COMMUN A TOUS LES ENONCES :

Il est demandé, pour le problème suivant, de proposer une solution en pseudo-langage, puis une implantation en C++. L'évaluation tiendra compte de la lisibilité de votre réponse (commentaires et indentation).

Sujet : calcul des variations d'un polynôme de degré ≤ 2

Écrire un programme qui lit trois réels a, b et c et qui donne les variations de la fonction $f : x \mapsto ax^2+bx+c$.

On tiendra compte du fait que a (et b, et c) peuvent être nuls. On rappelle que, si $a \neq 0$, le changement de variation de f intervient à l'abscisse $-\frac{b}{2a}$.

Par exemple :
- Si $f(x) = x^2 + 2x + 6$, le programme doit répondre :
  `f est décroissante entre -inf et -1 et croissante entre -1 et +inf.`
- Si $f(x) = 3x + 7$, le programme doit répondre :
  `f est croissante entre -inf et +inf.`
- Si $f(x) = 3$ (fonction constante), le programme doit répondre :
  `f est constante entre -inf et +inf.`

Sujet : calcul de l'image $f(R)$ d'un polynôme f de degré ≤ 2

Écrire un programme qui lit trois réels a, b et c, coefficients de la fonction $f(x) = ax^2 + bx + c$, et qui calcule l'ensemble image $f(R)$, c'est-à-dire l'ensemble des valeurs prises par $f(x)$ lorsque x décrit R.

On tiendra compte du fait que a (et b, et c) peuvent être nuls. On rappelle que, si $a \neq 0$, le changement de variation de f intervient à l'abscisse $x = -\frac{b}{2a}$.

Par exemple :
- Si $f(x) = x^2 + 2x + 6$, le programme doit répondre :
  `[5, +inf[`
- Si $f(x) = -3x^2 + 7$, le programme doit répondre :
  `]-inf, 7]`
- Si $f(x) = 2x + 5$, le programme doit répondre :
  `]-inf, +inf[`
- Si $f(x) = 3$ (fonction constante), le programme doit répondre :
  `3`

Sujet : tester si un triangle de longueurs des côtés données est rectangle

Écrire un programme qui lit trois réels a, b et c, et qui répond si oui ou non un triangle de côtés a, b et c est rectangle. Si au moins l'un des trois réels est négatif alors le triangle n'est pas rectangle.

On rappelle qu'un triangle est rectangle si et seulement si les trois longueurs x, y et z de ses côtés, avec z la plus grande longueur, sont positives ou nulles et vérifient $x^2 + y^2 = z^2$ (théorème de Pythagore).

Par exemple :
- On rentre 1, 2, 3. Le programme répond :
  `Triangle non rectangle`
  car $1^2 + 2^2 \neq 3^2$.
- On rentre 3, 5, 4. Le programme répond :
  `Triangle rectangle`
  car $3^2 + 4^2 = 5^2$.
- On rentre 5, -3, 4. Le programme répond :
  `Triangle non rectangle`
  car $-3 < 0$.

Sujet : tester si un triangle de longueurs des côtés données existe.

Écrire un programme qui lit trois réels a, b et c, et qui répond s'il existe ou non un triangle de côtés a, b et c dans le plan. (Les triangles aplatis sont considérés comme des triangles.)

On rappelle qu'on peut dessiner dans le plan un triangle dont les longueurs des côtés sont x, y et z, où z est la plus grande des longueurs, si et seulement si $z \leq x + y$ (inégalité triangulaire).

Par exemple :
- On rentre 1, 2, 12. Le programme répond :
  `Ce n'est pas un triangle.`
  car $12 > 1 + 2$.
- On rentre 3, 5, 4. Le programme répond :
  `C'est un triangle.`
  car $5 \leq 3 + 4$.

– On rentre 5, -3, 4. Le programme répond :
`Ce n'est pas un triangle.`
car $5 > 4 + (-3)$.

