# LIBRASYNC DOCUMENTATION

# INDEX

# LIBRASYNC DOCUMENTATION

VERSION: 1.0 25-04-2024

GITHUB: https://github.com/krishna18developer/CampusConnect/tree/main/LibraSync

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.

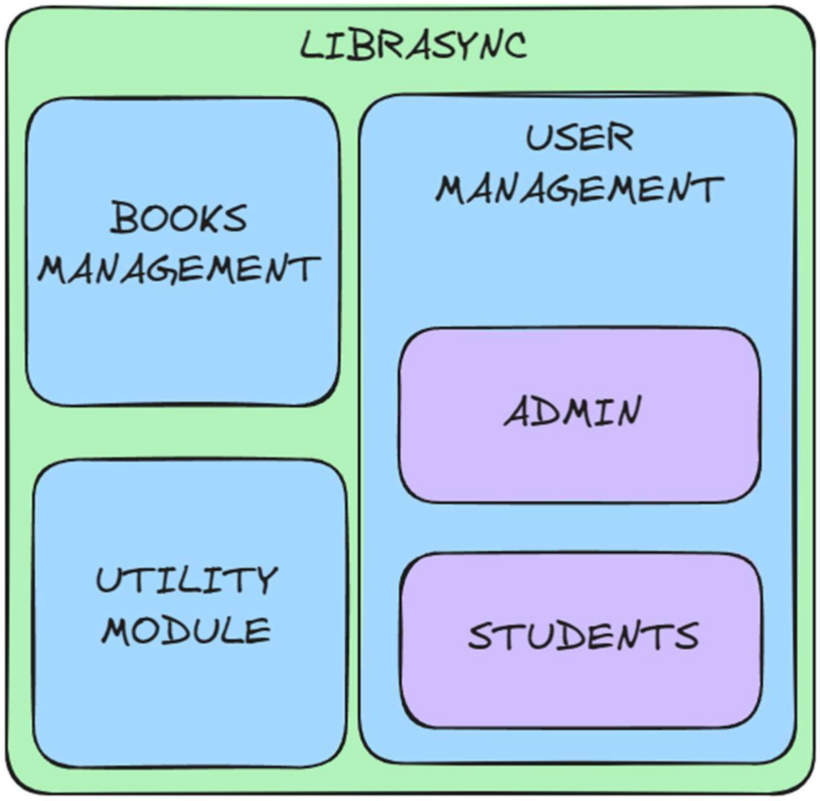
PROGRAMMING LANGUAGE: C (C99)

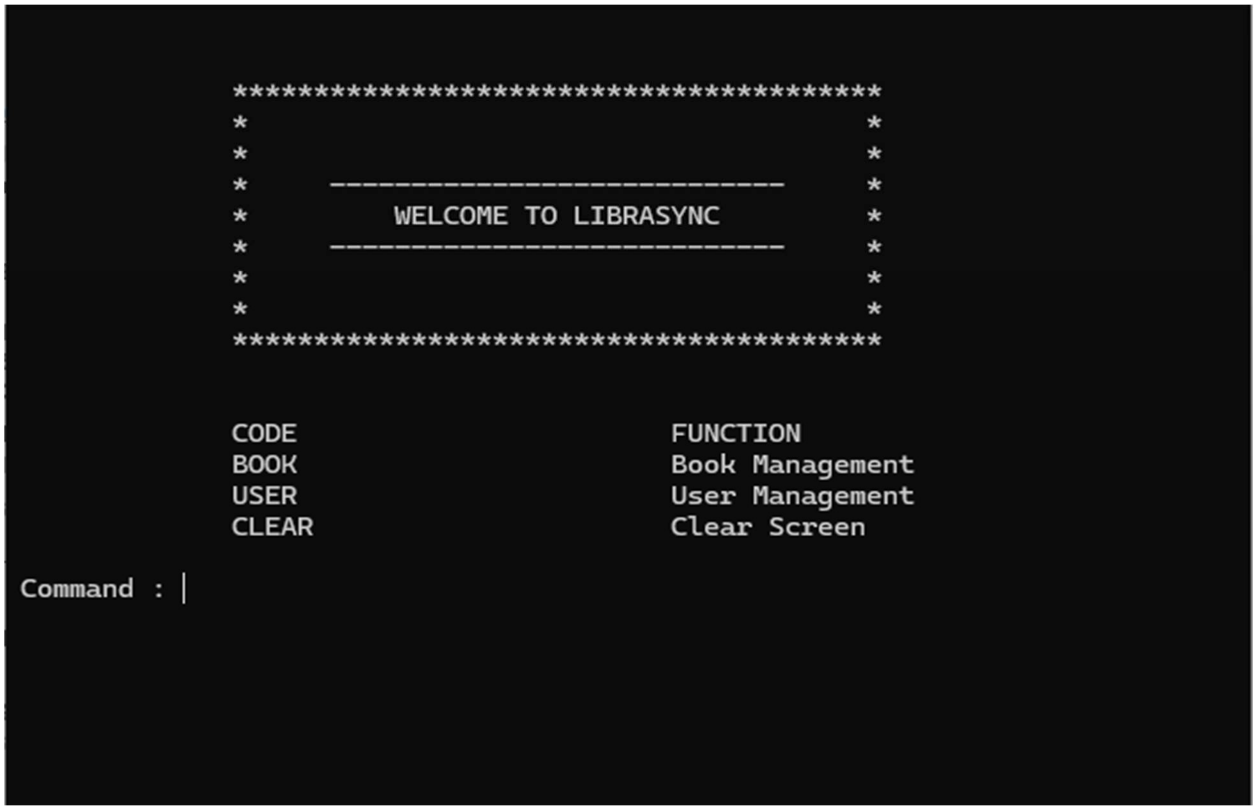
<u>THE LIBRASYNC WILL HEREBY BE REFERENCED AS THE MAIN PROGRAM IN THE FOLLOWING PAGES.</u>

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.

