GROUP-1 CSE

MEMBERS:

N AKHILA 23P81A0502 CSE – A

J BABITHA 23P81A0523 CSE - A

S ISHANTH REDDY 23P81A0552 CSE - A

K HEMASHASHEENDRA 23P81A0599 CSE - B

LEADER: K RUSHI 23P81A0528 CSE - A

LIBRARY BOOKS

SOFTWARE USED:

* DEV C++
* C COMPILER

PROJECT STATEMENT:

Save details of at least 10 books in a library in a binary file.

REQUIREMENTS:

Category Wise list.

Category Wise total and average book cost

For a given book number, give its details.

**PROCEDURAL PROGRAMMING:**

* + void saveBookDetails();
  + void displayCategoryWiseList();
  + void displayCategoryWiseTotalAndAverage();
  + void displayBookDetails();
  + void saveBookDetailsToFile();
  + int loadBookDetailsFromFile();
  + void deleteBook();
  + void clearAllBooks()
  + int main().

**ALGORITHM:**

**Step-1:**Start.

**Step-2:**Include necessary header files: stdlib.h, string.h, and stdio.h.

**Step-3:**Define constants and macros: MAX\_BOOKS for the maximum number of books and FILENAME for the binary file name.

**Step-4:**Declare an enumeration enum Category to represent book categories: Fictional, Physics, and History.

**Step-5:**Define a structure struct Book to store book details: book number, title, author, number of pages, category, and cost.

**Step-6:**Declare the following function prototypes :

void saveBookDetails(struct Book \*books, int count);

void displayCategoryWiseList(struct Book \*books, int count);

void displayCategoryWiseTotalAndAverage(struct Book \*books, int count);

void displayBookDetails(struct Book \*books, int count, int bookNumber);

void saveBookDetailsToFile(struct Book \*books, int count);

int loadBookDetailsFromFile(struct Book \*books);

void deleteBook(struct Book \*books, int \*count, int bookNumber);

void clearAllBooks(struct Book \*books, int \*count);

**Step-7:**Dynamically allocate memory for an array of struct Book to store book details (books).

**Step-8:**Declare bookCount , choice and initialize to bookCount=0.

**Step-9:**Display a welcome message and the main menu.

**Step-10:**Assign bookCount = loadBookDetailsFromFile(books), call the function and pass parameters, to check number of books

