CS301-01 Data Structures and Algorithms
Professor Zahra Derakhshandeh
Department: Computer Science
California State University-East Bay

Group # 3 members:
Jose Andino net ID # ZM6694
Hao Lam net ID # EF3628
Ming-Him Yeung net ID# AZ7537

Group Project: Library Management System

Required parameters:

1. All program files (.h && .cpp) with line numbers
2. Brief report about code (refer to exact line numbers if needed), what used in certain parts of the algorithm
3. Time complexity of our algorithm, explaining why our code has the exact time complexity specified in our report, and counting the number of basic operations (comparing and arithmetic operations)

1. Here will be inserted all code with numbered lines in following order:
Book.h, book.cpp, library.h, library.cpp, renter.h, renter.cpp & LibraryManagementSystem.cpp

```
1. ////////////////////////////////////////////////////////////////////////////
                                        2. BOOK HEADER CLASS
3. ////////////////////////////////////////////////////////////////////////////
#pragma once
#include <string>
#include <vector>
       using namespace std;
class book
{
public:
       book();

       void setBook(string name, string code, vector<string> info, bool status);
       void setRent(bool status);
       string getcode();
       string getname();
       vector<string> getinfo();
       bool isRent();
       ~book();
private:
       string book_name;
       string book_code;
       vector<string> book_info;
       bool is_Rent;
4. };
5. ////////////////////////////////////////////////////////////////////////////
                                    6. //////////BOOK CPP CLASS
7. ////////////////////////////////////////////////////////////////////////////
8. //#include "pch.h"
9. #include "book.h"
10. #include <string>
11. #include <vector>
12. using namespace std;
13. book::book()
14. {
15. }
16.
17. void book::setBook(string name, string code, vector<string> info, bool status)
```

```
18. {
19.  book_name = name;
20.  book_code = code;
21.  for (int i = 0; i < info.size(); i++)
22.  {
23.         //pushing line by line
24.         book_info.push_back(info.at(i));
25.  }
26.  is_Rent = status;
27. }
28. void book::setRent(bool status)
29. {
30.  is_Rent = status;
31. }
32. string book::getcode()
33. {
34.  return book_code;
35. }
36. string book::getname()
37. {
38.  return book_name;
39. }
40. vector<string> book::getinfo()
41. {
42.  return book_info;
43. }
44. bool book::isRent()
45. {
46.  return is_Rent;
47. }
48. book::~book()
49. {
50. }
51. ////////////////////////////////////////////////////////////////////////////////
                                52. //////////LIBRARY HEADER CLASS
53. ////////////////////////////////////////////////////////////////////////////////
54. #pragma once
55. #include "book.h"
56. #include "renter.h"
57. #include <string>
58.  using namespace std;
59. struct NodeType
60. {
61.  book info;
62.  NodeType* next;
63. };
64. struct NodeType2
65. {
66.  renter info;
67.  NodeType2* next;
68. };
69.
70. class library
71. {
72. public:
73.  library();
74.  ~library();
75.  // For book:
76.
77.  bool isAvalable_4rent(string book_code); // check if book is rent or not
78.  bool isHere(string book_name); //search book, return true/false indicate book exist in library
    or not
79.  void markRentStatus(string book_name, bool isRent); //update book status base on "isRent"
```

```
void insert_Book(book item); // base on category_code + name to order books
void delete_book(string id);
string returnBookName(string id);
// For renter:
bool find_renter(string id, renter &item);
//+ return true/false for renter exist in the system or not
//+ pass by reference renter's info if found.
void add_renter(renter item);
void delete_renter(string id); // delete renter base from id , since name could be the same

// For book_search
void printBook_info(string name);
void printBook_byCatg(string catg); //Print all book follow by the category

private:
 NodeType *bookList; // pointer point the head list of the book
 NodeType2* renterList; //pointer point to the head of renters list
};
////////////////////////////////////////////////////////////////////////////////
                                    //////////LIBRARY CPP CLASS
////////////////////////////////////////////////////////////////////////////////
//#include "pch.h"
#include "library.h"
#include <string>
#include <iostream>
        using namespace std;
library::library()
{
        bookList = NULL;
        renterList = NULL;
}


library::~library()
{
}
bool library::isAvalable_4rent(string book_code)
{
        if (bookList == NULL)
        {
                return false;
        }
        NodeType*temp = bookList;

        while (temp != NULL)
        {
                if (book_code == temp->info.getcode())
                {
                        return true;
                }
                else
                {
                        temp = temp->next;
                }
        }
        return false;
}
string library::returnBookName(string id)
{
        if (bookList == NULL)
                return "";
        else
        {
```

```
143.                NodeType* temp = bookList;
144.                while (temp != NULL)
145.                {
146.                       if (temp->info.getcode() == id)
147.                       {
148.                              return temp->info.getname();
149.                       }
150.                       temp = temp->next;
151.                }
152.         }
153. }
154. bool library::isHere(string book_name)
155. {
156.         //check if book exist in the library or not
157.         if (bookList == NULL)
158.                return false;
159.         NodeType* location = bookList;
160.         string currBook;
161.         while (location != NULL)
162.         {
163.                currBook = location->info.getname();
164.                if (currBook == book_name)
165.                {
166.                       return true;
167.                }
168.                else
169.                {
170.                       location = location->next;
171.                }
172.         }
173.         return false;
174. }
175.
176. void library::insert_Book(book item) // base on category_code + name to order books
177. {
178.         //Case 0: First book in the list
179.         NodeType* newBook;
180.         newBook = new NodeType;
181.         newBook->info = item;
182.         newBook->next = NULL;
183.         if (bookList == NULL)
184.         {
185.                bookList = newBook;
186.         }
187.         else
188.         {
189.                NodeType * curr = bookList;
190.                string currentBook = curr->info.getcode();
191.                int compare = item.getcode().compare(currentBook);
192.
193.                //Case 1: insert book as a head
194.                if (compare < 0)
195.                {
196.                       newBook->next = curr;
197.                       bookList == newBook;
198.                }
199.
200.                //Case 2: insert in the middle
201.                else
202.                {
203.                       NodeType *prev = bookList;
204.                       curr = curr->next;
205.                       while (curr != NULL)
```

```
                        {
                                compare = item.getcode().compare(currentBook);
                                currentBook = curr->info.getcode();
                                compare = item.getcode().compare(currentBook);
                                if (compare == 0) //two book are same, do nothing
                                {
                                        return;
                                }
                                else if (compare < 0)
                                {
                                        prev->next = newBook;
                                        newBook->next = curr;
                                        return;
                                }
                                prev = curr;
                                curr = curr->next;
                        }

                        //Case 3: Insert last
                                //after loop, prev is last node
                        prev->next = newBook;
                }

        }
}
void library::delete_book(string id)
{
        if (bookList == NULL)
