

Calculatrices, téléphones mobiles et tout appareil électronique non autorisé doivent être éteints et déposés avec vos affaires personnelles.

Seul document autorisé : une feuille au format A4 avec, au recto, le résumé de la syntaxe C++ de la fiche de TD 2 et, au verso, des notes manuscrites. Pour les étudiants inscrits en Français Langue Étrangère, un dictionnaire est autorisé.

Les exercices sont indépendants les uns des autres ; il n'est pas nécessaire de les faire dans l'ordre ; ceux marqués d'un ♣ sont plus difficiles mais font partie du barème sur 100. Le barème est indicatif et sujet à ajustements.

Les réponses sont à donner, autant que possible, sur le sujet ; sinon, demander un intercalaire et mettre un renvoi.

Les enseignants collecteront votre copie à votre place.

Exercice 1 (Cours : compilation séparée (10 points)).

On souhaite implanter une bibliothèque `bla`, utilisée par un programme `programme.cpp`.

(1) Lister ci-dessous les fichiers requis pour la bibliothèque :

—
—
—

(2) Indiquer le rôle de chacun des extraits de code ci-dessous et le ou les fichiers le contenant :

— Rôle : _____, fichier(s) :

```
/** Calcule ...
**/
```

— Rôle : _____, fichier(s) :

```
int bla(int a, int b);
```

— Rôle : _____, fichier(s) :

```
int bla(int a, int b) {
    return a + b;
}
```

— Rôle : _____, fichier(s) :

```
CHECK( bla(1, 2) == 3 );
```

— Rôle : _____, fichier(s) :

```
#include "bla.h"
```

— Rôle : _____, fichier(s) :

```
int c = bla(3, 4);
```

(3) Donner les commandes pour compiler la bibliothèque `bla` et le programme séparément puis les lier entre eux :

Exercice 2 (Plus grand commun diviseur et l'indicatrice d'Euler : fonctions, boucles (20 points)). Le Plus Grand Commun Diviseur (pgcd) de deux nombres entiers x, y non nuls est définie comme le plus grand entier qui les divise simultanément. Par exemple :

$$\text{PGCD}(3, 5) = 1, \text{PGCD}(2, 6) = 2, \text{PGCD}(15, 10) = 5.$$

Dans cet exercice, vous êtes autorisé à utiliser sans les définir les fonctions `min` et `max`, qui reçoivent deux entiers et renvoient leur minimum et leur maximum respectivement.

- (1) Implanter en utilisant des boucles la fonction `pgcd`, dont la documentation et l'entête sont donnés ci-dessous :

Indications : On rappelle que pour tous entiers a et b , le reste de la division euclidienne de a par b est donné en C++ par `a%b`. Ce reste est nul si et seulement si a est divisible par b .

```
/** Calcul le plus grand commun diviseur de deux entiers
 * @param x: un entier
 * @param y: un entier
 * @return le pgcd de x et y
 **/
int pgcd(int x, int y) {
```

- (2) On peut montrer que le pgcd a une définition récursive donnée par :

$$\text{PGCD}(x, y) = \begin{cases} \text{PGCD}(x - y, y) & \text{si } x > y \\ \text{PGCD}(x, y - x) & \text{si } y > x \\ x & \text{si } x = y \end{cases}$$

Écrire une fonction qui utilise cette récursion pour calculer le pgcd :

```
/** Calcul le plus grand commun diviseur de deux entiers de façon recursive
 * @param x: un entier
 * @param y: un entier
 * @return le pgcd de x et y
 **/
int pgcdRecuratif(int x, int y) {
```

(3) Il existe une autre définition récursive donnée par :

$$\text{PGCD}(x, y) = \begin{cases} \max(x, y) & \text{si } \min(x, y) = 0 \\ \text{PGCD}(\min(x, y), \max(x, y) \% \min(x, y)) & \text{sinon} \end{cases}$$

Écrire une fonction qui utilise cette définition récursive pour calculer le pgcd.

```
/** Calcul le plus grand commun diviseur de deux entiers de façon recursive
 * @param x: un entier
 * @param y: un entier
 * @return le pgcd de x et y
 **/
int pgcdRecuratif2(int x, int y) {
```

(4) La fonction indicatrice d'Euler φ d'un entier x , est définie comme la quantité des nombres naturels y , plus petits que x , telles que $\text{PGCD}(x, y) = 1$. En d'autres termes :

$$\varphi(x) = \#\{y < x \text{ telles que } \text{PGCD}(x, y) = 1\}.$$

Par exemple $\varphi(2) = 1$, $\varphi(9) = 6$, $\varphi(17) = 16$.

