Waitong Suen a1790760 – Hyun Ji Moon a1793295 - OOP Semester 2, 2020 - The University of Adelaide

**Project Specification - Major Practical**

**Introduction**

Our project is about library systems that we use in real life, such as logging system, borrow-return material (book, e-book and DVD) system and managing library material system for staff. The library system allows the current users (public, staff) log in, log out, borrow materials, return materials and change password. It also allows a new user to make a new account. There are more specific behaviours that the staff is able to access, such as check history and add materials.

**Design Description**

**Assessment Concepts**

**Memory allocation from stack and the heap**

* **Arrays :** we are using a dynamic array for storing the books(DVDs) that the user have borrowed.
* **Vectors :** we are using a vector for storing all the materials(ebooks, books, dvds) and user(public, staff) lists
* **Strings :** material names, author names and user names
* **Objects :** Ebook, book, DVD, public, staff

**User Input and Output**

* **I/O of different data types :** Command-line.
* A person enters a number to choose log in types(user, staff or new user) - console displays the possible options that user or staff can select.
* The user is asked to enter a behaviour number to borrow materials(ebook, book or dvd), return materials or change password - the program displays corresponding outputs.
* The staff is asked to input a number to choose the opitions of borrowing materials, returning materials, changing password, checking history or adding material – the program prints out the corresponding outputs.

**Object-oriented programming and design**

* **Inheritance:** 
  + material -parent class / Ebook, book and DVD – child classes
    - There are different types of materials such as Ebook, book and DVD.
    - Each child class (Ebook, book, DVD) inherits from the parent class(material)
    - Each child class has few different behaviours to other child class
  + user(staff and public)
    - There are two types of users such as staff and public
    - Both child classes inherit from parent class(user)
    - Each class has few different behaviours (the staff user can access more library system)
* **Polymorphism:** 
  + test\_use\_material\_setup(string t\_name, string t\_author) function
    - Child classes(Ebook, book and DVD) have the same function for quick setting up the objects but each child class has different implementations
  + check\_history() function
    - Child classes(Ebook, book and DVD) have the same function check\_history() but the outcomes are different. The check history for book(DVD) prints out the list of people who borrowed the book(DVD) and the check history for ebook prints out the number of ebooks that people have downloaded
* **Abstract classes:**
  + borrow\_material(string borrow\_user) = 0;
    - Every child class has the function called borrow\_material to borrow the material, which has different outcomes.
    - According to ebook function(borrow\_material), we are going to count out how many ebooks the user has downloaded. However, the borrow\_material function for book and DVD shows the situation of availability of borrowing the book and DVD.
  + add\_new\_material() = 0;
    - Parent class does not declare the add\_new\_material function details but every child class (ebook, book, dvd) defines different function details.
    - The add\_new\_material function for Ebook and book asks user for entering material’s name, author’s name and page numbers but the add\_new\_material function for DVD save the information about material’s name, author’s name and length.

**Class Diagram**

A picture containing table

Description automatically generated

Diagram

Description automatically generated

**Class Descriptions**

**material:**

the superclass of all material in the library.(include book, ebook and DVD), contain attribute :

material name

author\_name

situation (of available to borrow or not)

vector<string> borrow\_user\_list (record the name of user who borrow this material)

**ebook:**

child class of material, one of the material type. contain extra attribute:

download\_count (count the time user download this book instead of borrow)

page\_num (pages record)

**book:**

child class of material, one of the material type. contain extra attribute:

page\_num

**DVD:**

child class of material, one of the material type. contain extra attribute:

length (time length of CD);

**user:**

the superclass of staff. contain attribute

user\_ID;

maximum borrow (limit number of book can borrow)

current borrow (count of recent book borrow)

user\_password

user\_name

book\_count & DVD count (use to calculate the user borrow and check it hit the limit.)

mybook, myDVD (array for storing material that user have borrowed)

**User Interface**

The users (public, staff) is going to input the numbers to choose one of the options using the command-line.

