Université Grenoble Alpes **Année 2025/2026**
L3 MIAGE

Introduction au Système

Planche de TP n°2

Exercice 1. *Exercice rappel de la mise à niveau (ex.12) :*
*- **si vous n'aviez pas fait cet exercice** : faites-le avec soin*
*- **si vous aviez fait cet exercice** : reprenez-le ici, puis faites vérifier à votre encadrant que votre solution et votre compréhension sont correctes…*

Soit le code ci-dessous :

```
#include <stdio.h>
#define N 50

char *tableau_rempli (char c) {
  char tabl[N];
  int i;
  for (i=0; i < N; i++)
    tabl[i] = c;  // tous les elements recoivent le caractere c
  return tabl;
}

int main(void) {
  char x;
  char *t;
  int j;
  printf("Quel caractere met-on dans le tableau ? ");
  scanf("%c", &x);
  t = tableau_rempli(x);
  printf("Voici le tableau rempli : \n");
  printf(" [ ");
  for (j=0; j < N; j++) {
    printf("%c ", t[j]);
  }
  printf(" ] \n");
  return 0;
}
```

1. Analyse : si on exécute en saisissant le caractère 'A', **que vous attendez-vous** à avoir comme comportement pour ce programme ?

2. Test : saisir ce code dans un fichier exo1.c, compiler exo1.c, puis **exécuter**. Obtenez-vous ce que vous aviez prévu ? **Pourquoi** ? Expliquer précisément, par ex avec un schéma : où se trouve le tableau (pile ou tas) ? quelle en est la conséquence ?...

3. Que devez-vous corriger dans la fonction *tableau_rempli* pour obtenir le comportement attendu, et **pourquoi** : où se trouve le tableau (pile ou tas) ? quelle en est la conséquence ?...

Conclusion : que devez-vous retenir ici ?

Exercice 2.
Reprendre le programme de l'exercice 1 (corrigé !) comme suit :
- la commande produite **devra recevoir en arguments** (dans cet ordre) :
    * le caractère à insérer dans le tableau,
    * et le nom d'un fichier dans lequel le contenu du tableau sera écrit
- elle **pourra de plus recevoir** (à la fin de la ligne de commande) une **option `-s`** et, si cette option est présente, ce n'est pas le caractère indiqué en argument qui devra être inséré dans le tableau, *mais son suivant dans la table ASCII*. Toute autre option sera invalide.

On prendra soin à la **gestion des arguments** de *main*. De plus **la fonction *tableau_rempli* sera adaptée** : elle recevra un argument supplémentaire *next* (entier) qui, s'il est à 1, indique que le caractère suivant dans la table ASCII est à utiliser.

Pour **l'écriture dans le fichier**, on procédera avec des E/S de haut niveau (structurées), et on se rappellera que la fonction *fopen* positionne la variable *errno* en cas d'erreur.

Voici des exemples d'exécutions possibles de cette commande :

```
$ ./exo2 A
Usage: ./exo2 char filename [ -s ]

$ ./exo2 A myfile.txt -h
Usage: ./exo2 char filename [ -s ]

$ ./exo2 A myfile.txt
$ cat myfile.txt
Voici le tableau rempli :
 [ A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
A A A A A A A A A A A A A A A  ]

$ ./exo2 A myfile.txt -s
$ cat myfile.txt
Voici le tableau rempli :
 [ B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B B
B B B B B B B B B B B B B B B  ]
```

Exercice 3.

Ecrire un **programme** C qui reçoit en arguments les noms de 2 fichiers (un fichier source et un fichier destination) et qui crée le deuxième fichier comme une **copie** du fichier source.
On vérifiera bien sûr que 2 arguments ont bien été reçus.

NOTE. On procédera à la recopie en utilisant des fonctions d'entrée/sortie de bas niveau, en copiant par blocs de 256 octets.
On testera bien sûr avec des fichiers de taille suffisante pour s'assurer de tester la copie itérative…

Exercice 4.

