

LIBRASYNC DOCUMENTATION

VERSION: 1.0

GITHUB: <https://github.com/krishna18developer/CampusConnect/>

Librasync is a command-line application designed to automate library tasks. It facilitates book and patron management, borrowing, returning, and searching functionalities.

TEAM DETAILS:

1. KRISHNA TEJA MEKALA - 23EG109A34
2. KSHITIJ TIWARI - 23EG109A35
3. J. SANJANA - 23EG109A25
4. TARAKA SRINIVAS MERUGA - 23EG109A41
5. V. RASMISHA - 23EG109A65

THIS DOCUMENT CONTAINS CLEAR WORKING MECHANISM FOR EACH AND EVERY FUNCTION PRESENT IN THE LIBRASYNC - LIBRARY MANAGEMENT SYSTEM.

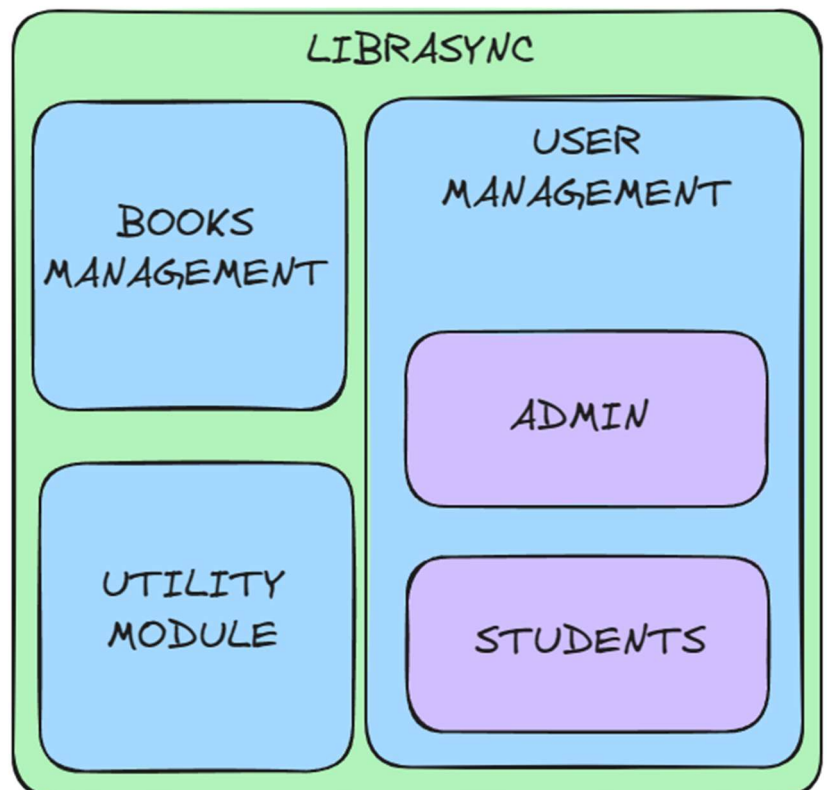
PROGRAMMING LANGUAGE: C (C99)

THE LIBRASYNC WILL HEREBY BE REFERENCED AS THE MAIN PROGRAM IN THE FOLLOWING PAGES.

