Library Management System

Generated by Doxygen 1.10.0

Chapter 1

Library Management System (LMS)

A C++ based Library Management System designed to efficiently manage library resources, track book availability, and handle user interactions.

Library Management System will provide capabilities to manage entire library by a single (or multiple) librarian. It will provide options to issue books to students and teacher. LMS will make librarian encharge of whole library.

1.1 Features

- Command line interface
- Add, remove, and update book records by librarians
- Search and filter books by various criteria
- User authentication for librarians
- Borrow and return book functionality
- Different criteria for students and teachers

1.2 Installation

Clone the repository:
```
git clone https://github.com/david-ptrk/Library-Management-System.git
```

Compile on computer:
```
cd Library-Management-System

g++ test.cpp -o library_management_system
```

Run:
```
./library_management_system
```

1.3 Usage

```
1. Launch the application.
2. Log in as a librarian.
3. Explore the librarian features such as adding new librarian, adding new books, removing old books, and
      checking students' details.
3. Explore the public features such as borrowing a book, returing a book, renewing return date, and search a
      book.
```

1.4 Screenshots

1.5 Acknowledgments

- `ReadMe.so` - Simplifying README creation.
- `Doxygen Website` - Creating Documentation.

1.6 Authors

- Daud Anjum
- Ahmad Azhar

1.7 Contact

For inquiries, please contact Daud Anjum at `davidptrk7@gmail.com`.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

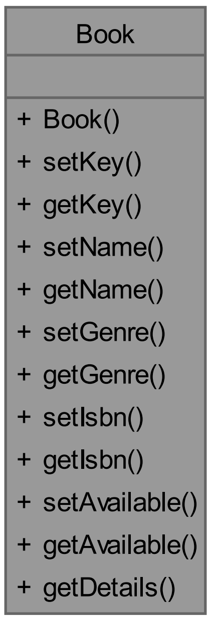
Chapter 5

Class Documentation

5.1 Book Class Reference

```
#include <book.hpp>
```

Collaboration diagram for Book:



Public Member Functions

- Book (int keyV=-1, std::string isbn="", std::string name="", std::string genre="", bool availableV=false)
- void setKey (int num)

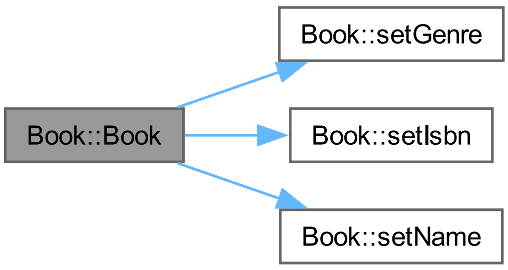
- int getKey () const
- void setName (std::string bookName)
- std::string getName () const
- void setGenre (std::string bookGenre)
- std::string getGenre () const
- void setIsbn (std::string isbnum)
- std::string getIsbn () const
- void setAvailable (bool n)
- bool getAvailable () const
- std::string getDetails () const

5.1.1 Constructor & Destructor Documentation

5.1.1.1 Book()

```
Book::Book (
             int keyV = -1,
             std::string isbn = "",
             std::string name = "",
             std::string genre = "",
             bool availableV = false )  [inline]
```

Here is the call graph for this function:

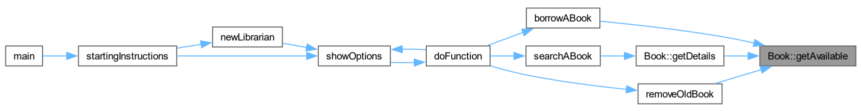


5.1.2 Member Function Documentation

5.1.2.1 getAvailable()

```
bool Book::getAvailable ( ) const  [inline]
```

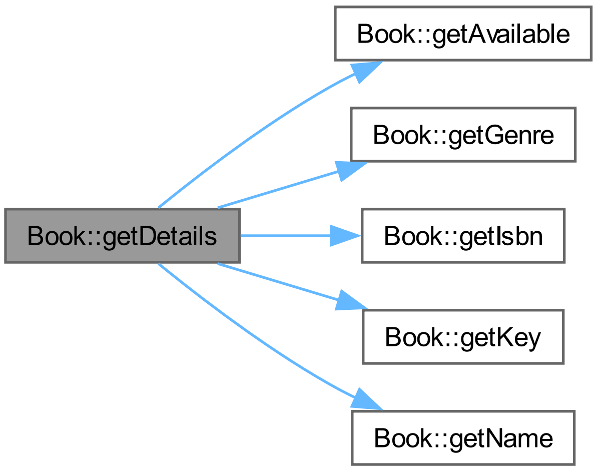
Here is the caller graph for this function:



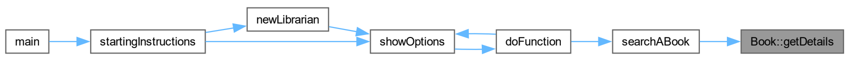
5.1.2.2 getDetails()

```
std::string Book::getDetails ( ) const  [inline]
```

Here is the call graph for this function:



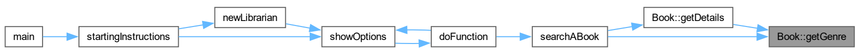
Here is the caller graph for this function:



5.1.2.3 getGenre()

```
std::string Book::getGenre ( ) const  [inline]
```

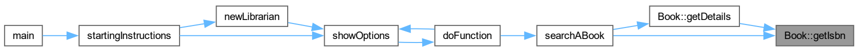
Here is the caller graph for this function:



5.1.2.4 getIsbn()

```
std::string Book::getIsbn ( ) const  [inline]
```

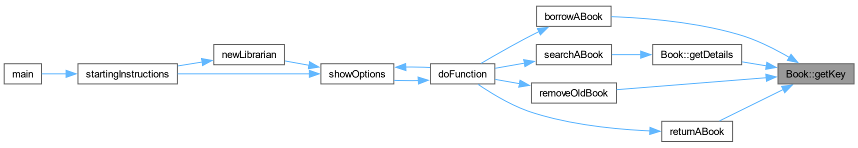
Here is the caller graph for this function:



5.1.2.5 getKey()

```
int Book::getKey ( ) const  [inline]
```

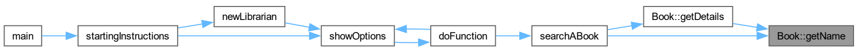
Here is the caller graph for this function:



5.1.2.6 getName()

```
std::string Book::getName ( ) const  [inline]
```

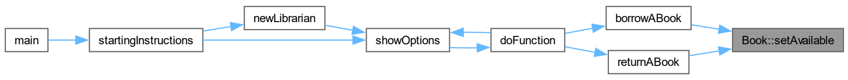
Here is the caller graph for this function:



5.1.2.7 setAvailable()

```
void Book::setAvailable (
            bool n )  [inline]
```

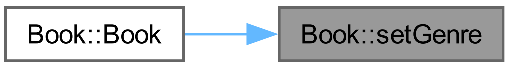
Here is the caller graph for this function:



5.1.2.8 setGenre()

```
void Book::setGenre (
             std::string bookGenre )  [inline]
```

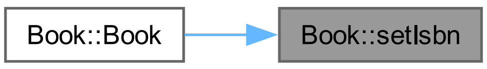
Here is the caller graph for this function:



5.1.2.9 setIsbn()

```
void Book::setIsbn (
             std::string isbnum )  [inline]
```

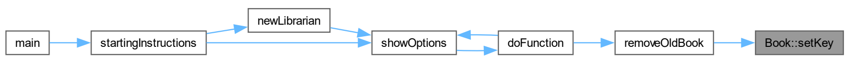
Here is the caller graph for this function:



5.1.2.10 setKey()

```
void Book::setKey (
             int num )  [inline]
```

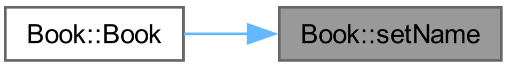
Here is the caller graph for this function:



5.1.2.11 setName()

```
void Book::setName (
             std::string bookName )  [inline]
```

Here is the caller graph for this function:



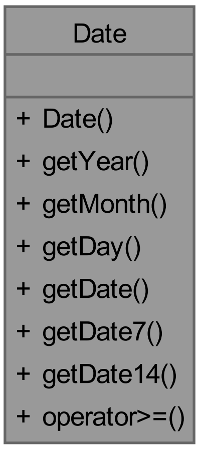
The documentation for this class was generated from the following file:

- book.hpp

5.2 Date Class Reference

```
#include <utilitys.hpp>
```

Collaboration diagram for Date:



Public Member Functions

- Date ()
- int getYear () const
- int getMonth () const
- int getDay () const
- std::string getDate () const
- std::string getDate7 () const
- std::string getDate14 () const
- bool operator>= (const Date &other) const

Friends

- std::istream & operator>> (std::istream &, Date &)

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Date()

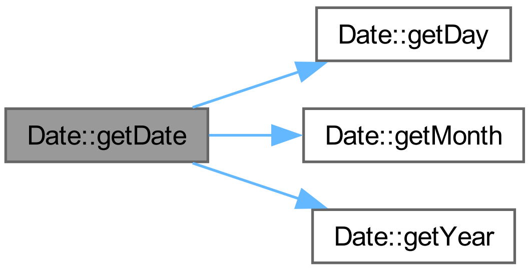
```
Date::Date ( )  [inline]
```

5.2.2 Member Function Documentation

5.2.2.1 getDate()

```
std::string Date::getDate ( ) const  [inline]
```

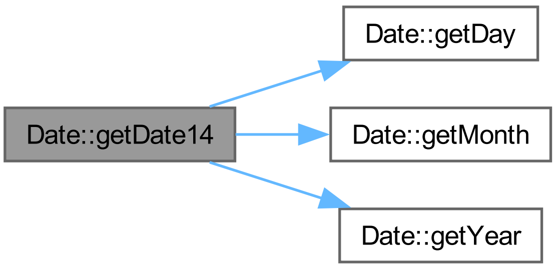
Here is the call graph for this function:



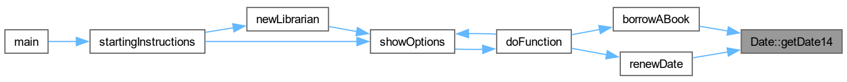
5.2.2.2 getDate14()

```
std::string Date::getDate14 ( ) const  [inline]
```

Here is the call graph for this function:



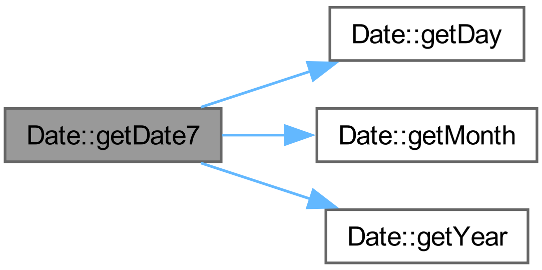
Here is the caller graph for this function:



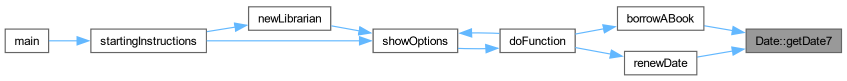
5.2.2.3 getDate7()

```
std::string Date::getDate7 ( ) const  [inline]
```

Here is the call graph for this function:



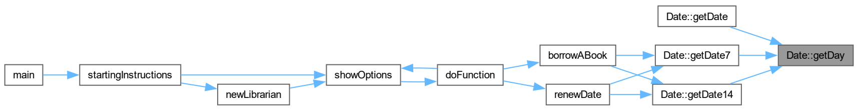
Here is the caller graph for this function:



5.2.2.4 getDay()

```
int Date::getDay ( ) const  [inline]
```

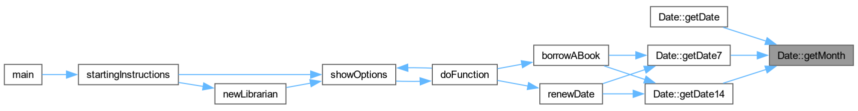
Here is the caller graph for this function:



5.2.2.5 getMonth()

```
int Date::getMonth ( ) const  [inline]
```

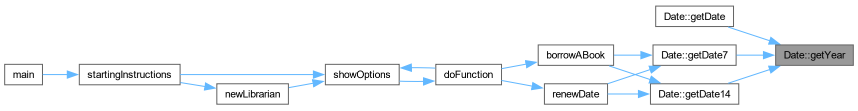
Here is the caller graph for this function:



5.2.2.6 getYear()

```
int Date::getYear ( ) const  [inline]
```

Here is the caller graph for this function:



5.2.2.7 operator>=()

```
bool Date::operator>= (
            const Date & other ) const  [inline]
```

5.2.3 Friends And Related Symbol Documentation

5.2.3.1 operator>>

```
std::istream & operator>> (
            std::istream & input,
            Date & obj )  [friend]
```

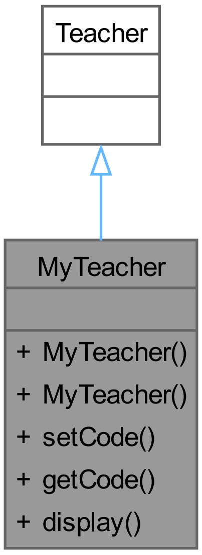
The documentation for this class was generated from the following file:

- utilitys.hpp

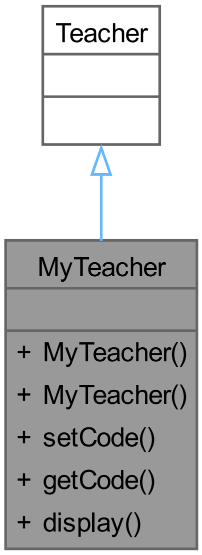
5.3 MyTeacher Class Reference

```
#include <utilitys.hpp>
```

Inheritance diagram for MyTeacher:



Collaboration diagram for MyTeacher:



Public Member Functions

- MyTeacher ()=default
- MyTeacher (const std::string first, const std::string last, int age, const std::string card, bool gen, const std↩ ::string phone, const std::string department, const std::string rank, int code)
- void setCode (int v)
- int getCode () const
- std::string display () const

Friends

- std::ostream & operator<< (std::ostream &, const MyTeacher &)
- std::istream & operator>> (std::istream &, MyTeacher &)

5.3.1 Constructor & Destructor Documentation

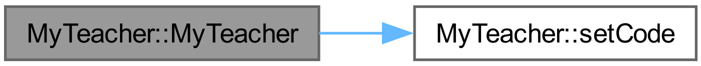
5.3.1.1 MyTeacher() [1/2]

```
MyTeacher::MyTeacher ( )  [default]
```

5.3.1.2 MyTeacher() [2/2]

```
MyTeacher::MyTeacher (
             const std::string first,
             const std::string last,
             int age,
             const std::string card,
             bool gen,
             const std::string phone,
             const std::string department,
             const std::string rank,
             int code )  [inline]
```

Here is the call graph for this function:

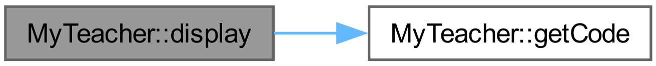


5.3.2 Member Function Documentation

5.3.2.1 display()

```
std::string MyTeacher::display ( ) const  [inline]
```

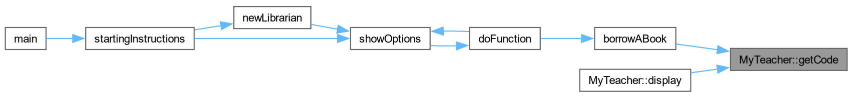
Here is the call graph for this function:



5.3.2.2 getCode()

```
int MyTeacher::getCode ( ) const  [inline]
```

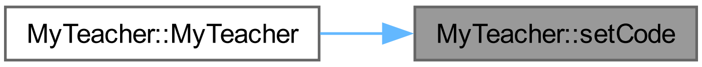
Here is the caller graph for this function:



5.3.2.3 setCode()

```
void MyTeacher::setCode (
             int v )  [inline]
```

Here is the caller graph for this function:



5.3.3 Friends And Related Symbol Documentation

5.3.3.1 operator<<

```
std::ostream & operator<< (
             std::ostream & output,
             const MyTeacher & teacher )  [friend]
```

5.3.3.2 operator>>

```
std::istream & operator>> (
             std::istream & input,
             MyTeacher & teacher )  [friend]
```

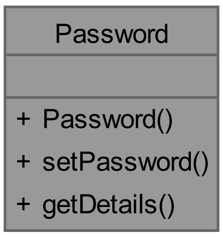
The documentation for this class was generated from the following file:

- utilitys.hpp

5.4 Password Class Reference

```
#include <encryption.hpp>
```

Collaboration diagram for Password:



Public Member Functions

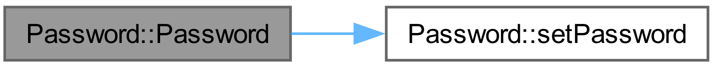
- Password (std::string name, std::string password)
- void setPassword (std::string value)
- std::string getDetails (void) const

5.4.1 Constructor & Destructor Documentation

5.4.1.1 Password()

```
Password::Password (
            std::string name,
            std::string password )  [inline]
```

Here is the call graph for this function:

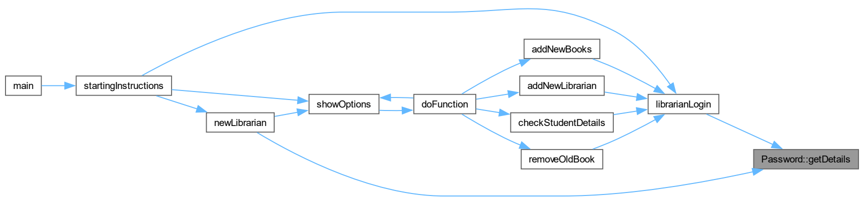


5.4.2 Member Function Documentation

5.4.2.1 getDetails()

```
std::string Password::getDetails (
            void  ) const  [inline]
```

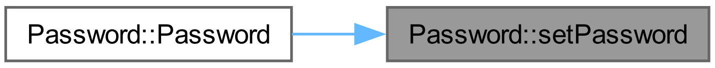
Here is the caller graph for this function:



5.4.2.2 setPassword()

```
void Password::setPassword (
             std::string value )  [inline]
```

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

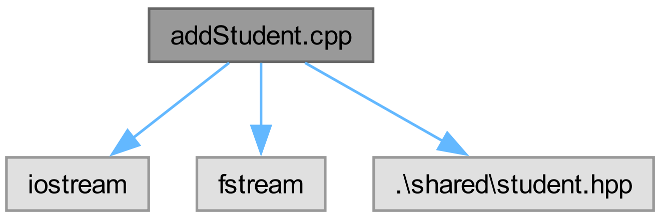
- encryption.hpp

Chapter 6

File Documentation

6.1 addStudent.cpp File Reference

```
#include <iostream>
#include <fstream>
#include ".\shared\student.hpp"
```
Include dependency graph for addStudent.cpp:



Functions

- int main ()

6.1.1 Function Documentation

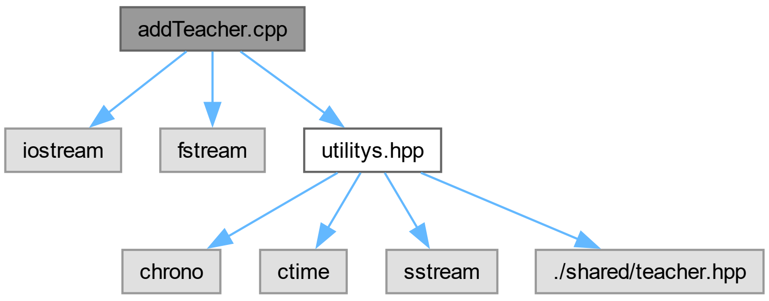
6.1.1.1 main()

```
int main ( )
```

6.2 addTeacher.cpp File Reference

```
#include <iostream>
#include <fstream>
#include "utilitys.hpp"
```

Include dependency graph for addTeacher.cpp:



Functions

- int main ()

6.2.1 Function Documentation

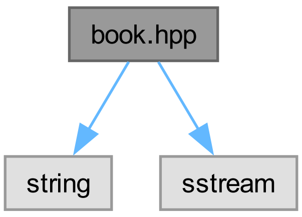
6.2.1.1 main()

```
int main ( )
```

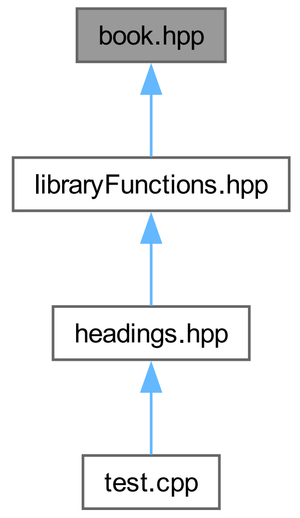
6.3 book.hpp File Reference

```
#include <string>
#include <sstream>
```

Include dependency graph for book.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Book

6.4 book.hpp

Go to the documentation of this file.

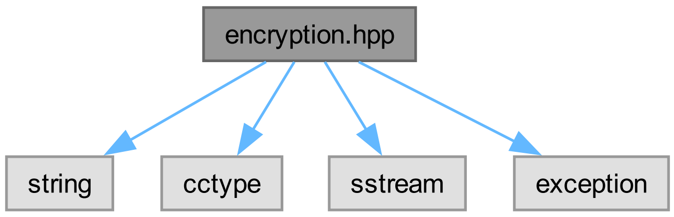
```
// This header file consist of class 'Book' that will hold book information before it is wrote to file
or after it is read from file. Class is necessary to hold book information since this information is
written in binary file with fixed length for each record i.e. book

#ifndef BOOK_HPP
#define BOOK_HPP

// preprocessor directives
#include <string>
#include <sstream>

class Book
{
public:
    // constructor provide default value for all attributes
    // it help to make object without providing autual data
    Book(int keyV = -1, std::string isbn = "", std::string name = "", std::string genre = "", bool
availableV = false)
        : key{keyV}, available{availableV} {
        // functions must be call since 'string' must be converted to 'C-typed string'
        setIsbn(isbn);
        setName(name);
        setGenre(genre);
    }

    // for data memeber 'key'
    void setKey(int num)
    {
        this->key = num; // store key
    }
    int getKey() const
    {
        return key; // return key
    }

    // for data member 'name'
    void setName(std::string bookName)
    {
        size_t length = bookName.size(); // get size of string
        length = length < 100 ? length : 99; // size must be one less then size of 'name'
        bookName.copy(name, length); // copy characters from string to 'name'
        name[length] = '\0'; // add terminating letter at end of 'name'
    }
    std::string getName() const
    {
        std::string bookName{name}; // convert 'name' array to string
        return bookName; // return as string
    }

    // for data member 'genre'
    void setGenre(std::string bookGenre)
    {
        size_t length = bookGenre.size(); // get size of string
        length = length < 30 ? length : 29; // size must be one less then size of 'genre'
        bookGenre.copy(genre, length); // copy characters from string to 'genre'
        genre[length] = '\0'; // add terminating letter at end of 'genre'
    }
    std::string getGenre() const
    {
        std::string bookGenre{genre}; // convert 'genre' array to string
        return bookGenre; // return as string
    }

    // for data member 'isbn'
    void setIsbn(std::string isbnum)
    {
        size_t length = isbnum.size(); // get size of string
        length = length < 14 ? length : 13; // size must be one less then size of 'isbn'
        isbnum.copy(isbn, length); // copy characters from string to 'isbn'
        isbn[length] = '\0'; // add terminating letter at end of 'isbn'
    }
    std::string getIsbn() const
    {
        std::string bookIsbn{isbn}; // convert 'isbn' array to string
        return bookIsbn; // return as string
    }

    // for data member 'available'
    void setAvailable(bool n)
    {
        available = n;
    }
    bool getAvailable() const
    {
        // this tell if book is available for issuing it to people
        return available;
    }

    // to concatenate all data
    std::string getDetails() const
    {
        std::ostringstream output;
        output « getKey() « ' ' « getName() « ' ' « getIsbn() « ' ' « getGenre() « ' ' «
getAvailable(); // concatenate all member with spaces in between

        return output.str(); // return as string
    }
private:
    int key; // unique key of book
    bool available; // availability of book

    // 'c-type strings' are used since it is fixed sized, and fixed size is necessary for writing
records in binary file
    char isbn[14]; // isbn of book
    char name[100]; // name of book
    char genre[30]; // genre of book
};

#endif
```

6.5 encryption.hpp File Reference

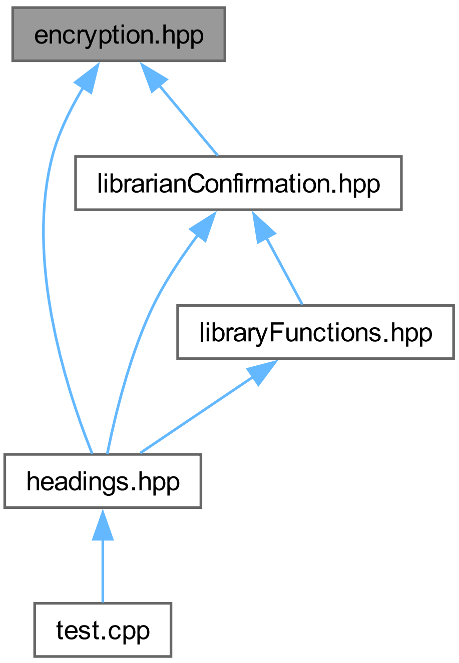
```
#include <string>
#include <cctype>
```

```
#include <sstream>
#include <exception>
```

Include dependency graph for encryption.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Password

6.6 encryption.hpp

Go to the documentation of this file.

