Aleksander Czachór EOOP project – keyword „Library”

1. Description:

The program is designed to serve as a digital database for a public library. It is responsible for managing books and clients, generating and archiving borrow Instances, and searching for any element by any of it's data (Ex. A book by it's title, or a client by his phone number).

To ensure data encapsulation and cohesion throughout the project, it's structure consists of following classes:

- Book: Encapsulates following data of an individual book: ID, name, title, and a bool values for borrowed and deleted state. Has member functions responsible for getting from and setting data in an instance, equality, write and read operators.
- Book Collection: Has a list of books and a variable responsible for assigning unique ID's to said books. It is responsible for assigning books unique ID's, searching for a book by any of it's data, and changing the borrowed or deleted status of a particular book.
- Client: Encapsulates following data of an individual client: ID, name, phone number, number of books borrowed, overdue and a bool value for deleted state. Has member functions responsible for getting from and setting data in an instance, equality, write and read operators.
- Client Base: Has a list of clients, and a variable responsible for assigning unique ID's to said clients. It is responsible for assigning clients unique ID's, searching for a client by any of it's data, and changing the deleted status of a particular client.
- Borrow Data: Encapsulates following data of an individual borrow instance: ID, book ID, client ID, and borrow and return dates. Has member functions responsible for getting from and setting data in an instance, equality, write and read operators.
- Borrow Records: Has a list of borrow instances, and a variable responsible for assigning unique ID's to said borrows. It is responsible for creating borrows, assigning them their unique ID's, and searching for a borrow by any of it's data.
- Library: Base class for the project. Has a book collection, client base and two borrow records instances. It is responsible for linking collections of elements, performing borrows, returns, adding books and clients, calculating overdue, and reading or saving data to a text file.

Project also includes files with exception overloads, and with enumeration classes for search functions.

Program's strengths:

- Data in a library is encapsulated, to make sure that no elements are deleted
- Efficient and easy to use find functions
- Program has custom errors to tell user directly where the error is
- Program supports reading from and writing to filed

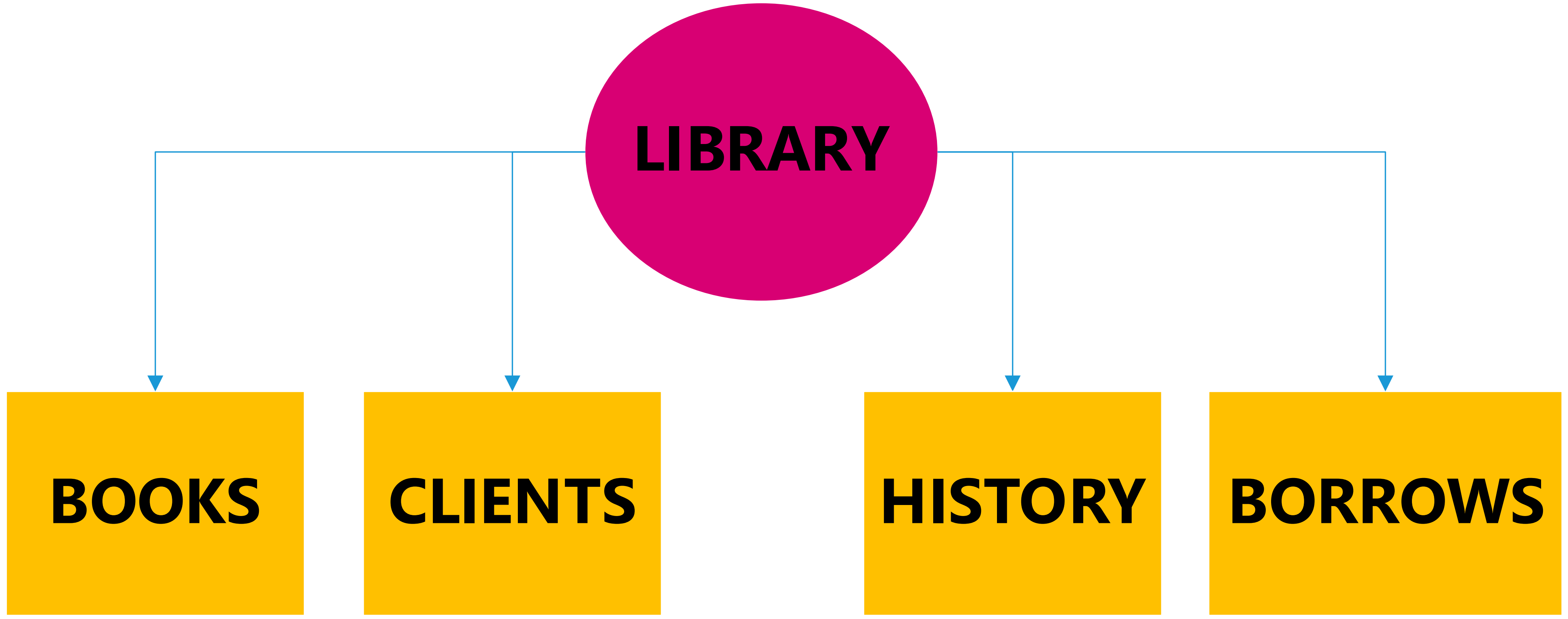
Program's limitations:

- Program can support only one library, not a net of libraries
- Data can't be deleted

Aleksander Czachór EOOP project – keyword „Library”

