

# Library Management System

Generated by Doxygen 1.10.0



<b>1 Library Management System (LMS)</b>	<b>1</b>
1.1 Features	1
1.2 Installation	1
1.3 Usage	1
1.4 Screenshots	2
1.5 Acknowledgments	2
1.6 Authors	2
1.7 Contact	2
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>9</b>
5.1 Book Class Reference	9
5.1.1 Constructor & Destructor Documentation	10
5.1.1.1 Book()	10
5.1.2 Member Function Documentation	10
5.1.2.1 getAvailable()	10
5.1.2.2 getDetails()	11
5.1.2.3 getGenre()	11
5.1.2.4 getIsbn()	12
5.1.2.5 getKey()	12
5.1.2.6 getName()	12
5.1.2.7 setAvailable()	12
5.1.2.8 setGenre()	13
5.1.2.9 setIsbn()	13
5.1.2.10 setKey()	13
5.1.2.11 setName()	14
5.2 Date Class Reference	14
5.2.1 Constructor & Destructor Documentation	15
5.2.1.1 Date()	15
5.2.2 Member Function Documentation	15
5.2.2.1 getDate()	15
5.2.2.2 getDate14()	16
5.2.2.3 getDate7()	16
5.2.2.4 getDay()	17
5.2.2.5 getMonth()	17
5.2.2.6 getYear()	17

5.2.2.7 operator>=()	18
5.2.3 Friends And Related Symbol Documentation	18
5.2.3.1 operator>>	18
5.3 MyTeacher Class Reference	18
5.3.1 Constructor & Destructor Documentation	19
5.3.1.1 MyTeacher() [1/2]	19
5.3.1.2 MyTeacher() [2/2]	20
5.3.2 Member Function Documentation	20
5.3.2.1 display()	20
5.3.2.2 getCode()	20
5.3.2.3 setCode()	21
5.3.3 Friends And Related Symbol Documentation	21
5.3.3.1 operator<<	21
5.3.3.2 operator>>	21
5.4 Password Class Reference	21
5.4.1 Constructor & Destructor Documentation	22
5.4.1.1 Password()	22
5.4.2 Member Function Documentation	22
5.4.2.1 getDetails()	22
5.4.2.2 setPassword()	23
<b>6 File Documentation</b>	<b>25</b>
6.1 addStudent.cpp File Reference	25
6.1.1 Function Documentation	25
6.1.1.1 main()	25
6.2 addTeacher.cpp File Reference	26
6.2.1 Function Documentation	26
6.2.1.1 main()	26
6.3 book.hpp File Reference	26
6.4 book.hpp	27
6.5 encryption.hpp File Reference	28
6.6 encryption.hpp	30
6.7 headings.hpp File Reference	31
6.7.1 Function Documentation	32
6.7.1.1 doFunction()	32
6.7.1.2 newLibrarian()	33
6.7.1.3 showOptions()	33
6.7.1.4 startingInstructions()	34
6.8 headings.hpp	35
6.9 librarianConfirmation.hpp File Reference	37
6.9.1 Enumeration Type Documentation	38
6.9.1.1 IN	38

6.9.2 Function Documentation . . . . .	39
6.9.2.1 getPassword() . . . . .	39
6.9.2.2 librarianLogin() . . . . .	39
6.10 librarianConfirmation.hpp . . . . .	40
6.11 libraryFunctions.hpp File Reference . . . . .	41
6.11.1 Function Documentation . . . . .	42
6.11.1.1 addNewBooks() . . . . .	42
6.11.1.2 addNewLibrarian() . . . . .	42
6.11.1.3 borrowABook() . . . . .	43
6.11.1.4 checkStudentDetails() . . . . .	44
6.11.1.5 removeOldBook() . . . . .	44
6.11.1.6 renewDate() . . . . .	45
6.11.1.7 returnABook() . . . . .	45
6.11.1.8 searchABook() . . . . .	46
6.12 libraryFunctions.hpp . . . . .	46
6.13 README.md File Reference . . . . .	58
6.14 test.cpp File Reference . . . . .	58
6.14.1 Function Documentation . . . . .	59
6.14.1.1 main() . . . . .	59
6.15 utilitys.hpp File Reference . . . . .	60
6.15.1 Function Documentation . . . . .	61
6.15.1.1 alreadyIssued() . . . . .	61
6.15.1.2 operator<<() . . . . .	61
6.15.1.3 operator>>() [1/2] . . . . .	61
6.15.1.4 operator>>() [2/2] . . . . .	61
6.16 utilitys.hpp . . . . .	62



# 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, returning 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](mailto:davidptrk7@gmail.com).



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Book . . . . .	9
Date . . . . .	14
Password . . . . .	21
Teacher	
MyTeacher . . . . .	18



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Book</a>	9
<a href="#">Date</a>	14
<a href="#">MyTeacher</a>	18
<a href="#">Password</a>	21



# Chapter 4

## File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">addStudent.cpp</a>	25
<a href="#">addTeacher.cpp</a>	26
<a href="#">book.hpp</a>	26
<a href="#">encryption.hpp</a>	28
<a href="#">headings.hpp</a>	31
<a href="#">librarianConfirmation.hpp</a>	37
<a href="#">libraryFunctions.hpp</a>	41
<a href="#">test.cpp</a>	58
<a href="#">utilities.hpp</a>	60



## Chapter 5

# Class Documentation

### 5.1 Book Class Reference

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

Collaboration diagram for Book:

Book
+ Book() + setKey() + getKey() + setName() + getName() + setGenre() + getGenre() + setIsbn() + getIsbn() + setAvailable() + getAvailable() + getDetails()

#### Public Member Functions

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

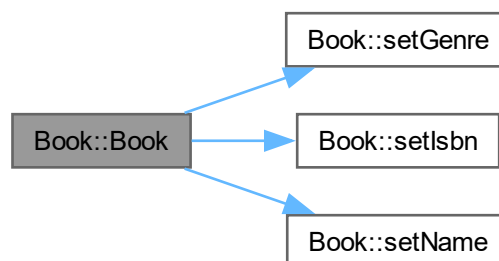
- int `getKey` () const
- void `setName` (std::string bookName)
- std::string `getName` () const
- void `setGenre` (std::string bookGenre)
- std::string `getGenre` () const
- void `setIsbn` (std::string isbnum)
- std::string `getIsbn` () const
- void `setAvailable` (bool n)
- bool `getAvailable` () const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

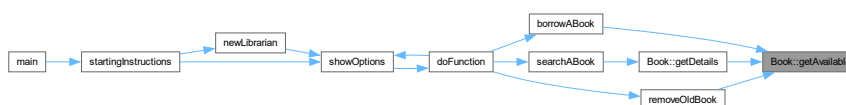


## 5.1.2 Member Function Documentation

### 5.1.2.1 getAvailable()

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

Here is the caller graph for this function:

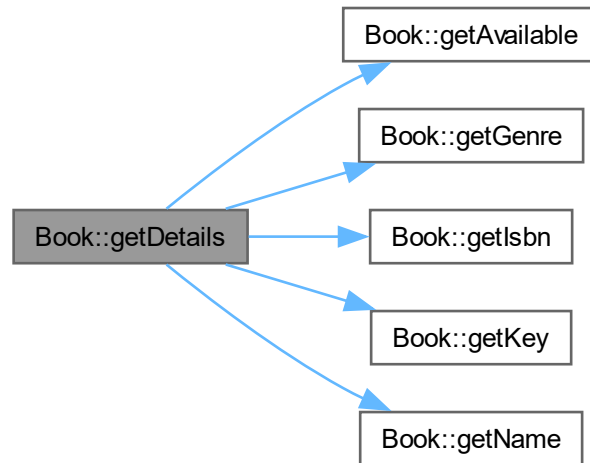




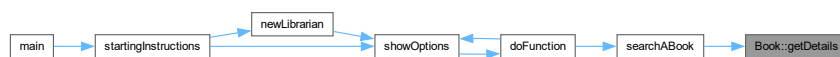
### 5.1.2.2 getDetails()

```
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]
```

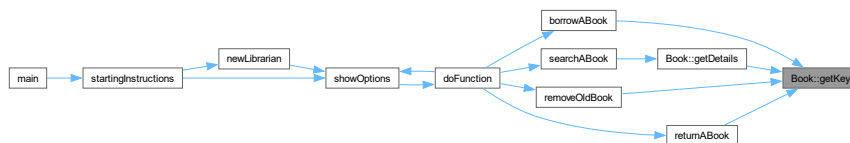
Here is the caller graph for this function:



#### 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]
```

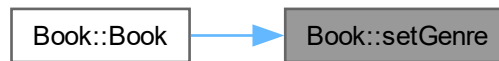
Here is the caller graph for this function:



### 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]
```

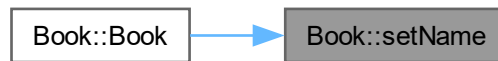
Here is the caller graph for this function:



#### 5.1.2.11 setName()

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

Here is the caller graph for this function:



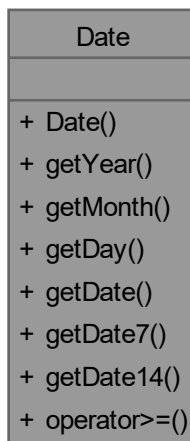
The documentation for this class was generated from the following file:

- [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()

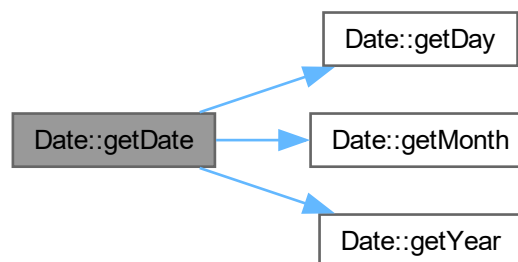
```
Date::Date ( ) [inline]
```

## 5.2.2 Member Function Documentation

### 5.2.2.1 getDate()

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

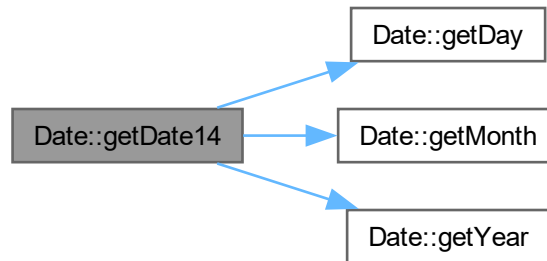
Here is the call graph for this function:



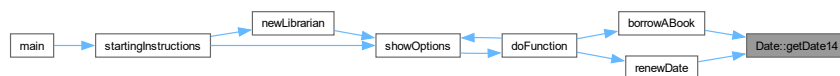
### 5.2.2.2 getDate14()

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

Here is the call graph for this function:



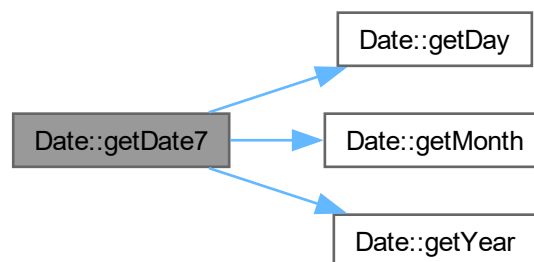
Here is the caller graph for this function:



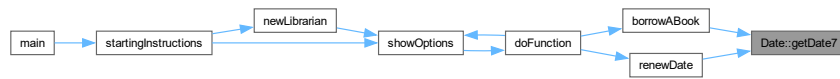
### 5.2.2.3 getDate7()

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

Here is the call graph for this function:



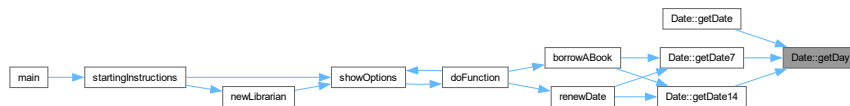
Here is the caller graph for this function:



#### 5.2.2.4 getDate()

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

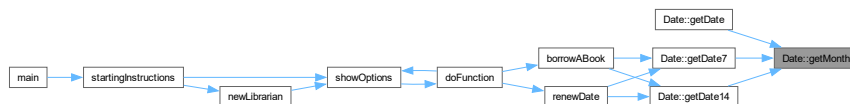
Here is the caller graph for this function:



#### 5.2.2.5 getMonth()

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

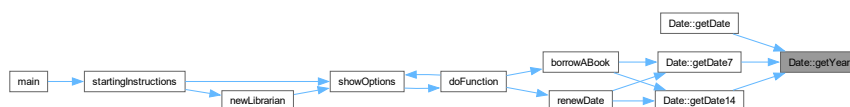
Here is the caller graph for this function:



#### 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]
```

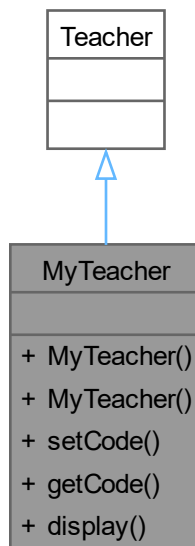
The documentation for this class was generated from the following file:

- [utilitys.hpp](#)

## 5.3 MyTeacher Class Reference

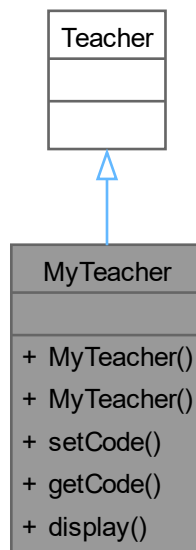
```
#include <utilitys.hpp>
```

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]
```

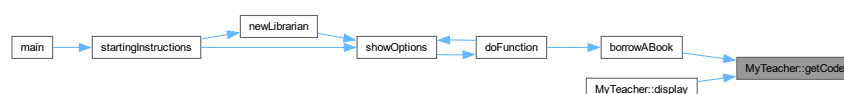
Here is the call graph for this function:



### 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]
```

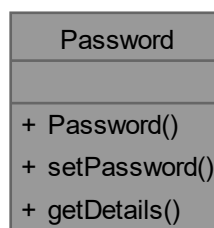
The documentation for this class was generated from the following file:

- [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

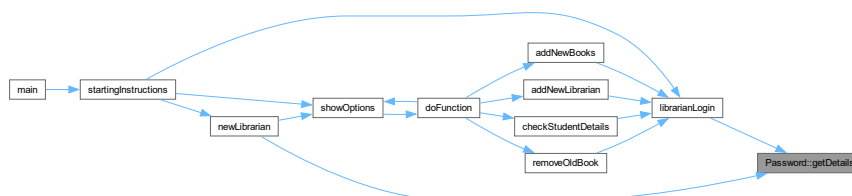
### 5.4.2.1 getDetails()

```

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

```

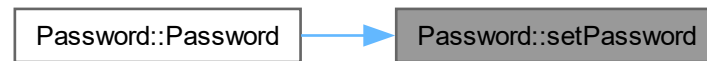
Here is the caller graph for this function:



### 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](#)

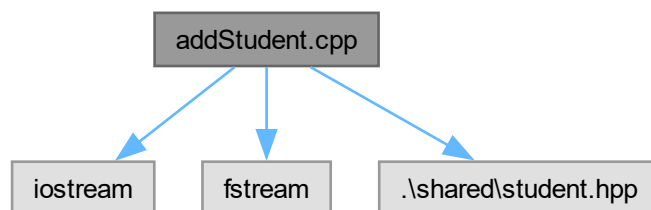


## Chapter 6

# 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` ()

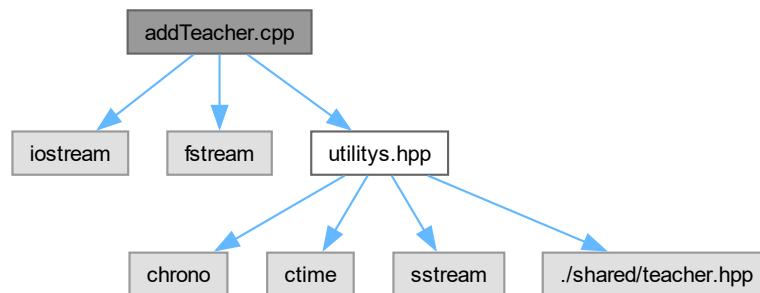
#### 6.1.1 Function Documentation

##### 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` ()

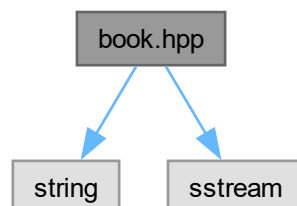
### 6.2.1 Function Documentation

#### 6.2.1.1 `main()`

```
int main ( )
```

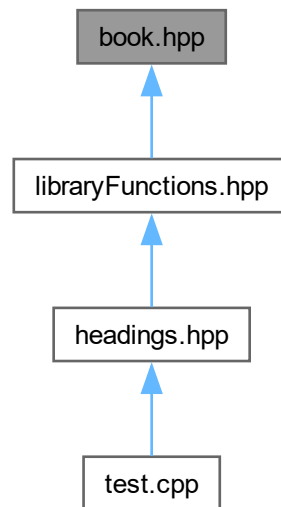
## 6.3 book.hpp File Reference

```
#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.](#)

```

00001 // 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
00002
00003 #ifndef BOOK_HPP
00004 #define BOOK_HPP
00005
00006 // preprocessor directives
00007 #include <string>
00008 #include <sstream>
00009
00010 class Book
00011 {
00012 public:
00013     // constructor provide default value for all attributes
00014     // it help to make object without providing actual data
00015     Book(int keyV = -1, std::string isbn = "", std::string name = "", std::string genre = "", bool
availableV = false)
00016         : key{keyV}, available{availableV} {
00017         // functions must be call since 'string' must be converted to 'C-typed string'
00018         setIsbn(isbn);
00019         setName(name);
00020         setGenre(genre);
00021     }
00022
00023     // for data memeber 'key'
00024     void setKey(int num)
00025     {
00026         this->key = num; // store key
00027     }
00028     int getKey() const
00029     {

```

```

00030         return key; // return key
00031     }
00032
00033     // for data member 'name'
00034     void setName(std::string bookName)
00035     {
00036         size_t length = bookName.size(); // get size of string
00037         length = length < 100 ? length : 99; // size must be one less then size of 'name'
00038         bookName.copy(name, length); // copy characters from string to 'name'
00039         name[length] = '\0'; // add terminating letter at end of 'name'
00040     }
00041     std::string getName() const
00042     {
00043         std::string bookName(name); // convert 'name' array to string
00044         return bookName; // return as string
00045     }
00046
00047     // for data member 'genre'
00048     void setGenre(std::string bookGenre)
00049     {
00050         size_t length = bookGenre.size(); // get size of string
00051         length = length < 30 ? length : 29; // size must be one less then size of 'genre'
00052         bookGenre.copy(genre, length); // copy characters from string to 'genre'
00053         genre[length] = '\0'; // add terminating letter at end of 'genre'
00054     }
00055     std::string getGenre() const
00056     {
00057         std::string bookGenre(genre); // convert 'genre' array to string
00058         return bookGenre; // return as string
00059     }
00060
00061     // for data member 'isbn'
00062     void setIsbn(std::string isbnnum)
00063     {
00064         size_t length = isbnnum.size(); // get size of string
00065         length = length < 14 ? length : 13; // size must be one less then size of 'isbn'
00066         isbnnum.copy(isbn, length); // copy characters from string to 'isbn'
00067         isbn[length] = '\0'; // add terminating letter at end of 'isbn'
00068     }
00069     std::string getIsbn() const
00070     {
00071         std::string bookIsbn(isbn); // convert 'isbn' array to string
00072         return bookIsbn; // return as string
00073     }
00074
00075     // for data member 'available'
00076     void setAvailable(bool n)
00077     {
00078         available = n;
00079     }
00080     bool getAvailable() const
00081     {
00082         // this tell if book is available for issuing it to people
00083         return available;
00084     }
00085
00086     // to concatenate all data
00087     std::string getDetails() const
00088     {
00089         std::ostringstream output;
00090         output << getKey() << ' ' << getName() << ' ' << getIsbn() << ' ' << getGenre() << ' ' <<
getAvailable(); // concatenate all member with spaces in between
00091
00092         return output.str(); // return as string
00093     }
00094 private:
00095     int key; // unique key of book
00096     bool available; // availability of book
00097
00098     // 'c-type strings' are used since it is fixed sized, and fixed size is necessary for writing
    records in binary file
00099     char isbn[14]; // isbn of book
00100     char name[100]; // name of book
00101     char genre[30]; // genre of book
00102 };
00103
00104 #endif

```

## 6.5 encryption.hpp File Reference

```

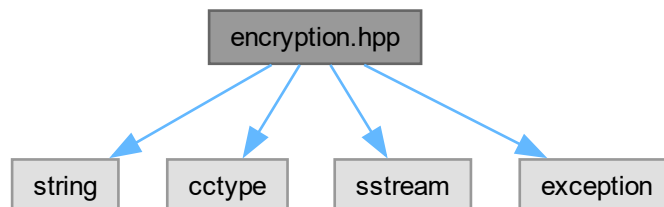
#include <string>
#include <cctype>

```

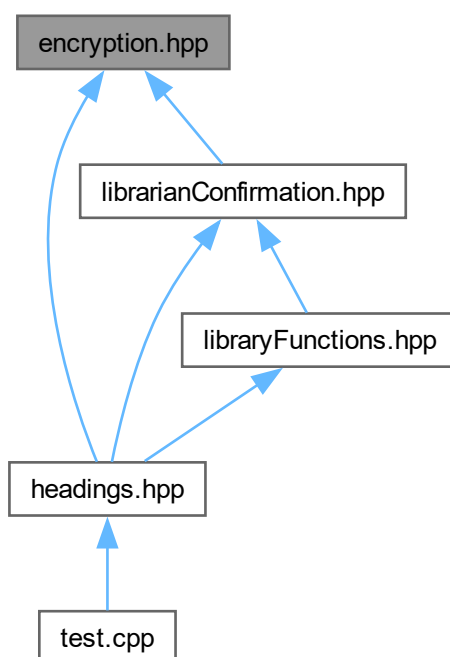
```
#include <sstream>
```

```
#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.](#)

```

00001 // This header file consist of class Password. This class is used to encrypt user password. As an
      object of class Password is instantiated, the password is already stored in encrypted form.
00002
00003 #ifndef ENCRYPTION_HPP
00004 #define ENCRYPTION_HPP
00005
00006 // preprocessor directives
00007 #include <string>
00008 #include <cctype>
00009 #include <sstream>
00010 #include <exception>
00011
00012 // class will encrypt the password entered by librarian
00013 // if return name and encrypted password concatenated
00014 class Password
00015 {
00016 public:
00017     // constructor
00018     Password(std::string name, std::string password) // cannot instantiate object without name and
      password supplied
00019     : librarianName{name} {
00020         setPassword(password); // call 'setPassword' to encrypt it
00021     }
00022
00023     // setPassword can be accessed outside of class
00024     void setPassword(std::string value)
00025     {
00026         encryptPassword(value); // call 'encryptPassword' to encrypt the password
00027         this->password = value; // store in data member 'password'
00028
00029         return; // return control
00030     }
00031
00032     // to return encrypted password attached with user name
00033     std::string getDetails(void) const
00034     {
00035         std::ostringstream output;
00036         output << librarianName << password; // concatenate both
00037
00038         return output.str(); // return them as string
00039     }
00040 private:
00041     // data member to store password
00042     // password will be store in encrypted form only
00043     std::string password;
00044
00045     // data member to store name
00046     std::string librarianName;
00047
00048     // shift key
00049     // it is used to encrypt the password
00050     const int SHIFT_VALUE{7};
00051
00052     // utility functions
00053     void encryptPassword(std::string& value) // function to shift every letter in password
00054     {
00055         // loop to iterate over every character of 'value' i.e. password entered by user
00056         for(char& element : value)
00057         {
00058             // if character is lower-case
00059             if(islower(element)) {
00060                 shiftCharacter(element, 'a', 'z'); // to shift character in range a-z
00061             }
00062             // if character is upper-case
00063             else if(isupper(element)) {
00064                 shiftCharacter(element, 'A', 'Z'); // to shift character in range A-Z
00065             }
00066             // if character is digit
00067             else if(isdigit(element)) {
00068                 shiftCharacter(element, '0', '9'); // to shift character in range 0-9
00069             }
00070             // no other character is allowed
00071             else {
00072                 throw std::runtime_error{"password must be between a-z, A-Z, 0-9"}; // throw an
      exception
00073             }
00074         }
00075
00076         return; // return control
00077     }
00078
00079     void shiftCharacter(char& character, char start, char end) // to shift each individual letter by
      shift key

```

