**SOURSE CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct {

int id;

char cour\_obj\_code[15];

char cour\_id[15];

char bloom\_id[10];

float e\_proficiency;

float e\_attainment;

} CourseOutcome;

CourseOutcome course\_outcomes[MAX];

int course\_outcome\_count = 0;

const char\* FILE\_NAME = "course\_outcomes.txt";

void load\_from\_file();

void save\_to\_file();

void create\_course\_outcome();

void update\_course\_outcome();

void retrieve\_course\_outcomes();

void delete\_course\_outcome();

void search\_course\_outcome\_by\_code();

void insertion\_sort(CourseOutcome arr[], int n);

void merge(CourseOutcome arr[], int left, int mid, int right);

void merge\_sort(CourseOutcome arr[], int left, int right);

void sort\_course\_outcomes();

void load\_from\_file() {

FILE \*file = fopen(FILE\_NAME, "r");

if (file == NULL) {

return; // No file exists yet

}

course\_outcome\_count = 0;

while (fscanf(file, "%d %s %s %s %f %f\n", &course\_outcomes[course\_outcome\_count].id,

course\_outcomes[course\_outcome\_count].cour\_obj\_code,

course\_outcomes[course\_outcome\_count].cour\_id,

course\_outcomes[course\_outcome\_count].bloom\_id,

&course\_outcomes[course\_outcome\_count].e\_proficiency,

&course\_outcomes[course\_outcome\_count].e\_attainment) == 6) {

course\_outcome\_count++;

}

fclose(file);

}

void save\_to\_file() {

FILE \*file = fopen(FILE\_NAME, "w");

if (file == NULL) {

printf("Error opening file!\n");

return;

}

for (int i = 0; i < course\_outcome\_count; i++) {

fprintf(file, "%d %s %s %s %.2f %.2f\n", course\_outcomes[i].id,

course\_outcomes[i].cour\_obj\_code,

course\_outcomes[i].cour\_id,

course\_outcomes[i].bloom\_id,

course\_outcomes[i].e\_proficiency,

course\_outcomes[i].e\_attainment);

}

fclose(file);

}

void create\_course\_outcome() {

if (course\_outcome\_count >= MAX) {

printf("Course outcome list is full!\n");

return;

}

CourseOutcome co;

printf("Enter Course Outcome ID: ");

scanf("%d", &co.id);

printf("Enter Course Objective Code: ");

scanf("%s", co.cour\_obj\_code);

printf("Enter Course ID: ");

scanf("%s", co.cour\_id);

printf("Enter Bloom ID: ");

scanf("%s", co.bloom\_id);

printf("Enter Expected Proficiency: ");

scanf("%f", &co.e\_proficiency);

printf("Enter Expected Attainment: ");

scanf("%f", &co.e\_attainment);

course\_outcomes[course\_outcome\_count++] = co;

save\_to\_file();

printf("Course outcome created successfully!\n");

}

void update\_course\_outcome() {

int id;

printf("Enter Course Outcome ID to update: ");

scanf("%d", &id);

for (int i = 0; i < course\_outcome\_count; i++) {

if (course\_outcomes[i].id == id) {

printf("Enter new Course Objective Code: ");

scanf("%s", course\_outcomes[i].cour\_obj\_code);

printf("Enter new Course ID: ");

scanf("%s", course\_outcomes[i].cour\_id);

printf("Enter new Bloom ID: ");

scanf("%s", course\_outcomes[i].bloom\_id);

printf("Enter new Expected Proficiency: ");

scanf("%f", &course\_outcomes[i].e\_proficiency);

printf("Enter new Expected Attainment: ");

scanf("%f", &course\_outcomes[i].e\_attainment);

save\_to\_file();

printf("Course outcome updated successfully!\n");

return;

}

}

printf("Course outcome with ID %d not found.\n", id);

}

void retrieve\_course\_outcomes() {

printf("\nList of Course Outcomes:\n");

for (int i = 0; i < course\_outcome\_count; i++) {

printf("ID: %d\nCourse Objective Code: %s\nCourse ID: %s\nBloom ID: %s\nExpected Proficiency: %.2f\nExpected Attainment: %.2f\n\n",

course\_outcomes[i].id,

course\_outcomes[i].cour\_obj\_code,

course\_outcomes[i].cour\_id,

course\_outcomes[i].bloom\_id,

course\_outcomes[i].e\_proficiency,

course\_outcomes[i].e\_attainment);