                return;
        // 1st book is the one
        if (bookList->info.getcode() == id)
        {
                NodeType* temp = bookList;
                bookList = bookList->next;
                delete temp;
        }
        else
        {
                NodeType* prev = bookList;
                NodeType* location = bookList->next;
                string currBook;

                while (location != NULL)
                {
                        currBook = location->info.getcode();
                        if (currBook == id)
                        {
                                prev->next = location->next;
                                delete location;
                                return;
                        }
                        else
                        {
                                prev = location;
                                location = location->next;
                        }
                }
        }
}
void library::printBook_info(string name)
{
        if (bookList == NULL)
                return;
```

```
269.        if (name == "")
270.               return;
271.        NodeType* temp = bookList;
272.        vector<string> bookinfo;
273.        while (temp != NULL)
274.        {
275.               if (temp->info.getname() == name)
276.               {
277.                      cout << "-------- " << name << " ---------" << endl;
278.                      cout << temp->info.getcode() << endl;
279.                      bookinfo = temp->info.getinfo();
280.                      for (int i = 0; i < bookinfo.size(); i++)
281.                      {
282.                             cout << bookinfo.at(i) << endl;
283.                      }
284.                      cout << "------------------------------------------------------------" <<
    endl << endl;
285.                      return;
286.               }
287.               else
288.               {
289.                      temp = temp->next;
290.               }
291.        }
292.        cout << "\n>>>Cannot Dectect The Book! " << endl;
293. }
294.
295. void library::markRentStatus(string book_name, bool isRent)
296. { // mark the book as rented or not base on isRent
297.        if (bookList == NULL)
298.               return;
299.        NodeType* location = bookList;
300.        string currBook;
301.        while (location != NULL)
302.        {
303.               currBook = location->info.getname();
304.               if (currBook == book_name)
305.               {
306.                      location->info.setRent(isRent);
307.               }
308.               else
309.               {
310.                      location = location->next;
311.               }
312.        }
313. }
314. void library::add_renter(renter item)
315. {
316.        NodeType2* newRenter = new NodeType2;
317.        newRenter->info = item;
318.        newRenter->next = NULL;
319.        if (renterList == NULL)
320.        {
321.               renterList = newRenter;
322.        }
323.        else
324.        {
325.               NodeType2* curr = renterList;
326.               string currRenter = curr->info.get_netID();
327.               int compare = item.get_renterName().compare(currRenter);
328.               if (compare < 0)
329.               {
330.                      newRenter->next = curr;
```

```
331.                        renterList == newRenter;
332.                }
333.
334.                //Case 2: insert in the middle
335.                else
336.                {
337.                        NodeType2 *prev = renterList;
338.                        curr = curr->next;
339.                        while (curr != NULL)
340.                        {
341.                                currRenter = curr->info.get_netID();
342.                                compare = item.get_netID().compare(currRenter);
343.                                if (compare == 0)
344.                                {
345.                                        return;
346.                                }
347.                                else if (compare < 0)
348.                                {
349.                                        prev->next = newRenter;
350.                                        newRenter->next = curr;
351.                                        return;
352.                                }
353.                                prev = curr;
354.                                curr = curr->next;
355.                        }
356.
357.                        //Case 3: Insert last
358.                                //after loop, prev is last node
359.                        prev->next = newRenter;
360.                }
361.        }
362.
363. }
364. void library::delete_renter(string id)
365. {// delete renter base from id , since name could be the same
366.        if (renterList == NULL)
367.                return;
368.
369.        // 1st renter is the one
370.        if (renterList->info.get_netID() == id)
371.        {
372.                NodeType2* temp = renterList;
373.                renterList = renterList->next;
374.                delete temp;
375.        }
376.        else
377.        {
378.                NodeType2* prev = renterList;
379.                NodeType2* location = renterList->next;
380.                string currRenter;
381.
382.                while (location != NULL)
383.                {
384.                        currRenter = location->info.get_netID();
385.                        if (currRenter == id)
386.                        {
387.                                prev->next = location->next;
388.                                delete location;
389.                                return;
390.                        }
391.                        else
392.                        {
393.                                prev = location;
```

```
                                location = location->next;

                        }
                }
        }
}
bool library::find_renter(string id, renter &item)
{
        //+ return true/false for renter exist in the system or not
        //+ pass by reference renter's info if found.
        if (renterList == NULL)
        {
                return false;
        }
        else
        {
                NodeType2* temp = renterList;
                while (temp != NULL)
                {

                        if (temp->info.get_netID() == id)
                        {
                                item = temp->info;
                                return true;
                        }
                        temp = temp->next;
                }
        }
        return false;

}
void library::printBook_byCatg(string catg) //Print all book follow by the category
{
        if (bookList == NULL)
        {
                return;
        }
        else
        {

                NodeType *temp = bookList;
                string code;
                while (temp != NULL)
                {
                        code = temp->info.getcode().substr(0, 4);
                        if (code == catg)
                        {
                                //print the book_info
                                printBook_info(temp->info.getname());
                        }
                        temp = temp->next;

                }
        }
}
////////////////////////////////////////////////////////////////////////////////
                                450.//////////RENTER HEADER CLASS
////////////////////////////////////////////////////////////////////////////////
#pragma once
#include<vector>
#include <string>

        using namespace std;
```

```
457. class renter
458. {
459. public:
460.         renter(); // constructor should initialize is date_return, isLate, late_fee
461.         ~renter();
462.         void setRenterInfo(string name, string id, int dayRent, string dateRent, vector<string>
        bookrent); //initialize
463.         string get_renterName();
464.         string get_netID();
465.         string get_rentDate();
466.         vector<string> get_bookRented(); // return the rented book list
467.         int get_day4rent(); // how many day renter will rent the  book
468.         bool is_Late(int numDayReturn, int &late); // checking late return, calculate late_date+
        lateFee
469.         float get_lateFee(); //fee need to pay for late return
470.
471.
472. private:
473.         string renter_name;
474.         string date_rent;    //the date rent the book; should in form xx/xx/xx can easy to
475.                                                 //keep track for late fee
476.         int day_rent;        // will rent the book for how many day
477.         bool isLate;         //late return flag
478.         float late_fee;             //late fee
479.         string netID;
480.         vector<string> bookRented; //name of rented books
481. };
482. ////////////////////////////////////////////////////////////////////////////////
                                    483. ///////////RENTER CPP CLASS
484. ////////////////////////////////////////////////////////////////////////////////
485. //#include "pch.h"
486. #include "renter.h"
487. #include <string>
488. #include <vector>
489. #include <iostream>
490. using namespace std;
491. renter::renter()// constructor should initialize is date_return, isLate, late_fee
492. {
493. renter_name = "";
494. date_rent = "";
495. day_rent = 0;
496. isLate = false;
497. late_fee = 0;
498. netID = "";

499. }

500. renter::~renter()
501. {
502. }

503. void renter::setRenterInfo(string name, string id, int dayRent, string dateRent, vector<string>
        bookrent) //initialize
504. {
505. renter_name = name;
506. date_rent = dateRent;
507. day_rent = dayRent;
508. isLate = false;
509. late_fee = 0;
510. netID = id;
511. bookRented = bookrent;
512. }
513. bool renter::is_Late(int numDayReturn, int&late) // checking late return
```

```
514. {
515. if (numDayReturn > day_rent)
516. {
517. late = numDayReturn - day_rent;
518. late_fee = late * 1.50;
519. return true;
520. }
521. return false;
522. }
523. vector<string> renter::get_bookRented()
524. {
525. return bookRented;
526. }
527. string renter::get_renterName()
528. {
529. return renter_name;
530. }
531. string renter::get_netID()
532. {
533. return netID;
534. }
535. string renter::get_rentDate()
536. {
537. return date_rent;
538. }
539. int renter::get_day4rent()
540. {
541. return day_rent;
542. }
543. float renter::get_lateFee()
544. {
545. return late_fee;
546. }
547. ////////////////////////////////////////////////////////////////////////////////
                          548. ///////////LIBRARY MANAGEMENT DRIVER  CLASS
549. ////////////////////////////////////////////////////////////////////////////////
550.
551. #include <time.h>
552. #include <ctime>
553. //#include "pch.h"
554. #include <iostream>
555. #include "book.h"
556. #include "library.h"
557. #include "renter.h"
558. #include <sstream>
559. #include <fstream>
560. #include <string>
561. #include <vector>
562. using namespace std;
563. void book_searchbyCatg(library &list);
564. void book_searchbyBook(library &list);
565. void book_search(library &list);
566. void book_addnew(library &list);
567. void book_takeout(library &list, bool for_rent, string id, string renter_name);
568. void book_checkin(library &list, string id);
569. void adminService(library &list);
570. void studentService(library &list, string id);
571. string todayDate;
572. int main()
573. {
574. // read .txt file → insert books to library
575. library list;
576. string info, book_name, book_code;
```

```
577. ifstream infile;
578. infile.open("booklist.txt");
579. cout << "-->OPEN FILE FILE COMPLETED!" << endl;
580. cout << "-->Start reading books..." << endl;
581. while (!infile.eof())
582. {
583. //read from file
584. vector <string> book_info;
585. getline(infile, book_name); //read name
586. getline(infile, book_code); // read code
587. do                                                  // read info
588. {
589. getline(infile, info);
590. book_info.push_back(info);
591. } while (info != "*****");
592. book_info.pop_back(); // pop the line "*****"
593. //set book

594. book input;
595. input.setBook(book_name, book_code, book_info, false);
596. cout << "--> Inserting books..." << endl;
597. //add book to library
598. list.insert_Book(input);
599. cout << input.getcode() << "-->INSERTED" << endl;
600. }
601. infile.close();//close file
602. cout << "-->IN-PUT FILE COMPLETED!" << endl << endl;
603. //Add sample renter.
604. renter p2;
605. vector<string>bookrent;
606. string book = "HIST-1111";
607. bookrent.push_back(book);
608. p2.setRenterInfo("Hao Lam", "ef3628", 5, "05/01/2019", bookrent);
609. list.add_renter(p2);

610. // library services:
611. // +Admin: add, delete, search bool
612. // +Student:
613. // + book renting/ book checkout
614. // + book info/ book search
615. // + book return/ book check-in
616. // +library info: cout min/max day for rent( maybe have this value at 14 day period),
617. //late fees( $1.50 a day/per day until book is returned),
618. //max book for rent per renter (this I am assuming would be max times a book can be renewed and
    maybe we can set this at 3 times)

619. cout << endl;
620. cout << "/////// LIBRARY SYSTEM ///////" << endl << endl;
621. string id;
622. do
623. {

624. cout << "Enter your ID: " << endl;
625. //ID rule: admin-> first 2 char is "ad", else is student
626. cout << "-> ";
627. getline(cin, id);
628. if (id[0] == 'a'&& id[1] == 'd')
629. {
630. cout << "\n-->ADMIN Log-In---" << endl << endl;
631. adminService(list);
632. }
633. else
```

```
634. {
635. cout << "\n-->STUDENT Log-In---" << endl << endl;
636. studentService(list, id);
637. }

638. } while (id != "0");
639. cout << "____________________EXIT LIBRARY SYSTEM__________________" << endl;
640. }

641. void adminService(library &list)
642. {
643. cout << "-------------ADMIN SERVICES-------------" << endl << endl;
644. int opt;
645. do
646. {
647. do
648. {
649. cout << "Options:\n";
650. cout << "1. Add Book\n";
651. cout << "2. Delete Book\n";
652. cout << "3. Book Search:\n";
653. cout << "Please choose your option: " << endl;
654. cout << "-> ";
655. cin >> opt;
656. } while (opt < 0 || opt>3);
657. if (opt == 1)
658. {
659. book_addnew(list);
660. }
661. else if (opt == 2)
662. {
663. book_takeout(list, false, "", "");
664. }
665. else
666. {
667. book_search(list);
668. }
669. cout << "Enter 0 for exit, 1 for choose other services: " << endl;
670. cout << "-> ";
671. cin >> opt;
672. } while (opt == 1);

673. }
674. void studentService(library &list, string id)
675. {
676. cout << "-------------STUDENT SERVICES-------------" << endl << endl;
677. cout << "Enter today date (mm/dd/yy): " << endl;
678. cout << "-> ";
679. getline(cin, todayDate);


680. string name;
681. cout << "Please enter your name: " << endl;
682. cout << "-> ";
683. getline(cin, name);
684. int opt;
685. do
686. {
687. do
688. {
689. cout << "\nOptions:\n";
690. cout << "1. Rent Books\n";
691. cout << "2. Check-in books\n";
```

```
692. cout << "3. Book Search:\n";
693. cout << "Please choose your option: " << endl;
694. cout << "-> ";
695. cin >> opt;
696. } while (opt < 0 || opt>3);
697. if (opt == 1)
698. {
699. book_takeout(list, true, id, name);
700. }
701. else if (opt == 2)
702. {
703. book_checkin(list, id);
704. }
705. else
706. {
707. book_search(list);
708. }
709. cout << "Enter 0 for exit, 1 for choose other services: " << endl;
710. cout << "-> ";
711. cin >> opt;
712. cout << "------------------------------------------------------------" << endl;
713. } while (opt == 1);


714. }
715. void book_takeout(library &list, bool for_rent, string id, string renter_name)
716. {
717. // 2 cases of book taking out:
718. /*
719. + Admin delete book -> permenantly delete book out of list
720. + Student rent book -> check status-> fix renting status if needed ;
721. */

722. int num = 0;
723. do
724. {
725. cout << "\nHow many book: " << endl;
726. cout << "-> ";
727. cin >> num;
728. } while (num < 0);
729. if (num == 0)
730. {
731. cout << "0 book is required" << endl;
732. }
733. else
734. {
735. vector <string> bookcode;
736. string idbook;
737. cin.ignore();
738. cout << "Enter book ID: " << endl;
739. for (int i = 0; i < num; i++)
740. {
741. cout << "-> " << i + 1 << " : ";
742. getline(cin, idbook);
743. bookcode.push_back(idbook);
744. }
745. cout << "\n>>>Searching in the system....\n ";
746. //CASE 1: Admin-> delete books
747. if (!for_rent)
748. {
749. for (int i = 0; i < num; i++)
750. {
751. cout << "\n>>>Find and Deleting..." << bookcode.at(i) << endl;
```

```
752. list.delete_book(bookcode.at(i));
753. }
754. cout << "\nCOMPLETE!" << endl;
755. }
756. //CASE 2: Student-> create renter, book mark "rented"
757. else
758. {
759. short dayrent;