```

00080     {
00081         for(int i{1}; i <= SHIFT_VALUE; ++i)
00082         {
00083             ++character; // increment individual letter
00084
00085             // if letter is greater than its range
00086             if(character > end) {
00087                 character = start; // move to start of range
00088             }
00089         }
00090
00091         return; // return control
00092     }
00093 };
00094
00095 #endif

```

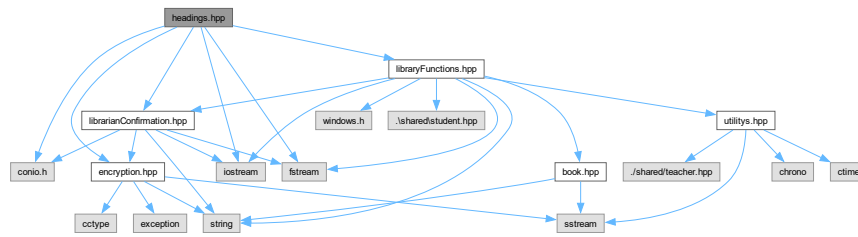
## 6.7 headings.hpp File Reference

```

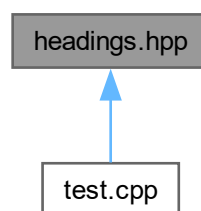
#include <iostream>
#include <fstream>
#include <conio.h>
#include "encryption.hpp"
#include "libraryFunctions.hpp"
#include "librarianConfirmation.hpp"

```

Include dependency graph for headings.hpp:



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



### Functions

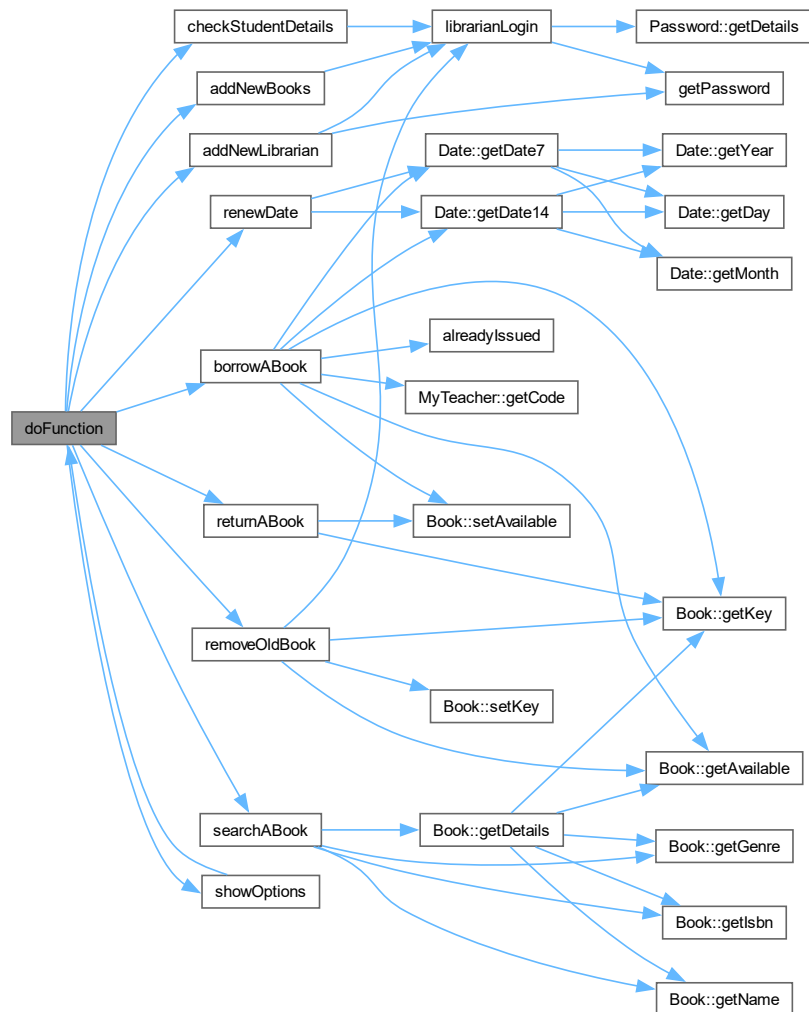
- void [startingInstructions](#) (void)
- void [newLibrarian](#) (void)
- void [showOptions](#) (void)
- void [doFunction](#) (int)

## 6.7.1 Function Documentation

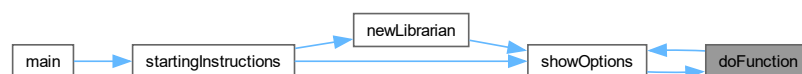
### 6.7.1.1 doFunction()

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

Here is the call graph for this function:



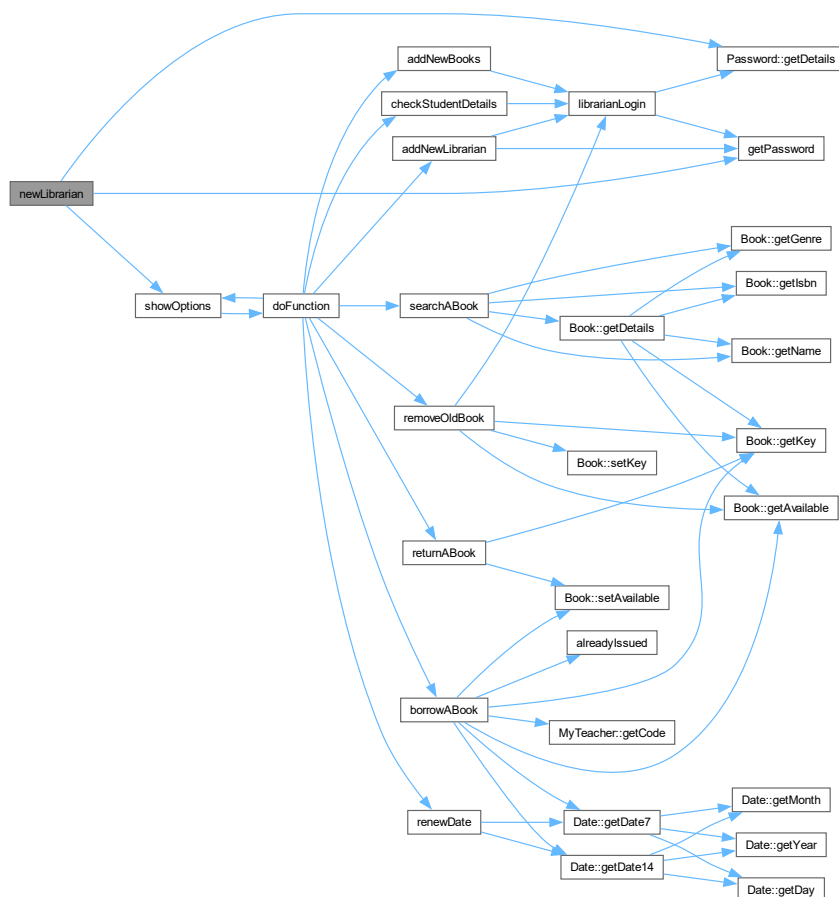
Here is the caller graph for this function:



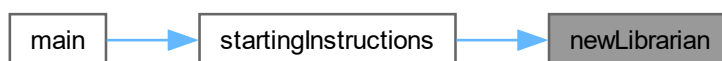
### 6.7.1.2 newLibrarian()

```
void newLibrarian (
    void )
```

Here is the call graph for this function:



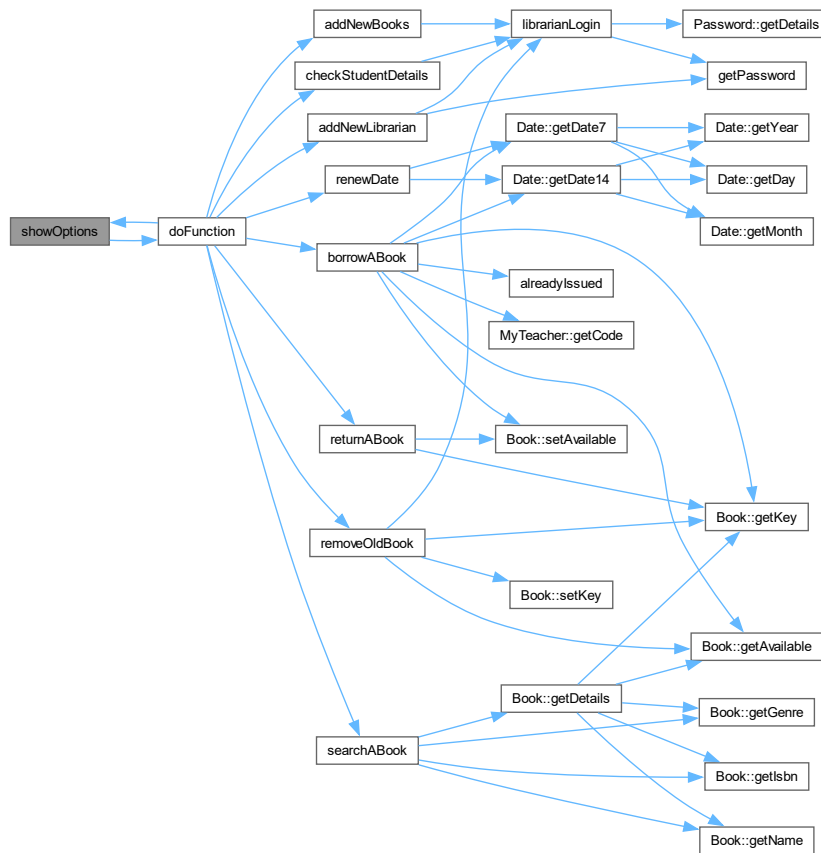
Here is the caller graph for this function:



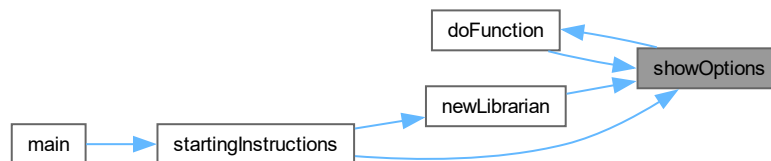
### 6.7.1.3 showOptions()

```
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 )

```





```

00014 void newLibrarian(void); // function to create new Librarian
00015 void showOptions(void); // navigation options of library
00016 void doFunction(int); // call proper function as specified by user
00017
00018 void startingInstructions()
00019 {
00020     // welcome display
00021     std::cout << "-----" << std::endl;
00022     std::cout << "        Welcome to Library Management System        " << std::endl;
00023     std::cout << "-----" << std::endl;
00024
00025     std::ifstream dataFile{"librarian.txt", std::ios::in}; // try to open librarian.txt
00026
00027     // if file doesn't exist, no librarian exists
00028     if(!dataFile) {
00029         newLibrarian(); // create new librarian
00030     }
00031     // if file exist, at least one librarian exists
00032     else {
00033         std::cout << "\nLibrarian Login\n"; // librarian login
00034
00035         // call 'librarianLogin' to perform login, it return true or false
00036         if(librarianLogin()) {
00037             // if librarian is logged in
00038             // show options since he is librarian
00039
00040             system("cls"); // clear screen
00041             showOptions(); // show navigation options
00042         }
00043         // else show message since he is not librarian
00044         else {
00045             std::cout << "\nAccess Denied" << std::endl;
00046         }
00047     }
00048
00049     dataFile.close(); // close the file
00050     return; // return control
00051 }
00052
00053 void newLibrarian(void) // to create new librarian
00054 {
00055     std::ofstream dataFile{"librarian.txt", std::ios::out}; // create new file
00056     // if file is not created, exit the program
00057     if(!dataFile) {
00058         std::cerr << "cannot create file";
00059         exit(EXIT_FAILURE);
00060     }
00061
00062     std::string name, password; // to hold typed data
00063
00064     // new librarian signup
00065     std::cout << "\nLibrarian Signup\n";
00066     std::cout << "\nEnter your Name: "; // prompt to enter name
00067     getline(std::cin, name); // using 'getline' to insert spaces
00068     std::cout << "Enter new Password (only letters or digits): "; // prompt to enter password
00069     password = getPassword(); // using 'getPassword' to hide it from console
00070
00071     // create Password object to encrypt password
00072     Password newPerson{name, password};
00073     dataFile << newPerson.getDetails() << std::endl; // store the data in file
00074
00075     // show options since he is new librarian
00076     system("cls"); // clear screen
00077     showOptions();
00078
00079     dataFile.close(); // close the file
00080
00081     return; // return control
00082 }
00083
00084 void showOptions() // library navigation
00085 {
00086     // these options will require librarian password again
00087     std::cout << "Librarian's Access:\n";
00088     std::cout << "1 - Add Another Librarian" << '\t' << "2 - Add New Books" << '\n' << "3 - Remove Old
Books" << "\t\t" << "4 - Check Student Details" << std::endl;
00089
00090     // these options will not require password again
00091     // anyone can check them
00092     std::cout << "\nPublic's Access:\n";
00093     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;
00094
00095     const int LOWEST_OPTION{1};
00096     const int HIGHEST_OPTION{9};
00097
00098     int num;

```

