print_book(book abook)
{
    cout << "   Название:   ";
    cout << abook.title << endl;
    cout << "   Автор(ы):   ";
    cout << abook.authors << endl;
```

```

        cout << "    Издательство: ";
        cout << abook.publisher << endl;
        cout << "    Год выпуска: ",
        cout << abook.year << endl;
        cout << "    Страниц:      ",
        cout << abook.pages << endl;
    }

```

Здесь формальный параметр `abook` – представляет собой структуру типа `book`. Содержимое отдельных полей и комментарии к ним выводятся на экран через `cout`.

Следующий фрагмент кода демонстрирует пример вызова функции `print_book` из основной программы. Функция вызывается в цикле несколько раз, при этом ей последовательно передаются в качестве фактического параметра элементы массива `collection`. Это позволяет вывести на экран информацию обо всех книгах в коллекции.

```

for (int i = 0; i < N; i++)
    print_book(collection[i]);

```

Функция C++ может не только принимать структуру в качестве параметра, но и возвращать ее как результат своей работы. Такая функция, как правило, создает новую структуру внутри себя, заполняет поля необходимыми значениями, а затем возвращает структуру целиком как результат. В следующем примере показано определение функции `get_book`, которая запрашивает у пользователя информацию о новой книге и возвращает ее в виде структуры `book`

```

book get_book(void)
{
    book newbook;
    cout << "\nВВЕДИТЕ ДАННЫЕ О НОВОЙ КНИГЕ:\n";
    cout << "    Название    : ";
    getline(cin, newbook.title);
    cout << "    Автор(ы)    : ";
    getline(cin, newbook.authors);
    cout << "    Издатель    : ";
    getline(cin, newbook.publisher);
    cout << "    Год выпуска : ";
    cin >> newbook.year;
    cout << "    Страниц    : ";
    cin >> newbook.pages;
    cin.get();
    return newbook;
}

```

С помощью функции `get_book` теперь можно заполнить массив `collection`

```

for (int i = 0; i < N; i++)
    collection[i] = get_book();

```

2.6) Динамическое выделение памяти. Операторы new и delete.

C++ поддерживает динамическое выделение и освобождение памяти с использованием операторов new и delete. Эти операторы выделяют память для объектов из пула памяти, называемого свободным хранилищем.

Синтаксис операторов new и delete демонстрируют следующие примеры.

```
float *p1;           // объявляем указатель на вещественную переменную
p1 = new float;      // и динамически выделяем для нее память
int *p2 = new int[5]; // выделяем память под массив из 5-ти элементов int
book *p3 = new book[7]; // выделяем память под массив из 7-ти структур book
delete p1;           // освобождаем память под указателем p1 (4 байта)
delete[] p2;          // освобождаем память под массивом p2 (4x5=20 байт)
delete[] p3;          // освобождаем память под массивом структур
```

Из приведенных примеров видно, что для динамического выделения памяти под массив в операторе new достаточно указать тип элементов и их количество в квадратных скобках. С помощью этого оператора в памяти могут размещаться элементы как встроенных, так и пользовательских типов. В случае неудачи динамического размещения (например, при нехватке оперативной памяти) оператор new возвращает пустой указатель (NULL) или выбрасывает исключение.

Обратим внимание на синтаксис оператора delete в случаях, когда освобождается память, занимаемая массивом. В этих случаях перед указателем должны быть обязательно записаны квадратные скобки (см. примеры выше).

2.7) Пример приложения Visual C++.

Рассмотрим пример программы, в которой используются рассмотренные выше языковые конструкции. Программа считывает информацию о книгах из текстового файла (см. пример в Приложении Д), выводит ее на экран, и производит поиск нужной книги по ее названию. Для хранения текстовых строк используется класс string. Ввод данных с клавиатуры и вывод на экран организован через потоковые объекты cin и cout. Некоторые операции со структурами и массивами структур оформлены в виде функций. Оперативная память под массив распределяется динамически с помощью оператора new.

===== ЛИСТИНГ 2.1 =====

```
#include "stdafx.h"
#include <locale>      // поддержка русского алфавита
#include <iostream>    // потоковый ввод/вывод с консоли
#include <fstream>     // файловые потоки
#include <string>       // текстовые строки C++
#include <Windows.h>   // решение проблем кодировки текста
using namespace std;
struct book           // определение структуры book
```

```

{
    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;
};

// Функция read_book реализует считывание информации
// о некоторой книге из файлового потока. Ссылка на
// файловый поток передается как параметр. Считанная
// информация записывается в новую структуру book.
// Структура возвращается в вызывающую программу.
book read_book(ifstream &file)
{
    book abook; // создаем новую структуру
    getline(file, abook.title); // считываем данные из файла
    getline(file, abook.authors);
    getline(file, abook.publisher);
    file >> abook.year;
    file >> abook.pages;
    file.get();
    return abook; // возвращаем результат
}

// Функция print_book выводит на экран информацию
// о книге, переданной в качестве параметра.
// Никакого результата не возвращает.
void print_book(book abook)
{
    cout << "\n Название: "; // используем поток cout
    cout << abook.title << endl; // для вывода на экран
    cout << " Автор(ы): "; // содержимого всех полей
    cout << abook.authors << endl; // структуры abook
    cout << " Издательство: ";
    cout << abook.publisher << endl;
    cout << " Год выпуска: ",
    cout << abook.year << endl;
    cout << " Страниц: ",
    cout << abook.pages << endl;
}

// Функция find_book производит поиск книги в массиве по ее названию.
// Аргументами функции являются: 1) массив структур book, 2) число
// элементов в массиве, 3) название искомой книги.
// Функция не возвращает результата. В случае, если поиск завершен
// успешно, на экран выводится полная информация о найденной книге.
// В случае неудачи на экран выводится соответствующее сообщение.
void find_book(book acollection[], int num, string atitle)
{
    bool found = false; // флаг found показывает, найдена ли книга
    for (int i = 0; i < num; i++) // перебираем все элементы массива
    {
        if (acollection[i].title == atitle) // книга найдена?
        {

```

```

        found = true;                // устанавливаем флаг
        cout << "\n\nНАЙДЕНА КНИГА:"; // и выводим на экран
        print_book(acollection[i]);   // информацию о книге
    }
}
if (!found)                          // если книга не найдена
    cout << "\n\nКНИГИ С ТАКИМ НАЗВАНИЕМ НЕ НАЙДЕНЫ!\n";
}
int N;                              // глобальная переменная - число книг в коллекции
book *collection;                   // указатель для размещения массива
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    ifstream infile;                // файловый поток для ввода данных
    infile.open("my_books.txt");     // открываем файл my_books.txt
    infile >> N;                     // считываем из файла число книг N
    infile.get();                    // очищаем буфер для чтения
    collection = new book[N];        // распределяем в памяти массив из N структур
    for (int i = 0; i < N; i++)      // в цикле считываем из файла
        collection[i] = read_book(infile); // данные о каждой книге
    infile.close();                  // закрываем файловый поток
    for (int i = 0; i < N; i++)      // в цикле выводим на экран
        print_book(collection[i]);  // информацию о всех книгах
    string find_title;               // новая текстовая строка
    cout << "\n\n Введите название искомой книги - ";
    SetConsoleCP(1251);              // переключаем кодовую страницу
    getline(cin, find_title);        // считываем название искомой книги
    SetConsoleCP(866);               // возвращаем кодовую страницу
    find_book(collection, N, find_title); // вызываем функцию поиска книги
    delete[] collection;             // освобождаем выделенную ранее память
    system("pause");                  // задержка завершения программы
    return 0;
}
=====

```

Задание на лабораторную работу

- 1) Реализуйте описанное в лабораторной работе №1 индивидуальное задание с помощью новых возможностей языка C++. А именно, используйте:
 - класс string для работы с текстовыми строками,
 - потоковые объекты cin и cout для ввода/вывода данных с консоли,
 - классы ifstream и ofstream для ввода/вывода данных из файлов,
 - динамическое выделение памяти под массив структур с помощью new и delete.
- 2) Реализуйте ввод исходных данных (массива структур) из текстового файла. Текстовый файл должен содержать информацию о количестве записей и, последовательно, значения всех полей. Оформите чтение данных из файла в виде отдельной процедуры.

- 3) Реализуйте вывод на экран всего массива структур, оформив в виде отдельной процедуры. Для вывода значений вещественных переменных используйте форматный вывод.
- 4) Реализуйте дополнительные функции из своего варианта задания (см. работу №1), оформив их в виде отдельных функций.
- 5) Продемонстрируйте использование встроенного отладчика для пошагового выполнения программы (с просмотром промежуточных значений переменных).

Содержание отчета по лабораторной работе

- 1) Стандартная «шапку» отчета
- 2) Цель: формулировка цели работы
- 3) Теория: краткие сведения о конструкциях C++, изученных в работе (объем 1-2 стр.)
- 4) Программа:
 - код некоторых компонентов разработанного приложения
 - определение используемой структуры,
 - размещение массива структур в динамической памяти,
 - функции, реализующие индивидуальное задание к Вашему варианту (2 функции).

Контрольные вопросы

- 1) Поточковый ввод/вывод на консоль: библиотеки, потоковые объекты, операторы, потоковые манипуляторы.
- 2) Форматный вывод вещественных переменных в потоки. Примеры.
- 3) Файловый ввод/вывод. Примеры.
- 4) Использование структурных переменных в качестве параметра и возвращаемого значения функции. Примеры.
- 5) Сколько символов содержит каждая из представленных ниже констант?
 - а) `'n'`
 - б) `'\n'`
 - в) `"Megan"`
 - г) `"Megan\n"`
 - д) `"M"`
- 6) Какова максимальная длина строки, которую можно поместить в строковую переменную, объявленную ниже? Поясните ответ.
`char symb[7];`
- 7) Какое действие оказывает каждый из перечисленных ниже флагов на формат выводимой информации?
 - а) `ios::fixed;`

- б) `ios::scientific;`
- в) `ios::showpos;`
- г) `ios::right;`
- д) `ios::left.`

- 8) Как происходит работа с динамической памятью в программах на языке C++. Приведите примеры.

ЛАБОРАТОРНАЯ РАБОТА №3

Классы и объекты

Цель работы: Изучение языковых конструкций для работы с классами и объектами, приобретение навыков разработки и использования классов.

Теоретические сведения

3.1) Объектно-ориентированное программирование (ООП) и классы.

Понятие класса является одним из ключевых для объектно-ориентированной разработки программ. ООП дает программисту возможности, которые нельзя реализовать в рамках процедурной парадигмы, то есть с помощью функций C++. Одной из таких возможностей является защита обрабатываемых программой данных от некорректных действий пользователя. Кроме того, использование классов способно сделать структуру программы более прозрачной и легко модифицируемой. По этой причине в настоящее время ООП – это основной метод создания «больших» программных пакетов, содержащих десятки тысяч (и более) строк исходного кода и требующих усилий нескольких групп разработчиков.

С точки зрения внутренней организации, класс представляет собой объединение данных (вообще говоря, разнотипных) с функциями, которые эти данные обрабатывают. Можно говорить о том, что класс C++ является расширением понятия структуры C++. Вместе с тем, объединение данных и функций не является механическим, так как при таком объединении реализуется разграничение доступа к данным.

В общем виде объявление класса выглядит следующим образом

```
class MyClass    // объявление класса MyClass
{
    private:      // закрытая часть класса (по умолчанию)
        // элементы в этой части доступны только из класса
    protected:   // защищенная часть класса
        // элементы в этой части доступны из класса и его потомков
    public:       // открытая часть класса
        // элементы в этой части доступны из любой части программы
};
```

Объявление начинается с ключевого слова `class`, вслед за которым указывается имя разрабатываемого класса (в приведенном примере – `MyClass`). Далее в фигурных скобках идет список *полей* (элементов данных) и *методов* (функций обработки). При этом поля и методы класса разделены на три группы – закрытую (записывается после спецификатора доступа `private`), защищенную (после спецификатора `protected`) и открытую (после спецификатора `public`). Если спецификатор доступа не указан, то по умолчанию элемент(ы) класса считаются закрытыми.

Расположение элемента в той или иной группе (private, protected, public) определяет режим доступности этого элемента. К примеру, поля и методы, объявленные в public-области класса, являются видимыми из любой части программы. С другой стороны, поля и метод из private-области видимы только «внутри» класса, но невидимы «извне». Этот механизм, позволяющий защищать данные, скрывая их от внешнего пользователя, называют *инкапсуляцией данных*. Подробнее о разграничении доступа к полям и методам класса – см. лекции и справочные материалы по языку C++.

Рассмотрим далее следующий пример

```
class book                      // объявление класса book
{
    private:                    // закрытая часть класса
        string title;
        string authors;
        string publisher;
        int year;
        unsigned int pages;
    public:                     // открытая (интерфейсная) часть
        void read_from(ifstream &file); // чтение из файла
        void display();                // вывод на экран
};
```

Здесь объявляется класс book, который содержит несколько полей данных (они повторяют поля структуры из предыдущей лабораторной работы), а также прототипы компонентных функций read_from и display. Обратим внимание на то, что все поля данных расположены в закрытой части класса, а прототипы функций – в открытой (интерфейсной) части.

Метод read_from будем использовать для считывания информации о книге из текстового файла. Структура файла my_books.txt описана в предыдущей работе. Файловый поток, из которого будут считываться данные, передается функции в качестве параметра. Метод display будем использовать для вывода полей book на экран.

Определим далее компонентные функции read_from и display, то есть запишем их программную реализацию. Так как они являются методами класса book, определения имеют следующий вид:

```
void book::read_from(ifstream &file)
{
    getline(file, title);
    getline(file, authors);
    getline(file, publisher);
    file >> year;
    file >> pages;
    file.get();
}
```

```

void book::display()
{
    cout << "=====\\n";
    cout << "  Название:      " << title << endl;
    cout << "  Автор(ы):       " << authors << endl;
    cout << "  Издательство:    " << publisher << endl;
    cout << "  Год выпуска:     " << year << endl;
    cout << "  Страниц:         " << pages << endl;
}

```

Запись вида `void book::display()` сообщает компилятору о том, что далее определяется метод `display` класса `book`. Согласно указанной сигнатуре, эта компонентная функция не принимает параметров и не возвращает результата.

3.2) Использование класса. Создание объектов.

В рамках ООП классы часто называют *абстрактными типами данных* (АТД). В рассмотренном примере класс `book` может считаться описанием некоторой абстрактной книги, обладающей заданными свойствами и поведением. Свойства книги здесь определяются ее полями (название, авторы, издательство и т.д.), а поведение – методами класса (считать данные из файла, вывести на экран и др.).

Реальное использование разработанного класса становится возможным только после того, как в программе будет создан один или несколько экземпляров класса. Эти экземпляры называют *объектами*. Объект выступает воплощением класса (абстрактного типа данных) и имеет конкретное содержание (конкретное название, конкретных авторов и т.д.). Кроме того, объект всегда имеет конкретный адрес в оперативной памяти компьютера. Отношения между классами и объектами в ООП во многом аналогичны отношениям между типами данных (`int`, `float`, `char`, ...) и переменными (`x`, `y`, `z`, `index`, ...).

Создание объекта некоторого класса похоже на объявление переменной. В случае, если объект создается статически, синтаксис имеет вид

```
имя_класса имя_объекта;
```

Статический массив объектов объявляется в программе следующим образом

```
имя_класса имя_массива[число_элементов];
```

Для динамического размещения массива объектов в памяти используется оператор `new`

```
имя_класса *имя_массива = new имя_класса[число_элементов];
```

Освободить память, занятую массивом объектов, можно с помощью оператора `delete`

```
delete[] имя_массива;
```

Рассмотрим несколько примеров

```

book book1, book2, book3;      // создаем 3 объекта класса book
book library[10000];          // создаем массив из 10000 объектов book
book *mylib = new book[n];     // выделяем память под массив из n объектов
delete[] mylib;                // освобождаем выделенную память

```

3.3) Доступ к полям объектов и вызов методов.

Изменение данных в полях объекта и вызов его методов производится с помощью оператора «точка»

```

имя_объекта.поле = ...;
имя_объекта.метод(параметры);

```

Если в программе задано не имя объекта, а указатель на него, то для работы с отдельными компонентами используется оператор «->»

```

указатель_на_объект -> поле = ...;
указатель_на_объект -> метод(параметры);

```

Рассмотрим следующие операторы:

```

book1.title = "Приключения Робинзона Крузо"; // ошибка - доступ закрыт!
book1.year = 2001;                          // ошибка - доступ закрыт!
book1.pages = book2.pages + 1;               // ошибка - доступ закрыт!
book1.display();
library[100].read_from(infile);
mylib[5].display();

```

В последнем примере использованы объявленные ранее объекты book1 и book2, а также массивы library и mylib. Все приведенные операторы соответствуют синтаксису языка C++, однако часть из этих операторов некорректна с точки зрения доступности используемых данных. Вспомним, что поля title, authors, publisher, year и pages объявлены нами в закрытой (private) части класса book. Это означает, что прямой доступ «извне» к этим полям для любого объекта невозможен. В результате попытка изменить, например, значение поля title объекта book1 (см. 1-ю строку) приведет к ошибке времени компиляции.

Таким образом, в нашем случае работа с объектами класса book ограничена вызовом методов read_from и display. Только эти функции могут изменять значения полей объекта (путем ввода из файла) или просто считывать данные (для вывода на экран). Эта технология обеспечивает сохранность и целостность данных внутри объекта.

3.4) Методы геттеры и сеттеры.

В некоторых случаях полный запрет на доступ к полю класса оказывается неудобным решением. Предположим, что нам необходимо найти книгу с названием «Гиперболоид инженера Гарина» в массиве объектов book. В этом случае мы не сможем выполнить поиск с помощью цикла

```

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

```

```

{
    if (collection[i].title == "Гиперболоид инженера Гарина") // ошибка!
        cout << "КНИГА НАЙДЕНА!";
}

```

так как поле title закрыто и попытка доступа к нему через конструкцию collection[i].title приведет к сообщению об ошибке. Решением в данном случае может стать использование специального метода класса book, который будет возвращать значение поля title в качестве результата. Ниже приводится новое объявление класса book, в интерфейсную (открытую) часть которого добавлена функция get_title

```

class book
{
private:
    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;
public:
    void read_from(ifstream &file);
    void display();
    string get_title(); // метод-геттер
};
// реализация метод get_title
string book::get_title()
{
    return title;
}

```

Единственным назначением метода get_title, как показывает этот пример, является передача названия книги. Вызывая эту функцию, мы получаем содержимое поля title из закрытой части объекта, не нарушая при этом его целостность. Методы такого вида часто называют *геттерами*, а в их названии используется слово get (*англ.* получать). Поиск нужной книги в массиве с помощью метода get_title иллюстрирует следующий фрагмент

```

for (int i = 0; i < N; i++)
{
    if (collection[i].get_title() == "Гиперболоид инженера Гарина")
        cout << "КНИГА НАЙДЕНА!";
}

```

В данном случае ошибка доступа не возникает, так как get_title находится в открытой части класса.

Из приведенного примера понятно, что геттеры могут использоваться только для считывания значений полей из закрытой части объекта. В ситуациях, когда программе время от времени необходимо изменять это значение, используют другой специальный метод – сет-

тер. Сеттер устанавливает значение поля объекта, ничего не возвращая. В названии этого метода используют слово *set* (англ. устанавливать). Для поля *title* класса *book* можно определить следующую функцию-сеттер

```
void book::set_title(string atitle)
{
    title = atitle;
}
```

Прототип этой функции, так же, как и функции *get_title*, должен быть включен в объявление класса *book*.

3.5) Указатель *this*.

Обратим теперь внимание на то, что все рассмотренные нами методы класса *book* (*read_from*, *display*, *get_title*, *set_title*) работают с полями объекта, но при этом не принимают объект в качестве аргумента. Вопрос можно поставить следующим образом. Если мы вызываем метод *display()* для конкретного объекта (скажем, *book1*), то каким образом функция *display* «узнает» о том, какие именно данные хранятся в полях объекта *book1*? Ответ прост: эти данные передаются в функцию *неявно* с помощью специального указателя с фиксированным именем *this*. Этот указатель делает объект доступным внутри принадлежащей классу функции.

Изменить значение *this* нельзя, так как это константный указатель. Он является дополнительным скрытым параметром любой нестатической компонентной функции. Вместе с тем, в большинстве случаев у программиста не возникает необходимость обращаться к указателю *this* напрямую, так как он задействуется даже в том случае, если не называется явным образом. В качестве иллюстрации к сказанному далее приведены два варианта реализации метода *set_title*. С точки зрения компилятора, оба эти варианта эквивалентны, хотя в одном из них указатель *this* задействован явно, а в другом – нет.