**Step-11:**Input choice.

**Step-12:**If user choice =1,Goto step (a).Else goto step 13.

**(a):**Call the function --- saveBookDetails(struct Book \*books, int count) and pass the parameters books, bookCount.

**(b):**If count<MAX\_BOOKS, Goto step (c), else goto step (l).

**(c):**Input book number.Goto step (d).

**(d):**Validate input for book details.Goto step (e).

**(e):**declare and initiate i to zero.

**(f):**Set a for loop: If i<count, goto step (g). Else goto step (j).

**(g):**If books[i].bookNumber == bookNumber.Goto step (h).Else goto step (i).

**(h):**Display book already exists. Goto step (c).

**(i):**Increment i. Goto step (f).

**(j):**Set books[count].bookNumber = bookNumber.

**(k):**Input book title, author, pages, category ,cost. Goto step (m).

**(l):**Display maximum limit reached and break out.

**(m):**Call the function --- saveBookDetailsToFile and pass the parameters: books, bookCount.

**(n):**Declare a FILE pointer file and open the file in "rb+" mode.

**(o):**If file != NULL,goto step (p),else goto step (r).

**(p):**Write data to file using fwrite. [fwrite(books, sizeof(struct Book), count, file)]

**(q):**Close the file.

**(r):**Display error opening file.

**(s):**Increment bookCount.Goto step 11.

**Step-13:**If user choice =2,Goto step (a).Else goto step 14.

**(a):**Call the function --- displayCategoryWiseList and pass the parameter: books, bookCount.

**(b):**Display heading to display Fictional category.

**(c):**Declare and set i=0.

**(d):**If i<count, goto step (e), else goto step (h).

**(e):**If books[i].category == 1, goto step (f), else goto step (g).

**(f):**Print details of book[i].

**(g):**Increment i.

**(h):**Repeat the process for physics and history.

**(i):**Goto step 11.

**Step-14:**If user choice =3,Goto step (a).Else goto step 15.

**(a):**Call the function --- displayCategoryWiseTotalAndAverage and pass the parameters : books, bookCount)

**(b):**Declare categoryChoice.

**(c):**Input categoryChoice.

**(d):**Check the validity of input number i.e is number b/w 1-3.

**(e):**Declare totalCost, averageCost,bookCount, i and set to 0.

**(f):**If i<count, goto step (g), else goto step (j).

**(g):**If books[i].category == categoryChoice, goto step (h), else goto (i).

**(h):**totalCost += books[i].bookCost and increment bookCount. Goto step (i).

**(i):**Increment i.

**(j):**If bookCount >0, goto step (k), else goto step (m).

**(k):**Calculate averageCost = totalCost/bookCount.

**(l):**Print totalCost and averageCost.

**(m):**Display no books found in selected category.

**(n):**Goto step 9.

**Step-15:**If choice=4,Goto step (a), else goto step 16.

**(a):**Declare bookNumber.

**(b):**Input bookNumber

**(c):**Check the validity of input.

**(d):**Call the function --- displayBookDetails and pass the parameters: books, bookCount, bookNumber

**(e):**Declare i and index, set index=-1,i=0.

**(f):**If i<count, goto step (g), else goto step (k).

**(g):**If books[i].bookNumber == bookNumber, goto step (h), else goto step (i).

**(h):**index=i.Goto step (k).

**(i):**Increment i.

**(k):**If index!=-1, goto step (l), else goto step (m).

**(l):**Display book[i] details.

**(m):**Display Book with the specified number not found.

**(n):**Goto step 9.

**Step-16:**If choice=5,Goto step (a), else goto step 17.

**(a):**Declare bookNumber.

**(b):**Input bookNumber.

**(c):**Call the function deleteBook and pass the parameters: books, &bookCount, bookNumber.

**(d):**Declare i,index.Set i=0, index=-1.

**(e):**If i<\*count,goto step (f), else goto step (i).

**(f):**If books[i].bookNumber == bookNumber, goto step (g), else goto step (h).

**(g):**index=i. Goto step (i).

**(h):**Increment i.Goto step (e).

**(i):**If index!=-1, goto step (j), else goto step (o).

**(j):**If i<\*count, goto step (k), else goto step (m).

**(k):**books[i] = books[i + 1].

**(l):**Increment i.Goto step (j).

**(m):**\*count--.

**(n):**Display book has been deleted.Goto step (p).

**(o):**Display Book number not found.Goto step (p).

**(p):**Call the saveBookDetailsToFile function and pass the parameters: books, bookCount.

**(q):**Goto step 9.

**Step-17:** If choice=6, goto step (a), else goto step 18.

**(a):**Call the function clearAllBooks and pass the parameters :books, &bookCount.

**(b):**Set \*count=0.

**(c):**Display all books cleared.

**(d):**Call the function -- saveBookDetailsToFile and pass the parameters :books, bookCount.

**(e):**Perform the function as told from step 12(m) to 12(r).

**(f):**Goto step 9.

**Step-18:** If choice=7, Display Exiting the program, goto step 20;else goto step 19.

**Step-19:** Display Enter proper Input.Goto step 20.

**Step-20:**Stop.

**FLOWCHART:**

**FLOWCHART:**

Define constants and macros: MAX\_BOOKS for 10 and FILENAME for the binary file name.

no

no

yes

yes

Input choice

Display maximum limit reached

If i<count

Declare and assign i =0

Validate input for book details

Input book number

Is count<MAX\_BOOKS

Call the function saveBookDetails(struct Book \*books, int count)

and pass the parameters books, bookCount.

Is choice =1?

Display the main menu

Dynamically allocate memory for books and Declare bookCount,choice and initialize to bookCount=0.

Assign bookCount = loadBookDetailsFromFile(books), call the function and pass parameters, to check number of books

Declare function prototypes: void saveBookDetails(struct, int), void displayCategoryWiseList(struct , int),

void displayCategoryWiseTotalAndAverage(struct, int),void displayBookDetails(struct, int, int,); void clearAllBooks(struct, int);

void saveBookDetailsToFile(struct, int);int loadBookDetailsFromFile(struct);void deleteBook(struct, int, int);

Declare an enumeration enum Category {FICTIONAL, PHYSICS, HISTORY} and Define a structure struct Book

{int bookNumber, char bookTitle[50],char author[50], int numPages, enum Category category, float bookCost

no

yes

If books[i].bookNumber == bookNumber

yes

no

Display book already exists

yes

Display Fictional, history, physics books in order

Call the function displayCategoryWiseList and pass the parameter

If choice =2

Display error opening file

BookCount++

Close the file

Write data to file using fwrite

If file != NULL

Declare a FILE \*file and open the file in "rb+" mode

8 file and open the file in "rb+" mode

Call the function saveBookDetailsToFile and pass parameters

Input book details

Set books[count].bookNumber = bookNumber

i++

yes

no

no

no

yes

If choice =3

no

yes

Call the function --- displayBookDetails and pass the parameters: books, bookCount, bookNumber

Declare and input bookNumber

Is choice=4?

no

yes

Display no books found in selected category

Display averageCost and totalCost.

