Informatique *TD*
Programmation impérative avec C Ipesup

1 – Café Numérique

Vous tenez un café où les clients commandent des "cafés modifié". On souhaite écrire un programme qui gère la note d'un client.

1. Écrire une fonction
   `int prix_cafe(int nb_sucres, bool lait, bool rum, bool glace)`
   qui renvoie le prix (en euro) d'un café, en suivant la règle suivante :
   - un café sans rien ajouter coûte 150 centimes ;
   - chaque sucre ajouté augmente le prix de 10 centimes ;
   - ajouter du lait augmente le prix de 20 centimes ;
   - ajouter du rum augmente le prix de 70 centimes ;
   - ajouter de la glace augmente le prix de 100 centimes
2. Écrire une fonction `int addition(int nb_cafes)` ; qui, pour un client, lit depuis l'entrée standard les informations de chaque café (nombre de sucres, lait ou non) et renvoie le prix total payé par ce client.
3. Écrire un programme complet qui :
   (a) demande à l'utilisateur le nombre de cafés consommés ;
   (b) appelle `addition` ;
   (c) affiche le montant total sous la forme `"Total : X euro"`.

2 – Statistiques sur la Ligne de Commande

On souhaite écrire un petit outil qui additionne des entiers passés sur la ligne de commande.

1. Écrire un programme dont la fonction `main` a pour signature
   `int main(int argc, char **argv).`
   Le programme doit :
   - vérifier que l'utilisateur a fourni au moins un entier en argument ;
   - convertir chaque argument en entier avec `atoi` ;
   - calculer et afficher la somme, la valeur minimale et la valeur maximale, sous la forme :

     ```
     Somme : S
     Min   : m
     Max   : M
     ```

2. En cas d'erreur (aucun entier fourni), le programme doit afficher un message d'erreur sur la sortie d'erreur `stderr` en utilisant `fprintf` et terminer avec un code de retour différent de 0.

3 – Clinique vétérinaire

Vous travaillez dans une clinique vétérinaire qui doit suivre le poids des animaux.

1. Écrire une fonction `void ajuster_poids(int *poids, int delta)` ; qui modifie *en place* le poids pointé par `poids` en lui ajoutant `delta` (qui peut être négatif).
2. Écrire une fonction `void normaliser(int *poids, int n)` ; qui prend un tableau de `n` poids (en kilogrammes) et applique :
   — `+1` kg à tous les poids strictement inférieurs à 5 ;
   — `-1` kg à tous les poids strictement supérieurs à 50.
   On utilisera la fonction `ajuster_poids` et l'arithmétique de pointeurs.
3. Écrire un programme qui lit `n`, alloue dynamiquement un tableau de `n` entiers avec `calloc`, lit les `n` poids, appelle `normaliser`, puis affiche le tableau normalisé.

4 – Atelier de Chaînes

On considère des chaînes de caractères terminées par le caractère nul `'\0'`.

1. Écrire une fonction `size_t my_strlen(const char *s)` ; qui renvoie la longueur de la chaîne `s` (sans utiliser `strlen`).
2. Écrire une fonction `void my_strcpy(char *dest, const char *src)` ; qui copie la chaîne `src` dans `dest` (sans utiliser `strcpy`). On suppose que `dest` est assez grande.
3. Écrire une fonction `char *dupliquer_sans_espaces(const char *s)` ; qui alloue avec `malloc` une nouvelle chaîne contenant tous les caractères de `s` sauf les espaces `' '` et renvoie le pointeur vers cette nouvelle chaîne.
4. Écrire un programme test qui lit une ligne avec `fgets`, appelle `dupliquer_sans_espaces` et affiche la chaîne obtenue, puis libère la mémoire.

5 – Fichier de Notes

Une école stocke les notes des élèves dans un fichier texte. Chaque ligne contient le nom de l'etudiant et une note entière, par exemple :

```
Alexandre 12
```

1. Écrire un programme qui prend en argument de la ligne de commande le nom d'un fichier de notes, l'ouvre en lecture avec `fopen` et :
   — lit toutes les notes avec `fscanf` ;
   — calcule le nombre de notes, la moyenne, la note minimale et la note maximale ;
   — affiche ces informations sur la sortie standard.
2. En cas d'erreur d'ouverture du fichier, afficher un message d'erreur sur `stderr` et terminer le programme avec un code de retour non nul.
3. Modifier le programme pour qu'il crée un deuxième fichier (dont le nom est fourni en second argument sur la ligne de commande) et y écrive uniquement les notes strictement supérieures à la moyenne, une par ligne.