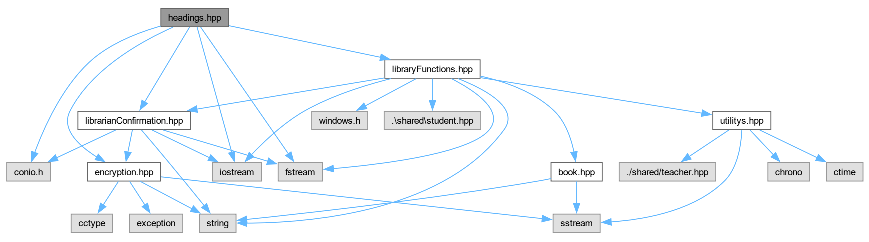
```
// This header file consist of class Password. This class is used to encrpyt user password. As an
object of class Password is instantiated, the password is already stored in encrypted form.

#ifndef ENCRYPTION_HPP
#define ENCRYPTION_HPP

// preprocessor directives
#include <string>
#include <cctype>
#include <sstream>
#include <exception>

// class will encrypt the password entered by librarian
// if return name and encryted password concatenated
class Password
{
public:
    // contructor
    Password(std::string name, std::string password) // cannot instantiate object without name and
password supplied
        : librarianName{name} {
            setPassword(password); // call 'setPassword' to encrypt it
        }

    // setPassword can be accessed outside of class
    void setPassword(std::string value)
    {
        encryptPassword(value); // call 'encryptPassword' to encrypt the password
        this->password = value; // store in data member 'password'

        return; // return control
    }

    // to return encryted password attached with user name
    std::string getDetails(void) const
    {
        std::ostringstream output;
        output « librarianName « password; // concatenate both

        return output.str(); // return them as string
    }
private:
    // data member to store password
    // password will be store in encrypted form only
    std::string password;

    // data member to store name
    std::string librarianName;

    // shift key
    // it is used to encrypt the password
    const int SHIFT_VALUE{7};

    // utility functions
    void encryptPassword(std::string& value) // function to shift every letter in password
    {
        // loop to interate over every character of 'value' i.e. password entered by user
        for(char& element : value)
        {
            // if character is lower-case
            if(islower(element)) {
                shiftCharacter(element, 'a', 'z'); // to shift character in range a-z
            }
            // if character is upper-case
            else if(isupper(element)) {
                shiftCharacter(element, 'A', 'Z'); // to shift character in range A-Z
            }
            // if character is digit
            else if(isdigit(element)) {
                shiftCharacter(element, '0', '9'); // to shift character in range 0-9
            }
            // no other character is allowed
            else {
                throw std::runtime_error{"password must be between a-z, A-Z, 0-9"}; // throw an
exception
            }
        }

        return; // return control
    }

    void shiftCharacter(char& character, char start, char end) // to shift each individual letter by
shift key
```

```
    {
        for(int i{1}; i <= SHIFT_VALUE; ++i)
        {
            ++character; // increment individual letter

            // if letter is greater than its range
            if(character > end) {
                character = start; // move to start of range
            }
        }

        return; // return control
    }
};

#endif
```

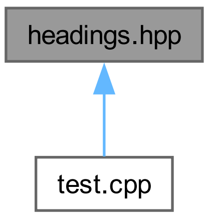
6.7 headings.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include "encryption.hpp"
#include "libraryFunctions.hpp"
#include "librarianConfirmation.hpp"
```

Include dependency graph for headings.hpp:



This graph shows which files directly or indirectly include this file:



Functions

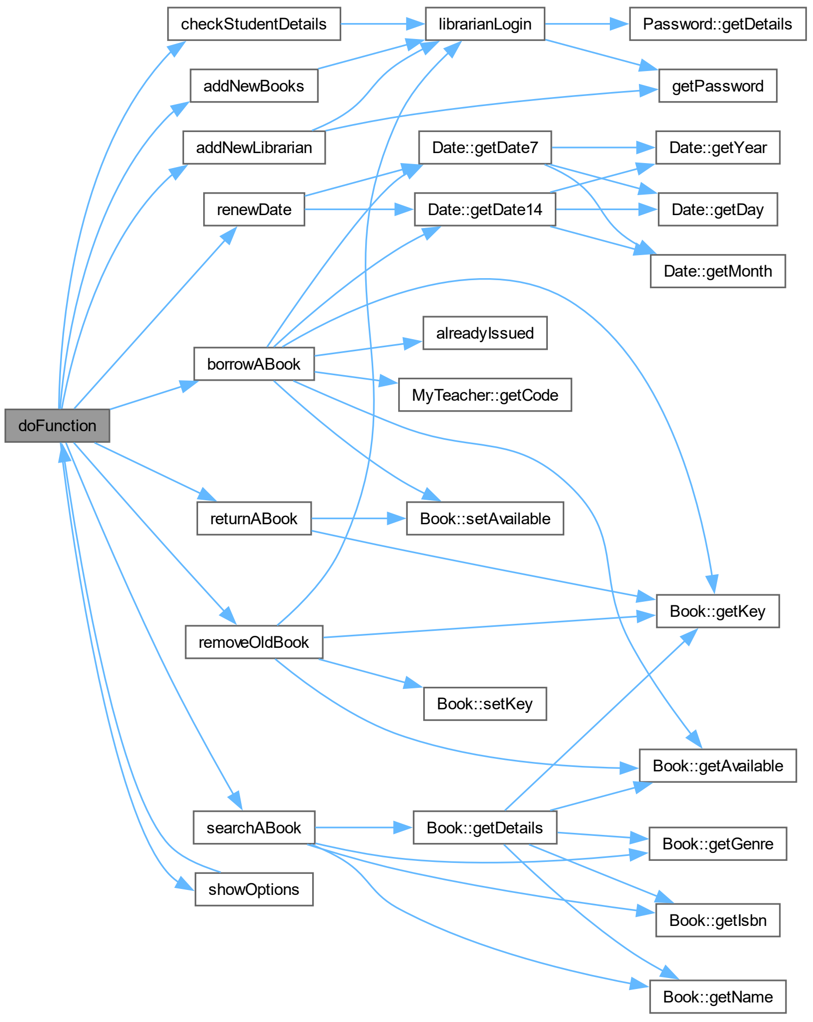
- void startingInstructions (void)
- void newLibrarian (void)
- void showOptions (void)
- void doFunction (int)

6.7.1 Function Documentation

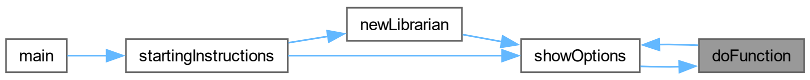
6.7.1.1 doFunction()

```
void doFunction (
            int choice )
```

Here is the call graph for this function:



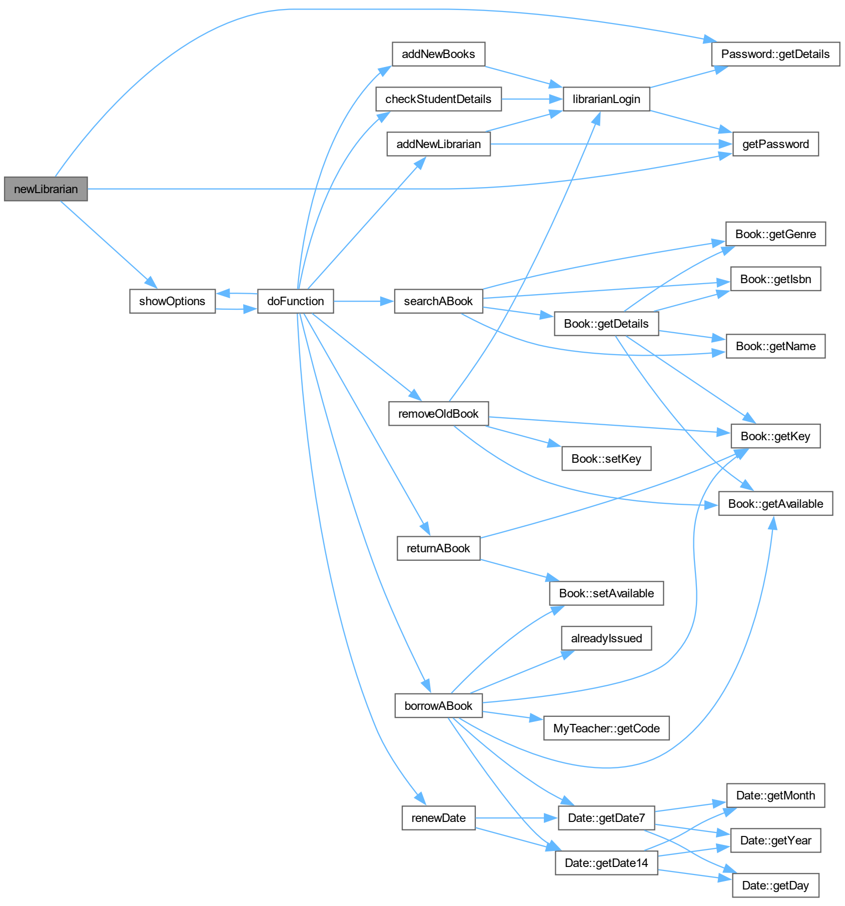
Here is the caller graph for this function:



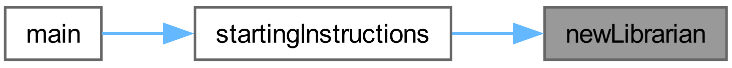
6.7.1.2 newLibrarian()

```
void newLibrarian (
            void   )
```

Here is the call graph for this function:



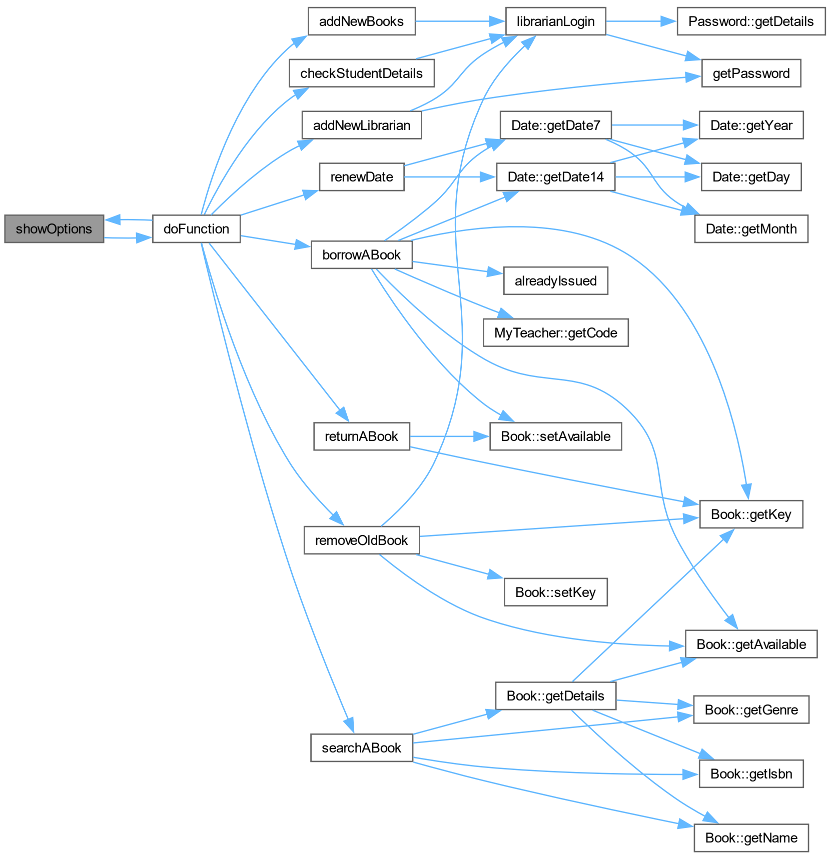
Here is the caller graph for this function:



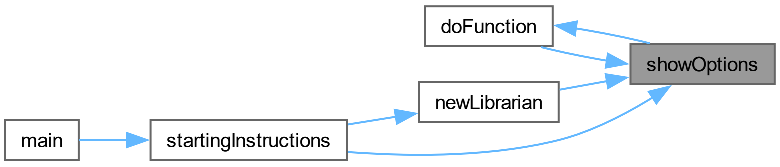
6.7.1.3 showOptions()

