BTS Informatique de gestion 1re année

Frédérique Parisot

Algorithmique et langages

Autocorrection 2

Directrice de publication : Valérie Brard-Trigo

Sommaire

Séquence 5

Les tableaux

Partie 1

Les tableaux à une dimension

Exercice 60

Écrire l'algorithme de remplissage par saisie d'un tableau de 25 entiers.

La correction est dans le cours.

Exercice 61

Écrire l'algo du jeu du pendu. Les mots (une vingtaine) sont saisis dans un tableau. Le mot à trouver est choisi aléatoirement dans le tableau par l'ordinateur. Le nombre d'essais possibles est saisi par l'utilisateur.

Lexique	Jeu du pendu

tabMots (tableau [1..20] de chaîne, saisi) : tableau contenant les mots pour le jeu.
j (entier, calculé) : compteur d'itérations.
motATrouver (chaîne, calculé) : mot que le joueur 2 doit trouver.
hasard (fonction) résultat : entier : fonction qui retourne un nombre compris entre les deux nombres passés en paramètres.
essaisAutor (entier, saisi) : nombre d'essais autorisés .
essai (chaîne, saisie) : mot saisi par le joueur 2.
nbEssais (entier, calculé) : nombre d'essais courant.
i (entier, calculé) : compteur d'itérations, sur la longueur du mot à trouver.
	Début
	// Saisie des mots dans le tableau
	Pour j de 1 à 20
	Faire Afficher ("Saisir un mot")
	Saisir (tabMots[i])
	FinPour
	// Choix aléatoire du mot dans le tableau
	motATrouver ← tabMots[hasard (1, 20)]
	// Le reste du traitement reste rigoureusement identique à ce
	// qu'on avait fait dans les chapitres précédents
	Afficher ("Nombre d'essais autorisés :")
	Saisir (essaisAutor)
	Afficher ("Premier essai :")
	Saisir (essai)
	nbEssais ← 1
	TantQue essai <> motATrouver et nbEssais <= essaisAutor
	Faire Pour i de 1 à longueur (motATrouver)

longueur (fonction)
résultat : entier : fonction qui retourne le nombre de caractères contenus dans la chaîne passée en paramètre.
	Faire Si essai[i] <> motATrouver[i]
	Alors essai[i] ← "-"
	FinSi
	FinPour
	Afficher ("Voici les bonnes lettres :", essai)
	nbEssais ← nbEssais + 1
	Si nbEssais = essaisAutor
	Alors Afficher ("Saisis ton dernier essai")
	Sinon Afficher ("Joueur 2, saisis ton", nbEssais, "° essai")
	FinSi
	Saisir (essai)
	FinTantQue
	// Si on sort, c'est que le joueur a trouvé le mot ou bien qu'il
	// est arrivé au bout du nombre d'essais autorisés
	Si essai = motATrouver
	Alors Afficher ("Bravo, tu as enfin trouvé le mot")
	Sinon Afficher ("Tu as définitivement perdu, vraiment, tu n'es pas doué, le mot à trouver était ", motATrouver, ", c'était pourtant simple!")
	FinSi
	Fin

Voilà. Ce que l'on peut regretter avec cette solution, c'est que le joueur 1 soit obligé de saisir sa collection de mots à chaque nouveau lancement du programme. Nous pourrons supprimer cet inconvénient lorsque nous saurons recopier le contenu d'un tableau dans un fichier, et récupérer le contenu d'un fichier dans un tableau.

Exercice 62

Écrire l'algorithme qui affiche à l'écran la décomposition d'une somme d'argent saisie au clavier par l'utilisateur en un minimum de pièces et de billets.

Pièces et billets autorisés : de 0.5 € à 50 €

Vous avez, dans cet exercice, un exemple de déclaration de tableau de constantes. J'en profite pour vous rappeler qu'il n'y a généralement que lors de la déclaration d'un tableau de constantes qu'on a le droit d'écrire quelque chose du genre **monTableau = (valeurs des éléments du tableau séparées par des virgules)**.

Lexique	Décompo avec tableau Version avec test pour savoir si ce sont des pièces ou des billets
somme (entier, saisi) : somme à décomposer. cpt (entier, calculé) : compteur d'itérations et indice de parcours du tableau tabMontants. mot (chaîne, calculé) : sert à construire le message d'affichage. combien (entier, calculé) : nombre de pièce ou de billet du montant courant. tabMontants (tableau [1..7] de réels, constante) = (0.5, 1, 2, 5, 10, 20, 50) : tableau contenant les montants des pièces et billets.	Début Afficher ("Saisir la somme à décomposer") Saisir (somme) cpt ← 7 TantQue somme <> 0 et cpt >= 1 Faire Si cpt < 4 Alors mot ← "pièce(s)" Sinon mot ← "billet(s)" Finsi combien ← partieEntière (somme / tabMontants[cpt]) Afficher (combien, " ", mot, "de ", tabMontants[cpt], "€.") somme ← somme – combien * tabMontants[cpt] cpt ← cpt - 1 FinTantQue Fin

La deuxième version (celle page suivante, avec un deuxième tableau), nous évite le test à chaque passage dans la boucle.

Lexique	Décompo avec tableau Version avec un tableau pour savoir si ce sont des pièces ou des billets
somme (entier, saisi) : somme à décomposer. cpt (entier, calculé) : compteur d'itérations et indice de parcours du tableau tabMontants. tabMontants (tableau [1..7] de réels, constante) = (0.5, 1, 2, 5, 10, 20, 50) : tableau contenant les montants des pièces et billets. combien (entier, calculé) : nombre de pièce ou de billet du montant courant. tabMonnaies (tableau [1..7] de chaîne), constante) = ("pièce(s)", "pièce(s)", "pièce(s)", "billet(s)", "billet(s)", "billet(s)", "billet(s)") : tableau indiquant si la monnaie est en pièces ou en billets.	Début Afficher ("Saisir la somme à décomposer") Saisir (somme) cpt ← 7 TantQue somme <> 0 et cpt >= 1 Faire combien ← partieEntière (somme / tabMontants[cpt]) Afficher (combien, " ", tabMonnaies[cpt], "de ", tabMontants[cpt], "€.") somme ← somme – combien * tabMontants[cpt] cpt ← cpt - 1 FinTantQue Fin

Exercice 63

Soit un tableau pouvant contenir 100 mots. Le programme commence par permettre à l'utilisateur de saisir un certain nombre de mots, l'utilisateur indiquant qu'il ne désire plus saisir de mots en saisissant un mot vide (appui sur la touche « entrée » sans rien avoir saisi)…

… Ensuite, le programme permet les choix suivants à l'utilisateur.

Remarque : le fait que l'indice du premier élément du tableau soit zéro a pour conséquence qu'à chaque instant, la fonction **taille** retourne le nombre effectif d'éléments mais également l'**indice du premier emplacement libre** dans le tableau.

Lexique	Saisie des mots dans le tableau Version de base

tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
 	
i (entier, calculé) : indice de parcours du tableau et compteur d'itérations.
 	
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
 	
choix (entier, saisi) : choix de l'utilisateur dans le menu qui lui est proposé.
	Début
	
	// Boucle de saisie des mots
	
	Afficher ("Saisissez votre premier mot")
	
	Saisir (TabMots[0])
	
	// A partir d'ici, taille(tabMots) renvoie 1
	
	TantQue taille(tabMots) < 100 et tabMots[taille(tabMots)-1] <> ""
	
	Faire mot Afficher ("Saisissez le " + taille(tabMots) + 1 + "° (saisir un mot vide pour sortir)")
	Saisir (tabMots[taille(tabMots)])
	
	FinTantQue
	
	// Proposition des différents choix à l'utilisateur
	
	Afficher ("Pour afficher le mot dont l'indice est saisi, tapez 1")
	
	Afficher ("Pour afficher l'indice d'un mot dans le tableau, le mot étant saisi, tapez 2")
	
	Afficher (Pour afficher tous les indices auxquels a été trouvé le mot que vous avez saisi, tapez 3")
	
	Afficher ("Pour supprimer le mot dont vous avez saisi la position, tapez 4")
	
	Afficher ("Pour supprimer du tableau la première occurrence du mot que vous avez saisi, tapez 5")
	
	Afficher ("Pour insérer le mot saisi à l'indice spécifié dans le tableau, tapez 6")
	
	Afficher ("Pour supprimer du tableau le mot que vous avez saisi autant de fois qu'il se trouve dans le tableau, tapez 7")
	
	Afficher ("Pour supprimer les doublons dans le tableau, tapez 8")
	
	Afficher ("Pour rajouter le mot saisi en autorisant ou non l'existence de doublons dans le tableau, tapez 9")
	
	Afficher ("Saisissez votre choix")
	
	Saisir (choix)
	
	Selon cas choix faire….
	
	// Suite de l'algo dans les exos suivants….

Remarque : on est obligé de saisir le premier mot à l'extérieur du **TantQue** car on se sert de la valeur de ce premier mot saisi comme condition d'entrée dans le **TantQue**. En effet, si l'utilisateur appuie d'emblée sur entrée sans saisir aucun mot, on n'entrera pas dans le **TantQue**.

Exercice 64

Afficher le mot dont la position est saisie, avec affichage d'un message si aucun mot n'a été saisi à cette position.

Remarque : je ne mets dans le lexique de chaque exo que les informations utiles à cet exo. Le lexique de l'exercice tout entier est une fusion de tous les lexiques de chaque exercice.

Lexique	Partie de l'algo qui affiche le mot dont la position est saisie par l'utilisateur

tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
pos (entier, saisi) : position du mot à afficher.
	// Voir le début dans l'exo précédent
	// Si l'utilisateur a fait le choix n°1
	1 : Afficher ("Saisissez la position du mot que vous souhaitez afficher")
	Saisir (pos)
	Si pos <= taille(tabMots)
	Alors Afficher (tabMots[pos-1])
	Sinon Afficher ("Il n'y a pas de mot à cet emplacement du tableau")
	FinSi
	….
	// Pour la suite, voir l'exo suivant.

Exercice 65

Afficher la position d'un mot dans le tableau, le mot étant saisi, avec affichage d'un message si le mot ne figure pas dans le tableau.

Lexique	Partie de l'algo qui affiche la position du mot qui a été saisi

mot (chaîne, saisi) : mot dont l'utilisateur veut connaître la position dans le tableau.
tabMots (tableau[0..99]de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
ind (entier, calculé) : indice de parcours du tableau.
	……
	// Voir le début dans l'exo précédent
	2 : Afficher ("Saisissez le mot dont vous souhaitez connaître la position")
	Saisir (mot)
	ind ← 0
	// Parcours séquentiel du tableau
	TantQue mot <> tabMots[ind] et ind < taille(tabMots) - 1
	Faire ind ← ind + 1
	FinTantQue
	// Si on est sorti du TantQue, c'est qu'on a trouvé le
	// mot ou bien qu'on est arrivé à la fin du tableau.
	// On doit donc tester la raison pour laquelle on est sorti
	Si mot = tabMots[ind]
	Alors Afficher ("Le mot que vous avez saisi se trouve à la position n° ", ind +1, " dans le tableau.")
	Sinon Afficher ("Ce mot ne figure pas dans le tableau.")
	FinSi
	// Voir suite exo suivant.

Remarque : taille(tabMots) – 1 est l'indice du dernier mot figurant dans le tableau.

Autre remarque : je veux revenir rapidement sur les instructions qui suivent le **FinTantQue** ci-dessus.

Je vous explique, dans un des commentaires ci-dessus, que l'on doit tester la raison pour laquelle on est sorti du **TantQue**. Pour sortir de cette itération, il y a deux raisons possibles.

On a pu sortir du **TantQue** parce qu'on a trouvé le mot, ou bien parce qu'on est arrivé à la fin du tableau, ou bien pour ces deux raisons à la fois.

Nous, ce qui nous intéresse, ce n'est pas de savoir si on est arrivé ou non à la fin du tableau, mais de savoir si on a trouvé ou non le mot saisi par l'utilisateur.

C'est la raison pour laquelle le test qui suit le **FinTantQue** est **Si mot = tabMots[ind]** et pas **Si ind = taille(tabMots) – 1**.

Exercice 66

Afficher toutes les positions auxquelles a été trouvé le mot saisi par l'utilisateur, afficher également le nombre d'occurrences du mot (c'est-à-dire le nombre de fois qu'il apparaît dans le tableau). Prévoir l'affichage d'un message si le mot ne figure pas dans le tableau.

Lexique	Partie de l'algo qui affiche toutes les positions et le nombre d'occurrences du mot qui a été saisi

mot (chaîne, saisi) : mot dont l'utilisateur veut connaître les positions dans le tableau.
nbOcc (entier, calculé) : nombre d'occurrences du mot dans le tableau.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
ind (entier, calculé) : indice de parcours du tableau.
	// Voir le début dans l'exo précédent
	…
	3 : Afficher ("Saisissez le mot dont vous souhaitez connaître les positions et le nombre d'apparitions.")
	Saisir (mot)
	// Initialisation du nombre d'occurrences
	nbOcc ← 0
	// Parcours séquentiel de la partie du tableau contenant
	// des mots
	Pour ind de 0 à taille(tabMots) - 1
	Faire Si tabMots[ind] = mot
	Alors Afficher ("On trouve ", mot, " à la position", ind + 1)
	nbOcc ← nbOcc + 1
	FinSi
	FinPour
	Si nbOcc <> 0
	Alors Afficher ("On trouve le mot ", nbOcc, " fois dans le tableau)
	Sinon Afficher ("Ce mot ne figure pas dans le tableau.")
	FinSi
	// Voir suite exo suivant.

La façon précédente de faire est tout à fait correcte mais lors de l'exécution, si le mot apparaît un nombre conséquent de fois dans le tableau, on va avoir un affichage pas très convivial du type : *On trouve truc à la position 2. On trouve truc à la position 7. On trouve truc à la position 29.On trouve truc à la position 52.On trouve le mot 4 fois dans le tableau.*

Je trouve qu'il serait plus convivial qu'on ait un affichage du type :

On trouve truc à la position 2, à la position 7, à la position 29, à la position 52.

On trouve le mot 4 fois dans le tableau.

Lexique	Partie de l'algo qui affiche toutes les positions et le nombre d'occurrences du mot qui a été saisi

mot (chaîne, saisi) : mot dont l'utilisateur veut connaître les positions dans le tableau.
nbOcc (entier, calculé) : nombre d'occurrences du mot dans le tableau.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
ind (entier, calculé) : indice de parcours du tableau.
resultat (chaîne, calculé) : chaîne contenant le message que l'on affichera à la fin du traitement.
fchaîne (fonction) résultat chaîne : fonction qui convertit la valeur numérique passée en paramètre en chaîne de caractères.
	// Voir le début dans l'exo précédent
	….
	3 : Afficher ("Saisissez le mot dont vous souhaitez connaître les positions et le nombre d'apparitions.")
	Saisir (mot)
	// Initialisation du nombre d'occurrences
	nbOcc ← 0
	// Initialisation de la chaîne résultat
	resultat ← "On trouve " + mot
	// Parcours séquentiel du tableau
	Pour ind de 0 à taille(tabMots) -1
	Faire Si tabMots[ind] = mot
	Alors Si nbOcc <> 1
	Alors resultat ← resultat + ", "
	FinSi
	resultat ← resultat + " à la position " + fchaîne (ind + 1)
	nbOcc ← nbOcc + 1
	FinSi
	FinPour
	Si nbOcc <> 0
	Alors resultat ← resultat + ". On trouve le mot " + fchaîne (nbOcc), " fois dans le tableau.")
	Afficher (resultat)
	Sinon Afficher ("Ce mot ne figure pas dans le tableau.")
	FinSi
	// Voir suite exo suivant.

Exercice 67

Supprimer un mot dont la position a été saisie par l'utilisateur en décalant les autres mots afin qu'il n'y ait pas de « trou ». Si aucun mot ne figure à la position indiquée par l'utilisateur, un message le lui indique.

Lexique	Partie de l'algo qui supprime le mot dont la position a été saisie

pos (entier, saisi) : position du mot à supprimer.
ind (entier, calculé) : indice de parcours de tabMots.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
	// Voir le début dans l'exo précédent
	…
	4 : Si taille(tabMots) <> 0
	Alors Afficher ("Saisissez la position du mot dont vous souhaitez la suppression")
	Saisir (pos)
	Si pos > taille(tabMots) - 1
	Alors Afficher ("Il n'y a pas de mot à cette position")
	Sinon Pour ind de pos à taille(tabMots) – 1
	Faire tabMots[ind – 1] ← TabMots[ind]
	FinPour
	tabMots[taille(tabMots) - 1] ← nulle
	Afficher("Suppression effectuée")
	FinSi
	Sinon Afficher("Le tableau est vide, aucune suppression possible")
	FinSi
	// Voir suite exo suivant…

Remarque : vous pourrez remarquer qu'à chaque fois qu'il s'agit d'effectuer des suppressions de mot(s), on vérifie, avant de lancer le traitement de suppression, que le tableau n'est pas vide. En effet, il me paraît utile de signaler à l'utilisateur qu'il essaie de supprimer des éléments d'une collection vide.