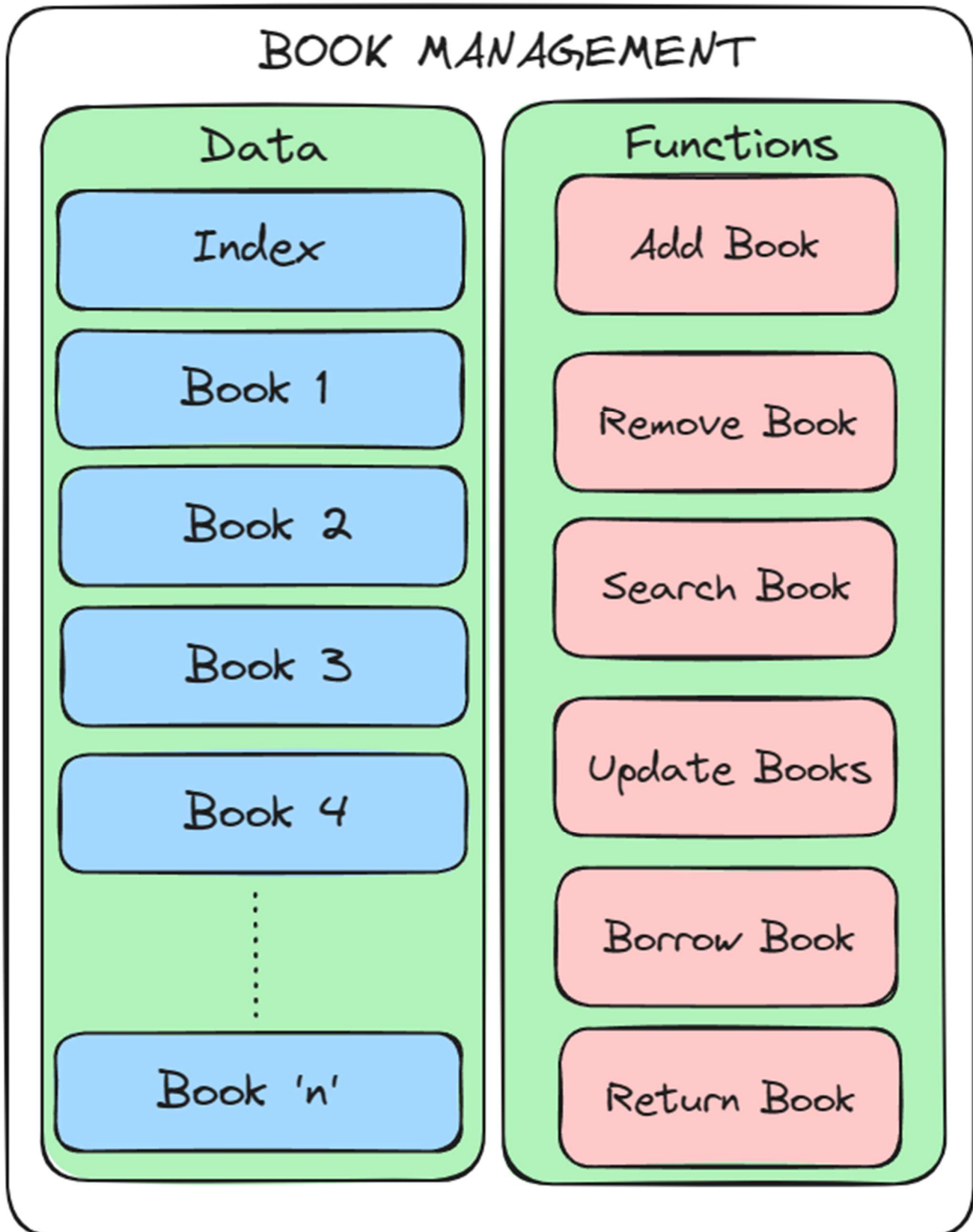
THE MAIN PROGRAM IS DIVIDED INTO 3 PARTS

1. BOOK MANAGEMENT
2. USER MANAGEMENT
3. UTILITY

EACH MODULE WILL BE COVERED IN THE FOLLOWING PAGES.