```
void showOptions (
            void   )
```

Here is the call graph for this function:



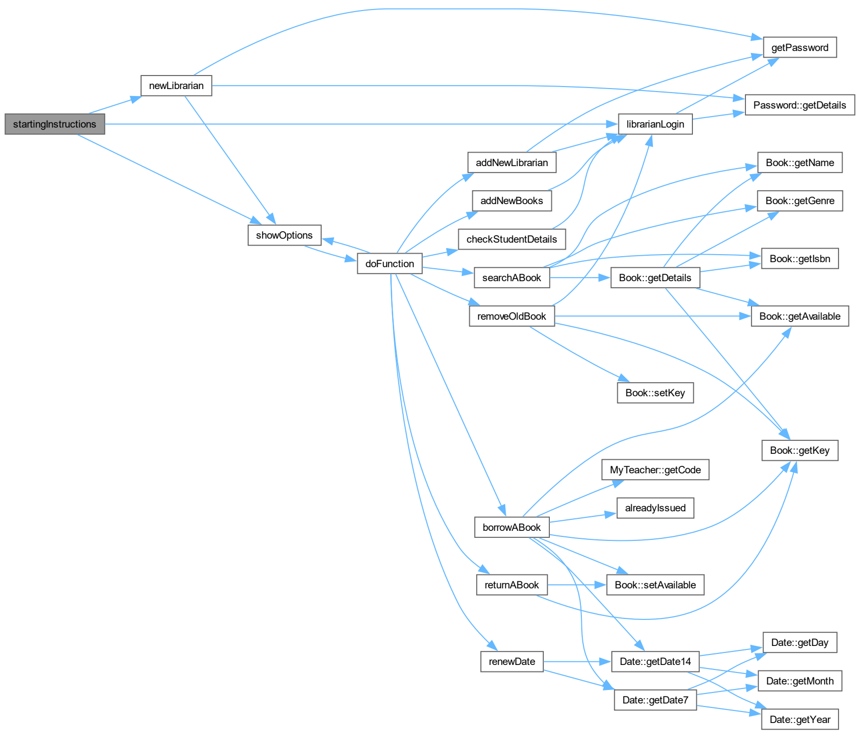
Here is the caller graph for this function:



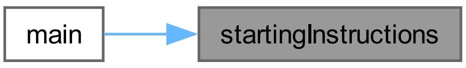
6.7.1.4 startingInstructions()

```
void startingInstructions (
             void   )
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.8 headings.hpp

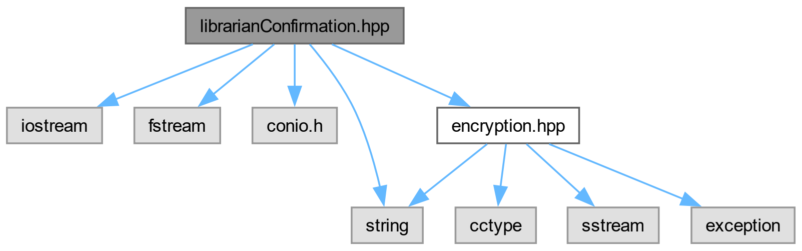
Go to the documentation of this file.

```
// This header file 'headings.hpp' consist of basic functions and implementations until a librarian is
logged in. It also provide functionalities to display navigation options. This file is starting and
ending point of application. Any of other files is not capable to start or end application.

#ifndef HEADING_HPP
#define HEADING_HPP

#include <iostream>
#include <fstream>
#include <conio.h>
#include "encryption.hpp" // consist of class 'Password'
#include "libraryFunctions.hpp" // consist of library internal functions
#include "librarianConfirmation.hpp" // librarian login functions

void startingInstructions(void); // welcome page of library
void newLibrarian(void); // function to create new Librarian
void showOptions(void); // navigation options of library
void doFunction(int); // call proper function as specified by user

void startingInstructions()
{
    // welcome display
    std::cout « "------------------------------------------------" « std::endl;
    std::cout « "      Welcome to Library Management System      " « std::endl;
    std::cout « "------------------------------------------------" « std::endl;

    std::ifstream dataFile{"librarian.txt", std::ios::in}; // try to open librarian.txt

    // if file doesn’t exist, no librarian exists
    if(!dataFile) {
        newLibrarian(); // create new librarian
    }
    // if file exist, at least one librarian exists
    else {
        std::cout « "\nLibrarian Login\n"; // librarian login

        // call ’librarianLogin’ to perform login, it return true or false
        if(librarianLogin()) {
            // if librarian is logged in
            // show options since he is librarian

            system("cls"); // clear screen
            showOptions(); // show navigation options
        }
        // else show message since he is not librarian
        else {
            std::cout « "\nAccess Denied" « std:: endl;
        }
    }

    dataFile.close(); // close the file
    return; // return control
}

void newLibrarian(void) // to create new librarian
{
    std::ofstream dataFile{"librarian.txt", std::ios::out}; // create new file
    // if file is not created, exit the program
    if(!dataFile) {
        std::cerr « "cannot create file";
        exit(EXIT_FAILURE);
    }

    std::string name, password; // to hold typed data

    // new librarian signup
    std::cout « "\nLibrarian Signup\n";
    std::cout « "\nEnter your Name: "; // prompt to enter name
    getline(std::cin, name); // using ’getline’ to insert spaces
    std::cout « "Enter new Password (only letters or digits): "; // prompt to enter password
    password = getPassword(); // using ’getPassword’ to hide it from console

    // create Password object to encrypt password
    Password newPerson{name, password};
    dataFile « newPerson.getDetails() « std::endl; // store the data in file

    // show options since he is new librarian
    system("cls"); // clear screen
    showOptions();

    dataFile.close(); // close the file

    return; // return control
}

void showOptions() // library navigation
{
    // these options will require librarian password again
    std::cout « "Librarian’s Access:\n";
    std::cout « "1 - Add Another Librarian" « ’\t’ « "2 - Add New Books" « ’\n’ « "3 - Remove Old Books" « "\t\t" « "4 - Check Student Details" « std::endl;

    // these options will not require password again
    // anyone can check them
    std::cout « "\nPublic’s Access:\n";
    std::cout « "5 - Borrow A Book" « "\t\t" « "6 - Return A Book" « ’\n’ « "7 - Renew Return Date" « "\t\t" « "8 - Search A Book" « ’\n’ « "9 - Exit Library\n" « std::endl;

    const int LOWEST_OPTION{1};
    const int HIGHEST_OPTION{9};

    int num;
    // loop until a number in range of lowest-highest is entered
    do
    {
        std::cout « "? ";
        std::cin » num;
    }while(num < LOWEST_OPTION || num > HIGHEST_OPTION);

    // option 9 is exit library
    // so if 9 is entered do not call 'doFunction'
    if(num != 9) {
        // call 'doFunction; which will in response call function associated with entered number
        doFunction(num);
    }

    // if 9 is entered by user
    // exit the library
    std::cout « "\nThankYou!" « std::endl;

    exit(EXIT_SUCCESS); // exit the application, this is ending point of application
}

void doFunction(int choice) // to call specified function as entered by user
{
    // send control to case as entered by user
    switch(choice)
    {
        case 1: // new librarian
            addNewLibrarian();
            break;
        case 2: // new books
            addNewBooks();
            break;
        case 3: // remove books
            removeOldBook();
            break;
        case 4: // check student details
            checkStudentDetails();
            break;
        case 5: // borrow a book
            borrowABook();
            break;
        case 6: // return a book
            returnABook();
            break;
        case 7: // renew date
            renewDate();
            break;
        case 8: // search books
            searchABook();
            break;
        default:
            // control should not reach here
            std::cerr « "control should not reach here";
            break;
    }

    std::cout « "\n\nPress any key to continue..."; // it is displayed at end of working of every
option
    getch(); // wait for a key
    system("cls"); // clear screen

    // again show options
    showOptions();
}

#endif
```

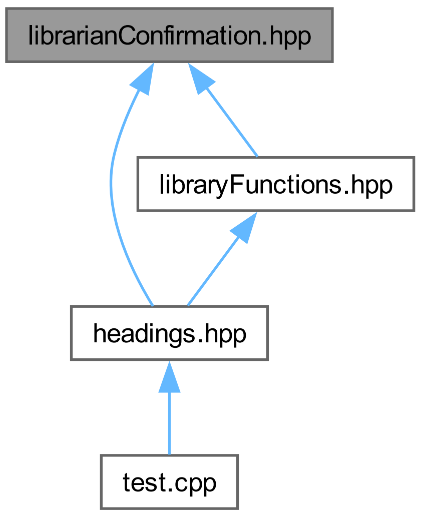
6.9 librarianConfirmation.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <string>
#include "encryption.hpp"
```

Include dependency graph for librarianConfirmation.hpp:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum IN { IN_BACK = 8 , IN_RET = 13 }

Functions

- std::string getPassword (char sp=' ∗')
- bool librarianLogin ()

6.9.1 Enumeration Type Documentation

6.9.1.1 IN

```
enum IN
```

Enumerator

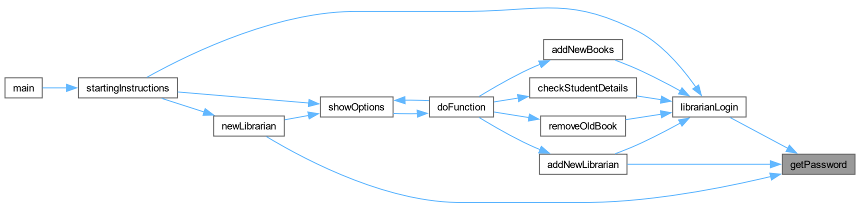
IN_BACK	
IN_RET	

6.9.2 Function Documentation

6.9.2.1 getPassword()

```
std::string getPassword (
            char sp = '*' )
```

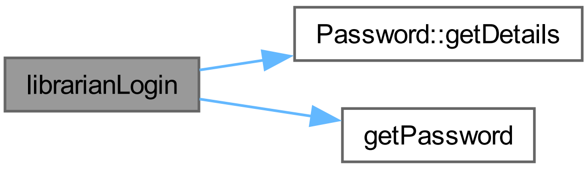
Here is the caller graph for this function:



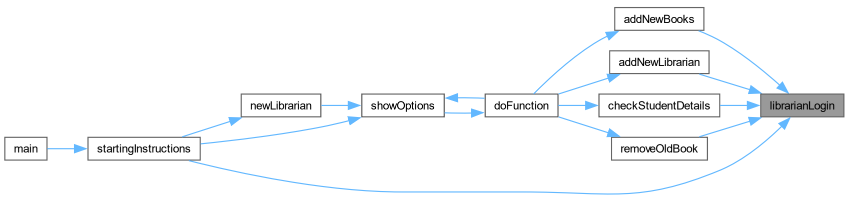
6.9.2.2 librarianLogin()

```
bool librarianLogin ( )
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.10 librarianConfirmation.hpp