```
void book::set_title(string atitle)
{
    this -> title = atitle;
}
```

```
void book::set_title(string atitle)
{
    title = atitle;
}
```

В программах C++ чаще можно встретить второй вариант (без явного обращения через указатель *this*).

3.6) Использование программных модулей. Многофайловые проекты.

Одним из способов структурирования кода программы является его разделение на программные модули, которые хранятся в отдельных дисковых файлах. Модули позволяют разбивать сложные задачи на более мелкие и решать их последовательно. Обычно модули проектируются так, чтобы дать возможность их многократного использования.

Модули часто объединяются в пакеты и библиотеки. Удобство использования модульной архитектуры заключается в возможности обновления или замены модуля, без необходимости изменения остальной программы. Примером отдельного программного модуля может служить любой компонент стандартной библиотеки языка C/C++.

Программный модуль C++ как правило представляет собой пару файлов в одном имени, но разными расширениями – «*.h» и «*.cpp». Файл с расширением «*.h» называется заголовочным (header) и выступает в качестве интерфейса для пользователя. В нем содержатся объявления структур, перечислений, классов и т.д., а также прототипы всех функций в библиотеке. Файл с расширением «*.cpp» называют файлом реализации, он содержит определения всех объявленных в заголовочном файле функций, включая методы классов.

Программные проекты, включающие два или более модулей, называют многофайловыми. Хорошей практикой программирования является выделение в отдельные модули логически связанных функций (например, функций для обработки текстовых строк), а также классов. Для удобства использования, имя создаваемого программного модуля должно соответствовать его фактическому содержанию.

В следующем пункте рассматривается пример проекта, реализованного в виде двух программных модулей. Модуль books является библиотечным и содержит объявление и реализацию класса book. Модуль lab3 используется в качестве основного, он содержит функцию main и *использует* модуль books. Добавление модулей в программу Visual Studio описано в Приложении А.

3.7) Пример приложения Visual C++.

Рассмотрим пример проекта, включающего библиотечный модуль books и основной модуль lab3. Заголовочный файл модуля books содержит объявление класса book и функции find_book.

```
===== файл books.h =====  
#ifndef BOOKS_H  
#define BOOKS_H  
#include <string>  
#include <fstream>  
using std::string;  
using std::ifstream;
```

```

class book
{
    private:
        string title;
        string authors;
        string publisher;
        int year;
        unsigned int pages;
    public:
        void read_from(ifstream &file);
        void display();
        string get_title();
};
void find_book(book[], int, string);
#endif

```

=====

Файл books.cpp включает в себя заголовочный файл books.h с помощью директивы #include и, далее, содержит реализацию функций.

===== файл books.cpp =====

```

#include "stdafx.h"
#include "books.h"
#include <iostream>
using namespace std;
// Геттер, передающий значение поля title в вызывающую программу.
string book::get_title()
{
    return title;
}
// Метод read_from класса book считывает информацию о книге
// из файлового потока. Ссылка на файловый поток передается
// как параметр. Метод не возвращает каких-либо значений.
void book::read_from(ifstream &file)
{
    getline(file, title);
    getline(file, authors);
    getline(file, publisher);
    file >> year;
    file >> pages;
    file.get();
}
// Метод display класса book выводит на экран информацию
// о книге. Не возвращает результата.
void book::display()
{
    cout << "=====\n";
    cout << "  Название:      " << title << endl;
    cout << "  Автор(ы):      " << authors << endl;
    cout << "  Издательство:  " << publisher << endl;
    cout << "  Год выпуска:   " << year << endl;
    cout << "  Страниц:       " << pages << endl;
}
// Функция поиска книги в массиве (см. описание в ЛР №2).
void find_book(book acollection[], int num, string atitle)
{
    bool found = false;

```

```

for (int i = 0; i < num; i++)
{
    if (acollection[i].get_title() == atitle)
    {
        found = true;
        cout << "\nНАЙДЕНА КНИГА:\n";
        acollection[i].display();
    }
}
if (!found)
    cout << "\nКНИГА С ТАКИМ НАЗВАНИЕМ НЕ НАЙДЕНА!\n";
}

```

=====

Основной модуль программы. Использует класс book и функцию find_book модуля books.

===== файл lab3.cpp =====

```

#include "stdafx.h"
#include <locale>
#include <iostream>
#include <Windows.h>
#include "books.h"
using namespace std;
int N; // число книг в коллекции
book *collection; // массив объектов класса book
void main(void)
{
    setlocale(LC_ALL, "rus");
    ifstream infile;
    infile.open("my_books.txt"); // открываем файл данных
    if (!infile.is_open()) // файл не найден?
    {
        cout << "Файл данных не найден!" << endl;
        system("pause");
        return;
    }
    infile >> N;
    infile.get();
    collection = new book[N]; // массив из N объектов класса book
    for (int i = 0; i < N; i++) // в цикле вызываем метод read_from
        collection[i].read_from(infile); // для каждого объекта коллекции
    infile.close();
    cout << " СОДЕРЖИМОЕ КНИЖНОЙ КОЛЛЕКЦИИ:\n\n";
    for (int i = 0; i < N; i++) // в цикле вызываем метод display
        collection[i].display(); // для всех объектов коллекции
    string find_title;
    cout << "\n\n Введите название искомой книги - ";
    SetConsoleCP(1251);
    getline(cin, find_title);
    SetConsoleCP(866);
    find_book(collection, N, find_title); // поиск книги
    delete[] collection;
    system("pause");
}

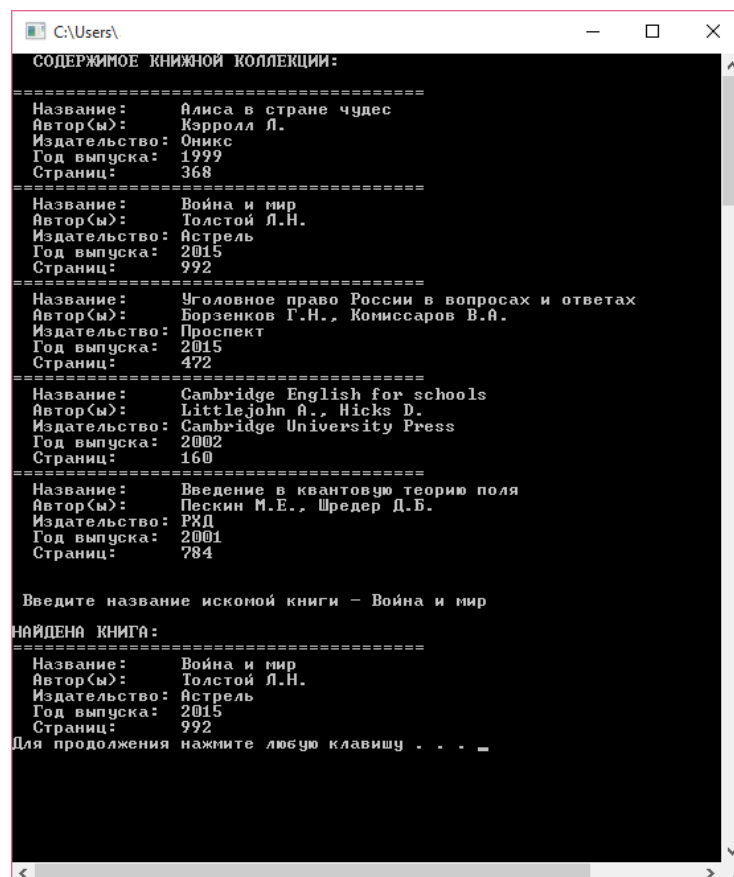
```

=====

Ниже на рисунке ниже приведены результаты работы приложения.

Задание на лабораторную работу

- 1) Разработайте программный класс, объединяющий поля и методы в соответствии со своим вариантом задания. Используйте возможности языка C++ для защиты данных. При необходимости используйте геттеры и сеттеры для доступа к закрытым данным. В качестве методов класса реализуйте:
 - функцию ввода данных с клавиатуры,
 - функцию чтения данных из текстового файла,
 - функцию вывода содержимого на экран,
 - функцию записи содержимого в текстовый или бинарный файл.
- 2) Разместите разработанный класс в отдельном программном модуле. В заголовочный файл модуля поместите объявление класса и прототипы всех используемых функций, в файл реализации – определения всех методов класса и отдельных функций.
- 3) В основном модуле программы:
 - создайте массив из нескольких объектов (динамически),
 - заполните массив данными, используя соответствующие методы,
 - выведите массив на экран, используя соответствующие методы,продемонстрируйте работу дополнительных функций из своего задания.



```
C:\Users\
СОДЕРЖИМОЕ КНИЖНОЙ КОЛЛЕКЦИИ:
=====
Название:      Алиса в стране чудес
Автор(ы):      Кэрролл Л.
Издательство:  Оникс
Год выпуска:   1999
Страниц:       368
=====
Название:      Война и мир
Автор(ы):      Толстой Л.Н.
Издательство:  Астрель
Год выпуска:   2015
Страниц:       992
=====
Название:      Уголовное право России в вопросах и ответах
Автор(ы):      Борзенков Г.Н., Комиссаров В.А.
Издательство:  Проспект
Год выпуска:   2015
Страниц:       472
=====
Название:      Cambridge English for schools
Автор(ы):      Littlejohn A., Hicks D.
Издательство:  Cambridge University Press
Год выпуска:   2002
Страниц:       160
=====
Название:      Введение в квантовую теорию поля
Автор(ы):      Пескин М.Е., Шредер Д.Б.
Издательство:  РХД
Год выпуска:   2001
Страниц:       784

Введите название искомой книги – Война и мир
НАЙДЕНА КНИГА:
=====
Название:      Война и мир
Автор(ы):      Толстой Л.Н.
Издательство:  Астрель
Год выпуска:   2015
Страниц:       992
Для продолжения нажмите любую клавишу . . .
```

Содержание отчета по лабораторной работе.

- 1) Стандартная «шапка» отчета
- 2) Цель: формулировка цели работы
- 3) Теория: краткие сведения о конструкциях C++, изученных в работе (объем 1-2 стр.)
- 4) Программа:
 - код некоторых компонентов разработанного приложения
 - содержимое заголовочного файла (целиком), включая объявление класса,
 - код одной из компонентных функций класса,
 - фрагменты программы, демонстрирующие создание объектов и вызов методов.

Контрольные вопросы

- 1) Объясните разницу между структурой и классом. Чего не хватает структуре, чтобы стать классом?
- 2) Синтаксис объявления класса. Поля и методы. Спецификаторы доступа. Примеры.
- 3) Способы создания объектов: статическое и динамическое размещение в памяти, одиночные объекты и массивы.
- 4) Механизмы разграничения доступа к полям и методам класса. Примеры.
- 5) Геттеры и сеттеры – назначение и синтаксис. Примеры.
- 6) Синтаксис доступа к полям и методам класса. Доступ через указатель на объект.
- 7) Указатель this и его назначение. Явное и неявное использование указателя this в методах класса.
- 8) Создание многофайлового проекта в среде MS Visual Studio.
- 9) Поясните назначение зарезервированных слов private и public в определении класса. Почему все члены класса не определяются как public?
- 10) Какой режим доступа к полям класса следует в определении класса по умолчанию?
- 11) Пусть программа содержит следующее объявление:

```
class Goods
{
public:
    void set_price(double new_price);
    void set_profit(double new_profit);
    double get_price();
private:
    double price;
    double profit;
    double get_profit();
};
```

В основной части программы (main) созданы объекты:

```
Goods Furniture, Textile;
```

Какие из приведенных ниже операторов являются допустимыми?

```
Furniture.price = 10000.0;  
Textile.set_price(2050.0);  
double aprice, aprofit;  
aprice = Furniture.get_price();  
aprofit = Furniture.get_profit();  
aprofit = Textile.get_profit();
```

ЛАБОРАТОРНАЯ РАБОТА №4

Композиция классов

Цель работы: Изучение видов взаимодействия между классами, методов агрегации и композиции классов, приобретение навыков композиции классов в языке C++.

Теоретические сведения

4.1) Взаимодействие классов. Композиция и агрегация.

В рамках объектно-ориентированного подхода изучаемая предметная область описывается в терминах одного или нескольких классов (в общем случае, системы классов). С формальной точки зрения, класс представляет собой «шаблон», который описывает общие свойства и общее поведение всех объектов данного типа. К примеру, программный класс Автомобиль (car) может содержать сведения о владельце автомобиля, его регистрационном номере, дате выпуска, типе и цвете кузова, мощности двигателя и т.д., которые хранятся в отдельных полях. Действия автомобиля на дороге также могут быть реализованы в виде методов класса car, например

- запустить двигатель,
- набрать скорость,
- остановиться,
- повернуть направо,
- включить свет фар и т.д.

Следующим шагом при разработке класса может стать описание взаимодействия автомобилей между собой (уступить дорогу автомобилю справа, включить сигнал поворота и др.), а также описание взаимодействия с пешеходами, автоинспекторами, дорожными знаками, разметкой и т.д. Такой подход позволит в итоге моделировать движение интенсивных транспортных потоков, включающих сотни и тысячи участников дорожного движения.

Когда один класс содержит («has a») в качестве составной части объекты другого класса, то говорят, что имеет место отношение *агрегации* между классами (отношение часть-целое). При нестрогой агрегации часть включается в целое по ссылке, т.е. с помощью указателя на соответствующий класс. Если этот указатель равен нулю, то компонент отсутствует. Строгая агрегация имеет специальное название – *композиция*. Различие между композицией и агрегацией определяется соотношением между временами жизни экземпляра класса-контейнера и экземпляров содержащегося в нем класса. В случае композиции, объекты включенного в контейнер класса прекращают свое существование после уничтожения их контейнера, в случае агрегации – могут продолжить независимое существование.

Рассмотрим пример. Известно, что каждый автомобиль содержит двигатель, кузов, четыре колеса и другие элементы. Учтем это при разработке класса `car` и создадим отдельно классы `engine`(двигатель), `body`(кузов), и `wheel`(колесо). Определим класс `engine` следующим образом

```
class engine          // класс двигателя
{
    ...
    float power;      // мощность
    int gear;         // передача
    ...
    void start();     // запустить двигатель
    void stop();      // остановить двигатель
    ...
};
```

Аналогично определим классы `body` и `wheel`. Затем определим класс автомобиля так

```
class car // класс автомобиля
{
    ...
    engine car_engine; // двигатель
    body car_body;     // кузов
    wheel wheels[4];   // 4 колеса
    ...
};
```

Как видно из приведенного фрагмента, объект `car_engine` класса `engine` включен в тело класса `car` в качестве поля данных. Этот объект описывает двигатель, входящий в состав автомобиля. Таким же способом в тело класса `car` включен один экземпляр класса `body` и массив из четырех экземпляров класса `wheel`. Эти объекты используются для хранения данных о кузове автомобиля и его колесах, соответственно.

Создадим далее экземпляр класса `car` с именем `my_car`

```
car my_car;
```

Доступ к компонентам этого сложного объекта производится стандартным способом – с помощью операции «точка». Следующий фрагмент демонстрирует вызов методов классов `engine` и `wheel` (только для случая, если поля и методы являются открытыми – `public`)