2. Memory Map:

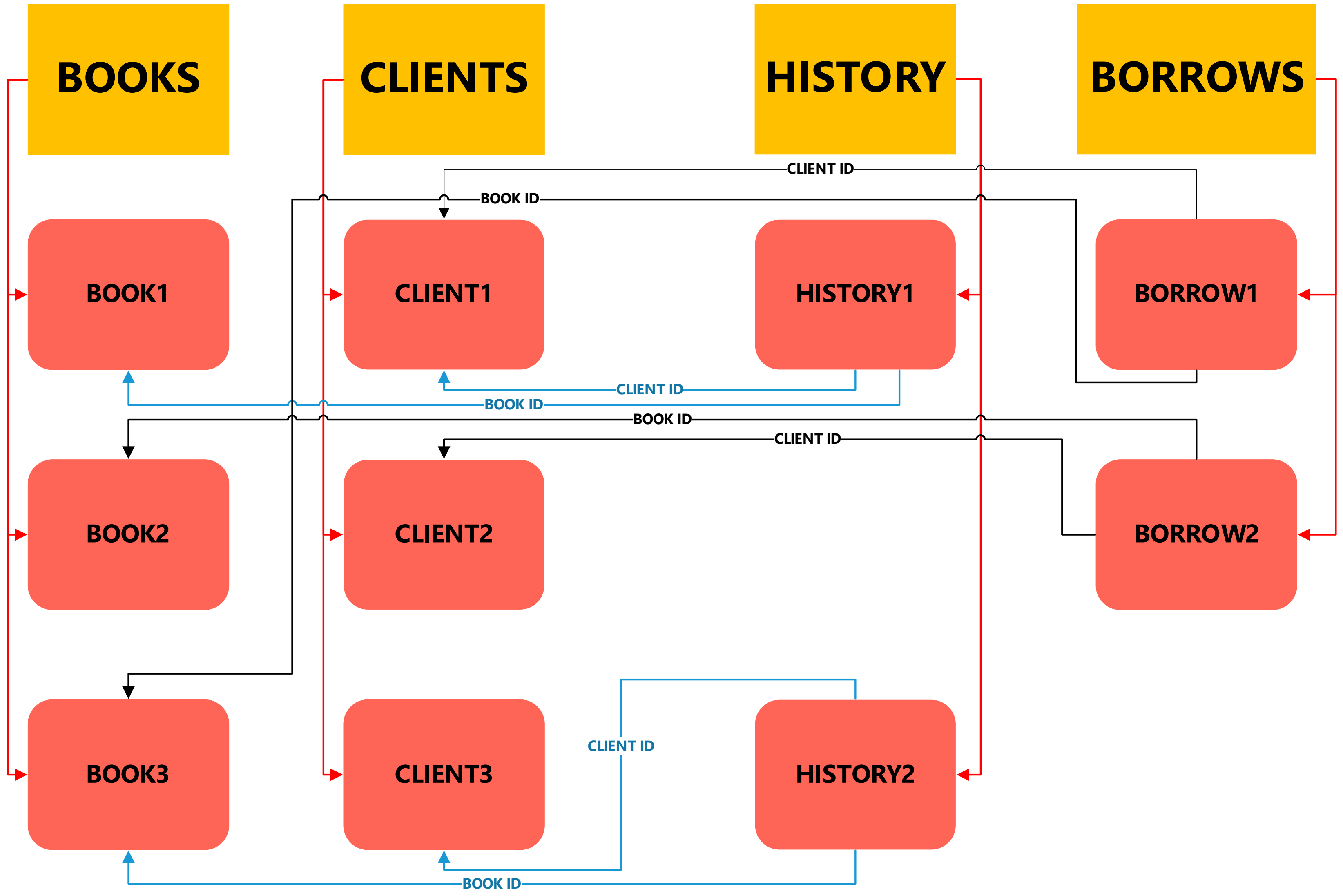
General library layout:



Aleksander Czachór EOOP project – keyword „Library”

Example relations between objects:

- **Red: iafiliation with a certain list**
- **Blue: Relation Between History and books/clients (past boorows)**
- **Back: Relation BetweenBorrows and books/clients (present boorows)**



Aleksander Czachór EOOP project – keyword „Library”

3. Declarations of classes, exceptions and fields:

- **Book:**

```
class book {
private:
    std::string title;  //title of a book
    std::string author; // author of a book
    unsigned int ID;    //unique identification number of a book
    bool borrowed;  //0 if a book is not borrowed, 1 if a book is borrowed
    bool available; //0 if a book was destroyed/deleted etc.

public:
    book(unsigned int Id, const std::string& sourceTitle,const std::string& sourceAuthor,bool Borrowed=false, boolAvailable=true);
    //creates a book with a given title and author
    //ID of a book is equal to bookID in the library with exception of loading from a file
    //after creating a book 1 is added to the bookID, to keep ID unique
    //borrowed is set to 0 (1 would mean that a book is borrowed)

    book(unsigned int Id, const book& source);
    //creates a book as a copy of another book
    //ID of a copy is not equal to source ID, ID has to be unique
    //ID of a copy is equal to bookID in the library
    //after creating a book 1 is added to the bookID, to keep the ID unique
    //copy's borrow is 0, even if the source book is already borrowed

    ~book();
    //deletes author and title strings

    [[nodiscard]] std::string getTitle() const;
    //returns title of a book

    [[nodiscard]] std::string getAuthor() const;
    //returns author of a book

    [[nodiscard]] unsigned int getID() const;
    //returns ID of a book

    [[nodiscard]] bool getBorrowed() const;
    //returns borrow value of a book

    [[nodiscard]] bool getAvailable() const;
```

Aleksander Czachór EOOP project – keyword „Library”

```
    void setBorrowed(bool value);
    //set the borrowed value of a book

    void setAvailable(bool value);
    //set the available value of a book

    bool operator==(const book& other) const;
    //operator is set to compare books
    //it compares them only by their ID

    friend std::ostream& operator<<(std::ostream& os, const book& data);
    //outputs book data in a following format
    //ID    title   author    borrowed    available

    friend class library;
    friend class bookCollection;
};
```

Aleksander Czachór EOOP project – keyword „Library”

- **BookCollection:**

```
class bookCollection{
private:
    std::list<book> bookList;
    unsigned int bookId;

    [[nodiscard]] bool match(const book& book) const {
        return true;
    }
    //base case for recursive call of match

    template<typename First, typename... Args>
    bool match(const book& book, BookSearch field, First&& first, Args&&... args) const {
        const auto& fieldValue = getField <std::decay_t<decltype(first)>>(book, field);

        if constexpr (std::is_same_v<std::decay_t<decltype(fieldValue)>, std::string>) {
            const auto& fieldValueStr = static_cast<const std::string&>(fieldValue);
            const std::string& firstStr = first;

            std::string lowercaseFieldValue = fieldValueStr;
            std::string lowercaseFirst = firstStr;

            std::transform(lowercaseFieldValue.begin(), lowercaseFieldValue.end(), lowercaseFieldValue.begin(), ::tolower);
            std::transform(lowercaseFirst.begin(), lowercaseFirst.end(), lowercaseFirst.begin(), ::tolower);

            if (lowercaseFieldValue.find(lowercaseFirst) != std::string::npos) {
                return match(book, std::forward<Args>(args)...);
            }
        } else {
            if (fieldValue == first) {
                return match(book, std::forward<Args>(args)...);
            }
        }

        return false;
    }
    //matches a parameter of book instance with a respective field
    //returns itself recursively with the rest of arguments if match passes
    template<typename Field>
    const Field& getField(const book& book, BookSearch field) const;
    //returns parameter of a book instance
    //respective to a field passed in arguments
```

Aleksander Czachór EOOP project – keyword „Library”

```cpp
public:
    bookCollection();
    ~bookCollection();

    void addBook(const std::string& title, const std::string& author);
    void print();
    //outputs books in bookList in a following format:
    //ID    title    author    borrowed/available

    static void print(const std::vector<book*>& booksFiltered);
    //prints books with pointers to books given by a vector in a following format:
    //ID    title    author    borrowed

    [[nodiscard]] int size() const;
    //returns size of bookList

    template<typename... Args>
    std::vector<book*> findBooks(Args&&... args) const {
        if(sizeof...(args) % 2 !=0) throw std::invalid_argument("Invalid number of arguments. Must provide search field and value
                                                                 pairs.");

        std::vector<book*> foundBooks;

        for (const auto& book : bookList) {
            if (match(book, std::forward<Args>(args)...)) {
                foundBooks.push_back(const_cast<::book*>(&book));  // Using const_cast to remove constness
            }
        }

        return foundBooks;
    }
    //returns a vector of pointers to books complying with parameters set in arguments
    //parameters have to be typed in a following format
    //field, value,...
    //where field is a parameter of a book (Ex. title or ID)

    friend std::ostream &operator<<(std::ostream &os, const bookCollection &data);
    //outputs each book in a separate line
    friend std::istream &operator>>(std::istream &is, bookCollection &data);
    //reads book data from a file and creates books complying with that data
};
```