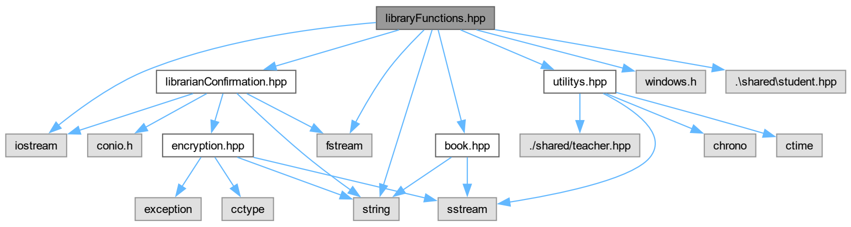
Go to the documentation of this file.

```
// This header file consist of functions that will be used at time of librarian login. The functions
'librarianLogin' is called anywhere we went to login librarian. This file consist of all
functionalities it needs except password encryption

#ifndef CONFIRMATION_HPP
#define CONFIRMATION_HPP

// preprocessor directives
#include <iostream>
#include <fstream>
#include <conio.h>
#include <string>
#include "encryption.hpp"

enum IN
{
    IN_BACK = 8, // ASCII value for Backspace
    IN_RET = 13 // ASCII value for Enter/Return
};

// return the password entered by user
// hide password typed by character supplied in 'sp'
std::string getPassword(char sp = '*')
{
    std::string password = ""; // to store password
    char ch_ipt; // to store character as it is typed

    // loop continues until Enter key is pressed
    while(true)
    {
        ch_ipt = getch(); // input a character

        // if Enter is pressed
        if(ch_ipt == IN::IN_RET) {
            std::cout « std::endl; // print a newline
            return password; // return password
        }
        // if Backspace is pressed and password is not empty
        else if((ch_ipt == IN::IN_BACK) && (password.length() != 0)) {
            password.pop_back(); // remove character from end of string
            std::cout « "\b \b"; // update console by backspace

            continue; // move to next iteration
        }
        // if Backspace is pressed and password is empty
        else if((ch_ipt == IN::IN_BACK) && (password.length() == 0)) {
            continue; // do nothing move to next iteration
        }

        // if any other key is pressed
        password.push_back(ch_ipt); // push it onto string
        std::cout « sp; // and display 'sp' character on screen in place of charcter typed
    }
}

// call this function simply when you want to login a librarian
// all input and checking is performed by it
// return bool value to indicate if login was successful or not
bool librarianLogin()
{
    std::ifstream dataFile{"librarian.txt", std::ios::in}; // open librarian file
    // if file doesn't exist
    if(!dataFile) {
        std::cerr « "file doesn't exists";
        exit(EXIT_FAILURE); // exit since no librarian exist, exist is necessary because new librarian
is not made as startup of application
    }

    std::string name, password; // to hold user typed data

    std::cout « "\nEnter your Name: "; // prompt to enter name
    getline(std::cin, name); // used 'getline' because name may consist of space
    std::cout « "Enter your Password: "; // prompt to enter password
    password = getPassword(); // call 'getPassword' so it can hide typed password

    Password loginPerson{name, password}; // creating object of Password, because this class can
automatically encrypt the password
    std::string thisData = loginPerson.getDetails(); // get concatenated name and password(encrypted)

    std::string fileRecord; // to hold librarian data from file

    // loop until all lines of file
    while(getline(dataFile, fileRecord)) // get a line from file, since name and password (of one
librarian) are stored in one line
    {
        // if it match with typed data
        if(fileRecord == thisData) {
            dataFile.close(); // close the file
            return true; // and return true, since data matched and authentication is done
        }
    }
    // if the loop ends and not one record in file matched with typed data

    dataFile.close(); // close the file
    return false; // and return false, since no data matched
}

#endif
```

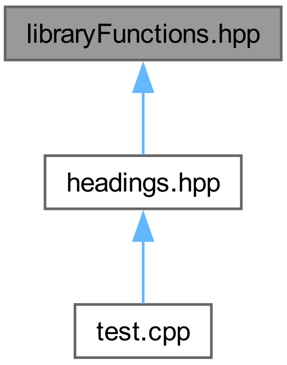
6.11 libraryFunctions.hpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <windows.h>
#include "book.hpp"
#include "librarianConfirmation.hpp"
#include ".\shared\student.hpp"
#include "utilitys.hpp"
```

Include dependency graph for libraryFunctions.hpp:



This graph shows which files directly or indirectly include this file:



Functions

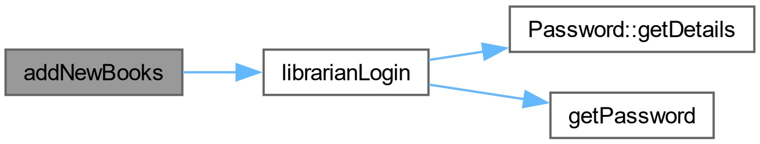
- void addNewLibrarian ()
- void addNewBooks ()
- void removeOldBook ()
- void checkStudentDetails ()
- void borrowABook ()
- void returnABook ()
- void renewDate ()
- void searchABook ()

6.11.1 Function Documentation

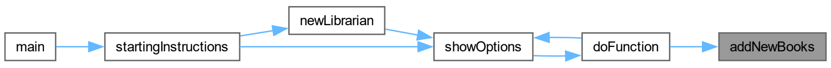
6.11.1.1 addNewBooks()

```
void addNewBooks ( )
```

Here is the call graph for this function:



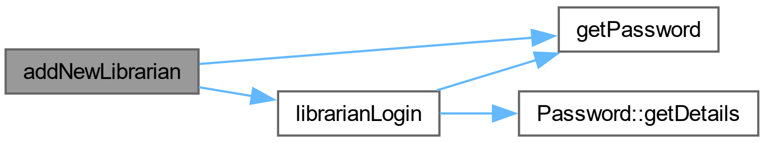
Here is the caller graph for this function:



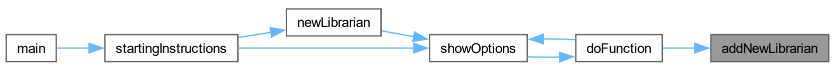
6.11.1.2 addNewLibrarian()

```
void addNewLibrarian ( )
```

Here is the call graph for this function:



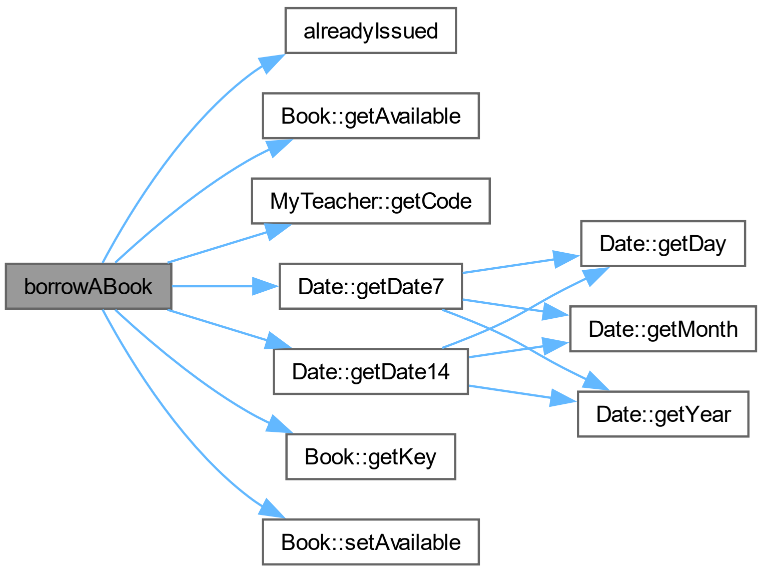
Here is the caller graph for this function:



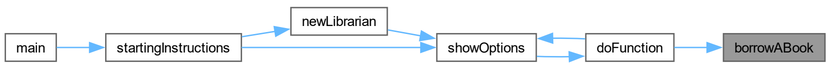
6.11.1.3 borrowABook()

```
void borrowABook ( )
```

Here is the call graph for this function:



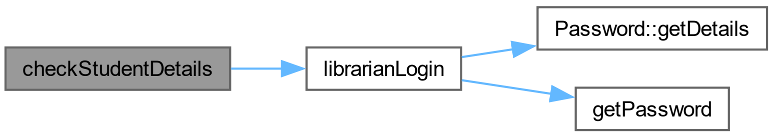
Here is the caller graph for this function:



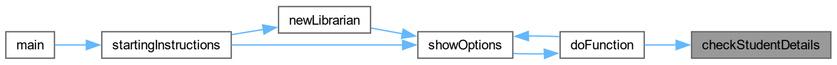
6.11.1.4 checkStudentDetails()

```
void checkStudentDetails ( )
```

Here is the call graph for this function:



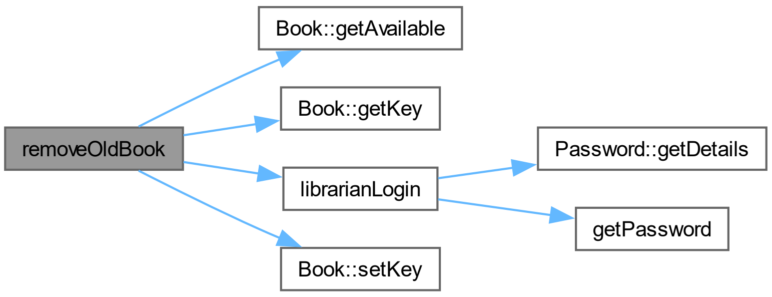
Here is the caller graph for this function:



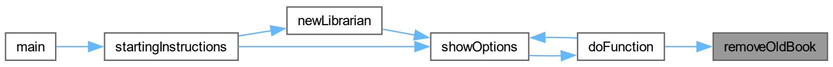
6.11.1.5 removeOldBook()

```
void removeOldBook ( )
```

Here is the call graph for this function:



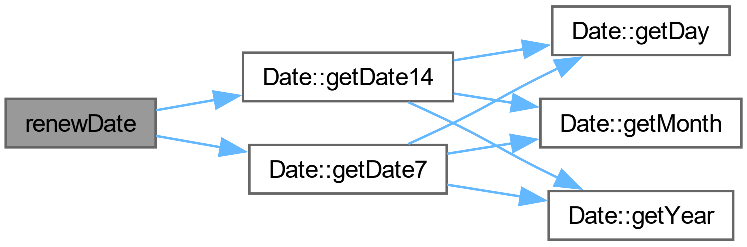
Here is the caller graph for this function:



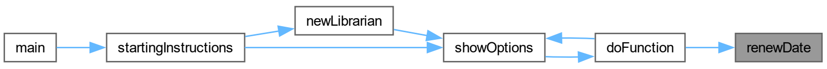
6.11.1.6 renewDate()

```
void renewDate ( )
```

Here is the call graph for this function:



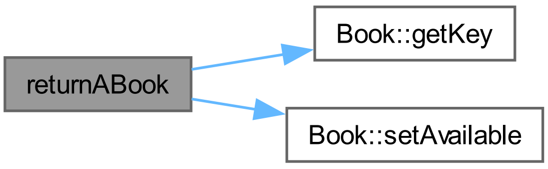
Here is the caller graph for this function:



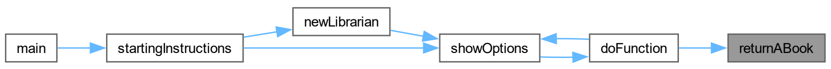
6.11.1.7 returnABook()

```
void returnABook ( )
```

Here is the call graph for this function:



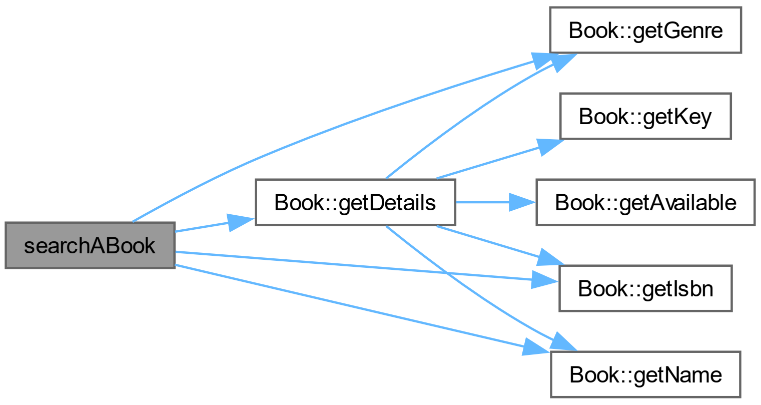
Here is the caller graph for this function:



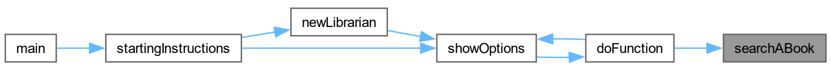
6.11.1.8 searchABook()

```
void searchABook ( )
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.12 libraryFunctions.hpp

Go to the documentation of this file.

```
// This header file consist of functions that can be selected by user in library. Each corresponds to
a specific work as entered by user. Each function here is self implemented and is not depended on any
other function in this file

#ifndef LIBRARYFUNCTIONS_HPP
#define LIBRARYFUNCTIONS_HPP

