LIFAP1 – CC mi-parcours – Séquence 1

Contrôle Continu (Durée totale : 1h)
Mardi 2 novembre 2021

NOM :

……

PRENOM :

……

Numéro Etudiant :

……

Recommandations : *Les documents, calculatrice, téléphone portable sont interdits. La qualité de l'écriture et de la présentation seront prises en compte dans la note finale. Vous veillerez à* ***respecter*** *les notations et les règles d'écriture des algorithmes vues en cours et en TD. Un soin tout particulier devra être apporté à l'écriture des entêtes des différents sous-programmes.*

Partie A – Algorithmique

La suite de Padovan est une suite d'entiers définie par récurrence par $\mathcal{P}_{n+3} = \mathcal{P}_{n+1} + \mathcal{P}_n$ pour tout entier *n* positif.

Les trois valeurs initiales sont : $\mathcal{P}(0) = \mathcal{P}(1) = \mathcal{P}(2) = 1$

1. Écrire l'algorithme d'une **procédure** `saisie_valeur` qui "retourne" une valeur choisie par l'utilisateur strictement supérieure à 2 et inférieure à une constante MAX définie au préalable. La saisie devra être recommencée tant que la valeur ne respecte pas les contraintes.

```
Procédure saisie_valeur (int n)
Précondition : aucune
Donnée : aucune
Donnée / résultat : n
Description : "retourne" une valeur choisie par l'utilisateur strictement supérieure à 2 et inférieure à MAX
Variables locales : aucune
Début
        Faire
                Afficher ("Donnez un entier entre 3 et MAX")
                Saisir (n)
        Tant que (n<3 ou n>=MAX)
Fin
```

2. Écrire l'algorithme d'une **fonction** `terme_padovan` qui calcule le n^{ième} terme de la suite de Padovan, `n` étant passé en paramètre de la fonction, **sans passer par la construction d'un tableau**.

```
Fonction terme_padovan (n : entier) : entier
Précondition : n>=0
Données : n
Données / résultat : aucune
Résultat : entier
Description : calcule et retourne le nième terme de la suite de Padovan
Variables locales : res, p1, p2, p3, i : entiers
Début
        p1← 1
        p2← 1
        p3← 1
        pour i allant de 3 à n par pas de 1 faire
                res ← p2 + p3
                p3 ← p2
                p2 ← p1
                p1 ← res
        fin pour
        retourner res
fin
```

3. **Sans utiliser la fonction précédente**, écrire l'algorithme d'une **procédure** `tableau_padovan` qui remplit un tableau avec les `n` premiers termes de la suite de Padovan `n` étant passé en paramètre. On pourra utiliser la constante `MAX` pour la définition du tableau.

```
Procédure tableau_padovan (T tableau [MAX] d'entiers, n : entier)
Précondition : n>=0
Données : n
Données / résultat : T
Description : remplit le tableau avec les n premiers termes de la suite de Padovan
Variables locales : i : entier
Début
        T[0] ← 1
        T[1] ← 1
        T[2] ← 1
        pour i allant de 3 à n par pas de 1 faire
                T[i]← T[i-2] + T[i-3]
        fin pour
fin
```

4. En utilisant les sous-programmes précédents, écrire l'algorithme du programme principal permettant
   a. de saisir une valeur n comprise entre 3 et MAX,
   b. de calculer le $n^{ième}$ terme de la suite en utilisant respectivement les sous-programmes écrits en 2 et 3,
   c. et d'afficher un message confirmant, après les avoir calculées, que ces deux valeurs sont bien identiques.

```
Programme principal

Début
        val, res_suite, res_tab : entier
        tab_padovan : tableau [MAX] d'entiers
        saisie_valeur(val)
        res_suite ← terme_padovan(val)
        tableau_padovan (tab_padovan, val)
        res_tab ← tab_padovan[val]
        si  (res_tab = res_suite)    alors afficher ("Les deux valeurs sont bien identiques")
                                     sinon afficher ("Problème de codage !")
        fin si
Fin
```

Partie B – Langage C/C++

On dispose d'un tableau recensant les statistiques de performance d'un coureur sur une distance de 10 km. Dans le tableau ci-après, la première ligne représente les kilomètres, la seconde ligne la vitesse moyenne sur ce kilomètre, et la troisième le nombre de calories brûlées sur ce même kilomètre.

Km	1	2	3	4	5	6	7	8	9	10
Vitesse	11.5	12.3	12.3	11.9	13.0	11.7	12.6	12.8	13.5	11.2
Calories	49	55	58	57	64	67	77	80	92	75

1. Définir en C/C++ un tableau pouvant contenir ces informations. On pourra ne mémoriser que les informations des 2 dernières lignes du tableau précédent : vitesse et calories.

```
float tab_perf [2] [10] ;
```

2. Écrire en langage C/C++ un **sous-programme** `extraire_stats` qui en un seul parcours de ce tableau calcule et "retourne" la vitesse moyenne sur la distance totale (`vmoy`), le kilomètre le plus rapide (`fast`), le kilomètre le plus lent (`slow`) et le nombre total de calories brûlées (`nrj`) pendant la séance.

```
void extraire_stats (float tab_perf [2] [10] , float & vmoy , int &fast , int &slow, float & nrj)
{
   int i;
   vmoy = 0;
   fast = 0 ;
   slow = 0 ;
   nrj = 0 ;
   for (i=0 ; i<10;i++)
   {
      vmoy += tab_perf[0][i];
      nrj += tab_perf[1][i];
      if (tab_perf[0][fast]<tab_perf[0][i])
         fast = i;
      if (tab_perf[0][slow]>tab_perf[0][i])
         slow = i;
   }
   vmoy/=10;
   fast++;
   slow++;
}
```

3. On dispose d'une procédure `affiche_tab` qui affiche le contenu d'un tableau 2D de 10 réels et d'une procédure `remplit_tab` qui remplit le contenu d'un tableau 2D de 10 réels. Écrire en C/C++ le programme principal qui remplit le tableau de performances, l'affiche puis affiche les statistiques obtenues en 2.

```
int main (void)
{
  float perf [2] [10];
  float vit_moy, calories ;
  int  plus_lent, plus_rapide;
  remplit_tab (perf) ;
  affiche_tab(perf) ;
  extraire_stats(perf,vit_moy,plus_rapide, plus_lent,calories);
  cout<<"Vitesse moyenne : "<<vit_moy<<endl;
  cout<<"Kilomètre le plus rapide : "<<plus_rapide<<endl;
  cout<<"Kilomètre le plus lent : "<<plus_lent<<endl;
  cout<<"Calories dépensées : "<<calories<<endl;
  return 0 ;
}
```