Université Aboubekr BELKAID
Faculté des Sciences
Département d'informatique

Module : Compilation **Année universitaire 2021-2022**
Classe : L3- Informatique
Réalisé par : Mr MERZOUG Mohamed
Mr ETCHIALI Abdelhak

TP N° 02

Exercice 1 : VERIFICATION DE LOGIN ET MOT DE PASSE

Après être employé à une boite de développement, votre chef vous demande de créer un programme de vérification du login / mot de passe. Ce programme sera utilisé seulement pour vérifier le *login / mot de passe* lors de la première création d'un compte dans un site web (comme Gmail, Yahoo, etc).

Ainsi ce programme doit vérifier :

- Que le *login* commence par une lettre alphabétique ;
- Le *login* doit contenir que des caractères de nature alpha-numérique ;
- Le *mot de passe* doit contenir au moins un caractère majuscule ;
- La taille du *mot de passe* doit être supérieure à 10 ;
- *Le mot de passe doit être différent du login ;*

Par exemple : le login "*2med*" n'est pas valide car ce login commence par un chiffre.

(par contre "*med2*" est valide).

Le mot de passe "*mdpkst2018*" n'est pas valide car celui-ci ne contient pas un caractère majuscule.

Question : Ecrire un programme qui permet de lire le login / mot de passe et de vérifier les conditions précédentes.

Exercice 2 : DECOUPAGE D'UNE PHRASE

Ecrire un programme qui permet de découper une phrase et d'enregistrer les mots de la phrase dans une ***Liste Chaînée (ou File).***

Les mots de la phrase peuvent être séparés par des espaces ou par des séparateurs (, ; . ! ? -)

Indice : Créer une fonction qui teste si un caractère appartient à la liste des caractères séparateurs pour découper une phrase.

Rappel :

1. Structure d'une liste chaînée :

```
typedef struct Mot_De_La_Phrase {
        char     Mon_mot[20];
        struct   Mot_De_La_Phrase*  Adresse_Mot_suivant;
} Mot;
```

2. Insertion à la fin de la liste chaînée :

```
Mot* Inserer_a_la_fin(Mot* les_mots_de_laphrase, char* mot){
    // si la liste chainee est vdie
    if(les_mots_de_laphrase==NULL){
            // cration de la liste / reservation de memoire pour le 1er element
            les_mots_de_laphrase=malloc(sizeof(Mot));
            // c'est le 1er et dernier element (donc le prochain element est NULL)
            les_mots_de_laphrase->Adresse_Mot_suivant=NULL;
            // copier le mot
            strcpy(les_mots_de_laphrase->Mon_mot,mot);
            // retourner l'addresse du 1er element
            return les_mots_de_laphrase;
    }
    // si la liste chainee n'est pas vide affecter le 1er element a un pointeur temporaire
    Mot* dernier_mot=les_mots_de_laphrase;
    // tanque le pointeur suivant n'es pas NULL (tanqu'on est pas arrivé au dernier
    while(dernier_mot->Adresse_Mot_suivant!=NULL){
            // passer a l'element suivant
            dernier_mot=dernier_mot->Adresse_Mot_suivant;
    }
    // creer un nouvel element a la fin de la liste
    dernier_mot->Adresse_Mot_suivant=malloc(sizeof(Mot));
    // copier le mot
    strcpy(dernier_mot->Adresse_Mot_suivant->Mon_mot,mot);
    // c'est le 1er et dernier element (donc le prochain element est NULL)
    dernier_mot->Adresse_Mot_suivant->Adresse_Mot_suivant=NULL;
    // retourner l'addresse du 1er element
    return les_mots_de_laphrase;
}
```

3. Insertion à la fin de la liste chaînée :

```
Mot* Inserer_au_debut(Mot* les_mots_de_laphrase,char* mot){
    // creer un nouvel element
    Mot* premier_mot=malloc(sizeof(Mot));
    // copier la chaine de caractere
    strcpy(premier_mot->Mon_mot,mot);
    // affecter le pointeur du suivant au debut de le liste principale
    premier_mot->Adresse_Mot_suivant=les_mots_de_laphrase;
    // retourner l'@ du 1er element
    return premier_mot;}
```

Exercice 3: PILE ET LECTURE DE FICHIER

Ecrire un programme permettant de :

- Lire le texte contenu d'un fichier texte.
- Empiler chaque ligne dans une pile.

Fonctions utiles

A- Fonctions d'entrées/sorties pour les chaînes (stdio.h) :

- **gets(char*)** lecture d'une chaîne sur stdin.
- **puts(char*)** affiche, sur stdout, la chaîne de caractères puis positionne le curseur en début de ligne suivante.
- Dans **<conio.h>**, on trouve les fonctions de base de gestion de la console :
- **putch(char)** : Affiche sur l'écran (stdout) le caractère fourni en argument, cette fonction rend le caractère affiché ou EOF en cas d'erreur.
- **getch(void)** : Attend le prochain appui sur le clavier, et rend le caractère qui a été saisi. L'appui sur une touche se fait sans écho.
- **getche(void)** : Idem getch mais avec écho.
- **getchar(void)** fonctionne comme getche, mais utilise le même tampon que scanf.

B- Fonctions utiles à la manipulation de chaînes (string.h) :

- **int strlen(chaîne)** donne la longueur de la chaîne (\0 non compris)
- **char *strcpy(char *dest,char *src)** recopie la source dans la destination, rend un pointeur sur la destination.
- **char *strncpy(char *destination,char *source,int max)** idem que strcpy mais s'arrête au \0 ou au max caractères lus ;
- **char *strcat(char *dest,char *src)** concatène la source à la suite de la destination, rend un pointeur sur la destination
- **char *strncat(char *destination,char *source,int longmax)** idem que strcat mais s'arrête au \0 ou au max caractères lus ;
- **int strcmp(char *str1,char*str2)** rend 0 si str1==str2, <0 si str1<str2, >0 si str1>str2. Idem strncmp

C- Fonctions de conversions entre scalaires et chaînes (stdlib.h) :

- **int atoi(char *s)** traduit la chaîne en entier (s'arrête au premier caractère impossible, 0 si erreur dès le premier caractère) (voir aussi **atol** et **atof**)

D- Fonctions limitées au caractères (ctype.h) :

- **int isdigit(int c)** rend un entier non nul si c'est un chiffre ('0' à '9'), 0 sinon
- de même : **isalpha (int c)** (c de A à Z et a à z, mais pas les accents),
- **isalnum (int c) (isalpha|isdigit), islower (minuscule), isupper, isspace (blanc, tab, return...), isxdigit (0 à 9,A à F,a à f)**...
- **int toupper(int c)** rend A à Z si c est a à z, rend c sinon. (voir aussi **tolower(int c)**) ;

E- Fonctions de la gestion dynamique de mémoire (alloc.h)

- **void *malloc(int taille) :** réserve une zone mémoire contiguë de taille octets, et retourne un pointeur sur le début du bloc réservé. Retourne le pointeur NULL en cas d'erreur (en général car pas assez de mémoire).
- **void *calloc(int nb, int taille)** : équivalent à malloc(nb* taille).
- **void free(void *pointeur)** libère la place réservée auparavant par malloc ou calloc. Pointeur est l'adresse retournée lors de l'allocation.