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 contenant la concaténation des deux.
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.

Note: 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 de flottants de taille variable. Pour cela nous aurons besoin d'une variable de type `int` pour enregistrer la taille du tableau et d'une variable de type `float*` pour enregistrer les éléments.

1. Créez une fonction qui initialise le tableau avec un élément à l'interieur. La fonction acceptera les arguments suivants:
    - Un pointeur vers la taille du tableau que la fonction passera à 1.
    - Un flottant qui sera placé dans un nouveau tableau de taille 1.

    La fonction retournera le tableau créé.
2. Créez une fonction permettan d'ajouter un element à la fin du tableau. La fonction acceptera les arguments suivants:
    - Un pointeur vers la taille du tableau pour la mettre à jour.
    - Un pointeur vers le tableau qui sera modifié pour pointer vers un nouveau tableau plus grand.
    - Un flottant qui sera placé à la fin du tableau

La fonction devra allouer un nouveau tableau, copier les éléments de l'ancien vers le nouveau et ajouter le nouvel element à la fin avant de libérer l'ancien tableau et de modifier le pointeur passé en argument pour qu'il pointe sur le nouveau tableau.

3. Créez une fonction qui permet de libérer la mémoire allouée par le tableau.
4. 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. Données sensibles

Ci-dessous le code d'une application qui permet à un utilisateur de se connecter avec son nom, de renseigner des informations et de les afficher. L'application permet aussi de se déconnecter, effacant ainsi toutes les données entrées par l'utilisateur.

1. *(Partie plus facile)* L'application contient un bug qui permet à un utilisateur de lire les données de l'utilisateur précédement connecté.
    - Exploitez ce bug pour lire les données d'un autre utilisateur.
    - Corrigez le bug.
2. *(Difficile, n'hésitez pas à demander de l'aide)* La fonction `get_str_value` contient un défaut d'implémentation. Ce défaut permet à un utilisateur de lire une partie des données entrées par l'utilisateur précédent.
    - Exploitez ce bug.
    - Corrigez le.

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

// cette fonction demande à l'utilisateur de taper plusieurs charactères et les enregistre dans la mémoire si ce sont des charactères ASCII affichables
// Elle donne aussi l'option de réutiliser un bloc mémoire déjà alloué
char* get_str_value(char* value_name, int count, char* reuse_memory){
  // Si la mémoire passée en argument est valide, on la réutilise, sinon on en alloue une nouvelle
  char* ret = reuse_memory ? reuse_memory : malloc(sizeof(char) * 4);
  printf("Entrez %d characteres pour %s\n", count - 1, value_name);
  for (int i = 0; i < count - 1; i++){
    char next_char;
    scanf(" %c", &next_char);
    if (isprint(next_char)){
      ret[i] = next_char;
    }
  }
  ret[count - 1] = 0;
  return ret;
}


int main(){
    char* nom = NULL;
    char* sensitive_data = NULL;
    int numero_secu = -1;
```

```
    for (;;){
        nom = get_str_value("votre nom", 4, nom);
        printf("Bienvenue %s\n", nom);
        while (1){
            printf("Tapez:\n\ts pour renseigner votre numéro de sécu\n\td pour
renseigner des données personnelles\n\ti pour afficher vos informations\n\tq pour
vous déconnecter\n");
            char response;
            scanf(" %c", &response);
            if (response == 's'){
                printf("Quel est votre numéro de sécu ?");
                scanf("%d", &numero_secu);
            } else if (response == 'd'){
                sensitive_data = get_str_value("vos données sensibles", 30,
sensitive_data);
            } else if (response == 'i'){
                printf("Vous êtes : %s\n", nom);
                if (numero_secu > -1){
                    printf("Votre numero de secu est %d\n", numero_secu);
                }
                if (sensitive_data){
                    printf("Vos données sensibles sont %s\n", sensitive_data);
                }
            } else if (response == 'q'){
                printf("Au revoir\n");
                numero_secu = -1;
                free(nom);
                nom = 0;
                break;
            }
        }
    }
}
```