

---

# CRO - 3TC+3TCA - examen 2022-2023

---

*Durée : 2h heures, tous documents autorisés*

## Remarques :

- Le barème est donné à titre indicatif (il pourra être modifié éventuellement).
- Pour évaluer les programmes C, on prendra en compte (dans l'ordre décroissant d'importance) : la validité de l'algorithme, la validité syntaxique du programme, la clarté du programme et les commentaires, la forme générale de la présentation.

## 1 Compilation (3 pts)

1. Expliquer (en moins de 10 lignes) à quoi correspondent les deux étapes de production d'un exécutable à partir d'un fichier source C : compilation et édition de lien.
2. Écrire le fichier Makefile pour le projet C comportant les fichiers suivants :
  - deux fichiers de fonctions C : `serveur.c` et `client.c` (ainsi que leur fichiers `.h` associés)
  - Un fichier principal : `testSocket.c` contenant une fonction `main()`

## 2 Quelques exercices simples (6pts)

### 2.1 Minimum d'un tableau

Écrire une fonction `minTableau` :

```
int minTableau(int tab[],int N)
```

qui prend en entrées un tableau `tab` d'entiers et la taille (`N`) de ce tableau et qui renvoie le plus petit élément du tableau.

### 2.2 Entier premier

Écrire une fonction `premierQ` :

```
int premierQ(int n)
```

qui renvoie 1 si l'entier `n` passé argument est premier, 0 sinon. On rappelle qu'un entier est premier si il a exactement deux diviseurs entiers distincts : 1 et lui-même (donc 1 n'est pas premier, mais 3, 5 et 7 le sont). L'opérateur modulo en C est `%` : `n % i` vaut zero si et seulement si `i` est un diviseur entier de `n`.

### 2.3 Tableau dynamique

Écrire une fonction `tabEntierPair` :

```
int *tabEntierPair(int n)
```

qui renvoie un pointeur sur une zone mémoire de `n` entiers, c'est à dire un tableau de taille `n`. Ce tableau sera alloué dans la fonction `tabEntierPair` et il sera rempli avec les `n` premiers entiers pairs. Par exemple, l'appel à la fonction `tabEntierPair[5]` renverra un pointeur sur un tableau de 5 entiers contenant les entiers : 2, 4, 6, 8, 10.

### 3 Produit de matrices (4pts)

On rappelle la formule pour le calcul des coefficients  $c_{ij}$  d'une matrice d'entiers  $C$  de taille  $N \times N$  qui est le produit de deux matrices  $A$  et  $B$  de taille  $N \times N$  :  $C = A \times B$

$$c_{ij} = \sum_{k=1}^N (a_{ik} * b_{kj})$$

Dans cette question, on utilise un tableau à deux dimensions pour stocker une matrice d'entiers de taille  $N \times N$  (ici  $N$  vaut 256), on va utiliser le type `MATRICE` :

```
#define N 256
typedef int MATRICE[N][N];
```

- Écrire une fonction `void prodMat(MATRICE A, MATRICE B, MATRICE C)` qui calcule le produit  $C = A \times B$
- Écrire une fonction `void printMat(MATRICE A)` qui affiche (joliement) la matrice `A`.

---

## 4 Arbre et Tas (7pts)

On veut manipuler des tas, qui sont des arbres binaires dans lesquels chaque noeud a une valeur supérieure à celle de son fils droit et de son fils gauche. La figure 1 montre un exemple de tas. Dans cet exercice on ne se préoccupera pas du fait que l'arbre soit équilibré ou quasi complet, on s'assure juste qu'un noeud ait une valeur supérieure à ses fils. On utilise la structure de donnée suivante pour représenter un tas, chaque noeud stockant un entier :

```
typedef struct cell {  
    int val;  
    struct cell *fDroit;  
    struct cell *fGauche;  
} CELL;  
  
typedef CELL *ARBRE;
```

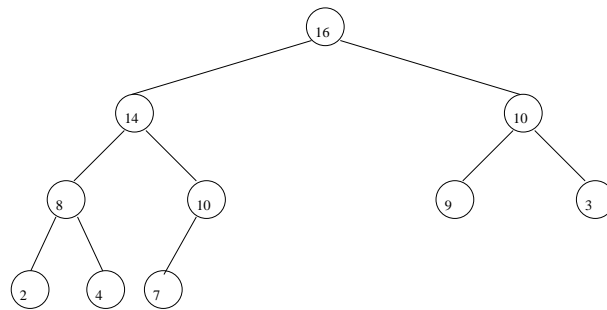


FIGURE 1 – Exemple de graphe qui a la structure de tas

1. Ecrire une fonction `nouvelArbre` :  
`ARBRE nouvelArbre(int val, ARBRE fg, ARBRE fd)`  
qui crée un arbre dont la racine contient la valeur `val` et dont `fg` et `fd` sont respectivement les fils gauche et fils droit. On ne se préoccupera pas du fait que cet arbre soit un tas ou non.
2. Écrivez une fonction `rechercheTas` :  
`int rechercheTas(ARBRE root, int valeur)`  
qui recherche la valeur `valeur` dans le tas `root`, renvoie 1 si elle est présente dans le tas, 0 sinon.
3. Écrivez une fonction `profondeurTas` :  
`int profondeurTas(ARBRE root)`  
qui renvoie la profondeur du tas `root`. La profondeur est le nombre maximum de

liens qu'il y a de la racine à une feuille. Par exemple le tas de la figure 1 a une profondeur 3 et un tas limité à une feuille à une profondeur zéro.

4. Écrivez une fonction `insertionTas` :

`ARBRE insertionTas(ARBRE root,int valeur)`

qui insère un nouvel entier `valeur` dans un tas. L'arbre résultant de la fonction est toujours un tas. Le nouveau tas est renvoyé comme résultat de la fonction.

## 5 Algorithme de Bellman (Bonus)

On présente l'algorithme de Bellman pour calculer les plus courts chemins entre un sommet  $S$  et tous les autres sommets d'un graphe orienté  $G$ , avec  $G = (V, E, P)$   $P$  étant le poids des arêtes et  $\Gamma^-(x)$  représentant les prédécesseurs du sommet  $x$ . Contrairement à l'algorithme de Dijkstra, l'algorithme de Bellman fonctionne avec des poids négatifs (mais le graphe ne doit pas avoir de circuit ou la somme des poids est négative). L'algorithme est le suivant :

1. Chaque nœud  $x$  est initialisé à une distance infinie de  $S$  :  $\forall x \in V, x \neq s \text{ } dist_0(x) = \infty$ , et  $dist_0(S) = 0$ .
2. À chaque étape  $i$  on réévalue  $dist$  :

$$dist_i(x) = \text{Min}(dist_{i-1}(x), \text{Min}_{y \in \Gamma^-(x)}(dist_{i-1}(y) + P(xy)))$$

3. On s'arrête au bout de  $|V| - 1$  itérations.

Proposez une structure de donnée pour implémenter cet algorithme et écrivez une fonction qui l'implémente.