Exercice 68

Supprimer du tableau la première occurrence (= la première apparition) du mot qui a été saisi.

Lexique	Partie de l'algo qui supprime le mot saisi

mot (chaîne, saisi) : mot à supprimer.
ind (entier, calculé) : indice de parcours de TabMots.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
	// Voir le début dans l'exo précédent
	…
	5 : Si taille(tabMots) <> 0
	Alors Afficher ("Saisissez le mot dont vous souhaitez la suppression")
	Saisir (mot)
	ind ← 0
	TantQue tabMots[ind] <> mot
	et ind < taille(tabMots) - 1
	Faire ind ← ind + 1
	FinTantQue
	Si tabMots[ind] = mot
	Alors Pour i de ind + 1 à taille(tabMots) – 1
	Faire tabMots[i – 1] ← tabMots[i]
	FinPour
	TabMots[taille(tabMots) – 1] ← nulle
	Sinon Afficher ("Ce mot n'est pas dans le tableau")
	FinSi
	Sinon Afficher ("Le tableau est vide, aucune suppression possible")
	FinSi
	// Voir suite exo suivant…

Exercice 69

Écrire un traitement permettant d'insérer le mot saisi par l'utilisateur, à l'indice saisi par l'utilisateur, dans le tableau.

Si l'indice saisi dépasse l'indice de la première case libre, ranger le mot à la première case libre du tableau. Dans ce cas, la position d'insertion est indiquée à l'utilisateur par un message d'information. Si le tableau est plein, demander à l'utilisateur de confirmer l'insertion et écraser l'élément présent dans le tableau à l'indice saisi.

Lexique	Solution 1 Partie de l'algo permettant d'insérer le mot saisi à la position saisie.

mot (chaîne, saisi) : mot à insérer dans le tableau.
pos (entier, saisi) : position d'insertion souhaitée pour le mot saisi.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
leChoix (entier, saisi) : choix de l'utilisateur concernant l'écrasement ou non du mot présent à l'emplacement d'insertion du mot saisi.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
posInser (entier, calculé) : position d'insertion du mot.
	6 : Afficher ("Saisissez le mot à insérer, et sa position souhaitée")
	Saisir (mot, pos)
	Si taille(tabMots) < 100
	Alors Si pos >= taille(tabMots)
	Alors tabMots[taille(tabMots)] ← mot
	posInser ← taille(tabMots)
	Sinon // on met le mot à la fin du tableau
	tabMots[taille(tabMots)] ← tabMots[pos – 1]
	// on le remplace par mot
	tabMots [pos – 1] ← mot
	posInser ← pos
	FinSi
	Afficher ("Le mot a été inséré à la ", posInser, " ° position")
	Sinon Afficher ("Tableau plein, tapez 1 pour remplacer le mot à la position", pos, "par", mot, "tapez 2 pour abandonner l'insertion")
	Saisir (leChoix)
	Si leChoix = 1
	Alors tabMots[pos – 1] ← mot
	FinSi
	FinSi
	// Voir suite exo suivant…

Lexique	Solution 2 Partie de l'algo permettant d'insérer le mot saisi à la position saisie

Le lexique de cette solution est le même que pour la solution précédente sauf qu'en plus, il y a la variable ind :
 	
ind (entier, calculé) : indice de parcours du tableau.
	// Voir le début dans l'exo précédent
	6 : Afficher ("Saisissez le mot à insérer, et sa position souhaitée")
	Saisir (mot, pos)
	Si taille(tabMots) < 100
	Alors Si pos > taille(tabMots)
	Alors tabMots[taille(tabMots)] ← mot
	Sinon // On décale tous les mots en partant de //la fin du tableau pour ne pas les écraser
	Pour ind de taille(tabMots) à pos (diminuer de 1)
	Faire tabMots[ind] ← tabMots[ind – 1]
	FinPour
	// on le remplace le mot d'indice pos – 1 par mot
	tabMots [pos - 1] ← mot
	FinSi
	// La suite reste la même que dans la solution 1
	// Voir suite exo suivant…

Exercice 70

Supprimer du tableau le mot saisi par l'utilisateur autant de fois qu'il se trouve dans le tableau.

Remarque : comme pour n'importe quel algorithme, il n'existe pas qu'une solution pour résoudre le problème posé par cet énoncé. Pour exemple, mes étudiants de première année ont proposé 4 solutions assez différentes pour l'exercice 70.

Si votre solution vous paraît bonne et est différente de celle proposée ci-dessous, gardez-la, et en TP, programmez-la. Cela vous permettra de vérifier si votre solution convient, et les éventuelles corrections à lui apporter.

Lexique	Algo2 Partie de l'algo qui supprime toutes les occurrences du mot saisi

mot (chaîne, saisi) : mot à insérer dans le tableau.
ind (entier, calculé) : indice de parcours du tableau
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
i (entier, calculé) : indice de parcours utilisé pour le décalage.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
	// Voir le début dans l'exo précédent
	…
	7 : Si taille(tabMots) <> 0
	Alors Afficher ("Saisissez le mot dont vous souhaitez supprimer toutes les occurrences")
	Saisir (mot)
	ind ← 0
	// Parcours du tableau en entier
	TantQue ind < taille(tabMots)
	Faire // Recherche d'une occurrence
	TantQue tabMots[ind] <> mot
	et ind < taille(tabMots)
	Faire ind ← ind + 1
	FinTantQue
	// Si on sort du TantQue c'est soit parce
	// qu'on a trouvé une occurrence du mot,
	// soit parce qu'on est à la fin du tableau,
	// soit pour ces deux raisons à la fois
	// Suppression de l'occurrence éventuel-
	// lement trouvée
	// Si on est sorti du TantQue parce qu'on
	// était à la fin du tableau, alors on
	// n'entre pas dans le pour ci-dessous, car
	// la borne inférieure de l'itération
	// est supérieure à la borne supérieure de // l'itération
	Pour i de ind + 1 à taille(tabMots) - 1
	Faire tabMots[i – 1] ← tabMots[i]
	FinPour

	tabMots[taille(tabMots)] ← nulle
	FinTantQue
	// On remonte au TantQue pour traiter l'occurrence suivante
	Sinon Afficher ("Le tableau est vide, aucune suppression possible")
	FinSi
	// Voir suite exo suivant…

Exercice 71

Supprimer les doublons dans le tableau. Après exécution de ce traitement, chaque mot ne doit avoir qu'une seule occurrence dans le tableau.

Lexique	Partie de l'algo qui supprime les doublons dans le tableau

mot (chaîne, saisi) : mot à insérer dans le tableau.
k, i , j (entier, calculé) : indices de parcours du tableau.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
taille (fonction) résultat entier : fonction qui retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
	// Voir le début dans l'exo précédent
	…
	8 : Pour k de 0 à taille(tabMots) – 1
	// Traitement du mot se trouvant à l'indice k dans tabMots
	Faire // On parcourt le tableau à partir du mot suivant
	// le mot courant
	Pour i de k + 1 à taille(tabMots) –1
	Faire Si tabMots[i] = tabMots[k]
	Alors // On a trouvé un doublon, on le
	// supprime en l'écrasant
	Pour j de taille(tabMots) – 1 à i+1 (diminuer de 1)
	Faire tab[j – 1] ← tab[j]
	FinPour
	tabMots[taille(tabMots – 1)] ← nulle
	// Après cette instruction, la taille de
	// tabMots est décrémentée de 1
	FinSi
	FinPour
	FinPour
	FinSelon

Cette solution est assez simple dans la mesure où on écrase un doublon dès qu'on l'a trouvé.

Lors de la traduction de cet exo en windev, on risque d'avoir des problèmes avec les deux pour extérieurs, car la valeur de **taille(tabmots)** change d'un passage à l'autre dans l'itération. En effet, certains langages de programmation (comme le langage Pascal), n'acceptent pas, une fois qu'on a démarré l'itération, que ses bornes changent.

Si c'est le cas en windev, il nous suffira de remplacer les **pour** par des **TantQue** et le tour sera joué.

Exercice 72

L'utilisateur saisit une date au format jj/mm/aaaa, par exemple 15/10/2001 et le programme lui affiche : lundi, 15 Octobre 2001.

Cumul du nombre de jours des mois complets	

totalJours (entier, calculé) : nombre total de jours séparant les deux dates.
tabNbJoursParMois (Tableau [1..12] d'entiers, constante) = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31) : tableau contenant le nombre de jours de chaque mois.
annee (entier, calculé) : année en cours de traitement.
mois (entier, calculé) : mois en cours de traitement.
jourSais (entier, calculé) : jour de la date saisie.
moisSais (entier, calculé) : mois de la date saisie.
annSais (entier, calculé) : année de la date saisie.
	.....
	totalJours ← tabNbJoursParMois[moisSais] – jourSais
	annee ← annSais
	mois ← moisSais +1
	TantQue mois <> moisCour – 1 ou annee <> annCour
	Faire totalJours ← totalJours + tabNbJoursParMois[mois]
	Si ((annee MOD 4 = 0 et annee MOD 4 <> 100)
	ou annee MOD 400 = 0)
	et mois = 2
	Alors totalJours ← totalJours + 1
	FinSi
	Si mois = 12
	Alors mois ← 1
	annee ← annee + 1
	Sinon mois ← mois + 1
	FinSi
	FinTantQue
	// C'est vrai que c'est beaucoup plus court à écrire de cette
	// façon

Lexique	« Prise de tête » Version où la date saisie est antérieure à la date courante

date (chaîne, saisie) : date pour laquelle l'utilisateur veut retrouver le nom du jour (a t'on idée!).
jourSais (entier, calculé) : jour de la date saisie.
moisSais (entier, calculé) : mois de la date saisie.
annSais (entier, calculé) : année de la date saisie.
	Début
	Afficher ("Saisissez la date au format jj/mm/aaaa")
	Saisir (date)
	// Découpage de la date saisie
	jourSais ← val (sousChaîne (date,1,2))
	moisSais ← val (sousChaîne (date,3,2))
	annSais ← val (sousChaîne (date,7,4))
	// Initialisation du nombre de jours séparant les deux dates

Lexique	« Prise de tête » Version où la date saisie est antérieure à la date courante

totalJours (entier, calculé) : cumul des jours séparant la date courante de la date saisie.
 	
tabNbJoursParMois (Tableau [1..12] d'entiers, constante) = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31) : tableau contenant le nombre de jours de chaque mois.
 	
bissext (booléen, calculé) : vrai si l'année courante est bissextile.
 	
i, j, k (entiers, calculés) : compteurs d'itérations.
 	
annCour (entier, calculé) : année courante, fournie par le système d'exploitation.
 	
moisCour (entier, calculé) : mois courant, fourni par le système d'exploitation.
 	
jourCour (entier, calculé) : jour courant, fourni par le système d'exploitation.
 	
ind (entier, calculé) : indice du nom du jour courant dans le tableau des jours.
 	
nomJourCour (chaîne, calculé) : nom du jour courant, fourni par le système d'exploitation.
 	
nomJourSais (chaîne, calculé) : nom du jour de la date saisie, fruit de tant de sueur et d'efforts.
 	
tabJours (Tableau [1..7] de chaîne, constante) = ("lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche") : tableau des jours de la semaine.
	totalJours ← tabNbJoursParMois[moisSais] – jourSais
	// Les trois itérations concernant les mois et les années
	// complets
	bissext ← (annSais MOD 4 = 0 et annSais MOD 4 <> 100) ou annSais MOD 400 = 0
	Pour i de moisSais + 1 à 12
	Faire totalJours ← totalJours + tabNbJoursParMois[i]
	Si bissext et i = 2
	Alors totalJours ← totalJours + 1
	FinSi
	FinPour
	Pour j de annSais + 1 à annCour - 1
	Faire Si (j MOD 4 = 0 et j MOD 4 <> 100)
	ou j MOD 400 = 0
	Alors totalJours ß totalJours + 366
	Sinon totalJours ß totalJours + 365
	FinSi
	FinPour
	bissext ← (annCour MOD 4 = 0 et annCour MOD 4 <> 100) ou annCour MOD 400 = 0
	Pour k de 1 à moisCour - 1
	Faire totalJours ← totalJours + tabNbJoursParMois[k]
	Si bissext et k = 2
	Alors totalJours ← totalJours + 1
	FinSi
	FinPour
	// On rajoute ensuite à totalJours le nombre de jours
	// entre le premier jour du mois courant et jourCour
	totalJours ← totalJours + jourCour
	// Recherche de la position du jour courant dans le tableau
	// des jours
	ind ← 1
	TantQue tabJours[ind] <> nomJourCour
	Faire ind← ind + 1

Lexique	« Prise de tête » Version où la date saisie est antérieure à la date courante

tabMois(Tableau [1..12] de chaîne, constante) = ("janvier", "février", "mars", "avril", "mai", "juin", "juillet", "août", "septembre", "octobre", "novembre", "décembre") : tableau des mois de l'année.	FinTantQue
	// ind contient la position du nom du jour courant dans le
	// tableau pour trouver le nom du jour de la date saisie, on
	// enlève à ind le décalage que représente totalJours
	// MOD 7
	ind ← ind - totalJours MOD 7
	// Il se peut très bien que ind soit négatif suite à ce
	// calcul. Exemple : si le jour courant est un mardi et
	// qu'on a 6 jours de décalage (le jour à trouver est un
	// mercredi). Si c'est le cas, il faut rajouter 7 à ind pour le
	// recadrer par rapport aux indices du tableau.
	Si ind < 1
	Alors ind ← ind + 7
	FinSi
	nomJourSais ← tabJours[ind]
	// On affiche le message tant attendu
	Afficher (nomJourSais, ", ", jourSais, " ", tabMois[moisSais], " ", annSais)
	// Tout ça pour ça! Eh ben dis donc!
	Fin

Si vous préférez la solution qui utilise une seule itération pour le cumul des jours des mois et années complets, remplacez la partie concernée dans l'algo.

Lexique	« Prise de tête » Version qui marche pour une date saisie antérieure ou postérieure à la date courante

date (chaîne, saisie) : date pour laquelle l'utilisateur veut retrouver le nom du jour (a t'on idée!).
jourSais (entier, calculé) : jour de la date saisie.
moisSais (entier, calculé) : mois de la date saisie.
annSais (entier, calculé) : année de la date saisie.
totalJours (entier, calculé) : cumul des jours séparant la date courante de la date saisie.
tabNbJoursParMois (Tableau [1..12] d'entiers, constante) = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31) : tableau contenant le nombre de jours de chaque mois.
bissext (booléen, calculé) : vrai si l'année courante est bissextile.
i, j, k (entiers, calculés) : compteurs d'itérations.
annCour (entier, calculé) : année courante, fournie par le système d'exploitation.
moisCour (entier, calculé) : mois courant, fourni par le système d'exploitation.
jourCour (entier, calculé) : jour courant, fourni par le système d'exploitation.
petitJour, petitMois, petitAnn, grandJour, grandMois, grandAnn (entier, calculés) : respectivement jour, mois et année de la plus petite et de la plus grande date.
ind (entier, calculé) : indice du nom du jour courant dans le tableau des jours.
	Début
	Afficher ("Saisissez la date au format jj/mm/aaaa")
	Saisir (date)
	// Découpage de la date saisie
	jourSais ← val (sousChaîne (date,1,2))
	moisSais ← val (sousChaîne (date,3,2))
	annSais ← val (sousChaîne (date,7,4))
	// Recherche de la plus petite et de la plus grande date
	Si date < dateCour
	Alors petitJour ← jourSais
	petitMois ← moisSais
	petitAnn ← annSais
	grandJour ← jourCour
	grandMois ← moisCour
	grandAnn ← annCour
	Sinon petitJour ← jourCour
	petitMois ← moisCour
	petitAnn ← annCour
	grandJour ← JourSais
	grandMois ← moisSais
	grandAnn ← annSais
	FinSi
	// Initialisation du nombre de jours séparant les deux
	// dates
	totalJours ← tabNbJoursParMois[petitMois] – petitJour
	// Je mets ici la version avec une seule itération pour
	// calculer les mois et les années complets
	annee ← petitAnn
	mois ← petitMois + 1
	TantQue mois <> grandMois – 1 ou annee <> grandAnn
	Faire totalJours ← totalJours + tabNbJoursParMois[mois]
	Si ((annee MOD 4 = 0 et annee MOD 4 <> 100) ou annee MOD 400 = 0) et mois = 2
	Alors totalJours ← totalJours + 1

Lexique	« Prise de tête » Version qui marche pour une date saisie antérieure ou postérieure à la date courante

nomJourCour (chaîne, calculé) : nom du jour courant, fourni par le système d'exploitation.
 	
nomJourSais (chaîne, calculé) : nom du jour de la date saisie, fruit de tant de sueur et d'efforts.
 	
tabJours (Tableau [1..7] de chaîne, constante) = ("lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche") : tableau des jours de la semaine.
 	
tabMois(Tableau [1..12] de chaîne, constante) = ("janvier", "février", "mars", "avril", "mai", "juin", "juillet", "août", "septembre", "octobre", "novembre", "décembre") : tableau des mois de l'année.
	FinSi
	Si mois = 12
	Alors mois ← 1
	annee ← annee + 1
	Sinon mois ← mois + 1
	FinSi
	FinTantQue
	// On rajoute ensuite à totalJours le nombre de jours
	// entre le premier jour du grand mois et grandJour
	totalJours ← totalJours + grandJour
	// Recherche de la position du jour courant dans
	// tabJours
	ind ← 1
	TantQue tabJours[ind] <> nomJourCour
	Faire ind ← ind + 1
	FinTantQue
	// ind contient la position du nom du jour courant dans
	// le tableau pour trouver le nom du jour de la date
	// saisie, on enlève ou on rajoute à ind le décalage que
	// représente totalJours MOD 7
	Si date < dateCour
	Alors ind ← ind – totalJours MOD 7
	Sinon ind ← ind + totalJours MOD 7
	FinSi
	// On recadre la valeur de ind par rapport aux indices
	// du tableau
	Si ind < 1
	Alors ind ← ind + 7
	FinSi
	Si ind > 7
	Alors ind ← ind – 7

Lexique	« Prise de tête » Version qui marche pour une date saisie antérieure ou postérieure à la date courante

	FinSi
	nomJourSais ← tabJours[ind]
	// On affiche le message tant attendu
	Afficher (nomJourSais, ", ", jourSais, " ", tabMois[moisSais], " ", annSais)
	// Tout ça pour ça! Eh ben dis donc!
	Fin

Partie 2

Algos de recherche dans un tableau

Exercice 73

Écrire un algorithme qui vérifie et affiche si un tableau est trié dans l'ordre croissant.