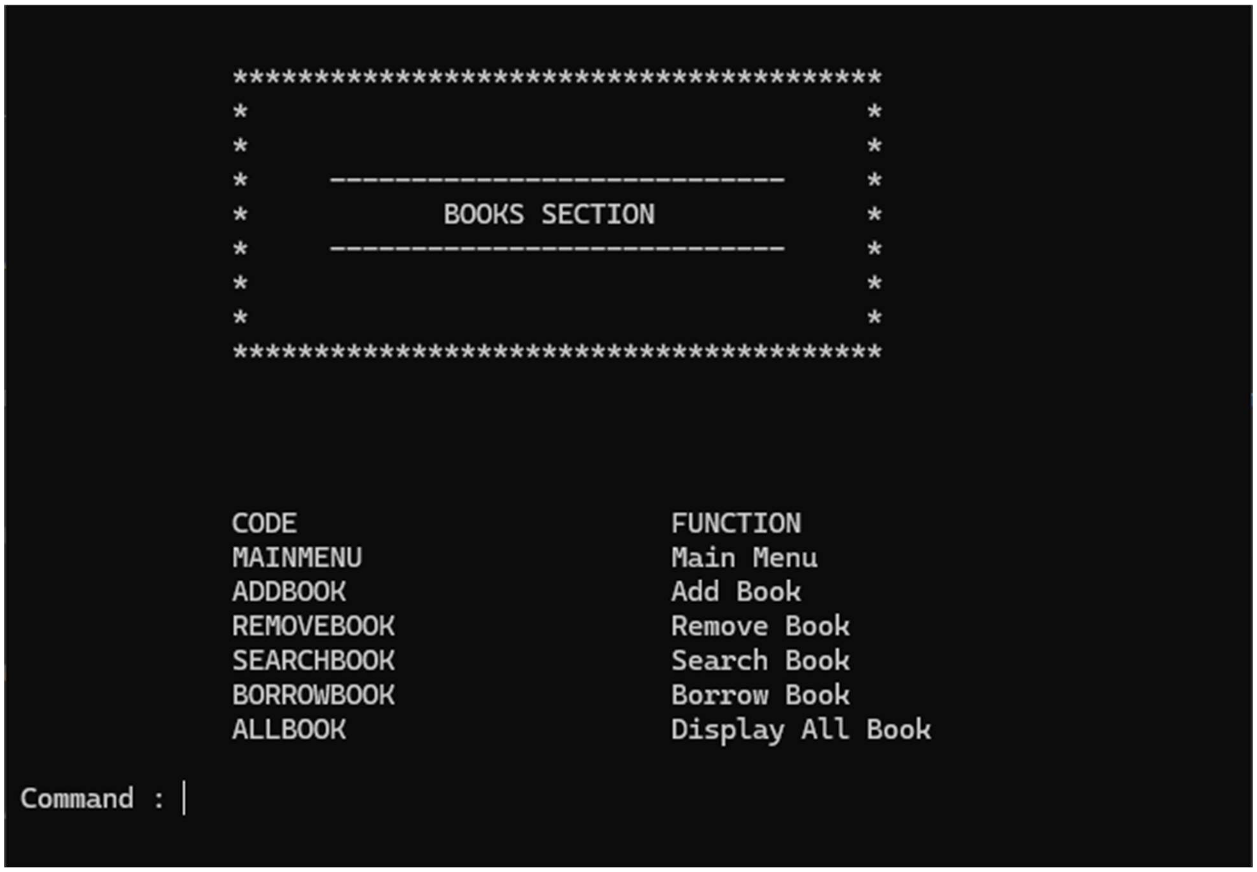




```
          ****************************************
          *                                      *
          *                                      *
          *     ----------------------------     *
          *         WELCOME TO LIBRASYNC         *
          *     ----------------------------     *
          *                                      *
          *                                      *
          ****************************************


          CODE                        FUNCTION
          BOOK                        Book Management
          USER                        User Management
          CLEAR                       Clear Screen

Command : |
```



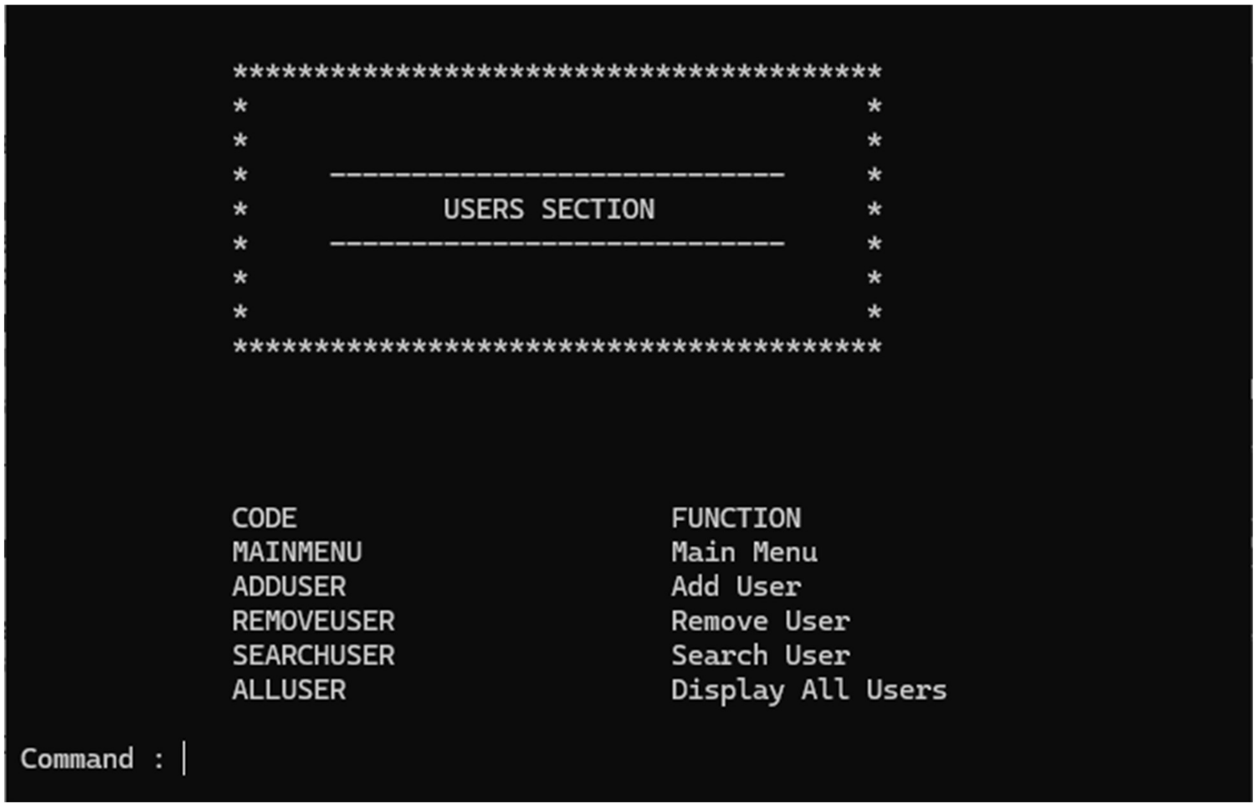
```
          ****************************************
          *                                      *
          *                                      *
          *     ----------------------------     *
          *            BOOKS SECTION             *
          *     ----------------------------     *
          *                                      *
          *                                      *
          ****************************************



          CODE                        FUNCTION
          MAINMENU                    Main Menu
          ADDBOOK                     Add Book
          REMOVEBOOK                  Remove Book
          SEARCHBOOK                  Search Book
          BORROWBOOK                  Borrow Book
          ALLBOOK                     Display All Book

Command : |
```