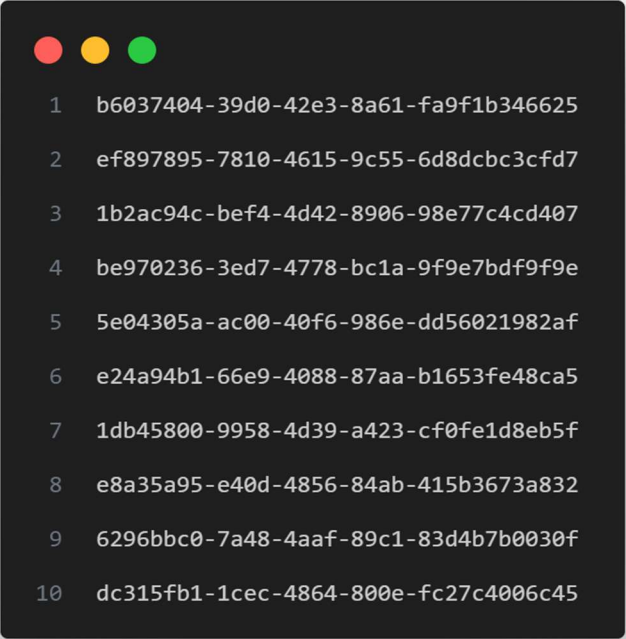
BOOK MANAGEMENT



DATA

INDEX

1. MORE PRECISELY “ BINDEX.TXT “ CONTAINS THE RELEVANT DATA ENTRY OF NUMBER OF BOOKS PRESENT IN THE MAIN PROGRAM
2. IT CONTAINS THE UID OF EACH AND EVERY BOOK.
3. THEREBY HELPS US RETRIEVE THE REFERENCE TO ALL THE BOOKS



```

1  b6037404-39d0-42e3-8a61-fa9f1b346625
2  ef897895-7810-4615-9c55-6d8dc3c3cf7
3  1b2ac94c-bef4-4d42-8906-98e77c4cd407
4  be970236-3ed7-4778-bc1a-9f9e7bdf9f9e
5  5e04305a-ac00-40f6-986e-dd56021982af
6  e24a94b1-66e9-4088-87aa-b1653fe48ca5
7  1db45800-9958-4d39-a423-cf0fe1d8eb5f
8  e8a35a95-e40d-4856-84ab-415b3673a832
9  6296bbc0-7a48-4aaf-89c1-83d4b7b0030f
10 dc315fb1-1cec-4864-800e-fc27c4006c45

```

BOOK

1. EACH BOOK FILE NAME IS AN UUID
2. EXAMPLE “B-b6037404-39d0-42e3-8a61-fa9f1b346625.txt”
3. EACH FILE WILL CONTAIN DETAILS OF THE BOOK SUCH AS
 - a) UUID
 - b) NAME
 - c) AUTHOR
 - d) GENRE
 - e) PRICE
 - f) PUBLISHED YEAR
 - g) NUMBER OF COPIES
 - h) NUMBER OF PEOPLE BORROWED
 - i) BORROWED USERS



```

1  b6037404-39d0-42e3-8a61-fa9f1b346625
2  The Great Gatsby
3  F. Scott Fitzgerald
4  Fiction
5  10.99
6  1925
7  10
8  1
9  f7e11472-aba9-4979-bd02-56f0bc9d3d63

```

VARIABLES REQUIRED

```
1 Book* totalBooks, *newSetOfBooks;
2 Index* bIndex;
3 int numberOfBooks, numberOfNewBooks;
```

FUNCTIONS

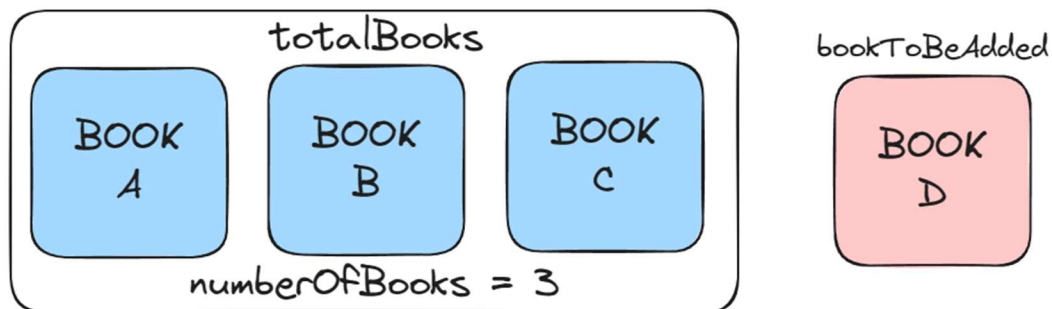
ADD BOOK

- THIS FUNCTION TAKES 1 PARAMETER, Book bookToBeAdded.

```
1 void AddBook(Book bookToBeAdded);
```

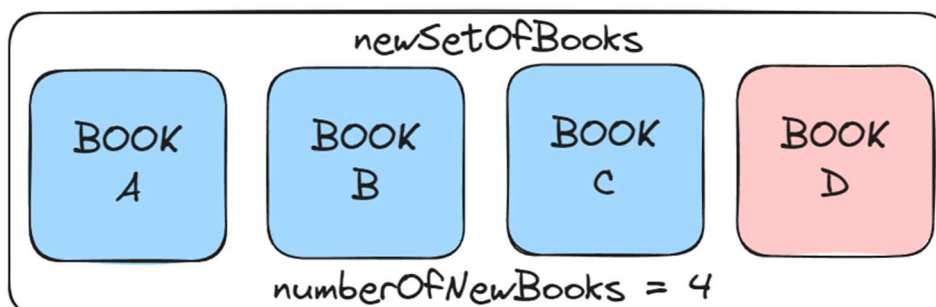
WORKING PRINCIPLE

Step - 1 :



TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE `numberOfNewBooks = numberOfBooks + 1`

