TP n°3 – Tableaux en C

1 Échauffement

- **Q1.** Définir le tableau statique suivant : $[5, 6, 2, 12]$.
- **Q2.** Faire la même chose avec un tableau dynamique.
- **Q3.** Définir un tableau dynamique de taille 10 dont les premières valeurs sont 3.5, 2.2, 1.0, 0.1.
- **Q4.** Écrire une fonction `int maximum(int* tab, int n)` qui calcule le maximum d'un tableau dynamique donné en entrée.
- **Q5.** Écrire une fonction `float moyenne(float* tab, int n)` qui calcule la moyenne d'un tableau de flottants.

2 Avec les tableaux dynamiques

- **Q6.** Écrire une fonction `void puissance_trois(int* tab, int n)` qui modifie le tableau en entrée (qui est de taille n) en mettant chacun de ses éléments à la puissance 3.
- **Q7.** Écrire une fonction `range` qui renvoie le tableau $[1; 2; \ldots; n]$ pour un n donné.
- **Q8.** Écrire une fonction `float* copy(float* tab, int n)` qui copie un tableau donné en entrée et renvoie une copie. Dans `main`, on changera l'élément 0 de la copie et on affichera les deux tableaux pour vérifier qu'on a bien deux tableaux indépendants.

On peut représenter un polynôme $P(X)$ par un tableau `tab` en construisant un tableau tel que `tab[i]` est le coefficient correspondant associé à X^i dans P et tel que la taille de `tab` est le degré de P.

Par exemple $3X^3 + 2X + 1$ est représenté par le tableau $[1; 2; 0; 3]$ (l'indice 0 est à gauche).

- **Q9.** Écrire une fonction `applique_polynome` qui prend en entrée un tableau représentant un polynôme P et un flottant x et calcule $P(x)$.
- **Q10.** Combien d'additions et de multiplications effectue votre algorithme ? (Cela va dépendre du degré n de P).

On observe que si on a la décomposition $P(X) = \Sigma_{i=0}^{n} a_i X^i$, alors on peut écrire :

$$P(X) = a_0 + X * (a_1 + X * (a_2 + X * (\ldots(a_{n-1} + X * a_n))))$$

On peut utiliser cet observation pour écrire un algorithme calculant $P(x)$ pour un x donné en utilisant un nombre d'additions et de multiplications qui est une fonction affine de n, le degré de P. Cette méthode est appelée **algorithme de Horner**.

- **Q11.** Écrire une fonction `applique_horner` qui prend en entrée un tableau représentant un polynôme P et un flottant x et calcule $P(x)$ en suivant la méthode de Horner.

3 Chaînes de caractères

- **Q12.** Écrire une fonction qui prend en entrée une chaîne de caractères et compte le nombre de 'e' dans cette chaîne.
- **Q13.** Implémenter la fonction de comparaison de deux chaines de caractères `int strcmp(char* s1, char* s2)`.
- **Q14.** Écrire une fonction qui prend une chaîne de caractères qui représente un nombre décimal et renvoie le flottant qu'elle représente. *Remarque :* l'entrée sera de la forme "34" ou "34,5" (avec ou sans virgule).
  On n'utilisera pas de fonction pré-implémentée, sauf `strlen`

4 Tri par dénombrement

Un des problèmes importants sur les tableaux est comment trier leurs éléments (dans l'ordre croissant ou décroissant). On verra de nombreux algorithmes de tri au cours de l'année, en voici un premier.

Le *tri par dénombrement* peut être utilisé dans le cas où l'on doit trier des entiers positifs, pas nécessairement distincts, dont on connaît un majorant M. Il est efficace si ce majorant n'est pas trop grand par rapport au nombre d'entiers à trier. Son principe est le suivant : on crée un tableau `compteurs` de taille M dont la case i va servir à compter combien de fois la valeur i apaprait. Ensuite, on parcourt les données et pour chaque valeur qui apparaît, on ajoute 1 à la case de `compteurs` qui convient. Avec ces nombres d'occurrences, on peut à la fin reconstruire les données dans l'ordre croissant.

Par exemple pour $t = [3; 6; 7; 1; 10; 1; 3]$, un majorant est $M = 10$ et le tableau des compteurs sera $[0; 2; 0; 2; 0; 0; 1; 1; 0; 0; 1]$. Au final on obtient $t = [1; 1; 3; 3; 6; 7; 10]$.

- **Q15.** Écrire un programme C qui lit au clavier un entier naturel $n \leqslant 42000$ puis n entiers naturels dans $[\![0; 1000]\!]$ et qui affiche sur la sortie standard ces n entiers triés dans l'ordre croissant en utilisant le tri par dénombrement proposé ci-dessus.

5 Avec des matrices

Dans cette section on propose d'implémenter quelques opérations bien connues des matrices carrées, sur des matrices de taille $n \times n$ quelconque.

- **Q16.** Écrire une fonction `int** multiplication_scalaire(int** m, int n, int a)` qui étant donné une matrice m et un entier a, calcule am, c'est à dire calcule une nouvelle matrice dont chaque case (i, j) contient $am_{i,j}$.
- **Q17.** Écrire une fonction `int** somme(int** mat1, int** mat2, int n)` qui calcule la matrice somme des matrices `mat1` et `mat2`.
- **Q18.** Écrire une fonction `int** produit(int** mat1, int** mat2, int n)` qui calcule la matrice produit des matrices `mat1` et `mat2`.
- **Q19.** Écrire une fonction `void transpose(int** mat1, int n)` qui transpose la matrice `mat1` par effet de bord (c'est à dire modifie `mat1` pour qu'elle devienne sa transposée).

6 Le crible d'Ératosthène

Le crible d'Eratosthène est un algorithme permettant de trouver tous les nombres premiers inférieurs ou égaux à N où N est un entier qu'on se fixe à l'avance, **sans divisions ni multiplications**.

L'idée du crible d'Eratosthène consiste à écrire tous les nombres de 2 à N, puis à effectuer le processus suivant :

- Déterminer le plus petit entier n non barré et non encore étudié (on commence à 2).
- Barrer tous les multiples de n sauf n.
- S'il reste des nombres non barrés strictement supérieurs à n, retourner à l'étape 1.
- À la fin de l'algorithme, les nombres non barrés sont les nombres premiers plus petits que N.

Par exemple pour $N = 9$:

- initialisation : 2 3 4 5 6 7 8 9
- étape 1 - n = 2, on barre donc 4, 6 et 8 : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9
- étape 2 - n = 3, on barre donc 6 et 9 (mais 6 est déjà barré) : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~
- étape 3 - n = 5, mais il n'y a pas de multiple de 5 : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~
- étape 4 - n = 7, mais il n'y a pas de multiple de 7 : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~
- étape 5 - il n'y a plus de nombre non barré, on a donc trouvé les premiers inférieurs ou égaux à 9 : 2 3 5 7.

- **Q20.** Écrire l'algorithme du crible d'Ératosthène. Il prendra en entrée un tableau `tab` de $N-1$ booléens, initialement rempli de `true` et modifiera ce tableau pour qu'à la fin `tab[i]==true` si et seulement si $i + 2$ est premier. Une case barrée sera donc représentée par un `false` .
  Remarque : pour barrer tous les multiples de n, il ne faut pas tester si $n|x$ pour tous les x. Ce n'est pas efficace.
- **Q21.** Afficher les nombres premiers entre 1 et 1000.