**27/03/2024**

**Program 1:**

**You are tasked with writing a C program that checks the strength of a password entered by a user. The program should analyze the password based on the following criteria:**

**The password must be at least 8 characters long.**

**The password must contain at least one uppercase letter, one lowercase letter, one digit, and one special character (such as !, @, #, $, %, etc.).**

**No spaces are allowed in the password.**

**Your program should prompt the user to enter a password and then evaluate its strength according to the criteria mentioned above. If the password meets all criteria, it should be deemed strong; otherwise, it should be considered weak.**

**Your program should provide informative messages to the user regarding the strength of their password and which criteria it fails to meet if it is weak.**

**Here are some guidelines to help you implement the program:**

**Use string functions like strlen(), isupper(), islower(), isdigit(), and strchr() for string manipulation and character checking.**

**You may use loops and conditional statements as necessary.**

**Ensure error handling for cases where the password does not meet the minimum length requirement or if spaces are found.**

**Utilize meaningful prompts and messages for user interaction.**

**Output:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void checkPassword(char password[])

{

int length;

int upper = 0, lower = 0, digit = 0, special = 0;

length = strlen(password);

if (length < 8)

{

printf("Password is too short. It must be at least 8 characters long.\n");

}

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

{

char character = password[i];

if (isupper(character))

{

upper = 1;

}

if (islower(character))

{

lower = 1;

}

if (isdigit(character))

{

digit = 1;

}

if ((character >= 33 && character <= 42) || (character >= 59 && character <= 64))

{

special = 1;

}

if (strchr(" ", character))

{

printf("Spaces are not allowed in the password.\n");

}

}

if (upper && lower && digit && special)

{

printf("Strong password!\n");

}

else

{

printf("Weak password.!\n");

if (!upper) {

printf("Missing uppercase letter.\n");

}

if (!lower) {

printf("Missing lowercase letter.\n");

}

if (!digit) {

printf("Missing digit.\n");

}

if (!special) {

printf("Missing special character.\n");

}

}

}

int main()

{

char password[30];

printf("Enter your password: ");

scanf("%s", password);

checkPassword(password);

return 0;

}

**Program 2:**

**You are tasked with writing a C program that analyzes a sentence entered by a user. The program should prompt the user to enter a sentence containing multiple words. After receiving the input, the program should tokenize the sentence into individual words using strtok and then perform the following tasks:**

**Count the total number of words in the sentence.**

**Determine the average length of the words in the sentence.**

**Identify and print the longest word in the sentence.**

**Output:**

#include <stdio.h>

#include <string.h>

int main()

{

char sentence[50];

int words\_count=0,total\_length = 0,max\_length=0;

char longest\_word[50];

printf("Enter a sentence: ");

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

char \*delimiters=" ";

char\* token = strtok(sentence, delimiters);

while (token != NULL)

{

words\_count++;

printf("Token: %s\n", token);

total\_length += strlen(token);

if (strlen(token) > max\_length)

{

max\_length = strlen(token);

strcpy(longest\_word, token);

}

token = strtok(NULL, delimiters);

}

printf("Total no of words(count):%d\n",words\_count);

int average\_length = total\_length / words\_count;

printf("Average length of words: %d\n", average\_length);

printf("Longest word: %s", longest\_word);

return 0;

}

**Program 3:**

**Question:**

char source[50]="apPle,is,a fruit,onion is a,vegetable";

char \*delimiters=",";

char\* token = strtok(source, delimiters);

while (token != NULL) {

printf("Token: %s\n", token);

token = strtok(NULL, delimiters);

**Output:**

#include <stdio.h>

#include <string.h>

int main()

{

char sentence[50]="apPle,is,a fruit,onion is a,vegetable";

int words\_count=0,total\_length = 0,max\_length=0;

char longest\_word[50];

char \*delimiters=",";

char\* token = strtok(sentence, delimiters);

while (token != NULL) {

words\_count++;

printf("Token: %s\n", token);

total\_length += strlen(token);

if (strlen(token) > max\_length) {

max\_length = strlen(token);

strcpy(longest\_word, token);

}

token = strtok(NULL, delimiters);

}

printf("Total length : %d",total\_length);

printf("Total no of words(count):%d\n",words\_count);

int average\_length = total\_length / words\_count;

printf("Average length of words: %d\n", average\_length);

printf("Longest word: %s", longest\_word);

return 0;

}

}

**Assignments:**

**Question :**

**You are tasked with developing a program to manage the employee database of a medium-sized company. The HR department requires a software solution that allows them to efficiently maintain employee records, update information as needed, and generate reports when necessary.**

**Your task is to design and implement a program using C language that fulfills the requirements outlined below:**

**Employee Database Structure: Define a structure to represent employee records. Each record should include the following fields:**