760. string dateRent;
761. do
762. {
763. cout << "You will rent for how many days? (No longer than 30 days)" << endl;
764. cout << "-> ";
765. cin >> dayrent;
766. } while (dayrent > 30);

767. renter p1;
768. bool check;
769. string name;
770. for (int i = 0; i < num; i++)
771. {
772. //check book is exist in library or not
773. name = list.returnBookName(bookcode.at(i));
774. check = list.isHere(name);
775. if (!check)
776. {
777. cout << bookcode.at(i) << " IS NOT EXIST!!" << endl;
778. bookcode.erase(bookcode.begin() + i);
779. }
780. //check book's availibility
781. else
782. {
783. check = list.isAvalable_4rent(bookcode.at(i));
784. if (check == true)
785. {
786. list.markRentStatus(bookcode.at(i), true);
787. cout << bookcode.at(i) << "-->RENTED";
788. }
789. else
790. cout << bookcode.at(i) << " currently is not available!" << endl;
791. }
792. }
793. //Finalize renter's info
794. p1.setRenterInfo(renter_name, id, dayrent, todayDate, bookcode);
795. list.add_renter(p1);
796. cout << "\n\n>>>Renter Confirmation:" << endl;
797. cout << renter_name << "\nnetID: " << id << endl;
798. cout << "Number of day for rent: " << dayrent << endl;
799. cout << "Book Check-out: " << bookcode.size() << endl;
800. for (int i = 0; i < bookcode.size(); i++)
801. {
802. cout << bookcode.at(i) << endl;
803. }

804. }

805. }
806. cout << "------------------------------------------------------------" << endl;

807. }
808. void book_addnew(library &list)
809. {
810. //Admin add new books
```

```
811. cout << "\n--------------------ADD NEW BOOK------------------------------" << endl;
812. int num = 0;
813. do
814. {
815. cout << "\nHow many books: " << endl;
816. cout << "-> ";
817. cin >> num;
818. } while (num < 0);
819. cin.ignore();
820. if (num == 0)
821. {
822. cout << "0 book is required" << endl;
823. }
824. else
825. {
826. vector <string> bookname;
827. string name;
828. string code;
829. cout << "Enter book name: " << endl;
830. for (int i = 0; i < num; i++)
831. {
832. cout << "-> " << i + 1 << " : ";
833. getline(cin, name);
834. bookname.push_back(name);
835. }

836. for (int i = 0; i < num; i++)
837. {
838. cout << "\n>>>Adding..." << bookname.at(i) << endl;
839. cout << "Please add book code: " << endl;
840. cout << "->";
841. getline(cin, code);
842. cout << "Please give the book info (end with '*****'): " << endl;
843. cout << "-> ";
844. vector<string> book_info;
845. string info;
846. do
847. {
848. getline(cin, info);
849. book_info.push_back(info);
850. } while (info != "*****");
851. cout << "\n>>>Book info is added!" << endl;
852. // creat new book
853. book newbook;
854. newbook.setBook(bookname.at(i), code, book_info, false);
855. list.insert_Book(newbook);
856. }
857. cout << "\nCOMPLETE!" << endl;
858. }
859. cout << "------------------------------------------------------------" << endl;
860. }
861. void book_checkin(library &list, string id)
862. {

863. /*
864. Student check-in book -> fix book mark "for rent" only;

865. */
866. cout << ">>>Checking netID..." << endl;
867. renter p1;
868. bool check;
869. check = list.find_renter(id, p1);
870. if (!check)
```

```
871. {
872. cout << "...CANNOT find your info in the System!" << endl;
873. }
874. else
875. {
876. string answer;
877. vector<string> bookReturn = p1.get_bookRented();
878. cout << "\n-------------------------------------------------------------" << endl;
879. cout << ">>>Renter Info:" << endl;
880. cout << p1.get_renterName() << "\t" << p1.get_netID() << endl;
881. cout << "List of book have rented: " << endl;
882. for (int i = 0; i < bookReturn.size(); i++)
883. {
884. cout << i + 1 << ". " << bookReturn.at(i) << endl;
885. }
886. cout << "Date rent: " << p1.get_rentDate() << endl;
887. cout << "Number of day in rent: " << p1.get_day4rent() << endl;
888. cin.ignore();
889. cout << "\n-----------------------------------------------------------------" << endl;
890. cout << "\nAre you going to check-in books today? (yes/no)" << endl;
891. cout << "-> ";
892. getline(cin, answer);
893. if (answer == "yes")
894. {
895. int dayrent = 0;
896. int lateday = 0;
897. cout << "How many day did you rent? " << endl;
898. cout << "-> ";
899. cin >> dayrent;
900. //Renter return books-->Update book status again...
901. cout << "\n-------------------------------------------------------------" << endl;
902. cout << ">>>System checking-in books..." << endl;

903. for (int i = 0; i < bookReturn.size(); i++)
904. {
905. // mark book as False -> available for rent
906. list.markRentStatus(bookReturn.at(i), false);
907. cout << bookReturn.at(i) << " -->CHECKED-IN" << endl;
908. }
909. //Check for late days

910. bool late_check = p1.is_Late(dayrent, lateday);

911. if (late_check == false) //not late
912. {
913. cout << "\n*Thank you for return books on time! No late-fee required." << endl;
914. }
915. else
916. {
917. cout << "You are late for " << lateday << " days!" << endl;
918. cout << "Unfortunately, you have to pay for the late fee: $" << p1.get_lateFee() << endl;
919. }
920. list.delete_renter(id);
921. }
922. }



923. cout << "-------------------------------------------------------------" << endl;
924. }
925. void book_search(library &list)
926. {
927. //Finding book, printing info, offer renting status
```

```
928. cout << "------------------------------------------------------------" << endl;
929. cin.ignore();
930. int opt;
931. do
932. {
933. cout << "You want to search by: ";
934. cout << "\n\t#1. By Book Title\n\t#2. By Categories" << endl;
935. cout << "(Enter your option (by number 1 or 2)" << endl;
936. cout << "-> ";
937. cin >> opt;
938. } while (opt > 2 || opt < 0);
939. switch (opt)
940. {
941. case 1:
942. {
943. book_searchbyBook(list);
944. break;
945. }
946. case 2:
947. {
948. book_searchbyCatg(list);
949. break;
950. }
951. default:
952. break;
953. }

954. }
955. void book_searchbyCatg(library &list)
956. {
957. string answer;
958. int opt;
959. bool check;
960. cout << "------------------------------------------------------------" << endl;
961. do
962. {
963. cout << "There are 3 categories in the system:" << endl;
964. cout << "\n1. Mathematics";
965. cout << "\n2. English";
966. cout << "\n3. Economics";
967. cout << "\n4. History" << endl;
968. cout << "\nPlease enter your choice: " << endl;
969. cout << "-> ";
970. cin >> opt;
971. } while (opt > 4 || opt < 0);
972. if (opt == 1)
973. {
974. answer = "MATH";
975. cout << ">>>Book list fall under category Mathematics: " << endl;
976. list.printBook_byCatg(answer);
977. }
978. else if (opt == 2)
979. {
980. answer = "ENGL";
981. cout << ">>>Book list fall under category English: " << endl;
982. list.printBook_byCatg(answer);
983. }
984. else if (opt == 3)
985. {
986. answer = "ECON";
987. cout << ">>>Book list fall under category History: " << endl;
988. list.printBook_byCatg(answer);
989. }
```

```
990. else
991. {
992. answer = "HIST";
993. cout << ">>>Book list fall under category History: " << endl;
994. list.printBook_byCatg(answer);
995. }
996. cin.ignore();
997. cout << "\nSearch for book details? (enter yes/no)" << endl;
998. cout << "-> ";
999. getline(cin, answer);
1000.       if (answer == "yes")
1001.       {
1002.       book_searchbyBook(list);
1003.       }
1004.       cout << "------------------------------------------------------------" << endl;
1005.       }

1006.       void book_searchbyBook(library &list)
1007.       {
1008.       string answer;
1009.       bool check;
1010.       cout << "------------------------------------------------------------" << endl;
1011.       cin.ignore();
1012.       do
1013.       {
1014.       cout << "\nEnter name of the book: " << endl;
1015.       cout << "-> ";
1016.       getline(cin, answer);
1017.       if (list.isHere(answer) == true)
1018.       {
1019.       if (list.isAvalable_4rent(answer))
1020.       {
1021.       cout << "\n--->THIS BOOK IS AVAILABLE FOR RENT----" << endl << endl;
1022.       list.printBook_info(answer);
1023.       }
1024.       }
1025.       else
1026.       {
1027.       cout << "Sorry! " << answer << " is not in the library" << endl;
1028.       }
1029.       cout << "Do you want to search for another one? (enter yes/no): " << endl;
1030.       cout << "-> ";
1031.       getline(cin, answer);

1032.       } while (answer == "yes");
1033.       cout << "------------------------------------------------------------" << endl;
1034.       }
```

