TP 4 - C

Les Tps se font sous un environnement linux, vous pouvez utiliser l'éditeur de texte que vous preferez.

Bases

1. LowerCase

1. Créez une fonction qui accepte en argument un pointeur vers un char et, si la valeur pointée est un char, la change de majuscule en minuscule.
2. Appelez cette fonction sur tous les charactères d'une chaine saisie par l'utilisateur.

2. Echange

1. Ecrivez une fonction `swap(int* a, int* b)` qui accepte en argument deux pointeurs vers des int et qui échange leur valeurs.

3 Concat

L'objectif de cet exercice est de comprendre le fonctionnement de `strcat` . Il ne faut donc pas utiliser cette fonction pour cet exercice.

1. Ecrivez une fonction `char* concat(char*, char*)` qui accepte en argument deux chaines de charactères et renvoie une nouvelle chaine.
2. Ecrivez un programme qui demande plusieurs chaines de charactères à l'utilisateur et les concatène en utilisant la fonction précédente.
3. Ecrivez une fonction `char* concatmul(int n, char**)` qui accepte en argument une liste de n chaines de charactères et qui les concatène en allouant une seule fois de la mémoire.

L'utilisation de mémoire plus optimale dans la deuxième fonction est la raison pour laquelle on préfère souvent utiliser des objets spécifiques pour concatener plusieurs chaines de charactères dans d'autres langages (stringstream en C++, StringBuilder en Java, join en Python)

4. Vecteur

L'objectif de cet exercice est de créer un tableau d'entiers de taille variable. Pour cela nous stockeront la taille du tableau dans le premier élément.

1. Créez une fonction qui crée le tableau, qui alloue de la mémoire pour un entier et y stoque la taille.
2. Créez une fonction permettan d'ajouter un element à la fin du tableau.

Note : Comme le premier élément est utilisé pour la taille, les index commenceront donc à 1.

1. Créez une fonction qui permet de libérer la mémoire allouée par le tableau.
2. Utilisez ce tableau pour créer un programme qui permet à l'utilisateur de le remplir à sa guise puis de l'afficher a l'envers.

Exercices avancés

5. Oups

Le code ci-dessous permet à des utilisateurs de créer un compte avec un mot de passe puis de stoquer des données importantes Il contient deux failles de sécurité qui permettent d'accéder aux informations d'un autre utilisateur et de faire crasher le serveur.

1. Familiarisez vous avec le code et son utilisation.
2. Trouvez les failles, exploitez les puis corrigez les.

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

#define GET_VALUE(format, name, var) printf("please enter : " #name "\n");scanf(" "
#format, var);

#define USER_LEN 4
#define MAX_USER_COUNT 6

int login(char** users, char** passwords, int* current_count);

void add_user(char** users, char** passwords, int* current_count){
    passwords[*current_count] = malloc(USER_LEN * sizeof(char));
    users[*current_count] = malloc(USER_LEN * sizeof(char));

    GET_VALUE(%4s, your user name, users[*current_count]);

    GET_VALUE(%4s, your password, passwords[*current_count]);
    (*current_count)++;
}


void remove_user(char** users, char** passwords, char** data, int* current_count){
    int i;
    if ((i = login(users, passwords, current_count)) != -1){
        free(users[i]);
        free(passwords[i]);
        free(data[i]);
        (*current_count)--;
    } else {
        printf("not logged in \n");
    }
}

void add_data(char** users, char** passwords, char** data, int* current_count){
    int i;
    if ((i = login(users, passwords, current_count)) != -1){
        if (data[i]){
            printf("previous data was (%p): %s\n", data[i], data[i]);
        }

        data[i] = malloc(sizeof(char) * USER_LEN);
        GET_VALUE(%4s, your sensitive data, data[i]);
    }
}
```

```
int login(char** users, char** passwords, int* current_count){
    char* user_name = malloc(sizeof(char) * USER_LEN);
    GET_VALUE(%s, your user name, user_name);
    char* password = malloc(sizeof(char) * USER_LEN);
    GET_VALUE(%s, your password, password);

    for (int i = 0; i < *current_count; i++){
        if (!strcmp(users[i], user_name) && !strcmp(passwords[i], password)){
            return i;
        }
    }
    printf("could not log you in sorry\n");
    return -1;
}



int main(){
    int current_count = 0;
    char** passwords = malloc(MAX_USER_COUNT * sizeof(char*));
    char** users = malloc(MAX_USER_COUNT * sizeof(char*));
    char** user_data = malloc(MAX_USER_COUNT * sizeof(char*));

    for(;;){
        char response;

        GET_VALUE(%c, \na to add user \nr to remove user\ns to show all users\nd to
view or edit data of an user, &response);
        if (response == 'a'){
            if (current_count < MAX_USER_COUNT){
                add_user(users, passwords, &current_count);
            }
        } if (response == 's'){
            for (int i = 0; i < current_count; i++){
                printf("user %s logged recently\n", users[i]);
            }
        } if (response == 'r'){
            remove_user(users, passwords, user_data, &current_count);
        } if (response == 'd'){
            add_data(users, passwords, user_data, &current_count);
        }if (response == 't'){
            for (int i = 0; i < current_count; i++){
                printf("%p %p %p\n", users[i], passwords[i], user_data[i]);
            }
        }
        printf("\n\n");
    }
}
```