-

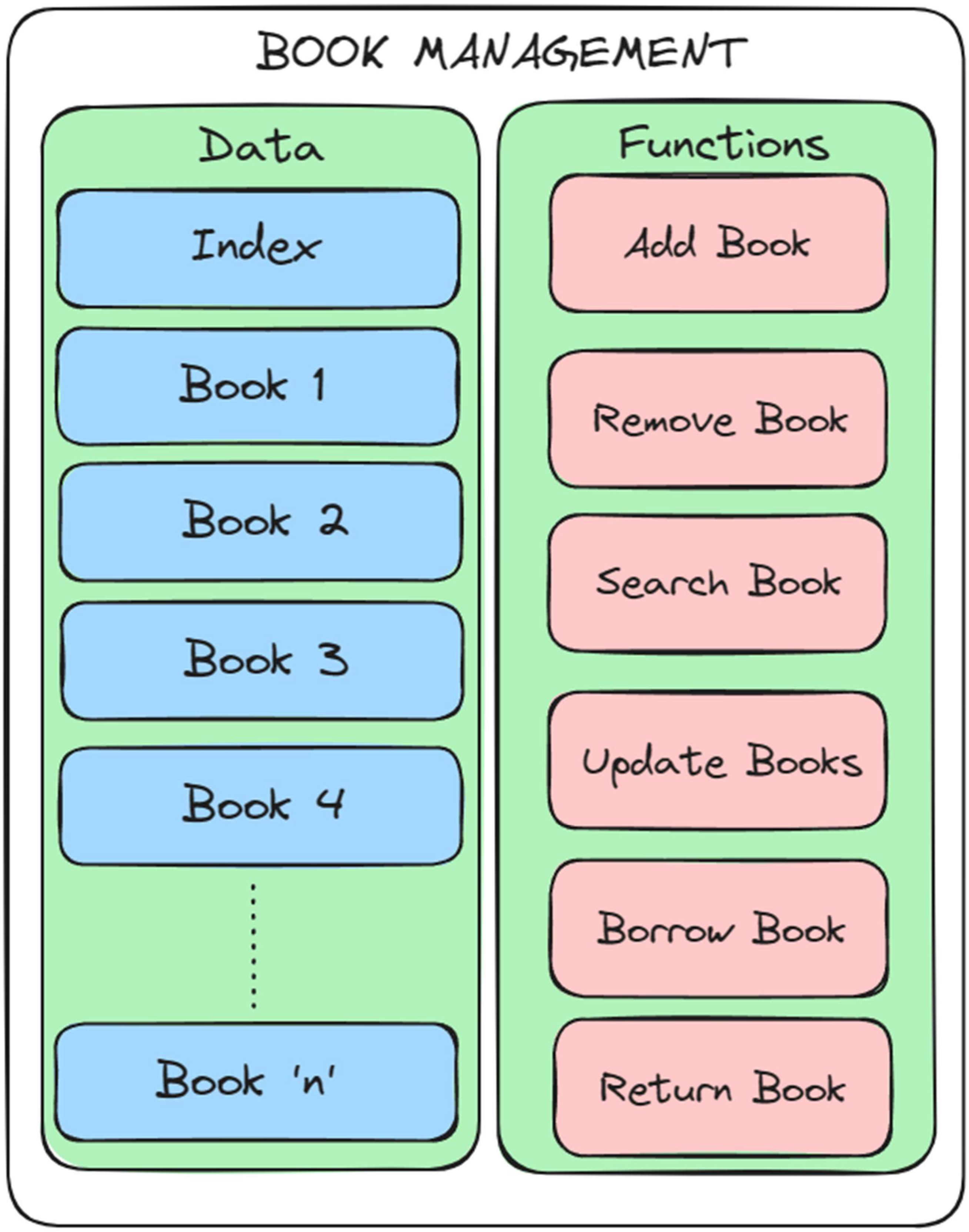


```
          ****************************************
          *                                      *
          *                                      *
          *     ----------------------------     *
          *            USERS SECTION             *
          *     ----------------------------     *
          *                                      *
          *                                      *
          ****************************************



          CODE                       FUNCTION
          MAINMENU                   Main Menu
          ADDUSER                    Add User
          REMOVEUSER                 Remove User
          SEARCHUSER                 Search User
          ALLUSER                    Display All Users
Command : |
```

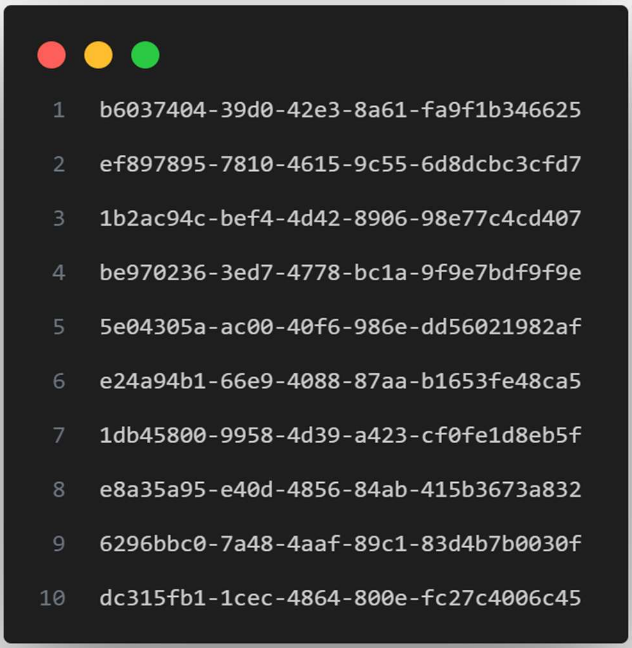
## BOOK MANAGEMENT



## DATA

### INDEX

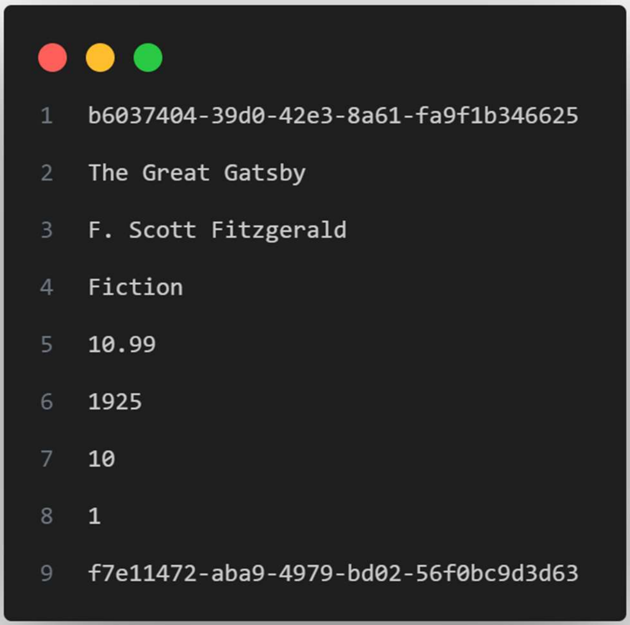
1. MORE PRECISELY " BINDEX.TXT " CONTAINS THE RELEVANT DATA ENTRY OF NUMBER OF BOOKS PRESENT IN THE MAIN PROGRAM
2. IT CONTAINS THE UUID OF EACH AND EVERY BOOK.
3. THEREBY HELPS US RETRIEVE THE REFERENCE TO ALL THE BOOKS



```
b6037404-39d0-42e3-8a61-fa9f1b346625
ef897895-7810-4615-9c55-6d8dcbc3cfd7
1b2ac94c-bef4-4d42-8906-98e77c4cd407
be970236-3ed7-4778-bc1a-9f9e7bdf9f9e
5e04305a-ac00-40f6-986e-dd56021982af
e24a94b1-66e9-4088-87aa-b1653fe48ca5
1db45800-9958-4d39-a423-cf0fe1d8eb5f
e8a35a95-e40d-4856-84ab-415b3673a832
6296bbc0-7a48-4aaf-89c1-83d4b7b0030f
dc315fb1-1cec-4864-800e-fc27c4006c45
```

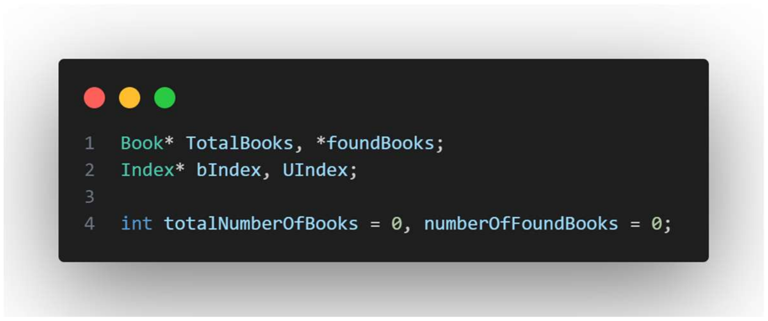
### BOOK

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



```
b6037404-39d0-42e3-8a61-fa9f1b346625
The Great Gatsby
F. Scott Fitzgerald
Fiction
10.99
1925
10
1
f7e11472-aba9-4979-bd02-56f0bc9d3d63
```