Il est demandé, pour le problème suivant, de :

1. *proposer une solution en pseudo-langage* [env. 15 pts/20] ;
2. *pour la (ou les) boucle(s) introduite(s) dans cette solution, écrire l'invariant correspondant (on ne demande pas de le prouver). Terminaison de l'algorithme ?* [env. 5 pts/20].

Aucune ligne en C++ ne doit donc être écrite. La solution sera rédigée sur cette feuille. L'évaluation tiendra compte de la lisibilité de votre réponse :
- *commentaires (aux endroits où cela semble nécessaire : rôle de chaque variable, boucle, ...) ;*
- *indentation.*

L'énoncé de l'invariant pourrait commencer de la façon suivante : « Si l'on considère la suite $a_1, \ldots, a_n$ des nombres entrés jusqu'à présent, alors telle variable vérifie telle propriété... ».

Écrire un programme qui lit une suite de nombres entiers croissante (au sens large), et qui affiche le nombre de nombres distincts dans cette suite. La suite s'arrête quand l'utilisateur entre un nombre qui est inférieur au précédent.

Par exemple :
- L'utilisateur entre `1, 3, 4, 2` ; la lecture des nombres s'arrête ici car $2 < 4$. La suite considérée est $1, 3, 4$, qui contient trois nombres distincts. Le programme répond : `3`.
- L'utilisateur entre `8, 8, 8` puis un nombre < 8. La suite $8, 8, 8$ ne contient qu'un seul nombre, 8. Le programme répond : `1`.
- L'utilisateur entre `7, 7, 7, 9, 10, 12, 12`, et un nombre < 12. La suite $7, 7, 7, 9, 10, 12, 12$ contient 4 nombres distincts. Le programme répond : `4`.

Écrire un programme qui lit une suite de nombres entiers croissante (au sens large), et qui affiche le nombre qui apparaît le plus souvent dans cette suite. Si plusieurs nombres apparaissent avec la même fréquence, le programme renvoie un des nombres qui apparaît le plus fréquemment.

La suite s'arrête quand l'utilisateur entre un nombre qui est inférieur au précédent.

Par exemple :
- L'utilisateur entre `3, 3, 4, 2` ; la lecture des nombres s'arrête ici car $2 < 4$. La suite considérée est $3, 3, 4$. Le nombre qui apparaît le plus fréquemment dans cette suite est 3 : le programme répond `3`.
- L'utilisateur entre `8, 8, 8` puis un nombre < 8. Le nombre qui apparaît le plus souvent dans la suite $8, 8, 8$ est 8 (c'est d'ailleurs le seul). Le programme répond : `8`.
- L'utilisateur entre `7, 7, 7, 9, 10, 12, 12`, et un nombre < 12. Le nombre qui revient le plus fréquemment dans la suite $7, 7, 7, 9, 10, 12, 12$ est 7. Le programme répond : `7`.
- L'utilisateur entre `7, 7, 8, 9, 9`, et un nombre < 9. Le programme peut alors répondre : `7`, ou bien : `9` (les deux réponses sont acceptées).

Écrire un programme qui lit une suite de nombres entiers positifs ou nuls, et qui affiche `OUI` s'il y a deux nombres consécutifs de cette suite qui sont zéro et un (ou un et zéro), et `NON` sinon.

La suite s'arrête quand l'utilisateur entre un nombre strictement négatif.

Par exemple :
- L'utilisateur entre `3, 4, 1, -2` ; la lecture des nombres s'arrête ici car $-2 < 0$. La suite considérée est $3, 4, 1$. Le programme répond `NON`.
- L'utilisateur entre `1, 0, 8` puis un nombre négatif. Le programme répond `OUI` car la suite $1, 0, 8$ contient un 1 et un 0 consécutifs.
- L'utilisateur entre `8, 0, 1` puis un nombre négatif. Le programme répond `OUI` car la suite $8, 0, 1$ contient un 0 et un 1 consécutifs.
- L'utilisateur entre `3, 0, 1, 0` puis un nombre négatif. Le programme répond `OUI`.
- L'utilisateur entre `1, 4, 0` puis un nombre négatif. Le programme répond `NON` (le 1 et le 0 ne sont pas consécutifs).

