**BACHELOR OF COMPUTER SCIENCE**

**SCHOOL OF COMPUTER SCIENCE**

**BINA NUSANTARA UNIVERSITY**

**JAKARTA**

**ASSESSMENT FORM**

**Course: COMP6791051 -** **Program Design Methods**

**Method of Assessment: Case Study**

**Semester/Academic Year : Odd /2022-2023**

**Name of Lecturer : D6672 – Anang Prasetyo**

**Date : 13 December 2022**

**Class** **: LA95, LB95, LC95**

**Topic** **: Review Material**

|  |  |
| --- | --- |
| **Name** | **Michael Geraldin Wijaya** |
| **NIM** | **2602238021** |
| **Class** | **LC95** |

**Student Outcomes:**

**(SO 1) Able to analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions (SO 2) Able to design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of computer science**

**Learning Objectives:**

**(LObj 1.2) Able to apply the principles of computing and other relevant disciplines to identify solutions**

**(LObj 2.2) Able to implement a computing-based solution to meet a given set of computing requirements in the context of computer science**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NO** | **Learning Outcome** | **Weight** | **Key Indicator** |  | **Proficiency Level** | |  | **Score** | **Weight x Score** |
| **Excellent** | **Good** | **Average** | **Poor** |
| **(85 - 100)** | **(75 - 84)** | **(65 - 74)** | **(0 - 64)** |
| 1 | LO1: Explain the  requirements of writing a  program or system | 15% | The ability to gather the requirements of a program | The requirement gathered are all correct and complete | The requirement gathered are all correct and almost complete | The requirement gathered are partly correct and almost complete | The requirement gathered are incorrect and incomplete |  |  |
| The ability to explain the requirements for a program | All of the requirements are explained correctly | Some of the requirements are explained correctly | Some of the requirements are explained partly correct | Some of the requirements are explained incorrectly |
| 2 | LO2: Apply pseudocode to solve problem | 30% | The ability to identify the programming problem | At least 90% of the problem is identified correctly | At least 75% of the problem is identified correctly | At least 60% of the problem is identified correctly | Less than 60% of the problem is identified correctly |  |  |
| The ability to apply pseudocode to solve the programing probem | The pseudocode written solves at least 90% of the programming problem | The pseudocode written solves at least 75% of the programming problem | The pseudocode written solves  at least 60% of the programming problem | The pseudocode written solves less than 60% of the programming problem |
| 3 | LO3: Analyze the  requirements of a system | 25% | The ability to collect the requirements for a system | All of the requirement of a system is collected | Some of the requirement of a system is collected | A few of the requirement of a system is collected | The requirement of a system is not collected |  |  |
| The ablity to analyze the collected requirements in order to solve the system | All of the requirements are analyzed correctly | Some of the requirements are analyzed correctly | Some of the requirements analyzed are partly correct | The requirements are analyzed incorrectly |
| 4 | LO4: Solve system design problems using UML | 30% | The ability to design a system using UML Structural Modeling | The ability to design at least 90% correct using Structural Model1 | The ability to design at least 75% correct using Structural Model | The ability to design at least  60% correct using Structural  Model | The ability to design less than  60% correct using Structural  Model |  |  |
| The abIlity to design a system using UML Behavioral Modeling | The ability to design at least 90% correct using behavioral Modeling | The ability to design at least 90% correct using behavioral Modeling | The ability to design at least  90% correct using behavioral  Modeling | The ability to design at least  90% correct using behavioral  Modeling |

Remarks:

Case study is a personal work

**ASSESSMENT METHOD**

Instructions

The library is one of the most important parts of the Binus @Semarang Campus, the library is a place for students to look for study references, discussions, or just read books. As a programmer, you are asked to help the library section to develop a library system with the following criteria:

- The system must include features for borrowing books, returning books, searching for books, managing book stock

- The system can also calculate penalties if students are late in returning books

To build a library system, first make some of the things needed below:

1. Make a **pseudocode / flowchart (choose one),** the features for borrowing books and returning books
2. Write down the **functional and non-functional requirements** for the case above
3. Make a **use case diagram** for the library system
4. Make a **class diagram** of the case above

Final Output

Format :

* + Personal Assignment (**Please use application tools for creating any diagrams for this assignment**).
  + Format file please export to **pdf format**.