## VARIABLES REQUIRED



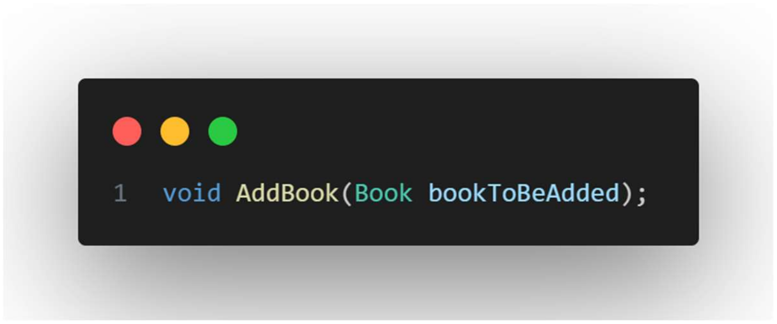
```cpp
Book* TotalBooks, *foundBooks;
Index* bIndex, UIndex;

int totalNumberOfBooks = 0, numberOfFoundBooks = 0;
```

## FUNCTIONS

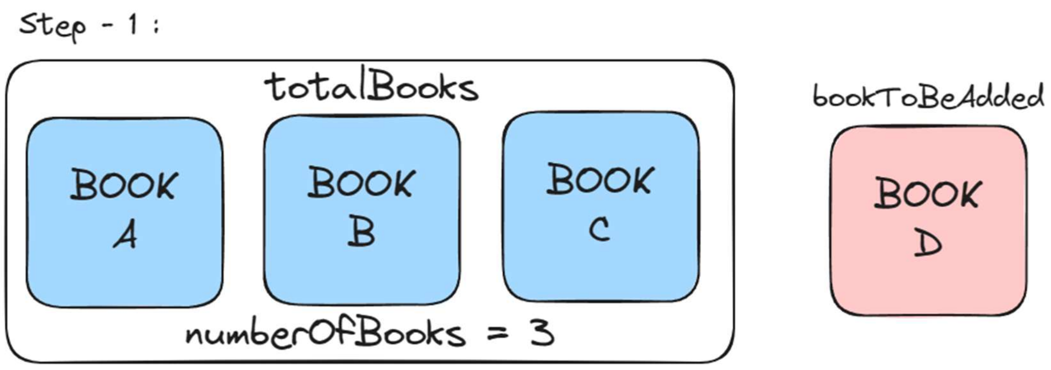
### ADD BOOK

- THIS FUNCTION TAKES 1 PARAMETER, Book bookToBeAdded.

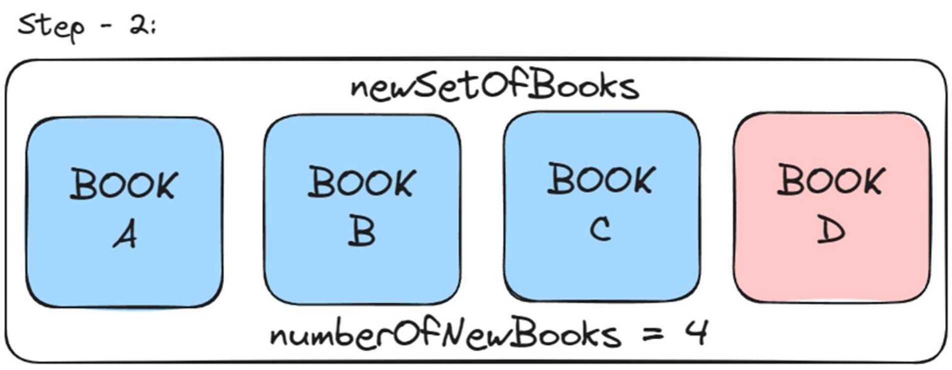


```cpp
void AddBook(Book bookToBeAdded);
```

### WORKING PRINCIPLE

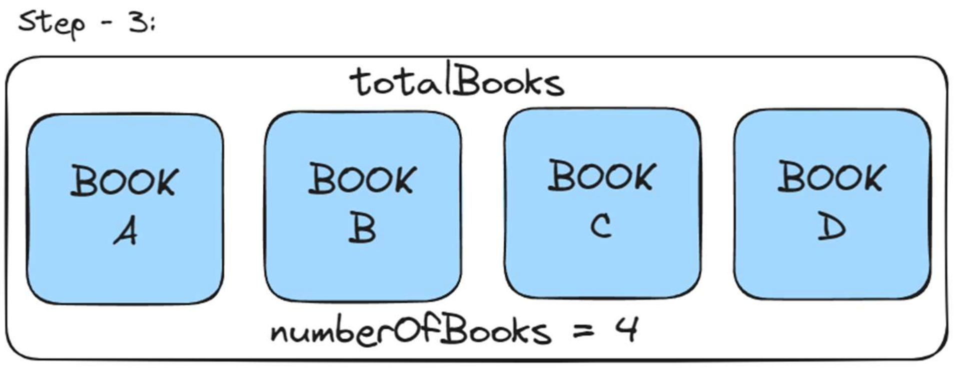


TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE numberOfNewBooks = numberOfBooks + 1



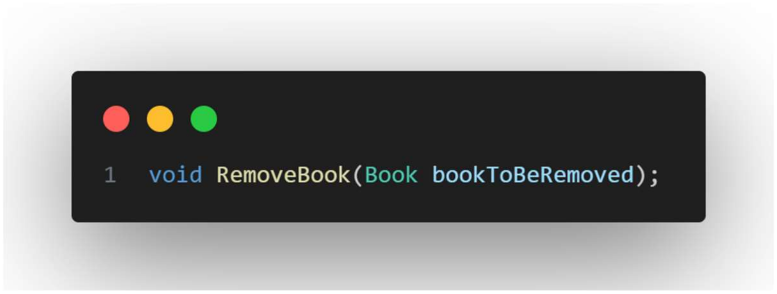
STORE ALL THE BOOKS FROM totalBooks IN newSetOfBooks, AND STORE THE bookToBeAdded AT THE index numberOfBooks, INCREMENT numberOfbooks BY 1. ADD THE UUID TO INDEX LIST.