```
my_car.car_engine.start();    // запустить двигатель my_car
my_car.wheels[0].turn_right(); // повернуть направо ...
my_car.wheels[1].turn_right(); // .. два передних колеса
```

Как показывает приведенный пример, композиция помогает создавать сложные классы на основе уже существующих, более простых классов. Сложные классы, полученные в результате композиции, часто называют *классами-контейнерами* или *агрегатными классами*. Так, класс `car` является контейнером для объекта класса `engine`, объекта класса `body` и массива из 4-х объектов класса `wheel`.

В рассмотренном примере, удаление из памяти объекта класса `car` приведет к одновременному уничтожению содержащихся в нем объектов классов `engine`, `body` и `wheel`, поэтому такая связь является композицией. Для композиции характерно включение объектов по значению, для агрегации – по адресу или ссылке.

Проиллюстрируем различие между композицией и агрегацией классов еще одним примером. Рассмотрим отношения между программными классами ВУЗ, Студент и Факультет. Примем во внимание, что в ВУЗе может обучаться любое количество студентов, причем каждый студент может обучаться в одном или нескольких вузах. ВУЗ может состоять из одного или нескольких факультетов, но каждый факультет принадлежит только одному ВУЗу. Тогда отношение между классами ВУЗ и Факультет является композицией, так как при расформировании ВУЗа все Факультеты также должны быть автоматически расформированы. С другой стороны, отношение между классами Студент и ВУЗ является агрегацией, так как студента нельзя удалить при расформировании ВУЗа.

4.2) Диаграмма классов.

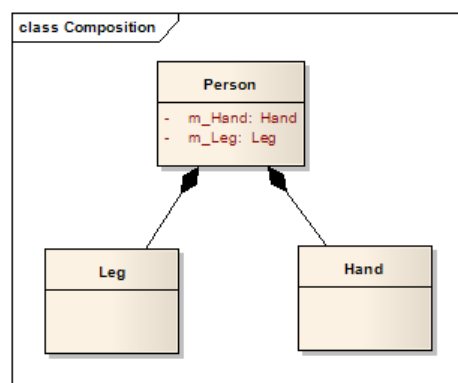
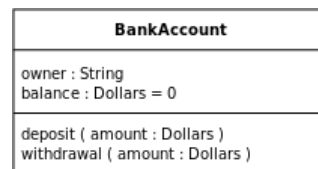
Для изображения структуры классов и отношений между ними используют специальные диаграммы. Эти диаграммы демонстрируют все входящие в систему классы, их атрибуты и методы, а также связи классов друг с другом.

Диаграмма классов используется как при разработке общей концепции приложения (его проектировании), так и на этапе перевода концептуальных моделей в программный код. Каждый класс на диаграмме изображается в виде прямоугольника, разделенного на три части.

Эти части содержат соответственно

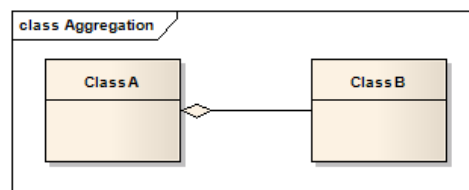
- 1) Имя класса.
- 2) Атрибуты класса (его поля).
- 3) Методы класса.

Пример изображения класса на диаграмме показан справа (Рисунок 3). Класс `BankAccount` (банковский счет) содержит поля `owner` (владелец) и `balance` (баланс), а также методы `deposit` (пополнить счет) и `withdrawal` (снять деньги). Тип данных здесь указан после имени поля через двоеточие.



Для каждого элемента класса на диаграмме может быть указан уровень доступа. Для этого перед именем элемента из области `public` ставится знак «+», из области `private` – знак «-», из области `protected` – знак «#».

Отношения на диаграмме изображаются линиями, соединяющими соответствующие классы. Для изображения композиции используют линию, на одном из концов которой располагается ромб сплошной заливкой. Линия всегда развернута ромбом в сторону класса-контейнера. На рисунке справа показано отношение композиции между классом Person (человек) и классами Hand (рука) и Leg (нога).



Для изображения агрегации рисуют линию, оканчивающуюся полым ромбом (см. рисунок слева).

Для обозначения количества объектов (экземпляров класса), которые могут принимать участие в композиции или агрегации, используется следующая нотация:

Таблица 3 – Обозначения количества объектов

<i>Обозначение</i>	<i>Число экземпляров</i>
0..1	Нет экземпляров или один экземпляр
1	Ровно один экземпляр
0..*	Ноль и более экземпляров
1..*	Один и более экземпляров

Соответствующие обозначения указывается на обоих концах линии, соединяющей классы.

4.3) Класс-контейнер bookstore.

Вернемся далее к приложению, которое разрабатывалось нами в предыдущих лабораторных работах. На данный момент нами спроектирован и реализован класс book, моделирующий отдельную печатную книгу, а также реализованы некоторые процедуры и функции для работы с книжной коллекцией – динамическим массивом из объектов класса book. А именно, реализованы

- 1) процедура загрузки коллекции книг из текстового файла,
- 2) процедура вывода на экран содержимого всей коллекции,
- 3) функция поиска книги по ее названию.

Предположим теперь, что нам необходимо программно представить ассортимент книг некоторого книжного магазина. С точки зрения покупателя, ассортимент представляет собой информацию обо всех печатных изданиях, имеющихся в наличии в магазине на текущий момент времени. Будем считать, что ассортимент может расширяться за счет поставки новых книг. Предусмотрим для нашей программной модели возможность загрузки данных об ассортименте из файла и записи в файл, вывода всего ассортимента книг на экран, а также поиска необходимой книги по названию.

С точки зрения программиста C++, ассортимент представляет собой коллекцию объектов класса `book`, к которой добавлены функции файлового ввода/вывода, вывода на экран, поиска и т.д. Для операций над книжной коллекцией разработаем специальный класс-контейнер `bookstore`.

```
class bookstore// класс книжногомагазина
{
private:
    int max_num_books;      // максимальное кол-во книг (вместимость)
    int num_books;          // текущее число книг в магазине
    book *books;            // массив объектов класса book (коллекция книг)
public:
    bookstore(unsignedint max_nb);      // конструктор класса bookstore
    ~bookstore();                       // деструктор класса bookstore
    void add_book(book abook);          // добавить книгу
    void read_from_file(string filename); // ввести данные из файла
    void write_to_file(string filename); // записать данные в файл
    void display_all();                 // вывести на экран
    void find_book(stringatitle);       // найти книгу по названию
};
```

Указатель `books` в закрытой области этого класса будем использовать для хранения динамического массива объектов `book`. Класс `bookstore` также содержит закрытое поле `max_num_books`, хранящее максимальное число книг в магазине, и закрытое поле `num_books`, показывающее количество книг на данный момент. Предполагается, что значение `num_books` может изменяться по ходу выполнения программы (за счет пополнения ассортимента), тогда как вместимость магазина задается один раз и далее не изменяется.

Рассмотрим далее более подробно методы класса `bookstore`, не обращая пока внимания на конструктор и деструктор (функции `bookstore` и `~bookstore`). В частности, метод `add_book` будет использоваться нами для добавления нового элемента в коллекцию. Этот элемент передается в функцию в качестве параметра.

```
void bookstore::add_book(book abook)
{
    if (num_books < max_num_books)    // можем добавить еще одну книгу?
    {
        books[num_books] = abook;     // заносим книгу в массив
        num_books++;                  // увеличиваем счетчик книг
    }
}
```

По условию, массив `books` не может содержать более, чем `max_num_books` элементов, поэтому используем оператор условия для контроля выхода за его пределы.

Рассмотрим теперь метод `read_from_file`, позволяющий загрузить коллекцию целиком из текстового файла. Имя файла передается функции как параметр.

```
void bookstore::read_from_file(string filename)
{
```



```

ifstream infile;
infile.open(filename);// открываем файл с заданным именем
if (!infile.is_open())
{
    cout<<"\n\nФайл данных не найден!"<< endl;
    system("pause");
    exit(1);
}
int N;
infile>>N;// считываем количество книг в файле
infile.get();// переход на следующую строку
for (int i = 0; i< N; i++)
{
    book new_book;// создаем новый объект
    new_book.read_from(infile);// заполняем его данными из файла
    add_book(new_book);// добавляем в коллекцию
}
infile.close();
cout<<"\nЗагружены данные из файла "<<filename<<":";
cout <<"\n    число загруженных книг - "<< num_books;
}

```

Обратим здесь внимание на внутренний цикл по переменной *i*, в котором: 1) создается новый объект класса *book*; 2) с помощью метода *read_from* данные об очередной книге загружаются из файла; 3) книга заносится в коллекцию.

Разберем также реализацию метода *find_book*.

```

void bookstore::find_book(string atitle)
{
    cout <<"\n\nИщем книгу с названием \""<<atitle<<"\"";
    bool found = false;
    for (inti = 0; i<num_books; i++)
    {
        if (books[i].get_title() == atitle)
        {
            found = true;
            cout<<"\nНайдена книга:\n";
            books[i].display();
        }
    }
    if (!found)
        cout <<"\nКнига с таким названием не найдена!\n";
}

```

Легко заметить, что сигнатура функции *find_book* претерпела изменения по сравнению с предыдущим вариантом (см. лаб. работу №3), так как теперь у нас нет необходимости передавать в функцию массив данных и его размер. Так как *find_book* является методом класса *bookstore*, она имеет прямой доступ ко всем его полям, в том числе массиву *books* и переменной *num_books*.

4.4) Конструкторы и деструкторы.

Работа любого класса C++ невозможна без специальных функций, которые называются *конструктором* и *деструктором* класса. Конструктор класса вызывается *неявно* всякий раз, когда новый объект этого класса размещается в оперативной памяти. Задача конструктора – привести объект в корректное начальное состояние. Обычно для этого требуется инициализировать поля объекта некоторыми значениями, однако иногда необходимы и более сложные действия, такие, как выделение памяти для встроженных массивов, вызов конструкторов встроженных объектов и т.д.

Деструктор класса вызывается неявно в момент, когда объект удаляется из памяти. Задача деструктора – выполнить необходимые действия перед уничтожением объекта – освободить используемую внутри объекта динамическую память, закрыть открытые файлы, послать другим объектам сообщения о прекращении своей работы и т.д.

В случае, если программист не определил конструктор и/или деструктор класса, он будет создан компилятором автоматически, но при этом не будет выполнять никаких действий, кроме размещения объекта в памяти (конструктор) или удаления из нее (деструктор).

Существует ряд правил, касающихся определения этих методов:

- 1) конструктор и деструктор всегда объявляются в разделе `public`;
- 2) тип возвращаемого значения для них никогда не указывается (в том числе – `void`);
- 3) деструктор не может принимать параметров;
- 4) имя конструктора совпадает с именем класса;
- 5) имя деструктора совпадает с именем класса, но с приставкой `~` (тильда);
- 6) в классе допустимо наличие нескольких конструкторов с различающимися параметрами (перегрузка функций), но только один деструктор.

Рассмотрим теперь реализацию конструктора класса `bookstore`, то есть метода `bookstore::bookstore`. Ключевой задачей конструктора будет выделение динамической памяти для массива `books`. Количество элементов в массиве определяется максимальным числом книг и передается в функцию как параметр. Так как в начальный момент времени массив не заполнен, значение `num_books` устанавливается равным нулю.

```
bookstore::bookstore(unsignedint max_nb)
{
    max_num_books = max_nb;
    books = new book[max_num_books];
    num_books = 0;
    cout << "\nВызван конструктор класса bookstore:";
    cout << "\n    выделено объектов - "<< max_num_books;
    cout<< "\n    загружено книг - "<< num_books<< endl;
}
```

Дополнительная информация выводится на экран для контроля работы конструктора.

В свою очередь, деструктор класса `bookstore` должен освобождать память, выделенную конструктором.

```
bookstore::~~bookstore()
{
    max_num_books = 0;
    delete[] books;
    num_books = 0;
    cout<<"\nВызван деструктор класса bookstore:";
    cout <<"\n    выделенная память освобождена";
}
```

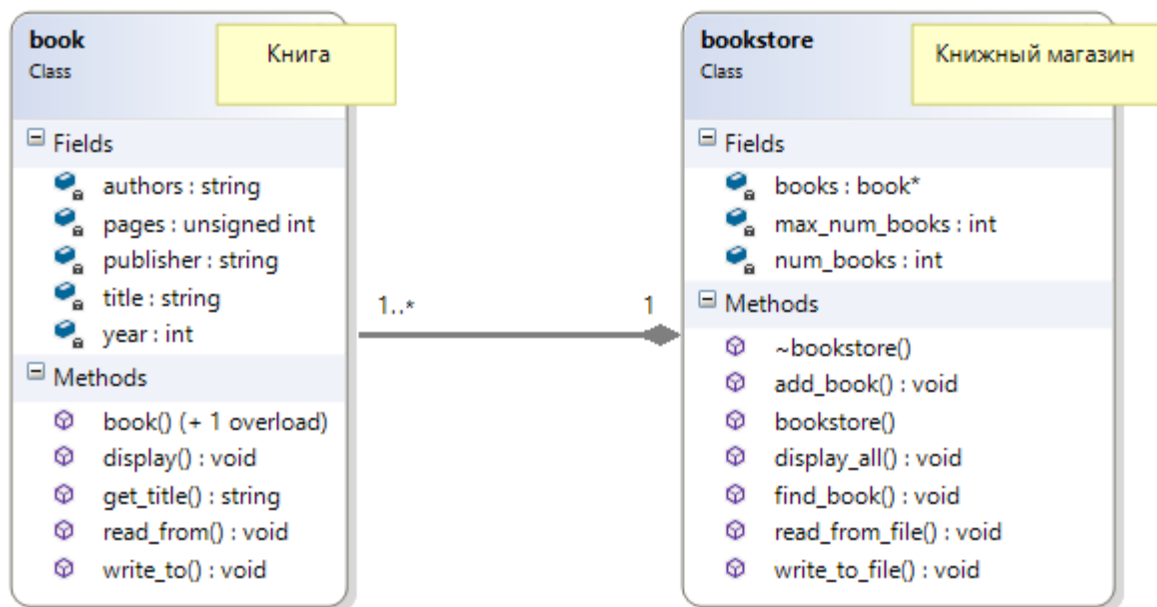


Рисунок 1- Диаграмма классов

4.5) Пример приложения Visual C++.

Рассмотрим теперь целиком программный код проекта, включающего библиотечные модули `books` (разработан нами ранее) и `bookstore`, а также основной модуль `lab4`. Заголовочный файл модуля `bookstore` содержит объявление класса-контейнера. На рисунке 1 изображена диаграмма классов, соответствующая разработанному приложению.

```
===== файл books.h =====
#ifndef BOOKS_H
#define BOOKS_H
#include <string>
#include <fstream>
using std::string;
using std::ifstream;
using std::ofstream;
class book
{
private:
```

```

    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;
public:
    book() : title(""), authors(""), publisher(""), year(0), pages(0) {};
    book(string, string, string, int, unsigned int);
    string get_title();
    void read_from(ifstream& file);
    void write_to(ofstream& file);
    void display();
};
#endif

===== файл books.cpp =====

#include "stdafx.h"
#include "books.h"
#include <iostream>
using namespace std;
book::book(string ttl, string aut, string pbl, int yer, unsigned int pag)
{
    title = ttl;
    authors = aut;
    publisher = pbl;
    year = yer;
    pages = pag;
}
string book::get_title()
{
    return title;
}
void book::read_from(ifstream& file)
{
    getline(file, title);
    getline(file, authors);
    getline(file, publisher);
    file>> year;
    file>> pages;
    file.get();
}
void book::write_to(ofstream& file)
{
    file<< title <<endl;
    file<< authors <<endl;
    file<< publisher <<endl;
    file<< year <<endl;
    file<< pages <<endl;
}
void book::display()
{
    cout<<"\n===== \n";
    cout<<"  Название:      "<< title <<endl;

```

```

        cout<<"  Автор(ы):      "<< authors <<endl;
        cout<<"  Издательство: "<< publisher <<endl;
        cout<<"  Годвыпуска:  "<< year <<endl;
        cout<<"  Страниц:      "<< pages <<endl;
    }

===== файл bookstore.h =====
#ifdef BOOKSTORE_H
#define BOOKSTORE_H
#include <string>
#include <fstream>
#include "books.h"
class bookstore// книжный магазин
{
private:
    int max_num_books;           // максимальное кол-во книг в магазине (вместимость)
    int num_books;               // текущее число книг в магазине
    book *books;                 // массив объектов класса book
public:
    bookstore(unsigned int max_nb);           // конструктор класса bookstore
    ~bookstore();                             // деструктор класса bookstore
    void add_book(book abook);                // добавить книгу
    void read_from_file(string filename);      // ввести данные из файла
    void write_to_file(string filename);       // записать данные в файл
    void display_all();                        // вывести на экран
    void find_book(string atitle);             // найти книгу по названию
};
#endif

===== файл bookstore.cpp =====
#include "stdafx.h"
#include <iostream>
#include "bookstore.h"
using namespace std;
bookstore::bookstore(unsigned int max_nb)
{
    max_num_books = max_nb;
    books = new book[max_num_books];
    num_books = 0;
    cout <<"\nВызван конструктор класса bookstore:";
    cout <<"\n    выделено объектов - "<< max_num_books;
    cout<<"\n    загружено книг - "<<num_books<<endl;
}

bookstore::~~bookstore()
{
    max_num_books = 0;
    delete[] books;
    num_books = 0;
    cout<<"\nВызван деструктор класса bookstore:";
    cout <<"\n    выделенная память освобождена";
}

void bookstore::add_book(book abook)
{

```

```

        if (num_books < max_num_books)    // можем добавить еще одну книгу?
        {
            books[num_books] = abook;      // заносим книгу в массив
            num_books++;                    // увеличиваем счетчик книг
        }
    }
void bookstore::read_from_file(string filename)
{
    ifstream infile;
    infile.open(filename);
    if (!infile.is_open())
    {
        cout<<"\n\nФайл данных не найден!"<<endl;
        system("pause");
        exit(1);
    }
    int N;
    infile>> N;
    infile.get();
    for (inti = 0; i< N; i++)
    {
        book new_book;
        new_book.read_from(infile);
        add_book(new_book);
    }
    infile.close();
    cout <<"\nЗагружены данные из файла "<<filename<<":";
    cout <<"\n    число загруженных книг - "<< num_books;
}
void bookstore::write_to_file(string filename)
{
    ofstream outfile;
    outfile.open(filename);
    outfile<<num_books<<endl;
    for (inti = 0; i<num_books; i++)
        books[i].write_to(outfile);
    outfile.close();
    cout<<"\nДанные записаны в файл "<<filename<<":";
    cout <<"\n    число записанных книг - "<< num_books;
}
void bookstore::display_all()
{
    cout <<"\n\nПОЛНЫЙ АССОРТИМЕНТ КНИЖНОГО МАГАЗИНА \n";
    for (inti = 0; i<num_books; i++)
        books[i].display();
}
void bookstore::find_book(string atitle)
{
    cout <<"\n\nИщем книгу с названием \""<<atitle<<"\"";
    bool found = false;
    for (int i = 0; i<num_books; i++)
    {

```

```

        if (books[i].get_title() == atitle)
        {
            found = true;
            cout<<"\nНайдена книга:\n";
            books[i].display();
        }
    }
    if (!found)
        cout <<"\nКнига с таким названием не найдена!\n";
}

===== файл lab4.cpp =====
#include"stdafx.h"
#include"bookstore.h"
void main()
{
    setlocale(LC_ALL, "rus");
    bookstore my_store(25);
    my_store.read_from_file("my_books.txt");
    my_store.display_all();
    my_store.find_book("Война и мир");
    system("pause");
}