Lexique	Vérif tri
i (entier, calculé) : indice. tableau (tableau [1..20] de entiers) : tableau à vérifier.	Début // On suppose que le tableau est rempli i ← 1 TantQue tableau[i] <= tableau[i + 1] et i < 20 Faire i ← i + 1 FinTantQue Si i = 20 Alors Afficher ("Tableau trié") Sinon Afficher ("Tableau non trié") Fin

Si, si, je vous jure, ça marche ! Vérifiez si vous voulez.

Partie 3

Tri des éléments d'un tableau

Exercice 74

Écrire l'algorithme qui affiche dans l'ordre décroissant des nombres saisis dans un ordre quelconque. Ces nombres sont saisis dans un tableau, le tri se fait au fur et à mesure de la saisie.

Lexique	Tri élémentaire lors de la saisie

tab (tableau de[1..20] entiers, saisi).
i (entier, calculé) : indice d'itérations.
nombre (entier, saisi) : nombre courant à insérer.
ind (entier, calculé) : indice de parcours de tab.
indDec (entier, calculé) : indice de parcours de tab, utilisé pour les décalages.
	Début
	Afficher ("Saisissez le premier nombre")
	Saisir (tab[1])
	Pour i de 2 à 20
	Faire Afficher ("Saisissez le ", i, "° nombre")
	Saisir (nombre)
	ind ← 1
	TantQue ind < i et nombre <= tab[ind]
	Faire ind ← ind + 1
	FinTantQue
	// Si on est sorti du TantQue parce que ind < i,
	// alors on décale tous les nombres à partir de ind vers
	// la droite, sinon, on n'entrera pas dans le pour
	Pour indDec de i - 1 à ind (diminuer de 1)
	Faire tab[indDec + 1] ← tab[IndDec]
	FinPour
	// Si on est sorti du TantQue parce que ind > i,
	// alors c'est que nombre
	// est le plus petit on ne fait rien de spécial, on
	// insèrera nombre à l'indice ind, à la fin du tableau
	// on insère nombre soit en dernier, soit à la place qu'on
	// lui a libérée.
	tab[ind] ← nombre
	FinPour
	Fin

Exercice 75

Écrire l'algorithme qui affiche dans l'ordre croissant des nombres saisis dans un ordre quelconque. Ces nombres sont saisis dans un tableau, le tri se fait au fur et à mesure de la saisie, à l'aide d'une recherche dichotomique puis d'un décalage et d'une insertion à l'emplacement libéré.

Lexique	Insertion dichotomique au fur et à mesure de la saisie

i (entier, calculé) : nombre d'éléments du tableau et compteur d'itérations.
nombre (entier, saisi) : nombre courant à insérer.
inf (entier, calculé) : borne inférieure de la recherche.
sup (entier, calculé) : borne supérieure de la recherche.
milieu (entier, calculé) : milieu du tableau.
tab (tableau [1..20] d'entiers, saisi) : tableau à remplir et à trier.
j (entier, calculé) : indice utilisé pour les décalages.
	Début
	Pour i de 1 à 20
	Faire Afficher("Saisissez le ", i, "° nombre")
	Saisir (nombre)
	// initialisation des bornes inférieures et
	// supérieures de la recherche
	inf ← 1
	sup ← i - 1
	TantQue sup >= inf
	Faire // Indice du milieu du tableau
	milieu ← (inf + sup) div 2
	Si nombre < tab[milieu]
	Alors sup ← milieu – 1
	Sinon inf ← milieu + 1
	FinSi
	FinTantQue
	// Décalage
	Pour j de i – 1 à inf
	Faire tab[j + 1] ← tab[j]
	FinPour
	// Insertion
	tab[inf] ← nombre
	FinPour
	Fin

Exercice 76

Soit un tableau rempli de 20 entiers. Écrire l'algorithme qui trie ce tableau dans l'ordre croissant. Le résultat du tri est un deuxième tableau de 20 entiers.

Lexique	Tri élémentaire en utilisant un deuxième tableau

tab (tableau de[1..20] entiers, saisi) : tableau à trier.
i (entier, calculé) : indice d'itérations.
ind (entier, calculé) : indice de parcours de tab.
tabTri (tableau de[1..20] entiers, saisi) : tableau trié.
indDec (entier, calculé) : indice de parcours de tab, utilisé pour les décalages.
	Début
	// Remplissage du tableau
	Pour i de 1 à 20
	Faire saisir (tab[i])
	FinPour
	// Tri
	tabTri[1] ← tab[1]
	Pour i de 2 à 20
	Faire ind ← 1
	TantQue ind < i et tab[i] > tabtri[ind]
	Faire ind ← ind + 1
	FinTantQue
	// si nécessaire, on décale tous les nombres à partir de
	// ind vers la droite. Si ind = i, on n'entre pas dans le pour et
	// tab[i] est mis en dernier dans tabTri.
	Pour indDec de i à ind + 1 (diminuer de 1)
	Faire tabTri[indDec] ← tabTri[indDec - 1]
	FinPour
	// on insère tab[i] soit en dernier, soit à la place qu'on lui a
	// libérée.
	tabTri[ind] ← tab[i]
	FinPour
	Fin

Exercice 77

Écrire l'algo qui trie un tableau sur lui-même dans l'ordre décroissant et à l'aide de la méthode du tri par insertion simple.

Lexique	Tri par insertion simple, une fois la saisie terminée

tab (tableau de[1..20] entiers, saisi).
i (entier, calculé) : indice d'itérations.
temp (entier, calculé) : nombre à insérer dont on a sauvegardé la valeur.
j (entier, calculé) : indice de parcours de tab, utilisé pour les décalages.
	Début
	// On suppose le tableau déjà rempli
	// On commence la boucle à 2 car on considère
	// que l'élément d'indice 1 constitue un tableau
	// de 1 élément trié
	Pour i de 2 à 20
	Faire // Récupération de la valeur à insérer dans la partie triée du
	// tableau
	temp ← tab[i]
	//Initialisation de la borne de recherche de l'emplacement
	// d'insertion.
	// On commence la comparaison une case avant tab[i]
	j ← i – 1
	TantQue j >=1 et temp > tab[j]
	Faire tab[j + 1] ← tab[j]
	j ← j – 1
	FinTantQue
	// On insère
	tab[j + 1] ← temp
	FinPour
	Fin

Exercice 78

Trier sur lui-même dans l'ordre décroissant un tableau d'entiers déjà rempli à l'aide de la méthode du tri à bulle.

Lexique	Tri à bulle

tab (tableau de[?..?] entiers, saisi).
borneMin, borneMax (entiers, ?) : borne minimale et maximale entre lesquelles se passe le tri.
i (entier, calculé) : indice d'itérations.
j (entier, calculé) : indice de parcours de tab.
	Début
	// remplissage du tableau en vrac
	…
	// tri du tableau rempli
	Pour i de borneMin à borneMax - 1
	Faire Pour j de i +1 à borneMax
	Faire si tab[i] < tab[j]
	Alors permuter (tab[i], tab[j])
	FinSi
	FinPour
	FinPour
	Fin

Exercice 79

Écrire l'algorithme de création d'un index de tri par ordre alphabétique des mots d'un tableau de 20 mots indicé de 0 à 19. L'index est lui-même un tableau, indicé de 1 à 20.

Lexique	Création d'un index de tri

i (entier, calculé) : indice d'itération utilisé pour remplir le tableau.
tabMots (tableau [0..19] chaîne, saisi) : tableau des mots.
tabIndex(tableau [1..20] entier, saisi) : index.
j (entier, calculé) : indice d'itération utilisé pour remplir l'index.
	Début
	// Remplissage du tableau de mots
	Pour i de 0 à 19
	Faire saisir (tabMots[i])
	FinPour
	// Remplissage en vrac de tabIndex
	Pour j de 1 à 20
	Faire tabIndex[j] ← j -1
	FinPour
	// Tri des éléments de tabIndex suivant la valeur des éléments
	// de tabMots. Moi, j'ai choisi un tri à bulles.
	Pour i de 1 à 19
	Faire Pour j de i +1 à 20
	Faire si tabMots[tabIndex[i]] > tabMots[tabIndex[j]]
	Alors permuter (tabIndex[i], tabIndex[j])
	FinSi
	FinPour
	FinPour
	Fin

Exercice 80

On suppose qu'on a deux tableaux de mots, triés dans l'ordre lexicographique. Écrire l'algorithme de fusion de ces deux tableaux en un seul. Le résultat de la fusion est un tableau trié dans l'ordre alphabétique lui aussi.

Lexique	Fusion de deux tableaux

indTab1, indTab2 (entiers, calculés) : indices de parcours de tab1 et tab2.
nbElemTab1, nbElemTab2 (entiers, constantes) : bornes supérieures de tab1 et tab2.
tab1 (tableau [1.. nbElemTab1] de chaîne) : tableau à fusionner.
tab2 (tableau [1.. nbElemTab2] de chaîne) : tableau à fusionner.
tabRes (tableau [1.. nbElemTab1 + nbElemTab2] de chaîne) : tableau résultant de la fusion.
j (entier, calculé) : indice de parcours de tabRes.
i (entier, calculé) : indice d'itération.
	Début // On suppose les tableaux remplis
	// Tant qu'on n'est au bout d'aucun des deux tableaux, on fusionne
	indTab1 ← 1, indTab2 ← 1, j ← 1
	TantQue indTab1 <= nbElemTab1 et indTab2 <= nbElemTab2
	Faire Si tab1[indTab1] < tab2[indTab2]
	Alors tabRes[j] ← tab1[indTab1]
	indTab1 ← indTab1 + 1
	Sinon tabRes[j] ← tab2[IndTab2]
	indTab2 ← indTab2 + 1
	FinSi
	j ← j + 1
	FinTantQue
	// Si on sort, c'est qu'on a fini de parcourir un des deux tableaux, il
	// ne reste donc plus qu'à rajouter au bout les éléments du tableau
	// qu'on n'a pas fini de parcourir
	// Si indTab1 > nbElemTab1, on n'entrera pas dans le pour
	Pour i de indTab1 à nbElemTab1
	Faire tabRes[j] ← tab1[i]
	j ← j + 1
	FinPour
	// Si indTab2 > nbElemTab2, on n'entrera pas dans le pour
	Pour i de indTab2 à nbElemTab2
	Faire tabRes[j] ← tab2[i]
	j ← j + 1
	FinPour
	Fin

Partie 4

Les tableaux à deux dimensions

Exercice 81

Écrire l'algorithme permettant à l'utilisateur la saisie des différentes formes conjuguées d'un verbe du premier groupe au présent, à l'imparfait et au futur.

Lexique	Saisie des terminaisons
i (entier, calculé) : indice de parcours de tabTemps et indice de parcours des colonnes de tabConjug. tabTemps (Tableau [1..3] de chaîne, constante) = ("Présent", "Imparfait", "Futur") j (entier, calculé) : indice de parcours de tabPers et indice de parcours des lignes de tabConjug. tabPers (Tableau [1..6] de chaîne, constante) = ("Je", "Tu", "Il, elle, on", "Nous", "Vous", "Ils, elles") tabConjug (Tableau [1..6, 1..3] de chaîne, saisi) : tableau destiné à contenir les terminaisons des verbes du premier groupe.	Début Pour i de 1 à 3 Faire Afficher ("Terminaisons pour le ", tabTemps[i]) Pour j de 1 à 6 Faire Afficher("Saisissez la terminaison pour", tabPers[j]) Saisir (tabConjug[j,i]) FinPour FinPour Fin

Exercice 82

Écrire l'algorithme qui affiche la conjugaison d'un verbe du premier groupe saisi par l'utilisateur, au temps (présent, imparfait ou futur) et à la personne choisis par l'utilisateur.

Exemple

Au lancement, un message invite l'utilisateur à saisir le verbe qu'il souhaite conjuguer.

Deux messages s'affichent ensuite, lui permettant de choisir d'une part le pronom de conjugaison, d'autre part le temps choisi.

Si l'utilisateur saisit « arriver » puis choisit « 2e personne » et « présent » le message « Tu arrives » s'affiche.

Lexique	Conjugaison avec plusieurs temps
verbe (chaîne, saisi) : verbe à conjuguer. pers (entier, saisi) : numéro de la personne à utiliser pour la conjugaison. nombre (caractère, saisi) : s pour singulier, p pour pluriel. temps (entier, saisi) : temps de conjugaison choisi par l'utilisateur. racine (chaîne, calculé) : racine du verbe à conjuguer. souschaîne (fonction) résultat chaîne : renvoie le découpage la chaîne passée en paramètre à partir de la position spécifiée en 2ème paramètre sur la longueur spécifiée en 3ème paramètre. longueur (fonction) résultat entier : renvoie le nombre de caractères de la chaîne passée en paramètre. tabPers (Tableau [1..6] de chaîne, constante) = ("Je", "Tu", "Il, elle, on", "Nous", "Vous", "Ils, elles") tabConjug (Tableau [1..6, 1..3] de chaîne, saisi) : tableau destiné à contenir les terminaisons des verbes du premier groupe.	Début Afficher ("Saisissez le verbe à conjuguer") Saisir (verbe) Afficher ("Choisissez 1 pour la 1ère, 2 pour la 2ème, 3 pour la 3ème personne") Saisir (pers) Afficher ("Choisissez s pour singulier, p pour pluriel") Saisir (nombre) Si nombre = "p" Alors pers ← pers + 3 FinSi Afficher ("Saisir le temps de conjugaison souhaité (1 pour présent, 2 pour imparfait, 3 pour futur)") Saisir (temps) // Récupérer la racine du verbe racine ← sousChaîne (verbe, 1, longueur (verbe) – 2) Afficher (tabPers[pers] + "" + racine + tabConjug[pers, temps]) Fin

Exercice 83

Reprenez l'exercice précédent et modifiez-le de manière à ce qu'il affiche la conjugaison complète d'un verbe saisi au temps choisi.

Lexique	Conjugaison avec plusieurs temps et toutes les personnes
verbe (chaîne, saisi) : verbe à conjuguer. temps (entier, saisi) : temps de conjugaison choisi par l'utilisateur. racine (chaîne, calculé) : racine du verbe à conjuguer. SousChaîne (fonction) résultat chaîne : renvoie le découpage de la chaîne passée en paramètre à partir de la position spécifiée en 2ème paramètre sur la longueur spécifiée en 3ème paramètre. longueur (fonction) résultat entier : renvoie le nombre de caractères de la chaîne passée en paramètre. tabPers (Tableau [1..6] de chaîne, constante) = ("Je", "Tu", "Il, elle, on", "Nous", "Vous", "Ils, elles") tabConjug (Tableau [1..6, 1..3] de chaîne, saisi) : tableau contenant les terminaisons des verbes du premier groupe.	Début Afficher ("Saisissez le verbe à conjuguer") Saisir (verbe) Afficher ("Saisir le temps de conjugaison souhaité (1 pour présent, 2 pour imparfait, 3 pour futur)") Saisir (temps) // Récupérer la racine du verbe racine ← sousChaîne (verbe, 1, longueur (verbe) –2) Pour pers de 1 à 6 Faire Afficher (tabPers[pers] + "" + racine + tabConjug[pers, temps]) FinPour Fin

Exercice 84

Écrire l'algorithme de saisie des moyennes de x élèves à y matières.