Aleksander Czachór EOOP project – keyword „Library”

- **Client:**

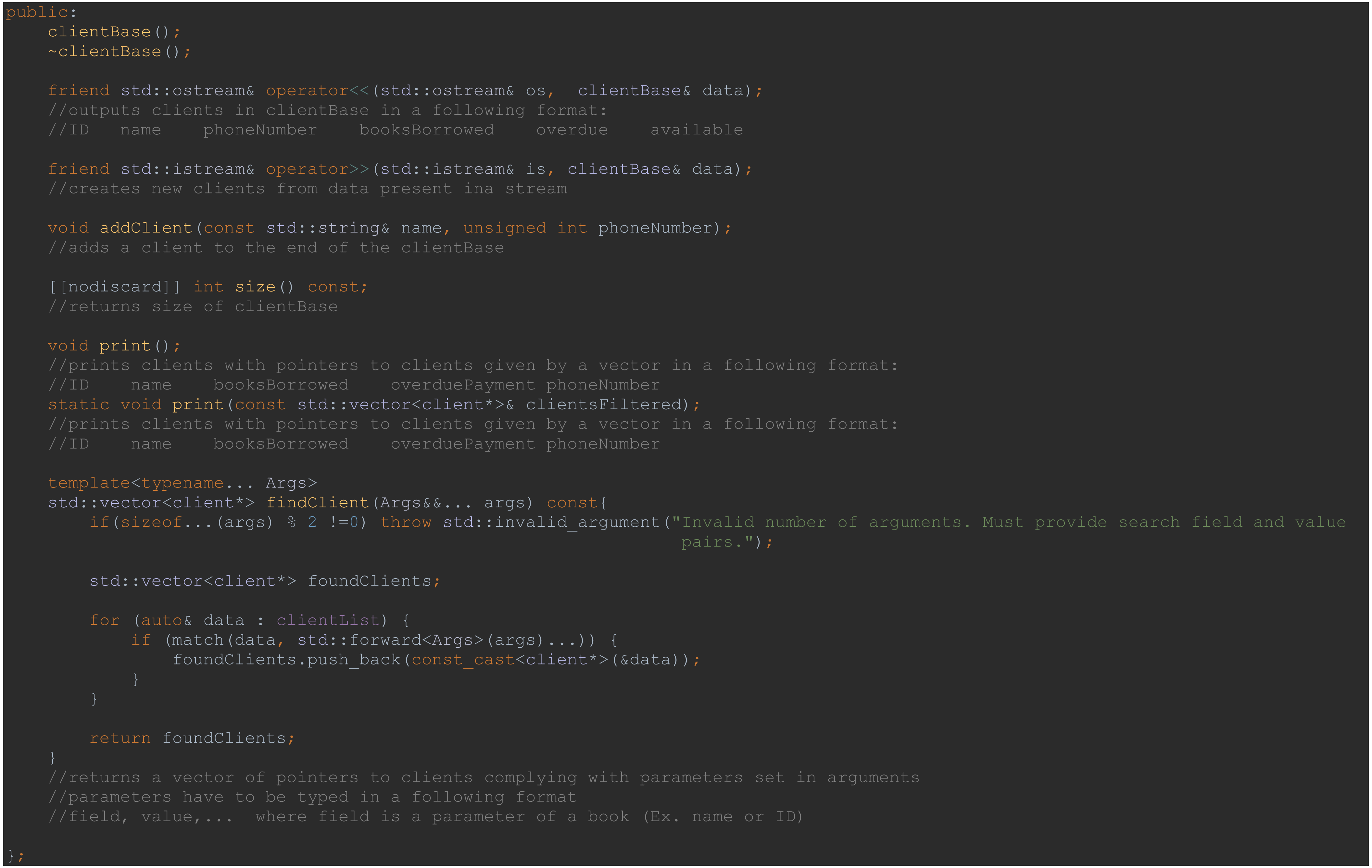
```
class client{
private:
    std::string name;   //name of a client
    unsigned int ID;    //unique identification number of a client
    unsigned int booksBorrowed;   //number of books borrowed, ranging from 0 to maximum set in library class
    float overdue;   //the amount of money owed to the library for returning the books late
    unsigned int phoneNumber;   //phone number of a client, stored as a measure of contacting a client
    bool available; //0 if a client was destroyed/deleted etc.
public:
    client(unsigned int Id, const std::string& Name, unsigned int PhoneNumber, unsigned int BooksBorrowed=0, float Overdue=0, bool
           Available=true);
    //creates a client with a given name and phone number
    //ID of a book is equal to clientID in the library
    //after creating a book 1 is added to the clientID, to keep ID unique
    //overduePayment and booksBorrowed are set to 0

    ~client();
    //deletes name string

    [[nodiscard]] std::string getName() const;
    //returns name of a client

    [[nodiscard]] unsigned int getID() const;
    //returns ID of a client

    [[nodiscard]] unsigned int getBorrowedNumber() const;
    //returns number of books borrowed by a client

    [[nodiscard]] float getOverdue() const;
    //returns overdue payment of a client

    [[nodiscard]] unsigned int getPhoneNumber() const;
    //returns phone number of a client

    void setBorrowed(int val);
    //add val to clients borrowedBooks
    //if booksBorrowed>0, or booksBorrowed<maxBorrowed
    // prints error, and sets to minimum or maximum respectively

    void setOverdue(float val);
    //sets the client's overduePayment to val
    // 0 is given as a default, because most often clients would pay all of their overdue in one go
    void setAvailable(bool val);
    //sets the client's overduePayment to val

    bool operator==(const client& other) const;
    //operator is set to compare clients
    //it compares them by their ID

    friend std::ostream& operator<<(std::ostream& os, const client& data);
    //outputs client data in a following format
    //ID    name    phoneNumber booksBorrowed   overdue

    friend std::istream& operator>>(std::istream& is, client& obj);
    //loads data into a client

    friend class library;
    friend class clientBase;
};
```

Aleksander Czachór EOOP project – keyword „Library”