```

00099 // loop until a number in range of lowest-highest is entered
00100 do
00101 {
00102     std::cout << "? ";
00103     std::cin >> num;
00104 }while(num < LOWEST_OPTION || num > HIGHEST_OPTION);
00105
00106 // option 9 is exit library
00107 // so if 9 is entered do not call 'doFunction'
00108 if(num != 9) {
00109     // call 'doFunction; which will in response call function associated with entered number
00110     doFunction(num);
00111 }
00112
00113 // if 9 is entered by user
00114 // exit the library
00115 std::cout << "\nThankYou!" << std::endl;
00116
00117 exit(EXIT_SUCCESS); // exit the application, this is ending point of application
00118 }
00119
00120 void doFunction(int choice) // to call specified function as entered by user
00121 {
00122     // send control to case as entered by user
00123     switch(choice)
00124     {
00125         case 1: // new librarian
00126             addNewLibrarian();
00127             break;
00128         case 2: // new books
00129             addNewBooks();
00130             break;
00131         case 3: // remove books
00132             removeOldBook();
00133             break;
00134         case 4: // check student details
00135             checkStudentDetails();
00136             break;
00137         case 5: // borrow a book
00138             borrowABook();
00139             break;
00140         case 6: // return a book
00141             returnABook();
00142             break;
00143         case 7: // renew date
00144             renewDate();
00145             break;
00146         case 8: // search books
00147             searchABook();
00148             break;
00149         default:
00150             // control should not reach here
00151             std::cerr << "control should not reach here";
00152             break;
00153     }
00154
00155     std::cout << "\n\nPress any key to continue..."; // it is displayed at end of working of every
option
00156     getch(); // wait for a key
00157     system("cls"); // clear screen
00158
00159     // again show options
00160     showOptions();
00161 }
00162
00163 #endif

```

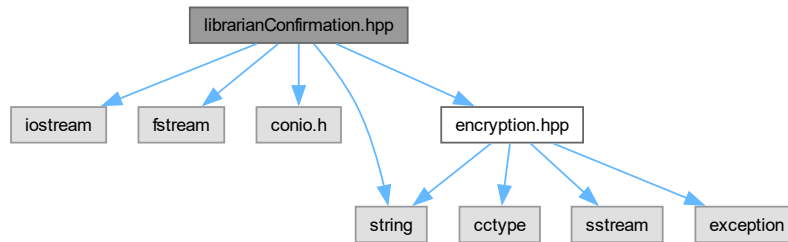
## 6.9 librarianConfirmation.hpp File Reference

```

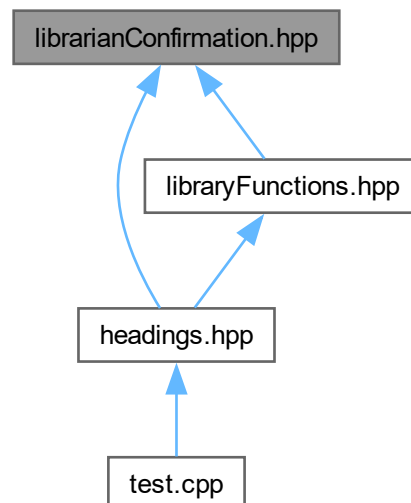
#include <iostream>
#include <fstream>
#include <conio.h>
#include <string>
#include "encryption.hpp"

```

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

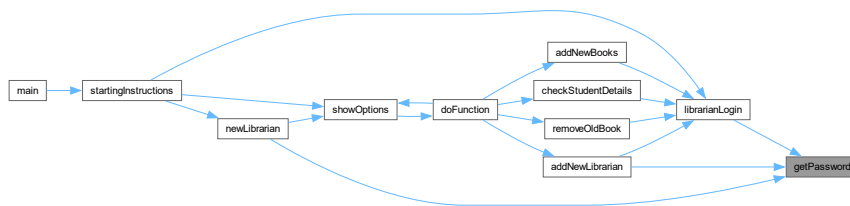
IN_BACK	
IN_RET	

## 6.9.2 Function Documentation

## 6.9.2.1 getPassword()

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

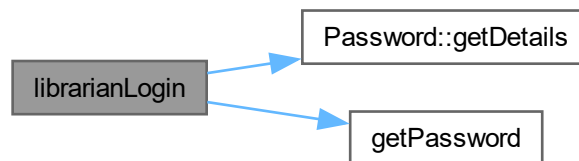
Here is the caller graph for this function:



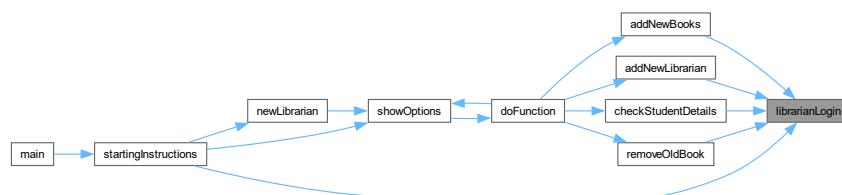
## 6.9.2.2 librarianLogin()

```
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.](#)

```

00001 // 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
00002
00003 #ifndef CONFIRMATION_HPP
00004 #define CONFIRMATION_HPP
00005
00006 // preprocessor directives
00007 #include <iostream>
00008 #include <fstream>
00009 #include <conio.h>
00010 #include <string>
00011 #include "encryption.hpp"
00012
00013 enum IN
00014 {
00015     IN_BACK = 8, // ASCII value for Backspace
00016     IN_RET = 13 // ASCII value for Enter/Return
00017 };
00018
00019 // return the password entered by user
00020 // hide password typed by character supplied in 'sp'
00021 std::string getPassword(char sp = '*')
00022 {
00023     std::string password = ""; // to store password
00024     char ch_ipt; // to store character as it is typed
00025
00026     // loop continues until Enter key is pressed
00027     while(true)
00028     {
00029         ch_ipt = getch(); // input a character
00030
00031         // if Enter is pressed
00032         if(ch_ipt == IN::IN_RET) {
00033             std::cout << std::endl; // print a newline
00034             return password; // return password
00035         }
00036         // if Backspace is pressed and password is not empty
00037         else if((ch_ipt == IN::IN_BACK) && (password.length() != 0)) {
00038             password.pop_back(); // remove character from end of string
00039             std::cout << "\b\b"; // update console by backspace
00040
00041             continue; // move to next iteration
00042         }
00043         // if Backspace is pressed and password is empty
00044         else if((ch_ipt == IN::IN_BACK) && (password.length() == 0)) {
00045             continue; // do nothing move to next iteration
00046         }
00047
00048         // if any other key is pressed
00049         password.push_back(ch_ipt); // push it onto string
00050         std::cout << sp; // and display 'sp' character on screen in place of charcter typed
00051     }
00052 }
00053
00054 // call this function simply when you want to login a librarian
00055 // all input and checking is performed by it
00056 // return bool value to indicate if login was successful or not
00057 bool librarianLogin()
00058 {
00059     std::ifstream dataFile("librarian.txt", std::ios::in); // open librarian file
00060     // if file doesn't exist
00061     if(!dataFile) {
00062         std::cerr << "file doesn't exists";
00063         exit(EXIT_FAILURE); // exit since no librarian exist, exist is necessary because new librarian
      is not made as startup of application
00064     }
00065
00066     std::string name, password; // to hold user typed data
00067
00068     std::cout << "\nEnter your Name: "; // prompt to enter name
00069     getline(std::cin, name); // used 'getline' because name may consist of space
00070     std::cout << "Enter your Password: "; // prompt to enter password
00071     password = getPassword(); // call 'getPassword' so it can hide typed password
00072
00073     Password loginPerson{name, password}; // creating object of Password, because this class can
      automatically encrypt the password
00074     std::string thisData = loginPerson.getDetails(); // get concatenated name and password(encrypted)
00075
00076     std::string fileRecord; // to hold librarian data from file
00077
00078     // loop until all lines of file

```

```

00079     while(getline(dataFile, fileRecord)) // get a line from file, since name and password (of one
        librarian) are stored in one line
00080     {
00081         // if it match with typed data
00082         if(fileRecord == thisData) {
00083             dataFile.close(); // close the file
00084             return true; // and return true, since data matched and authentication is done
00085         }
00086     }
00087     // if the loop ends and not one record in file matched with typed data
00088     dataFile.close(); // close the file
00089     return false; // and return false, since no data matched
00091 }
00092
00093 #endif

```

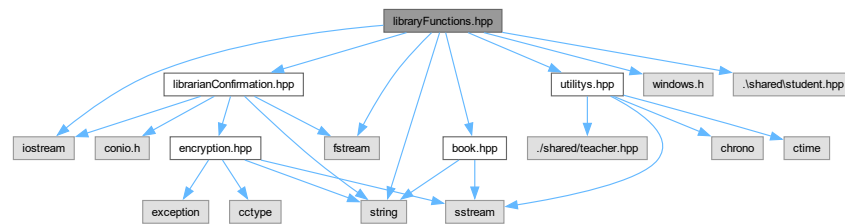
## 6.11 libraryFunctions.hpp File Reference

```

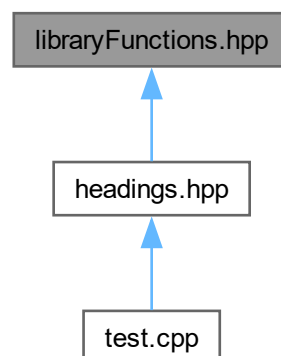
#include <iostream>
#include <string>
#include <fstream>
#include <windows.h>
#include "book.hpp"
#include "librarianConfirmation.hpp"
#include "..\shared\student.hpp"
#include "utilitys.hpp"

```

Include dependency graph for libraryFunctions.hpp:



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



## Functions

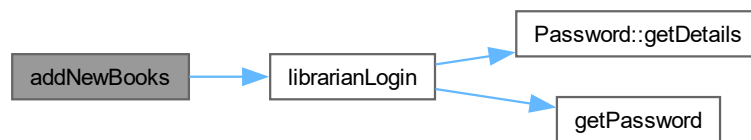
- void [addNewLibrarian](#) ()
- void [addNewBooks](#) ()
- void [removeOldBook](#) ()
- void [checkStudentDetails](#) ()
- void [borrowABook](#) ()
- void [returnABook](#) ()
- void [renewDate](#) ()
- void [searchABook](#) ()

### 6.11.1 Function Documentation

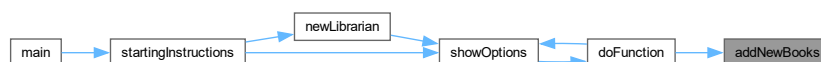
#### 6.11.1.1 addNewBooks()

```
void addNewBooks ( )
```

Here is the call graph for this function:



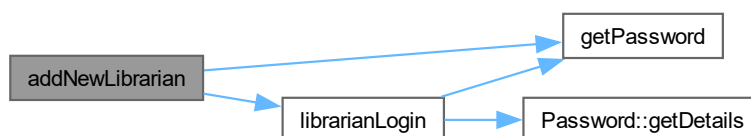
Here is the caller graph for this function:



#### 6.11.1.2 addNewLibrarian()

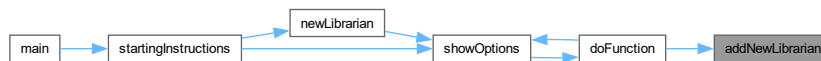
```
void addNewLibrarian ( )
```

Here is the call graph for this function:





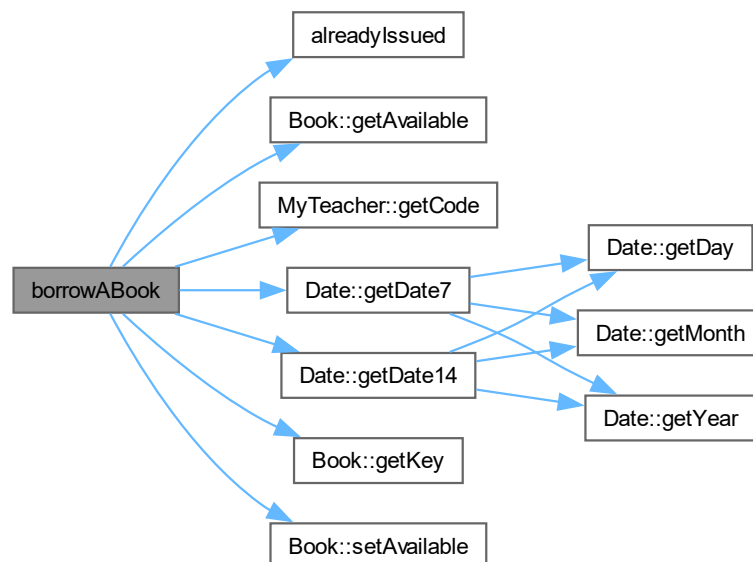
Here is the caller graph for this function:



### 6.11.1.3 borrowABook()

```
void borrowABook ( )
```

Here is the call graph for this function:



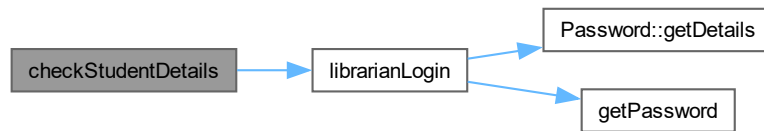
Here is the caller graph for this function:



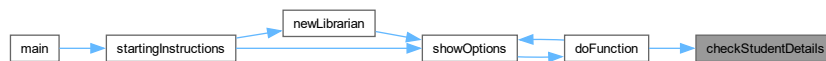
#### 6.11.1.4 checkStudentDetails()

```
void checkStudentDetails ( )
```

Here is the call graph for this function:



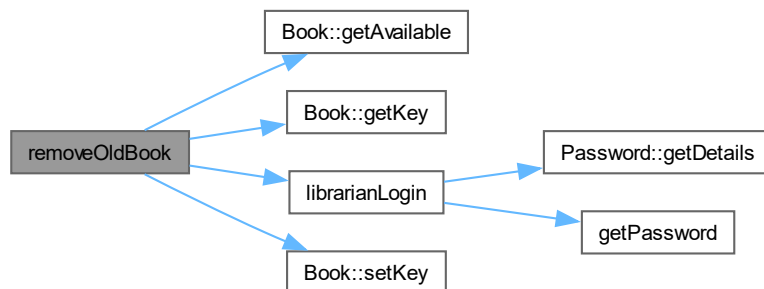
Here is the caller graph for this function:



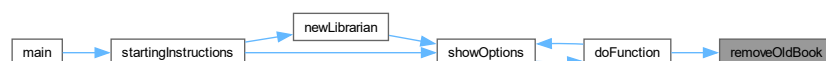
#### 6.11.1.5 removeOldBook()

```
void removeOldBook ( )
```

Here is the call graph for this function:



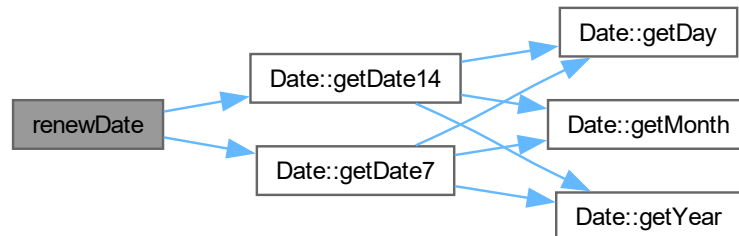
Here is the caller graph for this function:



### 6.11.1.6 renewDate()

```
void renewDate ( )
```

Here is the call graph for this function:



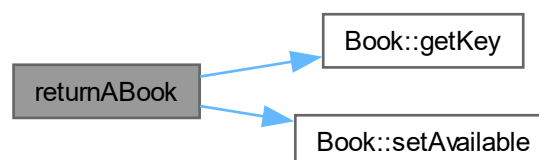
Here is the caller graph for this function:



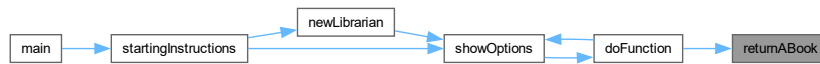
### 6.11.1.7 returnABook()

```
void returnABook ( )
```

Here is the call graph for this function:



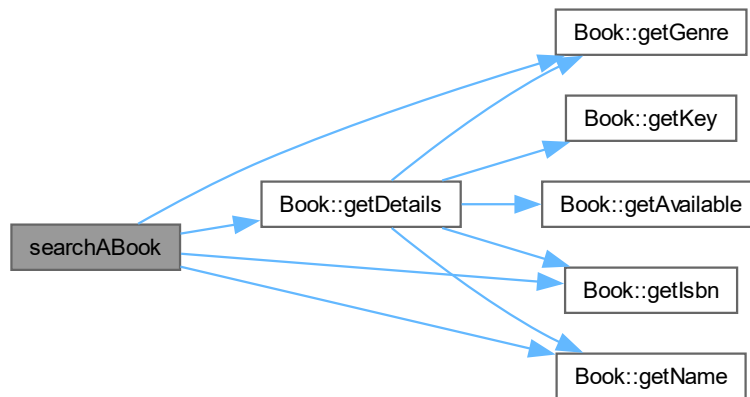
Here is the caller graph for this function:



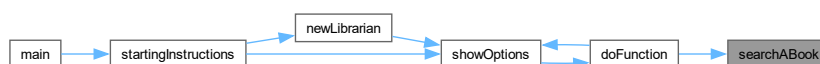
### 6.11.1.8 searchABook()

```
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.](#)

```

00001 // 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
00002
00003 #ifndef LIBRARYFUNCTIONS_HPP
00004 #define LIBRARYFUNCTIONS_HPP
00005

```

```

00006 #include <iostream>
00007 #include <string>
00008 #include <fstream>
00009 #include <windows.h>
00010 #include "book.hpp" // definition of class 'Book' to make record of book
00011 #include "librarianConfirmation.hpp" // for librarian login
00012 #include ".\shared\student.hpp" // definition of class 'Student'
00013 #include "utilitys.hpp" // definition of class 'Date' and 'MyTeacher'
00014
00015 // 1 - Add New Librarian
00016 void addNewLibrarian()
00017 {
00018     system("cls"); // clear screen
00019
00020     std::cin.ignore(); // ignore last typed Enter
00021
00022     // if librarian is logged in
00023     if(librarianLogin()) {
00024         std::string name, password; // to hold data
00025
00026         system("cls"); // clear screen
00027         std::cout << "Enter New Librarian Name: "; // prompt for name
00028         getline(std::cin, name);
00029         std::cout << "Enter new Password (only letters or digits): "; // prompt for password
00030         password = getPassword();
00031
00032         Password newPerson{name, password}; // create object of 'Password' using typed data
00033
00034         std::fstream file{"librarian.txt", std::ios::in | std::ios::out | std::ios::app}; // open
librarian file
00035         if(!file) {
00036             std::cerr << "librarian file doesn't open";
00037             exit(EXIT_FAILURE);
00038         }
00039
00040         file << newPerson.getDetails() << '\n'; // print new librarian data in file as a record
00041         std::cout << "\nLibrarian Added"; // display message
00042     }
00043     // if not logged in
00044     else {
00045         std::cout << "\nAccess Denied"; // display message
00046     }
00047
00048     return; // return control
00049 }
00050
00051 // 2 - Add New Books
00052 void addNewBooks()
00053 {
00054     system("cls"); // clear screen
00055
00056     std::cin.ignore(); // ignore last typed Enter
00057
00058     // if librarian is logged in
00059     if(librarianLogin()) {
00060         std::fstream booksFile{"bookRecords.dat", std::ios::out | std::ios::in | std::ios::binary |
std::ios::app}; // open books record file
00061
00062         if(!booksFile) {
00063             std::cerr << "Cannot open file to add books";
00064             exit(EXIT_FAILURE);
00065         }
00066
00067         // variables to hold information of book
00068         int key;
00069         std::string isbn;
00070         std::string name;
00071         std::string genre;
00072         bool available;
00073
00074         int next; // used by loop
00075
00076         do
00077         {
00078             system("cls"); // clear screen
00079
00080             //deciding key, since it is generated automatically
00081             booksFile.seekg(0, std::ios::end); // move to end of file
00082             int fileSize = booksFile.tellg(); // get size of file
00083             int numBooks = fileSize / sizeof(Book); // divide full file size to size of one record to
get number of books
00084             key = numBooks > 0 ? numBooks : 0; // set key
00085
00086             std::cout << "Enter ISBN: "; // prompt for isbn
00087             std::cin >> isbn;
00088             std::cout << "Enter Book Name: "; // prompt for name
00089             std::cin.ignore();

```

```

00090         getline(std::cin, name); // name may include spaces
00091         std::cout << "Enter Book Genre: "; // prompt for genre
00092         std::cin >> genre;
00093
00094         available = true; // set availability to true
00095
00096         Book newBookRecord{key, isbn, name, genre, available}; // make an object using above typed
data
00097
00098         booksFile.write(reinterpret_cast<const char *>(&newBookRecord), sizeof(Book)); // write
this record in binary file
00099
00100         std::cout << "Book Added" << std::endl; // successful message
00101
00102         std::cout << "\nAdd another book(1 or 0): "; // if want to add more books
00103         std::cin >> next;
00104         }while(next == 1);
00105     }
00106     // if not logged in
00107     else {
00108         std::cout << "\nAccess Denied"; // show message
00109     }
00110
00111     return; // return control
00112 }
00113
00114 // 3 - Remove Old Book
00115 void removeOldBook()
00116 {
00117     system("cls"); // clear screen
00118
00119     std::cin.ignore(); // ignore last 'enter' from the stream
00120
00121     // if librarian is logged in
00122     if(librarianLogin()) {
00123         std::ifstream inputFile{"bookRecords.dat", std::ios::in | std::ios::binary}; // open book
records file
00124         if(!inputFile) {
00125             std::cerr << "Cannot open file to remove book";
00126             exit(EXIT_FAILURE);
00127         }
00128
00129         // check if any book is issued to someone
00130         int availFlag = 1;
00131         Book checkBooks;
00132         while(inputFile.read(reinterpret_cast<char *>(&checkBooks), sizeof(Book)))
00133         {
00134             // if a book is issued to someone i.e. available = 0
00135             if(checkBooks.getAvailable() == 0) {
00136                 availFlag = 0; // set flag to false
00137                 break;
00138             }
00139         }
00140
00141         // if flag is flag, at least one book is issued
00142         if(availFlag == 0) {
00143             std::cout << "Not all books are available";
00144             return; // return control, and doesn't allow to remove old book
00145         }
00146
00147         inputFile.seekg(0, std::ios::beg); // seek to beginning of file
00148
00149         int delKey;
00150
00151         system("cls");
00152         std::cout << "Enter Book's key to be deleted: "; // prompt to enter key of book to be deleted
00153         std::cin >> delKey;
00154
00155         std::ofstream outputFile{"temp.dat", std::ios::out | std::ios::binary}; // create a file
'temp.dat'
00156         if(!outputFile) {
00157             std::cerr << "Cannot open file to remove book";
00158             inputFile.close();
00159             exit(EXIT_FAILURE);
00160         }
00161
00162         int newKeys{0}; // assigning new key to each book
00163         bool flag = false;
00164
00165         Book record;
00166         while(inputFile.read(reinterpret_cast<char *>(&record), sizeof(Book))) // read a record from
books' file
00167         {
00168             // is book's key is not equal to delKey
00169             if(record.getKey() != delKey) {
00170                 record.setKey(newKeys++);
00171                 // write the record in 'temp' file

```