book.h (book header file):

- This is the beginning of the project where we begin by setting some of the basic structure in reference to the “books” to be able to extract the information when calling it from the Linked List “bookList.txt”
    - setter functions initiate here for the book information(name, code, info & status of the book.
    - getter functions initiate here for the book which will retrieve the information when called upon.

book.cpp (book cpp file):

- In this part of the program, is here all of the implementation done I header file are perform for the book. When the program executes this file helps the program perform certain tasks:
    - setBook allows us to retrieve the string of information of book name, book code, and the for loop(time complexity : 0(n)) which goes through the key and check the information
    - serRent helps determine if a book is currently already being rented out or not

- the getcode getname and get info do basic return on that information when the program is executed.

library.h (library header file):

- The library class is where the construction of the Nodes begin for the Linked Lists; we had to create two linked lists:
  - NodeType is for the general linked list which will allow us to traverse through and perform functions that will be listed below revolving around
  - NodeType2 is for use with the renter which has a few functions that attach to this.
  - Library class will have function headers that will better help determine if we are able to in fact rent a book, it is available, if it located inside of the library, if the book is already rented out, as well as having the administrator ability to add or delete a book if needed.
  - Renter functions are located here where you are able to find_renter, add_renter, delete_renter
  - The last part of this classes function revolve around printing the books information and category of which the book falls into and lastly getting the book names if searched by category or by author.

library.cpp (library cpp file):

- The library class will be where we will be able to basically search through an actual library for book, alter the books if need to add or delete, and obtain renter information.
  - Line 124-128: is the constructor for library function
  - Line 129-131:desconstrutor for library
  - Line 132-151: isAvailable_4rent is implemented and checks the bookList if it is empty, while it is not empty we want to be able to match an inputed book name with the names retrieved from the List, if this book returns then the book lets the student/user know that it is available. **Time complexity: 0(n)**
  - Line 152-170: isHere function is implemented, allows the user to search for a book to see if it is inside of the list. The book is able to be inside of this list, but not be available for rent as it may be possible to have been checked out but not yet picked up yet. **Time Complexity: 0(n)**
  - Line 171-221: insert_book is implemented here. We begin by initializing the Linked List in order to traverse through it properly. Here we have different cases available in case we choose to insert the book in different locations depending on the bookList and where this new book would be inserted into the sorted Linked list. **Time complexity of multiple if else statements :0(n^2)**
  - Line 222-240: delete_book is located. Allows us to search through list for certain book, if book found, delete book. **Time complexity: 0(1)**
  - Line 242-266: printBook_info is implemented, will traverse through the linked List until book searching for is found and output the information. **time complexity: 0(n)**
  - Line 267-283: markRentStatus is implemented which will return if the current book is rented out or not. It will traverse. **Time Complexity 0(n)**
  - Line 284-329: add_Renter is implemented. This will utilize the "NodeType2" Linked list which will create basic user account information for a renter in order to rent out books **Time Complexity: 0(n^2) due to nested if else statements**
  - Line 331-351: delete_Renter is implemented which will delete a renter if needed **time complexity:0(1)**
  - Line 352--373:PrintBook_byCatg will print all the books by the category.

Renter.h (Renter header file):

- The renter class is where they store all the information on each renter we input.
  - The function included: set ranter information, get renter name, get net ID, get rent date, get book name, get day on rent, check if it is late, and calculate the late fee.
  - On the private area, it will store the confidential information of the renter name, date, day of rent ,their ID, the status of is it late, the late fee, and the book they rent.

