**ICS 1101 Problem Solving and Programming in C**

**Assignment 2**

**Name: K.S.Vijayaraaghavan** **Date:7/12/24**

**ID:2470005**

1. Develop a program using structures to store user credentials (email and password). Create and initialize an array of any 5 user credentials.

***Code:***

#include <stdio.h>

#include <string.h>

// Define a structure to store user credentials

typedef struct UserCredentials {

char email[50];

char password[50];

}Cred;

int main() {

// Initialize an array of 5 user credentials

Cred users[5] = {

{"user1@example.com", "passwd1234"},

{"user2@example.com", "securePass56"},

{"user3@example.com", "myPassword789"},

{"user4@example.com", "iamwhoiam12"},

{"user5@example.com", "passispass1@"}

};

// Display the user credentials

printf("Stored User Credentials:\n");

for (int i = 0; i < 5; i++) {

printf("User %d:\n", i + 1);

printf(" Email: %s\n", users[i].email);

printf(" Password: %s\n\n", users[i].password);

}

return 0;

}

}

***Explanation:***

***Structure Definition:***A structure UserCredentials is defined to hold an email and a password, both as strings.

***Initialization:***An array of 5 UserCredentials structures is created and initialized with sample credentials.

***Display:***A loop iterates through the array to print each user's email and password.

***Output:***

A screen shot of a computer

Description automatically generated

1. Input a username and password and demonstrate the working of your authentication system for all possible test cases.

***Code:***

#include <stdio.h>

#include <string.h>

// Define a structure to store user credentials

typedef struct UserCredentials {

char email[50];

char password[50];

}Cred;

// Function to validate the email format

int isValidEmail(char email[]) {

int atFlag = 0, dotFlag = 0;

for (int i = 0; email[i] != '\0'; i++) {

if (email[i] == '@') {

atFlag = 1;

}

if (email[i] == '.' && atFlag) {

dotFlag = 1;

}

}

return atFlag && dotFlag;

}

// Function to validate password format

int isValidPassword(char password[]) {

if(strlen(password) >= 8;)

return 1;

else return 0;

}

// Function to authenticate user

int authenticateUser(Cred users[], int size, char email[], char password[]) {

for (int i = 0; i < size; i++) {

if (strcmp(users[i].email, email) == 0 && strcmp(users[i].password, password) == 0) {

return 1; // Authentication successful

}

}

return 0; // Authentication failed

}

int main() {

// Initialize an array of 5 user credentials

struct User users[5] = {

{"user1@example.com", "password1234"},

{"user2@example.com", " securePass56"},

{"user3@example.com", "myPassword789"},

{"user4@example.com", " iamwhoiam12"},

{"user5@example.com", “passispass1@"}

};

char inputEmail[50], inputPassword[50];

// Input email and password

printf("Enter your email: ");

scanf("%s", inputEmail);

printf("Enter your password: ");

scanf("%s", inputPassword);

// Validate email and password format

if (!isValidEmail(inputEmail)) {

printf("Error: Invalid email format.\n");

return 1;

}

if (!isValidPassword(inputPassword)) {

printf("Error: Password must be at least 8 characters long.\n");

return 1;

}

// Authenticate user

if (authenticateUser(users, 5, inputEmail, inputPassword)) {

printf("Login successful.\n");

} else {

printf("Error: Invalid credentials.\n");

}

return 0;

}

***Explanation:***

1. ***Structure and Data:***

The UserCredentials structure holds email and password.

An array of 5 users is initialized with example credentials.

1. ***Functions:***

isValidEmail: Checks if the email contains both @ and . after @.

isValidPassword: Ensures the password is at least 8 characters long.

authenticateUser: Compares the entered email and password with the stored credentials in the users array.

1. ***Main Program:***

The program prompts the user for their email and password.

It validates the input formats using isValidEmail and isValidPassword.

If valid, it calls authenticateUser to check against stored credentials.

It prints appropriate messages for success or failure.

***Test Cases:***

***Case 1:*** Valid email and password, correct credentials

**Input:**

Enter your email: user1@example.com

Enter your password: password1234

**Output:**

Login successful.

***Case 2:*** Valid email and password, incorrect credentials

**Input:**

Enter your email: user1@example.com

Enter your password: wrongPassword

**Output:**

Error: Invalid credentials.

***Case 3:*** Invalid email format

**Input:**

Enter your email: user1example.com

Enter your password: password1234

**Output:**

Error: Invalid email format.

***Case 4:*** Password too short

**Input:**

Enter your email: user1@example.com

Enter your password: short

**Output:**

Error: Password must be at least 8 characters long.

1. Write a C program to manage and analyse sales data for any number of days using dynamic memory allocation.

***Code:***

#include <stdio.h>

#include <stdlib.h>

// Function prototypes

void inputSales(float \*sales, int days);

void addNewSale(float \*\*sales, int \*days);

void displaySales(float \*sales, int days);

float calculateTotalSales(float \*sales, int days);

int main() {

int days;

float \*sales = NULL; // Pointer for dynamically allocated memory

int choice;

// Ask the user to input the number of days

printf("Enter the number of days to track sales: ");

scanf("%d", &days);

// Allocate memory for the sales data

sales = (float \*)malloc(days \* sizeof(float));

if (sales == NULL) {

printf("Memory allocation failed.\n");

return 1;

}

// Menu-driven program

do {

printf("\nMenu:\n");

printf("1. Input Sales\n");

printf("2. Add a New Sale\n");

printf("3. Display Sales\n");

printf("4. Calculate Total Sales\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

inputSales(sales, days);

break;

case 2:

addNewSale(&sales, &days);

break;

case 3:

displaySales(sales, days);

break;

case 4: {

float total = calculateTotalSales(sales, days);

printf("Total Sales: %.2f\n", total);

break;

}

case 5:

printf("Exiting the program.\n");

break;

default:

printf("Invalid choice. Please try again.\n");

}

} while (choice != 5);

// Free the allocated memory

free(sales);

return 0;

}

// Function to input sales data for all days

void inputSales(float \*sales, int days) {

for (int i = 0; i < days; i++) {

printf("Enter sales for day %d: ", i + 1);

scanf("%f", &sales[i]);

}

}

// Function to add sales data for an additional day

void addNewSale(float \*\*sales, int \*days) {

(\*days)++;

\*sales = (float \*)realloc(\*sales, (\*days) \* sizeof(float));

if (\*sales == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

printf("Enter sales for day %d: ", \*days);

scanf("%f", &(\*sales)[\*days - 1]);

}

// Function to display sales data

void displaySales(float \*sales, int days) {

printf("Sales data:\n");

for (int i = 0; i < days; i++) {

printf("Day %d: %.2f\n", i + 1, sales[i]);

}

}

// Function to calculate the total sales

float calculateTotalSales(float \*sales, int days) {

float total = 0.0;

for (int i = 0; i < days; i++) {

total += sales[i];

}

return total;

}

***Explanation***

1. ***Dynamic Memory Allocation:***

The program uses malloc to allocate memory dynamically for the sales data array.

realloc is used to expand the memory when adding a new sale.

1. ***Functions:***

inputSales: Inputs sales data for all days.

addNewSale: Adds sales data for an additional day by reallocating memory.

displaySales: Displays all sales data entered by the user.

calculateTotalSales: Computes the total of all sales

1. ***Menu:***

A menu allows the user to choose actions repeatedly until they decide to exit.

***Output:***

A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

***Key Features:***

1. The program handles any number of days as specified by the user.
2. It ensures efficient memory usage by dynamically allocating and reallocating memory.
3. It supports adding more days dynamically and recalculates totals accurately.