Les noms des élèves seront saisis dans un tableau à part, les noms et les coefficients des matières aussi.

Lexique	Saisie des élèves, de leurs notes, des matières et des coefficients des matières
j (entier, calculé) : indice de parcours du tableau des matières de celui des coefficients et des colonnes du tableau des notes. tabMat (Tableau [1..y] de chaîne, saisi) : tableau des matières. tabCoeff (Tableau [1..y] d'entiers, saisi) : tableau des coefficients des matières. i (entier, calculé) : indice de parcours du tableau des élèves, de celui des coefficients et des lignes du tableau des notes. tabEleves (Tableau [1..x] de chaîne, saisi) : tableau des noms des élèves. tabNotes (Tableau [1..x, 1..y] d'entiers, saisi) : tableau contenant les moyennes des x élèves dans les y matières.	Début Afficher ("Saisie des matières et de leurs coefficients) Pour j de 1 à y Faire Afficher ("Saisissez la matière ainsi que son coefficient") Saisir (tabMat[j], tabCoeff[j]) FinPour Afficher ("Saisie des élèves et de leurs moyennes dans les différentes matières") Pour i de 1 à x Faire Afficher ("Saisissez le nom du", i, "° élève.") Saisir (tabEleves[i]) Afficher ("Saisie des moyennes de ", tabEleves[i], "dans les différentes matières") Pour j de 1 à y Faire Afficher ("Moyenne de ", tabMat[j]) Saisir (tabNotes[i,j]) FinPour FinPour Fin

Exercice 85

En utilisant les tableaux saisis dans l'exercice ci-dessus, écrire l'algorithme permettant, à partir d'un choix proposé à l'utilisateur, le calcul et l'affichage des moyennes :

- **d'un élève dont le nom a été saisi ;**
- **de chaque élève ;**
- **moyenne d'une matière, le nom de la matière ayant été saisi par l'utilisateur ;**
- **générale (moyenne des moyennes).**

Lexique	Calcul des moyennes

choix (entier, saisi) : choix de l'utilisateur.
sommeCoeff (entier, calculé) : somme des coefficients.
tabCoeff (Tableau [1..y] d'entiers, saisi) : tableau des coefficients des matières.
s, i, j (entiers, calculés) : indices d'itérations.
matiere (chaine, saisi) : nom de la matière.
nom (chaine, saisi) : nom de l'élève.
tabEleves (Tableau [1..x] de chaîne, saisi) : tableau des noms des élèves.
moyGen(entier, calculé) : moyenne générale.
somme (entier, calculé) : somme pondérée des notes de l'élève.
sommeMat (entier, calculé) : somme des notes dans une matière.
tabMat (Tableau [1..y] de chaîne, saisi) : tableau des matières.
	Début
	// Menu à afficher à l'utilisateur
	Afficher ("Pour calculer")
	Afficher ("La moyenne d'un élève dont le nom a été saisi, tapez 1")
	Afficher ("La moyenne de chaque élève, tapez 2")
	Afficher ("La moyenne dans une matière, le nom de la matière ayant été saisi par l'utilisateur, tapez 3")
	Afficher ("La moyenne générale (moyenne des moyennes), tapez 4")
	Afficher ("votre choix")
	Saisir (choix)
	// calcul de la somme des coefficients
	sommeCoeff ← tabCoeff[1]
	Pour s de 2 à y
	Faire sommeCoeff ← sommeCoeff + tabCoeff[s]
	FinPour
	// Traitements selon le choix de l'utilisateur
	Selon cas choix faire
	// Moyenne de l'élève dont le nom a été saisi
	1 : Afficher ("Saisissez le nom de l'élève")
	Saisir (nom)
	// Recherche de l'indice de l'élève dans tabEleves et
	// dans tabNotes
	i ← 1
	TantQue tabEleves[i] <> nom et i < x
	Faire i ← i + 1
	FinTantQue
	// Si on a trouvé l'élève, on calcule sa moyenne
	Si tabEleves[i] = nom
	Alors somme ← 0
	Pour j de 1 à y
	Faire somme ← somme + tabNotes[i,j] * tabCoeff[j]
	FinPour

Lexique	Calcul des moyennes

```
                    Afficher ("Moyenne de ", nom, " :",
                    somme / sommeCoeff)
            Sinon   Afficher ("Cet élève n'est pas répertorié.")
            FinSi
// Moyenne de chaque élève
2 :         Pour i de 1 à x
            faire   somme ← 0
                    Pour j de 1 à y
                    Faire   somme ← somme + tabNotes[i,j] *
                    tabCoeff[j]
                    FinPour
                    Afficher ("Moyenne de ", tabEleves[i], " :", somme
                    / sommeCoeff)
            FinPour
// Moyenne dans la matière saisie
3 :         Afficher ("Saisissez la matière")
            Saisir (matiere)
            // Recherche de l'indice de la matière
            j ← 1
            TantQue tabMat[j] <> matiere et j < y
            Faire   j ← j + 1
            FinTantQue
            Si tabMat[j] = matiere
            Alors   somme ← tabNotes[1,j]
                    Pour i de 2 à x
                    Faire somme ← somme + tabNotes[i,j]
                    FinPour
                    Afficher ("Moyenne de ", matiere, " :",
                    somme / x)
            Sinon   Afficher ("Matière non répertoriée.")
            FinSi
// Moyenne générale, on peut faire la moyenne des matières,
// ou la moyenne des élèves, ça revient au même.
```

Lexique	Calcul des moyennes
	4 : moyGen ← 0 Pour j de 1 à y Faire sommeMat ← 0 Pour i de 1 à x Faire sommeMat ← sommeMat + tabNotes[i,j] FinPour moyGen ← moyGen + sommeMat*tabCoeff[j] FinPour Afficher ("Moyenne générale :", moyGen / x) Fin

Facile, mais long. Il faut savoir faire ça aussi. Intellectuellement, c'est plus reposant que les algos de tris ou autres trucs biscornus, mais il faut rester très rigoureux pour ne rien omettre.

Exercice 86

Écrire aussi l'algorithme qui trie le tableau des notes et celui des élèves par ordre croissant sur la moyenne d'algorithme (ça, c'est bien un énoncé de prof, tout ce qu'il y a de plus classique !).

Lexique	Tri d'un tableau à deux dimensions

j, i, indTemp, k, indIns (entiers, calculés) : indices de parcours et d'itération.
tabMat (Tableau [1..y] de chaîne, saisi) : tableau des matières.
nom (chaîne, calculée) : variable temporaire contenant le nom de l'élève courant.
temp (Tableau [1..y] d'entiers) : contient la ligne de notes courante.
tabEleves (Tableau [1..x] de chaîne, saisi) : tableau des noms des élèves.
tabNotes (Tableau [1..x, 1..y] d'entiers, saisi) : tableau contenant les moyennes des x élèves dans les y matières.
	Début
	// Recherche de l'indice de la matière "algorithme" dans tabMat
	j ← 1
	TantQue tabMat[j] <> "Algorithme" et j < y
	Faire j ← j + 1
	FinTantQue
	// j contient l'indice de la matière "Algorithme"
	// On trie les éléments de tabNotes et de tabEleves suivant la
	// valeur de tabNotes[j]. J'ai choisi un tri par insertion sauf
	// qu'on sauvegarde les données à déplacer dans un tableau
	Pour i de 2 à x
	Faire // Sauvegarde de la ligne de notes courante dans un // tableau
	Pour indTemp de 1 à y
	Faire temp[indTemp] ← tabNotes[i,indTemp]
	FinPour
	// Sauvegarde du nom de l'élève courant

Lexique	Tri d'un tableau à deux dimensions

```
        nom ←tabEleves[i]
        k ← i – 1
        TantQue k >=1 et temp[k] < tabNotes[k,j]
        Faire    // Déplacement d'une ligne
                 Pour indDep de 1 à y
                 Faire tabNotes[k + 1, indDep] ← tab[k, indDep]
                 FinPour
                 // Déplacement du nom
                 tabEleves[k + 1] ← tabEleves[k]
                 k ← k – 1
        FinTantQue
        // On insère
        Pour indIns de 1 à y
        Faire tabNotes[k + 1, indIns] ← temp[indIns]
        FinPour
        tabNom[k +1] ← nom
FinPour
Fin
```

Séquence 6

Les sous-programmes : modules, procédures et fonctions

Partie 1

Différence entre module, procédure et fonction

Exercice 87

Écrire l'algorithme du sous-programme qui effectue une recherche séquentielle et renvoie la position d'un élément dans un tableau à une dimension, indicé à partir de 0 et contenant des chaînes de caractère.

On passe en paramètre à ce sous-programme : le nom du tableau, la valeur de l'élément cherché, l'indice de début de la recherche et l'indice de fin de la recherche.

Si l'élément cherché ne figure pas dans le tableau, ce sous-programme renvoie la valeur –1.

Lexique	Fonction rechSeq (monTab : tableau [0..99] de chaîne, elemCherch : chaîne, indDeb, indFin : entiers) : entier
	Début TantQue elemCherch <> monTab[indDeb] et indDeb < indFin - 1 Faire indDeb ← indDeb + 1 Fin Si elemCherch = monTab[indDeb] Alors Renvoyer indDeb Sinon Renvoyer –1 FinSi

Exercice 88

Réécrire les exercices 65 et 68 en utilisant le sous-programme que vous avez écrit à l'exercice 87.

Exo 65 : afficher la position d'un mot dans le tableau, le mot étant saisi, avec affichage d'un message si le mot ne figure pas dans le tableau.

Lexique	Version qui utilise la fonction rechSeq

mot (chaîne, saisi) : mot dont l'utilisateur veut connaître la position dans le tableau.
resRecherch (entier, calculé) : résultat d'appel de la fonction de recherche d'indice.
rechSeq (fonction) : retourne l'indice du mot passé en paramètre dans le tableau passé en paramètre.
tabMots (tableau[0..99]de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
taille (fonction) résultat entier : retourne le nombre effectif d'éléments contenu dans le tableau passé en paramètre.
	// Voir le début dans l'exo précédent
	…
	2 : Afficher ("Saisissez le mot dont vous souhaitez connaître la position")
	Saisir (mot)
	resRecherch ← rechSeq (tabMots, mot, 0, taille(tabMots)
	Si resRecherche <> -1
	Alors Afficher ("Le mot que vous avez saisi se trouve à la position n° ", resRecherch + 1, " dans le tableau.")
	Sinon Afficher ("Ce mot ne figure pas dans le tableau")
	FinSi
	// Voir suite exo suivant

On aurait pu remplacer les lignes :

resRecherch ← rechSeq (tabMots, mot, 0, taille(tabMots)
Si resRecherche <> -1
Alors Afficher ("Le mot que vous avez saisi se trouve à la position n° ", resRecherch + 1, " dans le tableau.")

Par
Si resRecherch ← rechSeq (tabMots, mot, 0, taille(tabMots)
Alors Afficher ("Le mot que vous avez saisi se trouve à la position n° ", resRecherch ← rechSeq (tabMots, mot, 0, taille(tabMots)+ 1, " dans le tableau.")

C'est juste mais moins correct que la première version car vous pouvez voir qu'on appelle deux fois la fonction avec les mêmes paramètres (ce qui va donc provoquer deux fois de suite le calcul du même résultat). Il vaut mieux dans ce cas ne l'appeler qu'une fois, ranger son résultat dans une variable, et utiliser ensuite la variable partout où on veut utiliser le résultat d'appel de la fonction.

Exo 68 : supprimer du tableau la première occurrence (= la première apparition) du mot qui a été saisi.

Lexique	Version qui utilise la fonction rechSeq

mot (chaîne, saisi) : mot à supprimer.
resRecherch (entier, calculé) : résultat d'appel de la fonction de recherche d'indice.
rechSeq (fonction) : retourne l'indice du mot passé en paramètre dans le tableau passé en paramètre.
tabMots (tableau[0..99]de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
	// Voir le début dans l'exo précédent
	…
	5 : Si taille(tabMots) <> 0
	Alors Afficher ("Saisissez le mot dont vous souhaitez la suppression")
	Saisir (mot)
	resRecherch ← rechSeq (tabMots, mot, 0, taille(tabMots))
	Si resRecherch <> -1
	Alors Pour i de resRecherch + 1 à taille(tabMots) - 1
	Faire tabMots[i – 1] ← tabMots[i]
	FinPour
	Sinon Afficher ("Ce mot n'est pas dans le tableau")
	FinSi
	Sinon Afficher("Le tableau est vide, aucune suppression possible")
	FinSi
	// Voir suite exo suivant.

Remarque : il y a un autre ensemble d'instructions qu'on va transformer en sous-programme, c'est l'ensemble des instructions qui permettent de décaler les éléments d'un tableau d'une case vers la gauche ou vers la droite.

On fera ça à la suite des deux parties de cours suivantes.

Autre remarque : on peut aussi réécrire l'exercice 66 en utilisant la fonction **rechSeq**.

Partie 2
Les paramètres

Exercice 89

Écrire un sous-programme, que vous appellerez « remplir », qui permet de saisir des données dans le tableau passé en paramètres et renvoie également comme résultat le nombre de mots effectivement saisis dans le tableau.

Ce tableau peut contenir jusqu'à 100 chaînes de caractères et l'indice de son premier élément est 0.

L'utilisateur indique par la saisie d'une chaîne vide qu'il ne désire plus saisir de mot.

Attention : ce sous-programme doit permettre de saisir des mots dans un tableau, même non vide. Dans ce cas, les mots saisis sont rajoutés à la suite des mots déjà présents dans le tableau.

Voici ci-dessous la partie de l'exercice 63 pouvant nous servir à écrire correctement notre procédure **remplir** :

```
…
Afficher ("Saisissez votre premier mot")
Saisir (tabMots[0])
// A partir d'ici, taille(tabMots) renvoie 1
TantQue taille(tabMots) < 100 et
tabMots[taille(tabMots)-1] <> ""
Faire   Afficher ("Saisissez le " +
        taille(tabMots) + 1 + "° mot (saisir un
        mot vide pour sortir)")
        Saisir (tabMots[taille(tabMots)])
FinTantQue
…
```

Une remarque au passage : le dernier mot pris en compte étant une chaîne vide, on peut prévoir une instruction qui fasse en sorte que ce mot ne soit pas pris en compte dans la collection, ceci afin que la collection de mots ne contienne pas de mot valant la chaîne vide. Je n'ai pas prévu cette instruction dans le corrigé de l'exercice 63, alors je vous propose ici d'adapter l'algorithme de l'exercice 63.

On a deux solutions pour faire en sorte que le dernier mot saisi, qui vaut la chaîne vide, ne soit pas pris en compte.

- On rajoute une variable intermédiaire leMot, qui contient le dernier mot saisi par l'utilisateur, que l'on ne rajoute dans le tableau que si ce dernier mot n'est pas la chaîne vide. Ca donne :

```
…
Saisir (leMot)
TantQue taille(tabMots) < 100 et leMot <> ""
Faire   tabMots[taille(tabMots)] ← leMot
Afficher ("Saisissez le " + taille(tabMots) + 1 +
"° mot (saisir un mot vide pour sortir)")
Saisir (leMot)
FinTantQue
…
```

• On supprime systématiquement du tableau le dernier mot saisi s'il vaut la chaîne vide.

```
…
TantQue taille(tabMots) < 100 et tabMots[taille(tabMots)-1] <> ""
Faire     Afficher ("Saisissez le " + taille(tabMots) + 1 + "° mot (saisir un mot vide pour sortir)")
          Saisir (tabMots[taille(tabMots)])
FinTantQue
Si tabMots[taille(tabMots) – 1] = ""
Alors supprimer(tabMots, taille(tabMots) – 1)
…
```

On supposera ici que la procédure **supprimer** utilisée a pour effet de supprimer du tableau passé en paramètre l'élément dont l'indice est passé en paramètre.

Dans le corrigé ci-dessous, je choisis la première des deux solutions proposées ci-dessus.

Lexique	Procédure remplir (données modifiées leTableau : tableau [0..99] de chaîne, nbCh : entier)
leMot (chaîne, saisi) : mot courant à ranger dans le tableau.	Début Afficher ("Saisissez votre", nbCh + 1, "° mot") Saisir (leMot) TantQue taille(leTableau) < 100 et leMot <> "" Faire leTableau[taille(leTableau)] ← leMot Afficher ("Mot suivant") Saisir (leMot) FinTantQue nbCh ← taille(leTableau) Fin

Remarque : on a déclaré une variable dans le lexique de la procédure. Il s'agit d'une variable **locale** à la procédure, c'est-à-dire non visible à l'extérieur de la procédure.