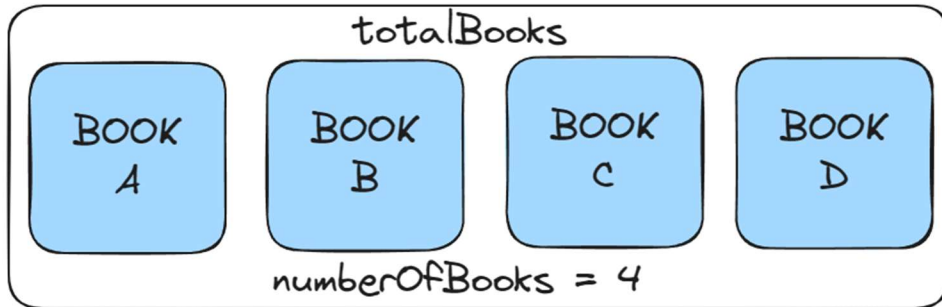
Step - 2:



STORE ALL THE BOOKS FROM `totalBooks` IN `newSetOfBooks`, AND STORE THE `bookToBeAdded` AT THE index `numberOfBooks`, INCREMENT `numberOfBooks` BY 1. ADD THE UUID TO INDEX LIST.

FREE THE `totalBooks` MEMORY SPACE AND THEN EQUATE IT TO `newSetOfBooks`.

Step - 3:



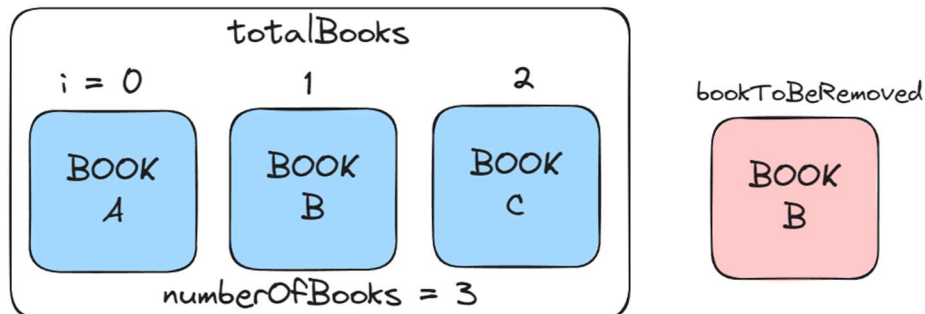
REMOVE BOOK

- THIS FUNCTION TAKES 1 PARAMETER,
Book bookToBeRemoved

```
1 void RemoveBook(Book bookToBeRemoved);
```

WORKING PRINCIPLE

Step - 1 :

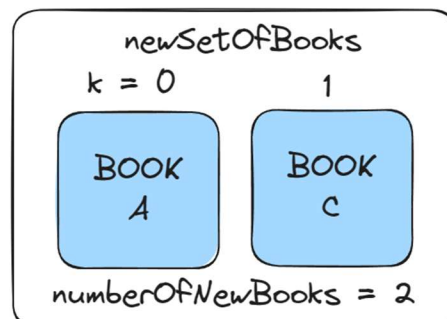


TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE `numberOfNewBooks = numberOfBooks - 1`

NOW COMPARE EACH BOOK IN `totalBooks` WITH `bookToBeRemoved`, IF THE BOOKS ARE NOT EQUAL THEN ADD IT INTO THE `newSetOfBooks`

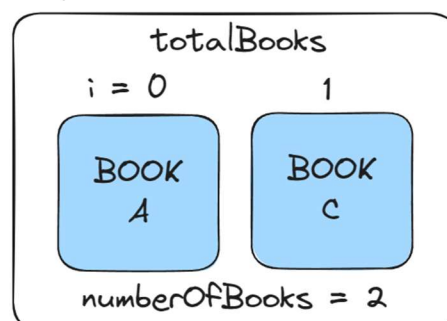
DECREMENT `numberOfbooks` BY 1. REMOVE THE UUID FROM INDEX LIST.

Step - 2 :



FREE THE `totalBooks` MEMORY SPACE AND THEN EQUATE IT TO `newSetOfBooks`.

Step - 3 :



SEARCH BOOK

- THIS FUNCTION TAKES 1 PARAMETER,
int searchType.