**Due Date** : Submit this final assignment in week 13

1. **PSEUDOCODE**

**Struct** Book(

integer id

string title

string author

string subject

integer stock

)

**Struct** Loan(

integer id

integer book\_id

integer student\_id

time\_t due\_date

real late\_fee

)

**Struct** User(

int id

string name

string email

string address

)

Declare integer num\_books

Set num\_books = 0

**Struct** Book as array books

Declare integer num\_loans

Set num\_loans = 0

**Struct** Loan as array loans

Declare integer num\_users

Set num\_students = 0

**Struct** Student as array students

**Module loadData()**

**Declare InputFile book\_file**

**Open book\_file “Books.txt”**

**If book\_file == NULL then** if (book\_file == NULL)

Display “Error opening Books file (Books file doesn’t exist)” printf("Error opening books file\n");

Return //When the return statement is executed, control is returned to the calling module, and any subsequent statements in the loadData() module are not executed

**End if**

**While not EOF (book\_file)**

Read book\_file books[num\_books].id, books[num\_books].title, books[num\_books].author, books[num\_books].subject, books[num\_books].stock

Set num\_books++

**End while**

**Close book\_file**

**Declare InputFile loan\_file**

**Open loan\_file “Loans.txt”**

**If loan\_file == NULL then** if (loan\_file == NULL)

Display “Error opening Loans file (Loans file doesn’t exist)” printf("Error opening loans file\n");

Return //When the return statement is executed, control is returned to the calling module, and any subsequent statements in the loadData() module are not executed

**End if**

**While not EOF (loan\_file)**

Read loan\_file loans[num\_loans].id, loans[num\_loans].book\_id, loans[num\_loans].student\_id, loans[num\_loans].due\_date, loans[num\_loans].late\_fee

Set num\_loans++

**End while**

**Close loan\_file**

**Declare InputFile student\_file**

**Open student\_file “Students.txt”**

**If student\_file == NULL then**

Display “Error opening Students file (Students file doesn’t exist)”

Return //When the return statement is executed, control is returned to the calling module, and any subsequent statements in the loadData() module are not executed

**End if**

**While not EOF (student\_file)**

Read student\_file students[num\_students].id, students[num\_students].name, students[num\_students].email, students[num\_students].address

Set num\_students++

**End while**

**Close student\_file**

**End module**

**Module saveData()**

Declare integer i

**Declare OutputFile book\_file**

**Open book\_file “Books.txt”**

**For i:=0 to num\_books**

Write book\_file books[i].id, books[i].title, books[i].author, books[i].subject, books[i].stock

**End for**

**Close book\_file**

**Declare OutputFile loan\_file**

**Open loan\_file “Loans.txt”**

**For i:=0 to num\_loans**

Write loan\_file loans[num\_loans].id, loans[num\_loans].book\_id, loans[num\_loans].student\_id, loans[num\_loans].due\_date, loans[num\_loans].late\_fee

**End for**

**Close loan\_file**

**Declare OutputFile student\_file**

**Open student\_file “Students.txt”**

**For i:=0 to num\_students**

Write student\_file students[i].id, students[i].name, students[i].email, students[i].address

**End for**

**Close student\_file**

**End module**

**Module borrowBook()**

Declare integer book\_id, student\_id, i

Display “Enter the Book ID”

Input book\_id

Display “Enter the Student ID: ”

Input student\_id

// Check if book is in stock

Declare integer book\_index

Set book\_index = -1

**For i:=0 to num\_books for (int i = 0; i < num\_books; i++)**

**If books[i].id == book\_id then**

Set book\_index = i

Break

**End if**

**End for**

**If book\_index == -1 then**

Display “Book not found”

Return//When the return statement is executed, control is returned to the calling module, and any subsequent statements in the borrowBook() module are not executed

**End if**

**If books[book\_index].stock == 0 then**

Display “Book is not in stock”

Return//When the return statement is executed, control is returned to the calling module, and any subsequent statements in the borrowBook() module are not executed