```

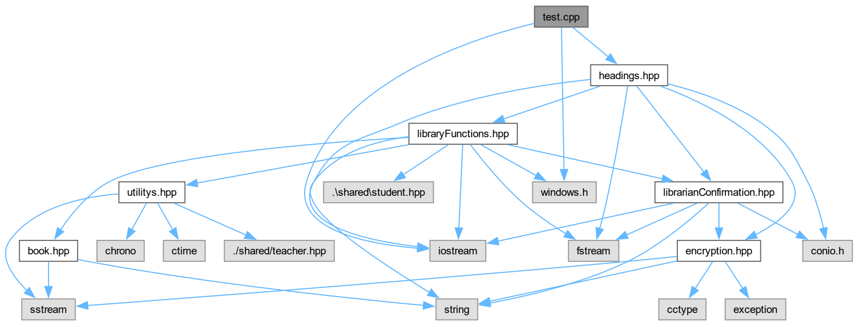
```
#include <iostream>
#include <string>
#include <fstream>
#include <windows.h>
#include "book.hpp" // definition of class 'Book' to make record of book
#include "librarianConfirmation.hpp" // for librarian login
#include ".\shared\student.hpp"  // defintion of class 'Student'
#include "utilitys.hpp" // definition of class 'Date' and 'MyTeacher'

// 1 - Add New Librarian
void addNewLibrarian()
{
    system("cls"); // clear screen

    std::cin.ignore(); // ignore last typed Enter

    // if librarian is logged in
    if(librarianLogin()) {
        std::string name, password; // to hold data

        system("cls"); // clear screen
        std::cout « "Enter New Librarian Name: "; // prompt for name
        getline(std::cin, name);
        std::cout « "Enter new Password (only letters or digits): "; // prompt for password
        password = getPassword();

        Password newPerson{name, password}; // create object of 'Password' using typed data

        std::fstream file{"librarian.txt", std::ios::in | std::ios::out | std::ios::app}; // open
librarian file
        if(!file) {
            std::cerr « "librarian file doesn't open";
            exit(EXIT_FAILURE);
        }

        file « newPerson.getDetails() « '\n'; // print new librarian data in file as a record
        std::cout « "\nLibrarian Added"; // display message
    }
    // if not logged in
    else {
        std::cout « "\nAccess Denied"; // display message
    }

    return; // return control
}

// 2 - Add New Books
void addNewBooks()
{
    system("cls"); // clear screen

    std::cin.ignore(); // ignore last typed Enter

    // if librarian is logged in
    if(librarianLogin()) {
        std::fstream booksFile{"bookRecords.dat", std::ios::out | std::ios::in | std::ios::binary |
std::ios::app}; // open books record file

        if(!booksFile) {
            std::cerr « "Cannot open file to add books";
            exit(EXIT_FAILURE);
        }

        // variables to hold information of book
        int key;
        std::string isbn;
        std::string name;
        std::string genre;
        bool available;

        int next; // used by loop

        do
        {
            system("cls"); // clear screen

            //deciding key, since it is generated automatically
            booksFile.seekg(0, std::ios::end); // move to end of file
            int fileSize = booksFile.tellg(); // get size of file
            int numBooks = fileSize / sizeof(Book); // divide full file size to size of one record to
get number of books
            key = numBooks > 0 ? numBooks : 0; // set key

            std::cout « "Enter ISBN: "; // prompt for isbn
            std::cin » isbn;
            std::cout « "Enter Book Name: "; // prompt for name
            std::cin.ignore();
            getline(std::cin, name); // name may include spaces
            std::cout « "Enter Book Genre: "; // prompt for genre
            std::cin » genre;

            available = true; // set availability to true

            Book newBookRecord{key, isbn, name, genre, available}; // make an object using above typed data

            booksFile.write(reinterpret_cast<const char *>(&newBookRecord), sizeof(Book)); // write this record in binary file

            std::cout « "Book Added" « std::endl; // successful message

            std::cout « "\nAdd another book(1 or 0): "; // if want to add more books
            std::cin » next;
        }while(next == 1);
    }
    // if not logged in
    else {
        std::cout « "\nAccess Denied"; // show message
    }

    return; // return control
}

// 3 - Remove Old Book
void removeOldBook()
{
    system("cls"); // clear screen

    std::cin.ignore(); // ignore last ’enter’ from the stream

    // if librarian is logged in
    if(librarianLogin()) {
        std::ifstream inputFile{"bookRecords.dat", std::ios::in | std::ios::binary}; // open book records file
        if(!inputFile) {
            std::cerr « "Cannot open file to remove book";
            exit(EXIT_FAILURE);
        }

        // check if any book is issued to someone
        int availFlag = 1;
        Book checkBooks;
        while(inputFile.read(reinterpret_cast<char *>(&checkBooks), sizeof(Book)))
        {
            // if a book is issued to someone i.e. available = 0
            if(checkBooks.getAvailable() == 0) {
                availFlag = 0; // set flag to false
                break;
            }
        }

        // if flag is flag, at least one book is issued
        if(availFlag == 0) {
            std::cout « "Not all books are available";
            return; // return control, and doesn’t allow to remove old book
        }

        inputFile.seekg(0, std::ios::beg); // seek to beginning of file

        int delKey;

        system("cls");
        std::cout « "Enter Book’s key to be deleted: "; // prompt to enter key of book to be deleted
        std::cin » delKey;

        std::ofstream outputFile{"temp.dat", std::ios::out | std::ios::binary}; // create a file ’temp.dat’
        if(!outputFile) {
            std::cerr « "Cannot open file to remove book";
            inputFile.close();
            exit(EXIT_FAILURE);
        }

        int newKeys{0}; // assigning new key to each book
        bool flag = false;

        Book record;
        while(inputFile.read(reinterpret_cast<char *>(&record), sizeof(Book))) // read a record from books’ file
        {
            // is book’s key is not equal to delKey
            if(record.getKey() != delKey) {
                record.setKey(newKeys++);
                // write the record in ’temp’ file
                outputFile.write(reinterpret_cast<const char *>(&record), sizeof(Book));
            }
            else {
                // if a book's key matched don't write it into temp file and set flag to true, indicating that the book is founded
                flag = true;
            }
        }

        // if flag is true i.e. the delKey book is not written into temp file
        if(flag == true) {
            std::cout « "Book Deleted" « '\n';
        }
        else {
            std::cout « "Book doesn't exist" « '\n';
        }

        inputFile.close();
        outputFile.close();

        remove("bookRecords.dat"); // remove books' record file
        rename("temp.dat", "bookRecords.dat"); // rename temp file to books' record
    }
    else {
        // if librarian doesn't log in
        std::cout « "\nAccess Denied";
    }

    return;
}

// 4 - check student details
void checkStudentDetails()
{
    system("cls");

    std::cin.ignore();

    // if librarian is logged in
    if(librarianLogin()) {
        system("cls");

        std::cout « "1 - Search with RollNo" « '\t' « "2 - Display all Students" « std::endl « std::endl;

        const int LOWEST_OPTION{1};
        const int HIGHEST_OPTION{2};

        int searchChoice;
        // validating the input number
        do
        {
            std::cout « "? ";
            std::cin » searchChoice;
        }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);

        std::ifstream studentFile{"studentRecords.dat", std::ios::in | std::ios::binary}; // open student's records file

        if(!studentFile) {
            std::cerr « "Cannot open file to read books";
            exit(EXIT_FAILURE);
        }

        switch(searchChoice)
        {
            case 1: // display student with roll no
            {
                int rollNo;
                std::cout « "\nEnter roll no: "; // prompt to enter rollNo
                std::cin » rollNo;

                Student student;

                studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student)); // read a record from file
                while(studentFile)
                {
                    // if rollNo matched
                    if(student.getRollNo() == rollNo) {
                        std::cout « "\nResult:\n";
                        std::cout « student; // display the student's details
                        break; // break through the loop
                    }

                    // if rollNo doesn't match, read next record, until the end of file
                    studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student));
                }
            }
                break;
            case 2: // display all students;
            {
                Student student;

                // read a record from the file
                studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student));
                std::cout « "\nResult:\n";
                while(studentFile) // while not 'end of file'
                {
                    std::cout « student.display() « '\n'; // display the record

                    // read next record
                    studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student));
                }
            }
                break;
            default: // because of validation, control should not reach here
                std::cerr « "control should not reach here";
                break;
        }

        studentFile.close(); // close the file
    }
    // if the librarian is not logged in
    else {
        std::cout « "\nAccess Denied";
    }

    return;
}

// 5 - Borrow A Book
void borrowABook()
{
    system("cls");
    // students are issued books for 7 days and teachers for 14 days
    std::cout « "1 - For Student" « '\t' « "2 - For Teacher\n" « std::endl;

    const int LOWEST_OPTION{1};
    const int HIGHEST_OPTION{2};

    int searchChoice;
    // validating input
    do
    {
        std::cout « "? ";
        std::cin » searchChoice;
    }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);

    // if student want to borrow book
    if(searchChoice == 1) {
        int rollNo;

        std::cout « "\n\nEnter your roll no: "; // ask for student's roll no
        std::cin » rollNo;

        std::ifstream studentFile{"studentRecords.dat", std::ios::in | std::ios::binary}; // open
students' records file
        if(!studentFile) {
            std::cerr « "cannot open student's records";
            exit(EXIT_FAILURE);
        }

        Student student;
        bool studentExist = false; // to mark if student doesn't exist
        while(studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student))) // read a student
record from file
        {
            // if roll no matches
            if(student.getRollNo() == rollNo) {
                studentExist = true; // student exist

                if(alreadyIssued(rollNo)) {
                    std::cout « "The Student already have a book issued";
                    studentFile.close();

                    return;
                }

                int key;

                std::cout « "Enter book key: "; // prompt to enter book key
                std::cin » key;

                std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out | std::ios::binary}; // open books' records file
                if(!bookFile) {
                    std::cerr « "cannot open book's records";
                    studentFile.close();
                    exit(EXIT_FAILURE);
                }

                Book book;
                bool bookExist = false; // to mark if book doesn't exist
                while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read a book record
                {
                    // if the key of book matched with input key, and the book is available
                    if((book.getKey() == key) && (book.getAvailable() == true)) {
                        bookExist = true; // book exist

                        Date today; // create object of 'Date' initialized with current date
                        std::cout « "\nBook Issued till " « today.getDate7(); // issue book for 7 days

                        std::fstream issueFile{"issued.txt", std::ios::out | std::ios::app}; // open issued file
                        if(!issueFile) {
                            std::cerr « "cannot open issue file";
                            // close all opened files
                            studentFile.close();
                            bookFile.close();
                            exit(EXIT_FAILURE);
                        }

                        // write data of student and issued book is file
                        issueFile « rollNo « ' ' « key « ' ' « today.getDate7() « '\n';

                        book.setAvailable(false); // set book availability to false;

                        // seek back to start of record
                        bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
                        bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write book record with availability marked to false

                        break;
                    }
                }

                // if book doesn't exist
                if(bookExist == false) {
                    std::cout « "\nBook not found";
                }

                bookFile.close();
                break;
            }
        }

        studentFile.close();

        // if student doesn't exist
        if(studentExist == false) {
            std::cout « "\nStudent not found";
        }
    }
    // if teacher want to borrow a book
    else if(searchChoice == 2) {
        int code;

        std::cout « "\n\nEnter your code: "; // prompt to enter teacher's code
        std::cin » code;

        std::ifstream teacherFile{"teacherRecords.dat", std::ios::in | std::ios::binary}; // open teachers' records file
        if(!teacherFile) {
            std::cerr « "cannot open student's records";
            exit(EXIT_FAILURE);
        }

        MyTeacher teacher;
        bool teacherExist = false; // to mark if teacher doesn't exist
        while(teacherFile.read(reinterpret_cast<char *>(&teacher), sizeof(MyTeacher))) // read a record from teacher file
        {
            // if code matched
            if(teacher.getCode() == code) {
                teacherExist = true; // teacher exists

                if(alreadyIssued(code)) {
                    std::cout « "The Teacher already have a book issued";
                    teacherFile.close();

                    return;
                }

                int key;

                std::cout « "Enter book key: "; // prompt to enter book key
                std::cin » key;

                std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out | std::ios::binary}; // open books’ records file
                if(!bookFile) {
                    std::cerr « "cannot open book’s records";
                    teacherFile.close();
                    exit(EXIT_FAILURE);
                }

                Book book;
                bool bookExist = false; // to mark if book doesn’t exist
                while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read a record from book file
                {
                    // if book key matched, and book is avaiable
                    if((book.getKey() == key) && (book.getAvailable() == true)) {
                        bookExist = true; // book exists

                        Date today; // object of ’Date’ will be initialized with current date
                        std::cout « "\nBook Issued till " « today.getDate14(); // issued for 14 days

                        std::fstream issueFile{"issued.txt", std::ios::out | std::ios::app}; // open issued file
                        if(!issueFile) {
                            std::cerr « "cannot open issue file";
                            // closing opened files
                            teacherFile.close();
                            bookFile.close();
                            exit(EXIT_FAILURE);
                        }

                        issueFile « code « ’ ’ « key « ’ ’ « today.getDate14() « ’\n’; // write issued book and teacher details in issue file

                        book.setAvailable(false); // set book’s availability to false

                        // seek back to start of the record