The options that the users can select is like:

*What would you like to do today?*

*1: Borrow material*

*2: Return material*

*3. Change current password*

*4. Check history (only for staff)*

*5. Add material (only for staff)*

*6. Delete material (only for staff)*

**Testing Plan**

We are going to compile all the codes through our makefile and for the accuracy, we are going to check our codes through various testing cases, such as unit testing, input/ouput testing, integration testing, boundary testing, automated testing and regression testing.

**Unit Testing**

We have separate class function files and corresponding test files to check if the individual files can be implemented. When each individual class is tested successfully, we test the main file to see if all files are well compiled and run completely.

* *test\_choose\_material: material.o choose\_material.cpp choose\_material\_test.cpp*

*g++ material.o choose\_material.cpp choose\_material\_test.cpp -o test\_choose\_material $(FLAGS)*

* *test\_log\_in: user.o log\_in.cpp log\_in\_test.cpp material.o choose\_material.cpp*

*g++ user.o log\_in.cpp log\_in\_test.cpp material.o choose\_material.cpp -o test\_log\_in $(FLAGS)*

* *user\_unit\_testing : material.o user.o user\_unit\_testing.cpp choose\_material.cpp*

*g++ material.o user.o user\_unit\_testing.cpp choose\_material.cpp -o user\_unit\_testing $(FLAGS)*

* *material\_unit\_testing : material.o user.o material\_unit\_testing.cpp choose\_material.cpp*

*g++ material.o user.o material\_unit\_testing.cpp choose\_material.cpp -o material\_unit\_testing $(FLAGS)*

**Input/output testing**

Through the input/output testing, we check if the output is equal to the expected output. we have several separate input and expected output files to go through every possible result.

*inout\_test:*

*./test < input01.txt | diff - output01.txt*

*./test < input02.txt | diff - output02.txt*

*./test < input03.txt | diff - output03.txt*

*./test < input04.txt | diff - output04.txt*

*./test < input05.txt | diff - output05.txt*

*./test < input06.txt | diff - output06.txt*

*./test < input07.txt | diff - output07.txt*

*./test < input08.txt | diff - output08.txt*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | input file | functions(input) | output file | expected output |
| 1 | input01.txt | user log in  quit the system | output01.txt | log in type  ID, password  initial screen |
| 2 | input02.txt | new user log in  quit the system | output02.txt | log in type  set up ID, name, password  log in ID, password  initial screen |
| 3 | input03.txt | user log in  borrow material(book)  return material  quit the system | output03.txt | log in type  log in ID, password  options  material list  borrow(material name, author, page number)  return type  initial screen |
| 4 | input04.txt | user log in  change password  quit the system | output04.txt | log in type  log in ID, password  options  new password  options  initial screen |
| 5 | input05.txt | user log in  borrow material(DVD)  check current borrow list  quit the system | output05.txt | log in type  log in ID, password  options  borrow(material list)  current borrow list  initial screen |
| 6 | input06.txt | staff log in  add material(ebook)  add material(book)  add material(DVD)  quit the system | output06.txt | log in type  staff ID, password  options  add material(name, autor, page number)  (ebook, book, DVD)  initial screen |
| 7 | input07.txt | staff log in  borrow material  check borrow list  return material  quit the system | output07.txt | log in type  staff ID, password  options  (borrow)material lists  material name, author, page number  borrow list  return  initial screen |
| 8 | input08.txt | staff log in  change password  quit the system | output08.txt | log in type  staff ID, password  options  new password  initial screen |

**Integration testing**

We have a main.cpp file, which integrates all functions together that we have tested in separate test files so far. Through the main file, we do integration testing to see if all files are well compiled and run completely.