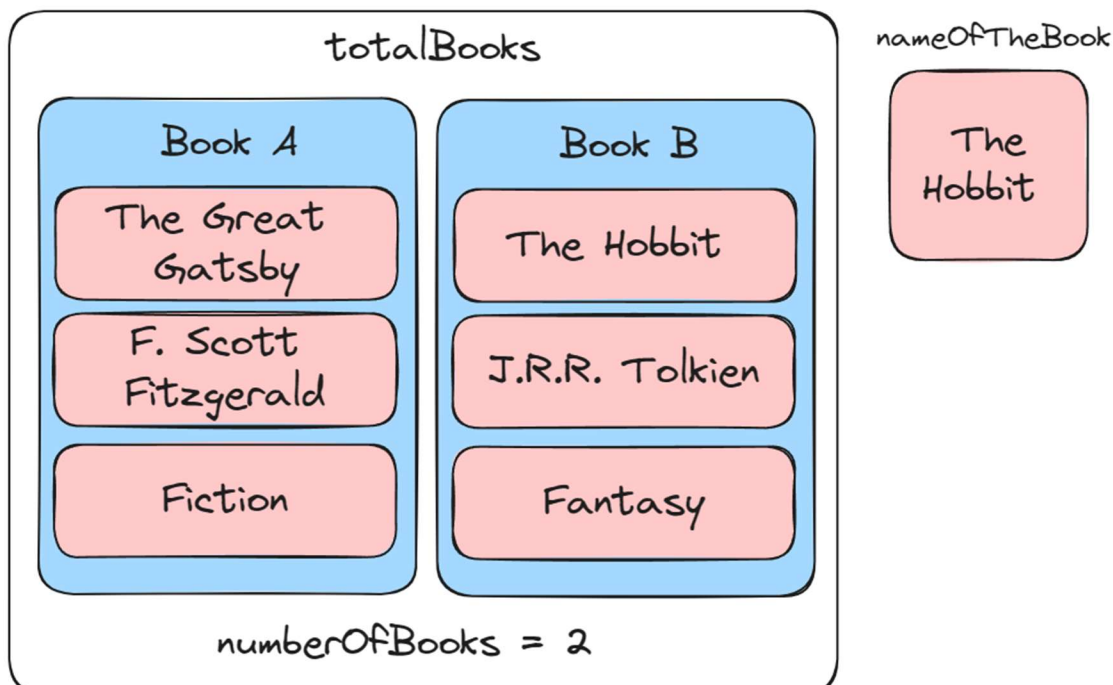
searchType acceptable values are

BYNAME, BYAUTHOR, BYGENRE (REFER
CONSTANTS.H)

```
1 void SearchBook(int searchType);
```

WORKING PRINCIPLE

INTRODUCE A SWITCH CASE ON searchType AND IMPLEMENT APPROPRIATE LOGIC FOR
SEARCHING OF BOOKS EITHER BY NAME, AUTHOR OR GENRE.



BUT HERE LIES GENERALISED WORKING PRINCIPLE FOR CHECKING.

IN THIS EXAMPLE, CHECKING BY NAME IS IMPLEMENT.

FIRST CHECK IF THE NAME OF BOOK IS EQUAL TO NAME IN EACH BOOK, IF TRUE PRINT THE
BOOKS FOUND WITH MATCHING FACTOR.

UPDATE BOOKS

STORE THE LIST OF BOOK UUIDS IN INDEX FILE
THEN STORE EACH AND EVERY BOOK TO ITS INDIVIUDAL FILE
WITH ITS UUID NAME AND PREFIX "B-"

```
1 void UpdateBooks();
```

