Fiche N° 6 : Fonctions

Dans ce TD vous devez décrire les fonctions et Main comme pour les algorithmes.
Vous les implanterez en C# en TP

Exercice 1 – Premiers pas

1. Écrire la fonction « Carre » qui retourne le carré d'un entier passé en paramètre. Écrire un Main qui demande à l'utilisateur un entier N et affiche son carré.
2. Écrire la fonction « Factorielle » qui retourne (calcule) la factorielle d'un nombre donnée en paramètre. Écrire un Main qui demande à l'utilisateur un entier N et affiche sa factorielle.
3. Écrire la fonction «Div10 » qui retourne la résultat de la division par 10 d'un nombre donné en paramètre. Écrire un Main qui demande à l'utilisateur un entier N et affiche le résultat de sa division par 10.
4. Écrire un Main qui demande à l'utilisateur un entier N puis une opération à effectuer sur cet entier (1 : Carré, 2 : Factorielle, 3 : Div10). Le Main doit afficher le résultat.
5. Écrire le programme complet
6. Écrire une fonction qui demande à l'utilisateur de saisir 10 entiers et retourne le minimum de ces dix valeurs. Écrire un Main qui appelle cette fonction et affiche ce minimum
7. Écrire la procédure « TableMult » qui affiche la table de multiplication d'un nombre donné. Écrire un Main qui demande à l'utilisateur un entier N puis demande l'affichage de la table de multiplication de N

Exercice 2 – Inverse d'un réel

1. Écrire la fonction « Inverse » qui calcule l'inverse d'un réel X donné. Il y a une précondition (c'est quelque chose à vérifier avant d'exécuter la fonction).
2. Écrire le Main qui saisit un réel, appelle la fonction et affiche le résultat
3. On ne pose plus de précondition. Comment peut-on gérer le cas où X = 0 ?
4. Écrire la fonction correspondante « InverseF_SansPré » et le Main qui saisit un réel et affiche son inverse.

Exercice 3 – Fonctions à partir d'algorithmes existants

Dans cet exercice vous devez reprendre des algorithmes vus dans les exercices 1, 4 et 6 du TD1 et les transformer en fonction.

Pour cela, vous devez remplacer les « Affiche » par un retour de fonction et les Entrées par des paramètres de la fonction.

Par exemple :

- La question 1 du TD 1 : écrire un algorithme qui demande à l'utilisateur d'entrer le prix HT d'un produit quelconque et qui affiche son prix TTC.
- Devient : Écrire une fonction qui prend le prix HT d'un produit en entrée et retourne son prix TTC. Écrire un Main qui demande à l'utilisateur d'entrer le prix HT d'un produit quelconque et qui affiche son prix TTC

Dans toute la suite des exercices, on supposera que les tableaux utilisés pour faire les tests sont donnés « en dur » dans l'algorithme principal (le Main). Leur taille est passée en paramètre des fonctions.

Exercice 4 – Fonctions sur des tableaux

1. Écrire une fonction « TabTrie » qui teste si tableau d'entiers passé en paramètre est trié. Elle doit retourner Vrai si le tableau est trié, faux sinon.
2. Écrire une fonction « RechercheTab » qui recherche si un entier passé en paramètre est présent dans le tableau d'entiers passé en paramètre. Elle doit retourner le nombre de fois qu'il apparait. Elle retournera 0 si l'entier n'est pas présent.
3. Écrire une fonction « MultiRecherche » qui recherche pour tous les entiers compris entre 0 et un entier N donné en paramètre s'ils apparaissent dans un tableau donné en paramètre. Pour chacun de ces entiers (donc compris entre 0 et N), la fonction doit afficher s'il est présent ou non : si l'entier est présent, elle doit afficher nombre de fois qu'il apparait sinon 0.

Exercice 4 – Fonctions sur des chaînes de caractères de taille connues

Dans la suite on représente souvent une chaine de caractères par un tableau de caractères. On suppose que l'on connait la taille L de la chaîne. C'est le cas des strings que nous utiliserons en C#.

// Dans cet exercice, pensez à utiliser sur la documentation partielle sur moodle (document « string.txt »)

1. Écrire une fonction « Palindrome » qui teste si un string S est un palindrome. S et sa taille sont des paramètres de la fonction. Elle doit retourner Vrai ou Faux.
2. Écrire une fonction « CompareChaine » qui teste si deux chaînes de caractères T1 et T2 de longueur L sont identiques. T1, T2 et L sont des paramètres de la fonction. Elle doit retourner Vrai ou Faux.
3. Idem mais T1 est de taille L1 et T2 de taille L2 (L1 et L2 peuvent être différents)

Exercice 5 – Tableau de notes.

