МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 1**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Классы и объекты в С++»**

*Вариант № 16*

Выполнил:

Студент группы

КТбо2-8

Липов Э. А.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# Цель работы

Ознакомление с технологией построения классов, разработки методов, пе­регрузки операций и обработки исключений.

# Задание, вариант № 16

Определить классы Книга (Book), Библиотека (Library) и читатели (Readers). Поля книги: ФИО автора (ов), название (Title), год издания (Year), из­дательство (Publishers), Id, выдана/на полке. Библиотека содержит массив книг и методы выдачи и возвращения книги. Читатели – свой Id и Id книги, которая им выдана. Предусмотреть выдачу и возвращение книг, а также поиск книги по Id с сообщением выдана она или нет и если выдана, то кому. Список книг для ини­циализации программы хранить в файле (не менее 10).

# Ход работы

## Спецификация класса “Book”

Диаграмма класса приведена на Рисунок 1.

Класс “Book” со всеми методами объявлен в заголовочном файле “Book.hpp”. Все функции определены в файле “Book.cpp”.

В классе “Book” определены шесть закрытых полей, описывающих данные книги: поле “author\_” (имя автора), поле “title\_” (название книги), поле “year\_” (год издания), поле “publisher\_” (издатель), поле “bid\_” (ID) и поле “is\_issued\_” (выдана книга или нет).

У Book определен конструктор “Book()” для создания пустой книги и кон­структор “Book(std::string& author, std::string& title, std::string& publisher, int year = 0, int bid = 0, bool is\_issued = false)”, инициализирующий поля, описанные выше, а также конструктор создания книги из массива строк и конструктор копирования.

Для класса определены методы “GetBid()”, “GetIssued()”, дающие воз­можность прочесть закрытые поля класса извне, а также метод “SetIssued()”, дающий право записи данных в закрытое поля класса извне.

Text

Description automatically generated

Рисунок — Диаграмма класса “Book”

Также перегружен оператор присваивания (см. Листинг 1.1) экземпляра класса “Book” с другим экземпляром этого класса, оператор равенства двух эк­земпляров класса и оператор вывода на экран.

* + 1. Листинг 1.1

Book::Book() {

year\_ = 0;

bid\_ = 0;

is\_issued\_ = false;

}

Book::Book(std::string& author, std::string& title, std::string& publisher, int year, int bid, bool is\_issued)

: author\_(author), title\_(title), publisher\_(publisher), year\_(year), bid\_(bid), is\_issued\_(is\_issued) {}

Book::Book(std::string\* parameters) {

author\_ = parameters[0];

title\_ = parameters[1];

year\_ = std::stoi(parameters[2]);

publisher\_ = parameters[3];

bid\_ = std::stoi(parameters[4]);

is\_issued\_ = false;

}

Book::Book(const Book& other\_book) {

author\_ = other\_book.author\_;

title\_ = other\_book.title\_;

year\_ = other\_book.year\_;

publisher\_ = other\_book.publisher\_;

bid\_ = other\_book.bid\_;

is\_issued\_ = other\_book.is\_issued\_;

}

int Book::GetBid() const {

return bid\_;

}

bool Book::GetIsIssued() const {

return is\_issued\_;

}

void Book::SetIsIssued(bool is\_issued) {

is\_issued\_ = is\_issued;

}

Book& Book::operator=(const Book& book) {

if (this == &book) {

return \*this;

}

author\_ = book.author\_;

title\_ = book.title\_;

year\_ = book.year\_;

publisher\_ = book.publisher\_;

bid\_ = book.bid\_;

is\_issued\_ = book.is\_issued\_;

return \*this;

}

bool Book::operator==(const Book& book) {

bool expression = title\_ == book.title\_;

expression = expression && (title\_ == book.title\_);

expression = expression && (year\_ == book.year\_);

expression = expression && (publisher\_ == book.publisher\_);

expression = expression && (bid\_ == book.bid\_);

expression = expression && (is\_issued\_ == book.is\_issued\_);

return expression;

}

std::ostream& operator<<(std::ostream& os, const Book& book) {

os << "Автор: " << book.author\_ << std::endl;

os << "Название: " << book.title\_ << std::endl;

os << "Год: " << book.year\_ << std::endl;

os << "Издатель: " << book.publisher\_ << std::endl;

os << "BID: " << book.bid\_ << std::endl;

if (book.is\_issued\_) {

os << "Выдана: да";

} else {

os << "Выдана: нет";

}

return os;

}

## Спецификация класса “Reader”

Диаграмма класса приведена на Рисунок 2.