=====

```

Задание на лабораторную работу

- 1) Разработайте класс-контейнер, предназначенный для хранения коллекции объектов своей предметной области. Реализуйте следующие методы класса-контейнера:
 - конструктор и деструктор,
 - добавление нового элемента,
 - удаление элемента из коллекции,
 - загрузка из файла и запись в файл,
 - вывод содержимого на экран,
 - поиск данных в коллекции согласно своему индивидуальному заданию.
- 2) Разместите класс-контейнер в отдельном программном модуле. В заголовочном файле модуля (*.h) поместите объявление класса, в файле реализации (*.cpp) – определения методов класса.
- 3) В основном модуле программы:
 - создайте экземпляр класса-контейнера,
 - заполните контейнер данными из текстового файла,
 - выведите содержимое контейнера на экран,
 - продемонстрируйте выполнение дополнительных функций из своего задания.

Содержание отчета по лабораторной работе

- 1) Стандартная «шапка» отчета
- 2) Цель: формулировка цели работы
- 3) Теория: краткие сведения о композиции классов с примерами (объем 2-3 стр.)
- 4) Диаграмма классов с указанием атрибутов, методов, отношений между классами.
- 5) Программный код:
 - объявление класса-контейнера;
 - код конструктора и деструктора класса;
 - код одной из дополнительных функций (любой).

Контрольные вопросы

- 1) Дайте определение агрегации и композиции. В чем состоит основное отличие между ними?
- 2) Диаграмма классов и отношения агрегации (строгой и нестрогой).
- 3) Конструктор и деструктор: назначение и синтаксис. Примеры.
- 4) Перегрузка конструкторов. Конструктор по умолчанию. Пример.
- 5) Вызов конструктора при создании объекта. Пример.
- 6) Как создать конструктор копии для вашего класса? В каких случаях создание конструктора копирования является залогом правильного функционирования программы?
- 7) Что такое класс-контейнер? Приведите пример.

ЛАБОРАТОРНАЯ РАБОТА №5

Перегрузка операций

Цель работы: Изучение способов определения арифметических и логических операций над объектами C++, приобретение навыков использования операторных функций в консольных приложениях.

Теоретические сведения

5.1) Перегрузка операций C++.

Одной из замечательных особенностей языка C++ является возможность переопределения стандартных арифметических или логических действий (например, «+», «-», «=», «<», «==» и т.д.) для объектов того или иного класса. Такое переопределение называют *перегрузкой операций*.

Основная причина, по которой программист решает перегрузить некоторую операцию – это улучшение читаемости программного кода. В качестве классического примера можно рассмотреть операцию конкатенации (объединения) двух текстовых строк. В языке C для объединения двух нуль-терминированных строк используется стандартная функция

```
char* strcat(char* dest, const char* src);
```

которая добавляет содержимое текстовой строки `src` в конец текстовой строки `dest` и возвращает указатель на полученную строку. С помощью этой функции можно «склеить» две строки `str1` и `str2`, например, следующим образом

```
char str1[10] = "Инь";  
char str2[20] = "Янь";  
strcat(str1, str2);  
cout << str1 << endl;
```

С другой стороны, в языке C++ две текстовые строки можно объединить с помощью простой операции «+», например, так

```
string str1 = "Инь";  
string str2 = "Янь";  
string res = str1 + str2;  
cout << res << endl;
```

Легко видеть, что в данном случае программный код более прозрачен и понятен. Это достигается за счет перегрузки операции сложения «+» для объектов класса `string`.

С точки зрения программиста, перегрузка операции означает определение специальной *операторной функции*. Число аргументов этой функции зависит от арности оператора (унарный или бинарный), а также от способа объявления операторной функции. Существует 2 способа объявления операторной функции:

- 1) как глобальной функции,
- 2) как метода класса.

Рассмотрим эти способы подробнее.

5.2) Перегрузка через глобальную операторную функцию.

Для определения операторной функции в языке C++ используется ключевое слово `operator`, вслед за которым указывается знак операции (символ "+", "=" и т.д.). В программе могут быть определены несколько таких функций, отличающихся только количеством и типом параметров. Таким образом, один и тот же математический символ может приводить к различным действиям, если он применяется к объектам разных классов.

В общем виде операторная функция определяется следующим образом

```
тип_результата operator символ(аргументы)
{
    <тело_операторной_функции>
}
```

В качестве типа_результата здесь может выступать как встроенный тип данных (`int`, `float`, `bool` и т.д.), так и пользовательский класс (например, `book` или `bookstore`). Аргументы должны содержать объекты тех классов, для которых перегружается заданная операция. Количество аргументов функции в данном случае точно соответствует ариности оператора: для унарных операторов функция должна принимать один аргумент, для бинарных – 2 аргумента. Символ оператора может быть одним из следующих

Таблица 4 – Список операторов, которые возможно перегрузить в C++

оператор	значение	арность
,	запятая	бинарный
!	логическое НЕ	унарный
!=	не равно	бинарный
%	остаток от деления	бинарный
%=	остаток с присвоение	бинарный
&	побитовое И	бинарный
&	адрес	унарный
&&	логическое И	бинарный
&=	битовое И с присв.	бинарный
()	вызов функции	-
()	преобразование типа	унарный
*	умножение	бинарный
*	разымен. указателя	унарный
*=	умножение с присв.	бинарный
+	сложение	бинарный
+	унарный плюс	унарный

оператор	значение	арность
->	доступ к полю	бинарный
/	деление	бинарный
/=	деление с присв.	бинарный
<	меньше	бинарный
<<	сдвиг влево	бинарный
<<=	сдвиг с присв.	бинарный
<=	меньше или равно	бинарный
=	присвоение	бинарный
==	равно	бинарный
>	больше	бинарный
>>	сдвиг вправо	бинарный
>>=	сдвиг с присв.	бинарный
>=	больше или равно	бинарный
[]	элемент массива	-
^	исключающее ИЛИ	бинарный
^=	искл. ИЛИ с присв.	бинарный

++	инкремент	унарный
+=	сложение с присв.	бинарный
-	вычитание	бинарный
-	унарный минус	унарный
--	декремент	унарный
-=	вычитание с присв.	бинарный

	побитовое ИЛИ	бинарный
=	побит. ИЛИ с присв.	бинарный
	логическое ИЛИ	бинарный
~	побитовое НЕ	унарный
delete	удаление из памяти	-
new	размещ. в памяти	-

Существуют следующие ограничения на перегрузку операторов в C++.

- 1) Допускается перегружать только существующие операторы. Нельзя определить новый оператор (например, «**»).
- 2) Запрещается перегружать операторы для встроенных типов данных (int, float, ...).
- 3) Перегруженные операции подчиняются стандартным правилам старшинства при использовании в составе сложных выражений.
- 4) Если некоторая операция имеет две формы – унарную и бинарную (&, *, +, -), то каждая из этих форм может быть перегружена отдельно.
- 5) Аргументы операторной функции не могут иметь значений по умолчанию.
- 6) Перегруженные операторы наследуются производными классами (кроме оператора присваивания operator=).

В качестве примера рассмотрим класс комплексного числа, который содержит поля для хранения вещественной и мнимой части (re и im), конструктор для инициализации полей, и функцию вывода комплексного числа на экран

```
class complex
{
public:
    double re, im;
    complex(double r = 0, double i = 0) : re(r), im(i) {};
    void display() { cout << re << ", " << im << endl; }
};
```

Перегрузим операцию сложения двух комплексных чисел с помощью глобальной функции operator+. Так как в данном случае перегружается бинарный оператор, функция будет принимать в качестве аргументов два объекта класса complex. Результатом сложения также является комплексное число, поэтому возвращаемым значением будет объект класса complex

```
complex operator+(complex z1, complex z2)
{
    complex z3;
    z3.re = z1.re + z2.re;
    z3.im = z1.im + z2.im;
    return z3;
}
```

Обратим внимание на то, что внутри функции создается локальный объект с именем `z3`, полям которого присваиваются значения, вычисленные в соответствии с правилами комплексной арифметики. Этот объект затем возвращается в вызывающую программу. Можно предложить более лаконичный вариант записи той же самой операторной функции, где новый объект создается «на лету», а после инициализации полей сразу возвращается в вызывающую программу.

```
complex operator+(complex z1, complex z2)
{
    return complex(z1.re + z2.re, z1.im + z2.im);
}
```

Рассмотрим теперь пример использования разработанной операторной функции (в любой из записанных выше форм).

```
int main()
{
    complex a(1.2, 3.4);
    complex b(5.6, 7.8);
    complex c;
    c = a + b;
    c.display();
    system("pause");
}
```

С точки зрения компилятора, оператор `c = a + b` в приведенном выше фрагменте кода эквивалентен вызову операторной функции `c = operator+(a, b)`. Полям `re` и `im` объекта `c` в итоге будут присвоены значения 6.8 и 11.2, то есть выполнено сложение комплексных чисел.

5.3) Функции – друзья класса.

В рассмотренном выше примере поля `re` и `im` класса `complex` объявлены в его открытой (`public`) части, что, вообще говоря, нарушает инкапсуляцию данных. Однако такой способ объявления был выбран не случайно. Если бы эти поля были объявлены закрытыми (`private`), то прямое обращение к ним в функции `operator+` (в конструкциях вида `z1.re + z2.re`) было бы невозможным. Одним из выходов в данной ситуации может стать использование функций-друзей (`friend function`).

По определению, дружественная к некоторому классу функция имеет прямой доступ к его закрытым членам – полям и методам. Такие функции позволяют, с одной стороны, обеспечить сохранность данных объекта, с другой стороны, эффективно реализовать доступ к его содержимому. В некоторых случаях это решение является единственным.

Для того, что функция стала дружественной к заданному классу, она должна быть объявлена внутри этого класса с ключевым словом `friend`. Для класса комплексных чисел такое объявление может иметь вид

```
class complex
{
    private:
        double re, im;
    public:
        complex(double r = 0, double i = 0) : re(r), im(i) {};
        void display() { cout << re << ", " << im << endl; }

        friend complex operator+(complex z1, complex z2);
};
```

Несмотря на то, что данное объявление похоже на объявление метода, `friend`-функция не является методом класса, и остается глобальной (внешней по отношению к классу) функцией. Реализация этой функции внешне ничем не отличается от рассмотренного нами ранее варианта

```
complex operator+(complex z1, complex z2)
{
    return complex(z1.re + z2.re, z1.im + z2.im);
}
```

Однако легко заметить существенное «внутреннее» различие – в последнем случае мы манипулируем значениями закрытых полей данных внутри операторной функции.

5.4) Операторная функция как метод класса.

Операторная функция может быть объявлена не только как глобальная и дружественная, но и как метод класса. В этом случае она объявляется в открытой (`public`) части класса. Ключевой особенностью такого вида объявления является то, что число параметров операторной функции будет на единицу меньше, чем арность оператора (то есть, для бинарного оператора требуется один аргумент, для унарного оператора – аргументы отсутствуют). Причина «уменьшения» числа аргументов проста – раз мы имеем дело с методом класса, один из параметров передается неявно, через указатель `this`.

Перегрузим операцию сложения двух комплексных чисел, используя теперь не глобальную функцию, а метод класса `complex`

```
class complex
{
    private:
        double re, im;
    public:
```

```
complex(double r = 0, double i = 0) : re(r), im(i) {};  
void display() { cout << re << ", " << im << endl; }
```

```
complex operator+(const complex z);  
};
```

Реализация операторного метода в данном случае будет иметь вид

```
complex complex::operator+(const complex z)  
{  
    return complex(re + z.re, im + z.im);  
}
```

Использовать эту функцию можно точно так же, как и описанную ранее глобальную дружественную функцию. Приведенный ниже пример внешне ничем не отличается от примера из п. 5.2.

```
int main()  
{  
    complex a(1.2, 3.4);  
    complex b(5.6, 7.8);  
    complex c;  
    c = a + b;  
    c.display();  
    system("pause");  
}
```

Однако в данном случае фрагмент кода `c = a + b` будет означать вызов операторного метода вида `c = a.operator+(b)`.

5.5) Пример приложения Visual C++.

Вернемся теперь к нашему программному проекту, использующему классы `book` и `bookstore`. Реализуем некоторые из используемых нами функций как операторные. А именно, для класса `book` реализуем

- 1) операцию проверки двух книг на эквивалентность (`operator==`)
- 2) операцию чтения данных из потока `istream` (`operator>>`)
- 3) операцию записи данных в поток `ostream` (`operator<<`)
- 4) операцию чтения данных из файла `ifstream` (`operator>>`)
- 5) операцию записи данных в файл `ofstream` (`operator<<`)

===== файл `books.h` =====

```
#ifndef BOOKS_H  
#define BOOKS_H  
#include <string>  
#include <iostream>  
#include <fstream>
```

```

using namespace std;
class book
{
private:
    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;
public:
    book() : title(""), authors(""), publisher(""), year(0), pages(0) {};
    book(string, string, string, int, unsigned int);
    string get_title();

    bool operator ==(book another);
    friend ostream& operator<<(ostream& stream, const book& abook);
    friend istream& operator>>(istream& stream, book& abook);
    friend ofstream& operator<<(ofstream& stream, const book& abook);
    friend ifstream& operator>>(ifstream& stream, book& abook);
};
#endif
===== файл books.cpp =====
#include "stdafx.h"
#include "books.h"
using namespace std;
book::book(string ttl, string aut, string pbl, int yer, unsigned int pag)
{
    title = ttl;
    authors = aut;
    publisher = pbl;
    year = yer;
    pages = pag;
}
string book::get_title()
{
    return title;
}
bool book::operator ==(book another)
{
    if (title != another.title) return false;
    if (authors != another.authors) return false;
    if (publisher != another.publisher) return false;
    if (year != another.year) return false;
}

```

```

        if (pages != another.pages) return false;
        return true;
    }
ostream& operator<<(ostream& stream, const book& abook)
{
    stream << "Название: " << abook.title << endl;
    stream << "Автор(ы): " << abook.authors << endl;
    stream << "Издатель: " << abook.publisher << endl;
    stream << "Год:      " << abook.year << endl;
    stream << "Страниц: " << abook.pages << endl;
    return stream;
}
istream& operator>>(istream& stream, book& abook)
{
    getline(stream, abook.title);
    getline(stream, abook.authors);
    getline(stream, abook.publisher);
    stream >> abook.year;
    stream >> abook.pages;
    stream.get();
    return stream;
}
ofstream& operator<<(ofstream& stream, const book& abook)
{
    stream << abook.title << endl;
    stream << abook.authors << endl;
    stream << abook.publisher << endl;
    stream << abook.year << endl;
    stream << abook.pages << endl;
    return stream;
}
ifstream& operator>>(ifstream& stream, book& abook)
{
    getline(stream, abook.title);
    getline(stream, abook.authors);
    getline(stream, abook.publisher);
    stream >> abook.year;
    stream >> abook.pages;
    stream.get();
    return stream;
}

```

Для класса `bookstore`, в свою очередь, реализуем

- 1) операторную функцию `+=` вместо метода `add_book`

- 2) операторную функцию -= для удаления книги из списка
- 3) функцию записи в поток содержимого всего класса-контейнера

===== файл bookstore.h =====

```
#ifndef BOOKSTORE_H
#define BOOKSTORE_H
#include <string>
#include <fstream>
#include "books.h"
class bookstore
{
private:
    int max_num_books;
    int num_books;
    book *books;
public:
    bookstore(unsigned int max_nb);
    ~bookstore();
    void operator +=(const book &abook);
    void operator -=(const book &abook);
    friend ostream& operator<<(ostream& stream, const bookstore& astore);
    void read_from_file(string filename);
    void write_to_file(string filename);
    void display_all();
    void find_book(string atitle);
};
#endif
```

===== файл bookstore.cpp =====

```
#include "stdafx.h"
#include <iostream>
#include "bookstore.h"
using namespace std;
bookstore::bookstore(unsigned int max_nb)
{
    max_num_books = max_nb;
    books = new book[max_num_books];
    num_books = 0;
    cout << "\nВызван конструктор класса bookstore:";
    cout << "\n    выделено объектов - " << max_num_books;
    cout << "\n    загружено книг - " << num_books << endl;
}
bookstore::~bookstore()
{

```

```

    max_num_books = 0;
    delete[] books;
    num_books = 0;
    cout << "\nВызван деструктор класса bookstore:";
    cout << "\n    выделенная память освобождена";
}

void bookstore::operator+=(const book &abook)
{
    if (num_books < max_num_books)
    {
        books[num_books] = abook;
        num_books++;
    }
}

void bookstore::operator-=(const book &abook)
{
    if (num_books < 0)
        return;
    int index = -1;
    for (int i = 0; i < num_books; i++)
        if (books[i] == abook)
            index = i;
    if (index == -1)
        return;
    if (index != num_books - 1)
        books[index] = books[num_books - 1];
    num_books--;
}

ostream& operator<<(ostream& stream, const bookstore& astore)
{
    stream << "\n ВСЕЬ АССОРТИМЕНТ МАГАЗИНА:\n";
    for (int i = 0; i < astore.num_books; i++)
    {
        stream << "===== книга " << i+1 << " ===== \n";
        stream << astore.books[i] << endl;
    }
    return stream;
}

void bookstore::read_from_file(string filename)
{
    ifstream infile;
    infile.open(filename);
    if (!infile.is_open())

```

```

{
    cout << "\n\nФайл данных не найден!\n";
    system("pause");
    exit(1);
}
int N;
infile >> N;
infile.get();
for (int i = 0; i < N; i++)
{
    book new_book;
    infile >> new_book;
    this->operator+=(new_book);
}
infile.close();
cout << "\nЗагружены данные из файла " << filename << ":";
cout << "\n    число загруженных книг - " << num_books << endl;
}
void bookstore::write_to_file(string filename)
{
    ofstream outfile;
    outfile.open(filename);
    outfile << num_books << endl;
    for (int i = 0; i < num_books; i++)
        outfile << books[i];
    outfile.close();
    cout << "\nДанные записаны в файл " << filename << ":";
    cout << "\n    число записанных книг - " << num_books;
}
void bookstore::find_book(string atitle)
{
    cout << "\n\nИщем книгу с названием \"" << atitle << "\"";
    bool found = false;
    for (int i = 0; i < num_books; i++)
    {
        if (books[i].get_title() == atitle)
        {
            found = true;
            cout << "\nНайдена книга:\n";
            cout << books[i];
        }
    }
    if (!found)

```

```

        cout << "\nКнига с таким названием не найдена!\n";
    }

===== основной модуль - lab5.cpp =====

#include "stdafx.h"
#include <string>
#include "bookstore.h"
#include "books.h"
void main()
{
    setlocale(LC_ALL, "rus");

    bookstore my_store(25);
    my_store.read_from_file("my_books.txt");
    book book1("Дети капитана Гранта", "Верн Ж.", "Эксмо", 2003, 800);
    my_store += book1;
    cout << my_store;
    my_store.find_book("Война и мир");
    system("pause");
}