À la fin de cette procédure, **nbCh** et **leTableau**, qui ont été modifiés à l'intérieur de cette procédure, sont également modifiés dans le programme appelant, car ils ont été passés par adresse à la procédure.

Si ces paramètres avaient été passés par valeur, les modifications qu'ils ont subi à l'intérieur de la procédure n'auraient pas été répercutées dans l'algo ou le programme appelant.

Exercice 90

Réécrire l'algo de l'exercice 63 de la séquence 5 en appelant la procédure « remplir ».

Voici l'énoncé de l'exercice 63 : soit un tableau pouvant contenir 100 mots. Le programme commence par permettre à l'utilisateur de saisir un certain nombre de mots, l'utilisateur indiquant par la saisie d'un mot vide (appui sur la touche entrée sans avoir rien saisi au préalable) qu'il ne désire plus saisir de mots.

Ne réécrivez pas les instructions proposant les différents choix à l'utilisateur.

Lexique	**Saisie des mots dans le tableau** Version qui utilise la procédure remplir
tabMots (tableau[0..99]de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur. nbMots (entier, calculé) : nombre effectif de mots saisis dans le tableau et indice de la première case libre dans le tableau. remplir (procédure) : permet la saisie de mots dans le tableau spécifié à partir de l'indice spécifié, et retourne le tableau et le nombre de mots qu'il contient.	Début nbMots ← taille(tabMots) remplir (tabMots, nbMots) Fin

Partie 3

Les variables locales

Exercice 91

Écrire l'algo générique d'un sous-programme qui renvoie le plus grand des éléments du tableau passé en paramètre (générique signifie ici qu'on ne vous demande pas de préciser les bornes du tableau, ni le type des éléments). La recherche du maximum s'effectue entre les bornes minimales et maximales passées en paramètre.

Lexique	Fonction fMaxi (leTableau : tableau [?..?] de type , mini, maxi : même type que les bornes du tableau) : type
maximum (type, calculé) : maximum courant. i (même type que les bornes du tableau, calculé) : indice de parcours du tableau.	Début maximum ← leTableau[mini] Pour i de mini +1 à maxi Faire Si leTableau[i] > maximum Alors maximum ← leTableau[i] FinSi FinPour Renvoyer maximum Fin

Exercice 92

Écrire l'algorithme permettant de saisir des températures dans un tableau et affichant la plus haute des températures en utilisant le sous-programme écrit précédemment. Vous adapterez la description du sous-programme utilisé en fonction du type des paramètres et de la taille du tableau.

Lexique	Saisie de températures et affichage de la plus haute température saisie

nbTemp (entier, calculé) : nombre de températures saisies.
remplir (procédure) : permet de saisir des éléments dans le tableau spécifié. Retourne également le nombre d'éléments du tableau.
tabTemper (tableau[1..50] d'entiers, saisi) : tableau des températures.
fMaxi (fonction) : retourne le plus grand élément du tableau passé en paramètre. La recherche s'effectue entre les deux indices passés en paramètres.
	Début
	// Le tableau étant vide, on appelle la procédure remplir
	// en lui passant nbTemp qui vaut 0
	nbTemp ← 0
	remplir (tabTemper, nbTemp)
	Afficher ("La température maximale saisie est de ", fMaxi (tabTemper, 1, nbTemp))
	// ou bien : fMaxi (tabTemper, 1, taille(tabTemper))
	// On recherche la plus grande température dans la partie
	// du tableau où des températures ont effectivement
	// été saisies.
	Fin

Certains ou certaines d'entre vous se demandent peut-être pourquoi on n'appelle pas directement **remplir** en faisant comme ça **remplir(TabTemper, 0)** au lieu d'initialiser **nbTemp** à **0** et d'appeler ensuite **remplir** en lui passant **nbTemp** qui vaut **0**.

Si **nbTemp** était une *donnée utilisée*, on pourrait faire comme ça. Mais là, **nbTemp** est une *donnée modifiée*, sa valeur est modifiée à l'intérieur de la procédure, elle est incrémentée à chaque fois qu'un nouvel élément a été saisi dans le tableau. Or, on ne peut pas écrire *0 ← 0 + 1*. L'incrémentation d'une valeur ne peut se faire que si cette valeur est rangée dans une variable.

En outre, en sortie de procédure, on a besoin d'une variable portant un nom et qui contient le nombre d'éléments saisis dans le tableau. Voilà, c'est pour ça qu'on fait comme ça.

D'ailleurs, vous voyez bien que dans l'algo précédent on a écrit *fMaxi(tabTemper,1, nbTemp)*. Ici, le deuxième paramètre, qui est la borne où débute la recherche du maximum, est une constante. On a passé une constante comme paramètre à la fonction **fMaxi**. Ici, ce n'est pas gênant car ce paramètre est une *donnée utilisée*, les éventuelles modifications que ce paramètre peut subir à l'intérieur de la fonction ne seront pas répercutées à l'extérieur de la fonction. Ce paramètre n'a donc pas besoin de porter un nom de variable car on n'aura pas besoin d'utiliser sa valeur éventuellement modifiée après appel de la fonction.

Et la procédure **remplir** dans tout ça ? Sa description n'est forcément pas tout à fait la même que celle qu'on avait fait à l'exercice 89.

Lexique	**Procédure remplir (Donnée modifiées leTableau : tableau [1..50] d'entiers, nbCh : entier)**
rep (chaîne, saisie) : réponse de l'utilisateur.	Début Afficher ("Saisir une valeur ?(oui/non)") Saisir (rep) TantQue taille(leTableau) < 50 et rep = "oui" Faire Afficher ("Saisissez votre valeur.") saisir (leTableau[taille(leTableau) + 1]) Afficher ("Encore une valeur ?") Saisir (rep) FinTantQue nbCh ← taille(leTableau) Fin

Exercice 93

Modifier la fonction fMaxi de manière à ce qu'elle renvoie l'indice du plus grand élément, appelez la fMaxiBis.

Écrire ensuite un algo qui trie sur lui-même, dans l'ordre décroissant, le tableau tabElem, en utilisant la fonction fMaxiBis et la procédure permuter.

Lexique	**Fonction fMaxiBis (feTableau : tableau [?..?] de type, mini, maxi : entier) : entier**
indMax (type, calculé) : indice du maximum courant. i (même type que les bornes du tableau, calculé) : indice de parcours du tableau.	Début indMax ← mini Pour i de mini +1 à maxi Faire Si leTableau[i] > leTableau[indMax] Alors indMax ← i FinSi FinPour Renvoyer indMax Fin

Lexique	**Tri avec appel de FMaxiBis et de permuter**
i (entier, calculé) : indice de parcours du tableau. tabElem (tableau [1..?] de ?) : tableau à trier. permuter (procédure) : permute les valeurs des deux paramètres. taille (fonction) résultat entier : retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre. fMaxiBis (fonction) : retourne l'indice du plus grand élément dans le tableau passé en paramètre, entre les deux bornes passées en paramètres.	Début Pour i de 1 à taille(tabElem) Faire permuter (tabElem[i], tabElem[fMaxiBis(tabElem ,i, taille(tabElem))]) FinPour Fin

Expliquons l'instruction

permuter(tabElem[i], tabElem[fMaxiBis (tabElem, i, taille(tabElem))]).

La méthode de tri, je vous le rappelle, consiste à chercher le plus grand et à le mettre à l'indice courant, en permutant le plus grand avec l'élément courant.

L'instruction **permuter(tabElem[i], tabElem[fMaxiBis (tabElem, i, taille(tabElem))])** peut être décomposée en deux instructions successives :

// recherche de l'indice du ième plus grand élément

indMaxi ← fMaxiBis(tabElem, i, taille(tabElem))

// permutation des positions du ième plus grand élément et de l'élément courant

permuter(tabElem[i], tabElem[indMaxi]).

Exercice 94

Écrire la fonction sousChaîne de l'algo. Cette fonction retourne la partie de la chaîne de caractères spécifiée, à partir du caractère spécifié, de la longueur spécifiée. Lors de l'appel de la fonction, si la position de départ dépasse la longueur de la chaîne, la fonction renvoie la chaîne vide. Si la longueur spécifiée est supérieure à la longueur possible, la fonction renvoie la partie de la chaîne de caractères à partir de la position spécifiée.

Lexique	Fonction sousChaîne (leTexte : chaîne, deb, nbCar : entiers) : chaîne
res (chaîne, calculé) : résultat de la copie, renvoyé par la fonction au programme appelant. long (entier, calculé) : longueur du texte passé en paramètre. limite (entier, calculé) : valeur que deb ne doit pas dépasser. longueur (fonction) résultat entier : renvoie la longueur de la chaîne passée en paramètre.	Début res ← "" long ← longueur (leTexte) limite ← deb + nbCar - 1 TantQue deb <= long et deb <= limite Faire res ← res + leTexte[deb] deb ← deb + 1 FinTantQue Renvoyer res Fin

Cet exercice est un exemple de fonction qui appelle une autre fonction. En effet, on y appelle la fonction **longueur** afin de connaître la longueur effective de la chaîne.

On calcule la valeur de la variable **limite** à l'aide de l'instruction **limite ← deb + nbCar – 1**, car **deb**, une fois qu'on est entré dans la boucle, change de valeur à chaque tour de boucle.

On ne peut donc pas écrire la condition d'entrée comme ça :

TantQue deb <= long et deb <= deb + nbCar – 1.

En effet, puisque **deb** change de valeur à chaque passage dans le TantQue, la condition **deb + nbCar – 1,** ne sera jamais une condition d'arrêt.

Soyez attentifs, c'est difficile à expliquer.

Nous, quand on écrit **deb + nbCar – 1**, de part et d'autre du signe <=, on ne parle pas du même **deb**.

À gauche du signe <=, on parle de **deb** tel qu'il est après chaque passage dans la boucle. À droite du signe <=, on parle de **deb** tel qu'il est quand on arrive dans la fonction.

Seulement l'ordinateur lui, il ne peut pas deviner qu'on ne parle pas du même **deb**. Pour lui, deb c'est **deb**. Donc, pour l'aider à faire la différence entre les deux **deb**, on change le nom du **deb** initial. D'où l'instruction **limite ← deb + nbCar – 1** avant l'entrée dans la boucle.

Exercice 95

Écrire le sous-programme libereCase, qui décale les éléments du tableau passé en paramètre d'une case vers la droite entre les deux bornes spécifiées.

Il fallait bien sûr choisir une procédure, car on n'a pas besoin de récupérer le résultat dans une nouvelle variable, mais on a besoin de récupérer le tableau modifié.

Lexique	**Procédure libereCase (Donnée modifiée leTableau : tableau [?..?] de ?, données utilisées borneMin, borneMax : entiers)**
i (entier, calculé) : indice de parcours.	Début Pour i de borneMax à borneMin (diminuer de 1) Faire leTableau[i + 1] ← leTableau[i] FinPour Fin

Exercice 96

Écrire le sous-programme rechDicho qui effectue la recherche dichotomique de l'indice de l'élément passé en paramètre dans le tableau passé en paramètre, entre les deux bornes passées en paramètre. Comme pour la fonction rechSeq, si l'élément n'est pas dans le tableau, la fonction renvoie –1.

Lexique	**Fonction rechDicho (leTableau : tableau[?..?] de ?, elem : ?, borneInf, borneSup : entiers)**
milieu (entier, calculé) : milieu du tableau.	Début TantQue borneSup >= borneInf Faire milieu ← (borneInf + borneSup) div 2 Si elem < leTableau[milieu] Alors borneSup ← milieu – 1 Sinon borneInf ← milieu + 1 FinSi FinTantQue Si elem = leTableau[borneInf] Alors Renvoyer borneInf Sinon Renvoyer –1 FinSi Fin

Exercice 97

Réécrire les exos 74, 75 de la séquence précédente en utilisant, si possible les sous-programmes précédemment écrits.

Exo 74 : écrire l'algorithme qui affiche dans l'ordre décroissant des nombres saisis dans un ordre quelconque. Ces nombres sont saisis dans un tableau, le tri se fait au fur et à mesure de la saisie.

Cet énoncé d'exo comporte un piège… On a bien envie d'utiliser la fonction **rechSeq** pour réécrire l'algorithme de l'exercice 74 sauf que…d'insérer un élément dans le tableau, il y a peu de chance que l'élément se trouve déjà dans le tableau. Donc, la fonction **rechSeq** va nous retourner **-1**. Donc, la fonction **rechSeq** ne nous fournira pas l'indice d'insertion de l'élément à insérer.

Ce qu'il nous faut, c'est une fonction qui recherche **l'indice théorique d'insertion** d'un élément dans un tableau, cet élément ne se trouvant pas dans le tableau.
Je choisis d'appeler cette fonction **rechIndSeq**.

Lexique	Fonction rechIndSeq (monTab : tableau [0..99] de chaîne, elemCherch : chaîne, indDeb, indFin : entiers) : entier
	Début TantQue elemCherch > monTab[indDeb] et indDeb < indFin - 1 Faire indDeb ← indDeb + 1 Fin Renvoyer indDeb FinSi

Lexique	Tri élémentaire lors de la saisie
tab (tableau de[1..20] entiers, saisi). i (entier, calculé) : indice d'itérations. nombre (entier, saisi) : nombre courant à insérer. ind (entier, calculé) : indice d'insertion. rechIndSeq (fonction) : renvoie l'indice d'insertion de l'élément passé en paramètre, dans le tableau passé en paramètre, en effectuant une recherche séquentielle. libereCase (procédure) : libère la case passée en paramètre.	Début Afficher ("Saisissez le premier nombre") Saisir (tab[1]) Pour i de 2 à 20 Faire Afficher ("Saisissez le ", i, "° nombre") Saisir (nombre) ind ← rechIndSeq (tab, nombre, 1, i-1) libereCase (tab, ind, i-1) tab[ind] ← nombre FinPour Fin

Exo 75 : écrire l'algorithme qui affiche dans l'ordre croissant des nombres saisis dans un ordre quelconque. Ces nombres sont saisis dans un tableau, le tri se fait au fur et à mesure de la saisie, à l'aide d'une recherche dichotomique puis d'un décalage et d'une insertion à l'emplacement libéré.

Pour cet exercice, le problème est le même que précédemment, mais avec la fonction **rechDicho.**

Voici donc la fonction **rechDicho** modifiée de manière à ce qu'elle renvoie l'indice d'insertion de l'élément dans le tableau, j'ai appelé cette nouvelle fonction **rechIndDico.**

Lexique	Fonction rechIndDicho (leTableau : tableau[?..?] de ?, elem : ?, borneInf, borneSup : entiers)
milieu (entier, calculé) : milieu du tableau.	Début TantQue borneSup >= borneInf Faire milieu ← (borneInf + borneSup) div 2 Si elem < leTableau[milieu] Alors borneSup ← milieu – 1 Sinon borneInf ← milieu + 1 FinSi FinTantQue Renvoyer borneInf Fin

Lexique	Insertion dichotomique au fur et à mesure de la saisie

i (entier, calculé) : nombre d'éléments du tableau et compteur d'itérations.
nombre (entier, saisi) : nombre courant à insérer.
indIns (entier, calculé) : indice d'insertion du nombre saisi.
tab (tableau [1..20] d'entiers, saisi) : tableau à remplir et à trier
rechIndDicho (fonction) : renvoie l'indice d'insertion de l'élément passé en paramètre, dans le tableau passé en paramètre, en effectuant une recherche dichotomique.
libereCase (procédure) : libère la case passée en paramètre.
	Début
	Pour i de 1 à 20
	Faire Afficher ("Saisissez le ", i, "° nombre")
	Saisir (nombre)
	indIns ← rechIndDicho (tab, nombre, 1, i-1)
	// Décalage
	libereCase (tab, indIns, i-1)
	//Insertion
	tab[indIns] ← nombre
	FinPour
	Fin

Exercice 98

Écrire le sous-programme ecraseCase, qui écrase la case de tableau dont l'indice a été passé en paramètre en décalant les éléments d'une case vers la gauche. On passe également en paramètre l'indice de la dernière case à décaler.

Lexique	Procédure EcraseCase (donnée modifiée leTableau : tableau [?..?] de ?, données utilisées indiceCase, indDernier : entiers)
i (entier, calculé) : indice de parcours.	Début Pour i de indiceCase à indDernier - 1 (augmenter de 1) Faire leTableau[i] ← leTableau[i + 1] FinPour Fin

Exercice 99

Réécrire les exos 68 et 70 de la séquence précédente en utilisant le sous-programme ecraseCase et éventuellement d'autres sous-programmes.

Exo 68 : supprimer du tableau la première occurrence (= la première apparition) du mot qui a été saisi.