Calculate averageCost=totalCost/bookCount

Is bookCount>0?

Match the books of give category, Add their costs to totalCost. Increment bookCount everytime a book of the asked category is found

Declare and Input categoryChoice

Call the function displayCategoryWiseTotalAndAverage and pass the parameters.

Search for book with same bookNumber entered.

Is choice=5?

Call the function deleteBook and pass the parameters: books, &bookCount, bookNumber.

Declare and input bookNumber

Display specified book number not found

Display details of book[i]

If books[i].bookNumber == bookNumber

Search for book with same bookNumber entered.

no

no

yes

Yes

no

Yes

If books[i].bookNumber == bookNumber

Replace details of book[i] with book [i+1] until i<count.

no

Display enter valid input

yes

no

Display exiting the program

Is choice=7?

Run the saveBookDetailsToFile function again

Set \*count=0.

yes

Call the function clearAllBooks and pass the parameters :books, &bookCount.

Is choice=6

bookCount--

Run the saveBookDetailsToFile

Function again

Display book has been deleted

Display specified book details not found.

**CODE:**

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#define MAX\_BOOKS 10

#define FILENAME "booklibrary.bin"

// Enum to represent book categories

enum Category {

Fictional = 1,

Physics,

History

};

// Structure to store book details

struct Book {

int bookNumber;

char bookTitle[50];

char author[50];

int numPages;

enum Category category;

float bookCost;

};

// Function declarations

void saveBookDetails(struct Book \*books, int count);

void displayCategoryWiseList(struct Book \*books, int count);

void displayCategoryWiseTotalAndAverage(struct Book \*books, int count);

void displayBookDetails(struct Book \*books, int count, int bookNumber);

void saveBookDetailsToFile(struct Book \*books, int count);

int loadBookDetailsFromFile(struct Book \*books);

void deleteBook(struct Book \*books, int \*count, int bookNumber);

void clearAllBooks(struct Book \*books, int \*count);

int main() {

printf("\t\t\t###########################################################################");

printf("\n\t\t\t############ ############");

printf("\n\t\t\t############ Library management System Project in C ############");

printf("\n\t\t\t############ ############");

printf("\n\t\t\t###########################################################################");

printf("\n\t\t\t---------------------------------------------------------------------------\n");

// Allocate memory for books

struct Book \*books = (struct Book \*)malloc(MAX\_BOOKS \* sizeof(struct Book));

int bookCount = 0;

// Load existing book details from the binary file

bookCount = loadBookDetailsFromFile(books);

int choice;

do {

// Display menu

printf("\n\t\t\t\t\t1. Add a Book\n");

printf("\t\t\t\t\t2. Display Category-wise List\n");

printf("\t\t\t\t\t3. Category-wise Total and Average Book Cost\n");

printf("\t\t\t\t\t4. Display Book Details by Number\n");

printf("\t\t\t\t\t5. Delete Book by Number\n");

printf("\t\t\t\t\t6. Clear All Book Details\n");

printf("\t\t\t\t\t7. Exit\n");

printf("\t\t\t\t\tEnter your choice: ");

if (scanf("%d", &choice) != 1) {

printf("\t\t\t\t\tInvalid input for choice. Please enter a valid number.\n");

// Clear the input buffer

while (getchar() != '\n');

continue;

}

switch (choice) {

case 1:

saveBookDetails(books, bookCount);

saveBookDetailsToFile(books, bookCount); // Save to binary file after each addition

bookCount++;

break;

case 2:

displayCategoryWiseList(books, bookCount);

break;

case 3:

displayCategoryWiseTotalAndAverage(books, bookCount);

break;

case 4: {

int bookNumber;

printf("\t\t\t\t\tEnter Book Number: ");

if (scanf("%d", &bookNumber) != 1) {

printf("\t\t\t\t\tInvalid input for Book Number. Please enter a valid number.\n");

// Clear the input buffer

while (getchar() != '\n');

continue;

}

displayBookDetails(books, bookCount, bookNumber);

break;

}

case 5: {

int bookNumber;

printf("\t\t\t\t\tEnter Book Number to delete: ");

if (scanf("%d", &bookNumber) != 1) {

printf("\t\t\t\t\tInvalid input for Book Number. Please enter a valid number.\n");