}

}

void delete\_course\_outcome() {

int id;

printf("Enter Course Outcome ID to delete: ");

scanf("%d", &id);

for (int i = 0; i < course\_outcome\_count; i++) {

if (course\_outcomes[i].id == id) {

for (int j = i; j < course\_outcome\_count - 1; j++) {

course\_outcomes[j] = course\_outcomes[j + 1];

}

course\_outcome\_count--;

save\_to\_file();

printf("Course outcome deleted successfully!\n");

return;

}

}

printf("Course outcome with ID %d not found.\n", id);

}

void search\_course\_outcome\_by\_code() {

char code[15];

printf("Enter Course Objective Code to search: ");

scanf("%s", code);

for (int i = 0; i < course\_outcome\_count; i++) {

if (strcmp(course\_outcomes[i].cour\_obj\_code, code) == 0) {

printf("ID: %d\nCourse Objective Code: %s\nCourse ID: %s\nBloom ID: %s\nExpected Proficiency: %.2f\nExpected Attainment: %.2f\n\n",

course\_outcomes[i].id,

course\_outcomes[i].cour\_obj\_code,

course\_outcomes[i].cour\_id,

course\_outcomes[i].bloom\_id,

course\_outcomes[i].e\_proficiency,

course\_outcomes[i].e\_attainment);

return;

}

}

printf("Course outcome with code %s not found.\n", code);

}

void insertion\_sort(CourseOutcome arr[], int n) {

for (int i = 1; i < n; i++) {

CourseOutcome key = arr[i];

int j = i - 1;

while (j >= 0 && strcmp(arr[j].cour\_obj\_code, key.cour\_obj\_code) > 0) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

printf("Course outcomes sorted by Course Objective Code using Insertion Sort!\n");

}

void merge(CourseOutcome arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

CourseOutcome \*L = malloc(n1 \* sizeof(CourseOutcome));

CourseOutcome \*R = malloc(n2 \* sizeof(CourseOutcome));

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

// Merge the temporary arrays back into arr[left..right]

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (strcmp(L[i].cour\_obj\_code, R[j].cour\_obj\_code) <= 0) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

// Copy any remaining elements of L[]

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Copy any remaining elements of R[]

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

free(L);

free(R);

}

// Merge Sort function

void merge\_sort(CourseOutcome arr[], int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

merge\_sort(arr, left, mid); // Sort first half

merge\_sort(arr, mid + 1, right); // Sort second half

merge(arr, left, mid, right); // Merge sorted halves

}

}

// Function to sort the array of Course Outcomes

void sort\_course\_outcomes() {

int choice;

printf("\nChoose sorting algorithm:\n");

printf("1. Insertion Sort\n");

printf("2. Merge Sort\n");

scanf("%d", &choice);

switch(choice) {

case 1:

insertion\_sort(course\_outcomes, course\_outcome\_count);

retrieve\_course\_outcomes(); // Display sorted outcomes

break;

case 2:

merge\_sort(course\_outcomes, 0, course\_outcome\_count - 1);

printf("Course outcomes sorted by Course Objective Code using Merge Sort!\n");

retrieve\_course\_outcomes(); // Display sorted outcomes

break;

default:

printf("Invalid choice!\n");

break;

}

}

int main() {

load\_from\_file();

int choice;

while (1) {

printf("\n1. Create Course Outcome\n2. Update Course Outcome\n3. Retrieve Course Outcomes\n4. Delete Course Outcome\n5. Search by Code\n6. Sort by Code\n7. Exit\n");

printf("Enter your choice: ");

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

fprintf(stderr,"Invalid input! Exiting...\n");

exit(EXIT\_FAILURE);

}

switch (choice) {

case 1:

create\_course\_outcome();

break;

case 2:

update\_course\_outcome();

break;

case 3:

retrieve\_course\_outcomes();

break;

case 4:

delete\_course\_outcome();

break;

case 5:

search\_course\_outcome\_by\_code();

break;

case 6:

sort\_course\_outcomes();

break;

case 7:

exit(0);

default:

printf("Invalid choice!\n");

}

}

return 0;

}