Lexique	Exo 68

taille (fonction) : renvoie le nombre effectif d'éléments dans le tableau spécifié.
mot (chaîne, saisi) : mot à supprimer.
resRecherch (entier, calculé) : résultat d'appel de la fonction de recherche d'indice.
rechSeq(fonction) : retourne l'indice du mot passé en paramètre dans le tableau passé en paramètre.
tabMots (tableau [0..99] de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
ecraseCase (procédure) : écrase la case dont l'indice est passé en second paramètre.
	// Voir le début dans l'exo précédent
	Si taille(tabMots) <> 0
	Alors Afficher ("Saisissez le mot dont vous souhaitez la suppression")
	Saisir (mot)
	resRecherch ← rechSeq (tabMots, mot, 0, taille(tabMots))
	Si resRecherch <> -1
	Alors ecraseCase (tabMots, resRecherch, taille(tabMots) - 1)
	Sinon Afficher ("Ce mot n'est pas dans le tableau")
	FinSi
	Sinon Afficher("Le tableau est vide, aucune suppression possible.")
	FinSi
	// Voir suite exo suivant

Exo 70 : supprimer du tableau le mot « saisi » autant de fois qu'il se trouve dans le tableau.

Lexique	Exo 70

taille (fonction) : renvoie le nombre effectif d'éléments dans le tableau spécifié.
mot (chaîne, saisi) : mot à insérer dans le tableau.
ind (entier, calculé) : indice de parcours du tableau.
rechSeq (fonction) : retourne l'indice du mot passé en paramètre dans le tableau passé en paramètre.
tabMots (tableau[0..99]de chaîne, saisi) : tableau contenant les mots saisis par l'utilisateur.
ecraseCase (procédure) : écrase la case dont l'indice est passé en second paramètre.
	// Voir le début dans l'exo précédent
	…
	Si taille(tabMots) <> 0
	Alors Afficher ("Saisissez le mot dont vous souhaitez supprimer toutes les occurrences")
	Saisir (mot)
	ind ← 0
	TantQue ind < taille(tabMots) et ind <> -1
	Faire // recherche d'une occurrence
	ind ← rechSeq (tabMots, mot, ind, taille(tabMots))
	// si au lieu de l'instruction précédente, on écrit
	// ind ← rechSeq(tabMots, mot, 0, taille(tabMots)), alors la
	// recherche repart depuis le début et dans ce cas,
	// l'instruction ind ← 0 est inutile
	// suppression de l'occurrence trouvée
	Siind <> -1
	Alors ecraseCase (tabMots, ind, taille(tabMots) - 1)
	FinSi
	FinTantQue
	Sinon Afficher("Le tableau est vide, aucune suppression possible")
	FinSi
	// Voir suite exo suivant

Séquence 7

Les types créés par le développeur

Partie 1

Création d'un nouveau type

Exercice 100

Créer TypPers, le nouveau type de données correspondant aux informations à inscrire dans le carnet d'adresses.

TypPers (type) : Structure
persNom (chaîne, saisie) : nom de la personne.
persPrenom (chaîne, saisie) : prénom de la personne.
persTelephone (chaîne, saisie) : numéro de téléphone.
persNomRue (chaîne, saisie) : numéro et nom de la rue.
persCodePostal (entier, saisi) : code postal.
persVille (chaîne, saisie) : ville.
persEmail (chaîne, saisie) : adresse e-mail.
FinStructure.

Remarque : on pouvait aussi créer un type **TypAdresse** comme vu en cours.

Exercice 101

Écrire le traitement permettant de remplir ou de compléter un tableau de type TypPers à l'aide de données saisies au clavier, l'utilisateur indique en appuyant sur « entrée » qu'il ne désire plus saisir de personne dans son carnet d'adresses, et ceci lorsque le programme lui demande de saisir le nom de la personne. On considère que le tableau utilisé n'est pas limité en taille et que l'indice de son premier élément est 1.

Lexique	Partie de l'algo permettant de remplir ou compléter le tableau des personnes
rep (caractère, saisi) : réponse de l'utilisateur. tabPers (tableau de TypPers) : tableau des personnes du carnet d'adresses. indCour (entier, calculé) : indice courant d'insertion dans le tableau. TypPers (type) : Structure persNom (chaîne , saisie) : nom de la personne persPrenom (chaîne , saisie) : prénom de la personne persTelephone (chaîne , saisie) : numéro de téléphone persNomRue (chaîne , saisie) : numéro et nom de la rue persCodePostal (entier, saisi) : code postal persVille (chaîne , saisie) : ville persEmail (chaîne , saisie) : adresse e-mail FinStructure	….. Afficher ("Désirez-vous saisir une personne dans votre carnet d'adresses (o/n)") Saisir (rep) TantQue rep = "o" Faire indCour ← taille(tabPers) + 1 Afficher ("Saisir le nom de la personne") Saisir (tabPers[indCour].persNom) Afficher ("Saisir le prénom") Saisir (tabPers[indCour].persPrenom) // et ainsi de suite pour tous les champs de TypPers Afficher ("Une autre personne ? (o/n)") Saisir (rep) FinTantQue ….

Exercice 102

Écrire le traitement qui affiche le contenu du carnet d'adresse.

Lexique	Affichage du contenu du carnet d'adresse
Le lexique de cet exo est rigoureusement le même que celui de l'exo précédent avec en plus : i (entier, calculé) : compteur.	…. Pour i de 1 à taille(tabPers) Faire Afficher (tabPers[i]. persNom, tabPers[i]. persPrenom , tabPers[i].persTelephone, tabPers[i].persNomRue, tabPers[i]. persCodePostal, tabPers[i].persVille, tabPers[i].persEmail) FinPour ….

Exercice 103

Afficher la liste des personnes du carnet d'adresses qui habitent dans la ville saisie par l'utilisateur.

Lexique	Affichage du contenu du carnet d'adresse
Le lexique de cet exo est rigoureusement le même que celui de l'exo précédent avec en plus : villeSaisie (chaîne, saisie) : ville saisie par l'utilisateur.	.... Afficher ("Saisissez la ville") Saisir (villeSaisie) Pour i de 1 à taille(tabPers) Faire Si tabPers[i].persVille = villeSaisie Alors Afficher (tabPers[i]. persNom, tabPers[i]. persPrenom , tabPers[i].persTelephone, tabPers[i].persNomRue, tabPers[i].persCodePostal, tabPers[i].persVille, tabPers[i].persEmail) FinSi FinPour ....

Exercice 104

Afficher les coordonnées de la personne dont le nom a été saisi.

Lexique	Affichage des coordonnées de la personne dont le nom a été saisi
Le lexique de cet exo est rigoureusement le même que celui de l'exo précédent avec en plus : nomSaisi (chaîne, saisie) : nom de la personne dont l'utilisateur veut les coordonnées.	.... Afficher ("Saisissez le nom") Saisir (nomSaisi) i ← 1 TantQue tabPers[i].persNom <> nomSaisi et i < taille(tabPers) Faire i ← i + 1 FinTantQue Si tabPers[i].persNom = nomSaisi Alors Afficher (tabPers[i]. persNom, tabPers[i].persPrenom, tabPers[i].persTelephone, tabPers[i].persNomRue, tabPers[i].persCodePostal, tabPers[i].persVille, tabPers[i].persEmail) Sinon Afficher ("Cette personne n'est pas dans votre carnet d'adresse.") FinSi ....

Remarque : dans cet version d'algorithme, si plusieurs personnes portent le même nom, ce sont toujours les coordonnées de la première personne portant ce nom qui s'affichent.

Pour que ce traitement soit réaliste, il faudrait rechercher toutes les personnes portant ce nom et les afficher.

Exercice 105

Modifier l'adresse de la personne dont le nom a été saisi.

Lexique	Modification de l'adresse de la personne dont le nom a été saisi
Le lexique de cet exo est rigoureusement le même que celui de l'exo précédent.	.... Afficher ("Saisissez le nom") Saisir (nomSaisi) i ← 1 TantQue tabPers[i].persNom <> nomSaisi et i < taille(tabPers) Faire i ← i + 1 FinTantQue Si tabPers[i].persNom = nomSaisi Alors Afficher ("Saisissez la nouvelle adresse (le nom de la rue, le code postal, puis la ville)") Saisir (tabPers[i].persNomRue, tabPers[i].persCodePostal, tabPers[i].persVille) Sinon Afficher ("Cette personne n'est pas dans votre carnet d'adresse.") FinSi ....

Exercice 106

Supprimer la personne dont le nom a été saisi, en décalant les éléments pour qu'il n'y ait pas de trou.

Lexique	Suppression de la personne dont le nom a été saisi
Le lexique de cet exo est rigoureusement le même que celui de l'exo précédent, avec en plus : j (entier, calculé) : compteur. EcraseCase (procédure) : supprime la case d'indice spécifié en deuxième paramètre.	.... Afficher ("Saisissez le nom") Saisir (nomSaisi) i ← 1 TantQue tabPers[i].persNom <> nomSaisi et i < taille(tabPers) Faire i ← i + 1 FinTantQue Si tabPers[i].persNom = nomSaisi Alors EcraseCase (tabpers, i, taille(tabPers)) Sinon Afficher ("Cette personne n'est pas dans votre carnet d'adresse.") FinSi ....

Partie 2

Plus loin avec les tableaux de structure

Exercice 107

Afficher le contenu du tableau tabPersBis, de type TypPersBis.

Considérer que tabPersBis contient déjà cptPers enregistrements valides, et qu'il contient également des enregistrements qui ont été supprimés logiquement.

Lexique	SELECT * FROM tabPersBis WHERE valable = vrai Version avec itération sur le nombre réel d'enregistrements

tabPersBis (tableau de TypPersBis) : tableau des personnes (voir la description du type TypPersBis dans le cours).
cptPers (entier, calculé) : nombre d'enregistrements valides.
i (entier, calculé) : compteur d'itérations.
	Début
	…
	Pour i de 1 à taille (tabPersBis)
	Faire Si tabPersBis[i]. valable
	Alors Afficher (tabPersBis[i].persNom, tabPersBis[i].
	prenom,….., tabPersBis[i].persEmail)
	FinSi
	// On affiche tous les renseignements, sauf la valeur
	// du champ **valable**, qui n'est pas une information
	// pertinente pour l'utilisateur.
	FinPour
	…
	Fin

Lexique	SELECT * FROM tabPersBis WHERE valable = vrai Version avec itération sur le nombre d'enregistrements valides
tabPersBis (tableau de TypPersBis) : tableau des personnes (voir la description du type TypPersBis dans le cours). cptPers (entier, calculé) : nombre d'enregistrements valides. indParcours (entier, calculé) : indice de parcours du tableau. indVal (entier, calculé) : indice de l'élément valable courant.	Début … indVal ← 1, indParcours ← 1 TantQue indVal <= cptPers et indParcours <= taille(tabPersBis) Faire Si tabPersBis[indParcours]. valable Alors Afficher (tabPers[indParcours].persNom, tabPers[indParcours].prenom,…., tabPers[indParcours].persEmail) indVal ← indVal + 1 FinSi indParcours ← indParcours + 1 FinPour … Fin

Exercice 108

Écrire la partie d'algo permettant la suppression fictive d'un élément du tableau tabPersBis, de type TypPersBis, à partir de la saisie d'une information identifiant la personne de manière unique (information de votre choix).

Comme dans l'exercice précédent, considérer que tabPersBis contient déjà des enregistrements valides, et qu'il contient également des enregistrements qui ont été supprimés logiquement.

Lexique	Suppression pour de faux
phone (chaîne, saisie) : numéro de téléphone de l'enregistrement à supprimer. i (entier, calculé) : indice. tabPersBis (tableau de TypPersBis) : tableau des personnes (voir la description du type TypPersBis dans le cours). cptPers (entier, calculé) : nombre d'enregistrements valides.	Début … Afficher ("Saisir le numéro de téléphone de la personne à supprimer") Saisir (phone) i ← 1 TantQue tabPersBis[i].persTelephone <> phone et i < taille(tabPersBis) Faire i ← i + 1 FinTantQue Si phone = tabPersBis[i].persTelephone et tabPersBis[i].valable Alors tabPers[i].valable ← faux Sinon Afficher ("Ce numéro de téléphone est absent de votre carnet d'adresse.") FinSi … Fin

Exercice 109

Écrire la partie d'algo permettant l'ajout d'un enregistrement de type TypPersBis dans le tableau tabPersBis. On considère que le tableau n'est pas plein.

Laicsyk (je varie l'orthographe pour ne pas trop vous lasser).	Ajout en écrasant le premier enregistrement invalide
i (entier, calculé) : indice. tabPersBis (tableau de TypPersBis) : tableau des personnes (voir la description du type TypPersBis dans le cours). cptPers (entier, calculé) : nombre d'enregistrements valides.	Début … // recherche du premier enregistrement invalide TantQue tabPersBis[i].valable et i < taille(tabPersBis) Faire i ← i + 1 Fin Afficher ("Saisir les coordonnées de la nouvelle personne") Saisir (tabPersBis[i].persNom, …., tabPers[i].persEmail) tabPersBis[i].valable ← vrai cptPers ← cptPers +1 … Fin

Dans les lexiques des algos qui suivent, je ne redéclare pas **TabPers, CptPers**, et autres données qu'on se trimballe depuis le début de cette série d'exos, sauf quand j'ai assez de place dans le lexique.

Exercice 110

Trier le tableau tabPers sur la valeur du prénom, à l'aide de la méthode de tri de votre choix.

Lexique	Tri d'un tableau de structure
i (entier, calculé) : indice de parcours du tableau. permuter (procédure) : permute les valeurs des deux paramètres. fMini (fonction) : retourne l'indice du plus petit élément dans le tableau passé en paramètre, entre les deux bornes passées en paramètres.	Début Pour i de 1 à taille(tabPers) Faire permuter (tabPers[i], tabPers[fMini (tabPers, i, taille(tabPers)]) FinPour Fin

Lexique	Fonction fMini (leTableau : tableau de TypPers, mini, maxi : entier) : entier Fonction qui renvoie l'indice du plus petit élément du tableau spécifié, entre les bornes spécifiées.
indMin (entier, calculé) : indice du minimum courant. i (entier, calculé) : indice de parcours du tableau.	Début indMin ← mini Pour i de mini +1 à maxi Faire Si leTableau[i].persPrenom < leTableau[indMin].persPrenom Alors indMin ← i FinSi FinPour Renvoyer indMin Fin

Lexique	Procédure permuter (données modifiées x,y : TypPers)
temp (TypPers, calculé) : variable temporaire utilisée pour l'échange.	Début temp ← x x ← y y ← temp Fin

Exercice 111

Créer tabIndexNom, un tableau d'index qui contient les indices des éléments du tableau tabPers comme s'ils étaient triés sur le nom de la personne.

Lexique	Création d'un index de tri des éléments de TabPers sur le nom

j (entier, calculé) : indice d'itération.
tabIndexNom (tableau d'entier, calculé) : index.
i (entier, calculé) : indice d'itération.
tabPers (tableau de TypPers, saisi) : tableau des personnes du carnet d'adresse (voir déclaration du type TypPers dans l'exercice 101).
	Début
	…
	// TabPers est déjà rempli
	// Remplissage en vrac de tabIndexNom
	Pour j de 1 à taille(tabPers)
	Faire tabIndexNom[j] ← j
	FinPour
	// Tri à bulles
	Pour i de 1 à taille(tabPers) - 1
	Faire Pour j de i +1 à taille(tabPers)
	Faire Si tabPers[tabIndexNom[i]]> tabPers[tabIndexNom[j]]
	Alors permuter (tabIndexNom[i], tabIndexNom[j])
	FinSi
	FinPour
	FinPour
	…
	Fin

Exercice 112

Écrire l'algorithme qui affiche les noms des personnes du carnet d'adresse par ordre alphabétique (Utiliser l'index tabIndexNom pour ce parcours séquentiel).

Lexique	Parcours séquentiel d'un tableau suivant la valeur d'un index

tabIndexNom (tableau d'entier, calculé) : index.
i (entier, calculé) : indice d'itération.
tabPers (tableau de TypPers, saisi) : tableau des personnes du carnet d'adresse (voir déclaration du type TypPers dans l'exercice 101).
	Début
	…
	// tabPers est déjà rempli
	// tabIndexNom est créé et contient les indices des éléments de
	// tabPers triés sur le nom
	Pour i de 1 à CptPers
	Faire Afficher (tabPers[tabIndexNom[i]].persNom)
	FinPour
	…
	Fin

Remarque : le tableau **tabIndexNom** est parcouru séquentiellement une seule fois.

Exercice 113

Rajouter la personne saisie en insérant son indice au bon endroit dans tabIndexNom.