Renter.cpp (Rentercpp file):

- The renter class stored information of things like date, name, id, and fee.
- Here is the detail of how the renter functioning, specified by line number:
  - · Line 447 to 456: the constructor initializes renter name, date rent, day rent, is late, late fee, and net ID.
  - · Line 459 to 463: is the destructor
  - · Line 465 to 475: is the set renter information which include their name, date of rent, day of rent, is it late, late fee, the netID and the book that they rent.

- · Line 477 to 489: is late function which calculate the date and time that the book is overdue. The rate that we charge is 1.50 per day. It also set the status if it is late or not late.
- · Line 490 to 493: is get book rented which return the book that is rented out.
- · Line 495 to 498: is get renter name which return the name of the renter.
- · Line 499 to 502: is get net ID which return the ID of the renter.
- · Line 503 to 506: is get rent date which return the date that they rent the book.
- · Line 507 to 510: is get the day they rent which return the day that they rent the book.
- · Line 512 to 515: is get late fee which return the late fee of the renter.

Library System Project.cpp (main file):

- When the program is executed, there are two major task that will happen in this main file:
  - Create the library system and insert the books from another text file.
  - Operate the Library services which is specialized for administrator and student
    - For administrator: manage the book system (add/delete book) and search for book
    - For student : renting services(check-in/check-out book) and search for book
- Here are more detail of how the main functioning, specified by line number:
  - Line 563 to 570 identify the prototype of the functions that will be use in the main file, including:
    - void book_searchbyCatg(library &list);
    - void book_searchbyBook(library &list);
    - void book_search(library &list);
    - void book_addnew(library &list);
    - void book_takeout(library &list, bool for_rent, string id, string renter_name);
    - void book_checkin(library &list, string id);
    - void adminService(library &list);
    - void studentService(library &list, string id);
  - Line 571: identify the global variable time-t type called “todayDate”. This variable will be hold the date (input by user later on), which will be use for saving date checkout and date checkin for book renting later on.
  - From line 572-600: Start at the beginning of the main function, this part will read the information of each book from a separate text file named “booklist.txt”.
    - Inside the “booklist.txt", there are 14 book in total. Each book will have its individual information (name of the book, book code and book description) and be separated by the line “*****”
    - Line 573: create a library type named “list" to start operate the library system
    - From line 580 to line 598, the while loop will run through the “booklist.txt" until the end and reading information of each book into a book type named “input”.
    - Line 598: Then from that, insert “input” into the library “list" by calling the function list.insert_Book(input).
    - **Time complexity** for this while loop is O(n) since it has the loop and read every single line inside the “booklist.txt" file
  - From line 613 to 641 (end of the main function), the code operate the main part of the Library Service system:
    - The user will be asked for ID code. From the ID code provided, there will be an if/else statement to indicate user is student or administrator, in order to call out the correct function service. (There are 2 separate functions; one for student and one for administrator)
  - Line 641 to line 673: Operation Administrator Interface by adminService(library &list) function:
    - The whole function is a big do-while loop that will print out the available service for user and ask user to input their option. From user's input, program will go ahead indicate the service user want and call out the specific function for that service
    - There are another small do-while loop inside to checking the user's input. If the user's input is invalid, the program will go ahead print out the information lines again and request user to re-do the task. This loop complexity is O(n), it base on how many time user insert their option.
    - **Time complexity** for these two nested do-while loop is O(n^2). Program will operate as many time as user request for service or entering their service option.
  - Line 674 to line 719: Operation Student interface by studentService(library &list) function:

- As same as the adminService function, this function operated the similar way; whole function is a big do-while loop and a small do-while loop nested inside.
- **Time complexity** for these two nested do-while loop is also O(n^2). Program will operate as many time as user request for service or entering their service option.

- Line 715 to 803: book_takeout function:
  void book_takeout(library &list, bool for_rent, string id, string renter_name)

  - This function will illustrate two case of a book taking-out; one is the admin delete the book permanently out of the library system; other case is student check-out a book for rent, which system still hold the info of the rented book, but just simply marking it as “rented".

  - Line 723 to line 728 have a do-while loop asking the user to input how many books they want to “take out". **Time complexity** for this loop is O(n), the loop will continue request input if user enter an invalid value (a number < 0)

  - Line 739 to line 744:In the case user has entered the valid number of books they want to take out, program will run the for-loop asking the user to enter each name of the book. This loop will loop through multiple time base on how many book user enter. **Time complexity** for this for-loop is O(n)

  - Line 747 to line 803: After asking the list of books user want to “take-out". Function go ahead analyze the passing value “for_rent" to check whether what task it should do next.

    - Line 746 to 755: Case 1 → If user is administrator (“for_rent" = false), function will go through the list of book by a for-loop and delete each book, indicated by its name, out of the library permanently. **Time complexity** for this for-loop is O(n).
    - Line 756 to line 803: Case2 → If user is a student, function will do a few different task served as renting service set.
      - Line 770 to line 792: First, the system will check for the existence and the availability of each requested book. While loop, if the book is in the library and available for rent then system will print out “-->RENTED" and mark the book for a rented flag. If the book is not exist or marked “rented" already, program will print out the notification as the item is currently not available. Note: The name of the book that not exist will be eliminate from the list of the requested book by using erase function (in string library). This is done by a for-loop. **Time complexity** for this loop is O(n)
      - Line 761 to line 766: Second, the user will be ask if they want to rent for how many day. This task is done by a do-while loop; it will repeat asking if user enter more than 30 days. **Time complexity** for this loop is O(n)
      - Line 770: Create a specific variable renter type called “p1". Then set all the needed information of the user into this variable.
      - Line 795: Now, when every necessary information of the user whom using the renting service is saved in “p1", program go ahead call function list.add_renter(p1) to add new renter into the system.
      - Line 796 to line 799: Program will print out the user's info and list of book they had rent out in the screen as confirmation of what have been saved into the system.
      - Line 800 to line 803: A small for-loop to again printing out all the book user requested for rent. **Time complexity** for this is O(n).

- Line 808 to line 860: book_addnew function
  - This function is specifically for administrator access.
  - Line 813 to Line 818: A do-while loop ask for how many book user want to add into the library. The loop will keep looping until the value entered is larger or equal to 0.
  - Line 824 to line 857: In the case, the user’s input is larger than 0:
    - Line 830 to line 835: A for-loop will perform to ask each name of the book user want to add. Then the book name will be push into a vector string named “bookname". **Time complexity** for this loop is O(n)
    - Line 836 to line 856: Another for-loop will ask for more detail about each book user had entered. Then create a book type named “newbook" to save all the info it just received. Then put it into the library system by calling function list.insert_book(newbook).

Alos, there is a small do-while loop nested inside a bog for-loop to read all the input that user entered for the book information. **Time complexity** for this loop is O(n^2).