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



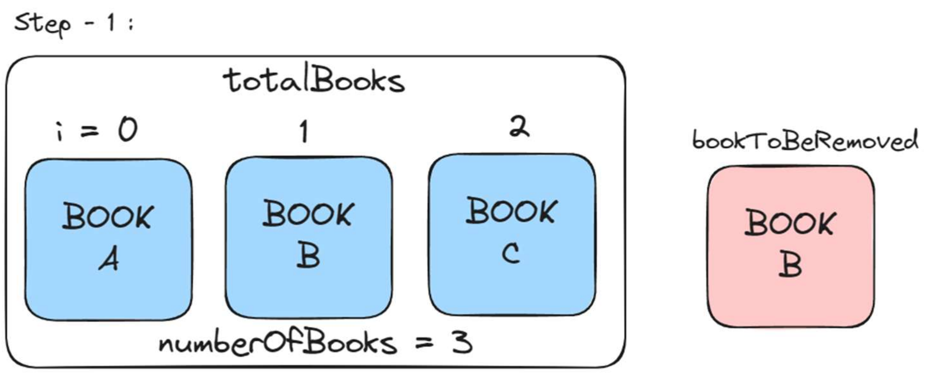
## REMOVE BOOK

- THIS FUNCTION TAKES 1 PARAMETER, Book bookTobeRemoved



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

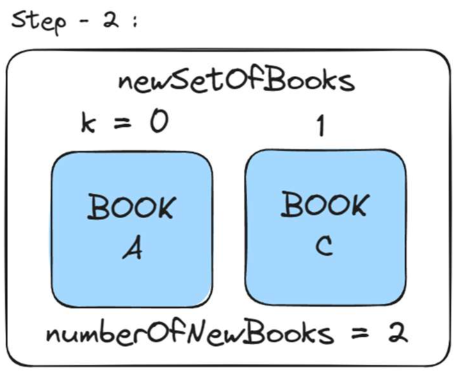
## WORKING PRINCIPLE



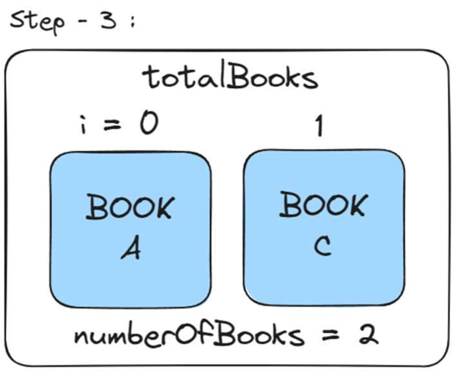
TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE numberOfNewBooks = numberOfBooks - 1

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.

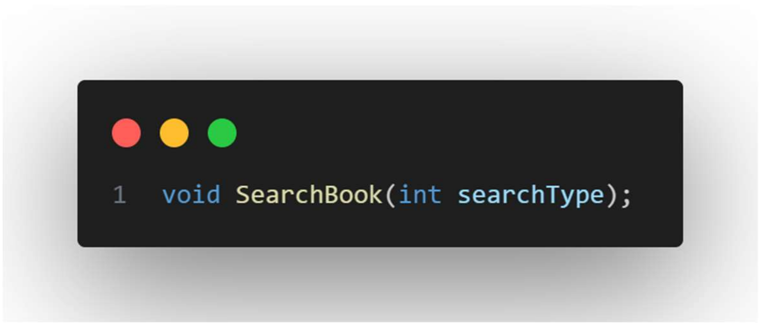


FREE THE totalBooks MEMORY SPACE AND THEN EQUATE IT TO newSetOfBooks.



## SEARCH BOOK

- THIS FUNCTION TAKES 1 PARAMETER, int searchType.



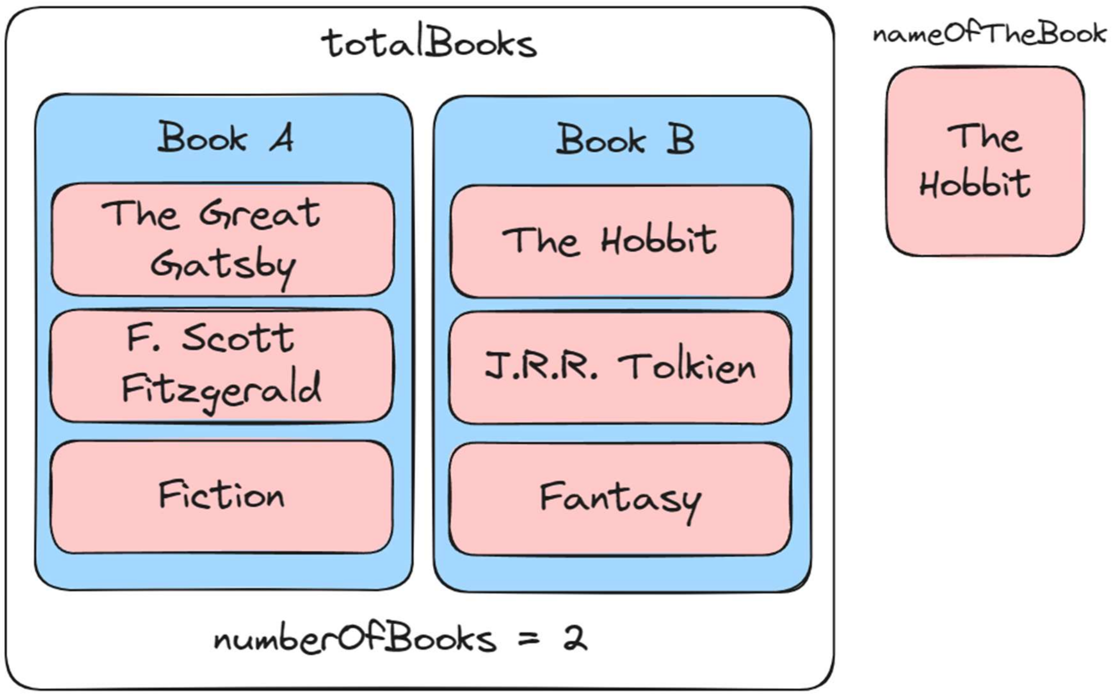
```c
void SearchBook(int searchType);
```

searchType accepetable values are

BYNAME, BYAUTHOR, BYGENRE ( REFER CONSTANTS.H )

## 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.

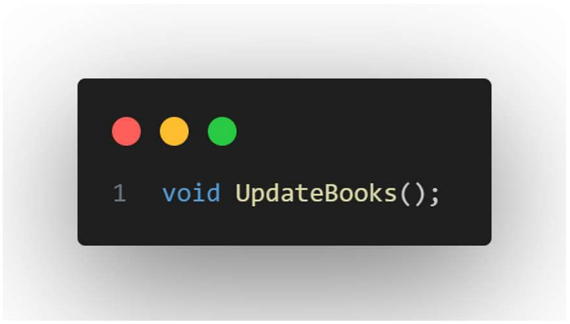
FIRST CHECK IF THE NAME OF BOOK IS EQUAL TO NAME IN EACH BOOK, IF TRUE PRINT THE BOOKS FOUND WITH MATCHING FACTOR.

## 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-“



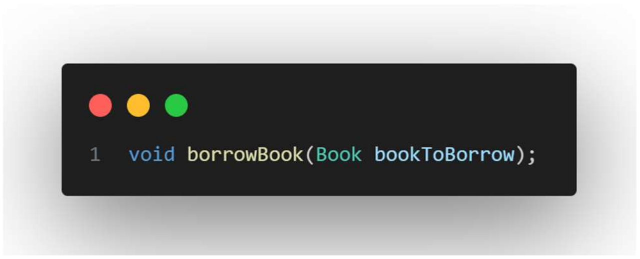
```
void UpdateBooks();
```

## BORROW BOOK

STORE THE UUID OF USER BORROWING IN BOOK FILE

DECREMENT NUMBER OF COPIES

INCREMENT NUMBER OF BORROWED PEOPLE



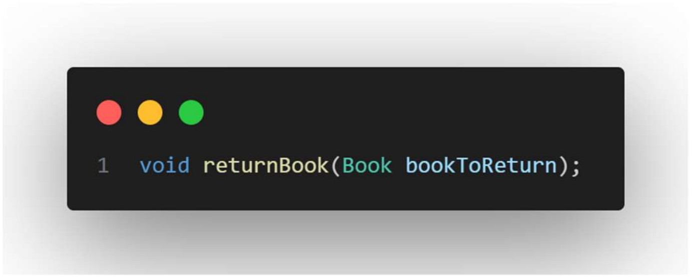
```
void borrowBook(Book bookToBorrow);
```