**End if**

// Check if user exists

Declare integer student\_index

Set student\_index = -1

**For i:=0 to num\_students for (int i = 0; i < num\_students; i++)**

**If students[i].id == student\_id then**

Set student\_index = i

Break

**End if**

**End for**

**If student\_index == -1 then**

Display “Student not found”

Return//When the return statement is executed, control is returned to the calling module, and any subsequent statements in the borrowBook() module are not executed

**End if**

// Update book stock

Set books[book\_index].stock--

// Create new loan

**Struct** Loan as loan

Set loan.id = num\_loans + 1

Set loan.book\_id = book\_id

Set loan.student\_id = student\_id

Set time\_t current\_time = time(NULL)

Set loan.due\_date = current\_time + 14 \* 24 \* 60 \* 60

Set loan.late\_fee = 0

Set loans[num\_loans] = loan

Set num\_loans++

Display “Book borrowed succesfully”

**End module**

**Module returnBook()**

Declare integer book\_id, student\_id, i

Display “Enter the Book ID: ”

Input book\_id

Display “Enter the Student ID: ”

Input student\_id

// Find loan

Declare integer loan\_index

Set loan\_index = -1

**For i:=0 to num\_loans for (int i = 0; i < num\_loans; i++) {**

**If loans[i].book\_id == book\_id AND loans[i].student\_id == student\_id then**

Set loan\_index = i

Break

**End if**

**End for**

**If loan\_index == -1 then if (loan\_index == -1)**

Display “Loan not found”

Return//When the return statement is executed, control is returned to the calling module, and any subsequent statements in the returnBook() module are not executed

**End if**

// Find book

Declare integer book\_index

Set book\_index = -1

**For i:= 0 to num\_books**

**If books[i].id == book\_id then**

Set book\_index = i

Break

**End if**

**End for**

**If book\_index == -1 then**

Display “Book not found”

Return//When the return statement is executed, control is returned to the calling module, and any subsequent statements in the returnBook() module are not executed

**End if**

// Update book stock

Set books[book\_index].stock++

// Check if the book was returned late

Set time\_t current\_time = time(NULL)

**If current\_time > loans[loan\_index].due\_date then**

Set loans[loan\_index].late\_fee = (current\_time - loans[loan\_index].due\_date) / (24 \* 60 \* 60) \* 100; // Late fee is $1 per day

Display “Book was returned late. The late fee is $loans[loan\_index].late\_fee/100”

**Else**

Display “Book returned succesfully”

**End if**

// Remove loan

**For i:= loan\_index to num\_loans for (int i = loan\_index; i < num\_loans - 1; i++) {**

**Set loans[i] = loans[i+1]**

**End for**

Set num\_loans--;

**End module**

**Module main()**

**Call** loadData()

Declare integer choice

**While choice not 5**

Display “BINUS @SEMARANG LIBRARY”

Display “1. Borrow Book”

Display “2. Return Book”

Display “3. Search for Book”

Display “4. Manage Book Stock”

Display “5. Exit”

Display “Enter Choice: ”

Input choice

**Select** choice

**Case 1:**

**Call** borrowBook()

Break

**Case 2:**

**Call** returnBook()

Break

**Case 3:** //Here I do not make pseudocode for cases 3 and 4 because the instructions given are only cases other than 3 and 4 ( 1 and 2)

**Case 4: //**Here I do not make pseudocode for cases 3 and 4 because the instructions given are only cases other than 3 and 4 ( 1 and 2)

**Case 5:**

**Call** saveData()

Display “Exit…”

Break

**Default:**

Display “Invalid choice, please input the valid seletion”

Break

**End select**

**End while**

**End module**

1. **Functional and Nonfunctional Requirements**

**Functional Requirements**

# **Borrow Books**

* 1. Ability to borrow books by logging in with a student ID and selecting a book from the available stock.

# **Return Books**

* 1. Ability to return books by logging in with a student ID and selecting a book from the list of borrowed books.
  2. Ability to calculate penalties for late book returns by comparing the return date with the due date

# **Search Books**

## Ability to search for books by title, author, or subject.

# **Manage Books**

* 1. Ability to manage book stock by adding new books, editing book information, and deleting books.