* *All:* *main.cpp material.o user.o log\_in.cpp choose\_material.cpp user\_behavior.cpp list\_of\_materials.cpp tests*

*g++ main.cpp material.cpp user.cpp log\_in.cpp choose\_material.cpp list\_of\_materials.cpp user\_behavior.cpp -o test $(FLAGS)*

* *test\_user\_setup : material.o user.o user\_setup\_test.cpp choose\_material.cpp*

*g++ material.cpp user.cpp user\_setup\_test.cpp choose\_material.cpp -o test\_user\_setup $(FLAGS)*

* *test\_user\_borrow\_return\_test :* *material.o user.o user\_borrow\_return\_test.cpp choose\_material.cpp*

*g++ user.cpp material.cpp user\_borrow\_return\_test.cpp choose\_material.cpp -o test\_user\_borrow\_return\_test $(FLAGS)*

* *test\_material :* *material.o material\_test.cpp*

*g++ material.cpp material\_test.cpp -o test\_material $(FLAGS)*

* *test\_material\_borrow\_return\_checkhistory :* *material.o material\_borrow\_return\_checkhistory\_test.cpp*

*g++ material.cpp material\_borrow\_return\_checkhistory\_test.cpp -o test\_material\_borrow\_return\_checkhistory $(FLAGS)*

* *test\_user\_staff :* *material.o user.o choose\_material.cpp user\_staff\_test.cpp*

*g++ material.cpp user.cpp choose\_material.cpp user\_staff\_test.cpp -o test\_user\_staff $(FLAGS)*

**Boundary Testing**

We check not only the boundary of every possible range but also upper and lower limits of the range of inputting data which should be accepted in the system. We also test invalid data that the program should not accept.

1. while(user\_choose<1 || user\_choose>4)

🡺 let the users know that they are out of the range they can choose and ask them to enter the number again

2. when the user input the wrong ID and the wrong password

🡺 inform them of wrong valid number and ask them to enter valid number again

3. when the user enters the material code that is out of the range

🡺 let them know they input wrong number and ask them re-enter valid number

4. when the staff adds material and enters the number (less than 1 and greater than 3)

🡺 going back to the previous screen so that they can choose the number again

**Automated Testing**

We have a makefile to check every file automatically and through the input/output testing, we do automated check of comparing the actual outcomes with the predicted outcomes.

**Regression Testing**

Whenever we changed the code or added the code, we checked if the code changes affected existing code and the program still performs after a change.

A close up of a screen

Description automatically generated

**Schedule Plan**

**Stretch Goals**

Our goal is to make a diagram to check if we satisfy all the rubrics and write down the pseudo codes for every class and functions. When we satisfy the requirements, we are going to expand our idea.

|  |  |
| --- | --- |
| week 8 | * Write down the diagram - *Waitong* * Make a group and create SVN – *Hyun Ji* * make our plan – Waitong, Hyun Ji |
| break | * organise the group works – *Waitong, Hyun Ji* * separate the classes and write some codes - *Waitong* * write down separate codes(class, function) - *Waitong* * material.h / material.cpp * user.h / user.cpp * add\_material\_function\_list.cpp * write down separate corresponding test files– *Waitong*, *Hyun Ji* * add\_material\_test.cpp * borrow\_return\_DVD\_test.cpp * borrow\_return\_book\_test.cpp |
| week 9 | * write down project plan – *Waitong, Hyun Ji* * write down separate codes(class, function) - *Waitong* * log-in.cpp * add\_material\_function\_list.cpp * user\_behaviour.cpp * write down separate corresponding test files and check if the individual files are run properly – *Waitong*, *Hyun Ji* * material\_test.cpp * user\_borrow\_return\_test.cpp * user\_setup\_test.cpp * log\_in\_test.cpp * write makefile - Waitong * testing |
| week 10 | * edit project plan – *Hyun Ji* * write down separate codes(class, function) - *Waitong*   + choose\_material.cpp * write down separate corresponding test files– *Waitong*, *Hyun Ji*   + material\_borrow\_return\_checkhistory\_test.cpp   + user\_staff\_test.cpp   + choose\_material\_test.cpp * write down the main file combining all the codes – *Waitong*   + main.cpp * correcting the codes and debug – *Waitong, Hyun Ji* * write makefile – *Waitong, Hyun Ji* |
| week 11 | * correcting the codes and debug – *Waitong* * fixing some problems that we made in week 10 *–Hyun Ji* * testing final main file – *Waitong, Hyun Ji* * write makefile – *Waitong, Hyun Ji* * write down specific testing files - *Waitong, Hyun Ji* |