BORROW BOOK

STORE THE UUID OF USER BORROWING IN BOOK FILE
DECREMENT NUMBER OF COPIES
INCREMENT NUMBER OF BORROWED PEOPLE

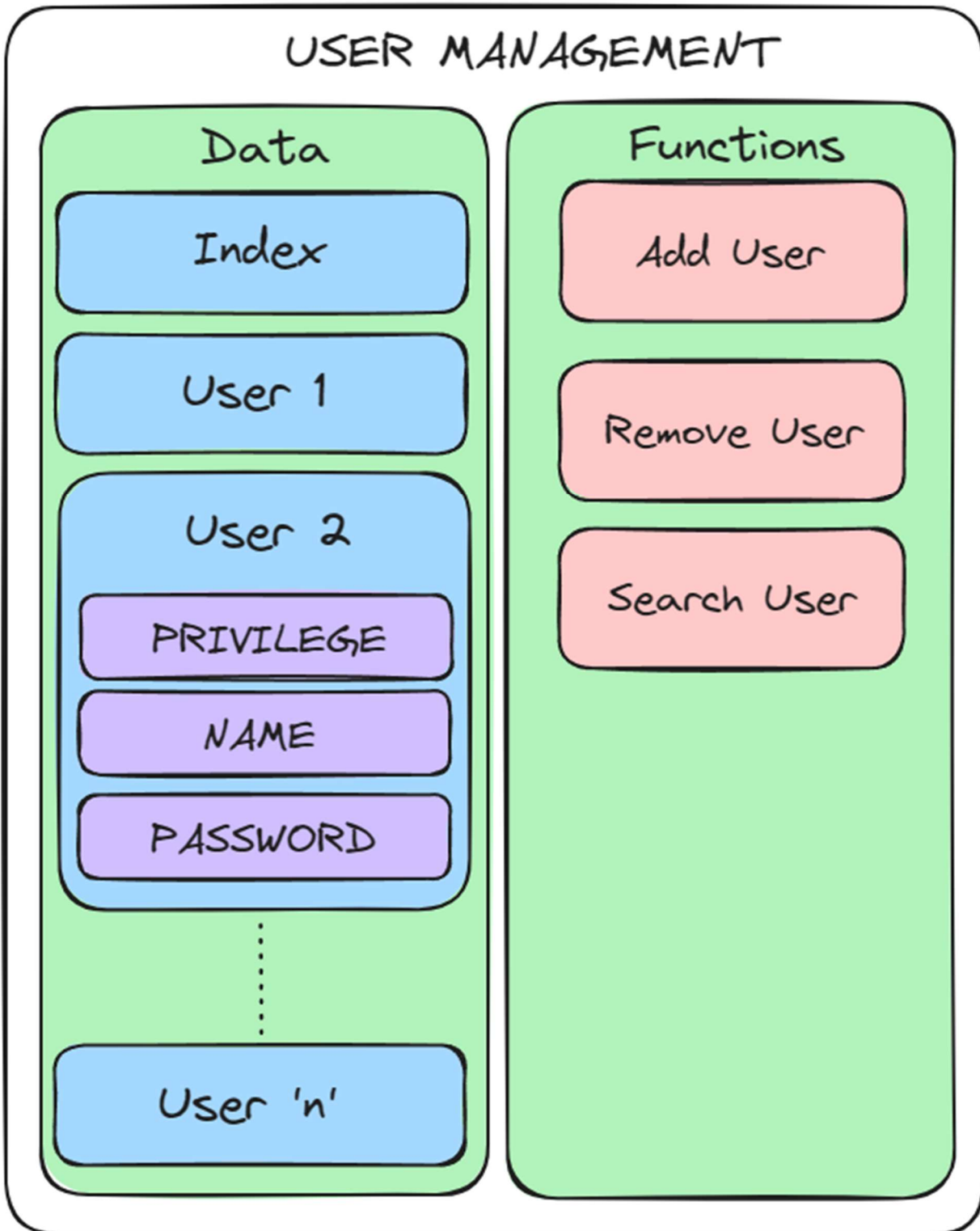
```
1 void borrowBook(Book bookToBorrow);
```

RETURN BOOK

REMOVE THE UUID OF USER RETURN IN BOOK FILE
INCREMENT NUMBER OF COPIES
DECREMENT NUMBER OF BORROWED PEOPLE

```
1 void returnBook(Book bookToReturn);
```



USER MANAGEMENT



DATA

INDEX

4. MORE PRECISELY “ UINDEX.TXT “
CONTAINS THE RELEVANT DATA
ENTRY OF NUMBER OF USERS
PRESENT IN THE MAIN PROGRAM
5. IT CONTAINS THE UID OF EACH AND
EVERY USER.
6. THEREBY HELPS US RETRIEVE THE
REFERENCE TO ALL THE USERS

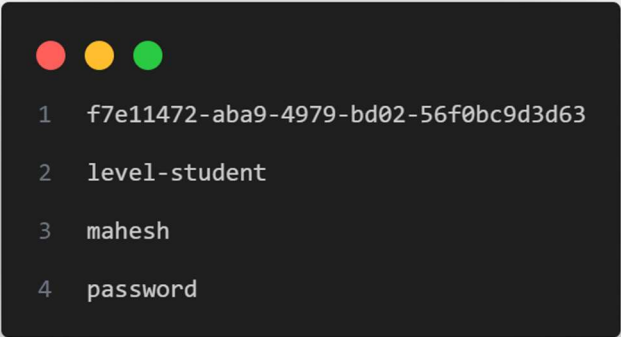


```

1  f7e11472-aba9-4979-bd02-56f0bc9d3d63
2  e57fa329-c35f-427a-8eb4-70fa24cb9ba4
3  6e43d304-8c5a-4da8-bf7f-5fcc393c4eaf
4  fcb7255f-8276-4d80-b253-8c75a32674c2
5  9976e4d0-427a-463c-92d8-08b319b90951
6  f297b17d-32e5-4b36-b240-15ddeb442f97
7  7c5e999a-788f-46d9-94b0-bd8d52275f55
8  54cdc01a-1772-4819-a633-f802518934e6
9  75f98852-89f7-4244-bdd9-9f68c3bed264
10 ea30c53e-ec80-44b2-9c89-40c0611a49ff
  
