CSC 3002 - Corrigé du Contrôle Final 2

Exercice d'algorithmique (sur 6 points)

Enoncé :

- **données** nb : Naturel
- **résultat** un booléen égal à VRAI si nb est un nombre de Harshad, à FAUX sinon.

Algorithme :

Définition de la fonction *principale* :

```
fonction principale ( ) : CodeRetour

    n : Naturel

    afficher ( "Saisir un entier strictement positif : " )
    saisir ( n )

    afficher ( n, " " )
    si estNombreDeHarshad ( n ) alors
        afficher ( "est" )
    sinon
        afficher ( "n'est pas" )
    fsi
    afficher ( " un nombre de Harshad." )
    allerAlaLigne

    retourner OK

ffct principale
```

Définition de la fonction *estNombreDeHarshad* :

```
fonction estNombreDeHarshad ( nb : Naturel ) : Booléen

    si nb mod sommeChiffres ( nb ) = 0 alors
        retourner VRAI
    fsi
    retourner FAUX

ffct estNombreDeHarshad
```

Définition de la fonction *sommeChiffres* :

```
fonction sommeChiffres ( nombre : Naturel ) : Naturel

    somme = 0 : Naturel // somme des chiffres
```

```
    tant que nombre > 0 faire
        somme ← somme + nombre mod 10
        nombre ← nombre div 10
    ftq

    retourner somme

ffct sommeChiffres
```

Exercice de langage C (sur 4 points)

Affichage produit

```
1 : m = 41, n = 20, *p = 41, *q = 41
2 : m = 17, n = 20, *p = 17, *q = 17, **x = 17
```

Exercice de structures de données (sur 10 points)

1. Définition de la fonction *calculerMoyenneEcartType*

```
void calculerMoyenneEcartType ( Liste l, double *pMoy, double *pET ) {

  int nbValeurs = 0 // nombre de valeurs
  double somme = 0 // somme des valeurs
  double sommeCarres = 0 // somme des carres des valeurs

  while ( !estListeVide ( l ) {

    double val = l->valeur;

    nbValeurs++;
    somme += val;
    sommeCarres += val * val;
    l = l->lien;
  }
  *pMoy = somme / nbValeurs;
  *pET = sqrt ( sommeCarres / nbValeurs - (*pMoy) * (*pMoy) );
}
```

2. Si on stocke dans une structure, en plus du pointeur sur le premier élément de la liste, le nombre d'éléments, la somme des valeurs et la somme des carrés des valeurs, on peut recalculer en en temps constant la moyenne et l'écart-type.

   Redéfinition du type *Liste*

```
typdef struct { int nbElements; // pour stocker le nombre d'elements
                double somme; // pour stocker la somme des valeurs
                double sommeCarres; // pour stocker la somme
```

```
                                    // des carres des valeurs
                Element *liste; } Liste;
```

3. Définition de la fonction *nouvelleListe*

```
Liste nouvelleListe ( void ) {

  Liste liste;

  liste.nbElements = 0;
  liste.somme = 0;
  liste.sommmeCarres = 0;
  liste.liste = NULL;

  return liste;
}

// appel : Liste liste = nouvelleListe ();
```

4. Définition de la fonction *inserer*

```
// prototype de la fonction enlister

Element *enlister ( Element *ptrElement, double val );

Liste inserer ( Liste liste, double val ) {

  liste.nbElements++;
  liste.somme += val;
  liste.sommmeCarres += val * val;
  liste.liste = enlister ( liste.liste, val );

  return liste;
}
```

5. Définition de la fonction *supprimer*

```
// prototype de la fonction delister

Element *delister ( Element *ptrElement, double val );

Liste supprimer ( Liste liste, double val ) {

  liste.nbElements--;
  liste.somme -= val;
  liste.sommmeCarres -= val * val;
  liste.liste = delister ( liste.liste, val );

  return liste;
}
```