=====

```

Задание на лабораторную работу

- 1) Для своего варианта задания реализуйте операторные функции:
 - проверки двух объектов основного класса на эквивалентность (operator==),
 - вывода информации в поток ostream (operator<<),
 - ввода данных из потока istream (operator>>),
 - записи в файловый поток ofstream (operator<<),
 - чтения из файлового потока ifstream (operator>>),
 - добавления нового элемента в класс-контейнер (operator+=).
- 2) Продемонстрируйте работу операторных функций.

Содержание отчета по лабораторной работе

- 1) Стандартная «шапка» отчета
- 2) Цель: формулировка цели работы
- 3) Теория: краткие сведения о перегрузке операций с примерами (объем 2-3 стр.)
- 4) Обновленная диаграмма классов.
- 5) Программный код
 - объявление основного класса и класса-контейнера (с операторными функциями);

- реализацию одной операторной функции как глобальной, другой – как метода класса.

Контрольные вопросы

- 1) Что такое операторные функции? Как создать операторную функцию?
- 2) Почему у функции `ostream& operator<<` одним из аргументов является константный параметр, передаваемый по ссылке?
- 3) Добавьте аргументы по умолчанию в конструктор класса `bookstore::bookstore`.
- 4) Пусть функция f имеет параметр с именем x , который изменяется в ее теле. Когда параметр x следует передавать по значению? Когда его следует передавать по ссылке? Всегда ли он может быть параметром, передаваемым по значению?
- 5) Для какой цели необходимы дружественные функции? Какие функции могут быть дружественными? Являются ли дружественные функции членами класса?
- 6) В каких случаях программисту следует использовать дружественные функции и когда лучше воздержаться от их использования?
- 7) Какие бинарные и унарные операторы можно перегружать?
- 8) Можно ли перегрузить тернарную операцию?
- 9) Перегрузите оператор `<` (меньше) для класса `complex` `{private double re, im; public ...};`
- 10) Перегрузите оператор `=` (присваивание) для класса `complex` `{private double re, im; public ...};`

ЛАБОРАТОРНАЯ РАБОТА №6

Наследование

Цель работы: Изучение языковых конструкций C++, реализующих механизм наследования классов; приобретение навыков разработки и использования производных классов.

Теоретические сведения.

6.1) Отношение обобщения.

В лабораторной работе №4 нами была рассмотрена композиция как способ создания новых классов на основе уже существующих. Однако композиция (и агрегация) не являются единственным механизмом взаимодействия классов между собой. Другим активно используемым в ООП методом является наследование классов (class inheritance).

Известно, что композиция основана на отношении вида «целое-часть» (еще говорят об отношении «имеет...», «содержит...» или «has a...»). В отличие от композиции, при наследовании классы находятся между собой в отношении *обобщения*, которое выражают словами «является...» или «is a...». Например, класс Животные и класс Млекопитающие находятся между собой в отношении обобщения, так как любое млекопитающее является животным, то есть понятие животного является более общим, чем понятие млекопитающего. Аналогичное отношение можно установить между классами Млекопитающие и Парнокопытные, классами Парнокопытные и Лошади и т.д. Отношения обобщения позволяют строить сложные многоуровневые иерархии понятий.

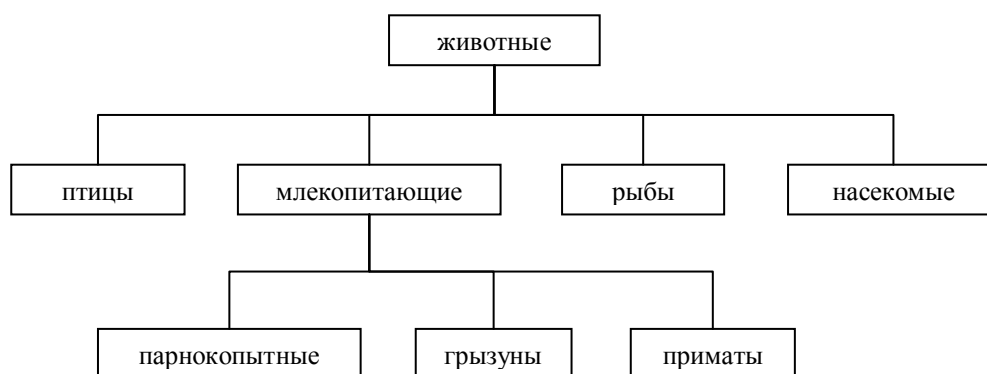


Рисунок 2 – Схема наследования характеристик

Рассмотрим теперь чуть подробнее, почему такого рода отношения в ООП реализуются через механизм наследования. Возьмем пару Млекопитающие/Парнокопытные, и рассмотрим отношения между соответствующими понятиями в природе. Действительно, любое животное отряда парнокопытных (лошадь, корова, жираф, бегемот и др.) обладает набором свойств, присущих всем млекопитающим (определенное строение тела, вскармливание де-

тенышей молоком и т.д.). Можно говорить о том, что в природе парнокопытные *наследуют* часть своих свойств и поведения от млекопитающих.

Эти свойства и поведение являются общими для всех млекопитающих. С другой стороны, парнокопытные имеют характеристики, отличающие их от других млекопитающих, например, грызунов или приматов (см. схему).

Аналогичные отношения реализуются между программными классами в языке C++. Введем термины, используемые для описания наследственных отношений. Будем называть *базовым* класс, от которого производится наследование (в паре млекопитающие/парнокопытные – это класс млекопитающих). Иногда его называют родительским классом или классом-предком. Будем называть *производным* класс, образованный в результате наследования от родительского класса (в паре млекопитающие/парнокопытные – класс парнокопытных). Его также называют классом-наследником, дочерним классом или классом-потомком. Будем называть *иерархией наследования* все отношения между родительским классом и его потомками.

Идея наследования связана с тем, что образованный в результате новый класс получает от родителя (наследует) его свойства и функциональность. При этом:

- класс-наследник имеет доступ к публичным и защищенным методам и полям класса родительского класса;
- класс-наследник может добавлять свои данные и методы, а также переопределять методы базового класса.

6.2) Объявление производного класса.

Рассмотрим синтаксис объявления производного класса

```
class имя_производного_класса : ключ_доступа имя_базового_класса
{
    // поля и методы производного класса
};
```

В качестве ключа доступа здесь может использоваться одно из ключевых слов `public`, `private` и `protected`, которые реализуют различные виды наследования – открытое, закрытое и защищенное, соответственно. Вид наследования (`public`, `private` и `protected`) в данном случае определяет доступность полей и методов базового класса для объектов производного класса.

При наследовании класс-потомок получает в свое распоряжение все поля и методы, имеющиеся в родительском классе. Однако он может также расширить класс родителя, добавив в него *новые* поля и/или методы. Эти дополнительные элементы объявляются в теле производного класса.

Рассмотрим в качестве примера приложение «Книжный магазин», которое разрабатывается нами на протяжении всего лабораторного практикума. Мы имеем класс `book` для описания отдельной книги из книжного магазина, который содержит поля для хранения названия книги (`title`), ее авторов (`authors`), издательства (`publisher`), года выпуска (`year`) и количества страниц (`pages`). У нас также имеется класс `bookstore`, хранящий данные обо всех книгах в книжном магазине.

Предположим теперь, что ассортимент книжного магазина составляют не только оригинальные книги, написанные на русском языке, но и переводные издания. И пусть информация о переводной книге, среди прочего, содержит указание языка оригинала (английский, немецкий, японский и т.д.) и фамилию переводчика.

Разработаем новый класс переводной книги – `translated_book`. В данном случае между классами `book` и `translated_book` имеет место отношение обобщения – переводная книга является частным случаем обыкновенной (непереводной) книги. Поэтому класс `translated_book` будет реализован нами как наследник класса `book`

```
class translated_book : public book
{
    private:
        string language;
        string translator;

    public:
        ...
};
```

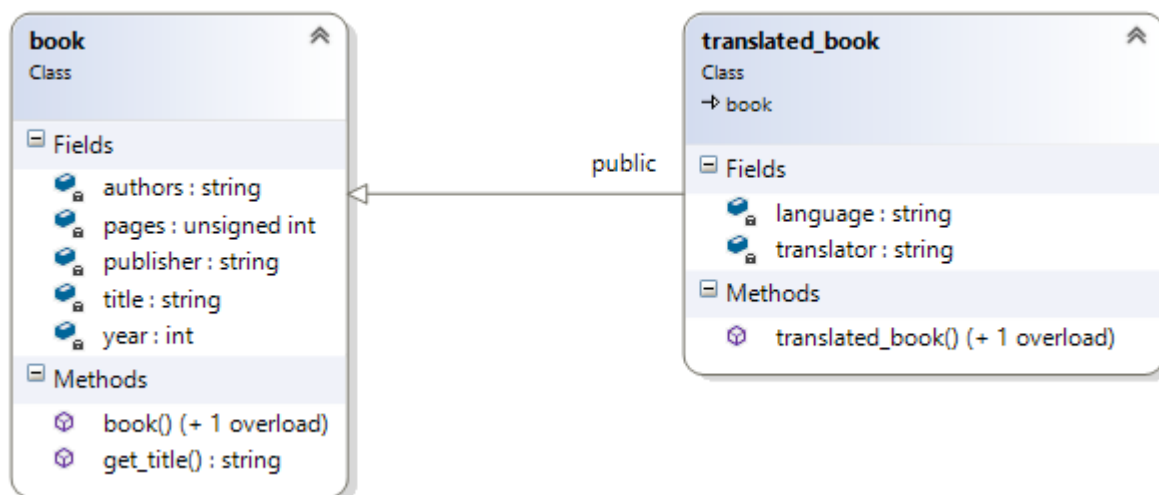


Рисунок 3 – Диаграмма, изображающая наследование классов

Обратим внимание на то, что в данном случае мы использовали открытое наследование (ключ доступа `public`). Это означает, что поля базового класса (`title`, `authors` и т.д.) будут иметь в производном классе тот же уровень доступа, что и в классе-родителе. В производном

классе мы объявили два дополнительных текстовых поля – это закрытые поля `language` (язык оригинала) и `translator` (переводчик). На UML-диаграмме классов наследование изображается стрелкой от производного класса к базовому (см. Рисунок 3). Над стрелкой указывается вид наследования.

6.3) Создание объекта производного класса. Доступ к элементам.

Создание объекта производного класса, то есть размещение экземпляра этого класса в компьютерной памяти, выполняется стандартным способом. Для этого необходимо указать имя производного класса и имя объекта, а также при необходимости указать параметры конструктора в круглых скобках.

```
translated_book abook(параметры_конструктора);
```

В случае, когда в классе определен конструктор по умолчанию, круглые скобки и параметры конструктора могут не указываться.

Для динамического размещения объекта используют оператор `new`

```
translated_book *pointer = new translated_book(параметры_конструктора);
```

Освобождение памяти в таком случае производится оператором `delete`

```
delete pointer;
```

Доступ к полям и методам производного класса осуществляется также обычным способом – с использованием оператора «точка» для статических объектов и оператора «стрелка» для указателей.

```
имя_объекта.поле_производного_класса = ...;
имя_объекта.метод_производного_класса(параметры);
указатель->поле_производного_класса = ...;
указатель->метод_производного_класса(параметры);
```

Здесь `имя_объекта` и `указатель` – имя и указатель на объект производного класса. Во всех указанных случаях применимы стандартные ограничения доступа: 1) закрытые элементы недоступны извне и из потомков, но доступны внутри класса; 2) защищенные элементы недоступны извне, но доступны из потомков и внутри класса; 3) открытые элементы доступны отовсюду.

Доступ к унаследованным элементам (т.е. полям и методам базового класса) осуществляется аналогичным образом

```
имя_объекта.поле_базового_класса = ...;
имя_объекта.метод_базового_класса(параметры);
указатель->поле_базового_класса = ...;
указатель->метод_базового_класса(параметры);
```

Следующий фрагмент кода демонстрирует инициализацию полей класса `translated_book`. Обратим внимание на то, что работа с собственными полями (`language` и `translator`) здесь ничем не отличается от работы с унаследованными полями (`title`, `authors` и т.д.).

```

translated_book book("", "", "", 0, 0, "", "");
book.title = "Эффективное использование C++";
book.authors = "Мейерс С.";
book.publisher = "ДМК Пресс";
book.year = 2014;
book.pages = 300;
book.language = "английский";
book.translator = "Мухин Н.А.";

```

Приведенный пример кода является корректным только для случая, если поля базового и производного классов объявлены открытыми (public). В противном случае попытка изменить значения любого поля будет приводить к ошибке времени компиляции.

6.4) Конструктор производного класса.

Объект производного класса можно рассматривать как объект в объекте: его «внутреннее ядро» составляет экземпляр базового класса, «внешнюю оболочку» – собственные поля и методы (см. рис. справа). Инициализация такого сложного объекта должна производиться в два этапа: сначала инициализируется ядро, затем – оболочка. При этом инициализацию ядра обеспечивает конструктор базового класса.

Рассмотрим возможную реализацию конструктора по умолчанию для класса `translated_book`

```

translated_book::translated_book() : book(), language(""), translator("")
{};

```

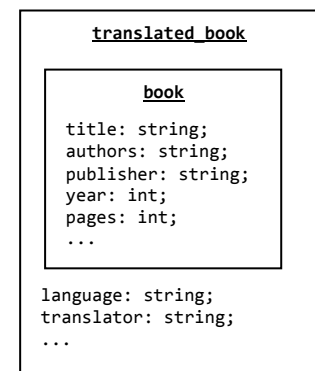
Обратим внимание на список инициализации этого конструктора: здесь сначала явным образом вызывается конструктор базового класса `book`, который «обнуляет» внутренние поля, а затем «обнуляются» собственные поля класса – `language` и `translator`. В результате получаем «пустой» объект, готовый к дальнейшему использованию.

Теперь рассмотрим конструктор переводной книги с аргументами. В качестве параметров мы будем передавать конструктору название книги, авторов, издательство, год выпуска, число страниц, язык оригинала и имя переводчика.

```

translated_book::translated_book(string ttl, string aut, string pbl, int yer,
                                unsigned int pag, string lang, string tran) :
    book(ttl, aut, pbl, yer, pag), language(lang), translator(tran)
{
    cout << "\nВызван конструктор класса translated_book";
}

```



Как видно из приведенного примера, часть параметров конструктора производного класса передается напрямую конструктору базового класса. Другая часть параметров используется для инициализации собственных полей класса `translated_book`.

6.5) Переопределение методов базового класса в производном классе.

Любая функция базового класса может быть *переопределена* в производном классе. Переопределение в данном случае состоит в том, что мы объявляем в производном классе функцию с такой же сигнатурой, как и в базовом классе. В результате она «заслоняет» собой функцию базового класса, то есть реализует свой вариант метода, специфичный для производного класса.

Рассмотрим следующий пример. Пусть метод `print()` класса `book` выводит на экран информацию об оригинальной (непереводной) книге, то есть значения полей `title`, `authors`, `publisher`, `year` и `pages`. При открытом наследовании этот метод будет доступен и для класса-потомка `translated_book`. Однако в последнем случае функциональность метода недостаточна, так как он не выводит на экран значения дополнительных полей переводной книги: `language` и `translator`. Выход в данной ситуации может быть следующим: мы переопределяем метод `print()` для производного класса `translated_book` таким образом, чтобы он выводил на экран значения полей базового класса и собственных полей.

В качестве иллюстрации переопределим оператор вывода объекта `translated_book` в выходной поток

```
ostream& operator<<(ostream& stream, const translated_book& tbook)
{
    stream << static_cast<book>(tbook);
    stream << "Оригинал: " << tbook.language << endl;
    stream << "Перевод : " << tbook.translator << endl;
    return stream;
}
```

Здесь мы сначала вызываем оператор `<<` базового класса `book`, который выводит на экран значения «внутренних» полей. Следом за этим на экран выводятся собственные поля производного класса. Функция `static_cast` здесь служит для преобразования типа `translated_book` в тип `book` и позволяет вызвать метод базового класса явным образом.

6.6) Пример приложения Visual C++.

Расширим программный проект «Книжный магазин», добавив в ассортимент магазина переводные книги, изначально написанные на иностранных языках. Разработаем класс переводной книги `translated_book` как производный от имеющегося класса `book`. Для класса `trans-`

lated_book переопределим операторы потокового ввода/вывода, операцию проверки на эквивалентность.