## RETURN BOOK

REMOVE THE UUID OF USER RETURN IN BOOK FILE

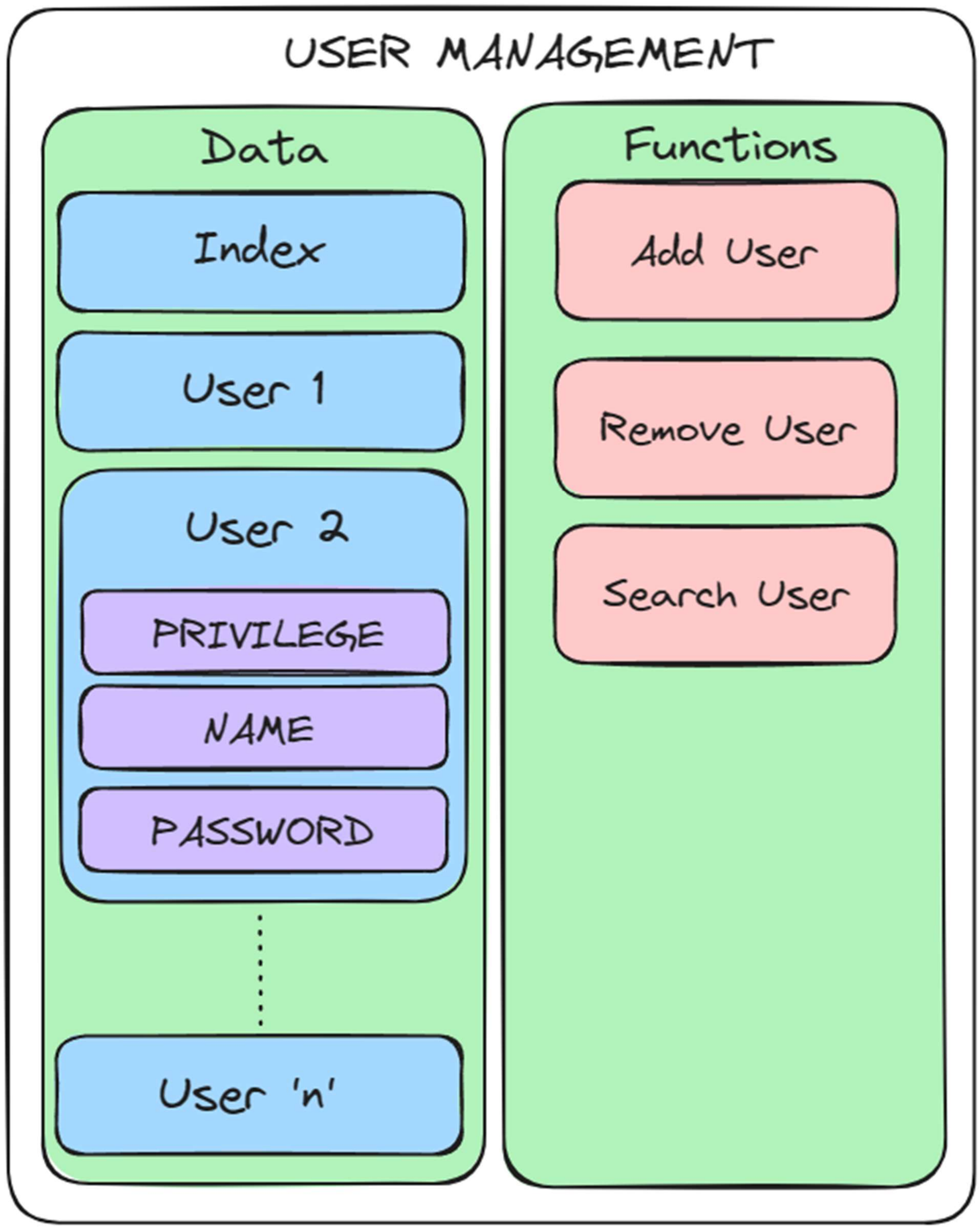
INCREMENT NUMBER OF COPIES

DECREMENT NUMBER OF BORROWED PEOPLE



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

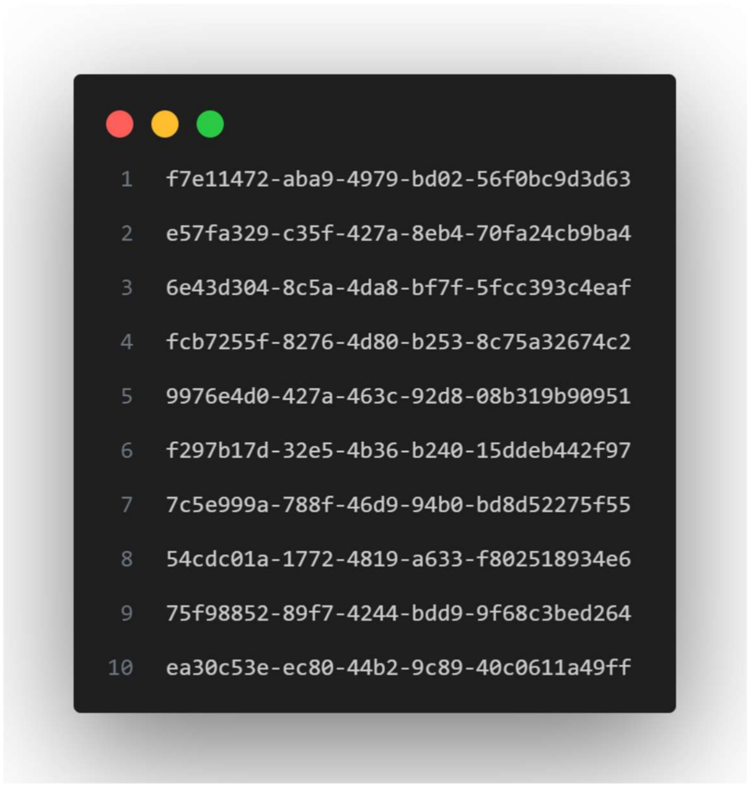
## USER MANAGEMENT



## DATA

### INDEX

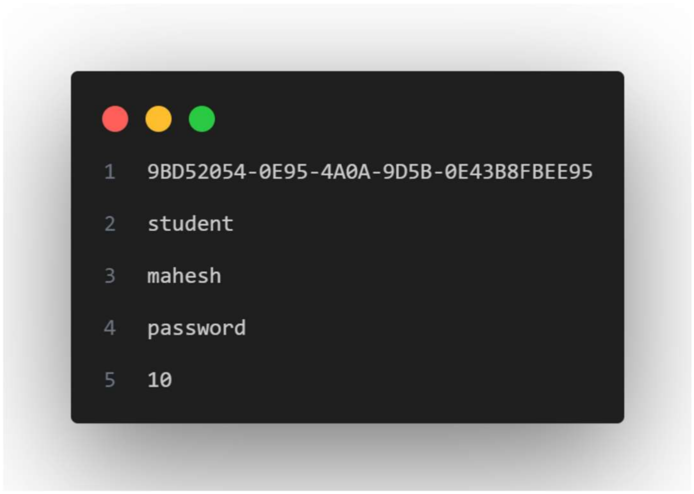
4. MORE PRECISELY " UINDEX.TXT " CONTAINS THE RELEVANT DATA ENTRY OF NUMBER OF USERS PRESENT IN THE MAIN PROGRAM
5. IT CONTAINS THE UUID OF EACH AND EVERY USER.
6. THEREBY HELPS US RETRIEVE THE REFERENCE TO ALL THE USERS