Text

Description automatically generated

Рисунок — Диаграмма класса “Reader”

Класс “Reader” со всеми методами объявлен в заголовочном файле “Reader.hpp”. Все функции определены в файле “Reader.cpp”.

В классе “Reader” определены два закрытых поля, описывающих данные читателя: поле “rid \_” (ID читателя), поле “bid\_” (ID взятой книги).

У “Reader” определен конструктор “Reader(rid : int, rid : int)” инициализирующий поля, описанные выше.

Для класса определены методы “GetRid()”, “GetBid()”, дающие возмож­ность прочесть закрытые поля класса извне, а также метод “SetBid()”, дающий право записи данных в закрытое поля класса извне.

Также перегружен (см. Листинг 1.2) оператор равенства двух экземпляров класса и оператор вывода на экран.

* + 1. Листинг 1.2

Reader::Reader(int rid, int bid) : rid\_(rid), bid\_(bid) {}

int Reader::GetRid() const {

return rid\_;

}

int Reader::GetBid() const {

return bid\_;

}

void Reader::SetBid(int bid) {

bid\_ = bid;

}

bool Reader::operator==(const Reader& reader) {

return ((rid\_ == reader.rid\_) && (bid\_ == reader.bid\_));

}

std::ostream& operator<<(std::ostream& os, const Reader& reader) {

os << "RID: " << reader.rid\_;

os << ", BID: " << reader.bid\_;

return os;

}

## Спецификация класса “Library”

Диаграмма класса приведена на Рисунок 3.

Text

Description automatically generated

Рисунок — Диаграмма класса “Library”

Класс “Library” со всеми методами объявлен в заголовочном файле “Library.hpp”. Все функции определены в файле “Library.cpp”.

В классе “Library” определены два закрытых поля, описывающих данные библиотеки: поле “books\_” (указатель на список книг), поле “readers\_” (указа­тель на список читателей).

У “Library” определен конструктор по умолчанию и конструктор “Library(LinkedList<Book>& books, LinkedList<Reader>& readers)” ини­циализирующий поля, описанные выше.

Для класса определены методы “GetBooks()”, “GetReaders()”, дающие возможность прочесть закрытые поля класса извне.

Также перегружен (см. Листинг 1.3) оператор присваивания экземпляра класса.

* + 1. Листинг 1.3

Library::Library(LinkedList<Book>& books, LinkedList<Reader>& readers) {

books\_ = new LinkedList<Book>(books);

readers\_ = new LinkedList<Reader>(readers);

}

LinkedList<Book>\* Library::GetBooks() const {

return books\_;

}

LinkedList<Reader>\* Library::GetReaders() const {

return readers\_;

}

Library& Library::operator=(const Library& library) {

if (this == &library) {

return \*this;

}

delete books\_;

delete readers\_;

books\_ = new LinkedList<Book>(\*(library.books\_));

readers\_ = new LinkedList<Reader>(\*(library.readers\_));

return \*this;

}

## Спецификация класса “Node”

Диаграмма класса приведена на Рисунок 4.

Graphical user interface, text, application

Description automatically generated

Рисунок — Диаграмма класса “Node”

Класс “Node” со всеми методами объявлен в заголовочном файле “Node.hpp”. Все функции определены в том же файле.

В классе “Node” определены три открытых поля, описывающих данные элемента списка: поле “data” (хранимые данные), поле “next” (указатель на сле­дующий элемент) и поле “prev” (указатель на предыдущий элемент).

У “Node” определен конструктор “Node(const T& data = T(), Node\* next = nullptr, Node\* prev = nullptr)” инициализирующий поля, описан­ные выше, и конструктор копирования (см. Листинг 1.4).