```
===== файл books.h =====

#ifndef BOOKS_H
#define BOOKS_H
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
class book
{
private:
    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;
public:
    book() : title(""), authors(""), publisher(""), year(0), pages(0) {};
    book(string, string, string, int, unsigned int);
    string get_title();
    friend bool operator ==(book book1, book book2);
    friend ostream& operator<<(ostream& stream, const book& abook);
    friend istream& operator>>(istream& stream, book& abook);
    friend ofstream& operator<<(ofstream& stream, const book& abook);
    friend ifstream& operator>>(ifstream& stream, book& abook);
};

class translated_book : public book
{
private:
    string language;
    string translator;
public:
    translated_book() : book(), language(""), translator("") {};
    translated_book(string, string, string, int, unsigned int, string, string);
    friend bool operator ==(translated_book book1, translated_book book2);
    friend ostream& operator<<(ostream& stream, const translated_book& abook);
    friend istream& operator>>(istream& stream, translated_book& abook);
    friend ofstream& operator<<(ofstream& stream, const translated_book& abook);
    friend ifstream& operator>>(ifstream& stream, translated_book& abook);
};

#endif

===== файл books.cpp =====
```

```

#include "stdafx.h"
#include "books.h"

using namespace std;

// Методы базового класса book

book::book(string ttl, string aut, string pbl, int yer, unsigned int pag)
{
    title = ttl;
    authors = aut;
    publisher = pbl;
    year = yer;
    pages = pag;
    cout << "\nВызван конструктор класса book";
}

string book::get_title()
{
    return title;
}

bool operator ==(book book1, book book2)
{
    if (book1.title != book2.title) return false;
    if (book1.authors != book2.authors) return false;
    if (book1.publisher != book2.publisher) return false;
    if (book1.year != book2.year) return false;
    if (book1.pages != book2.pages) return false;
    return true;
}

ostream& operator<<(ostream& stream, const book& abook)
{
    stream << "Название: " << abook.title << endl;
    stream << "Автор(ы): " << abook.authors << endl;
    stream << "Издатель: " << abook.publisher << endl;
    stream << "Год      : " << abook.year << endl;
    stream << "Страниц : " << abook.pages << endl;
    return stream;
}

istream& operator>>(istream& stream, book& abook)
{
    getline(stream, abook.title);
    getline(stream, abook.authors);
    getline(stream, abook.publisher);
}

```

```

        stream >> abook.year;
        stream >> abook.pages;
        stream.get();
        return stream;
    }
ofstream& operator<<(ofstream& stream, const book& abook)
{
    stream << abook.title << endl;
    stream << abook.authors << endl;
    stream << abook.publisher << endl;
    stream << abook.year << endl;
    stream << abook.pages << endl;
    return stream;
}
ifstream& operator>>(ifstream& stream, book& abook)
{
    getline(stream, abook.title);
    getline(stream, abook.authors);
    getline(stream, abook.publisher);
    stream >> abook.year;
    stream >> abook.pages;
    stream.get();
    return stream;
}
// Методы производного класса translated_book
translated_book::translated_book(string ttl, string aut, string pbl, int yer, unsigned int
pag, string lang, string tran) :
book(ttl, aut, pbl, yer, pag), language(lang), translator(tran)
{
    cout << "\nВызван конструктор класса translated_book";
}
bool operator ==(translated_book book1, translated_book book2)
{
    bool are_equal_books = static_cast<book>(book1) == static_cast<book>(book2);
    bool are_equal_langs = book1.language == book2.language;
    bool are_equal_trans = book1.translator == book2.translator;
    if (are_equal_books && are_equal_langs && are_equal_trans)
        return true;
    return false;
}
ostream& operator<<(ostream& stream, const translated_book& tbook)
{
    stream << static_cast<book>(tbook);

```

```

        stream << "Оригинал: " << tbook.language << endl;
        stream << "Перевод : " << tbook.translator << endl;
        return stream;
    }
    istream& operator>>(istream& stream, translated_book& tbook)
    {
        stream >> static_cast<book>(tbook);
        getline(stream, tbook.language);
        getline(stream, tbook.translator);
        return stream;
    }
    ostream& operator<<(ostream& stream, const translated_book& tbook)
    {
        stream << static_cast<book>(tbook) << endl;
        stream << tbook.language << endl;
        stream << tbook.translator << endl;
        return stream;
    }
    ifstream& operator>>(ifstream& stream, translated_book& tbook)
    {
        stream >> static_cast<book>(tbook);
        getline(stream, tbook.language);
        getline(stream, tbook.translator);
        return stream;
    }
}

```

Расширим также класс-контейнер `bookstore`, определив в нем два независимых массива: один для оригинальных русскоязычных книг (объектов класса `book`), другой – для переводных книг (объектов класса `translated_book`). Функции файлового ввода/вывода изменятся соответствующим образом (см. код ниже).

===== файл bookstore.h =====

```

#ifndef BOOKSTORE_H
#define BOOKSTORE_H
#include <string>
#include <fstream>
#include "books.h"
class bookstore
{
private:
    int max_num_obook;
    int max_num_tbooks;
    int num_obook;
    int num_tbooks;

```

```

    book *obooks;                // массив оригинальных (непереводных) книг
    translated_book *tbooks;     // массив переводных книг
public:
    bookstore(int max_ob, int max_tb);
    ~bookstore();
    void operator +=(const book &obook);
    void operator +=(const translated_book &tbook);
    friend ostream& operator<<(ostream& stream, const bookstore& astore);
    void read_obooks_from_file(string filename);
    void read_tbooks_from_file(string filename);
    void write_obooks_to_file(string filename);
    void write_tbooks_to_file(string filename);
    void display_all();
    void find_book(string atitle);
};
#endif

===== файл bookstore.cpp =====

#include "stdafx.h"
#include <iostream>
#include "bookstore.h"
using namespace std;
bookstore::bookstore(int max_ob, int max_tb)
{
    max_num_obooks = max_ob;
    max_num_tbooks = max_tb;
    obooks = new book[max_num_obooks];
    tbooks = new translated_book[max_num_tbooks];
    num_obooks = 0;
    num_tbooks = 0;
    cout << "\nВызван конструктор класса bookstore:";
    cout << "\n    выделено объектов (для оригинальных изданий) - " << max_num_obooks;
    cout << "\n    выделено объектов (для переводных изданий) - " << max_num_tbooks;
    cout << "\n    загружено книг (оригинальных) - " << num_obooks;
    cout << "\n    загружено книг (перводных) - " << num_tbooks;
    cout << endl;
}
bookstore::~bookstore()
{
    cout << "\nВызван деструктор класса bookstore:";
    max_num_obooks = 0;
    delete[] obooks;
    num_obooks = 0;
    max_num_tbooks = 0;
}

```



```

        delete[] tbooks;
        num_tbooks = 0;
        cout << "\n    выделенная память освобождена";
    }
    void bookstore::operator+=(const book &obook)
    {
        if (num_obooks < max_num_obooks)
        {
            obooks[num_obooks] = obook;
            num_obooks++;
        }
    }
    void bookstore::operator+=(const translated_book &tbook)
    {
        if (num_tbooks < max_num_tbooks)
        {
            tbooks[num_tbooks] = tbook;
            num_tbooks++;
        }
    }
    ostream& operator<<(ostream& stream, const bookstore& astore)
    {
        stream << "\n\n    ВЕСЬ АССОРТИМЕНТ КНИЖНОГО МАГАЗИНА:\n";
        stream << "\n    А) Оригинальные издания\n";
        for (int i = 0; i < astore.num_obooks; i++)
        {
            stream << "-----\n";
            stream << astore.obooks[i] << endl;
        }
        stream << "\n    Б) Переводные издания\n";
        for (int i = 0; i < astore.num_tbooks; i++)
        {
            stream << "-----\n";
            stream << astore.tbooks[i] << endl;
        }
        return stream;
    }
    void bookstore::read_obooks_from_file(string filename)
    {
        ifstream infile;
        infile.open(filename);
        if (!infile.is_open())
        {

```

```

        cout << "\n\nФайл данных не найден!\n";
        system("pause");
        exit(1);
    }
    int N;
    infile >> N;
    infile.get();
    for (int i = 0; i < N; i++)
    {
        book new_book;
        infile >> new_book;
        this->operator+=(new_book);
    }
    infile.close();
    cout << "\nЗагружены оригинальные книги из файла " << filename << ":";
    cout << "\n    число загруженных книг - " << num_obooks << endl;
}

void bookstore::write_obooks_to_file(string filename)
{
    ofstream outfile;
    outfile.open(filename);
    outfile << num_obooks << endl;
    for (int i = 0; i < num_obooks; i++)
        outfile << obooks[i];
    outfile.close();
    cout << "\nДанные записаны в файл " << filename << ":";
    cout << "\n    записано оригинальных книг - " << num_obooks;
}

void bookstore::write_tbooks_to_file(string filename)
{
    ofstream outfile;
    outfile.open(filename);
    outfile << num_tbooks << endl;
    for (int i = 0; i < num_tbooks; i++)
        outfile << tbooks[i];
    outfile.close();
    cout << "\nДанные записаны в файл " << filename << ":";
    cout << "\n    записано переводных книг - " << num_tbooks;
}

void bookstore::find_book(string atitle)
{
    cout << "\n\nИщем книгу с названием \"" << atitle << "\"";

```

```

bool found = false;
for (int i = 0; i < num_obook; i++)
{
    if (obook[i].get_title() == atitle)
    {
        found = true;
        cout << "\nНайдена книга:\n";
        cout << obook[i];
    }
}
for (int i = 0; i < num_tbook; i++)
{
    if (tbook[i].get_title() == atitle)
    {
        found = true;
        cout << "\nНайдена книга:\n";
        cout << tbook[i];
    }
}
if (!found)
    cout << "\nКнига с таким названием не найдена!\n";
}

===== основной модуль - lab6.cpp =====

#include "stdafx.h"
#include <string>
#include "bookstore.h"
#include "books.h"
void main()
{
    setlocale(LC_ALL, "rus");
    bookstore my_store(25, 10);
    book book1("Дети капитана Гранта", "Верн Ж.", "Эксмо", 2003, 800);
    my_store += book1;
    translated_book book2("Эффективное использование C++", "Мейерс С.",
                          "ДМК Пресс", 2014, 300, "английский", "Мухин Н.А.");
    my_store += book2;
    cout << my_store;
    cout << "\n\n";
    system("pause");
}

=====

```

Задание на лабораторную работу

- 1) Разработайте производный класс как наследник базового класса для своего варианта задания. Добавьте в производный класс несколько дополнительных полей. Определите конструктор по умолчанию и конструктор с параметрами для производного класса. Переопределите соответствующие методы в производном классе.
- 2) Реализуйте хранение объектов базового и производного классов в контейнере. Измените методы класса-контейнера соответствующим образом: реализуйте запись и чтение данных из файла, поиск нужного объекта в коллекции и др.
- 3) Пр продемонст рируйте работу механизмов наследования и переопределения функций в своей программе.

Содержание отчета по лабораторной работе

- 1) Стандартная «шапка» отчета
- 2) Цель: формулировка цели работы
- 3) Теория: краткие сведения о реализации наследования с примерами (объем 2-3 стр.)
- 4) Диаграмму классов программы.
- 5) Программный код
 - объявление производного класса,
 - определение конструктора по умолчанию и конструктора с параметрами для производного класса,
 - переопределение одного из методов производного класса.

Контрольные вопросы

- 1) Какое отношение между двумя понятиями называют отношением обобщения? Приведите примеры таких понятий.
- 2) Объясните различия между механизмами агрегации и наследования. В каких ситуациях следует применять тот или иной механизм?
- 3) Синтаксис объявления производного класса. Общее, частное и защищенное наследование.
- 4) Режим доступа к полям и методам класса при открытом наследовании. Поясните на примере.
- 5) Создание объекта производного класса. Конструктор производного класса.
- 6) Как переопределить метод базового класса в производном классе? Приведите пример.

ЛАБОРАТОРНАЯ РАБОТА №7

Виртуальные функции

Цель работы: Знакомство с динамическим полиморфизмом и поздним связыванием в ООП; изучение механизма виртуальных функций C++; приобретение навыков использования виртуальных функций при проектировании и разработке программ.

Теоретические сведения

7.1) Указатели и ссылки на базовые и производные классы.

Прежде, чем приступить к изучению собственно виртуальных функций, рассмотрим вопрос, связанный с наследованием и использованием указателей и ссылок на пользовательские типы данных – классы.

Хорошо известно, что в общем случае указатель одного типа не может указывать на объект другого типа. Например, под указателем типа `int` может размещаться только переменная типа `int`, но не может размещаться переменная типа `float`, `char`, `bool` и т.д. Это правило справедливо чуть реже, чем всегда, так как не распространяется на классы, находящиеся в отношении «предок - потомок». Исключение можно сформулировать так: **в C++ указатель на базовый класс может указывать на объект производного класса.**

Исключение связано с тем, что потомок наследует все свойства и поведение своего предка (поля и методы), то есть потомок фактически *является* своим предком. Мы можем утверждать, например, что парнокопытное является млекопитающим, а млекопитающее, в свою очередь, является животным. Аналогичным образом, в C++ между объектами базового и производного класса существует более тесная связь, чем между неродственными объектами. Связь между производным и базовым классом часто формулируется в виде отношения «есть». Например, непарнокопытное животное – это («есть») травоядное животное, так что имеет смысл реализовать класс `Perissodactyla` (непарнокопытные) в виде класса, производного от базового класса `Herbivore` (травоядные).

Рассмотрим следующий пример. Пусть `book` – базовый класс книги, содержащий сведения о названии книги, ее авторах и т.д., а `electronic_book` – производный от него класс электронной книги, содержащий дополнительно информацию о формате книги (`fb2`, `epub`, `djvu` и др.) и объеме файла в килобайтах. Предположим далее, что в программе имеются следующие объявления

```
book *pointer;           // объявляем указатель на объект класса book
book mybook1;            // создаем объект базового класса
electronic_book mybook2; // создаем объект производного класса
```

Тогда корректными с точки зрения компилятора будут следующие операции

```
pointer = &mybook1;      // здесь указатель pointer устанавливается на объект book
```

```
pointer = &mybook2; // а здесь он указывает на объект electronic_book
```

Правильнее будет считать, что во втором случае `pointer` указывает не на объект всей электронной книги, а только на его внутреннюю часть, унаследованную от `book`. Таким образом, указатель `pointer` может дать нам доступ к полям и методам базового класса, унаследованным всеми его потомками.

Допустим, метод `display()` базового класса `book` выводит на экран значения полей объекта, то есть название книги, ее авторов и год выпуска. Тогда пара операторов

```
pointer = &mybook1;  
pointer->display();
```

выведет на экран информацию об объекте базового класса с именем `mybook1`. Применим теперь аналогичную процедуру к объекту производного класса

```
pointer = &mybook2;  
pointer->display();
```

В этом случае на экран будут выведены сведения об электронной книге, содержащиеся в объекте `mybook2`. Однако выведены будут только унаследованные поля, а дополнительные данные (формат и размер файла в Кб) на экран выводиться не будут, так как в данном случае через `pointer` мы можем вызывать только методы базового класса.

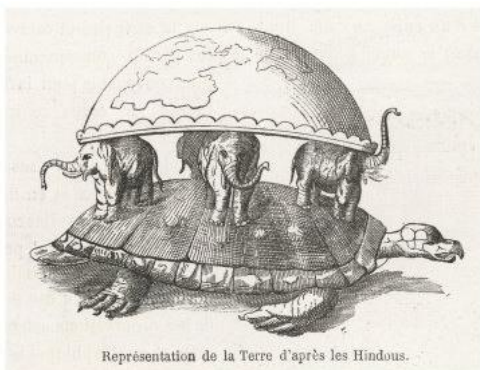
Отметим здесь следующее. Мы рассмотрели ситуацию, когда указатель на базовый класс используется в качестве указателя на производный объект. Обратное действие невозможно, то есть указатель на производный класс не может использоваться для доступа к объектам базового типа.

Рассмотренные особенности использования указателей в C++ применимы также к ссылкам. То есть, ссылки на базовый класс могут также использоваться как ссылки на производный тип. Эта техника нередко применяется при работе с функциями. А именно, если какой-либо параметр функции является ссылкой на базовый класс, то функция может принимать ссылку как на объект базового класса, так и на объекты производных классов.

7.2) Динамический полиморфизм и позднее связывание.

Динамический полиморфизм считается одной из трех ключевых парадигм объектно-

ориентированного программирования, наряду с инкапсуляцией и наследованием. Говоря кратко, динамический полиморфизм – это способность объекта вызывать методы производного класса, который может *даже не существовать* на момент создания базового класса.



Что это означает в реальности? Предположим, что программист разрабатывает класс Животное, который должен выступить в качестве прародителя для производных классов, таких как Млекопитающее, Птица, Земноводное, Рыба, и т.д. Одна из характерных особенностей животного – это его способность передвигаться, поэтому логичным было бы определить специальный метод `move()` для класса Животное. Этот метод будет унаследован производными классами, то есть по умолчанию млекопитающее, птица, рыба и земноводное будут перемещаться в пространстве одинаково. Однако мы знаем, что различные виды животных используют разные способы передвижения – млекопитающие бегают, птицы летают, рыбы плавают и т.д. Для программиста C++ это означает, что метод `move()` должен быть переопределен в производных классах.

Предположим теперь, что в программе имеется указатель `pointer` на объект класса Животное. В соответствии с правилами, рассмотренными в предыдущем пункте, `pointer` может указывать не только на животное, но и на экземпляр любого производного класса, то есть на млекопитающее, птицу, рыбу и т.д. Более того, под указателем `pointer` можно разместить объект нового класса, который даже не был разработан на момент создания класса животного. Вызывая метод `move()` следующим образом

```
pointer->move();
```

программист, вообще говоря, не знает, какой именно способ перемещения здесь будет задействован – бег, полет, плавание и т.д.

Таким образом, на момент редактирования и даже компиляции кода C++ вызов метода `move()` еще не связан с конкретной функцией. Эта связь появляется только на этапе выполнения программы, когда становится известно, какой именно объект располагается под указателем `pointer`. Этот тип связывания называется *поздним*, в отличие от *раннего* связывания, когда выбор конкретной функции происходит на этапе компиляции программы. Позднее связывание дает программисту преимущество – он может выполнять над объектами операции, даже не предполагая о том, как эти операции реализованы (в том числе, будут реализованы в будущем).

Вернемся теперь к приложению «Книжный магазин». Перепроектируем систему классов следующим образом. Пусть `book` – базовый класс книги, а `printed_book` и `electronic_book` – производные классы, описывающие печатную книгу и электронную книгу, соответственно. Каждый из производных классов задает несколько специфических полей: для печатной книги указывается издательство и число страниц, для электронной – формат файла и его объем в килобайтах. Получаем следующую иерархию классов

Как видно из диаграммы, производные классы переопределяют метод `display()` базового класса (вывод на экран содержимого полей), а также методы `read_from_file()` и `write_to_file()` (чтение из файла и запись в файл).

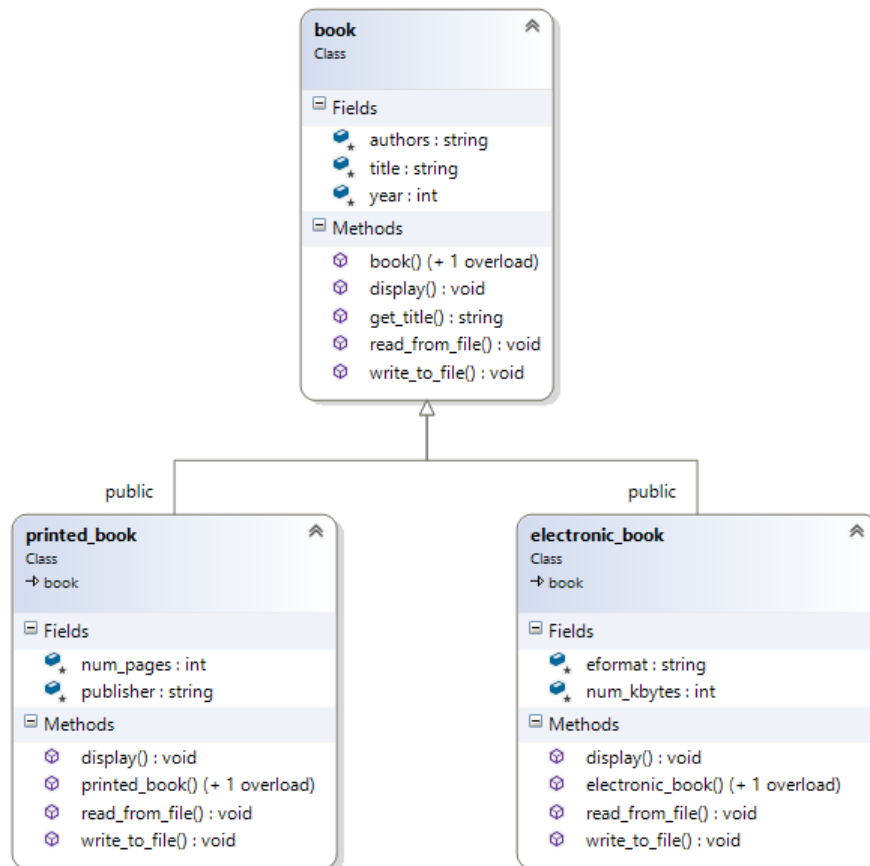


Рисунок 4 – Диаграмма классов, содержащая производные классы

Нашей следующей задачей будет создание единой коллекции книг, включающей разнотипные издания (электронные, печатные, или другие издания, являющиеся наследниками от класса `book`). Объявим массив указателей на объекты базового класса и выделим память под несколько элементов