```

00172         outputFile.write(reinterpret_cast<const char *>(&record), sizeof(Book));
00173     }
00174     else {
00175         // if a book's key matched don't write it into temp file and set flag to true,
indicating that the book is founded
00176         flag = true;
00177     }
00178 }
00179
00180 // if flag is true i.e. the delKey book is not written into temp file
00181 if(flag == true) {
00182     std::cout << "Book Deleted" << '\n';
00183 }
00184 else {
00185     std::cout << "Book doesn't exist" << '\n';
00186 }
00187
00188 inputFile.close();
00189 outputFile.close();
00190
00191 remove("bookRecords.dat"); // remove books' record file
00192 rename("temp.dat", "bookRecords.dat"); // rename temp file to books' record
00193 }
00194 else {
00195     // if librarian doesn't log in
00196     std::cout << "\nAccess Denied";
00197 }
00198
00199 return;
00200 }
00201
00202 // 4 - check student details
00203 void checkStudentDetails()
00204 {
00205     system("cls");
00206
00207     std::cin.ignore();
00208
00209     // if librarian is logged in
00210     if(librarianLogin()) {
00211         system("cls");
00212
00213         std::cout << "1 - Search with RollNo" << '\t' << "2 - Display all Students" << std::endl <<
std::endl;
00214
00215         const int LOWEST_OPTION{1};
00216         const int HIGHEST_OPTION{2};
00217
00218         int searchChoice;
00219         // validating the input number
00220         do
00221         {
00222             std::cout << "? ";
00223             std::cin >> searchChoice;
00224         }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);
00225
00226         std::ifstream studentFile("studentRecords.dat", std::ios::in | std::ios::binary); // open
student's records file
00227
00228         if(!studentFile) {
00229             std::cerr << "Cannot open file to read books";
00230             exit(EXIT_FAILURE);
00231         }
00232
00233         switch(searchChoice)
00234         {
00235             case 1: // display student with roll no
00236             {
00237                 int rollNo;
00238                 std::cout << "\nEnter roll no: "; // prompt to enter rollNo
00239                 std::cin >> rollNo;
00240
00241                 Student student;
00242
00243                 studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student)); // read a
record from file
00244                 while(studentFile)
00245                 {
00246                     // if rollNo matched
00247                     if(student.getRollNo() == rollNo) {
00248                         std::cout << "\nResult:\n";
00249                         std::cout << student; // display the student's details
00250                         break; // break through the loop
00251                     }
00252
00253                     // if rollNo doesn't match, read next record, until the end of file
00254                     studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student));

```

```

00255     }
00256     }
00257     break;
00258 case 2: // display all students;
00259     {
00260         Student student;
00261
00262         // read a record from the file
00263         studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student));
00264         std::cout << "\nResult:\n";
00265         while(studentFile) // while not 'end of file'
00266         {
00267             std::cout << student.display() << '\n'; // display the record
00268
00269             // read next record
00270             studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student));
00271         }
00272     }
00273     break;
00274 default: // because of validation, control should not reach here
00275     std::cerr << "control should not reach here";
00276     break;
00277 }
00278
00279 studentFile.close(); // close the file
00280 }
00281 // if the librarian is not logged in
00282 else {
00283     std::cout << "\nAccess Denied";
00284 }
00285
00286 return;
00287 }
00288
00289 // 5 - Borrow A Book
00290 void borrowABook()
00291 {
00292     system("cls");
00293     // students are issued books for 7 days and teachers for 14 days
00294     std::cout << "1 - For Student" << '\t' << "2 - For Teacher\n" << std::endl;
00295
00296     const int LOWEST_OPTION{1};
00297     const int HIGHEST_OPTION{2};
00298
00299     int searchChoice;
00300     // validating input
00301     do
00302     {
00303         std::cout << "? ";
00304         std::cin >> searchChoice;
00305     }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);
00306
00307     // if student want to borrow book
00308     if(searchChoice == 1) {
00309         int rollNo;
00310
00311         std::cout << "\n\nEnter your roll no: "; // ask for student's roll no
00312         std::cin >> rollNo;
00313
00314         std::ifstream studentFile("studentRecords.dat", std::ios::in | std::ios::binary); // open
students' records file
00315         if(!studentFile) {
00316             std::cerr << "cannot open student's records";
00317             exit(EXIT_FAILURE);
00318         }
00319
00320         Student student;
00321         bool studentExist = false; // to mark if student doesn't exist
00322         while(studentFile.read(reinterpret_cast<char *>(&student), sizeof(Student))) // read a student
record from file
00323         {
00324             // if roll no matches
00325             if(student.getRollNo() == rollNo) {
00326                 studentExist = true; // student exist
00327
00328                 if(alreadyIssued(rollNo)) {
00329                     std::cout << "The Student already have a book issued";
00330                     studentFile.close();
00331
00332                     return;
00333                 }
00334
00335                 int key;
00336
00337                 std::cout << "Enter book key: "; // prompt to enter book key
00338                 std::cin >> key;
00339

```



```

00340         std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out |
std::ios::binary}; // open books' records file
00341         if(!bookFile) {
00342             std::cerr << "cannot open book's records";
00343             studentFile.close();
00344             exit(EXIT_FAILURE);
00345         }
00346
00347         Book book;
00348         bool bookExist = false; // to mark if book doesn't exist
00349         while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read a book
record
00350         {
00351             // if the key of book matched with input key, and the book is available
00352             if((book.getKey() == key) && (book.getAvailable() == true)) {
00353                 bookExist = true; // book exist
00354
00355                 Date today; // create object of 'Date' initialized with current date
00356                 std::cout << "\nBook Issued till " << today.getDate7(); // issue book for 7 days
00357
00358                 std::fstream issueFile{"issued.txt", std::ios::out | std::ios::app}; // open
issued file
00359                 if(!issueFile) {
00360                     std::cerr << "cannot open issue file";
00361                     // close all opened files
00362                     studentFile.close();
00363                     bookFile.close();
00364                     exit(EXIT_FAILURE);
00365                 }
00366
00367                 // write data of student and issued book is file
00368                 issueFile << rollNo << ' ' << key << ' ' << today.getDate7() << '\n';
00369
00370                 book.setAvailable(false); // set book availability to false;
00371
00372                 // seek back to start of record
00373                 bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
00374                 bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write
book record with availability marked to false
00375
00376                 break;
00377             }
00378         }
00379
00380         // if book doesn't exist
00381         if(bookExist == false) {
00382             std::cout << "\nBook not found";
00383         }
00384
00385         bookFile.close();
00386         break;
00387     }
00388 }
00389
00390 studentFile.close();
00391
00392 // if student doesn't exist
00393 if(studentExist == false) {
00394     std::cout << "\nStudent not found";
00395 }
00396 }
00397 // if teacher want to borrow a book
00398 else if(searchChoice == 2) {
00399     int code;
00400
00401     std::cout << "\n\nEnter your code: "; // prompt to enter teacher's code
00402     std::cin >> code;
00403
00404     std::ifstream teacherFile{"teacherRecords.dat", std::ios::in | std::ios::binary}; // open
teachers' records file
00405     if(!teacherFile) {
00406         std::cerr << "cannot open student's records";
00407         exit(EXIT_FAILURE);
00408     }
00409
00410     MyTeacher teacher;
00411     bool teacherExist = false; // to mark if teacher doesn't exist
00412     while(teacherFile.read(reinterpret_cast<char *>(&teacher), sizeof(MyTeacher))) // read a
record from teacher file
00413     {
00414         // if code matched
00415         if(teacher.getCode() == code) {
00416             teacherExist = true; // teacher exists
00417
00418             if(alreadyIssued(code)) {
00419                 std::cout << "The Teacher already have a book issued";
00420                 teacherFile.close();

```

```

00421
00422         return;
00423     }
00424
00425     int key;
00426
00427     std::cout << "Enter book key: "; // prompt to enter book key
00428     std::cin >> key;
00429
00430     std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out |
std::ios::binary}; // open books' records file
00431     if(!bookFile) {
00432         std::cerr << "cannot open book's records";
00433         teacherFile.close();
00434         exit(EXIT_FAILURE);
00435     }
00436
00437     Book book;
00438     bool bookExist = false; // to mark if book doesn't exist
00439     while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read a record
from book file
00440     {
00441         // if book key matched, and book is available
00442         if((book.getKey() == key) && (book.getAvailable() == true)) {
00443             bookExist = true; // book exists
00444
00445             Date today; // object of 'Date' will be initialized with current date
00446             std::cout << "\nBook Issued till " << today.getDate14(); // issued for 14 days
00447
00448             std::fstream issueFile{"issued.txt", std::ios::out | std::ios::app}; // open
issued file
00449             if(!issueFile) {
00450                 std::cerr << "cannot open issue file";
00451                 // closing opened files
00452                 teacherFile.close();
00453                 bookFile.close();
00454                 exit(EXIT_FAILURE);
00455             }
00456
00457             issueFile << code << ' ' << key << ' ' << today.getDate14() << '\n'; // write issued
book and teacher details in issue file
00458
00459             book.setAvailable(false); // set book's availability to false
00460
00461             // seek back to start of the record
00462             bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
00463             bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write
record of book in file with availability set to false
00464
00465             break;
00466         }
00467     }
00468
00469     // is book doesn't exist
00470     if(bookExist == false) {
00471         std::cout << "\nBook not found";
00472     }
00473
00474     bookFile.close();
00475     break;
00476 }
00477 }
00478
00479 teacherFile.close();
00480
00481 // if teacher doesn't exist
00482 if(teacherExist == false) {
00483     std::cout << "\nTeacher not found";
00484 }
00485 }
00486
00487 return;
00488 }
00489
00490 // 6 - Return A Book
00491 void returnABook()
00492 {
00493     system("cls");
00494     std::cout << "1 - For Student" << '\t' << "2 - For Teacher\n" << std::endl;
00495
00496     const int LOWEST_OPTION{1};
00497     const int HIGHEST_OPTION{2};
00498
00499     int searchChoice;
00500     // input validation
00501     do
00502     {

```

```

00503         std::cout << "? ";
00504         std::cin >> searchChoice;
00505     }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);
00506
00507     if(searchChoice == 1) { // for student
00508         int rollNo;
00509
00510         std::cout << "\n\nEnter your roll no: "; // enter your roll no
00511         std::cin >> rollNo;
00512
00513         std::fstream issuedFile{"issued.txt", std::ios::in}; // open issue file
00514         if(!issuedFile) {
00515             std::cerr << "cannot open issued file";
00516             exit(EXIT_FAILURE);
00517         }
00518
00519         int fRollNo, fKey;
00520         Date fDate;
00521
00522         bool flag = false;
00523         while(issuedFile >> fRollNo >> fKey >> fDate) // read record from issued file
00524         {
00525             // if roll no matched
00526             if(fRollNo == rollNo) {
00527                 std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out |
std::ios::binary}; // open books file
00528                 if(!bookFile) {
00529                     std::cerr << "cannot open books file";
00530                     issuedFile.close();
00531                     exit(EXIT_FAILURE);
00532                 }
00533
00534                 Book book;
00535                 while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read a record
from books' file
00536                 {
00537                     // if key matched
00538                     if(book.getKey() == fKey) {
00539                         book.setAvailable(true); // set availability to true
00540
00541                         // seek back to start of record
00542                         bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
00543                         bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write
record with availability set to true
00544
00545                         Date toDay; // get current date
00546                         if(fDate >= toDay) { // if returned date is greated today i.e. not reached
last date
00547                             std::cout << "Book returned";
00548                         }
00549                         // if returned date is crossed
00550                         else {
00551                             std::cout << "Date exceeded, try to return book on time\nBook returned";
00552                         }
00553
00554                         break;
00555                     }
00556                 }
00557
00558                 bookFile.close();
00559
00560                 flag = true; // mark flag to true, since record in issue file is founded
00561                 break;
00562             }
00563         }
00564
00565         issuedFile.seekg(0, std::ios::beg); // seek to beginning of file
00566
00567         // if no record in issued file found
00568         if(flag != true) {
00569             std::cout << "No book issued";
00570         }
00571         // if a record is founded
00572         else {
00573             std::ofstream file{"temp.txt", std::ios::out}; // create a temp file
00574             if(!file) {
00575                 std::cerr << "cannot open file";
00576                 // close other files
00577                 issuedFile.close();
00578
00579                 exit(EXIT_FAILURE);
00580             }
00581
00582             int fRollNo, fKey;
00583             std::string fDate;
00584             while(issuedFile >> fRollNo >> fKey >> fDate) // read record from issued file
00585             {

```