```

USER


1. EACH USER FILE NAME IS AN UUID
2. EXAMPLE “U-f7e11472-aba9-4979-bd02-56f0bc9d3d63.txt”
3. EACH FILE WILL CONTAIN DETAILS
OF THE USER SUCH AS
 - a) UUID
 - b) PRIVILEGE
 - c) NAME
 - d) PASSWORD



```

1  f7e11472-aba9-4979-bd02-56f0bc9d3d63
2  level-student
3  mahesh
4  password
  
```

VARIABLES REQUIRED



```

1  Book* totalBooks, *newSetOfBooks;
2  Index* bIndex;
3  int numberOfBooks, numberOfNewBooks;
  
```

FUNCTIONS

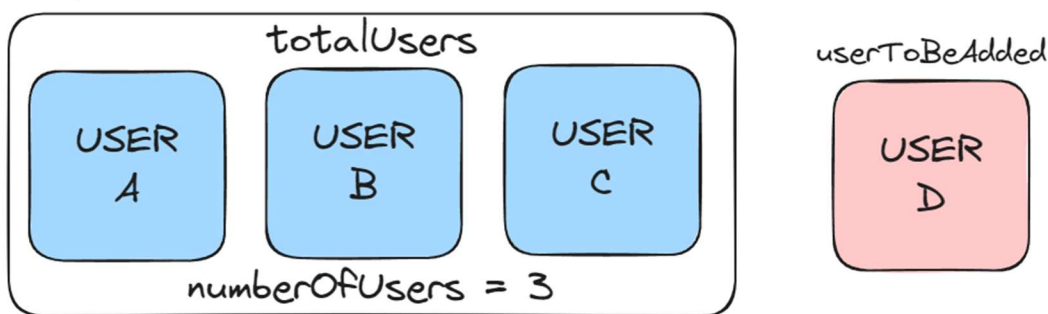
ADD USER

- THIS FUNCTION TAKES 1 PARAMETER,
User userToBeAdded.

```
1 void AddUser(User userToBeAdded);
```

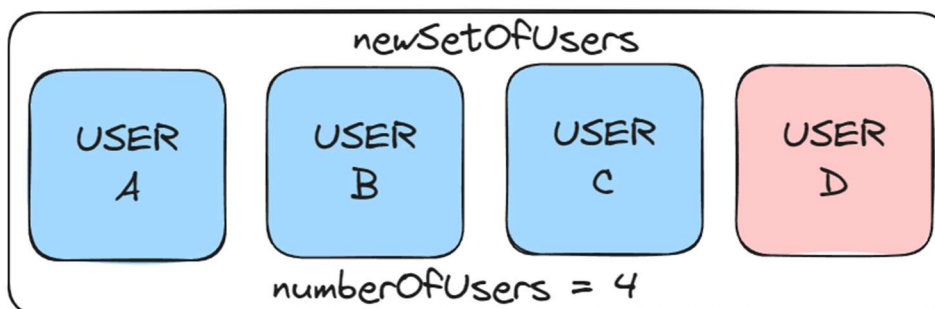
WORKING PRINCIPLE

Step - 1 :



TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE `numberOfNewUsers = numberOfUsers + 1`

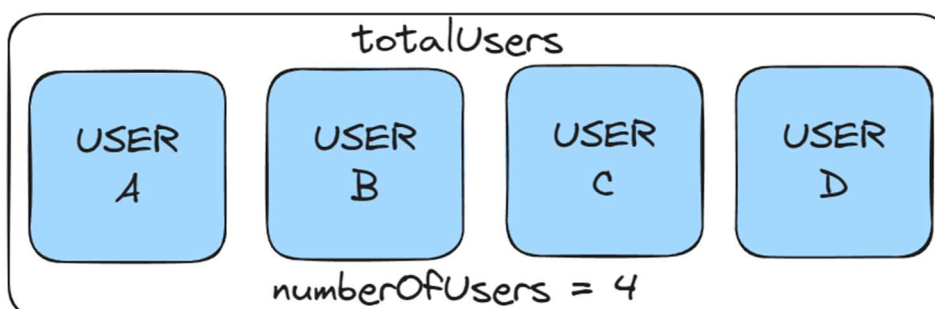
Step - 2:



STORE ALL THE USERS FROM `totalUsers` IN `newSetOfUsers`, AND STORE THE `userToBeAdded` AT THE index `numberOfUsers`, INCREMENT `numberOfUsers` BY 1. ADD THE UUID TO INDEX LIST.