```

book** p_books;           // массив указателей на объекты book
p_books = new book*[5];   // выделяем память под 5 указателей
  
```

Заполним массив, используя указатели как на объекты `book`, так и на объекты производных классов (см. п. 7.1)

```

p_books[0] = new printed_book("Дети капитана Гранта", "Верн Ж.", 2003, "Эксмо", 800);
p_books[1] = new electronic_book("Эффективный C++", "Мейерс С.", 2014, "djvu", 35);
p_books[2] = new book("Алиса в Зазеркалье", "Кэрролл Л.", 1871);
p_books[3] = new printed_book("Алгебра", "Виноградов И.М.", 1977, "Сов. энц.", 412);
p_books[4] = new electronic_book("Бесы", "Достоевский Ф.М.", 1866, "epub", 21);
  
```

Теперь массив содержит объекты разных классов. Посмотрим далее, сможем ли мы обрабатывать эти разнородные данные, используя общий интерфейс. Например, выведем на экран

содержимое всей коллекции, вызывая последовательно метод `display()` для всех элементов массива

```
for (int i = 0; i < 5; i++)  
    p_books[i]->display();
```

Результаты работы цикла, скорее всего, нас разочаруют: на экран будут выведены только внутренние (унаследованные) поля, то есть название книги (`title`), авторы (`authors`) и год публикации (`year`). Дополнительная информация из производных классов отображена не будет. Фактически вызов функции `display()` будет транслирован в вызов функции `book::display()`. Причиной является то, что к настоящему моменту мы не задействовали механизма позднего связывания.

7.3) Виртуальные функции.

Динамический полиморфизм (полиморфизм времени исполнения) в языке C++ реализуется через виртуальные функции. Виртуальная функция – это функция, объявленная в базовом классе с ключевым словом `virtual`, и затем переопределенная в одном или нескольких производных классах.

Вызов виртуальной функции обрабатывается особым способом. В процессе вызова виртуальной функции через указатель или ссылку на объект базового класса вначале происходит определение фактического типа объекта (к какому из производных классов относится объект), а затем происходит вызов соответствующего варианта функции. В частности, в рассмотренном нами примере для объектов класса `printed_book` будет вызвана функция `printed_book::display()`, а для объектов класса `electronic_book` – функция `electronic_book::display()`.

Общий синтаксис объявления виртуальной функции следующий

```
class имя_класса  
{  
    public:  
        virtual тип имя_функции(параметры);  
};
```

При переопределении виртуальной функции в производном классе ключевое слово `virtual` можно не указывать – компилятор автоматически сделает ее виртуальной.

Задействуем динамический полиморфизм в примере с книжной коллекцией. Для этого объявим метод `display()` базового класса `book` виртуальным

```
class book  
{  
    protected:  
        string title;  
        string authors;
```

```

        int year;
    public:
        ...
        virtual void display();
        ...
};

```

и реализуем этот метод стандартным способом

```

void book::display()
{
    cout << "Название: " << title << endl;
    cout << "Автор(ы): " << authors << endl;
    cout << "Год      : " << year << endl;
}

```

При объявлении производного класса `printed_book` переопределим метод `display()` и также сделаем его виртуальным

```

class printed_book : public book
{
    protected:
        string publisher;
        int num_pages;
    public:
        ...
        virtual void display();
        ...
};

```

Обратите внимание на то, как в приведенном ниже фрагменте кода унаследованная функция `book::display()` задействуется для реализации функции `printed_book::display()`

```

void printed_book::display()
{
    book::display();
    cout << "Издатель: " << publisher << endl;
    cout << "Страниц : " << num_pages << endl;
}

```

Этот код показывает, что для печатной книги мы сначала выводим на экран общую информацию (название, авторы, год), а затем дополнительную (издательство и число страниц). Аналогичным образом может быть переопределена виртуальная функция `display()` для производного класса `printed_book` и других производных классов в системе.

7.4) Фрагменты приложения Visual C++.

Создадим в рамках программного проекта «Книжный магазин» единую коллекцию разнотипных книг, включая печатные и электронные издания. Реализуем единый интерфейс

работы с этими книгами. В частности, используем метод `display()` для вывода информации на экран, методы `read_from_file(ifstream&)` и `write_to_file(ofstream&)` – для чтения и записи в файл. Задействуем класс-контейнер `bookstore` для хранения книжной коллекции.

Ниже приведены фрагменты кода, расположенные в заголовочных файлах модулей `books.h` и `bookstore.h`. Функции классов реализуются стандартным способом, поэтому их код ниже не приводится.

===== файл `books.h` =====

```
#ifndef BOOKS_H
#define BOOKS_H
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
class book
{
protected:
    string title;
    string authors;
    int year;
public:
    book();
    book(string tit, string aut, int yea);
    string get_title();
    virtual void display();
    virtual void read_from_file(ifstream& stream);
    virtual void write_to_file(ofstream& stream);
};
class printed_book : public book
{
protected:
    string publisher;
    int num_pages;
public:
    printed_book();
    printed_book(string tit, string aut, int yea, string pub, int pag);
    virtual void display();
    virtual void read_from_file(ifstream& stream);
    virtual void write_to_file(ofstream& stream);
};
class electronic_book : public book
{
protected:
```

```

    string eformat;
    int num_kbytes;
public:
    electronic_book();
    electronic_book(string tit, string aut, int yea, string eft, int kbs);
    virtual void display();
    virtual void read_from_file(ifstream& stream);
    virtual void write_to_file(ofstream& stream);
};
#endif

===== файл bookstore.h =====

#ifndef BOOKSTORE_H
#define BOOKSTORE_H
#include <string>
#include <fstream>
#include "books.h"
class bookstore
{
private:
    int max_num_books;
    int num_books;
    book **p_books;
public:
    bookstore(int mb);
    ~bookstore();
    void operator +=(book *abook);
    void display();
    void read_from_file(string filename);
    void write_to_file(string filename);
    void find_book(string atitle);
};
#endif
=====

```

Задание на лабораторную работу

- 1) Разработайте систему производных классов (как минимум 2), используя механизмы наследования.
- 2) Реализуйте хранилище объектов различных классов в контейнере через массив указателей базового класса.
- 3) Разработайте общий интерфейс доступа к объектам разного типа в хранилище. Задействуйте для этого механизм виртуальных функций. Реализуйте следующие функции:
 - а) вывода содержимого объекта на экран,

- б) чтение данных из файла,
 - в) запись данных в файл,
 - г) поиск элемента в хранилище,
 - д) свою собственную функцию (по деланию студента).
- 4) Продемонстрируйте работу виртуальных функций в своей программе.

Содержание отчета по лабораторной работе

- 1) Стандартная «шапка» отчета
- 2) Цель: формулировка цели работы
- 3) Теория: краткие сведения о динамическом полиморфизме в C++ и виртуальных функциях с примерами (объем 2-3 стр.)
- 4) Программный код
 - объявление базового класса с несколькими виртуальными функциями (целиком),
 - реализация одной из виртуальных функций базового класса,
 - объявление производного класса с виртуальными функциями (целиком),
 - реализация переопределенной виртуальной функции в производном классе,
 - код основной программы (целиком).

Контрольные вопросы

- 1) Сформулируйте общее правило, касающееся соответствия типов указателя и находящегося под указателем объекта. Какие исключения из этого правила существуют? Приведите примеры общего правила и исключений.
- 2) Перечислите ключевые парадигмы объектно-ориентированного программирования и кратко сформулируйте смысл каждой из них.
- 3) Что означает фраза «переопределить функцию в производном классе»? Придумайте пример такого переопределения и поясните его.
- 4) Синтаксис определения виртуальной функции и ее вызова в языке C++.
- 5) Поясните разницу между поздним и ранним связыванием.
- 6) Объясните, почему для хранения коллекции книг в разработанном нами приложении потребовался двойной указатель `book **p_books` (т.е. указатель на указатель).

ПРИЛОЖЕНИЕ А

Работа с консольными приложениями в Microsoft Visual Studio

Особенностью любого консольного приложения является использование клавиатуры для ввода данных, и текстового монитора – для вывода результатов. В операционной системе Microsoft Windows консольное приложение запускается в отдельном окне с черным фоном. В окне консольного приложения могут отображаться только текстовые символы.

Создание нового консольного приложения

1. Запускаем среду Microsoft Visual Studio. На экране появится стартовая страница – Start Page.
2. Выбираем пункт меню File → New → Project... (Файл → Создать → Проект ...).
3. В открывшемся окне New Project (Новый проект) слева на закладке Templates (Шаблоны) выбираем среду программирования Visual C++, справа – Win32 console application (Консольное приложение Win32).
4. Имя проекта (Name), его расположение на диске (Location), и имя решения (Solution name) задаются в соответствующих полях внизу окна. После выбора имен нажимаем кнопку «ОК».
5. На экране появится окно Win32 Application Wizard (Мастер приложения Win32). Нажимаем кнопку «Finish» («Готово») для завершения.
6. В результате будет создан новый проект консольного приложения. В левой верхней части экрана мы увидим окно редактора, которое используется для ввода кода.
7. Введем код приложения. Для сохранения программы на диске используем соответствующие пункты меню File (Файл).
8. По завершении редактирования программа может быть откомпилирована и запущена. Для этого достаточно нажать клавишу F5. При наличии в программе синтаксических ошибок компилятор сообщит о них в окне Error List в нижней части экрана.

Добавление нового модуля в проект (для многофайловых проектов)

При необходимости существующий проект консольного приложения может быть расширен путем добавления в него дополнительного модуля (или модулей), включающего заголовочный файл *.h и файл реализации *.cpp. Для добавления нового модуля в проект можно использовать Обозреватель решений (Solution Explorer), расположенный в правой части экрана. В окне Обозревателя решений правой кнопкой мыши выбираем Header Files → Add → New Item... (Заголовочные файлы → Добавить → Новый элемент...). В открывшемся окне остается только выбрать тип элемента (.h или .cpp) и имя нового модуля (поле ввода в нижней части окна).

ПРИЛОЖЕНИЕ Б

Директивы препроцессора

Основной функцией директив препроцессора является обработка исходного кода программы до ее компиляции. Для задания нужных действий, каждая из директив препроцессора (`#define`, `#include`, `#undef`, `#if`, `#ifdef`, `#ifndef`, `#else`, `#endif`, `#elif`, `#line`, `#error`, `#pragma`, `#`) помещается на отдельной строке и начинается с символа `#`.

Директива `#include` позволяет включать в код программы текст из выбранного файла. При включении заголовочного файла стандартной библиотеки C++, имя включаемого файла должно быть указано в угловых скобках. Для включения одного из файлов из текущего проекта необходимо использовать двойные кавычки:

```
#include <iostream>
#include "structure.cpp"
```

Использование заголовочных файлов является удобным инструментом при разработке модулей крупного программного продукта, при этом связь между модулями, размещенными в разных файлах, осуществляется через «внешние» объекты, глобальные для всей программы. При этом описания «внешних» или глобальных переменных, массивов, структур, классов помещаются в заголовочных файлах и с помощью директивы `#include` включаются в те модули программы, где они необходимы.

Директива `#define` используется для определения макросов и препроцессорных идентификаторов, каждому из которых ставится в соответствие некоторая символьная последовательность. В тексте же программы препроцессорные идентификаторы заменяются на последовательности символов, предусмотренных директивой `#define`. Макроподстановка – это замена одного текста на другой. Принято писать макросы прописными буквами, чтобы легче его идентифицировать в тексте и отличить от переменных и констант (например, `N`, `TEXT`, `NAME` и т.д.). Директива `#undef` отменяет действие директивы `#define`. Следующий пример демонстрирует синтаксис директив `#define` и `#undef`:

```
#define N 1000
#undef N
```

Директива `#if` и ее модификации `#ifdef` и `#ifndef` совместно с директивами `#else`, `#elif` и `#endif` позволяют организовать условную обработку текста программы (условную компиляцию). В результате компилируется не весь исходный код, а только те или иные фрагменты, которые выделены с помощью этих директив. Условную компиляцию удобно применять при отладке программ для включения/выключения отдельных участков кода.

Рассмотрим следующий пример

```
#include "stdafx.h"
#include <iostream>
```

```

using namespace std;
#define DEBUG
#ifdef DEBUG
    #define OUT_MSG(msg) cout << msg << endl;
#else
    #define OUT_MSG(msg)
#endif
int main()
{
    int bound, i, sum = 0;
    cin >> bound;
    OUT_MSG("Step 1 OK");
    for (i = 0; i < bound; i++) {
        sum += i;
    }
    OUT_MSG("Step 2 OK");
    cout << sum << endl;
    system("pause");
    return 0;
}

```

Если запустить такую программу и ввести значение 10, то на экран будет выведено