// Clear the input buffer

while (getchar() != '\n');

continue;

}

deleteBook(books, &bookCount, bookNumber);

saveBookDetailsToFile(books, bookCount); // Save to binary file after deletion

break;

}

case 6:

clearAllBooks(books, &bookCount);

saveBookDetailsToFile(books, bookCount); // Save to binary file after clearing all books

break;

case 7:

printf("\t\t\t\t\tExiting program.\n");

free(books);

break;

default:

printf("\t\t\t\t\tInvalid choice. Please enter a number between 1 and 7.\n");

}

} while (choice != 7);

// Free dynamically allocated memory

free(books);

return 0;

}

// Function to save book details with input validation

void saveBookDetails(struct Book \*books, int count) {

if (count < MAX\_BOOKS) {

int bookNumber;

printf("\t\t\t\t\tEnter Book Number: ");

if (scanf("%d", &bookNumber) != 1) {

printf("\t\t\t\t\tInvalid input for Book Number. Please enter a valid number.\n");

// Clear the input buffer

while (getchar() != '\n');

return;

}

getch();

// Validate that Book Number contains only numbers

if (bookNumber < 0) {

printf("\t\t\t\t\tInvalid input for Book Number. Please enter a non-negative number.\n");

return;

}

int i;

// Check if Book Number already exists

for (i = 0; i < count; i++) {

if (books[i].bookNumber == bookNumber) {

printf("\t\t\t\t\tBook Number %d already exists. Please enter a different Number.\n", bookNumber);

return;

}

}

// Continue with other inputs if Book Number is valid

books[count].bookNumber = bookNumber;

printf("\t\t\t\t\tEnter Book Title: ");

getchar();

gets(books[count].bookTitle);

printf("\t\t\t\t\tEnter Author: ");

gets(books[count].author);

getchar();

printf("\t\t\t\t\tEnter Number of Pages: ");

scanf("%d", &books[count].numPages);

int categoryChoice;

printf("\t\t\t\t\tEnter Category (1.Fictional / 2.Physics / 3.History): ");

scanf("%d", &categoryChoice);

// Validate that Category choice is 1, 2, or 3

if (categoryChoice < 1 || categoryChoice > 3) {

printf("\t\t\t\t\tInvalid input for Category. Please enter 1, 2, or 3.\n");

}

books[count].category = categoryChoice;

printf("\t\t\t\t\tEnter Book Cost: ");

scanf("%f", &books[count].bookCost);

}

else {

printf("\t\t\t\t\tMaximum number of books reached. Cannot add more books.\n");

}

}

// Function to display category-wise list of books

void displayCategoryWiseList(struct Book \*books, int count) {

// Display category-wise list of books

printf("\n\t\t\t\tCategory-wise List of Books:\n");

printf("\t\t\t\t----------------------------------------------------------\n");

printf("\t\t\t\t------------------------Fictional-------------------------\n");

printf("\t\t\t\tBook Number\tBook Title\tAuthor Pages Cost\n");

printf("\t\t\t\t-----------------------------------------------------------\n");

int i;

for ( i = 0; i < count; i++) {

if (books[i].category == 1) {

printf("\t\t\t\t%-10d\t%-15s\t%-15s\t %-2d\t %.2f\n", books[i].bookNumber, books[i].bookTitle, books[i].author,books[i].numPages, books[i].bookCost);

}

}

printf("\t\t\t\t-------------------------Physics-------------------------\n");

printf("\t\t\t\tBook Number\tBook Title\tAuthor Pages Cost\n");

printf("\t\t\t\t----------------------------------------------------------\n");

for ( i = 0; i < count; i++) {

if (books[i].category == 2) {

printf("\t\t\t\t%-10d\t%-15s\t%-15s\t %-2d\t %.2f\n", books[i].bookNumber, books[i].bookTitle, books[i].author,

books[i].numPages, books[i].bookCost);

}

}

printf("\t\t\t\t-----------------------History----------------------------\n");

printf("\t\t\t\tBook Number\tBook Title\tAuthor Pages Cost\n");

printf("\t\t\t\t----------------------------------------------------------\n");

for ( i = 0; i < count; i++) {

if (books[i].category == 3) {

printf("\t\t\t\t%-10d\t%-15s\t%-15s\t %-2d\t %.2f\n", books[i].bookNumber, books[i].bookTitle, books[i].author,

books[i].numPages, books[i].bookCost);

