Exercices: Série 3 - Fonctions et boucles - ICC-C 2025-2026

1. Nombres premiers

Écrire un programme `premier.c` qui va permettre de tester si un nombre est premier.

D'abord, écrire une (très courte) fonction `bool est_divisible(int a, int b)` qui teste si a est divisible par b. Elle renvoie `true` si c'est le cas, et `false` sinon. Appelez-là depuis `main` avec deux valeurs demandées avec `scanf` et testez-là avec quelques valeurs.

Ensuite, écrire la fonction `bool est_premier(int p)` qui teste si p est premier. Elle renvoie `true` si c'est le cas, et `false` sinon. Utilisez la stratégie suivante :

1. Si $p < 2$, alors p n'est pas premier.
2. Sinon, calculer $s = \lfloor \sqrt{p} \rfloor$.
3. S'il existe $b \in [2, s]$ tel que p est divisible par b, alors p n'est pas premier.
4. Sinon, p est premier.

Dans votre fonction `main`, demandez p avec `scanf`. Si p est premier, afficher (par exemple) « 7 est premier. ». Sinon, afficher « 15 n'est pas premier. ».

Testez votre programme avec les nombres $1, 2, 16, 17, 91, 589, 1001, 1009, 151\,189, 1\,299\,827$ et $2\,146\,654\,199$.

Pour rappel, $\lfloor x \rfloor$ est l'arrondi de x à l'entier inférieur. Si x est un `double` positif, on peut le calculer en C avec `(int) x`.

Exemple d'exécution du programme :

```
Entrez un nombre positif : 15
15 n'est pas premier.
```

Petit challenge : pouvez-vous ré-écrire la fonction `est_premier` sans la fonction `sqrt` ? Vous devez malgré tout seulement tester les nombres jusqu'à s ! Indice : remarquez que $s^2 \leq p$ et que $(s+1)^2 > p$.

2. Calcul du cosinus

*Cet exercice correspond à l'exercice n°14, page 32, de l'ouvrage **C++ par la Pratique (PPUR)** (que nous résolvons en C).*

On souhaite calculer $\cos(x)$ nous-mêmes, sans utiliser les fonctions de `math.h`. Pour cela, on va utiliser la formule suivante (avec $x \in [0, 2\pi]$) :

$$\cos(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^{2i}}{(2i)!} = \frac{x^0}{1} - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \cdots$$

Bien sûr, on ne peut pas calculer une infinité de termes dans notre programme. On va donc *approximer* la somme jusqu'à $n \geq 0$.

Écrire une fonction `double factorielle(int n)` qui calculue $n!$ (approximativement, à cause du `double`). Notez que la factorielle grandit très vite. Il n'est déjà pas possible de stocker 13! dans un `int`. Avec un `double`, on peut monter jusqu'à 170! (mais avec des pertes de précision).

Écrire une function `double approx_cos(double x, int n)` qui calcule la *somme partielle* de la série ci-dessus jusqu'à n. Autrement dit, elle calcule

$$\sum_{i=0}^{n} \frac{(-1)^i x^{2i}}{(2i)!}$$

Dans votre fonction `main`, demander x (un réel) et n (un entier) avec `scanf`. Afficher ensuite le résultat de la somme partielle qui approxime le cosinus. Utilisez `printf("%.10lf", ...)` pour afficher 10 chiffres après la virgule.

Exemple d'exécution du programme :

```
Entrez un réel : 1.23
Entrez un entier positif : 7
cos(1.230000) ~= 0.3342377271
```