                        bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
                        bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write record of book in file with availability set to false

                        break;
                    }
                }

                // is book doesn’t exist
                if(bookExist == false) {
                    std::cout « "\nBook not found";
                }

                bookFile.close();
                break;
            }
        }

        teacherFile.close();

        // if teacher doesn’t exist
        if(teacherExist == false) {
            std::cout « "\nTeacher not found";
        }
    }

    return;
}

// 6 - Return A Book
void returnABook()
{
    system("cls");
    std::cout « "1 - For Student" « ’\t’ « "2 - For Teacher\n" « std::endl;

    const int LOWEST_OPTION{1};
    const int HIGHEST_OPTION{2};

    int searchChoice;
    // input validation
    do
    {
        std::cout « "? ";
        std::cin » searchChoice;
    }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);

    if(searchChoice == 1) { // for student
        int rollNo;

        std::cout « "\n\nEnter your roll no: "; // enter your roll no
        std::cin » rollNo;

        std::fstream issuedFile{"issued.txt", std::ios::in}; // open issue file
        if(!issuedFile) {
            std::cerr « "cannot open issued file";
            exit(EXIT_FAILURE);
        }

        int fRollNo, fKey;
        Date fDate;

        bool flag = false;
        while(issuedFile » fRollNo » fKey » fDate) // read record from issued file
        {
            // if roll no matched
            if(fRollNo == rollNo) {
                std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out | std::ios::binary}; // open books file
                if(!bookFile) {
                    std::cerr « "cannot open books file";
                    issuedFile.close();
                    exit(EXIT_FAILURE);
                }

                Book book;
                while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read a record from books' file
                {
                    // if key matched
                    if(book.getKey() == fKey) {
                        book.setAvailable(true); // set availability to true

                        // seek back to start of record
                        bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
                        bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write record with availability set to true

                        Date toDay; // get current date
                        if(fDate >= toDay) { // if returned date is greated today i.e. not reached last date
                            std::cout « "Book returned";
                        }
                        // if returned date is crossed
                        else {
                            std::cout « "Date exceeded, try to return book on time\nBook returned";
                        }

                        break;
                    }
                }

                bookFile.close();

                flag = true; // mark flag to true, since record in issue file is founded
                break;
            }
        }

        issuedFile.seekg(0, std::ios::beg); // seek to beginning of file

        // if no record in issued file found
        if(flag != true) {
            std::cout « "No book issued";
        }
        // if a record is founded
        else {
            std::ofstream file{"temp.txt", std::ios::out}; // create a temp file
            if(!file) {
                std::cerr « "cannot open file";
                // close other files
                issuedFile.close();

                exit(EXIT_FAILURE);
            }

            int fRollNo, fKey;
            std::string fDate;
            while(issuedFile » fRollNo » fKey » fDate) // read record from issued file
            {
                // if roll no doesn’t match
                if(fRollNo != rollNo) {
                    file « fRollNo « ’ ’ « fKey « ’ ’ « fDate « ’\n’; // write in temp file
                }
            }

            file.close();
            issuedFile.close();

            remove("issued.txt"); // delete issued file
            rename("temp.txt", "issued.txt"); // rename temp file to issued
        }
    }
    // for teacher
    else if(searchChoice == 2) {
        int code;

        std::cout « "\n\nEnter your code: "; // prompt to enter code
        std::cin » code;

        std::fstream issuedFile{"issued.txt", std::ios::in}; // open issued file
        if(!issuedFile) {
            std::cerr « "cannot open issued file";
            exit(EXIT_FAILURE);
        }

        int fCode, fKey;
        Date fDate;

        bool flag = false; // to mark if a book is returned
        while(issuedFile » fCode » fKey » fDate) // read a record from the issued file
        {
            // if code matched
            if(fCode == code) {
                std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out | std::ios::binary};
                if(!bookFile) {
                    std::cerr « "cannot open books file";
                    issuedFile.close();
                    exit(EXIT_FAILURE);
                }

                Book book;
                while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read record from books’ records file
                {
                    // if key matched
                    if(book.getKey() == fKey) {
                        book.setAvailable(true); // set available to true

                        // seek to start of record
                        bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
                        bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write record as availability set to true

                        Date toDay; // get current date
                        if(fDate >= toDay) { // if return date is greater than today
                            std::cout « "Book returned";
                        }
                        // if returned date is crossed
                        else {
                            std::cout « "Date exceeded, try to return book on time\nBook returned";
                        }

                        break;
                    }
                }

                bookFile.close();

                flag = true; // mark true to indicate that a book is returned
                break;
            }
        }

        issuedFile.seekg(0, std::ios::beg); // seek to beginning to file

        // if no book is returned
        if(flag != true) {
            std::cout « "No book issued";
        }
        // if a book is returned
        else {
            std::ofstream file{"temp.txt", std::ios::out};
            if(!file) {
                std::cerr « "cannot open file";
                issuedFile.close();
                exit(EXIT_FAILURE);
            }

            int fCode, fKey;
            std::string fDate;
            while(issuedFile » fCode » fKey » fDate) // read record from issued file
            {
                // if code doesn't match
                if(fCode != code) {
                    // write record in temp file
                    file « fCode « ' ' « fKey « ' ' « fDate « '\n';
                }
            }

            file.close();
            issuedFile.close();

            remove("issued.txt"); // remove issued file
            rename("temp.txt", "issued.txt"); // rename temp file to issued
        }
    }

    return;
}

// 7 - Renew Date
void renewDate()
{
    system("cls");
    std::cout « "1 - For Student" « '\t' « "2 - For Teacher\n" « std::endl;

    const int LOWEST_OPTION{1};
    const int HIGHEST_OPTION{2};

    int searchChoice;
    // input validation
    do
    {
        std::cout « "? ";
        std::cin » searchChoice;
    }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);

    // for student
    if(searchChoice == 1) {
        int rollNo;

        std::cout « "\n\nEnter your roll no: "; // prompt to enter roll no
        std::cin » rollNo;

        std::fstream issuedFile{"issued.txt", std::ios::in}; // open issue file
        if(!issuedFile) {
            std::cerr « "cannot open issued file";
            exit(EXIT_FAILURE);
        }

        int fRollNo, fKey;
        Date fDate;

        bool flag = false;
        while(issuedFile » fRollNo » fKey » fDate) // read record from issued file
        {
            // if roll no matched
            if(fRollNo == rollNo) {
                flag = true; // mark flag as true
                break;
            }
        }

        // seek to beginning
        issuedFile.seekg(0, std::ios::beg);

        // if such student doesn't exist
        if(flag != true) {
            std::cout « "No book issued";
        }
        // if student exists
        else {
            std::ofstream file{"temp.txt", std::ios::out}; // create a temp file
            if(!file) {
                std::cerr « "cannot open file";
                issuedFile.close();
                exit(EXIT_FAILURE);
            }

            int fRollNo, fKey;
            std::string fDate;
            while(issuedFile » fRollNo » fKey » fDate) // read a record from issued file
            {
                // if roll no doesn't match
                if(fRollNo != rollNo) {
                    file « fRollNo « ' ' « fKey « ' ' « fDate « '\n'; // write record in temp file
                }
                // if roll no matched
                else {
                    Date today; // get current date
                    file « fRollNo « ' ' « fKey « ' ' « today.getDate7() « '\n'; // write record with
today+7 days
                }
            }

            file.close();
            issuedFile.close();

            remove("issued.txt"); // delete issued file
            rename("temp.txt", "issued.txt"); // rename temp to issued
        }
    }
    // for teacher
    // the below code work same as the one for teacher. the only different is teacher renewed returned
date is +14
    else if(searchChoice == 2) {
        int code;

        std::cout « "\n\nEnter your code: ";
        std::cin » code;

        std::fstream issuedFile{"issued.txt", std::ios::in};
        if(!issuedFile) {
            std::cerr « "cannot open issued file";
            exit(EXIT_FAILURE);
        }

        int fCode, fKey;
        Date fDate;

        bool flag = false;
        while(issuedFile » fCode » fKey » fDate)
        {
            if(fCode == code) {
                flag = true;
                break;
            }
        }

        issuedFile.seekg(0, std::ios::beg);

        if(flag != true) {
            std::cout « "No book issued";
        }
        else {
            std::ofstream file{"temp.txt", std::ios::out};
            if(!file) {
                std::cerr « "cannot open file";
                exit(EXIT_FAILURE);
            }

            int fCode, fKey;
            std::string fDate;
            while(issuedFile » fCode » fKey » fDate)
            {
                if(fCode != code) {
                    file « fCode « ' ' « fKey « ' ' « fDate « '\n';
                }
                else {
                    Date today;
                    // write with 14 days ahead from today
                    file « fCode « ' ' « fKey « ' ' « today.getDate14() « '\n';
                }
            }

            file.close();
            issuedFile.close();

            remove("issued.txt");
            rename("temp.txt", "issued.txt");
        }
    }

    return;
}

// 8 - Search A Book
void searchABook()
{
    system("cls");
    std::cout « "1 - Search with Key" « '\t' « "2 - Search with Name" « '\n' « "3 - Search with ISBN" « "\t" « "4 - Filter using genre" « '\n' « "5 - Get Full List" « std::endl « std::endl;

    const int LOWEST_OPTION{1};
    const int HIGHEST_OPTION{5};

    int searchChoice;
    // input validation
    do
    {
        std::cout « "? ";
        std::cin » searchChoice;
    }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);

    std::fstream booksFile{"bookRecords.dat", std::ios::in | std::ios::binary}; // open books' records file

    if(!booksFile) {
        std::cerr « "Cannot open file to read books";
        exit(EXIT_FAILURE);
    }

    switch(searchChoice)
    {
        case 1: // search with key
        {
            int key;
            std::cout « "\nEnter key: "; // prompt to enter key
            std::cin » key;

            Book libraryBook;

            // since file is binary, and books are sorted wrt key
            // seek directly to book
            booksFile.seekg(key * sizeof(Book));
            booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read the record

            std::cout « "\nResult:\n";
            std::cout « libraryBook.getDetails(); // display the record
        }
            break;
        case 2: // search with name
        {
            std::string name;
            std::cout « "\nEnter name: "; // prompt to enter book's name
            std::cin.ignore();
            getline(std::cin, name); // name may include spaces

            Book libraryBook;
            booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record from file
            while(booksFile)
            {
                // if name matches
                if(libraryBook.getName() == name) {
                    std::cout « "\nResult:\n";
                    std::cout « libraryBook.getDetails(); // display the record
                    break;
                }

                // else read the next record, until the end of file
                booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
            }
        }
            break;
        case 3: // search with ISBN
        {
            std::string isbn;
            std::cout « "\nEnter ISBN: "; // prompt to enter isbn of book
            std::cin » isbn;

            Book libraryBook;
            booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record from the file
            while(booksFile)
            {
                // if isbn matches
                if(libraryBook.getIsbn() == isbn) {
                    std::cout « "\nResult:\n";
                    std::cout « libraryBook.getDetails(); // display the record
                    break;
                }

                // else read the next record, until the end of file
                booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
            }
        }
            break;
        case 4: // Filter using genre
        {
            std::string genre;
            std::cout « "\nEnter Genre: "; // prompt to enter genre of book
            std::cin » genre;

            Book libraryBook;
            booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record from file
            std::cout « "\nResult:\n";
            while(booksFile)
            {
                // if genre matches
                if(libraryBook.getGenre() == genre) {
                    std::cout « libraryBook.getDetails() « '\n'; // display the record
                    // no break since we want to display all books with the genre
                }

                // read next record until the end of file
                booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
            }
        }
            break;
        case 5: // diplay full list
        {
            Book libraryBook;

            booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record
            std::cout « "\nResult:\n";
            while(booksFile)
            {
                std::cout « libraryBook.getDetails() « std::endl; // display it without any condition, since all record are to be displayed

                // read next record, until the end of file
                booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
            }
        }
            break;
        default: // control shouldn't reach here
            std::cerr « "control should not reach here";
            break;
    }

    booksFile.close();
    return;
}

#endif
```

