**ONLINE GDB CODE LINK:**

<https://onlinegdb.com/n8EZVCJXt>

**Time Capsule Organizer:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h> // for sleep function

#define MAX\_EVENTS 100

#define MAX\_TITLE\_LEN 50

#define MAX\_DESC\_LEN 100

#define MAX\_MEDIA\_LEN 200 // Adjusted the maximum length for a URL

#define MAX\_FILENAME\_LEN 100 // Maximum length for a filename

// ANSI color codes for better readability

#define ANSI\_COLOR\_RESET "\x1b[0m"

#define ANSI\_COLOR\_RED "\x1b[31m"

#define ANSI\_COLOR\_GREEN "\x1b[32m"

#define ANSI\_COLOR\_YELLOW "\x1b[33m"

#define ANSI\_COLOR\_BLUE "\x1b[34m"

#define ANSI\_COLOR\_MAGENTA "\x1b[35m"

#define ANSI\_COLOR\_CYAN "\x1b[36m"

#define ANSI\_COLOR\_BOLD "\x1b[1m"

// Structure to represent an event

struct Event

{

char title[MAX\_TITLE\_LEN];

char date[20];

char description[MAX\_DESC\_LEN];

char media[MAX\_MEDIA\_LEN];

char category[20];

int isActive;

};

void saveEventsToFile(const struct Event events[], int eventCount);

// Function to validate a date

int isValidDate(const char \*date)

{

int year, month, day;

if (sscanf(date, "%d-%d-%d", &year, &month, &day) != 3)

{

return 0; // Not a valid date format

}

if (year < 1900 || year > 2100 || month < 1 || month > 12 || day < 1)

{

return 0; // Invalid year, month, or day

}

const int daysInMonth[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

if (day > daysInMonth[month] || (month == 2 && year % 4 == 0 && (year % 100 != 0 || year % 400 == 0) && day > 29))

{

return 0; // Day exceeds the maximum for the given month and year

}

return 1; // Valid date

}

// Function to add a new event

void addEvent(struct Event events[], int \*eventCount)

{

if (\*eventCount < MAX\_EVENTS)

{

struct Event newEvent;

printf("Enter event title: ");

scanf(" %[^\n]", newEvent.title);

// Validate the date

do

{

printf("Enter event date (YYYY-MM-DD): ");

scanf("%s", newEvent.date);

} while (!isValidDate(newEvent.date));

printf("Enter event description: ");

scanf(" %[^\n]", newEvent.description);

printf("Enter event category: ");

scanf(" %[^\n]", newEvent.category);

// Assume that the photo is already stored locally or accessible via a URL

printf("Enter filename or URL for the photo: ");

scanf(" %[^\n]", newEvent.media);

newEvent.isActive = 1; // Event is initially active

events[\*eventCount] = newEvent;

(\*eventCount)++;

printf("Event added successfully!\n");

// Save events to file after adding a new event

saveEventsToFile(events, \*eventCount);

}

else

{

printf("Maximum number of events reached.\n");

}

}

// Function to save events to a file

void saveEventsToFile(const struct Event events[], int eventCount)

{

FILE \*file = fopen("events.txt", "w");

if (file == NULL)

{

perror("Error opening file for writing");

return;

}

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

{

fprintf(file, "%s,%s,%s,%s,%s,%d\n",events[i].title, events[i].date, events[i].description, events[i].media, events[i].category, events[i].isActive);

}

fclose(file);

}

// Function to load events from a file

int loadEventsFromFile(struct Event events[])

{

FILE \*file = fopen("events.txt", "r");

if (file == NULL)

{

perror("Error opening file for reading");

return 0;

}

int count = 0;

while (fscanf(file, "%49[^,],%19[^,],%99[^,],%199[^,],%19[^,],%d\n",events[count].title, events[count].date, events[count].description,events[count].media, events[count].category, &events[count].isActive) == 6)

{

count++;

if (count >= MAX\_EVENTS)

{

break; // Avoid reading more events than the array can hold

}

}

fclose(file);

return count;

}

// Function to display all events

void displayEvents(const struct Event events[], int eventCount)

{

printf("%s%s= = = = = = = = = = %s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET);

printf("%s%s %sEVENTS%s %s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET, ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s%s= = = = = = = = = = %s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET);

if (eventCount > 0)

{

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

{

if (events[i].isActive)

{

printf("Title: %s\n", events[i].title);

printf("Date: %s\n", events[i].date);

printf("Description: %s\n", events[i].description);

printf("Category: %s\n", events[i].category);

printf("Photo: %s\n", events[i].media);