FREE THE `totalUsers` MEMORY SPACE AND THEN EQUATE IT TO `newSetOfUsers`.

Step - 3:



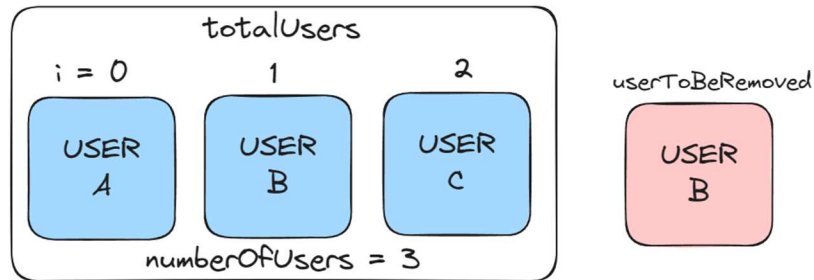
REMOVE USER

- THIS FUNCTION TAKES 1 PARAMETER,
User userToBeRemoved

```
1 void RemoveUser(User userToBeRemoved);
```

WORKING PRINCIPLE

Step - 1 :

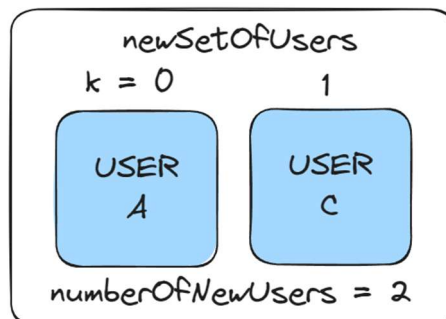


TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE `numberOfNewUsers = numberOfUsers - 1`

NOW COMPARE EACH USER IN `totalUsers` WITH `usersToBeRemoved`, IF THE USERS ARE NOT EQUAL THEN ADD IT INTO THE `newSetOfUsers`

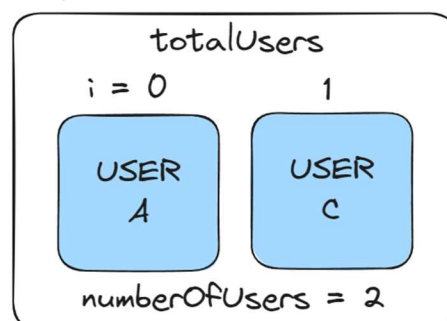
DECREMENT `numberOfUsers` BY 1. REMOVE THE UUID FROM INDEX LIST.

Step - 2 :



FREE THE `totalUsers` MEMORY SPACE AND THEN EQUATE IT TO `newSetOfUsers`.

Step - 3 :



SEARCH USER

- THIS FUNCTION TAKES 1 PARAMETER,
int searchType.

searchType acceptable values are

BYNAME, BYAUTHOR, BYGENRE (REFER
CONSTANTS.H)


WORKING PRINCIPLE

INTRODUCE A SWITCH CASE ON searchType AND IMPLEMENT APPROPRIATE LOGIC FOR
SEARCHING OF USER EITHER BY NAME OR UUID.

BUT HERE LIES GENERALISED WORKING PRINCIPLE FOR CHECKING.

IN THIS EXAMPLE, CHECKING BY NAME IS IMPLEMENT.

FIRST CHECK IF THE NAME OF USER IS EQUAL TO NAME IN EACH USER, IF TRUE PRINT THE
USER FOUND WITH MATCHING FACTOR.



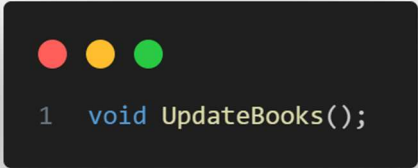
```
1 User SearchUser(User userToSearch);
```

UPDATE USER

STORE THE LIST OF USER UUIDS IN INDEX FILE

THEN STORE EACH AND EVERY USER TO ITS INDIVIDUAL FILE

WITH ITS UUID NAME AND PREFIX "U-"



```
1 void UpdateBooks();
```