6.13 README.md File Reference

6.14 test.cpp File Reference

```
#include <iostream>
#include <windows.h>
#include "headings.hpp"
```

Include dependency graph for test.cpp:



Functions

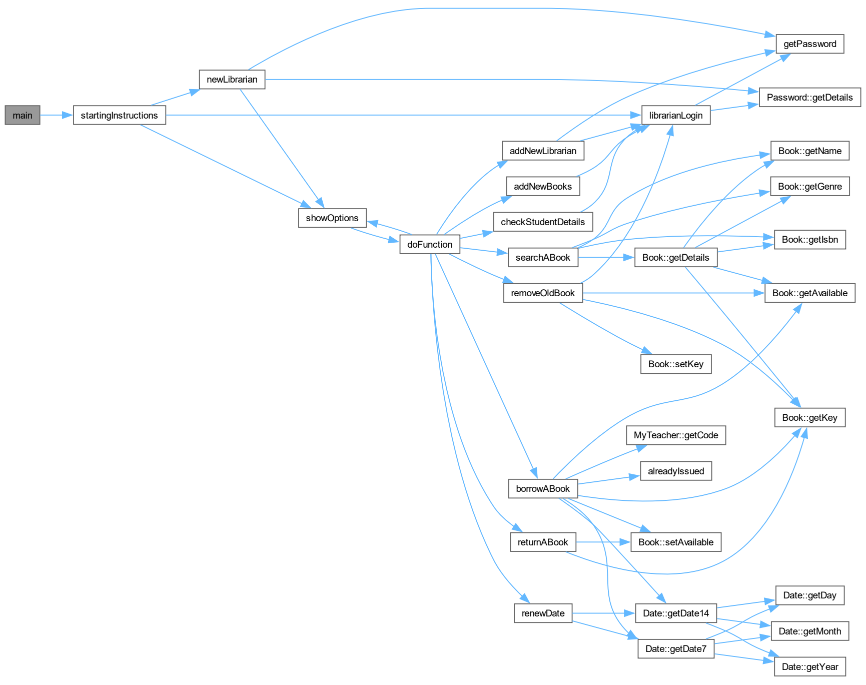
- int main ()

6.14.1 Function Documentation

6.14.1.1 main()

```
int main ( )
```

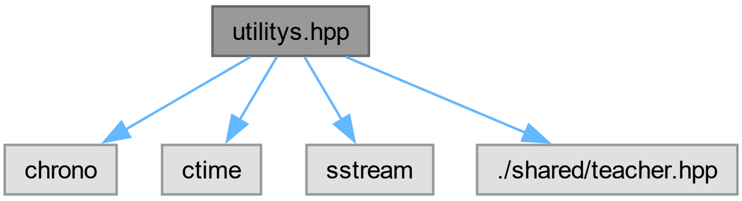
Here is the call graph for this function:



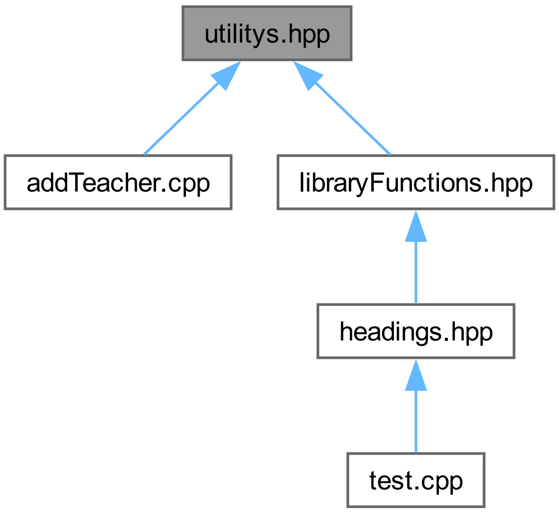
6.15 utilitys.hpp File Reference

```
#include <chrono>
#include <ctime>
#include <sstream>
#include "./shared/teacher.hpp"
```

Include dependency graph for utilitys.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Date
- class MyTeacher

Functions

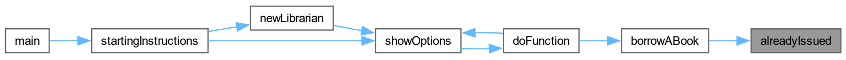
- std::istream & operator>> (std::istream &input, Date &obj)
- std::ostream & operator<< (std::ostream &output, const MyTeacher &teacher)
- std::istream & operator>> (std::istream &input, MyTeacher &teacher)
- bool alreadyIssued (int num)

6.15.1 Function Documentation

6.15.1.1 alreadyIssued()

```
bool alreadyIssued (
            int num )
```

Here is the caller graph for this function:



6.15.1.2 operator<<()

```
std::ostream & operator<< (
            std::ostream & output,
            const MyTeacher & teacher )
```

6.15.1.3 operator>>() [1/2]

```
std::istream & operator>> (
            std::istream & input,
            Date & obj )
```

6.15.1.4 operator>>() [2/2]

```
std::istream & operator>> (
            std::istream & input,
            MyTeacher & teacher )
```

6.16 utilitys.hpp

Go to the documentation of this file.

```
// This file consist of multiple classes that are independent of each other. All of these came under the single banner of utilities since they can be used by multiple functionalities

#ifndef UTILITYS_HPP
#define UTILITYS_HPP

#include <chrono>
#include <ctime>
#include <sstream>
#include "./shared/teacher.hpp"

// When created an object of 'Date', the object consist of current date, month, and year.
// It can also return currentDay+7days and currentDay+14days
class Date
{
    friend std::istream& operator»(std::istream&, Date&); // stream extraction operator
public:
    Date()
    {
        auto currentTime = std::chrono::system_clock::now(); // get the current time point

        std::time_t currentTime_t = std::chrono::system_clock::to_time_t(currentTime); // convert the time point to a time_t object

        std::tm* localTime = std::localtime(&currentTime_t); // convert the time_t object to a tm structure

        // Extract individual components of the date and time
        year = localTime->tm_year + 1900; // Years since 1900
        month = localTime->tm_mon + 1;    // Months start from 0
        day = localTime->tm_mday;
    }

    // return value of 'year'
    int getYear() const
    {
        return year;
    }
    // return value of 'month'
    int getMonth() const
    {
        return month;
    }
    // return value of 'day'
    int getDay() const
    {
        return day;
    }

    // return current date as a string
    std::string getDate() const
    {
        std::ostringstream output;
        output « getDay() « '-' « getMonth() « '-' « getYear(); // adding '-' between day, month, and year

        return output.str(); // return as a string
    }

    // return current date + 7 as a string
    std::string getDate7() const
    {
        // make copies of day, month, and year
        int dd = getDay();
        int mm = getMonth();
        int yy = getYear();

        // check for leap year
        bool leap = false;
        if(((yy % 100 == 0) && (yy % 400 == 0)) || ((yy % 100 != 0) && (yy % 4 == 0))) {
            leap = true;
        }

        dd += 7; // add 7 days to the date

        // Adjust the date if necessary
        if((mm == 4 || mm == 6 || mm == 9 || mm == 11) && dd > 30) {
            dd -= 30;
            ++mm;
        }
        else if((mm == 1 || mm == 3 || mm == 5 || mm == 7 || mm == 8 || mm == 10) && dd > 31) {
            dd -= 31;
            ++mm;
        }
        else if(mm == 12 && dd > 31) {
            dd -= 31;
            ++mm;
            ++yy;
        }
        else if(mm == 2 && leap && dd > 29) {
            dd -= 29;
            ++mm;
        }
        else if(mm == 2 && !leap && dd > 28) {
            dd -= 28;
            ++mm;
        }

        // Adjust the year if the month is now January of the next year
        if (mm == 1) {
            ++yy;
        }

        std::ostringstream output;
        output « dd « '-' « mm « '-' « yy; // join day, month, and year

        return output.str(); // return as a string
    }

    // return current day + 14 days
    std::string getDate14() const
    {
        // getting copies of day, month, and year
        int dd = getDay();
        int mm = getMonth();
        int yy = getYear();

        // check for leap year
        bool leap = false;
        if(((yy % 100 == 0) && (yy % 400 == 0)) || ((yy % 100 != 0) && (yy % 4 == 0))) {
            leap = true;
        }

        dd += 14; // add 14 days to the date

        // Adjust the date if necessary
        if((mm == 4 || mm == 6 || mm == 9 || mm == 11) && dd > 30) {
            dd -= 30;
            ++mm;
        }
        else if((mm == 1 || mm == 3 || mm == 5 || mm == 7 || mm == 8 || mm == 10) && dd > 31) {
            dd -= 31;
            ++mm;
        }
        else if(mm == 12 && dd > 31) {
            dd -= 31;
            ++mm;
            ++yy;
        }
        else if(mm == 2 && leap && dd > 29) {
            dd -= 29;
            ++mm;
        }
        else if(mm == 2 && !leap && dd > 28) {
            dd -= 28;
            ++mm;
        }

        // Adjust the year if the month is now January of the next year
        if (mm == 1) {
            ++yy;
        }

        std::ostringstream output;
        output « dd « '-' « mm « '-' « yy; // join all three of them

        return output.str(); // return as a string
    }

    // operator '>=' to check if a date is greater than or equal of another date
    bool operator>=(const Date& other) const
    {
        // If year is greater, the first date is surely greater
        if(year > other.year) {
            return true;
        }
        else if(year < other.year) {
            return false;
        }

        // If years are equal, compare months
        if(month > other.month) {
            return true;
        }
        else if(month < other.month) {
            return false;
        }

        // If months are equal, compare days
        return day >= other.day;
    }
private:
    int year;
    int month;
    int day;
};

// stream extraction operator for ’Date’
std::istream& operator»(std::istream& input, Date& obj)
{
    std::string dateString;

    // taking input as a string
    if (std::getline(input, dateString)) {
        std::istringstream dateStream(dateString); // make stream of string
        char delimiter;

        // extract individual elements from the stream
        dateStream » obj.day » delimiter » obj.month » delimiter » obj.year;
    }

    return input; // for cin » a » b » c
}


// This class is inherited from class ’Teacher’. It only add ’code’ on its base-class
class MyTeacher : public Teacher
{
    friend std::ostream& operator«(std::ostream&, const MyTeacher&); // overloaded stream extraction operator
    friend std::istream& operator»(std::istream&, MyTeacher&); // overloaded stream insertion operator
public:
    MyTeacher() = default; // default constructor
    MyTeacher(const std::string first, const std::string last, int age, const std::string card, bool gen, const std::string phone, const std::string department, const std::string rank, int code) // argumented constructor
        : Teacher(first, last, age, card, gen, phone, department, rank) {
            setCode(code); // set code
        }

    void setCode(int v)
    {
        code = v; // save in ’code’
        return;
    }
    int getCode() const
    {
        return code; // return ’code’
    }

    // overriding ’display’ function of class ’Teacher’
    std::string display() const
    {
        std::ostringstream output; // creating object of class ’ostringstream’
        output « Teacher::display() « ’ ’ « getCode(); // concatenate all data members with spaces in between

        return output.str(); // return it as string
    }
private:
    int code;
};

// stream insertion operator for MyTeacher
std::ostream& operator«(std::ostream& output, const MyTeacher& teacher)
{
    output « "First Name: " « teacher.getFirstName() « "\nLast Name: " « teacher.getLastName() « "\nAge: " « teacher.getAge() « "\nID Card Number: " « teacher.getIdCard() « "\nGender: " « (teacher.getGender() ? "Male" : "Female") « "\nCode: " « teacher.getCode() « "\nPhone Number: " « teacher.getPhoneNumber() « "\nDepartment: " « teacher.getDepartment() « "\nRank: " « teacher.getRank() « ’\n’; // display in a fixed format

    return output; // enablea cout « a « b « c
}
// stream extraction operator for MyTeacher
std::istream& operator»(std::istream& input, MyTeacher& teacher)
```

```
{
    std::string first, last, idCard, phone, department, rank;
    int age, gender, code;

    input » first » last » age » idCard » gender » code » phone; // input all data members
    getline(input, department);
    input » rank;

    // calling set functions to ensure our c-type strings proper handling
    teacher.setFirstName(first);
    teacher.setLastName(last);
    teacher.setAge(age);
    teacher.setIdCard(idCard);
    teacher.setGender(gender);
    teacher.setCode(code);
    teacher.setPhoneNumber(phone);
    teacher.setDepartment(department);
    teacher.setRank(rank);

    return input; // enables cin » a » b » c
}

bool alreadyIssued(int num)
{
    std::ifstream issuedFile{"issued.txt", std::ios::in};
    if(!issuedFile) {
        std::cerr « "cannot open issued file";
        exit(EXIT_FAILURE);
    }

    int unique, key;
    std::string date;

    while(issuedFile » unique » key » date)
    {
        if(unique == num) {
            issuedFile.close();
            return true;
        }
    }

    issuedFile.close();
    return false;
}

#endif
```