**Employee ID (an integer value)**

**Name (a string of characters)**

**Department (a string of characters)**

**Designation (a string of characters)**

**Salary (a floating-point value)**

**Functionality:**

**Add New Employees: Implement a function to add new employees to the database. The function should prompt the user to input employee details such as ID, name, department, designation, and salary.**

**Display Employee Details: Implement a function to display the details of all employees stored in the database.**

**Update Employee Information: Implement a function to update the information of existing employees. The function should allow the user to search for an employee by ID and then update their name, department, designation, or salary.**

**User Interface: Develop a user-friendly interface that allows HR staff to interact with the program easily. The interface should provide options for adding new employees, displaying employee details, updating employee information, and deleting employees.**

**Error Handling: Ensure robust error handling throughout the program. Validate user input to prevent invalid data entry and provide informative error messages when necessary.**

**Scalability: Design the program to handle a moderate number of employees efficiently. Consider using dynamic memory allocation to accommodate a growing database size if needed.**

**Output**

#include<stdio.h>

#include <stdlib.h>

#include <string.h>

struct Employee {

int Emp\_id;

char emp\_name[30];

char Department[30];

char Designation[30];

float Salary;

};

struct Employee \*emp = NULL;

int num\_of\_emps = 0;

void addNewEmployees() {

struct Employee newemp;

printf("\nEnter Employee ID: ");

scanf("%d", &newemp.Emp\_id);

printf("Enter Employee Name: ");

scanf(" %[^\n]s", newemp.emp\_name);

printf("Enter Employee Department: ");

scanf(" %[^\n]s", newemp.Department);

printf("Enter Employee Designation: ");

scanf(" %[^\n]s", newemp.Designation);

printf("Enter Employee Salary: ");

scanf("%f", &newemp.Salary);

num\_of\_emps++;

emp = (struct Employee \*)realloc(emp, num\_of\_emps \* sizeof(struct Employee));

emp[num\_of\_emps - 1] = newemp;

printf("New employee added successfully!\n");

}

void DisplayEmployees() {

printf("\nEmployee Details:\n");

for (int i = 0; i < num\_of\_emps; i++) {

printf("Employee ID: %d\n", emp[i].Emp\_id);

printf("Name: %s\n", emp[i].emp\_name);

printf("Department: %s\n", emp[i].Department);

printf("Designation: %s\n", emp[i].Designation);

printf("Salary: %.2f\n", emp[i].Salary);

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

}

}

void UpdateExistingEmployees() {

int Emp\_id;

printf("\nEnter Employee ID to update: ");

scanf("%d", &Emp\_id);

int found = 0;

for (int i = 0; i < num\_of\_emps; i++) {

if (emp[i].Emp\_id == Emp\_id) {

found = 1;

printf("Enter new name: ");

scanf(" %[^\n]s", emp[i].emp\_name);

printf("Enter new department: ");

scanf(" %[^\n]s", emp[i].Department);

printf("Enter new designation: ");

scanf(" %[^\n]s", emp[i].Designation);

printf("Enter new salary: ");

scanf("%f", &emp[i].Salary);

printf("Employee information updated successfully!\n");

break;

}

}

if (!found) {

printf("Employee with ID %d not found.\n", Emp\_id);

}

}

void DeleteEmployees() {

int Emp\_id;

printf("\nEnter Employee ID to delete: ");

scanf("%d", &Emp\_id);

int found = 0;

for (int i = 0; i < num\_of\_emps; i++) {

if (emp[i].Emp\_id == Emp\_id) {

found = 1;

emp[i] = emp[num\_of\_emps - 1];

num\_of\_emps--;

emp = (struct Employee \*)realloc(emp, num\_of\_emps \* sizeof(struct Employee));

printf("Employee with ID %d deleted successfully!\n", Emp\_id);

break;

}

}

if (!found) {

printf("Employee with ID %d not found.\n", Emp\_id);

}

}

int main() {

int option;

do {

printf("\nEmployee Database Management System\n");

printf("1- ADD EMPLOYEES\n");

printf("2- UPDATE EMPLOYEES\n");

printf("3 - DISPLAY EMPLOYEES\n");

printf("4 - DELETE EMPLOYEES\n");

printf("5 - EXIT\n");

printf("Enter the option : \n");

scanf("%d", &option);

switch(option) {

case 1:

addNewEmployees();

break;

case 2:

UpdateExistingEmployees();

break;

case 3:

DisplayEmployees();

break;

case 4:

DeleteEmployees();

break;

case 5:

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

break;

default:

printf("Enter the valid option...\n");

break;

}

} while(option != 5);

free(emp);

return 0;

}