```
float lav (float a, float b) {
  if (b < 1.0) {
    return 0.0;
  }
  return a + lav( a, b - 1.0);
}
```

Table 1 – Fonction lav

Programmation Impérative

22/2/2023

Nom _____
Prénom _____
numéro _____
mail _____

Les questions sont à traiter dans l'ordre qui vous plaira mais vous n'aurez peut-être pas le temps de tout faire... Ce sujet doit être remis avec votre copie, une version imprimable sera donnée sur la page du cours pour que vous puissiez retravailler les questions.
Quand vous simulez le fonctionnement de l'ordinateur lors de l'exécution d'une fonction, vous devez donner le contenu de la mémoire pour chacune des variables utilisées et ceci à chaque instant. Pour les fonctions récursives tous les appels doivent être spécifiés.

A) Simples

1. Soit la fonction `yep`, cf. Table 2, donnez toutes les étapes de l'exécution de cette fonction avec comme arguments 3.0 et 5.0.
2. Écrivez une fonction qui fait la somme des cubes de tous les entiers compris entre a et b, ses deux paramètres, bornes exclues.
3. Soit la fonction suivante :

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 2 \\ f(2) &= 4 \\ f(n) &= f(n-1) + f(n-2) - f(n-3) \end{aligned}$$

   – Donnez une fonction permettant de calculer ses valeurs.
   – Donnez une fonction itérative permettant
4. Vous trouverez en table 1 une fonction `lav`, donnez sa forme récursive terminale.

```
int hop (float w) {
  while (w >= 2.0)
    w = w - 2.0;
  if (w >= 0.5)
    return 0;
  return 1;
}
float yep (float x, float y) {
  float res;
  res = 0.0;
  while (y >= 1.0) {
    if (hop (y) != 1)
      res = res + x;
    y = y / 2.0;
    y = (float) ((int) y);
    x = x + x;
  }
  return res;
}
```

Table 2 – Fonction yep

```
typedef struct vect {
  int nbele;
  double t [100];
} vect_t;
```

TABLE 3 – Structure de vecteur

```
vect_t remplir (int nb) {
  double pred, ante;
  vect_t v;
  int i;

  if (nb > 99)
    exit(9);
  v.nbele = nb;
  pred = 13.0;
  ante = 8.0;
  v.t [0] = ante;
  v.t [1] = pred;
  for (i=2; i < nb; i++) {
     v.t [i] = (ante + pred);
     while (v.t[i] > 19.0)
        v.t[i] = v.t[i] - 19.0;
     ante = pred;
     pred = v.t [i];
  }
  return v;
}
```

TABLE 4 – Fonction remplir

B) Vecteurs

Vous trouverez dans la Table 3 une structure de vecteur. De même dans la Table 4 vous trouverez la fonction `remplir`.

1. Faites tourner cette fonction avec comme paramètre 9.
2. Donnez une fonction de ce vecteur qui renvoie la somme des éléments inférieurs ou égaux à une valeur y donnée.
3. Donnez une fonction qui remplace chaque valeur du vecteur par son carré.
4. Donnez une foncion qui prend un tel vecteur en argument et qui intervertit les deux premières valeurs si elles sont dans un ordre croissant.
5. Donnez une fonction qui trie un tel vecteur dans l'ordre décroissant selon l'algorithme **quicksort** vu en cours.

C) Tableau d'altitudes

Nous avons vu, en cours, une structure de tableau d'altitudes, cf Table 5.
Nous dirons qu'une case (x,y) est une **cuvette** si son altitude est inférieure à celle de ses quatre voisins.

1. Soient les deux points P (xp, yp) et Q (xq, yq) donnez une fonction qui renvoie le dénivlé lorsqu'on va de P à Q.

```
tyepdef struct table_alt {
  int xmax;
  int ymax;
  float t[100][100];
} table_alt_t;
```

TABLE 5 – Structure de tableau d'altitudes

2. Donnez une fonction qui calcule la somme des dénivelés négatifs sur la colonne `c`.
3. Soit une case (x,y) donnez une fonction qui renvoie 1 si cette case correspond à une cuvette et 0 sinon.
4. Que se passe-t-il si la case (x,y) est une case de bord ? Adaptez votre fonction précédente pour traiter ces exceptions.
5. Donnez une fonction qui renvoie le nombre de cuvettes de la surface.
6. Donnez une fonction qui renvoie la plus haute cuvette de la surface.