- Line 861 to line 924: book_checkin function
  - This function is specialized for student check-in book, part of the renting service. this function will demonstrate the act of a student return their book. System will check user's ID to get more information about their renting activities.
  - Line 870 to line 973: Case 1 → If the ID of user is not matched to any renter in the system, program will print out the notification line.
  - Line 874 to line 908: Case 2 → If the ID of the user is matched. Program will go ahead perform a quite few steps to process the check-in service, including:
    - Line 879 to line 890: Program will print out the info of user, then asking them if they want to return the book today.
    - Line 9903 to line 908: If the answer is yes, the program will start marking the availability of each book they had rented to "available" calling function list.markRentStatus(bookReturn.at(i)) and print out the notification on the screen to tell user each book are been checked-in. This task is done by one for-loop, which looping through the list of rented book that already have in the system. **Time complexity** for this loop is O(n).
    - Line 909 to line 918: Next, program will checking for renter's late fee. If the day they return the book exceeded the day they promised in the past by the code:

      late_check = p1.is_late(daylate,todayDate); (line 885)

      Then program will go ahead calculate the fee and print out the amount they had to paid by the code:

      `p1.get_lateFee() (line 894)`

      If they return book on time, system will print a notification on the screen that they don't have any late fee.

    - Line 920: Since the user has checked-in all books they had rented, therefore, there's information will be delete
      List.delete_renter(id)

- Line 925 to line 954: book_search function
  - This function is the main part of the book searching service.
  - In this function, the user will be given to option to search for book; one is search by book name; second is search by categories.
  - Line 931 to line 938 is the part where program asking user to choose their searching option by using a do-while loop. **Time complexity** for this is O(n).
  - Line 939 to line 952: The program will operate the switch to choose indicate the option user had chosen, then go ahead calling the appropriate function for the specific search option.
- Line 955 to line 1005: book_searchbyCatg function
  - This function is one of two function provide the option for searching service. Specifically, this function is called when user decide to search book by categories.
  - Line 963 to Line 971: The program will print out all the available categories currently in the library. Then it will ask user to enter their category choice. This is done by a do-while loop; the loop will be continue until user enter one of the provided option. **Time complexity** is O(n).
  - Line 972 to line 995: Base on the option user chose, the program will fall under a specific condition, and then call the function to print the list of book under the requested category

    list.printBook_byCatg(answer);

  - Line 1000 to line 1003: A extend service option is provided when user choose to search book by categories. User also will be asked by the program whether they want to move on to search for the info of a specific name book; that by chance they found interested when they look over the book list of the requested categories. If the answer is yes, the program will call the function that searching book by name.
- Line 1006 to line 1034: book_searchbyBook function

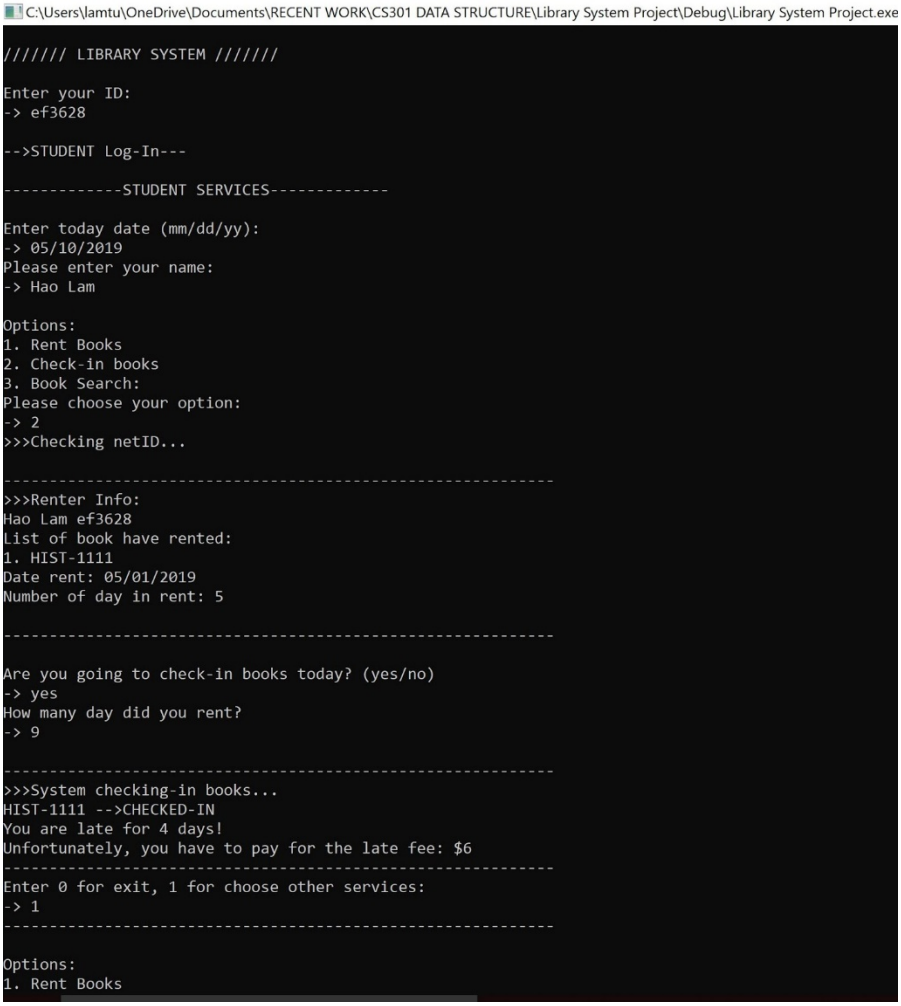
- This function is one of two function provide the option for searching service. Specifically, this function is called when user decide to search book by specific name.
- The whole function is the big do-while loop that will continue to loop if the answer of the user is a “yes" when they have been asked for continuing searching for another one. **Time complexity** of this loop is O(n).
- Inside the loop, the user will be asked to enter the name of the book they want to search. Then program will go ahead checking the book existence.
    - If the book is not in the library, the program will print out the notification
    - If the book is in the library, program will print out the notification about its availability for rent. Then go ahead call the function to print the requested book info

```
list.printBook_info(answer);
```