Écrire un programme qui lit une suite de nombres entiers positifs, et qui compte le nombre de croissances de cette suite, c'est-à-dire le nombre de termes plus grands (au sens large) que leur prédécesseur (on considère que le premier terme n'est jamais plus grand que son prédécesseur).

La suite s'arrête quand l'utilisateur entre un nombre négatif.

Par exemple :

– L'utilisateur entre 3, 4, 1, -2 ; la lecture des nombres s'arrête ici car $-2 < 0$. La suite considérée est $3, 4, 1$. Le programme répond 1 (seul 4 est un terme plus grand que son prédécesseur).
– L'utilisateur entre 1, 4, 8, 5, 8 puis un nombre négatif. Le programme répond 3 car la suite $1, 4, 8, 5, 8$ a trois termes plus grands que leurs prédécesseurs : le 4 et les deux 8.
– L'utilisateur entre 4, 3, 2, 1, puis un nombre négatif. Le programme répond 0 car aucun terme n'est plus grand que son prédécesseur puisque la suite est strictement décroissante.
– L'utilisateur entre 1, 1, 1 puis un nombre négatif. Le programme répond 2, car le deuxième et le troisième 1 sont des termes plus grands (au sens large) que leur prédécesseur ; le premier terme n'est pas considéré comme un nombre plus grand que son prédécesseur.

Soient `début` et `fin` deux constantes entières avec `début<fin`, I l'ensemble des entiers entre `début` et `fin`. On code une fonction f de I dans I par un tableau `f` indexé par l'ensemble des entiers I, à valeurs dans I. Ainsi, la valeur de $f(i)$ est codée en pseudo-langage par `f[i]`.

L'objet du programme en pseudo-langage suivant est de calculer l'itérée $n^{\text{ième}}$ de la fonction f, c'est-à-dire la fonction $g = f \circ f \circ \ldots \circ f$ (où n fonctions f apparaissent dans la formule précédente).

```
PROGRAMME itérée
    CONSTANTES entières : début = 3, fin = 8
    TYPE TabFonc : tableau d'entiers indexé par [début..fin]

DÉBUT itérée
    VARIABLES :
        n : entier      // le nombre d'itérations
        i : entier      // l'itération courante
        j : entier      // l'indice courant
        f : TabFonc     // la fonction f
        g : TabFonc     // la fonction g, qui est le résultat à la fin

    // lecture du nombre d'itérations
    AFFICHER "Entrez le nombre d'itérations > 0"
    RÉPÉTER
        LIRE n
    JUSQU'À ( n > 0 )

    // initialisation des variables
    INITIALISER f aléatoirement avec des entiers entre début et fin
    COPIER f dans g
    AFFICHER "La fonction f est :"
    AFFICHER f

    // calcul de g
    POUR i <- 2 jusqu'à n FAIRE                     // première boucle
        POUR j <- début jusquà fin FAIRE           // seconde boucle
            g[j] <- f[g[j]]
        FIN POUR                                    // fin seconde boucle
        // invariant première boucle : ici, g contient la i-ème itérée de f
    FIN POUR                                        // fin première boucle

    // affichage du résultat
    AFFICHER "La ", n, "ième itérée de f est"
    AFFICHER g
FIN itérée
```

1 Implantation en C++ [env. 4 pts/5]

Faire une implantation en C++ de ce programme.

La traduction des commentaires est facultative, mais la notation tiendra compte de l'indentation.

La première ligne du programme doit contenir, en commentaire, votre nom et votre prénom. Le programme doit être imprimé à la fin de la séance.

Attention, l'implantation utilise des tableaux de taille fixée au début du programme par des constantes `début` et `fin`. Votre programme doit encore fonctionner si l'on modifie la valeur de ces constantes.

