LIFAP1 – Partie A - Algorithmique

Contrôle Continu Terminal (Durée totale : 2h)

Lundi 14 décembre 2020

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.*

Exercice 1 : Tri par insertion ou tri du joueur de cartes

On souhaite écrire un programme permettant de trier un tableau d'entiers. Chacun des éléments de ce tableau sera inséré dans un nouveau tableau résultat de manière à ce que les éléments restent triés tout au long de l'exécution de l'algorithme. La taille maximale des tableaux notée `MAX` est une constante fixée à 50, et le nombre de cases remplies à chaque étape noté `tailleT` sera passé **en paramètre de tous les sous-programmes**.

Exemple : on dispose du tableau suivant

1	4	6	8	12					

On souhaite ajouter la valeur 11 dans le tableau,

1	4	6	8	11	12				

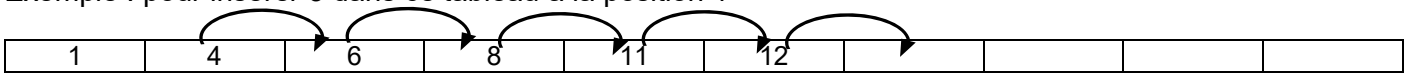
puis la valeur 3.

1	3	4	6	8	11	12			

1. Ecrire l'algorithme d'une **fonction** `trouve_pos` qui à partir d'un tableau d'entiers `T` (dont les `tailleT` premières cases seront remplies) et d'une valeur `val` retournera **l'indice** de la case dans laquelle `val` doit être insérée dans le tableau `T` afin de respecter l'ordre croissant. Dans l'exemple ci-dessus, pour la valeur 11 on aurait l'indice 4 et pour la valeur 3 on obtiendrait l'indice 1 comme résultats.

```
Fonction trouve_pos(tab : tableau [MAX] d'entiers, tailleT : entier,
val :entier) : entier
Préconditions : tailleT<=MAX
Données : tailleT, val
Données / résultat : tab
Resultat : entier
Description : retourne l'indice de la position à laquelle val doit être
insérée dans le tableau
Variables locales : i : entier
Début
     i ← 0
     tant que ( i<tailleT  et tab[i]<val) faire
        i ← i + 1
    fin tant que
    retourner i
 Fin
```

Pour insérer une valeur dans un tableau à un indice donné, il faut procéder à un décalage de toutes les valeurs qui la suivent dans le tableau.
Exemple : pour insérer 3 dans ce tableau à la position 1



Il faut décaler toutes les valeurs à partir de la position 1 puis insérer la valeur à l'indice 1. Attention, `tailleT` aura été incrémenté de 1 au final !!

1	**3**	4	6	8	11	12			

2. Ecrire l'algorithme d'une **procédure** `insere_valeur` qui à partir du tableau `T` contenant `tailleT` valeurs, insère la valeur `val` à l'indice `ind` donnés en paramètres en suivant l'algorithme décrit précédemment.

```
Procédure insere_valeur(tab : tableau[MAX] d'entiers, tailleT entier,
val : entier, indice : entier)
Préconditions : tailleT<MAX, indice>=0, indice<=tailleT
Données : indice, val
Données / résultat : tab, tailleT
Description : insere val dans tab à l'indice indice
Variables locales : i : entier
Début
     pour i allant de tailleT à indice-1 par pas de -1 faire
         tab[i+1] ← tab[i]
     fin pour
     tab[indice] ← val
     tailleT ← tailleT + 1
Fin
```

3. En utilisant les sous-programmes écrits en 1 et 2, écrire l'algorithme d'une procédure `tri_insertion` qui construit un tableau `T2` trié en insérant successivement chacun des éléments du tableau `T1`. Les tableaux `T1`, `T2` et la `tailleT` seront passés en paramètres.

```
Procédure tri_insertion (T1 tableau[MAX] d'entiers, T2[tableau[MAX]
d'entiers, tailleT : entier)
Préconditions : aucune
Données : tailleT
Données / résultat : T1,T2
Description : trie T1 en construisant T2
Variables locales : i, indice,nb : entier
Début
     nb←0
     pour i allant de 0 à tailleT-1 par pas de 1 faire
         indice ← recherche_pos(T2,nb,T1[i])
         insere_val_tab(T2,nb,T1[i], indice)
     fin pour
fin
```

Exercice 2 : Distance de Hamming

La distance de Hamming entre deux mots (chaines de caractères) **de même longueur** est égale au nombre de lettres, à la même position, qui diffèrent.

Par exemple la distance de Hamming entre "rose" et "ruse" est de 1, entre "pomme" et "poire" est de 2.

1- Ecrire l'algorithme d'un sous-programme `met_a_la_meme_longueur` qui à partir de deux chaines de caractères `ch1` et `ch2` passées en paramètres, tronque la chaine la plus longue à la longueur de la plus courte.
Exemple si ch1 = "cestbientotlafin" et ch2 = "boncourage", le sous-programme devra transformer ch1 en "cestbiento". On pourra utiliser la fonction `longueur` qui retourne la longueur d'une chaine.

```
Procédure met_a_la_meme_longueur (ch1 chaine[MAX] de caractères, ch2
chaine[MAX] de caractères)
Préconditions : aucune
Données : aucune
Données / résultat : ch1,ch2
Description : met ch1 et ch2 à la même longueur
Variables locales : lg1,lg2 : entier
Début
     lg1 ← longueur (ch1)
     lg2 ← longueur (ch2)
     si (lg1>lg2) alors
          ch1[lg2] ← '\0'
     sinon
          ch2[lg1] ← '\0'
     fin si
fin
```

2- Ecrire l'algorithme d'une **fonction** `distance_hamming` qui calcule la distance de Hamming entre deux chaines de caractères de même longueur `ch1` et `ch2` passées en paramètres.

```
fonction hamming (ch1 chaine[MAX] de caractères, ch2 chaine[MAX] de
caractères) : entier
Préconditions : aucune
Données : aucune
Données / résultat : ch1,ch2
Résultat : entier
Description : calcule et retourne la distance de hamming
Variables locales : lg,i,distance : entier
Début
         lg ← longueur(ch1)
         distance ← 0
         pour i allant de 0 à lg-1 par pas de 1 faire
            si (ch1[i]!=ch2[i])alors
                 distance ← distance +1
            fin si
         fin pour
         retourner distance
Fin
```

3- Ecrire l'algorithme du programme principal qui demande à l'utilisateur 2 chaines de caractères, et, après les avoir mises à la même longueur, calcule et affiche la distance de Hamming entre ces deux chaines. Vous utiliserez les sous-programmes écrits dans les questions précédentes.

```
Début
     chaine1 : chaine[MAX] de caractères
     chaine2 : chaine[MAX] de caractères
     Afficher ("donnez 2 chaines ")
     Saisir (chaine1)
     Saisir (chaine2)
     met_a_la_meme_longueur(chaine1,chaine2);
     Afficher (hamming(chaine1,chaine2)
Fin
```