* + 1. Листинг 1.4

Node(const T& data = T(), Node\* next = nullptr, Node\* prev = nullptr) : data(data), next(next), prev(prev) {}

Node(const Node& other\_node) : data(other\_node.data), next(other\_node.next), prev(other\_node.prev) {}

## Спецификация класса “Iterator”

Диаграмма класса приведена на Рисунок 5.

Text, letter

Description automatically generated

Рисунок — Диаграмма класса “Iterator”

Класс “Iterator” со всеми методами объявлен в заголовочном файле “Iteration.hpp”. Все функции определены в том же файле.

В классе “Iterator” определено одно закрытое поле, описывающее дан­ные итератора: поле “nodes\_” (указатель на элемент списка).

У “Iterator” определен конструктор “Iterator(Node<T>\* nodes = nullptr)” инициализирующий поле, описанное выше.

Для класса определены методы “StepForward()”, “StepBackward()”, да­ющие возможность передвигаться по списку (см. Листинг 1.5).

* + 1. Листинг 1.5

explicit Iterator(Node<T>\* nodes = nullptr) : nodes\_(nodes) {}

Node<T>\* StepForward() {

if (nodes\_ == nullptr) {

return nullptr;

}

nodes\_ = nodes\_->next;

return nodes\_;

}

Node<T> StepBackward() {

if (nodes\_ == nullptr) {

return nullptr;

}

nodes\_ = nodes\_->prev;

return nodes\_;

}

## Спецификация класса “LinkedList”

Диаграмма класса приведена на Рисунок 6.

Класс “LinkedList” со всеми методами объявлен в заголовочном файле “LinkedList.hpp”. Все функции определены в том же файле.

В классе “LinkedList” определено два закрытых поля, описывающее дан­ные списка: поле “head\_” (указатель на первый элемент списка), поле “tail\_” (указатель на последний элемент списка).

У “LinkedList” определен конструктор “LinkedList()” создающий пу­стой список и конструктор копирования.

Также перегружен (см. Листинг 1.6) оператор присваивания экземпляра класса.

Text

Description automatically generated

Рисунок — Диаграмма класса “LinkedList”

* + 1. Листинг 1.6

LinkedList() {

head\_ = new Node<T>();

tail\_ = new Node<T>();

}

LinkedList(const LinkedList<T>& other\_list) {

if (other\_list.IsEmpty()) {

head\_ = new Node<T>();

tail\_ = new Node<T>();

return;

}

auto\* list = other\_list.GetIteratorAtHead();

auto\* node = list->StepForward();

head\_ = new Node<T>();

tail\_ = new Node<T>();

PushBack(node->data);

node = list->StepForward();

while (node != nullptr) {

PushBack(node->data);

node = list->StepForward();

}

}

LinkedList<T>& operator=(const LinkedList<T>& linked\_list) {

if (this == &linked\_list) {

return \*this;

}

delete head\_;

delete tail\_;

head\_ = new Node<T>(\*linked\_list.head\_);

tail\_ = new Node<T>(\*linked\_list.tail\_);

return \*this;

}

## Спецификация класса “Console”

Диаграмма класса приведена на Рисунок 6.

Класс “Console” со всеми методами объявлен в заголовочном файле “Console.hpp”. Все функции определены в файле “Console.cpp”.

В классе “Console” определено одно закрытое поле для экземпляра класса “Library”: поле “library\_” (указатель на библиотеку).

У “Console” определен конструктор по умолчанию (см. Листинг 1.6).

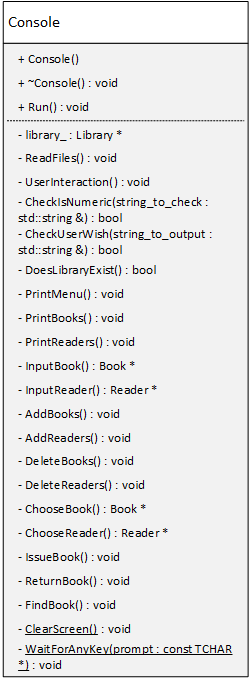


Рисунок — Диаграмма класса “Console”

* + 1. Листинг 1.7

Console() = default;

# Диаграмма классов

Диаграмма классов представлена на Рисунок 8.

Diagram

Description automatically generated

Рисунок — Диаграмма классов