- **ClientBase:**

```
class clientBase{
private:
    std::list<client> clientList;
    unsigned int clientId;

    template<typename Field>
    const Field& getField(const client& client, ClientSearch field) const;
    //returns parameter of a client instance
    //respective to a field passed in arguments

    [[nodiscard]] bool match(const client& client) const {
        return true;
    }
    //base case for recursive call of match

    template<typename First,typename... Args>
    bool match(const client& client, ClientSearch field, First&& first, Args&&... args) const{
        const auto& fieldValue = getField<std::decay_t<decltype(first)>>(client, field);

        if constexpr (std::is_same_v<std::decay_t<decltype(fieldValue)>, std::string>) {
            std::string lowercaseFieldValue = fieldValue;
            std::string lowercaseFirst = first;

            std::transform(lowercaseFieldValue.begin(), lowercaseFieldValue.end(), lowercaseFieldValue.begin(), ::tolower);
            std::transform(lowercaseFirst.begin(), lowercaseFirst.end(), lowercaseFirst.begin(), ::tolower);

            if (lowercaseFieldValue.find(lowercaseFirst) != std::string::npos) {
                return match(client, std::forward<Args>(args)...);
            }
        }
        else {
            if (fieldValue == first) {
                return match(client, std::forward<Args>(args)...);
            }
        }

        return false;
    }
    //matches a parameter of client instance with a respective field
    //returns itself recursively with the rest of arguments if match passes
```

Aleksander Czachór EOOP project – keyword „Library”



```cpp
public:
    clientBase();
    ~clientBase();

    friend std::ostream& operator<<(std::ostream& os,  clientBase& data);
    //outputs clients in clientBase in a following format:
    //ID   name    phoneNumber    booksBorrowed    overdue    available

    friend std::istream& operator>>(std::istream& is, clientBase& data);
    //creates new clients from data present ina stream

    void addClient(const std::string& name, unsigned int phoneNumber);
    //adds a client to the end of the clientBase

    [[nodiscard]] int size() const;
    //returns size of clientBase

    void print();
    //prints clients with pointers to clients given by a vector in a following format:
    //ID    name    booksBorrowed    overduePayment phoneNumber
    static void print(const std::vector<client*>& clientsFiltered);
    //prints clients with pointers to clients given by a vector in a following format:
    //ID    name    booksBorrowed    overduePayment phoneNumber

    template<typename... Args>
    std::vector<client*> findClient(Args&&... args) const{
        if(sizeof...(args) % 2 !=0) throw std::invalid_argument("Invalid number of arguments. Must provide search field and value
                                                                 pairs.");

        std::vector<client*> foundClients;

        for (auto& data : clientList) {
            if (match(data, std::forward<Args>(args)...)) {
                foundClients.push_back(const_cast<client*>(&data));
            }
        }

        return foundClients;
    }
    //returns a vector of pointers to clients complying with parameters set in arguments
    //parameters have to be typed in a following format
    //field, value,...  where field is a parameter of a book (Ex. name or ID)

};
```

Aleksander Czachór EOOP project – keyword „Library”

- **BorrowData**:

```
class borrowData{
private:
    unsigned int ID;    //ID of borrowData
    unsigned int bookID;    //ID of a borrowed book
    unsigned int clientID;  //ID of borrowing client
    tm borrowDate{};  //a time struct storing borrow date
    tm dateReturned{};  //a time struct storing return date (it's set to 0 if a book hasn't been returned yet)
public:
    borrowData(unsigned int ID, unsigned int bookId, unsigned int clientId, tm borrowDate, tm returnDate = {0,0,0,0,0,0});
    //sets borrowData's bookID to bookID, clientID to clientID, and tm borrowDate to borrowDate
    //sets ID to borrowID, then adds 1 to borrowID
    ~borrowData();
    //destructor will be empty here

    [[nodiscard]] unsigned int getID() const;          //returns ID
    [[nodiscard]] unsigned int getBookID() const;      //returns bookID
    [[nodiscard]] unsigned int getClientID() const;    //returns clientID
    [[nodiscard]] tm getBorrowDate() const;            //returns borrowDate
    [[nodiscard]] tm getReturnDate() const;            //returns returnDate

    bool operator==(const borrowData& other) const;
    //operator is set to compare borrow instances
    //compares them by their ID

    void setReturnDate(tm& date);
    //sets returnDate to given values
    //values not given (minute, second, etc.) are set to 0
    //can't set date prior to borrowDate

    friend std::ostream& operator<<(std::ostream& os, const borrowData& data);
    //outputs book data in a following format
    //ID    bookID   titleID    dd.mm.yyyy(borrowed) dd.mm.yyyy(returned, 0 if not returned yet)
    friend std::istream& operator>>(std::istream& is, borrowData& obj);
    //operator loading data into a borrowData object

    friend class borrowRecords;
    friend class library;
};
```

Aleksander Czachór EOOP project – keyword „Library”

- **BorrowRecords:**

```cpp
class borrowRecords {
private:
    std::list<borrowData> borrowInstances;
    unsigned int borrowId;

    template <typename Field>
    const Field& getField(const borrowData& borrowInstance, BorrowSearch field);
    //returns parameter of a borrow instance
    //respective to a field passed in arguments

    [[nodiscard]] bool match(const borrowData& borrowInstance)const;
    //base case for recursive match call
    template <typename T, typename... Args>
    bool match(const borrowData& borrowInstance, BorrowSearch field, const T&& first, const Args&... args) {
        const auto& newField = getField<std::decay_t<decltype(first)>>(borrowInstance, field);

        if (typeid(first)==typeid(unsigned int)) {
            if (first == newField)
                return match(borrowInstance, std::forward<Args>(args)...);
        }
        if constexpr (std::is_same_v<T, tm>) {
            const auto& borrowDate = borrowInstance.borrowDate;
            if (field == BorrowSearch::BorrowDate) {
                if (borrowDate.tm_year == first.tm_year && borrowDate.tm_mon == first.tm_mon && borrowDate.tm_mday ==
first.tm_mday)
                    return match(borrowInstance, std::forward<Args>(args)...);
            }
            else if (field == BorrowSearch::ReturnDate) {
                const auto& returnDate = borrowInstance.dateReturned;
                if (returnDate.tm_year == first.tm_year && returnDate.tm_mon == first.tm_mon && returnDate.tm_mday ==
first.tm_mday)
                    return match(borrowInstance, std::forward<Args>(args)...);
            }
        }
        return false;
    }
    //matches a parameter of borrow instance with a respective field
    //returns itself recursively with the rest of arguments if match passes
```

Aleksander Czachór EOOP project – keyword „Library”