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

// Add an option to view media

printf("Do you want to view the media? (1 for Yes, 0 for No): ");

int viewMedia;

scanf("%d", &viewMedia);

if (viewMedia)

{

printf("Opening media: %s\n", events[i].media);

char command[300];

sprintf(command, "xdg-open %s", events[i].media);

system(command);

}

}

}

}

else

{

printf("No events available.\n");

}

}

// Function to search events

void searchEvents(const struct Event events[], int eventCount)

{

char keyword[MAX\_DESC\_LEN];

printf("\nEnter keyword to search: ");

scanf(" %[^\n]", keyword);

int found = 0;

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

{

if (events[i].isActive && (strstr(events[i].title, keyword) || strstr(events[i].description, keyword) || strstr(events[i].category, keyword) || strstr(events[i].media, keyword)))

{

printf("Title: %s\n", events[i].title);

printf("Date: %s\n", events[i].date);

printf("Description: %s\n", events[i].description);

printf("Category: %s\n", events[i].category);

printf("Photo: %s\n", events[i].media);

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

found = 1;

}

}

if (!found)

{

printf("No events found with the given keyword.\n");

}

}

// Function to categorize events

void categorizeEvents(struct Event events[], int eventCount)

{

char category[MAX\_DESC\_LEN];

printf("\nEnter category to filter: ");

scanf(" %[^\n]", category);

int found = 0;

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

{

if (events[i].isActive && strcmp(events[i].category, category) == 0)

{

printf("Title: %s\n", events[i].title);

printf("Date: %s\n", events[i].date);

printf("Description: %s\n", events[i].description);

printf("Photo: %s\n", events[i].media);

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

found = 1;

}

}

if (!found)

{

printf("No events found in the given category.\n");

}

}

// Function to toggle time capsule

void toggleTimeCapsule(struct Event events[], int eventCount)

{

char title[MAX\_TITLE\_LEN];

printf("\nEnter the title of the event to toggle time capsule: ");

scanf(" %[^\n]", title);

int found = 0;

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

{

if (events[i].isActive && strcmp(events[i].title, title) == 0)

{

events[i].isActive = !events[i].isActive;

printf("Time capsule for event '%s' is now %s.\n", title, events[i].isActive ? "active" : "buried");

found = 1;

break;

}

}

if (!found)

{

printf("Event not found. Unable to toggle time capsule.\n");

}

// Save events to file after toggling time capsule

saveEventsToFile(events, eventCount);

}

// Function to display the interactive menu

void displayMenu()

{

printf("%s%sTIME CAPSULE ORGANIZER%s", ANSI\_COLOR\_CYAN, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET);

printf("\n%s%s= = = = = = = = = %s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET);

printf("%s%s %s MENU%s %s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET, ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s%s= = = = = = = = = %s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_BOLD, ANSI\_COLOR\_RESET);

printf("%s1. ADD EVENT%s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s2. DISPLAY ALL EVENTS%s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s3. SEARCH EVENTS%s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s4. CATEGORIZE EVENTS%s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s5. TOGGLE TIME CAPSULE%s\n", ANSI\_COLOR\_YELLOW, ANSI\_COLOR\_RESET);

printf("%s6. QUIT%s\n", ANSI\_COLOR\_RED, ANSI\_COLOR\_RESET);

}

int main()

{

struct Event events[MAX\_EVENTS];

int eventCount = 0;

// Load events from file at the start of the program

eventCount = loadEventsFromFile(events);

int choice;

do

{

displayMenu();

printf("%sEnter your choice (1-6): %s", ANSI\_COLOR\_BLUE, ANSI\_COLOR\_RESET);

// scanf("%d", &choice);

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

{

// Clear the input buffer

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

printf("%sInvalid input. Please enter a number.%s\n", ANSI\_COLOR\_RED, ANSI\_COLOR\_RESET);

continue; // Restart the loop

}

switch (choice)

{

case 1:

addEvent(events, &eventCount);

break;

case 2:

displayEvents(events, eventCount);

break;

case 3:

searchEvents(events, eventCount);

break;

case 4:

categorizeEvents(events, eventCount);

break;

case 5:

toggleTimeCapsule(events, eventCount);

break;

case 6:

printf("%sGoodbye!%s\n", ANSI\_COLOR\_RED, ANSI\_COLOR\_RESET);

break;

default:

printf("%sInvalid choice. Try again.%s\n", ANSI\_COLOR\_RED, ANSI\_COLOR\_RESET);

}

// Adding a small delay for better visualization

// sleep(1);

} while (choice != 6);

return 0;

}