6 – Carte de Températures

On modélise une carte de températures par une matrice d'entiers de taille `n` × `m`. On souhaite la stocker dans un fichier texte sous la forme :

```
n m
t11 t12 ... t1m
...
tn1 tn2 ... tnm
```

1. Écrire une fonction qui lit depuis un fichier ouvert en lecture un couple `(n,m)`, puis alloue dynamiquement une matrice `int **mat` de taille `n` × `m` en utilisant `calloc` comme dans le cours.
2. Écrire une fonction `int somme_ligne(int **mat, int m, int i)` ; qui renvoie la somme des éléments de la ligne `i`.
3. Écrire une fonction qui affiche, pour chaque ligne, la moyenne des températures sur cette ligne, en écrivant le résultat dans un deuxième fichier ouvert en écriture.
4. Écrire un programme complet qui :
   (a) lit les noms des fichiers d'entrée et de sortie sur la ligne de commande ;
   (b) lit la matrice depuis le fichier d'entrée ;
   (c) écrit les moyennes dans le fichier de sortie ;
   (d) libère toute la mémoire allouée.

7 – Journal de Bord des Vols

On veut gérer un journal de bord de vols d'avion. Chaque vol est représenté par une structure :

```
struct Vol {
    char code[10];   // ex : "AF1234"
    int h_d //heure_depart;
    int h_a //heure_arrivee;
};
```

Un fichier texte contient une liste de vols, un par ligne, au format : `CODE heure_d heure_a`

1. Écrire une fonction `int lire_vol(FILE *f, struct Vol *v)` ; qui lit un vol depuis le fichier `f` dans la structure pointée par `v` et renvoie 1 si la lecture a réussi, 0 sinon (fin de fichier).
2. Écrire une fonction `int duree(const struct Vol *v)` ; qui renvoie la durée du vol en minutes (on suppose que les heures sont données en minutes depuis le début de la journée).
3. Écrire un programme qui lit tous les vols d'un fichier passé sur la ligne de commande et affiche :
   — le nombre total de vols ;
   — la durée moyenne d'un vol ;
   — le code du vol le plus long.

8 – Bibliothèque Arithmétique

On souhaite écrire une petite bibliothèque arithmétique.

1. Créer un fichier d'en-tête `arith.h` contenant :
   — une garde d'inclusion ;
   — la déclaration d'une fonction `int arith_power(int x, int n)` ; qui calcule x^n pour $n \geq 0$;
   — la déclaration d'une fonction `int arith_gcd(int a, int b)` ; qui calcule le pgcd de a et b (algorithme d'Euclide).
2. Écrire un fichier `arith.c` qui inclut `"arith.h"` et définit les deux fonctions.
3. Écrire un fichier `main.c` qui :
   — lit deux entiers `x` et `n` depuis la ligne de commande ;
   — affiche `x^n` en utilisant `arith_power` ;
   — lit ensuite deux entiers `a` et `b` depuis l'entrée standard ;
   — affiche leur pgcd en utilisant `arith_gcd`.
4. Donner les commandes de compilation séparée permettant d'obtenir un exécutable `arith-main`.

9 – Ensemble d'Entiers

On souhaite concevoir une bibliothèque `set` pour manipuler des ensembles d'entiers, en cachant leur représentation.

1. Écrire un fichier d'en-tête `set.h` qui :
   — déclare un type incomplet `struct Set` et un alias `typedef struct Set set;` ;
   — déclare les fonctions suivantes :

```
set *set_create(void);
void set_add(set *s, int x);
bool set_mem(set *s, int x);
void set_remove(set *s, int x);
int set_card(set *s);
void set_delete(set *s);
```

2. Proposer, en commentaire, une représentation possible de la structure `Set` (par exemple tableau dynamique, liste chaînée, etc.).
3. Écrire un fichier `set.c` qui inclut `"set.h"` et qui définira plus tard les fonctions (on peut laisser les corps vides dans un premier temps).
4. Écrire un fichier `main.c` qui utilise la bibliothèque :
   — crée un ensemble ;
   — lit des entiers depuis l'entrée standard jusqu'à la fin de fichier ;
   — ajoute chaque entier lu dans l'ensemble ;
   — affiche finalement la cardinalité de l'ensemble (nombre de valeurs distinctes) ;
   — supprime l'ensemble.