Lexique	Rajout d'une personne dans le carnet d'adresse

tabIndexNom (tableau d' entier, calculé) : index.
ind (entier, calculé) : indice d'itération.
LibereCase (procédure) : libère la case dont l'indice est passé en paramètre.
	Début
	…
	Afficher ("Saisir les coordonnées de la personne à rajouter")
	indIns ← taille(tabPers) + 1
	Saisir (tabPers[indIns].persNom, tabPers[indIns].persPrenom
	….) // Et ainsi de suite pour tous les champs
	// Recherche de l'indice d'insertion de indIns dans tabIndexNom
	ind ← 1
	TantQue tabPers[indIns] > tabPers[tabIndexNom[ind]]
	Faire ind ← ind + 1
	FinTantQue
	// ind a pour valeur l'indice d'insertion du nouvel enregistrement
	// dans tabPers
	LibereCase (tabIndexNom, ind, indIns - 1)
	tabIndexNom[ind] ← indIns
	….
	Fin

Exercice 114

Compléter le type TypPers, appelez-le TypPersTer, de manière à ce que l'index de tri sur le nom de la personne fasse partie du type.

```
TypPersTer :    Structure
                persPrenom (chaîne , saisie) : prénom de la personne.
                persTelephone (chaîne , saisie) : numéro de téléphone.
                persNomRue (chaîne , saisie) : numéro et nom de la rue.
                persCodePostal (entier, saisi) : code postal.
                persVille (chaîne , saisie) : ville.
                persEmail (chaîne , saisie) : adresse e-mail.
                index (entier, calculé) : index du tri sur le nom.
                FinStructure
```

Exercice 115

Soit un tableau tabPersTer, de type TypPersTer. Pour l'instant, le champ index de chaque enregistrement de tabPersTer est désespérément vide. Indexer de manière interne ce tableau sur les valeurs croissantes du nom.

Lexique	Indexation interne de tabPersTer

// Voir dans les exos
// précédents.
	Début
	..
	// On considère que tabPersTer contient déjà des données
	// Remplissage en vrac du champ **index**
	Pour i de 1 à taille(tabPersTer)
	Faire tabPersTer[i].index ← i
	FinPour
	// Tri
	Pour i de 1 à taille(tabPersTer) – 1
	Faire Pour j de i + 1 à taille(tabPersTer)
	Faire Si tabPersTer[i].persNom > tabPersTer[j].persNom
	Alors permuter (tabPersTer[i].index, tabPersTer[j].index)
	FinSi
	FinPour
	FinPour
	....
	Fin

Exercice 116

Écrire l'équivalent de la requête :

SELECT persNom FROM tabPersTer ORDER BY persNom, en utilisant, bien sûr l'index interne.

Lexique	Parcours séquentiel galère d'un tableau qui a un index de tri interne

// Même lexique que dans
// les exos précédents, avec
// en plus :
indParcours (entier, calculé) :
indice de parcours de
tabPersTer.
	Début
	Pour i de 1 à taille(tabPersTer)
	Faire indParcours ← 1
	// On n'a pas d'autre choix que de rechercher
	// séquentiellement l'enregistrement dont le nom a i pour
	// numéro d'ordre.
	TantQue tabPersTer[indParcours].index <> i et indParcours < taille(tabPersTer)
	Faire indParcours ← indParcours + 1
	FintantQue
	Afficher (tabPers[indParcours].persNom)
	FinPour
	...
	Fin

Exercice 117

Soit tabPers4, d'éléments de type TypPers4, une structure issue du type TypPersTer, à laquelle on a rajouté un champ s'appelant suivant, qui contient l'indice du suivant dans la liste, triée sur le nom des personnes du carnet d'adresses.

Écrire l'équivalent de la requête : SELECT persNom FROM tabPers4 ORDER BY persNom

Lexique	SELECT PersNom FROM TabPers4 ORDER BY PersNom

indCourant (entier, calculé) :
indice du plus petit courant.
tabPers4 (tableau de TypPers4) :
tableau des personnes.
TypPers4 (type) :
Structure
persNom (chaîne , saisie)
persPrenom (chaîne , saisie)
persTelephone (chaîne , saisie)
persNomRue (chaîne , saisie)
persCodePostal (entier, saisi)
persVille (chaîne , saisie)
persEmail (chaîne , saisie)
suivant (entier, calculé) : indice du suivant dans la liste triée sur le nom. Le dernier contient –1.
FinStructure
	Début
	// Recherche du premier nom de la liste avec la fonction
	// fMini qu'on a réécrite pour l'exo 110
	indCourant ← fMini (tabPers4, 1, taille(tabPers4))
	TantQue indCourant <> -1
	Afficher (tabPers4[indCourant].persNom)
	indCourant ← tabPers4[indCourant].suivant
	FinTantQue
	Fin

Exercice 118

On reprend tabPers4. Écrire l'algo qui réalise le chaînage des enregistrements suivant les valeurs croissantes du champ persNom.

Solution facile : on considère qu'on dispose d'un champ **index** et qu'il faille remplir le champ **suivant**.

Lexique	La structure contient un champ index déjà rempli et un champ suivant qu'il faut remplir

// Voir déclaration des
// autres données dans les
// lexiques précédents.
// Données supplémentaires
// de cet algo :
mini (entier, calculé) : élément courant à chaîner.
suiv (entier, calculé) : suivant de l'élément courant à chaîner.
	Début
	Pour i de 1 à taille(tabPers4) –1
	Faire mini ← 1
	suiv ← 1
	// Recherche du i^{ème} minimum
	TantQue tabPers4[mini].index <> i
	Faire mini ← mini + 1
	FinTantQue
	// Recherche du i^{ème} suivant
	TantQue tabPers4[suiv].index <> i + 1
	Faire suiv ← suiv + 1
	FinTantQue
	TabPers4[mini].suivant ← suiv
	FinPour
	// On traite le suivant du dernier élément à part
	tabPers4[suiv].suivant ← -1
	Fin

Solution plus complexe : on ne dispose pas du champ **index**.

Lexique	Soluce complexe On ne dispose pas du champ index dans la structure

// Données
// spécifiques à cet
// algo :
indPetit (entier, calculé) :
élément courant à chaîner.
indSuiv (entier, calculé) : suivant
de l'élément courant à chaîner.
cptElem (entier, calculé) :
compteur des éléments chaînés.
	// Recherche du premier plus petit
	indPetit ← fMini (tabPers4, 1, taille(tabPers4))
	// Initialisation du compteur des éléments chaînés
	cptElem ← 0
	Répéter
	// On cherche le suivant (il doit être le plus petit mais
	// être supérieur au plus petit précédemment trouvé).
	indSuiv ← 1
	Pour i de 1 à taille(tabPers4)
	Faire Si tabPers4[i].persNom < tabPers4[indSuiv].persNom
	Et tabPers4[i].persNom >= tabPers4[indPetit].persNom
	Et i <> indPetit
	Alors indSuiv ← i
	FinSi
	FinPour
	// On range indSuiv dans le champ suivant du plus petit.
	tabPers4[indPetit].suivant ← indSuiv
	// Le suivant devient ensuite le plus petit.
	indPetit ← indSuiv
	// On incrémente le nombre d'éléments indexés
	cptElem ← cptElem + 1
	// Jusqu'à ce qu'on ait indexé taille(tabPers4) – 1 éléments
	Jusqu'à cptElem = taille(tabPers4) - 1
	// On indexe le dernier élément
	tabPers4[indSuiv].suivant ← -1
	Fin

Je veux vous expliquer en détail le sens de la condition

Si tabPers4[i].persNom < tabPers4[indSuiv].persNom

Et tabPers4[i].persNom >= tabPers4[indPetit].persNom

Et i <> indPetit

Si tabPers4[i].persNom < tabPers4[indSuiv].persNom signifie "si le nom de l'élément courant est inférieur au nom du suivant courant "…

Et tabPers4[i].persNom >= tabPers4[indPetit].persNom signifie "et que le nom de cet élément courant est supérieur au nom du plus petit courant" (ça, c'est dans le cas où on ait deux fois le même nom, pour que les noms identiques figurant à des indices différents puissent être chaînés entre eux) …

Et i <> indPetit signifie "et que cet élément courant n'est pas le plus petit courant"…(sinon on ne ferait que chaîner le même élément sur lui-même)…

Alors, cet élément peut être le suivant du plus petit courant.

Séquence 8

Les fichiers

Partie 2

Les modes d'accès aux données d'un fichier

Exercice 119

Soit un fichier fichPers contenant des enregistrements de type TypPers (pour la description de ce type, voir séquence 7).

Écrire l'algorithme qui lit séquentiellement et affiche tous les enregistrements du fichier un à un.

Le corrigé est dans le cours.

Exercice 120

Écrire l'algorithme permettant de charger le contenu du fichier fichPers dans un tableau.

Lexique	Chargement de fichPers dans un tableau

// Pour le type TypPers, voir le
// lexique de l'algo précédent. Je
// ne le remets pas car la
// déclaration du type prend
// beaucoup de place.
fichPers (fichier de TypPers) : fichier des personnes.
tabPers (tableau de TypPers) : tableau des personnes.
Taille (fonction) : retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.
	Début
	Ouvrir (fichPers, Lecture)
	Accès (fichPers, Séquentiel)
	Lire (fichPers, tabPers[taille(tabPers)])
	// On range directement l'enregistrement lu à la première
	// case libre du tableau
	TantQue Non finDeFichier (fichPers)
	Faire Lire (fichPers, tabPers[taille(tabPers)])
	FinTantQue
	Fermer (fichPers)
	Fin

Exercice 121

Écrire l'algorithme qui calcule le nombre de personnes ayant une adresse e-mail dans le fichier fichPers qui contient les données du carnet d'adresses.

Lexique	SELECT COUNT * FROM fichPers WHERE persEmail <> "" Version qui attaque le fichier fichPers
fichPers (fichier de TypPers) : fichier des personnes. nbMail (entier, calculé) : compteur. personne (TypPers, calculé) : personne courante, lue dans fichPers.	Début Ouvrir (fichPers, Lecture) Accès (fichPers, Séquentiel) nbMail ← 0 TantQue Non finDeFichier (fichPers) Faire Lire (fichPers, personne) Si personne.persEmail <> " " Alors nbMail ← nbMail + 1 FinSi FinTantQue Afficher (nbMail) Fermer (fichPers) Fin

Lexique	SELECT COUNT * FROM fichPers WHERE persEmail <> "" Version qui attaque le tableau tabPers
nbMail (entier, calculé) : compteur. tabPers (tableau de TypPers) : tableau des personnes. Taille (fonction) : retourne le nombre effectif d'éléments contenus dans le tableau passé en paramètre.	Début // tabPers contient déjà les enregistrements chargés depuis fichPers … nbMail ← 0 Pour i de 0 à taille(tabPers) - 1 Faire Si tabPers[i].persEmail <> " " Alors nbMail ← nbMail + 1 FinSi FinPour Afficher (nbMail) … Fin

Exercice 122

Écrire l'algorithme qui rajoute dans le fichier fichPers l'enregistrement dont les valeurs sont saisies par l'utilisateur.

Lexique	Ajout de la personne saisie
fichPers (fichier de TypPers) : fichier des personnes. personne (TypPers, saisie) : personne à rajouter dans fichPers.	Début Ouvrir (fichPers, ajout) // L'ouverture en ajout positionne le pointeur de fichier à la fin du fichier Accès (fichPers, séquentiel) Saisir (personne.persNom, personne.persPrenom,..., personne.persEmail) Ecrire (fichPers, personne) Fermer (fichPers) Fin

Exercice 123

Écrire l'algo de création et de remplissage du fichier fichPers, les données étant saisies au clavier.

Dans tous les langages de programmation, il existe des instructions qui permettent de vérifier si un fichier existe déjà ou non. Dans Windev, par exemple, il y a une instruction qui permet de créer à vide le fichier s'il n'existe pas déjà.

Pour simuler ce comportement, en algo, on va utiliser une fonction de mon invention, la fonction **existe**, qui renvoie **vrai** si le fichier passé en paramètre existe, et **faux** sinon.

Si le fichier existe, alors il faut l'ouvrir en **ajout**, sinon, il faut l'ouvrir en **écriture** (ce qui a pour effet, je vous le rappelle, de créer le fichier à vide).

On pourrait aussi se contenter d'ouvrir le fichier en ajout, qu'il existe ou non, en supposant que lorsqu'un fichier n'existe pas, l'ouverture en ajout a pour effet de le créer.

Le remplissage du fichier se fait à l'aide d'un **TantQue**, et on arrête la saisie lorsque l'utilisateur saisit la chaîne vide pour le nom de la personne.

Lexique	Création du fichier et ajout des personnes saisies

fichPers (fichier de TypPers) : fichier des personnes.
personne (TypPers, saisie) : personne à rajouter dans fichPers.
existe (fonction) résultat booléen : retourne vrai si le fichier passé en paramètre existe, faux sinon.
	Début
	Si existe (fichPers)
	Alors Ouvrir (fichPers, ajout)
	Sinon Ouvrir (fichPers, écriture)
	FinSi
	// On pourrait aussi se contenter d'écrire Ouvrir (fichPers, ajout) au
	// lieu des 4 lignes ci-dessus
	Accès (fichPers, séquentiel)
	Afficher ("Saisissez le nom :")
	Saisir (personne.persNom)
	TantQue personne.persNom <> ""
	Saisir (personne.persPrenom,..., personne.persEmail)
	Ecrire (fichPers, personne)
	Afficher ("Saisissez le nom :")
	Saisir (personne.persNom)
	FinTantQue
	Fermer (fichPers)
	Fin

Exercice 124

Écrire l'algorithme qui rajoute dans le fichier fichPers les enregistrements saisis par l'utilisateur et rangés dans le tableau tabPers.

Le tableau **tabPers** contient les enregistrements à rajouter dans **fichPers**.

Lexique	Ajout dans fichPers des enregistrements de tabPers

fichPers (fichier de TypPers) : fichier des personnes.
existe (fonction) résultat booléen : retourne vrai si le fichier passé en paramètre existe, faux sinon.
taille (fonction) résultat entier : retourne le nombre d'éléments du tableau passé en paramètre.
tabPers (tableau de TypPers) : tableau des personnes.
i (entier, calculé) : compteur.
	Début
	Si existe (fichPers)
	Alors Ouvrir (fichPers, ajout)
	Sinon Ouvrir (fichPers, écriture)
	FinSi
	// Comme dans l'exercice précédent, on
	// pourrait aussi se contenter d'écrire Ouvrir (fichPers, ajout) au
	// lieu des 4 lignes ci-dessus
	Accès (fichPers, séquentiel)
	pour i de 1 à taille(tabPers)
	Faire Ecrire (fichPers, tabPers[i])
	FinTantQue
	Fermer (fichPers)
	Fin

Exercice 125

Soit un tableau tabMots (utilisé pour le jeu du pendu dans la séquence 5). Écrire la partie d'algo correspondant au traitement lancé au moment où l'utilisateur quitte le jeu : le contenu du tableau est recopié dans un fichier fichMots, seulement si ce dernier a été modifié.

Bon, cet énoncé suppose qu'on a, au début du lancement de l'application, chargé le contenu de **fichMots** dans le tableau **tabMots** et que l'utilisateur a eu la possibilité de modifier le contenu de **tabMots** (ajout, suppression, modification de mots).

La question à se poser est : comment savoir si le tableau a été modifié.

Utiliser un booléen, initialisé à **faux** et passant à **vrai** à la première modification, quelle qu'elle soit. Dans ce cas, si il y a eu modif, on n'a pas le moyen de savoir exactement quelle(s) modif(s) a ou ont été effectuée(s). Il faut donc recopier **tabMots** dans **fichMots** dans sa totalité. Cette solution n'est pas géniale mais ce n'est pas non plus trop grave car **tabMots** est petit, donc, **fichMots** l'est aussi, donc, le traitement ne sera pas trop long.

Il existe des solutions moins sommaires, mais je pense que vous serez d'accord pour qu'on se contente de celle-là.

Conséquence de cette solution : on recrée le fichier **fichMots** à vide à chaque fois que l'utilisateur quitte le jeu et que le booléen indicateur de modification vaut vrai.

Si cet indicateur de modification vaut faux, on ne fait rien.

Si on reprend les exos de la séquence 5, **tabMots** est indicé à partir de 0.

Lexique	Sauvegarde dans FichMots des modifications apportées à TabMots
fichMots (fichier de chaîne) : fichier des mots. tabMots (tableau[0..100]de chaîne) : tableau des mots. i (entier, calculé) : compteur. taille (fonction) résultat entier : retourne le nombre d'éléments du tableau passé en paramètre.	Début Si modif Alors Ouvrir (FichMots, écriture) Accès (FichMots, séquentiel) Pour i de 0 à taille(tabMots) - 1 Faire Ecrire (fichMots, tabMots[i]) FinPour Fermer (FichMots) FinSi Fin

Exercice 126

Écrire l'algo permettant la modification des coordonnées de la personne dont le nom a été saisi.

Là, on est reparti avec le fichier **fichPers**.

Quelle démarche adopter pour écrire cet algo ? D'abord, il faut vérifier que le nom saisi figure bien dans le fichier en faisant une recherche séquentielle. Si la personne existe, on fait la modif, sinon, on affiche un message.