**Nonfunctional Requirements**

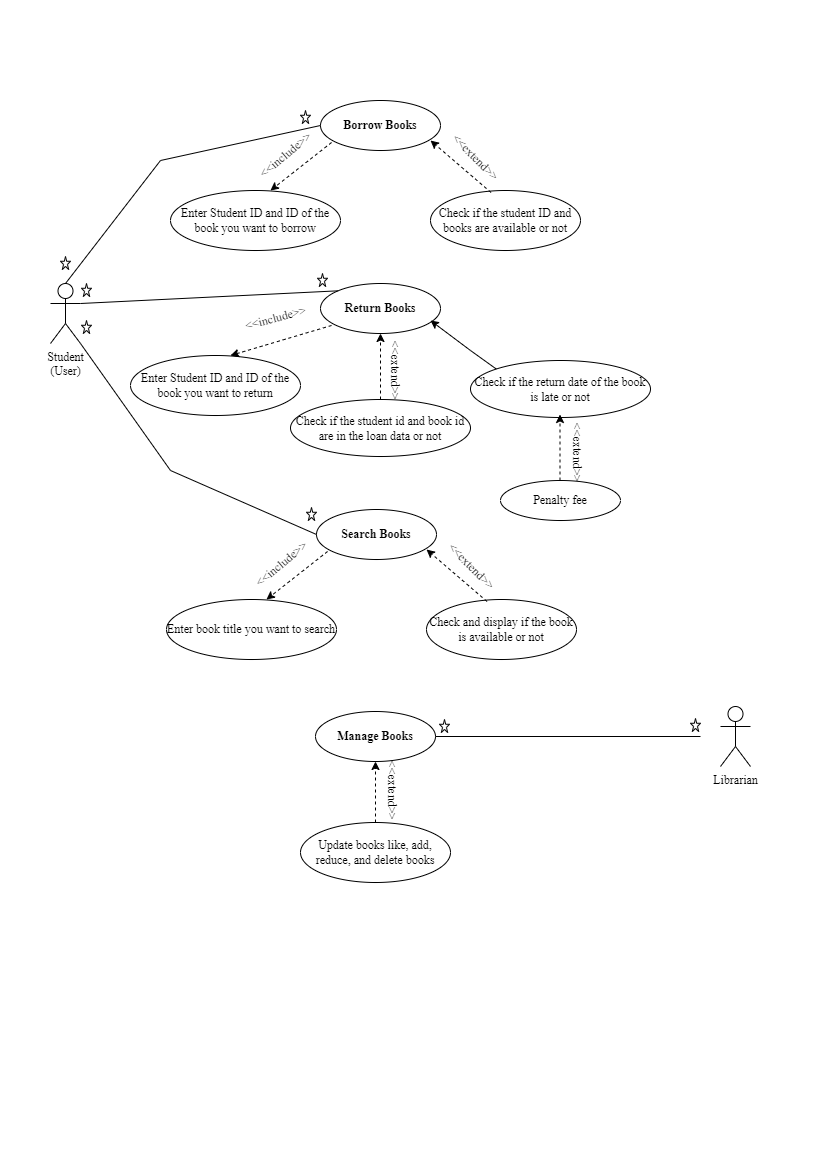
1. **Usability Requirements**
   1. The system should have a user-friendly interface with intuitive navigation and simple search capabilities.
2. **Maintenance Requirements**
   1. The system should be easy to maintain and update as needed.
3. **Security Requirements**
   1. The system should ensure the security and privacy of student and book information.
4. **Scalability Requirements**
   1. The system should be able to accommodate an increase in the number of books and students in the future.
5. **Integration Requirements**
   1. The system should be able to integrate with existing library systems and other university systems, such as student information systems.
6. **Performance Requirements**

The system must efficiently complete tasks without causing delays for the user.

1. **Compatibility Requirements**

The system should be able to function on a variety of devices and operating systems.

1. **Use Case Diagram**

****

1. **Class Diagram**

**Diagram

Description automatically generated**