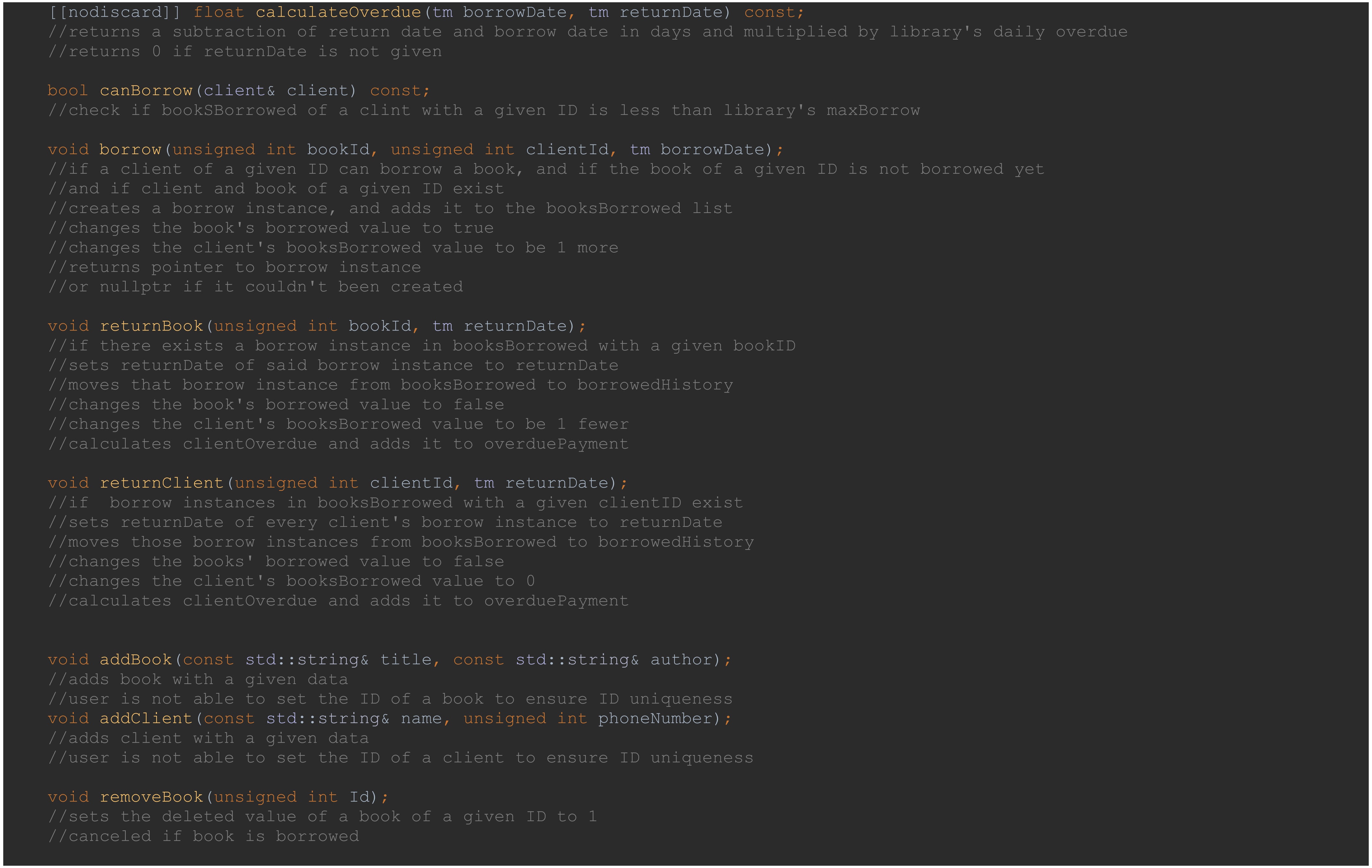
```cpp
public:
    borrowRecords();
    ~borrowRecords();
    friend std::ostream& operator<<(std::ostream& os, borrowRecords& data);
    //outputs books in bookList in a following format:
    //ID    bookID   clientID    dateBorrowed    dateReturned
    friend std::istream& operator>>(std::istream& is, borrowRecords& data);
    //creates borrow instances with data passed in the stream

    [[nodiscard]] unsigned int getID() const;   //returns ID assigning unique borrows
    [[nodiscard]] int size() const;   //returns size of borrowInstances

    void print();
    //prints list with pointers to borrow instances given by a vector in a following format:
    //ID    bookID    clientID   dateBorrowed    dateReturned
    static void print(const std::vector<borrowData*>& borrowDataFiltered);
    //prints list with pointers to borrow instances given by a vector in a following format:
    //ID    bookID    clientID   dateBorrowed dateReturned

    void removeBorrow(const borrowData& data);
    //removes certain borrow instance from borrowInstances list

    template <typename... Args>
    std::vector<borrowData*> findBorrowed(Args&&... args) {
        if(sizeof...(args) % 2 !=0) throw std::invalid_argument("Invalid number of arguments. Must provide search field and value
                                                               pairs.");
        std::vector<borrowData*> borrowFiltered;
        for (auto& it : borrowInstances) {
            if (match(it, std::forward<Args>(args)...)) {
                borrowFiltered.push_back(const_cast<borrowData*>(&it));
            }
        }
        return borrowFiltered;
    }
    //returns a vector of pointers to borrows complying with parameters set in arguments
    //parameters have to be typed in a following format
    //field, value,...  where field is a parameter of a book (Ex. dateReturned or ID)

    void addBorrow(unsigned int bookId,unsigned int clientId, tm borrowDate, tm returnDate={0,0,0,0,0,0});
    //adds new borrow Instance to the back of borrowInstances list
    void addBorrow(unsigned int ID, unsigned int bookId,unsigned int clientId, tm borrowDate, tm returnDate={0,0,0,0,0,0});
    //adds new borrow Instance with a given ID to the back of borrowInstances list
    //used when loading borrows from a file
};
```

Aleksander Czachór EOOP project – keyword „Library”

- **Library**:

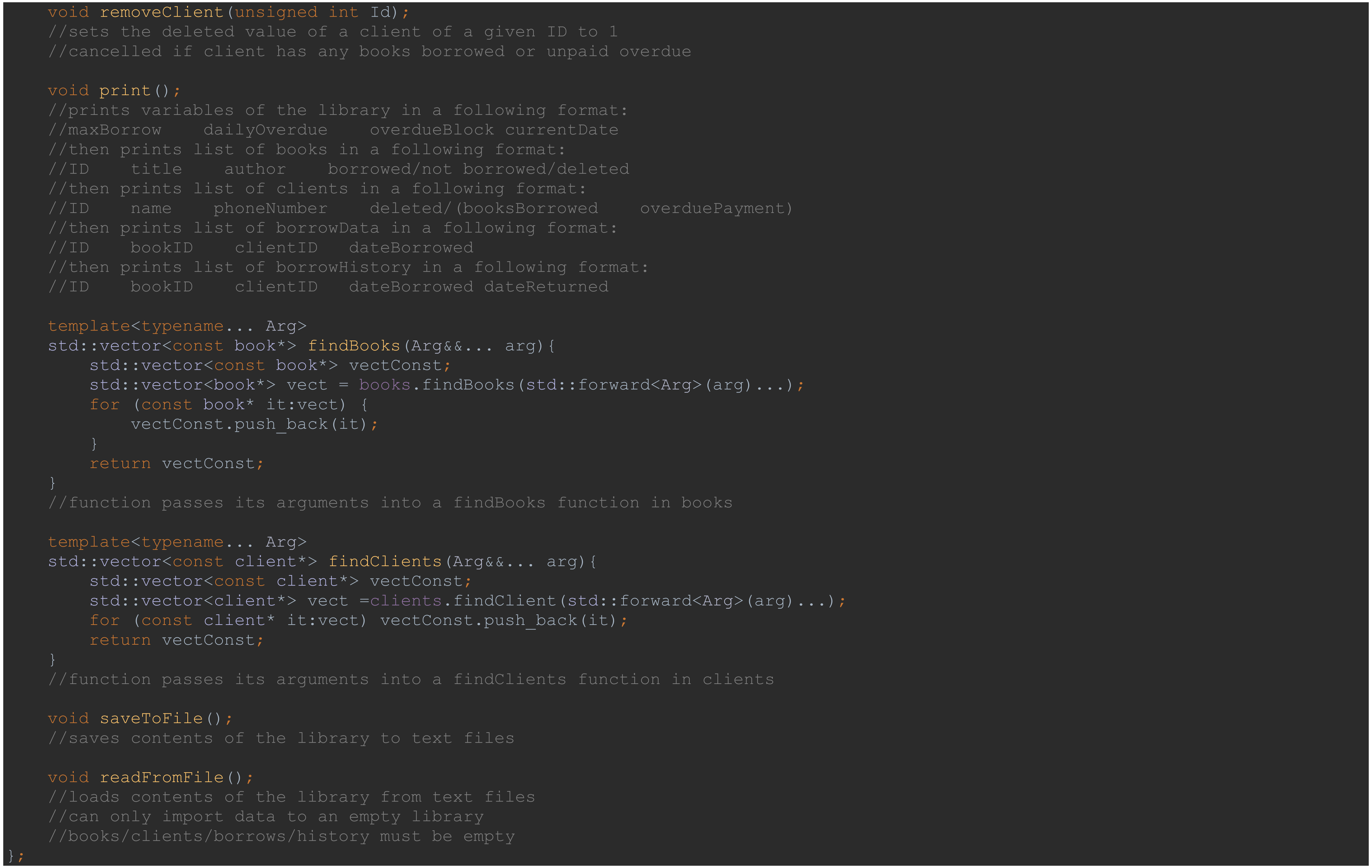
```
class library {
private:
    unsigned int maxBorrow; //maximum books a client can borrow simultaneously
    float dailyOverdue; //overdue amount for one day too long of keeping a book
    float overdueBlock; //maximum amount of overdue, after which a client cannot borrow a book
    tm currentDate{}; //time struct of current date in the library
    bookCollection books; //books in a library
    clientBase clients; //clientBase in a library
    borrowRecords borrows; //current borrows in a library
    borrowRecords history; //past borrows in a library
public:
    explicit library(int maxBorrows=3, float dailyAmount=0.3, float maxOverdue=5);
    //sets the maxBorrow, dailyOverdue and maxOverdue of a library
    ~library();
    //deletes all lists in a library

    [[nodiscard]]  unsigned int getMaxBorrow() const;
    //returns maxBorrow of the library

    [[nodiscard]] int getSize()const;

    void setMaxBorrow(int maxBorrow);
    //sets the maxBorrow of the library

    [[nodiscard]]  float getDailyOverdue() const;
    //returns daily overdue

    void setDailyOverdue(float value);
    //sets dailyOverdue to a given value

    [[nodiscard]]  float getOverdueBlock() const;
    //returns overdue block

    void setOverdueBlock(float value);
    //sets overdueBlock to a given value

    [[nodiscard]] tm getCurrentDate();
    //returns a time struct of a currentDate

    void setCurrentDate(unsigned int day,unsigned int month,unsigned int yearAfter1900);
    //sets currentDate to a given value
```

Aleksander Czachór EOOP project – keyword „Library”



```cpp
    [[nodiscard]] float calculateOverdue(tm borrowDate, tm returnDate) const;
    //returns a subtraction of return date and borrow date in days and multiplied by library's daily overdue
    //returns 0 if returnDate is not given

    bool canBorrow(client& client) const;
    //check if bookSBorrowed of a clint with a given ID is less than library's maxBorrow

    void borrow(unsigned int bookId, unsigned int clientId, tm borrowDate);
    //if a client of a given ID can borrow a book, and if the book of a given ID is not borrowed yet
    //and if client and book of a given ID exist
    //creates a borrow instance, and adds it to the booksBorrowed list
    //changes the book's borrowed value to true
    //changes the client's booksBorrowed value to be 1 more
    //returns pointer to borrow instance
    //or nullptr if it couldn't been created

    void returnBook(unsigned int bookId, tm returnDate);
    //if there exists a borrow instance in booksBorrowed with a given bookID
    //sets returnDate of said borrow instance to returnDate
    //moves that borrow instance from booksBorrowed to borrowedHistory
    //changes the book's borrowed value to false
    //changes the client's booksBorrowed value to be 1 fewer
    //calculates clientOverdue and adds it to overduePayment

    void returnClient(unsigned int clientId, tm returnDate);
    //if  borrow instances in booksBorrowed with a given clientID exist
    //sets returnDate of every client's borrow instance to returnDate
    //moves those borrow instances from booksBorrowed to borrowedHistory
    //changes the books' borrowed value to false
    //changes the client's booksBorrowed value to 0
    //calculates clientOverdue and adds it to overduePayment


    void addBook(const std::string& title, const std::string& author);
    //adds book with a given data
    //user is not able to set the ID of a book to ensure ID uniqueness
    void addClient(const std::string& name, unsigned int phoneNumber);
    //adds client with a given data
    //user is not able to set the ID of a client to ensure ID uniqueness

    void removeBook(unsigned int Id);
    //sets the deleted value of a book of a given ID to 1
    //canceled if book is borrowed
```

Aleksander Czachór EOOP project – keyword „Library”



```cpp
    void removeClient(unsigned int Id);
    //sets the deleted value of a client of a given ID to 1
    //cancelled if client has any books borrowed or unpaid overdue

    void print();
    //prints variables of the library in a following format:
    //maxBorrow    dailyOverdue    overdueBlock currentDate
    //then prints list of books in a following format:
    //ID    title    author    borrowed/not borrowed/deleted
    //then prints list of clients in a following format:
    //ID    name    phoneNumber    deleted/(booksBorrowed    overduePayment)
    //then prints list of borrowData in a following format:
    //ID    bookID    clientID   dateBorrowed
    //then prints list of borrowHistory in a following format:
    //ID    bookID    clientID   dateBorrowed dateReturned

    template<typename... Arg>
    std::vector<const book*> findBooks(Arg&&... arg){
        std::vector<const book*> vectConst;
        std::vector<book*> vect = books.findBooks(std::forward<Arg>(arg)...);
        for (const book* it:vect) {
            vectConst.push_back(it);
        }
        return vectConst;
    }
    //function passes its arguments into a findBooks function in books

    template<typename... Arg>
    std::vector<const client*> findClients(Arg&&... arg){
        std::vector<const client*> vectConst;
        std::vector<client*> vect =clients.findClient(std::forward<Arg>(arg)...);
        for (const client* it:vect) vectConst.push_back(it);
        return vectConst;
    }
    //function passes its arguments into a findClients function in clients

    void saveToFile();
    //saves contents of the library to text files

    void readFromFile();
    //loads contents of the library from text files
    //can only import data to an empty library
    //books/clients/borrows/history must be empty
};
```