L'exemple ci-dessous est un extrait de programme qui rappelle la syntaxe pour tirer des nombres au hasard :

```
#include<stdlib.h>
#include<time.h>
...
int a,b;
...
srand(time(NULL)); // ligne à ne mettre qu'une seule fois avant toute utilisation de rand()
a = rand() % b;    // a contient maintenant un nombre aléatoire entre 0 et b-1
```

2 Question [env. 1 pt/5]

(Réponse à donner ci-dessous).

L'algorithme fonctionne-t-il encore si l'on remplace la ligne : `g[j] <- f[g[j]]` par : `g[j] <- g[f[j]]` ? Justifiez.

Soient `début` et `fin` deux constantes entières avec `début`$<$`fin`, I l'ensemble des entiers entre `début` et `fin`. On code une fonction f de I dans I par un tableau `f` indexé par l'ensemble des entiers I, à valeurs dans I. Ainsi, la valeur de $f(i)$ est codée en pseudo-langage par `f[i]`.

L'objet du programme en pseudo-langage suivant est de générer une permutation aléatoire de I, c'est-à-dire une bijection de I dans I.

```
PROGRAMME permutation
    CONSTANTES entières : début = 3, fin = 8
    TYPE TabFonc : tableau d'entiers indexé par [début..fin]

DÉBUT permutation
    VARIABLES :
        i : entier      // un indice
        j : entier      // un indice qui va être échangé avec i
        f : TabFonc     // la fonction f

    INITIALISER le tableau f à l'identité (f[i]=i pour tout i)

    POUR i <- début jusqu'à fin  FAIRE          // boucle
        j <- un nombre au hasard entre i et fin (compris)
        SI (j différent de i)
        ALORS
            ÉCHANGER f[i] et f[j]
        FIN SI
    FIN POUR                                    // fin boucle

    // affichage du résultat
    AFFICHER "Une permutation aléatoire est :"
    AFFICHER f

FIN permutation
```

3 Implantation en C++ [env. 3,75 pts/5]

Faire une implantation en C++ de ce programme.

La traduction des commentaires est facultative, mais la notation tiendra compte de l'indentation.

La première ligne du programme doit contenir, en commentaire, votre nom et votre prénom. Le programme doit être imprimé à la fin de la séance.

Attention, l'implantation utilise des tableaux de taille fixée au début du programme par des constantes `début` et `fin`. Votre programme doit encore fonctionner si l'on modifie la valeur de ces constantes.

L'exemple ci-dessous est un extrait de programme qui rappelle la

syntaxe pour tirer des nombres au hasard :

```
#include<stdlib.h>
#include<time.h>
...
int a,b;
...
srand(time(NULL)); // ligne à ne mettre qu'une seule fois avant toute utilisation de rand()
a = rand() % b;    // a contient maintenant un nombre aléatoire entre 0 et b-1
```

4 Questions [env. 1,25 pt/5]

(Réponses à donner ci-dessous).

Que calcule l'algorithme, dans chacun des cas suivants ? Justifiez.

1. On remplace la ligne : `POUR i <- début jusqu'à fin FAIRE`
   par : `POUR i <- début jusqu'à fin-1 FAIRE`.
2. On remplace la ligne : `POUR i <- début jusqu'à fin FAIRE`
   par : `POUR i <- début+1 jusqu'à fin FAIRE`.

L'exercice suivant est à traiter en pseudo-langage (à rendre sur cette feuille) et en C++ (à imprimer en fin de séance, la première ligne du programme devant être un commentaire portant votre nom).

Une *sauterelle* est un objet, assimilé à un point matériel, qui est astreint à se déplacer dans le plan et est doté d'une orientation (direction dans laquelle il pointe). On peut demander à une sauterelle d'*avancer* d'une certaine distance (dans la direction vers laquelle elle pointe), de *tourner à gauche* ou *à droite* d'un certain angle (changer son orientation) ; on veut aussi pouvoir calculer facilement sa *distance à l'origine.*

Une sauterelle est parfaitement définie par trois flottants : ses coordonnées dans le plan x et y, et son angle θ (en radians, entre 0 et 2π) ; voir figure ci-dessous.