```

00586         // if roll no doesn't match
00587         if(fRollNo != rollNo) {
00588             file << fRollNo << ' ' << fKey << ' ' << fDate << '\n'; // write in temp file
00589         }
00590     }
00591
00592     file.close();
00593     issuedFile.close();
00594
00595     remove("issued.txt"); // delete issued file
00596     rename("temp.txt", "issued.txt"); // rename temp file to issued
00597 }
00598 }
00599 // for teacher
00600 else if(searchChoice == 2) {
00601     int code;
00602
00603     std::cout << "\n\nEnter your code: "; // prompt to enter code
00604     std::cin >> code;
00605
00606     std::fstream issuedFile{"issued.txt", std::ios::in}; // open issued file
00607     if(!issuedFile) {
00608         std::cerr << "cannot open issued file";
00609         exit(EXIT_FAILURE);
00610     }
00611
00612     int fCode, fKey;
00613     Date fDate;
00614
00615     bool flag = false; // to mark if a book is returned
00616     while(issuedFile >> fCode >> fKey >> fDate) // read a record from the issued file
00617     {
00618         // if code matched
00619         if(fCode == code) {
00620             std::fstream bookFile{"bookRecords.dat", std::ios::in | std::ios::out |
std::ios::binary};
00621             if(!bookFile) {
00622                 std::cerr << "cannot open books file";
00623                 issuedFile.close();
00624                 exit(EXIT_FAILURE);
00625             }
00626
00627             Book book;
00628             while(bookFile.read(reinterpret_cast<char *>(&book), sizeof(Book))) // read record
from books' records file
00629             {
00630                 // if key matched
00631                 if(book.getKey() == fKey) {
00632                     book.setAvailable(true); // set available to true
00633
00634                     // seek to start of record
00635                     bookFile.seekp(static_cast<std::streamoff>(bookFile.tellp()) - sizeof(Book));
00636                     bookFile.write(reinterpret_cast<const char *>(&book), sizeof(Book)); // write
record as availability set to true
00637
00638                     Date toDay; // get current date
00639                     if(fDate >= toDay) { // if return date is greater than today
00640                         std::cout << "Book returned";
00641                     }
00642                     // if returned date is crossed
00643                     else {
00644                         std::cout << "Date exceeded, try to return book on time\nBook returned";
00645                     }
00646
00647                     break;
00648                 }
00649             }
00650
00651             bookFile.close();
00652
00653             flag = true; // mark true to indicate that a book is returned
00654             break;
00655         }
00656     }
00657
00658     issuedFile.seekg(0, std::ios::beg); // seek to beginning to file
00659
00660     // if no book is returned
00661     if(flag != true) {
00662         std::cout << "No book issued";
00663     }
00664     // if a book is returned
00665     else {
00666         std::ofstream file{"temp.txt", std::ios::out};
00667         if(!file) {
00668             std::cerr << "cannot open file";
00669             issuedFile.close();

```

```

00670         exit(EXIT_FAILURE);
00671     }
00672
00673     int fCode, fKey;
00674     std::string fDate;
00675     while(issuedFile >> fCode >> fKey >> fDate) // read record from issued file
00676     {
00677         // if code doesn't match
00678         if(fCode != code) {
00679             // write record in temp file
00680             file << fCode << ' ' << fKey << ' ' << fDate << '\n';
00681         }
00682     }
00683
00684     file.close();
00685     issuedFile.close();
00686
00687     remove("issued.txt"); // remove issued file
00688     rename("temp.txt", "issued.txt"); // rename temp file to issued
00689 }
00690
00691 }
00692
00693 return;
00694 }
00695 // 7 - Renew Date
00696 void renewDate()
00697 {
00698     system("cls");
00699     std::cout << "1 - For Student" << '\t' << "2 - For Teacher\n" << std::endl;
00700
00701     const int LOWEST_OPTION{1};
00702     const int HIGHEST_OPTION{2};
00703
00704     int searchChoice;
00705     // input validation
00706     do
00707     {
00708         std::cout << "? ";
00709         std::cin >> searchChoice;
00710     }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);
00711
00712     // for student
00713     if(searchChoice == 1) {
00714         int rollNo;
00715
00716         std::cout << "\n\nEnter your roll no: "; // prompt to enter roll no
00717         std::cin >> rollNo;
00718
00719         std::fstream issuedFile{"issued.txt", std::ios::in}; // open issue file
00720         if(!issuedFile) {
00721             std::cerr << "cannot open issued file";
00722             exit(EXIT_FAILURE);
00723         }
00724
00725         int fRollNo, fKey;
00726         Date fDate;
00727
00728         bool flag = false;
00729         while(issuedFile >> fRollNo >> fKey >> fDate) // read record from issued file
00730         {
00731             // if roll no matched
00732             if(fRollNo == rollNo) {
00733                 flag = true; // mark flag as true
00734                 break;
00735             }
00736         }
00737
00738         // seek to beginning
00739         issuedFile.seekg(0, std::ios::beg);
00740
00741         // if such student doesn't exist
00742         if(flag != true) {
00743             std::cout << "No book issued";
00744         }
00745         // if student exists
00746         else {
00747             std::ofstream file{"temp.txt", std::ios::out}; // create a temp file
00748             if(!file) {
00749                 std::cerr << "cannot open file";
00750                 issuedFile.close();
00751                 exit(EXIT_FAILURE);
00752             }
00753
00754             int fRollNo, fKey;
00755             std::string fDate;
00756             while(issuedFile >> fRollNo >> fKey >> fDate) // read a record from issued file

```

```

00757         {
00758             // if roll no doesn't match
00759             if(fRollNo != rollNo) {
00760                 file << fRollNo << ' ' << fKey << ' ' << fDate << '\n'; // write record in temp file
00761             }
00762             // if roll no matched
00763             else {
00764                 Date today; // get current date
00765                 file << fRollNo << ' ' << fKey << ' ' << today.getDate7() << '\n'; // write record with
today+7 days
00766             }
00767         }
00768
00769         file.close();
00770         issuedFile.close();
00771
00772         remove("issued.txt"); // delete issued file
00773         rename("temp.txt", "issued.txt"); // rename temp to issued
00774     }
00775 }
00776 // for teacher
00777 // the below code work same as the one for teacher. the only different is teacher renewed returned
date is +14
00778 else if(searchChoice == 2) {
00779     int code;
00780
00781     std::cout << "\n\nEnter your code: ";
00782     std::cin >> code;
00783
00784     std::fstream issuedFile("issued.txt", std::ios::in);
00785     if(!issuedFile) {
00786         std::cerr << "cannot open issued file";
00787         exit(EXIT_FAILURE);
00788     }
00789
00790     int fCode, fKey;
00791     Date fDate;
00792
00793     bool flag = false;
00794     while(issuedFile >> fCode >> fKey >> fDate)
00795     {
00796         if(fCode == code) {
00797             flag = true;
00798             break;
00799         }
00800     }
00801
00802     issuedFile.seekg(0, std::ios::beg);
00803
00804     if(flag != true) {
00805         std::cout << "No book issued";
00806     }
00807     else {
00808         std::ofstream file("temp.txt", std::ios::out);
00809         if(!file) {
00810             std::cerr << "cannot open file";
00811             exit(EXIT_FAILURE);
00812         }
00813
00814         int fCode, fKey;
00815         std::string fDate;
00816         while(issuedFile >> fCode >> fKey >> fDate)
00817         {
00818             if(fCode != code) {
00819                 file << fCode << ' ' << fKey << ' ' << fDate << '\n';
00820             }
00821             else {
00822                 Date today;
00823                 // write with 14 days ahead from today
00824                 file << fCode << ' ' << fKey << ' ' << today.getDate14() << '\n';
00825             }
00826         }
00827
00828         file.close();
00829         issuedFile.close();
00830
00831         remove("issued.txt");
00832         rename("temp.txt", "issued.txt");
00833     }
00834 }
00835
00836 return;
00837 }
00838
00839 // 8 - Search A Book
00840 void searchABook()
00841 {

```

```

00842     system("cls");
00843     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;
00844
00845     const int LOWEST_OPTION{1};
00846     const int HIGHEST_OPTION{5};
00847
00848     int searchChoice;
00849     // input validation
00850     do
00851     {
00852         std::cout << "? ";
00853         std::cin >> searchChoice;
00854     }while(searchChoice < LOWEST_OPTION || searchChoice > HIGHEST_OPTION);
00855
00856     std::fstream booksFile{"bookRecords.dat", std::ios::in | std::ios::binary}; // open books' records
file
00857
00858     if(!booksFile) {
00859         std::cerr << "Cannot open file to read books";
00860         exit(EXIT_FAILURE);
00861     }
00862
00863     switch(searchChoice)
00864     {
00865         case 1: // search with key
00866         {
00867             int key;
00868             std::cout << "\nEnter key: "; // prompt to enter key
00869             std::cin >> key;
00870
00871             Book libraryBook;
00872
00873             // since file is binary, and books are sorted wrt key
00874             // seek directly to book
00875             booksFile.seekg(key * sizeof(Book));
00876             booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read the record
00877
00878             std::cout << "\nResult:\n";
00879             std::cout << libraryBook.getDetails(); // display the record
00880         }
00881         break;
00882         case 2: // search with name
00883         {
00884             std::string name;
00885             std::cout << "\nEnter name: "; // prompt to enter book's name
00886             std::cin.ignore();
00887             getline(std::cin, name); // name may include spaces
00888
00889             Book libraryBook;
00890             booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record
from file
00891             while(booksFile)
00892             {
00893                 // if name matches
00894                 if(libraryBook.getName() == name) {
00895                     std::cout << "\nResult:\n";
00896                     std::cout << libraryBook.getDetails(); // display the record
00897                     break;
00898                 }
00899
00900                 // else read the next record, until the end of file
00901                 booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
00902             }
00903         }
00904         break;
00905         case 3: // search with ISBN
00906         {
00907             std::string isbn;
00908             std::cout << "\nEnter ISBN: "; // prompt to enter isbn of book
00909             std::cin >> isbn;
00910
00911             Book libraryBook;
00912             booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record
from the file
00913             while(booksFile)
00914             {
00915                 // if isbn matches
00916                 if(libraryBook.getIsbn() == isbn) {
00917                     std::cout << "\nResult:\n";
00918                     std::cout << libraryBook.getDetails(); // display the record
00919                     break;
00920                 }
00921
00922                 // else read the next record, until the end of file
00923                 booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
00924             }

```

```

00925     }
00926     break;
00927     case 4: // Filter using genre
00928     {
00929         std::string genre;
00930         std::cout << "\nEnter Genre: "; // prompt to enter genre of book
00931         std::cin >> genre;
00932
00933         Book libraryBook;
00934         booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record
00935     from file
00936         std::cout << "\nResult:\n";
00937         while(booksFile)
00938         {
00939             // if genre matches
00940             if(libraryBook.getGenre() == genre) {
00941                 std::cout << libraryBook.getDetails() << '\n'; // display the record
00942                 // no break since we want to display all books with the genre
00943             }
00944
00945             // read next record until the end of file
00946             booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
00947         }
00948         break;
00949     case 5: // display full list
00950     {
00951         Book libraryBook;
00952
00953         booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book)); // read a record
00954         std::cout << "\nResult:\n";
00955         while(booksFile)
00956         {
00957             std::cout << libraryBook.getDetails() << std::endl; // display it without any condition,
00958             since all record are to be displayed
00959
00960             // read next record, until the end of file
00961             booksFile.read(reinterpret_cast<char *>(&libraryBook), sizeof(Book));
00962         }
00963         break;
00964     default: // control shouldn't reach here
00965         std::cerr << "control should not reach here";
00966         break;
00967     }
00968
00969     booksFile.close();
00970     return;
00971 }
00972
00973 #endif

```

## 6.13 README.md File Reference

## 6.14 test.cpp File Reference

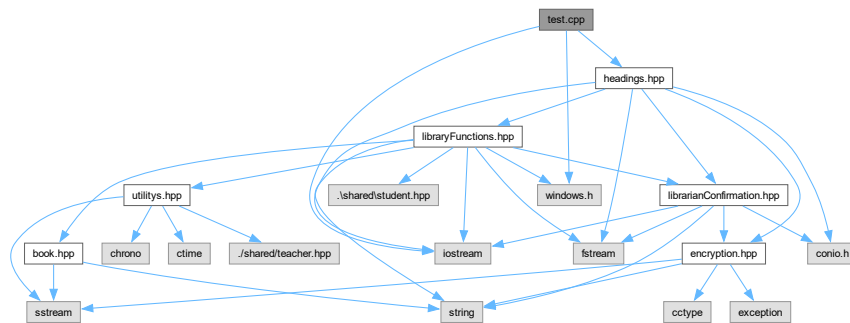
```