Utiliser la fonction pgcd pour écrire une fonction qui calcule l'indicatrice d'Euler d'un entier donné :

```
/** Calcul l'indicatrice d'Euler d'un entier
 * @param x: un entier
 * @return l'indicatrice d'Euler de x
 **/
int indicatrice(int x) {
```

Exercice 3 (Fichiers, tableaux (30 points)).

Une équipe de foot veut acheter des maillots pour ses joueurs. Les tailles des maillots vont de 0 à 4. On dispose d'un fichier `maillots-joueurs.txt` dont chaque ligne contient : le nom d'un joueur (en un mot) suivi de sa taille (de 0 à 4). Voici un exemple de fichier `maillots-joueurs.txt` :

```
Mbappé 2
Pogba 4
Griezmann 1
Lloris 4
Kanté 0
Benzema 3
Pavard 0
Giroud 4
```

(1) Considérons la fonction `mystere` suivante :

```
vector<string> mystere(string abc) {
    ifstream zut;
    zut.open(abc);
    string hop;
    int plop;
    vector<string> toto;
    toto = vector<string>(0);
    while ( zut >> hop and zut >> plop ) {
        toto.push_back(hop);
    }
    zut.close();
    return toto;
}
```

Deviner ce que cette fonction est sensée faire, proposer des noms informatifs pour elle-même et pour ses variables :

- `mystere` :
- `abc` :
- `zut` :
- `hop` :
- `plop` :
- `toto` :

et écrire sa documentation :

(2) Écrire un test pour la fonction de la question précédente. On pourra s'inspirer des tests écrits pour les questions suivantes.

(4) Implanter une fonction dont le test, la documentation et l'en-tête sont donnés ci-dessous :

```
CHECK( nomsParTaille("maillots-joueurs.txt") ==  
      vector<vector<string> >({{"Kanté", "Pavard"},  
                              {"Griezmann"},  
                              {"Mbappé"},  
                              {"Benzema"},  
                              {"Pogba", "Lloris", "Giroud"}})));
```

```
/** Donne la liste des noms de joueurs pour chaque taille de maillot  
 * @param le nom d'un fichier qui contient une liste des joueurs  
 *       et leur taille de maillots, au format <nom taille>  
 * @return un tableau à deux dimensions t tel que t[i] contient  
 *        la liste des joueurs qui portent un maillot de taille i  
 **/  
vector<vector<string> > nomsParTaille(string nomFichier) {
```

```
}
```


(3) Proposer des tests pour cette fonction (en utilisant CHECK).

(4) Implanter une fonction `sommeLigne` qui prend en paramètres un entier l et une grille et qui renvoie la somme des éléments de la l -ième ligne de la grille. Par exemple, l'appel sur l'exemple `sommeLigne(carreMagique, 1)` renvoie $1 + 5 + 9 = 15$. Le cas où l'utilisateur demanderait un l non valide ($l \leq 0$ ou $l \geq n$) doit être traité.

Dans la suite, on supposera que l'on dispose aussi d'une fonction `sommeColonne` analogue.

(5) Spécifier et implanter une fonction `sommeDiagonaleMajeure` qui prend en paramètre une grille carrée et renvoie la somme des éléments de la diagonale majeure de la grille (la *diagonale majeure* est celle qui commence en haut à gauche et termine en bas à droite ; dans l'exemple ci-dessus, elle contient 6, 5 et 4).

- (3) ♣ Implanter une fonction `histogramme` qui prend en paramètre une grille `g` et qui renvoie son histogramme, c'est-à-dire un tableau `H` de n^2 entiers (avec n le nombre de lignes de `g`) tel que pour tout v entre 1 et n^2 , `H[v-1]` contient le nombre d'occurrences de la valeur v dans la grille `g`.
- (4) ♣ Implanter une fonction `estCarreMagiqueNormal` qui prend en paramètre une grille, et renvoie `true` s'il s'agit d'un carré magique normal, `false` sinon.
Indication : on pourra par exemple construire l'histogramme de la grille.