On demande d'écrire les fonctions et procédures suivantes, **en prenant garde à utiliser les passages par référence et par valeur à bon escient** :

1. `init` : procédure qui prend en arguments x, y et θ et qui initialise la sauterelle à la position $(0,0)$, orientation 0 radians.
2. `tg` (tourner à gauche) : procédure qui prend en arguments x, y, θ, et l'angle α (toujours en radians), et qui augmente l'angle θ de α.
3. `avance` : procédure qui prend en arguments les trois paramètres x, y, θ de la sauterelle et la distance d qu'elle doit parcourir. Cette procédure change donc x et y en fonction de θ et d.
4. `dist` (distance à l'origine) : fonction qui prend en arguments x, y et θ et qui renvoie la distance de la sauterelle à l'origine.

Le programme principal (à écrire pour tester votre programme !) sera par exemple :

```
#include <iostream.h>
#include <math.h>
const float pi=3.14159265358979;
...
void main()
{
    float x,y,t;          // variables de la sauterelle, t=theta
    init (x,y,t);         // met x,y,t à 0
    avance (x,y,t,10);    // maintenant, x=10, y=0, t=0
    tg (x,y,t, PI/4);     // maintenant, x=10, y=0, t=pi/4
    cout << dist(x,y,t);  // doit renvoyer 10
    cout << endl;
}
```

Pour la troisième question, on notera qu'il faut augmenter x de $d \cdot \cos\theta$ et y de $d \cdot \sin\theta$. En C++, on utilisera les fonctions `cos` et `sin` (qui prennent un argument flottant, en radians, et qui calculent son cosinus et son sinus).

Note : ce problème est inspiré du langage de programmation LOGO, qui permet de réaliser des dessins assez jolis en utilisant récursivement les fonctions décrites dans le sujet et en créant les routines d'affichage de la trajectoire de la sauterelle.

L'exercice suivant est à traiter en pseudo-langage (à rendre sur cette feuille) et en C++ (à imprimer en fin de séance, la première ligne du programme devant être un commentaire portant votre nom).

La *suite de Fibonacci* est la suite $(F_n)_{n\in N}$ définie par $F_0 = F_1 = 1$ et, pour $n \geq 0$, $F_{n+2} = F_{n-1} + F_n$. Ses premiers termes sont : $1, 1, 2, 3, 5, 8, 13, 21, \ldots$

1. Écrire une procédure `etape_fibo` qui prend deux paramètres entiers a et b, et telle que, si on suppose que $a = F_n$ et $b = F_{n+1}$ pour un certain n avant l'exécution de la procédure, alors après son exécution $a = F_{n+1}$ et $b = F_{n+2}$.
2. Écrire une fonction `fibo` qui prend un paramètre entier n et qui retourne l'entier F_n (cette fonction utilisera la procédure `etape_fibo`). Donner l'invariant de la boucle (sans le prouver).
3. On admet que $\frac{F_{n+1}}{F_n}$ tend, quand $n \to +\infty$, vers $\frac{1+\sqrt{5}}{2}$. Écrire une fonction `terme` qui prend en entrée un flottant, ε, et qui renvoie le plus petit entier n tel que $|\frac{F_{n+1}}{F_n} - \frac{1+\sqrt{5}}{2}| < \varepsilon$.
4. Écrire le programme principal qui lit un entier n et affiche F_n, puis qui lit un flottant $\varepsilon > 0$ et qui affiche le résultat de `terme`(ε).

Pour les fonctions et procédures, on prendra garde d'utiliser les passages par référence et par valeur à bon escient.

Attention, pour la question 3, on rappelle que si a et b sont entiers, `a/b` désigne le quotient (entier) de a par b. Par contre, si c est déclaré comme flottant, `c=float(a)/float(b);` calculera $c = \frac{a}{b}$ (division exacte).

On rappelle que la fonction valeur absolue sur les flottants se note `fabs` (avec `math.h`).