Aleksander Czachór EOOP project – keyword „Library”

- **Exception overloads:**

```cpp
class InvalidID : public std::exception {
public:
    InvalidID(std::string  message, int argumentValue)
            : errorMessage(std::move(message)), argument(argumentValue) {}

    [[nodiscard]] const char* what() const noexcept override {
        return errorMessage.c_str();
    }
private:
    std::string errorMessage;
    int argument;
};

class MaxReached : public std::exception {
public:
    MaxReached(std::string error, int booksBorrowed,std::string Allowed,  int maxBorrow)
            : error(std::move(error)), value(booksBorrowed), allowed(std::move(Allowed)),  max(maxBorrow) {}

    [[nodiscard]] const char* what() const noexcept override {
        return generateErrorMessage().c_str();
    }

private:
    int max;
    int value;
    std::string allowed;
    std::string error;
    [[nodiscard]] std::string generateErrorMessage() const {
        std::stringstream ss;
        ss << error << value << allowed << max;
        return ss.str();
    }

};
```

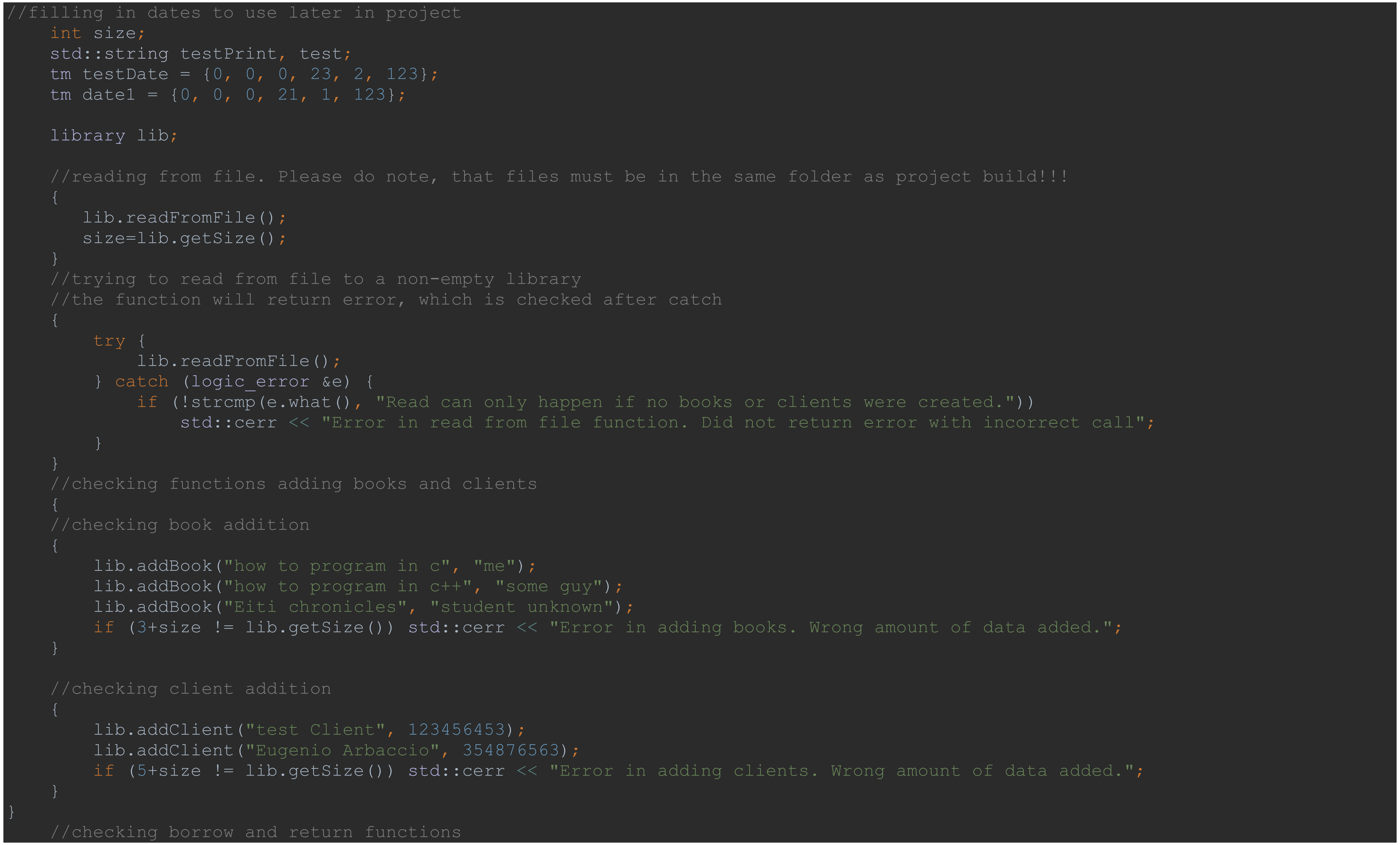
Aleksander Czachór EOOP project – keyword „Library”

- **Enumeration classes for find functions:**

```
enum class BookSearch {
    ID,
    Title,
    Author,
    Borrowed,
    Available
};
enum class ClientSearch{
    ID,
    Name,
    PhoneNumber,
    BooksBorrowed,
    Overdue,
    Available
};
enum class BorrowSearch {
    ID,
    BookID,
    ClientID,
    BorrowDate,
    ReturnDate
};
```

Aleksander Czachór EOOP project – keyword „Library”