}

}

}

// Function to display category-wise total and average book cost

void displayCategoryWiseTotalAndAverage(struct Book \*books, int count) {

int categoryChoice;

printf("\t\t\t\t\tEnter Category (1.Fictional / 2.Physics / 3.History): ");

if (scanf("%d", &categoryChoice) != 1) {

printf("\t\t\t\t\tInvalid input for Category. Please enter a valid number.\n");

// Clear the input buffer

while (getchar() != '\n');

return;

}

// Validate that Category choice is 1, 2, or 3

if (categoryChoice < 1 || categoryChoice > 3) {

printf("\t\t\t\t\tInvalid input for Category. Please enter 1, 2, or 3.\n");

return;

}

float totalCost = 0;

int bookCount = 0;

int i;

// Calculate category-wise total cost and count of books

for ( i = 0; i < count; i++) {

if (books[i].category == categoryChoice) {

totalCost += books[i].bookCost;

bookCount++;

}

}

if (bookCount > 0) {

float averageCost = totalCost / bookCount;

printf("\n\t\t\t\t\tCategory-wise Total and Average Book Cost:\n");

printf("\t\t\t\t\t-----------------------------------------\n");

printf("\t\t\t\t\tCategory: %d\n", categoryChoice);

printf("\t\t\t\t\tTotal Cost: %.2f\n", totalCost);

printf("\t\t\t\t\tAverage Cost: %.2f\n", averageCost);

} else {

printf("\t\t\t\t\tNo books found in the selected category.\n");

}

}

// Function to display book details by book number

void displayBookDetails(struct Book \*books, int count, int bookNumber) {

int index = -1;

int i;

// Find the index of the book with the specified book number

for ( i = 0; i < count; i++) {

if (books[i].bookNumber == bookNumber) {

index = i;

break;

}

}

if (index != -1) {

// Display book details

printf("\n\t\t\t\t\tBook Details:\n");

printf("\t\t\t\t\t--------------\n");

printf("\t\t\t\t\tBook Number: %d\n", books[index].bookNumber);

printf("\t\t\t\t\tBook Title: %s\n", books[index].bookTitle);

printf("\t\t\t\t\tAuthor: %s\n", books[index].author);

printf("\t\t\t\t\tNumber of Pages: %d\n", books[index].numPages);

printf("\t\t\t\t\tCategory: %d\n", books[index].category);

printf("\t\t\t\t\tBook Cost: %.2f\n", books[index].bookCost);

} else {

printf("\t\t\t\t\tBook with the specified number not found.\n");

}

}

// Function to save book details to binary file

void saveBookDetailsToFile(struct Book \*books, int count) {

FILE \*file = fopen(FILENAME, "rb+");

if (file != NULL) {

fwrite(books, sizeof(struct Book), count, file);

fclose(file);

} else {

printf("\t\t\t\t\tError opening the file %s for writing.\n", FILENAME);

}

}

// Function to load book details from binary file

int loadBookDetailsFromFile(struct Book \*books) {

FILE \*file = fopen(FILENAME, "rb");

int count = 0;

if (file != NULL) {

while (fread(&books[count], sizeof(struct Book), 1, file) == 1) {

count++;

}

fclose(file);

}

return count;

}

// Function to delete a book by book number

void deleteBook(struct Book \*books, int \*count, int bookNumber) {

int index = -1;

int i;

// Find the index of the book with the specified book number

for ( i = 0; i < \*count; i++) {

if (books[i].bookNumber == bookNumber) {

index = i;

break;

}

}

if (index != -1) {

int i;

// Shift elements to fill the gap left by the deleted book

for ( i = index; i < \*count - 1; i++) {

books[i] = books[i + 1];

}

(\*count)--;

printf("\t\t\t\t\tBook with Book Number %d has been deleted.\n", bookNumber);

} else {

printf("\t\t\t\t\tBook with the specified number not found.\n");

}

}

// Function to clear all book details

void clearAllBooks(struct Book \*books, int \*count) {

\*count = 0;

printf("\t\t\t\t\tAll book details are cleared.\n");

}

**OUTPUT:**

**TOPICS COVERED:**

Arrays

Structures

Pointers

Loops

Binary Files

User Defined Functions

Menu Driven

Macros

Dynamic Memory Allocation

**DEMONSTRATION:**

int main(). --- Rushi

void saveBookDetails(); --- Akhila

void saveBookDetailsToFile(); --- Akhila

void displayCategoryWiseList(); --- Shashi

void displayCategoryWiseTotalAndAverage();--- Shashi

void displayBookDetails(); --- Ishanth

int loadBookDetailsFromFile(); --- Ishanth

void deleteBook(); --- Babitha

void clearAllBooks() --- Babitha