Le programme ci-dessous a pour but de remplir un tableau dynamique de réels, et de calculer le **minimum** de tous les éléments de ce tableau. **Compléter ce programme** comme demandé (bien observer la fonction *main* au préalable).
On devra pouvoir avoir par exemple à l'exécution :

```
$ ./mon_pg_de_min
Combien voulez-vous mettre d'elements dans le tableau ? 6
Saisir tab[0] : 8.5
Saisir tab[1] : 3.8
Saisir tab[2] : 1.9
Saisir tab[3] : 7.3
Saisir tab[4] : 6.6
Saisir tab[5] : 4.7
Plus petit element = 1.900000, a l'indice 2
```

```
#include <stdio.h>
#include <stdlib.h>

// Ecrire ici une fonction "remplir" de remplissage (par saisies au
// clavier) d'un tableau t de taille size

// ...

// Ecrire ici une fonction "calcule_min" qui calcule et renvoie en
// résultat le minimum des éléments d'un tableau t de taille size. Un
// paramètre supplémentaire permettra de ramener également l'indice
```

```
// dans le tableau de ce minimum

// ...

// Compléter la fonction main :
int main(void) {
  float *tab, min;
  int nb, indicemin;

  // Mettre ici les instructions pour la saisie du nombre d'elements nb
  // et le dimensionnement du tableau en conséquence :


  // Remplissage du tableau :
  remplir(tab, nb);

  // Mettre ici l'instruction pour la recherche du min (affecte min avec
  // le résultat de calcule_min, et renvoie l'indice dans indicemin) :


  // Affichage des resultats :
  printf("Plus petit element = %f, a l'indice %d\n", min, indicemin);
  return 0;
}
```

Exercice 5. *(optionnel)*
0. Pour vous guider, voir en annexe une correction d'un script shell qui prend en paramètre un nom de répertoire et qui affiche "utilisateur present" ou "pas d'utilisateur" suivant qu'un utilisateur soit actuellement connecté sur chaque device dans le sous-répertoire `pts` de ce répertoire (TP 1).
Assurez-vous de comprendre les constructions utilisées
(à tester en passant `/dev` en paramètre bien sûr)

1. On souhaite écrire une commande `sauv`, sous forme d'un **script shell**, permettant à l'utilisateur de sauvegarder, dans un répertoire `sauvegarde` sous son répertoire de login, une copie des *fichiers dont les noms sont passés en argument à la commande* `sauv`, seulement si ce sont des fichiers de source C (extension `.c`).
Si le répertoire `sauvegarde` n'existe pas, il sera créé par le script.

Ce script fonctionne de la façon suivante : pour tout fichier dont le nom est passé en argument, pourvu que ce nom soit un identificateur de source C, **si** aucun fichier du même nom n'existe dans le répertoire `sauvegarde`, alors le fichier d'origine y est recopié, et **si** un fichier du même nom existe, le fichier d'origine n'y est recopié **que si** les contenus des deux fichiers sont différents (NB. voir l'utilisation de la commande `diff`).
Attention, une recopie ne peut se faire que si le fichier d'origine est accessible en lecture.

Ecrire le script réalisant ce travail. Afin que l'utilisateur soit en mesure de suivre le travail du script, on affichera les noms des fichiers en cours d'examen et l'action qui est faite, par exemple:

```
** TP1 n'est pas un fichier C, on ne le traite pas
** a.out n'est pas un fichier C, on ne le traite pas
** fic.txt n'est pas un fichier C, on ne le traite pas
** on traite le fichier p1.c
pas de fichier /home/Licence/dupont/sauvegarde/p1.c, on recopie
** on traite le fichier p2.c
p2.c et /home/Licence/dupont/sauvegarde/p2.c sont differents, on recopie
** on traite le fichier p3.c
p3.c et /home/Licence/dupont/sauvegarde/p3.c sont identiques, pas de copie
** tp n'est pas un fichier C, on ne le traite pas
```

Rappel : Pour pouvoir exécuter un script, il faut évidemment rendre le fichier exécutable !

Annexe. Correction d'un script shell qui prend en paramètre un nom de répertoire et qui affiche "utilisateur present" ou "pas d'utilisateur" suivant qu'un utilisateur soit actuellement connecté sur chaque device dans le sous-répertoire `pts` de ce répertoire.
(à tester en passant `/dev` en paramètre bien sûr)

```
#!/bin/bash

if [ $# != 1 ]
then echo Il faut 1 parametre...
elif [ -d $1 ]
  then
     for f in `ls $1/pts`
     do
       if who | grep -q "^[a-z][^ ]*[ ][ ]*pts/$f"
       then echo utilisateur present sur pts/$f
       else echo pas d'utilisateur sur pts/$f
       fi
     done
  else echo $1 "n'est pas un repertoire"
fi
```