4. Testing

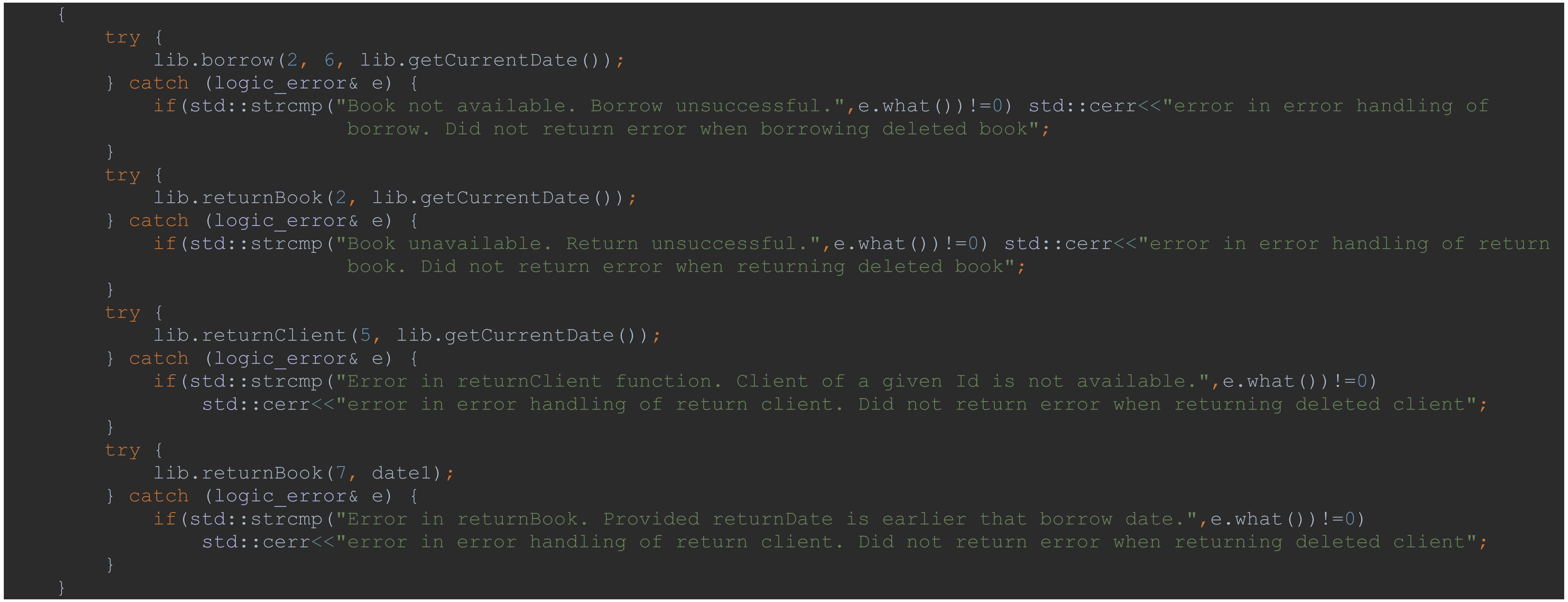


```
//filling in dates to use later in project
    int size;
    std::string testPrint, test;
    tm testDate = {0, 0, 0, 23, 2, 123};
    tm date1 = {0, 0, 0, 21, 1, 123};

    library lib;

    //reading from file. Please do note, that files must be in the same folder as project build!!!
    {
       lib.readFromFile();
       size=lib.getSize();
    }
    //trying to read from file to a non-empty library
    //the function will return error, which is checked after catch
    {
        try {
            lib.readFromFile();
        } catch (logic_error &e) {
            if (!strcmp(e.what(), "Read can only happen if no books or clients were created."))
                std::cerr << "Error in read from file function. Did not return error with incorrect call";
        }
    }
    //checking functions adding books and clients
    {
    //checking book addition
    {
        lib.addBook("how to program in c", "me");
        lib.addBook("how to program in c++", "some guy");
        lib.addBook("Eiti chronicles", "student unknown");
        if (3+size != lib.getSize()) std::cerr << "Error in adding books. Wrong amount of data added.";
    }

    //checking client addition
    {
        lib.addClient("test Client", 123456453);
        lib.addClient("Eugenio Arbaccio", 354876563);
        if (5+size != lib.getSize()) std::cerr << "Error in adding clients. Wrong amount of data added.";
    }
}
    //checking borrow and return functions
```

Aleksander Czachór EOOP project – keyword „Library”

```
    {
        lib.borrow(6, 3, testDate);
        lib.borrow(7, 4, testDate);
        lib.returnBook(1, lib.getCurrentDate());
        lib.borrow(1, 3, lib.getCurrentDate());
        lib.returnClient(3,lib.getCurrentDate());
        if (8+size != lib.getSize()) std::cerr << "Error in borrow function. Wrong amount of data added.";
    }
    //checking save to file. Function saves to another files, for the sake of repeating tests with same results
    lib.saveToFile();
    //checking remove functions
    {
        lib.removeBook(2);
        lib.removeClient(5);
        if(lib.findBooks(BookSearch::Available, false).empty()) std::cerr<<"remove book did not remove any book.";
        if(lib.findBooks(BookSearch::Available, false).size()!=2) std::cerr<<"remove book did not remove any book.";
    }
    //checking find functions
    {
        if(lib.findBooks(BookSearch::ID, 5).front()->getID()!=5)std::cerr<<"Error in findBooks function. Did not search by
                                                                           ID correctly";
        if(lib.findBooks(BookSearch::Borrowed, true).size()!=1)std::cerr<<"Error in findBooks function. Did not search by
                                                                       borrowed correctly";
        test="c++";
        if(lib.findBooks(BookSearch::Title, test).size()!=3)std::cerr<<"Error in findBooks function. Did not search by title
                                                                    correctly";
        test="author";
        if(lib.findBooks(BookSearch::Author, test).size()!=1)std::cerr<<"Error in findBooks function. Did not search by author
                                                                     correctly";

        if(lib.findClients(ClientSearch::ID, 1).front()->getID()!=1)std::cerr<<"Error in findClients function. Did not search
                                                                            by ID correctly";
        if(lib.findClients(ClientSearch::PhoneNumber, 123456789).size()!=1)std::cerr<<"Error in findClients function. Did not
                                                                                   search by phone Number correctly";
        if(lib.findClients(ClientSearch::BooksBorrowed, 3).size()!=1)std::cerr<<"Error in findClients function. Did not search
                                                                             by books borrowed correctly";
        if(lib.findClients(ClientSearch::Available, false).size()!=2)std::cerr<<"Error in findClients function. Did not
                                                                             searchby available correctly";
        test="spack jarrow";
        if(lib.findClients(ClientSearch::Name, test).size()!=1)std::cerr<<"Error in findClients function. Did not search by
                                                                       name correctly";
    }

    //a few incorrect cases
```

Aleksander Czachór EOOP project – keyword „Library”



```
    {
        try {
            lib.borrow(2, 6, lib.getCurrentDate());
        } catch (logic_error& e) {
            if(std::strcmp("Book not available. Borrow unsuccessful.",e.what())!=0) std::cerr<<"error in error handling of
                        borrow. Did not return error when borrowing deleted book";
        }
        try {
            lib.returnBook(2, lib.getCurrentDate());
        } catch (logic_error& e) {
            if(std::strcmp("Book unavailable. Return unsuccessful.",e.what())!=0) std::cerr<<"error in error handling of return
                        book. Did not return error when returning deleted book";
        }
        try {
            lib.returnClient(5, lib.getCurrentDate());
        } catch (logic_error& e) {
            if(std::strcmp("Error in returnClient function. Client of a given Id is not available.",e.what())!=0)
                std::cerr<<"error in error handling of return client. Did not return error when returning deleted client";
        }
        try {
            lib.returnBook(7, date1);
        } catch (logic_error& e) {
            if(std::strcmp("Error in returnBook. Provided returnDate is earlier that borrow date.",e.what())!=0)
                std::cerr<<"error in error handling of return client. Did not return error when returning deleted client";
        }
    }
```