Lexique	Modification des coordonnées de la personne dont le nom a été saisi
fichPers (fichier de TypPers) : fichier des personnes. nom (chaîne, saisi) : nom de la personne à modifier.	Début Ouvrir (fichPers, lecture/écriture) Accès (fichPers, séquentiel) Saisir (nom) Lire (fichPers, personne) TantQue personne.persNom <> nom et Non FinDeFichier (fichPers) Faire Lire (fichPers, personne) FinTantQue Si personne.persNom = nom Alors Saisir (personne.persNom, personne.persPrénom, .., personne.persEmail) Réécrire (fichPers, personne) Sinon Afficher ("Cette personne ne figure pas dans votre carnet d'adresse.") FinSi Fermer (fichPers) Fin

Cet exo est très réducteur dans le sens où il suppose qu'il n'y ait qu'une personne portant le nom saisi dans le fichier. Pourtant, plusieurs personnes différentes peuvent porter le même nom. Avec notre algo, l'utilisateur ne pourra jamais modifier que la première personne portant ce nom.

Dans le TP de cette séquence 8, je vous demande de modifier cet algo de manière à permettre à l'utilisateur de choisir la personne à modifier dans une liste que vous aurez constituée, contenant toutes les personnes de **fichPers** portant le nom saisi. Avec Windev, ça se fait « fingers in the nose », on ne fera donc pas l'algo correspondant.

Exercice 127

Écrire un algorithme qui rajoute le mot FRANCE dans la ville de chaque enregistrement possédant un code postal (le fait qu'un enregistrement possède un code postal est la preuve que la personne vit en France).

Une personne possède un code postal si son champ code postal est différent de la chaîne vide.

Lexique	Modification du champ ville des personnes vivant en France
fichPers (fichier de TypPers) : fichier des personnes. personne (TypPers, calculée) : personne courante, lue dans fichPers.	Début Ouvrir (fichPers, lecture/écriture) Accès (fichPers, séquentiel) TantQue Non FinDeFichier (fichPers) Faire Lire (fichPers, personne) Si personne.persCodePostal <> " " Alors personne.persVille ← personne.persVille + " FRANCE" Réécrire (fichPers, personne) FinSi FinTantQue Fermer (fichPers) Fin

Exercice 128

Écrire un algo qui affiche le mot dont le numéro d'ordre a été saisi, dans un fichier de chaînes s'appelant fichMots.

Il faut : positionner le pointeur de fichier à la position saisie, lire l'enregistrement et l'afficher. Mais attention, il faut penser à vérifier que la position saisie ne dépasse pas le nombre d'enregistrements du fichier.

Lexique	Algo exo 128 - Afficher un mot
fichPers (fichier de TypPers) : fichier des personnes. personne (TypPers, calculée) : personne courante, lue dans fichPers.	Début Afficher ("Saisissez la position du mot à afficher.") Saisir (pos) Si pos <= tailleEnOctets (FichMots) / tailleEnOctets (chaîne) Alors // On n'ouvre le fichier que si la position saisie est valide Ouvrir (fichMots, lecture) Accès (fichMots, direct) Position (fichMots, pos) Lire (fichMots, mot) Afficher (mot) Fermer (fichMots) Sinon Afficher ("La position saisie est invalide.") FinSi Fin

Exercice 129

Écrire le sous-programme RetournePos qui renvoie le numéro d'enregistrement du mot passé en paramètre, dans le fichier passé en paramètre.

Si on regarde bien l'exemple d'appel de la fonction **RetournePos** que je vous ai fourni dans l'énoncé du cours, on peut constater que son comportement ressemble étrangement à la fonction **RechSeq** que l'on a écrite et utilisée de nombreuses fois sur des tableaux.

Lexique	Algo exo 129 Fonction RetournePos (leFich : fichier de ?, elem : ?) : entier
enrLu (?, calculé) : enregistrement courant, lu dans leFich.	Début Ouvrir (leFich, lecture) Accès (leFich, séquentiel) TantQue elem <> enrLu et Non FinDeFichier (leFich) Faire Lire (leFich, enrLu) FinTantQue Si elem = enrLu Alors Renvoyer DonnePosition (leFich) - 1 Sinon Renvoyer –1 FinSi Fermer (leFich) Fin

La position à renvoyer est **DonnePosition (leFich) – 1** car si c'est l'enregistrement venant d'être lu qui est égal à **elem**, alors le pointeur de fichier pointe sur le début de l'enregistrement suivant, puisque **enrLu** vient d'être lu.

Exercice 130

Écrire un algo qui choisit aléatoirement un mot dans le fichier fichMots et range ce mot dans une variable s'appelant motATrouver (vous voyez où je veux en venir là ?).

La position doit être aléatoirement choisie dans une fourchette de positions valides à l'aide de la fonction **hasard**.

La borne minimale pour le choix aléatoire sera 1 (plus petit indice dans un fichier), et la borne maximale vaut le nombre d'enregistrements de **fichMots.**

Lexique	Algo exo 130 Choix aléatoire d'un mot dans fichMots
fichMots (fichier de chaîne) : fichier des mots. motATrouver (chaîne, calculé) : mot choisi aléatoirement dans fichMots.	Début Ouvrir (fichMots, lecture) Accès (fichMots, direct) Position (fichMots, hasard (1, tailleEnOctets (FichMots) / tailleEnOctets (chaîne)) Lire (fichMots, motATrouver) Fin

Exercice 131

Soit le fichier fichPers, de type TypPers, sur lequel existe l'index indexNom, un index sur le nom. Écrire l'algo de parcours séquentiel du fichier fichPers suivant les valeurs croissantes du nom des personnes.

Lexique	SELECT PersNom FROM FichPers ORDER BY PersNom
fichPers (fichier de TypPers) : fichier des personnes. personne (TypPers, calculée) : personne courante, lue dans fichPers.	Début Ouvrir (fichPers, lecture/écriture) Accès (fichPers, indexé) TantQue Non FinDeFichier (fichPers) Faire Lire (fichPers, indexNom, personne) Afficher (personne.persNom) FinTantQue Fermer (fichPers) Fin

Partie 3

Diverses manipulations sur les fichiers

Exercice 132

Créer un type TypPersBis, issu du type TypPers et contenant un champ de type booléen permettant la suppression logique. Recopiez le contenu du fichier fichPers dans un fichier fichPersBis, de type TypPersBis, en mettant à vrai le booléen de tous les enregistrements.

Retournez voir la description du type **TypPersBis** dans la séquence 7, partie 2, paragraphe 1. Le booléen qu'on a rajouté à la structure s'appelle **valable**.

Lexique	Recopiage (ou recopiement ?) de FichPers dans FichPersBis

fichPers (fichier de TypPers) : fichier des personnes.
fichPersBis (fichier de TypPersBis) : fichier des personnes avec un champ indiquant si l'enregistrement est valide ou non.
personne (TypPers, calculée) : personne courante, lue dans fichPers.
personneBis (TypPersBis, calculée) : personne courante, lue dans FichPersBis.
	Début
	Ouvrir (fichPers, lecture)
	Ouvrir (fichPersBis, écriture)
	Accès (fichPers, séquentiel)
	Accès (fichPersBis, séquentiel)
	// On donne sa valeur à ce champ une fois pour toutes.
	personneBis.valable ← vrai
	TantQue Non FinDeFichier (fichPers)
	Faire Lire (fichPers, personne)
	personneBis.persNom ← personne.persNom
	personneBis.persPrenom ← personne.persPrenom
	// et ainsi de suite jusqu'à
	personneBis.persEmail ← personne.persEmail
	Ecrire (fichPersBis, personneBis)
	FinTantQue
	Fermer (fichPers)
	Fermer (fichPersBis)
	Fin

Exercice 133

Supprimer logiquement du fichier fichPersBis tous les enregistrements concernant les personnes habitant à Trifouillis Aux Oies.

Lexique	Algo exo 133 Suppression logique
fichPersBis (fichier de TypPersBis) : fichier des personnes. persBis (TypPersBis, calculée) : personne courante.	Début Ouvrir (fichPersBis, lecture/écriture) Accès (fichPersBis, séquentiel) TantQue Non FinDeFichier (fichPersBis) Faire Lire (fichPersBis, persBis) Si persBis.persVille = "Trifouillis Aux Oies" Alors persBis.valable ← faux Réécrire (fichPersBis, persBis) FinTantQue Fermer (fichPersBis) Fin

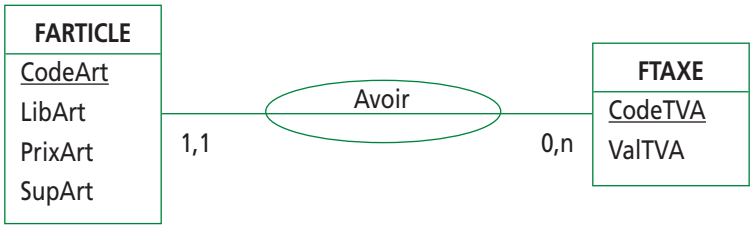
Exercice 134

Écrire l'algo qui « épure » le fichier fichPersBis, c'est-à-dire qui supprime physiquement tous les enregistrements qui ont été supprimés logiquement.

Lexique	Épuration
fichPersBis (fichier de TypPersBis) : fichier des personnes. tempo (fichier de TypPersBis) : fichier des personnes qui n'ont pas été supprimées par l'utilisateur. persBis (TypPersBis, calculée) : personne courante.	Début Ouvrir (fichPersBis, lecture), Ouvrir (tempo, écriture) Accès (fichPersBis, séquentiel), Accès (tempo, séquentiel) TantQue Non FinDeFichier (fichPersBis) Faire Lire (fichPersBis, persBis) Si persBis.valable Alors Ecrire (tempo, persBis) FinSi FinTantQue Fermer (fichPersBis), Fermer (tempo) Renommer (tempo, fichPersBis) Fin

Exercice 135

Soit le SCD (Schéma Conceptuel des Données) suivant :



On suppose que les fichiers fArticles et fTaxes correspondant au SCD ci-dessus existent.

Écrire l'algo qui affiche le nom et le prix TTC de tous les articles.

Là, le plus malin, c'est de charger les codes et les valeurs de TVA en mémoire vive, par exemple dans des variables, pour ne pas avoir à lire la valeur de la bonne TVA dans le fichier **fTaxes** pour chaque article.

Cela peut paraître inutile de créer un fichier ou une table juste pour stocker deux valeurs de TVA, mais là, dans notre cas, c'est comme ça, il ne nous reste qu'à faire avec cette situation.

Lexique	SELECT libArt, prixArt * valTVA FROM fArticles, fTaxes WHERE fArticles.codeTVA = fTaxes.codeTVA
fTaxe (fichier de Ttaxe) : fichiers contenant les codes et valeurs des taux de TVA. Ttaxe (type) : Structure codeTVA (entier) : code du taux de TVA. valTVA (réel) : valeur du taux. FinStructure fArticle (fichier de Tarticle) : fichiers contenant les articles. Tarticle (type) : Structure codeArt (entier) : code de l'article. libArt (chaîne) : nom de l'article. prixArt (réel) : prix de l'article. supArt (chaîne) : renseignements supplémentaires sur l'article. codeTVA (entier) : code du taux de TVA (clé étrangère issue du fichier fTaxes). FinStructure TVA1, TVA2 (Ttaxe, calculé) : variables contenant les valeurs des taux de TVA, lues dans fTaxes. val (réel) : valeur du taux de TVA courant.	Début Ouvrir (fTaxes, lecture), Ouvrir (fArticles, lecture) Accès (fTaxes, séquentiel), Accès (fArticles, séquentiel) Lire (fTaxes, TVA1), Lire (fTaxes, TVA2) TantQue Non FinDeFichier (fArticles) Faire Lire (fArticles, article) Si article.codeTVA = TVA1.codeTVA Alors val ← TVA1.valTVA Sinon val ← TVA2.valTVA FinSi Afficher (article.libArt, article.prixArt * (1 + val)) FinTantQue Fermer (fArticles), Fermer (fTaxes) Fin

Exercice 136

Créer, à partir du fichier fichPersBis, deux fichiers : un contenant les enregistrements valides, l'autre contenant les enregistrements que l'utilisateur a supprimés logiquement.

Là, on va repartir de l'exo 134 et le modifier de manière à récupérer les enregistrements logiquement supprimés dans un fichier. Fastoche!

Lexique	Éclatement (ou éclatage ?)
fichPersBis (fichier de TypPersBis) : fichier des personnes. fichPersBon (fichier de TypPersBis) : fichier des personnes qui n'ont pas été supprimées par l'utilisateur. fichPersMauv (fichier de TypPersBis) : fichier des personnes qui ont été supprimées par l'utilisateur. persBis (TypPersBis, calculée) : personne courante.	Début Ouvrir (fichPersBis, lecture), Ouvrir (fichPersBon, écriture) Ouvrir (fichPersMauv, écriture) Accès (fichPersBis, séquentiel), Accès (fichPersBon, séquentiel) Accès (fichPersMauv, séquentiel) TantQue Non FinDeFichier (fichPersBis) Faire Lire (fichPersBis, persBis) Si persBis.valable Alors Ecrire (fichPersBon, persBis) Sinon Ecrire (fichPersMauv, persBis) FinSi FinTantQue Fermer (fichPersBis), Fermer (fichPersBon), Fermer (fichPersMauv) Fin

Exercice 137

Soient deux fichiers fichPers1 et fichPers2, tous deux physiquement triés sur le nom de la personne. Écrire l'algorithme qui fusionne ces deux fichiers en un seul fichier fichPers.

Moi, pour écrire le corrigé, je suis repartie de l'exo 80, séquence 5 (fusion de deux tableaux).

Lexique	Fusion de deux fichiers
fichPers (fichier de TypPersBis) : fichier des personnes, résultat de la fusion. fichPers1, fichPers2 (fichier de TypPersBis) : fichiers à fusionner. personne1, personne2 (TypPersBis, calculée) : personnes courantes.	Début Ouvrir (fichPers1, lecture), Ouvrir (fichPers2, lecture) Ouvrir (fichPers, écriture) Accès (fichPers1, séquentiel), Accès (fichPers2, séquentiel) Accès (fichPers, séquentiel) TantQue Non FinDeFichier (fichPers1) et Non FinDeFichier (fichPers2) Faire Lire (fichPers1, personne1) Lire (fichPers2, personne2) Si personne1.persNom > personne2.persNom Alors Ecrire (fichPers, personne1) Lire (fichPers, personne1) Sinon Ecrire (fichPers, personne2) Lire (FichPers, personne2) FinSi FinTantQue TantQue Non FinDeFichier (fichPers1) Faire Lire (fichPers1, personne1) Ecrire (fichPers, personne1) FinTantQue TantQue Non FinDeFichier (fichPers2) Faire Lire (fichPers2, personne1) Ecrire (fichPers, personne1) FinTantQue Fermer (fichPers), Fermer (fichPers1), Fermer (fichPers2) Fin

Exercice 138

Soient deux fichiers de type TypPers, non triés. Écrire l'algo qui réalise le tri et la fusion en un seul de ces deux fichiers, en utilisant un tableau.

En réfléchissant au corrigé de cet exo, je me demande ce qui m'a pris de vous donner un exo pareil surtout en dernier, tout dernier exo.

Alors on ne va pas trop se casser, vous en avez marre ? Ça tombe bien, moi aussi.

Pour écrire cet algo, on va tout bêtement recopier les deux fichiers en vrac dans le tableau, puis trier le tableau avec un des tris de tableau de structure qu'on a fait à la séquence 7.

En fait, il est bête comme tout cet exo, c'est un puzzle.

Lexique	Puzzle

fichPers, fichPers1, fichPers2 (fichier de TypPers) : fichiers des personnes.
tabPers (tableau de TypPers) : tableau des personnes.
taille (fonction) résultat entier : retourne le nombre effectif d'éléments du tableau passé en paramètre.
i (entier, calculé) : compteur.
	Début
	Ouvrir (fichPers1, Lecture), Ouvrir (fichPers2, Lecture)
	Ouvrir (fichPers, Ecriture)
	Accès (fichPers, Séquentiel), Accès (fichPers1, Séquentiel),
	Accès (fichPers2, Séquentiel)
	// Chargement dans tabPers
	TantQue Non FinDeFichier (fichPers1)
	Faire Lire (fichPers1, tabPers[taille(tabPers) + 1])
	FinTantQue
	TantQue Non FinDeFichier (FichPers2)
	Faire Lire (fichPers2, tabPers[taille(tabPers) + 1])
	FinTantQue
	// Tri de tabPers
	Pour i de 1 à taille(tabPers)
	Faire
	permuter (tabPers[i], tabPers[fMini (tabPers, i, taille(tabPers))])
	FinPour
	// Recopie de tabPers dans fichPers
	Pour i de 1 à taille(tabPers)
	Faire Ecrire (fichPers, tabPers[i])
	FinPour
	Fermer (fichPers), Fermer (fichPers1), Fermer (fichPers2)
	Fin

Voilà, dernier corrigé d'exo.
So long…