Atelier 2

Objectifs de formation
— Manipuler les représentations des nombres flottants,
— Exécuter un code numérique sous les 4 modes d'arrondi
— Reconnaitre l'impact des erreurs d'arrondi sur un résultat numérique
— Optimiser une expression algébrique au regard des erreurs d'arrondi

1. : **45 mn**
   Écrire un programme qui prend nombre réel et affiche ses 4 représentation sur 32 bits en fonction des 4 modes d'arrondi et retourne son exposant non biaisé.
   Tester le programme avec $x = 156,7$ et $x = -7,110^{-5}$.

2. : **30 mn**
   — Écrire un programme qui calcule en arithmétique flotante sur 64 bits la suite a_n définiepar $a_0 = \frac{11}{2}, a_1 = \frac{61}{11}$ et
   $$a_{n+2} = 111 - \frac{1130}{a_{n+1}} + \frac{3000}{a_n.a_{n+1}}.$$
   Affichez les 20 premiers éléments de la suite.
   Qu'en déduisez-vous sur la limite de cette suite ?
   — Exécutez le code sous les 4 modes d'arrondi. Regardez a_{14} en particulier. Que constatez-vous ?

3. : **45 mn - activité donnant lieu à un rendu**
   Un automobiliste parcourt un trajet de 200km. Un second parcourt le même trajet mais avec une vitesse de 10km/h supérieur et arrive avec une heure d'avance. On veut calculer le temps de trajet du premier automobiliste.
   — Montrer que ce problème se ramène à une équation du second degré.
   — Écrire un programme qui résout une équation du second degré et l'appliquer au problème précédent.
   — Dans le cas général, les racines de l'équation
   $$ax^2 + bx + c = 0$$

sont données par $x_{1,2} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
Calculer les racines de l'équation

$$3,1 \times x^2 - 21000 \times x + 3,675 = 0$$

en exécutant le code sous les 4 modes d'arrondi. Expliquer les différences.

— Proposer une formulation moins sensible aux erreurs d'arrondi.

4. : **45 mn - Calcul de π par la méthode d'Archimède**

   On considère un polygone régulier à N côtés inscrit dans le cercle unité. Si L_N représente la longueur d'un côté, la circonférence du polygone NL_N approxime la circonférence du cercle 2π, donc $\pi \approx NL_N/2$ pour N suffisamment grand.

   — Prouver que $L_{2N}^2 = 2(1 - \sqrt{1 - L_N^2/4})$.

   — Montrer que cette récurrence est équivalente à :

   $$L_{2N}^2/4 = \frac{L_N^2/4}{2\,(1 + \sqrt{1 - L_N^2/4})}.$$

   — Écrire une fonction `double archimede1(int k)` qui renvoie L_{2^k} (donc $N = 2^k$) avec la première formule de récurrence et une fonction `double archimede2(int k)` qui renvoie L_{2^k} avec la seconde formule de récurrence.

   — Pour un N suffisamment grand (2^{10} à 2^{30}), comparez les approximations de π données par les deux formules de récurrence avec la valeur arrondie `M_PI` de la bibliothèque mathématique.

   — Expliquer la différence.

5. : **optionnelle**

   On désire calculer la somme J_n définie par $J_n = \displaystyle\sum_{i=1}^{i=n} \frac{1}{i}$.

   La valeur de n sera entrée au clavier.

   À titre de comparaison, voici les valeurs exactes tronquées à 16 chiffres significatifs :

N	Valeur
10	2.92896825396825396825
100	5.18737751763962026080
1000	7.48547086055034491265
10000	9.78760603604438226417
100000	12.09014612986342794736
1000000	14.39272672286572363138

- Écrire une fonction `float sumfd(unsigned int n)` qui calcule J_n en simple précision en commençant la sommation avec les termes les plus grands (donc avec $i = 1$).
- Écrire une fonction `float sumfi(unsigned int n)` qui calcule J_n en simple précision en commençant la sommation avec les termes les plus petits (donc avec $i = n$).
- Une fonction `double sumdd(unsigned int n)` qui calcule J_n en double précision en commençant la sommation avec les termes les plus grands (donc avec $i = 1$).
- Une fonction `double sumdi(unsigned int n)` qui calcule J_n en double précision en commençant la sommation avec les termes les plus petits (donc avec $i = n$).
- Écrire une fonction `main` qui teste les différentes fonctions, en particulier en calculant et affichant l'erreur relative entre les quatre résultats différents. Quelle fonction fournit le résultat le plus précis ?