- **SAMPLE RUN:**

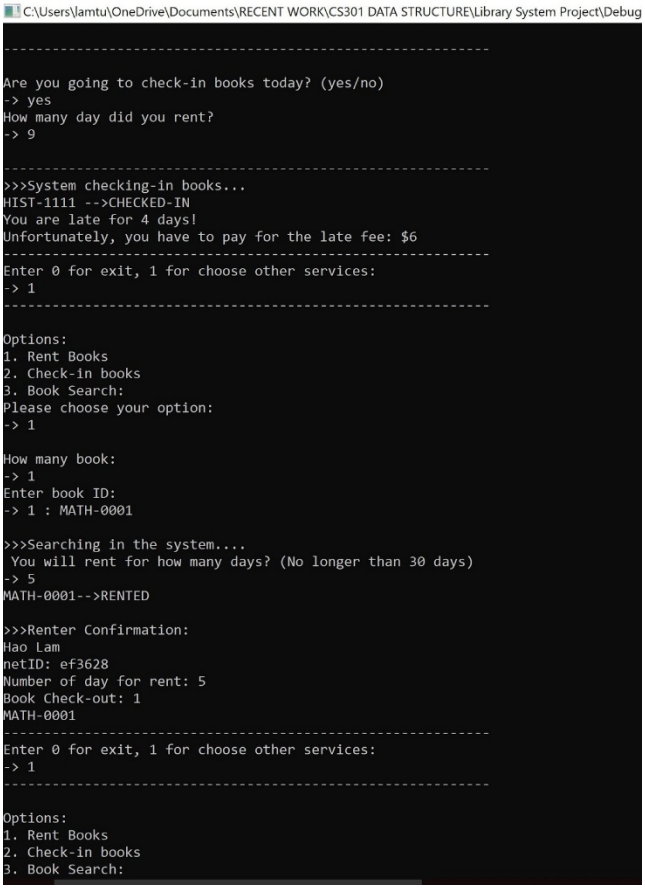
- **STUDENT INTERFACE TEST-RUN:**
    - Case scenario:
        - A student have already rented a book in the past. System has saved this student’s info. This step is added before the program operated library service.
        - On another day, this student come to check-in book she had rent. The program will print out her info as well as her list of rented book. When she decides to check-in, the program will calculate her late fee if she returned late and print out as a notification.

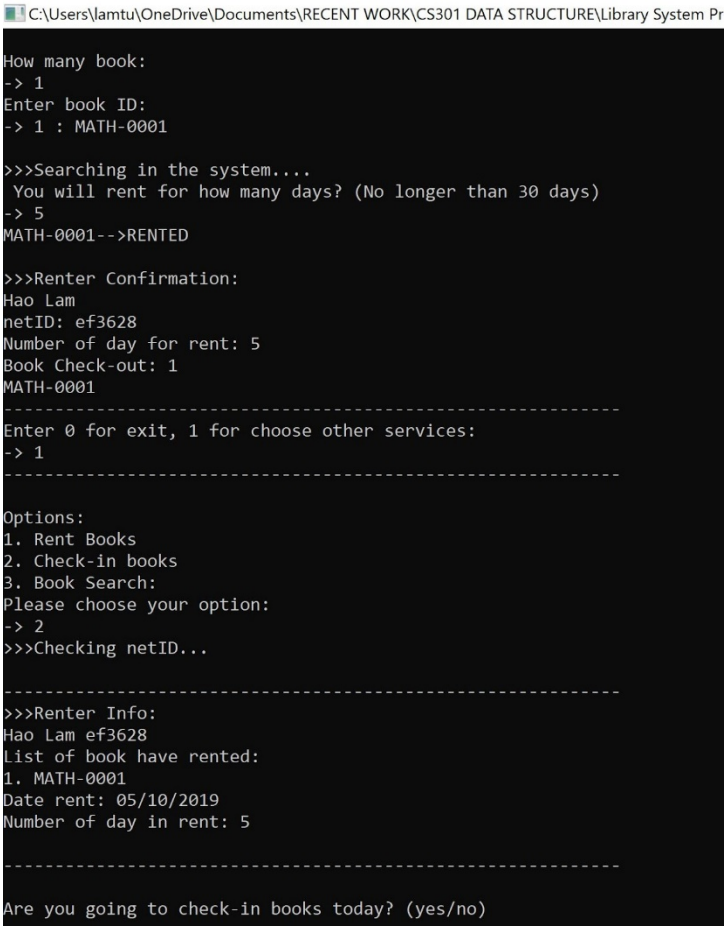
LIBRARY MANAGEMENY REPOR SAMPLE group 3



- Then use the renting service again to rent different book ad checkout it
- The student go to check-in again, at here, we could see her information has been changed; the newest book she rented has been updated on the list. The last book she checked-in in the past has been removed

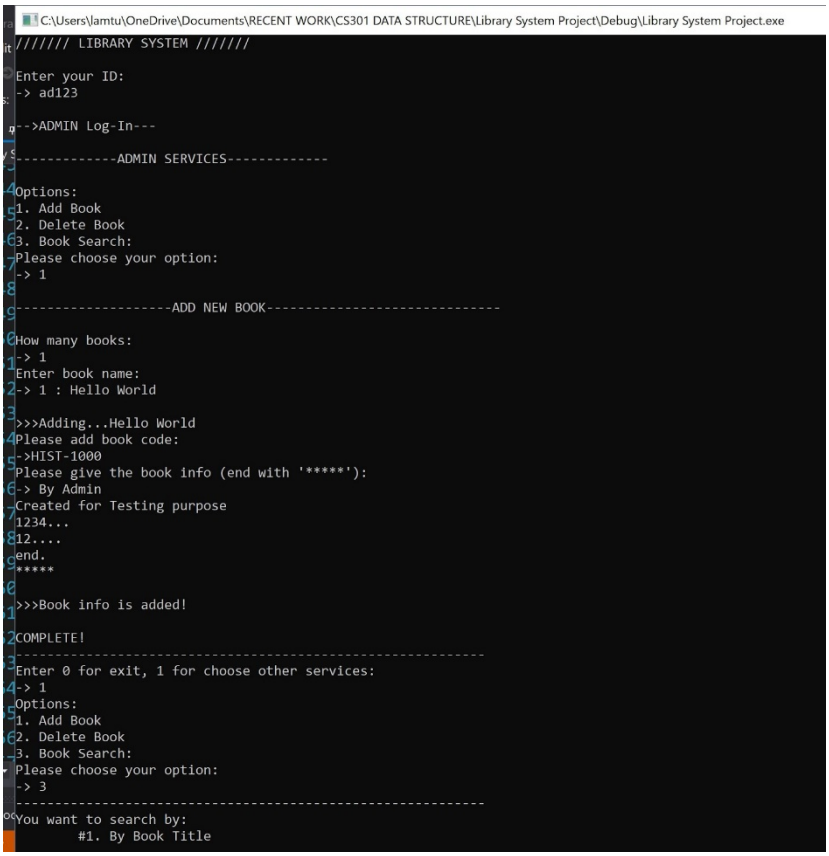
LIBRARY MANAGEMENY REPOR SAMPLE group 3





LIBRARY MANAGEMENY REPOR SAMPLE group 3

- **<u>ADMINISTRATOR INTERFACE TEST-RUN:</u>**
  - Case scenario:
    - An Admin add a new book into the library.



- Then search book by category. (category is purposely chosen as same as the category of the new book just added)
- In the book list, the “History” category only has 1 book. Now the list has become 2 since the new book is added (new book is in “history” category)