```

10
Step 1 OK
Step 2 OK
45

```

Если же удалить строку `#define DEBUG` из кода программы, то будет выведено только

```

10
45

```

Директивы `#ifdef`, `#else`, `#endif` могут также включаться в заголовочные файлы библиотечных модулей, разработанных пользователем (см. примеры в лабораторных работах). В этом случае директивы препроцессора используются для того, чтобы предотвратить многократное включение заголовочного файла в код основной программы. Такая ситуация возможна в многофайловых проектах.

ПРИЛОЖЕНИЕ В

Работа с отладчиком Debug в Microsoft Visual Studio

Независимо от того, насколько детально проработан алгоритм решения задачи, на этапе исполнения программы возможно возникновение непредвиденных ошибок. Эти ошибки могут быть связаны с опечатками при кодировании программы, логическими ошибками при разработке алгоритма, некорректным использованием языковых конструкций и т.д. Если причина неправильной работы программы неизвестна, определить ошибочный оператор (или группу операторов) поможет специальное программное средство – отладчик кода. Отладчик позволяет исполнять разработанную программу небольшими порциями (иногда – пошагово), контролируя при этом ход ее выполнения. Контроль осуществляется за счет (1) отслеживания последовательности исполняемых операторов, (2) отслеживания значений переменных на каждом этапе алгоритма. Появление непредвиденных (некорректных) результатов в процессе отладки в данном случае будет свидетельствовать о наличии ошибки.

Отладчики могут быть внешними (в виде отдельной программы), либо встроенными (включенными) в среду разработки. Среда Microsoft Visual Studio любой версии содержит встроенный отладчик Debug. В приведенной ниже таблице перечислены его основные инструменты. Для запуска пользовательской программы в режиме отладки достаточно войти в меню Debug → Start Debugging, либо нажать функциональную клавишу F5.

Таблица 5 – Инструменты отладчика Debug

<i>Инструмент отладчика</i>	<i>Горячая клавиша</i>	<i>Функциональное назначение</i>
Точка останова Toggle Breakpoint	F9	Точки останова используются для просмотра значений переменных или просмотра стека вызовов в определенный момент выполнения программы. Для установки точки останова нужно поместить курсор на нужную строку кода и нажать клавишу F9. Точка останова отображается в виде красной точки в левом поле. При запуске программы в режиме отладки ее выполнение будет остановлено при достижении этой точки.
Удалить точки останова Delete All Breakpoints	Ctrl+Shift+F9	Убирает все точки останова из программы.
Продолжить Continue	F5	Позволяет продолжить выполнение программы после точки останова в обычном режиме до следующей точки останова или конца программы.
Остановить отладчик Stop Debugging	Shift+F5	Завершает работу отладчика и останавливает выполнение программы.
Быстрая проверка Quick Watch	Shift+F9	Позволяет во время отладки задать переменные, значения которых могут дать информацию о корректности выполнения программы.

Шаг с заходом Step into	F11	Если строка содержит вызов функции, команда <i>Шаг с заходом</i> выполняет только сам вызов, а затем останавливает выполнение в первой строке кода внутри функции. В противном случае команда <i>Шаг с заходом</i> выполняет следующий оператор.
Шаг с обходом Step over	F10	Если строка содержит вызов функции, команда <i>Шаг с обходом</i> выполняет вызываемую функцию, а затем останавливает выполнение в первой строке кода внутри вызывающей функции. В противном случае команда <i>Шаг с обходом</i> выполняет следующий оператор.
Шаг с выходом Step out	Shift+F11	Команда <i>Шаг с выходом</i> возобновляет выполнение кода до возврата функции, а затем прерывает выполнение в точке возврата вызывающей функции.

В процессе работы с отладчиком могут быть открыты специальные окна (Debug → Windows) – Точки останова, Стеки вызовов, Авто, Локальные, и т.д. – позволяющие просматривать соответствующую информацию в реальном времени.

Алгоритм работы с отладчиком

Пусть у нас есть программа, которая работает некорректно, и мы хотим найти ошибку с помощью отладчика. Нужно проделать следующие действия:

- 1) Открыть файл в многофайловом проекте, в котором предположительно находится ошибка, и поставить точки останова в подозрительных строках кода. Для этого переходим на нужную строку и жмём F9.
- 2) Запускаем отладчик (F5) и программа остановится в первой точке останова. Внизу окна прописаны все точки останова с расположением в коде, а также контрольные значения переменных. Нужно отслеживать корректность контрольных значений. Для перехода к следующей точке жмем "Продолжить" (F5). Тогда выполнение программы остановится в точке останова. И так далее.
- 3) Если необходимо проверить код построчно, то после точки останова нужно жать "Шаг с обходом" (F10). Если более детально проверить каждую строчку, то "Шаг с заходом" (F11). Также есть возможность сделать "Шаг с выходом" (Shift+F11).
- 4) Если проблема найдена, то можно остановить отладчик "Остановить отладчик" (Shift+F5) и "Удалить все точки останова" (Ctrl+Shift+F9).

ПРИЛОЖЕНИЕ Г

Варианты заданий

Вариант 1.

Предметная область: Кадры предприятия

Структурный тип данных: Сотрудник предприятия (employee)

Элементы структуры:

Фамилия (surname)

Имя (name)

Отчество (patronymic)

Дата рождения (bdate)

Отдел (division)

Должность (position)

Задание:

- 1) Вывести на экран ФИО и должности всех сотрудников, работающих в заданном отделе (номер отдела вводится с клавиатуры).
- 2) Вывести на экран данные обо всех сотрудниках старше 50 лет.

Вариант 2.

Предметная область: Рынок недвижимости

Структурный тип данных: Квартира в многоэтажном доме (flat)

Элементы структуры:

Число комнат (rooms)

Общая площадь (total)

Жилая площадь (living)

Этаж (floor)

Этажность дома (storeys)

Район (district)

Задание:

- 1) Рассчитать общую площадь квартир, расположенных в заданном районе (название района вводится с клавиатуры)
- 2) Вывести на экран данные обо всех квартирах, расположенных на последнем этаже многоэтажного дома (5 и более этажей).

Вариант 3.

Предметная область: Магазин компьютерной техники

Структурный тип данных: Персональный компьютер (computer)

Элементы структуры:

Марка процессора (cpu_brand)
Тактовая частота процессора (cpu_clock)
Марка материнской платы (motherboard)
Марка жесткого диска (harddrive)
Марка видеокарты (videocard)
Цена в рублях (price)
Количество на складе (store)

Задание:

- 1) Рассчитать общую стоимость всех компьютеров, имеющих на складе.
- 2) Вывести на экран информацию о компьютере с самой высокой тактовой частотой в ценовом диапазоне от 20000 руб. до 30000 руб.

Вариант 4.

Предметная область: Пассажирские железнодорожные перевозки

Структурный тип данных: Железнодорожный билет (ticket)

Элементы структуры:

Номер рейса (train)
Станция отправления (station1)
Станция прибытия (station2)
Дата отправления (dep_day)
Время отправления (dep_time)
Вагон (coach)
Место (seat)
Стоимость (price)

Задание:

- 1) Вывести на экран информацию о самом дорогом из всех проданных билетов.
- 2) Вывести на экран количество свободных мест в заданном вагоне заданного поезда (номер рейса, дата отправления и номер вагона задаются с клавиатуры).

Вариант 5.

Предметная область: Техническое обслуживание (ТО) автомобилей

Структурный тип данных: Транспортное средство (vehicle)

Элементы структуры:

ФИО владельца (owner)
Регистрационный номер (reg_number)

Тип автомобиля (type)
Марка автомобиля (brand)
Объем двигателя (volume)
Мощность двигателя (power)
Дата последнего ТО (last_to)
Пробег (mileage)

Задание:

- 1) Вывести на экран номера всех автомобилей с пробегом свыше 100000 км.
- 2) Вывести на экран фамилии владельцев всех автомобилей, прошедших ТО в течение последнего месяца.

Вариант 6.

Предметная область: Растровая компьютерная графика

Структурный тип данных: Файл изображения (file)

Элементы структуры:

Имя файла (name)
Размер файла в байтах (size)
Формат данных (format)
Ширина в пикселях (width)
Высота в пикселях (height)
Разрешение (dpi)
Глубина цвета (depth)

Задание:

- 1) Вывести на экран название и формат всех графических файлов, имя которых начинается на заданную букву. Буква вводится с клавиатуры.
- 2) Найти самое большое по занимаемой площади изображение (учитываются поля width, height, dpi).

Вариант 7.

Предметная область: Домашняя DVD-фильмотека

Структурный тип данных: Диск формата DVD/Blu-ray (disk)

Элементы структуры:

Название фильма (title)
Режиссер (director)
Жанр фильма (genre)
Длительность в мин. (duration)

Год выпуска (year)

Рейтинг IMDB (IMDB)

Задание:

- 1) Найти самый поздний фильм заданного режиссера. Фамилия режиссера вводится с клавиатуры.
- 2) Вывести на экран все фильмы в жанре комедии с рейтингом IMBD выше 8.0.

Вариант 8.

Предметная область: Компьютерные игры

Структурный тип данных: Компьютерная игра (game)

Элементы структуры:

Название (title)

Жанр (genre)

Платформа (platform)

Год выпуска (year)

Компания-разработчик (developer)

Число участников (players)

Мин. частота процессора в МГц (min_cpu)

Мин. объем опер. памяти в МБ (min_ram)

Мин. объем места на диске в МБ (min_hdd)

Задание:

- 1) Найти компьютерную игру заданного жанра с самыми низкими требованиями по объему используемой оперативной памяти. Жанр игры задается с клавиатуры.
- 2) Вывести на экран названия всех компьютерных игр, которые могут быть установлены на заданном компьютере. Характеристики компьютера – частота процессора, объемы памяти RAM и HDD задаются с клавиатуры.

Вариант 9.

Предметная область: Рынок автотранспортных средств

Структурный тип данных: Автомобиль (car)

Элементы структуры:

Марка автомобиля (brand)

Название модели (model)

Название комплектации (complect)

Рыночный сегмент (segment)

Тип автомобильного кузова (body)

Объем двигателя в литрах (volume)

Мощность двигателя в л.с. (power)

Год выпуска (year)

Стоимость в рублях (price)

Задание:

- 1) Рассчитать и вывести на экран отношение мощности двигателя к стоимости автомобиля для всех представленных моделей и комплектаций. Найти модель с самым высоким отношением.
- 2) Вывести на экран названия всех комплектаций заданной модели автомобиля в порядке возрастания их стоимости. Название модели вводится с клавиатуры.

Вариант 10.

Предметная область: Политическая география

Структурный тип данных: Суверенное государство (state)

Элементы структуры:

Название государства (country)

Столица (capital)

Форма правления (government)

Язык(и) (language)

Религия (religion)

Площадь территории (area)

Численность населения (population)

Континент (continent)

Задание:

- 1) Рассчитать суммарную площадь и население государств Северной Америки.
- 2) Вывести на экран название и столицу самого крупного по численности населения испано-язычного государства.

Вариант 11.

Предметная область: Спортивные достижения человечества

Структурный тип данных: Спортивный рекорд (record)

Элементы структуры:

Вид спорта (sport)

Дисциплина (discipline)

Вид рекорда (мировой/европейский/...) (type)

Мужской/женский (gender)

Дата установления (date)
Фамилия рекордсмена (name)
Страна (country)
Значение рекорда (achievement)

Задание:

- 1) Рассчитать общее количество рекордов разного уровня для заданного вида спорта. Вид спорта вводится с клавиатуры.
- 2) Вывести на экран все мировые рекорды, установленные женщинами в течение последнего года.

Вариант 12.

Предметная область: Валютный рынок

Структурный тип данных: Текущий курс иностранной валюты (course)

Элементы структуры:

Денежная единица (currency)
Государство (state)
Код ISO4217 (code)
Дробная единица (subunit)
Величина дробления (fraction)
Текущий курс в рублях (rate)

Задание:

- 1) Вывести на экран текущие значения курсов всех иностранных валют в формате «денежная единица – курс в рублях».
- 2) Вывести на экран информацию обо всех денежных единицах, курс которых на текущий момент превышает курс американского доллара.

Вариант 13.

Предметная область: Глобальная сеть Интернет

Структурный тип данных: Интернет-ресурс, сайт (site)

Элементы структуры:

Название ресурса (name)
Тип ресурса - поисковый, информационный, ... (type)
Доменное имя (domain)
Среднее число пользователей в день (users)
Число веб-сайтов, ссылающихся на данный ресурс (linked)
Среднее время загрузки веб-страницы (time)

Используемые веб-технологии (technologies)

Задание:

- 1) Найдите и выведите на экран доменное имя самого быстрого поискового интернет-сервиса (т.е. с самой высокой скоростью загрузки страницы).
- 2) Выведите на экран информацию обо всех интернет-ресурсах, у которых среднее число пользователей в день превышает заданную величину (вводится с клавиатуры).

Вариант 14.

Предметная область: Высшее профессиональное образование

Структурный тип данных: Итоговая оценка студента по дисциплине (rating)

Элементы структуры:

- ФИО студента (name)
- Номер зачетной книжки (number)
- Специальность (occupation)
- Группа (group)
- Название дисциплины (course)
- Номер семестра (semester)
- Итоговая оценка по 100-балльной шкале (mark)

Задание:

- 1) Выведите на экран ФИО всех студентов, получивших неудовлетворительные оценки (менее 60 баллов) в заданном семестре. Номер семестра вводится с клавиатуры.
- 2) Выберите студента с самой высокой итоговой оценкой по дисциплине.

Вариант 15.

Предметная область: Авиаперевозки

Структурный тип данных: Авиарейс (flight)

Элементы структуры:

- Номер рейса (number)
- Авиакомпания (airline)
- Откуда (from)
- Куда (to)
- Протяженность в км (distance)
- Длительность полета (duration)
- Тип самолета (aircraft)
- Число посадочных мест (passengers)

Задание:

- 1) Найдите авиарейс с самой высокой средней скоростью полета. Выведите на экран тип используемого самолета.
- 2) Рассчитайте общую протяженность всех авиарейсов, выполняемых из аэропорта Москвы.

Вариант 16.

Предметная область: Учет больных в поликлинике по месту жительства

Структурный тип данных: Обращение в поликлинику (visit)

Элементы структуры:

ФИО пациента (patient)
Номер мед. полиса (insurance)
Номер мед. карты (card)
ФИО врача (doctor)
Дата обращения (date)
Диагноз (diagnosis)
Назначенное лечение (therapy)

Задание:

- 1) Найдите пациента в самой поздней датой обращения в поликлинику.
- 2) Рассчитайте общее количество пациентов, которые были приняты данным врачом за все время его работы. ФИО врача вводится с клавиатуры.

Вариант 17.

Предметная область: Чемпионат страны по командному виду спорта

Структурный тип данных: Команда-участник (participant)

Элементы структуры:

Название клуба (club)
Город (city)
ФИО тренера (coach)
Дата основания клуба (date)
Бюджет команды в руб. (budget)
Текущее количество очков (points)
Текущее место в чемпионате (place)

Задание:

- 1) Найдите команду с самым высоким отношением количества набранных очков к объему потраченных денег (бюджету).

2) Выведите на экран названия всех команд, выступающих за выбранный город. Название города вводится с клавиатуры.

Вариант 18.

Предметная область: Периодическая система химических элементов

Структурный тип данных: Химический элемент (element)

Элементы структуры:

Название элемента (name)

Символ (symbol)

Номер в таблице (number)

Тип (type)

Атомная масса (mass)

Электронная конфигурация (electrons)

Задание:

- 1) Найдите химический элемент по его атомной массе.
- 2) Выведите на экран названия всех имеющихся в массиве химических элементов - металлов.

Вариант 19.

Предметная область: Визит в библиотеку

Структурный тип данных: Читательский билет (library_record)

Элементы структуры:

Номер читательского билета (number)

ФИО читателя (name)

Адрес читателя (address)

Название книги (book)

Срок возврата книги (return)

Срок действия билета (expiration)

Задание:

- 1) Найдите всех читателей, у которых на руках находится «Справочник по общей химии».
- 2) Выведите на экран названия всех читателей библиотеки, которые сегодня должны вернуть книги в библиотеку.

Вариант 20.

Предметная область: Банковская система

Структурный тип данных: Банковский счет (account)

Элементы структуры:

Номер счета (number)
ФИО владельца (name)
Паспортные данные (ID)
Тип счета (type)
Размер счета (funds)
Валюта счета (currency)

Задание:

- 1) Найдите всех владельцев счетов типа “savings” (сберегательный).
- 2) Выведите на экран имена всех владельцев долларовых счетов, размер счета которых не превышает 100\$.

Вариант 21.

Предметная область: Научная премия

Структурный тип данных: Нобелевская премия (nobel_prize)

Элементы структуры:

ФИО (name)
Раздел науки (science)
Год присуждения (year)
Страна ученого (country)
Размер премии в млн \$ (size)

Задание:

- 1) Найдите всех лауреатов Нобелевской премии по физике.
- 2) Выведите на экран ФИО и дату присуждения премии всех лауреатов из России и СССР по физике и химии.

Вариант 22.

Предметная область: Визовая система

Структурный тип данных: Шенгенская виза (visa)

Элементы структуры:

Номер загранпаспорта (passport)
ФИО (name)
Дата рождения (bdate)
Средний месячный доход (income)
Гражданство (nationality)
Дата въезда в шенгенскую зону (arrival)

Дата выезда из шенгенской зоны (departure)

Задание:

- 1) Найдите человека с заданным номером загранпаспорта.
- 2) Найдите всех граждан РФ, находившихся во Франции в ночь с 31-го декабря 2015 года на 1 января 2016 года.

Вариант 23.

Предметная область: Жилищно-коммунальное хозяйство

Структурный тип данных: Платежный документ ЖКХ (document)

Элементы структуры:

Номер личного счета (account)
Платательщик (name)
Адрес (address)
Общая площадь, кв.м. (total_area)
Жилая площадь, кв.м. (living_area)
Количество проживающих (count)
Общая сумма оплаты, руб (total_price)

Задание:

- 1) Найдите всех плательщиков, проживающих в квартире с площадью более 50 кв.м.
- 2) Выведите на экран адреса всех квартир, в которых нет зарегистрированных проживающих и общая сумма оплаты коммунальных услуг превышает 3 тыс. руб.

Вариант 24.

Предметная область: Налоговая служба

Структурный тип данных: Налогоплательщик (taxpayer)

Элементы структуры:

ФИО (name)
ИНН (inn)
Адрес регистрации (address)
Общая инвентаризационная стоимость всех земельных участков (territory)
Общая инвентаризационная стоимость всех жилых помещений (habitable)
Общая инвентаризационная стоимость всех нежилых помещений (unhabitable)
Общая сумма налога (tax)
Общая сумма задолженности (debt)

Задание:

- 1) Найдите всех должников и выведите их ФИО и адрес.

2) Выведите на экран ФИО, адрес и общую сумму налога с учетом задолженности для каждого плательщика и в конце подведите итог – общую сумму неуплаченных денег в Федеральную налоговую службу.

Вариант 25.

Предметная область: Дорожное движение

Структурный тип данных: Водительское удостоверение (driver_license)

Элементы структуры:

Номер удостоверения (ser_num)

ФИО (name)

Дата рождения (b_date)

Дата выдачи удостоверения (begin)

Дата окончания срока действия удостоверения (end)

Регион (region)

Задание:

- 1) Выведите на экран номера всех истекших удостоверений водителей, проживающих на территории Волгоградской области.
- 2) Выведите на экран всех водителей, имеющих стаж вождения менее 3 лет и возраст менее 21 года.

ПРИЛОЖЕНИЕ Д
Структура файла my_books.txt

3

Приключения Тома Сойера. Приключения Гекльберри Финна.

Марк Твен

ДетГИЗ

1985

92

Лишний козырь в рукаве

Джеймс Хедли Чейз

Эксмо

2001

205

Структуры данных и другие объекты в C++

Майкл Мейн, Уолтер Савитч

Вильямс

2002

832

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
ЛАБОРАТОРНАЯ РАБОТА №1 Структуры C++	4
ЛАБОРАТОРНАЯ РАБОТА №2 Строки, потоки, функции	13
ЛАБОРАТОРНАЯ РАБОТА №3 Классы и объекты	24
ЛАБОРАТОРНАЯ РАБОТА №4 Композиция классов	36
ЛАБОРАТОРНАЯ РАБОТА №5 Перегрузка операций	49
ЛАБОРАТОРНАЯ РАБОТА №6 Наследование	62
ЛАБОРАТОРНАЯ РАБОТА №7 Виртуальные функции	77
ПРИЛОЖЕНИЕ А Работа с консольными приложениями в Microsoft Visual Studio	86
ПРИЛОЖЕНИЕ Б Директивы препроцессора	87
ПРИЛОЖЕНИЕ В Работа с отладчиком Debug в Microsoft Visual Studio	89
ПРИЛОЖЕНИЕ Г Варианты заданий	91
ПРИЛОЖЕНИЕ Д Структура файла my_books.txt	103
СПИСОК ЛИТЕРАТУРЫ.....	105

СПИСОК ЛИТЕРАТУРЫ

1. **Holub A.I.** Enough rope to shoot yourself in the foot: Rules for C and C++ programming [Книга]. - Indianapolis : McGraw-Hill, 1995. - стр. 186.
2. **Stroustrup B.** The C++ programming language [Книга]. - New Jersey : Pearson Education, Inc., 2014. - 4th edition : стр. 1352.
3. **Буч Г. [и др.]** Объектно-ориентированный анализ и проектирование приложений с примерами приложений. [Книга]. - Москва : Издательский дом "Вильямс", 2008. - 3-е изд.: пер. с англ. : стр. 720.
4. **Глушаков С. В., Коваль А. В. и Смирнов С. В.** Язык программирования C++. Учебный курс. [Книга]. - Ростов-на-Дону : "Феникс", 2001. - стр. 506.
5. **Джосаттис Н.** Стандартная библиотека C++. Справочное руководство. [Книга]. - Москва : Издательство "Вильямс", 2016. - 2-е издание : стр. 1136.
6. **Мейн М. и Савитч У.** Структуры и другие объекты в C++ [Книга]. - Москва : Издательский дом "Вильямс", 2002. - 2-е изд.: Пер. с англ. : стр. 832.
7. **Павловская Т. А. и Щупак Ю. А.** C++. Объектно-ориентированное программирование: Практикум [Книга]. - СПб. : Питер, 2004. - стр. 265.
8. **Прата С.** Язык программирования C++. Лекции и упражнения [Книга]. - Москва : Издательство "Вильямс", 2016. - 6-е издание : стр. 1248.
9. **Романов Е.Л.** Практикум по программированию на C++: Уч. пособие. [Книга]. - СПб. : БХВ, 2004. - стр. 432.
10. **Савитч У.** Язык C++. Курс объектно-ориентированного программирования [Книга]. - Москва : Издательский дом "Вильямс", 2001. - 3-е изд.: Пер. с англ. : стр. 704.
11. **Шилдт Г.** C++ Базовый курс [Книга]. - Москва : Издательство "Вильямс", 2016. - 3-е издание : стр. 624.
12. **Эккель Б. и Эллисон Ч.** Философия C++. Практическое программирование. [Книга]. - СПб. : Питер, 2004. - стр. 608.
13. **Эккель Б.** Философия C++. Введение в стандартный C++. [Книга]. - СПб. : Питер, 2004. - 2-е изд. : стр. 572.