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

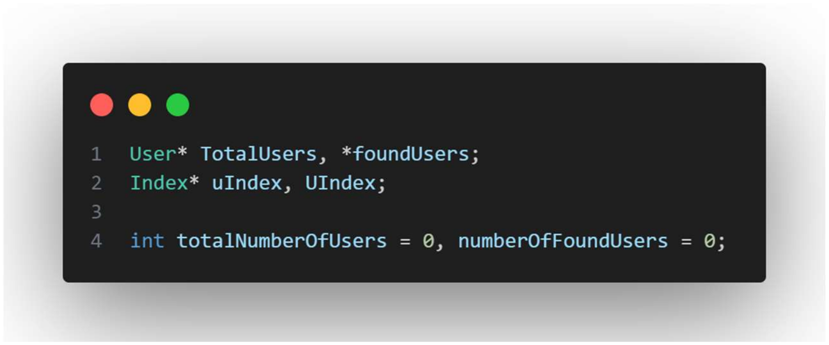
### USER

1. EACH USER FILE NAME IS AN UUID
2. EXAMPLE "U-9BD52054-0E95-4A0A-9D5B-0E43B8FBEE95.txt"
3. EACH FILE WILL CONTAIN DETAILS OF THE USER SUCH AS
   a) UUID
   b) PRIVILEGE
   c) NAME
   d) PASSWORD
   e) Roll Number



```
9BD52054-0E95-4A0A-9D5B-0E43B8FBEE95
student
mahesh
password
10
```

## VARIABLES REQUIRED



```
User* TotalUsers, *foundUsers;
Index* uIndex, UIndex;

int totalNumberOfUsers = 0, numberOfFoundUsers = 0;
```

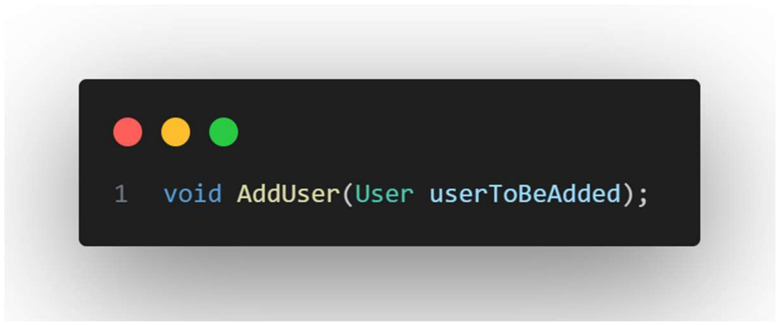
PRIVILEGE

1. admin
2. student

## FUNCTIONS

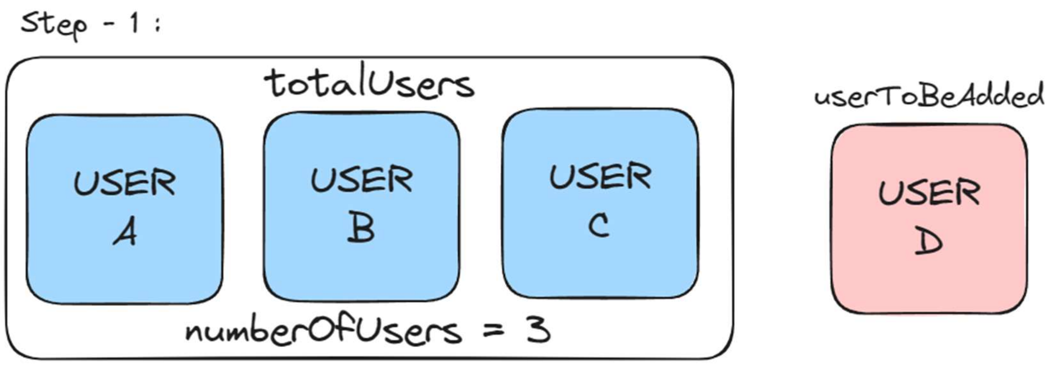
### ADD USER

- THIS FUNCTION TAKES 1 PARAMETER, User userToBeAdded.

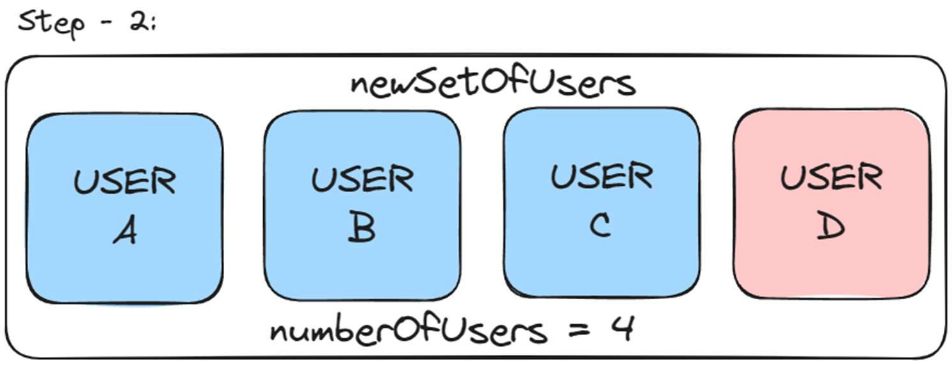


```
void AddUser(User userToBeAdded);
```

### WORKING PRINCIPLE

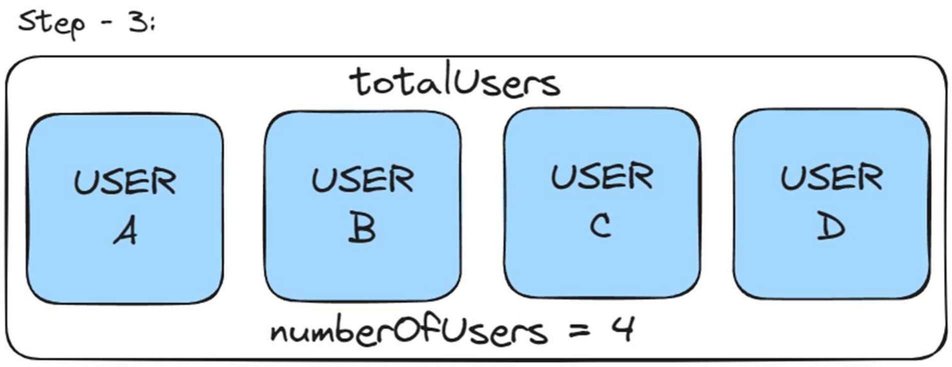


TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE numberOfNewUsers = numberOfUsers + 1



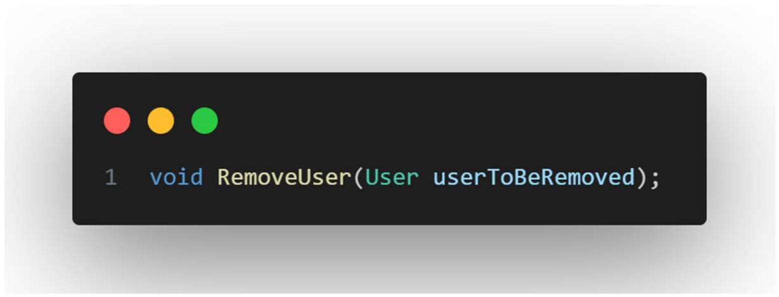
STORE ALL THE USERS FROM totalUsers IN newSetOfUsers, AND STORE THE userToBeAdded AT THE index numberOfUsers, INCREMENT numberOfUsers BY 1. ADD THE UUID TO INDEX LIST.