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

```



Include dependency graph for test.cpp:



## Functions

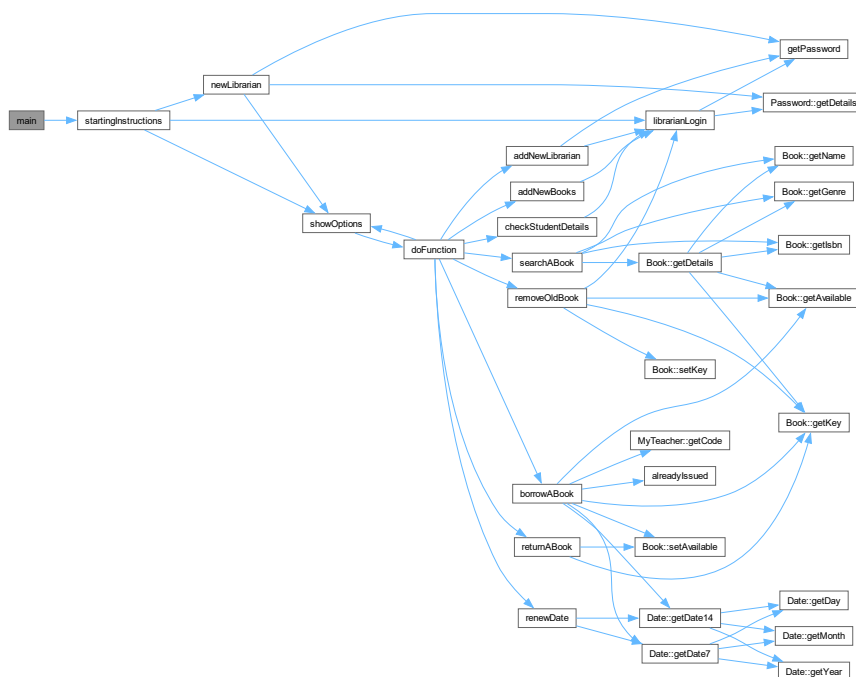
- int `main` ()

### 6.14.1 Function Documentation

#### 6.14.1.1 `main()`

```
int main ( )
```

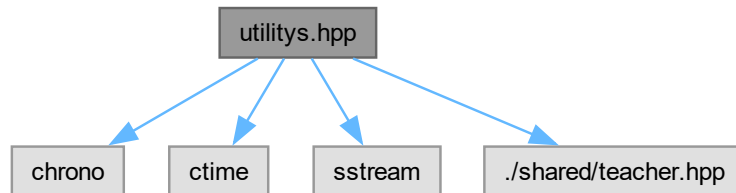
Here is the call graph for this function:



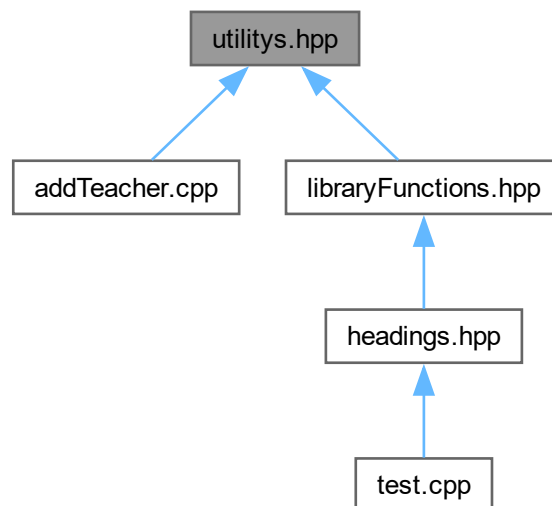
## 6.15 utilitys.hpp File Reference

```
#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

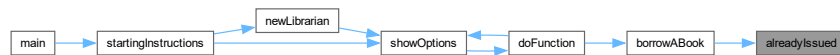
- `std::istream & operator>>` (`std::istream &input`, `Date &obj`)
- `std::ostream & operator<<` (`std::ostream &output`, `const MyTeacher &teacher`)
- `std::istream & operator>>` (`std::istream &input`, `MyTeacher &teacher`)
- `bool alreadyIssued` (`int num`)

## 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.](#)

```
00001 // 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
00002
00003 #ifndef UTILITYS_HPP
00004 #define UTILITYS_HPP
00005
00006 #include <chrono>
00007 #include <ctime>
00008 #include <sstream>
00009 #include "../shared/teacher.hpp"
00010
00011 // When created an object of 'Date', the object consist of current date, month, and year.
00012 // It can also return currentDay+7days and currentDay+14days
00013 class Date
00014 {
00015     friend std::istream& operator>>(std::istream&, Date&); // stream extraction operator
00016 public:
00017     Date()
00018     {
00019         auto currentTime = std::chrono::system_clock::now(); // get the current time point
00020
00021         std::time_t currentTime_t = std::chrono::system_clock::to_time_t(currentTime); // convert the
    time point to a time_t object
00022
00023         std::tm* localTime = std::localtime(&currentTime_t); // convert the time_t object to a tm
    structure
00024
00025         // Extract individual components of the date and time
00026         year = localTime->tm_year + 1900; // Years since 1900
00027         month = localTime->tm_mon + 1; // Months start from 0
00028         day = localTime->tm_mday;
00029     }
00030
00031     // return value of 'year'
00032     int getYear() const
00033     {
00034         return year;
00035     }
00036     // return value of 'month'
00037     int getMonth() const
00038     {
00039         return month;
00040     }
00041     // return value of 'day'
00042     int getDay() const
00043     {
00044         return day;
00045     }
00046
00047     // return current date as a string
00048     std::string getDate() const
00049     {
00050         std::ostringstream output;
00051         output << getDay() << '-' << getMonth() << '-' << getYear(); // adding '-' between day, month, and
    year
00052
00053         return output.str(); // return as a string
00054     }
00055
00056     // return current date + 7 as a string
00057     std::string getDate7() const
00058     {
00059         // make copies of day, month, and year
00060         int dd = getDay();
00061         int mm = getMonth();
00062         int yy = getYear();
00063
00064         // check for leap year
00065         bool leap = false;
00066         if(((yy % 100 == 0) && (yy % 400 == 0)) || ((yy % 100 != 0) && (yy % 4 == 0))) {
00067             leap = true;
00068         }
00069
00070         dd += 7; // add 7 days to the date
00071
00072         // Adjust the date if necessary
00073         if((mm == 4 || mm == 6 || mm == 9 || mm == 11) && dd > 30) {
00074             dd -= 30;
00075             ++mm;
00076         }
00077         else if((mm == 1 || mm == 3 || mm == 5 || mm == 7 || mm == 8 || mm == 10) && dd > 31) {
00078             dd -= 31;
```

```

00079         ++mm;
00080     }
00081     else if(mm == 12 && dd > 31) {
00082         dd -= 31;
00083         ++mm;
00084         ++yy;
00085     }
00086     else if(mm == 2 && leap && dd > 29) {
00087         dd -= 29;
00088         ++mm;
00089     }
00090     else if(mm == 2 && !leap && dd > 28) {
00091         dd -= 28;
00092         ++mm;
00093     }
00094
00095     // Adjust the year if the month is now January of the next year
00096     if (mm == 1) {
00097         ++yy;
00098     }
00099
00100     std::ostringstream output;
00101     output << dd << '-' << mm << '-' << yy; // join day, month, and year
00102
00103     return output.str(); // return as a string
00104 }
00105
00106 // return current day + 14 days
00107 std::string getDate14() const
00108 {
00109     // getting copies of day, month, and year
00110     int dd = getDay();
00111     int mm = getMonth();
00112     int yy = getYear();
00113
00114     // check for leap year
00115     bool leap = false;
00116     if(((yy % 100 == 0) && (yy % 400 == 0)) || ((yy % 100 != 0) && (yy % 4 == 0))) {
00117         leap = true;
00118     }
00119
00120     dd += 14; // add 14 days to the date
00121
00122     // Adjust the date if necessary
00123     if((mm == 4 || mm == 6 || mm == 9 || mm == 11) && dd > 30) {
00124         dd -= 30;
00125         ++mm;
00126     }
00127     else if((mm == 1 || mm == 3 || mm == 5 || mm == 7 || mm == 8 || mm == 10) && dd > 31) {
00128         dd -= 31;
00129         ++mm;
00130     }
00131     else if(mm == 12 && dd > 31) {
00132         dd -= 31;
00133         ++mm;
00134         ++yy;
00135     }
00136     else if(mm == 2 && leap && dd > 29) {
00137         dd -= 29;
00138         ++mm;
00139     }
00140     else if(mm == 2 && !leap && dd > 28) {
00141         dd -= 28;
00142         ++mm;
00143     }
00144
00145     // Adjust the year if the month is now January of the next year
00146     if (mm == 1) {
00147         ++yy;
00148     }
00149
00150     std::ostringstream output;
00151     output << dd << '-' << mm << '-' << yy; // join all three of them
00152
00153     return output.str(); // return as a string
00154 }
00155
00156 // operator '>=' to check if a date is greater than or equal of another date
00157 bool operator>=(const Date& other) const
00158 {
00159     // If year is greater, the first date is surely greater
00160     if(year > other.year) {
00161         return true;
00162     }
00163     else if(year < other.year) {
00164         return false;
00165     }

```

```

00166
00167     // If years are equal, compare months
00168     if(month > other.month) {
00169         return true;
00170     }
00171     else if(month < other.month) {
00172         return false;
00173     }
00174
00175     // If months are equal, compare days
00176     return day >= other.day;
00177 }
00178 private:
00179     int year;
00180     int month;
00181     int day;
00182 };
00183
00184 // stream extraction operator for 'Date'
00185 std::istream& operator>(std::istream& input, Date& obj)
00186 {
00187     std::string dateString;
00188
00189     // taking input as a string
00190     if (std::getline(input, dateString)) {
00191         std::istringstream dateStream(dateString); // make stream of string
00192         char delimiter;
00193
00194         // extract individual elements from the stream
00195         dateStream >> obj.day >> delimiter >> obj.month >> delimiter >> obj.year;
00196     }
00197
00198     return input; // for cin >> a >> b >> c
00199 }
00200
00201
00202 // This class is inherited from class 'Teacher'. It only add 'code' on its base-class
00203 class MyTeacher : public Teacher
00204 {
00205     friend std::ostream& operator<(std::ostream&, const MyTeacher&); // overloaded stream extraction
operator
00206     friend std::istream& operator>(std::istream&, MyTeacher&); // overloaded stream insertion operator
00207 public:
00208     MyTeacher() = default; // default constructor
00209     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
00210         : Teacher(first, last, age, card, gen, phone, department, rank) {
00211         setCode(code); // set code
00212     }
00213
00214     void setCode(int v)
00215     {
00216         code = v; // save in 'code'
00217         return;
00218     }
00219     int getCode() const
00220     {
00221         return code; // return 'code'
00222     }
00223
00224     // overriding 'display' function of class 'Teacher'
00225     std::string display() const
00226     {
00227         std::ostringstream output; // creating object of class 'ostringstream'
00228         output << Teacher::display() << ' ' << getCode(); // concatenate all data members with spaces in
between
00229
00230         return output.str(); // return it as string
00231     }
00232 private:
00233     int code;
00234 };
00235
00236 // stream insertion operator for MyTeacher
00237 std::ostream& operator<(std::ostream& output, const MyTeacher& teacher)
00238 {
00239     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
00240
00241     return output; // enable cout << a << b << c
00242 }
00243 // stream extraction operator for MyTeacher
00244 std::istream& operator>(std::istream& input, MyTeacher& teacher)

```

```
00245 {
00246     std::string first, last, idCard, phone, department, rank;
00247     int age, gender, code;
00248
00249     input » first » last » age » idCard » gender » code » phone; // input all data members
00250     getline(input, department);
00251     input » rank;
00252
00253     // calling set functions to ensure our c-type strings proper handling
00254     teacher.setFirstName(first);
00255     teacher.setLastName(last);
00256     teacher.setAge(age);
00257     teacher.setIdCard(idCard);
00258     teacher.setGender(gender);
00259     teacher.setCode(code);
00260     teacher.setPhoneNumber(phone);
00261     teacher.setDepartment(department);
00262     teacher.setRank(rank);
00263
00264     return input; // enables cin » a » b » c
00265 }
00266
00267 bool alreadyIssued(int num)
00268 {
00269     std::ifstream issuedFile("issued.txt", std::ios::in);
00270     if(!issuedFile) {
00271         std::cerr « "cannot open issued file";
00272         exit(EXIT_FAILURE);
00273     }
00274
00275     int unique, key;
00276     std::string date;
00277
00278     while(issuedFile » unique » key » date)
00279     {
00280         if(unique == num) {
00281             issuedFile.close();
00282             return true;
00283         }
00284     }
00285
00286     issuedFile.close();
00287     return false;
00288 }
00289
00290 #endif
```