Soit pTab un tableau de N nombres décimaux (float ou double) contenant les notes d'un groupe d'étudiants à un examen.

1. Ecrire une fonction nbBonnesNotes(pTab,pSeuil) qui renvoie le nombre de notes situées au-dessus d'un seuil entré en paramètre.
2. Ecrire une fonction ecart(pTab, pSeuil) qui affiche un nouveau tableau avec les écarts de chacune des notes avec le seuil entré en paramètre.

Exercices supplémentaires

Exercice 6 – Fonctions sur des chaînes de caractères de taille inconnues

Dans la suite on représente une chaine de caractères par un tableau de 101 caractères. On marque la longueur de la chaine par `\0`. Ainsi la chaîne de caractères « Hello » est représentée par un tableau de 101 caractères dont le 5 premiers sont 'H', 'e', 'l', 'l' ,'o' et le 6ieme `\0`.
Elle sera donc déclarée par S : tableau de 6 caractères initialisée à {'H', 'e', 'l', 'l' ,'o' ,`\0`}.

Cet exercice va vous montrer, à titre informatif, comme cela fonctionne en C.

1. Écrire une fonction « ChaineValideZero » qui teste si un tableau T de 101 caractères contient une chaîne de caractères. Elle doit retourner Vrai ou Faux.
2. Écrire une fonction « LongueurZero » qui donne la longueur d'une chaîne de caractères d'un tableau. Écrire un exemple de mail correspondant.

3. Écrire une fonction « CompareChaineZero » qui teste si deux chaînes de caractères T1 et T2 sont identique. Elle doit retourner Vrai ou Faux.

Exercice 7 - Les paliers en plongée

Afin d'éviter des accidents de décompression, les plongeurs en bouteilles d'air comprimé sont tenus de respecter des paliers d'attente lors de la remontée à chacune des profondeurs : 15, 12, 9, 6 et 3 mètres. Le temps d'attente à chaque palier est fonction de la profondeur et du temps que le plongeur a passé à la profondeur la plus basse. Les plongeurs se basent sur les tables NM90 dont voici celle utilisée pour une profondeur de 60 mètres :

		0	1	2	3	4	5	6	7	8	9	10
	Durées →	5	10	15	20	25	30	35	40	45	50	55
paliers ↓	**attentes**	0	1	2	3	4	5	6	7	8	9	10
0	3	2	6	19	32	41	48	54	62	69	78	88
1	6	0	2	4	8	15	22	28	30	35	37	40
2	9	0	0	1	3	5	8	11	17	19	22	24
3	12	0	0	0	0	0	1	4	6	9	13	15
4	15	0	0	0	0	0	0	0	0	1	2	5

Par exemple si un plongeur reste 25 minutes à 60 m, il doit respecter un palier de 5 minutes à 9 mètres, de 15 minutes à 6 mètres et de 41 minutes à 3 mètres (ce qui fait 61 minutes cumulées).
Pour simplifier, dans la suite on ne considérera que la profondeur de 60 mètres :

Il est à noter que, pour des raisons de sécurité, il est interdit de plonger plus longtemps que 55 minutes à 60 mètres (temps indiqué à la dernière case du tableau **durees**).

Dans la suite le tableau **paliers** sera créé en local dans la procédure, sa taille et son contenu sont toujours fixes. Le tableau **attente**s possède le même nombre de colonne que la taille du tableau **durees** et le même nombre de lignes que la taille du tableau **paliers**.

1. Écrire l'algorithme de la procédure afficherPalier() qui reçoit en paramètre :
   - la durée prévue de la plongée en minutes,
   - le tableau **durees** et sa **taille**
   - le tableau **attentes**

La procédure doit indiquer au plongeur si la plongée est interdite sinon elle afficher pour chaque palier le temps d'attente à respecter. Par exemple après avoir saisi 38 minutes, il faut appliquer les paliers de de la durée 40 minutes (temps supérieur le plus proche) et afficher :

```
6 minutes à 12 mètres
17 minutes à 9 mètres
30 minutes à 6 mètres
62 minutes à 3 mètres
Soit une durée cumulée de 115 minutes (01:55)
```

A noter que les temps d'attente à 0 ne sont pas affichés et que le temps de la durée cumulée est exprimé en minutes puis en heures minutes au format (hh:mm).

2. Écrire l'algorithme principal qui demande au plongeur de saisir la durée prévue de la plongée et qui appelle la procédure
3. Traduire les algorithmes en C#