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



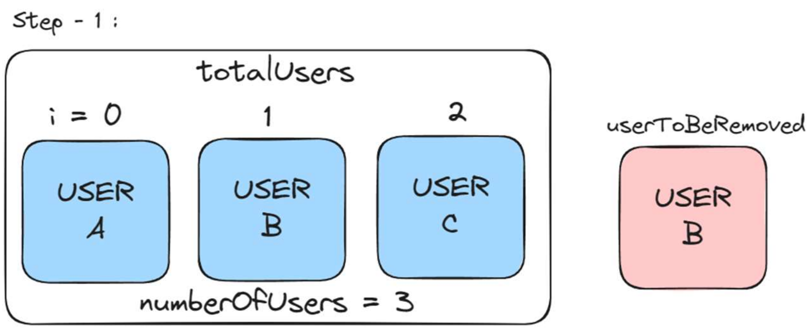
## REMOVE USER

- THIS FUNCTION TAKES 1 PARAMETER, User userTobeRemoved



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

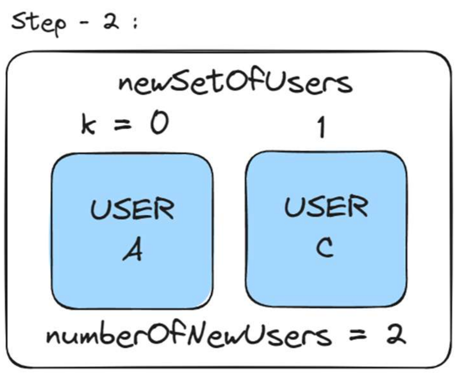
## WORKING PRINCIPLE



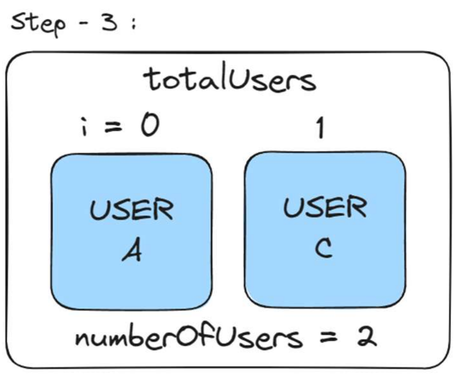
TAKE A NEW DYNAMICALLY ALLOCATED ARRAY OF SIZE numberOfNewUsers = numberOfUsers - 1

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

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

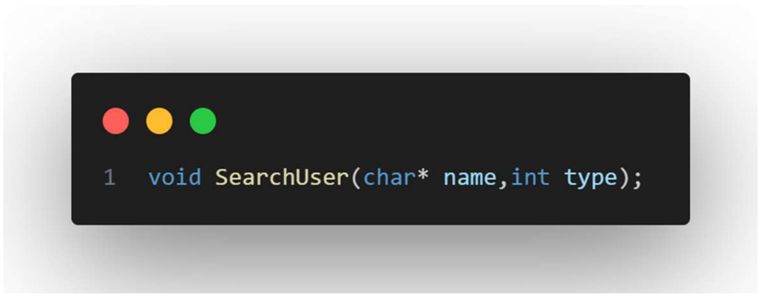


FREE THE totalUsers MEMORY SPACE AND THEN EQUATE IT TO newSetOUsers.



<u>SEARCH USER</u>

- THIS FUNCTION TAKES 2 PARAMETERS, String name, int searchType.



```
void SearchUser(char* name,int type);
```

searchType accepetable values are

BYNAME, BYROLLNUMBER, BYPRIVILEGELEVEL

( REFER CONSTANTS.H )

<u>WORKING PRINCIPLE</u>

INTRODUCE A SWITCH CASE ON searchType AND IMPLEMENT APPROPRIATE LOGIC FOR SEARCHING OF USER EITHER BY NAME, BY ROLL NUMBER OR BY PRIVILEGE LEVEL.

BUT HERE LIES GENERALISED WORKING PRINCIPLE FOR CHECKING.

IN THIS EXAMPLE, CHECKING BY NAME IS IMPLEMENT.

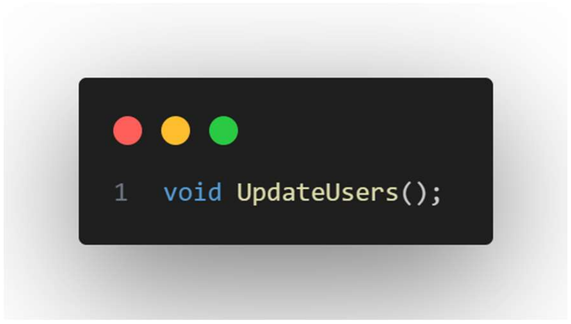
FIRST CHECK IF THE NAME OF USER IS EQUAL TO NAME IN EACH USER, IF TRUE PRINT THE USER FOUND WITH MATCHING FACTOR.

<u>UPDATE USER</u>

STORE THE LIST OF USER UUIDS IN INDEX FILE

THEN STORE EACH AND EVERY USER TO ITS INDIVIUDAL FILE

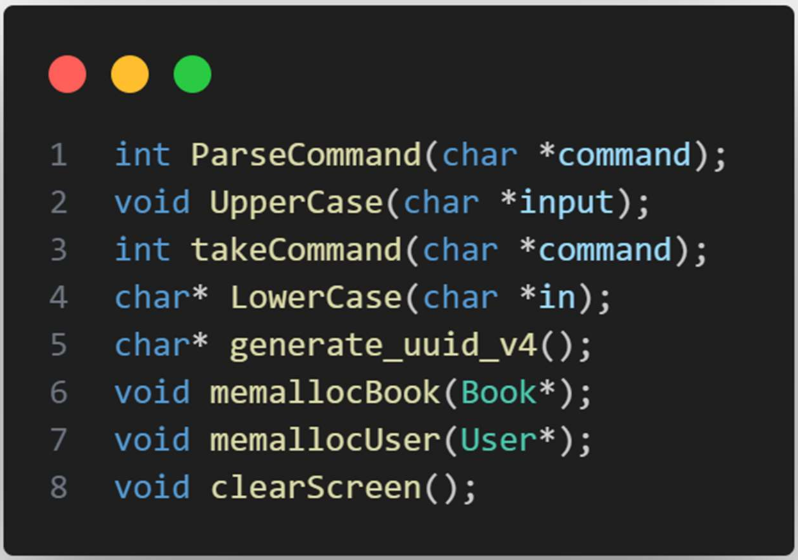
WITH ITS UUID NAME AND PREFIX “U-“ IS ADDED TO THE FILE NAME.



```
void UpdateUsers();
```

## UTILITY

THE LIST BESIDE SHOW THE FUNCTIONS PRESENT IN THE UTILITY MODULE WITH THEIR QUICK DESCRIPTION.



```c
int ParseCommand(char *command);
void UpperCase(char *input);
int takeCommand(char *command);
char* LowerCase(char *in);
char* generate_uuid_v4();
void memallocBook(Book*);
void memallocUser(User*);
void clearScreen();
```

1) ParseCommand – Takes the Command From The user and converts into a format the program understands and is applicable throughout the main program. This is mainly used in building menu systems in this program.
2) UpperCase – This Function converts the given string into uppercase, mainly used when checking two strings in Searching Functions.
3) LowerCase – This Function converts the given string into lowercase, mainly used when checking two strings in Searching Functions.
4) takeCommand – This function is directly interactive with the user and returns the required constant for the program's command system to work.
5) generate_uuid_v4 – This function is used to genereate UUID.
6) memallocBook – This function is used to allocate memory for Book Structure
7) memallocUser – This function is used to allocate memory for User Structure
8) clearScreen – This function is used to clear the screen.

**<u>THANK</u>**

**<u>YOU !</u>**