Partie 9
Les Fonctions Prédéfinies

« Il y a deux méthodes pour écrire des programmes sans erreurs. Mais il n'y a que la troisième qui marche »
Anonyme

Certains traitements ne peuvent être effectués par un algorithme, aussi savant soit-il. D'autres ne peuvent l'être qu'au prix de souffrances indicibles.

C'est par exemple le cas du calcul du sinus d'un angle : pour en obtenir une valeur approchée, il faudrait appliquer une formule d'une complexité à vous glacer le sang. Aussi, que se passe-t-il sur les petites calculatrices que vous connaissez tous ? On vous fournit quelques touches spéciales, dites **touches de fonctions**, qui vous permettent par exemple de connaître immédiatement ce résultat. Sur votre calculatrice, si vous voulez connaître le sinus de 35°, vous taperez 35, puis la touche SIN, et vous aurez le résultat.

Tout langage de programmation propose ainsi un certain nombre de **fonctions** ; certaines sont indispensables, car elles permettent d'effectuer des traitements qui seraient sans elles impossibles. D'autres servent à soulager le programmeur, en lui épargnant de longs – et pénibles - algorithmes.

1. Structure générale des fonctions

Reprenons l'exemple du sinus. Les langages informatiques, qui se doivent tout de même de savoir faire la même chose qu'une calculatrice à 10 €, proposent généralement une fonction SIN. Si nous voulons stocker le sinus de 35 dans la variable A, nous écrirons :

```
A ← Sin(35)
```

Une fonction est donc constituée de trois parties :

- le **nom** proprement dit de la fonction. Ce nom ne s'invente pas ! Il doit impérativement correspondre à une fonction proposée par le langage. Dans notre exemple, ce nom est SIN.
- Deux parenthèses, une ouvrante, une fermante.
- Une liste de valeurs, indispensables à la bonne exécution de la fonction. Ces valeurs s'appellent des **arguments**, ou des **paramètres**. Certaines fonctions exigent un seul argument, d'autres deux, etc. et d'autres encore aucun. A noter que même dans le cas de ces fonctions n'exigeant aucun argument, les parenthèses restent obligatoires. Le nombre d'arguments nécessaire pour une fonction donnée ne s'invente pas : il est fixé par le langage. Par exemple, la fonction sinus a besoin d'un argument (ce n'est pas surprenant, cet argument est la valeur de l'angle). Si vous essayez de l'exécuter en lui donnant deux arguments, ou aucun, cela déclenchera une erreur à l'exécution. Notez également que les arguments doivent être d'un certain **type**, et qu'il faut respecter ces types.

Exercice 9.1

2. Les fonctions de texte

Une catégorie privilégiée de fonctions est celle qui nous permet de manipuler des chaînes de caractères. Nous avons déjà vu qu'on pouvait facilement « coller » deux chaînes l'une à l'autre avec l'opérateur de concaténation &. Mais ce que nous ne pouvions pas faire, et qui va être maintenant possible, c'est pratiquer des extractions de chaînes (moins douloureuses, il faut le noter, que les extractions dentaires).

Tous les langages, je dis bien tous, proposent peu ou prou les fonctions suivantes, même si le nom et la syntaxe peuvent varier d'un langage à l'autre :

`Len(chaîne)` : renvoie le nombre de caractères d'une chaîne

`Mid(chaîne,n1,n2)` : renvoie un extrait de la chaîne, commençant au caractère n1 et faisant n2 caractères de long.

Ce sont les deux seules fonctions de chaînes réellement indispensables. Cependant, pour nous épargner des algorithmes fastidieux, les langages proposent également :

`Left(chaîne,n)` : renvoie les n caractères les plus à gauche dans chaîne.

`Right(chaîne,n)` : renvoie les n caractères les plus à droite dans chaîne

`Trouve(chaîne1,chaîne2)` ou `InStr(chaîne1,chaîne2)` : renvoie un nombre correspondant à la position de chaîne2 dans chaîne1. Si chaîne2 n'est pas comprise dans chaîne1, la fonction renvoie zéro.

Exemples :

`Len("Bonjour, ça va ?")`	vaut	`16`
`Len("")`	vaut	`0`
`Mid("Zorro is back", 4, 7)`	vaut	`"ro is b"`
`Mid("Zorro is back", 12, 1)`	vaut	`"c"`
`Left("Et pourtant…", 8)`	vaut	`"Et pourt"`
`Right("Et pourtant…", 4)`	vaut	`"t…"`
`InStr(1,"Un pur bonheur", "pur")`	vaut	`4`
`InStr(1,"Un pur bonheur", "techno")`	vaut	`0`

Il existe aussi dans tous les langages une fonction qui renvoie le caractère correspondant à un code Ascii donné (fonction `Asc`), et Lycée de Versailles (fonction `Chr`) :

`Asc("N")`	vaut	`78`
`Chr(63)`	vaut	`"?"`

J'insiste ; à moins de programmer avec un langage un peu particulier, comme le C, qui traite en réalité les chaînes de caractères comme des tableaux, on ne pourrait pas se passer des deux fonctions Len et Mid pour traiter les chaînes. Or, si les programmes informatiques ont fréquemment à traiter des nombres, ils doivent tout aussi fréquemment gérer des séries de caractères (des chaînes). Je sais bien que cela devient un refrain, mais connaître les techniques de base sur les chaînes est plus qu'utile : c'est indispensable.

Exercice 9.2
Exercice 9.3
Exercice 9.4
Exercice 9.5
Exercice 9.6

3. Trois fonctions numériques classiques

Partie Entière

Une fonction extrêmement répandue est celle qui permet de récupérer la partie entière d'un nombre :

```
Après :   A ← Ent(3,228)                A vaut 3
```

Cette fonction est notamment indispensable pour effectuer le célébrissime test de parité (voir exercice dans pas longtemps).

Modulo

Cette fonction permet de récupérer le reste de la division d'un nombre par un deuxième nombre. Par exemple :

```
A ← Mod(10,3)              A vaut 1 car 10 = 3*3 + 1

B ← Mod(12,2)              B vaut 0 car 12 = 6*2

C ← Mod(44,8)              C vaut 4 car 44 = 5*8 + 4
```

Cette fonction peut paraître un peu bizarre, est réservée aux seuls matheux. Mais vous aurez là aussi l'occasion de voir dans les exercices à venir que ce n'est pas le cas.

Génération de nombres aléatoires

Une autre fonction classique, car très utile, est celle qui génère un nombre choisi au hasard.

Tous les programmes de jeu, ou presque, ont besoin de ce type d'outils, qu'il s'agisse de simuler un lancer de dés ou le déplacement chaotique du vaisseau spatial de l'enfer de la mort piloté par l'infâme Zorglub, qui veut faire main basse sur l'Univers (heureusement vous êtes là pour l'en empêcher, ouf).

Mais il n'y a pas que les jeux qui ont besoin de générer des nombres aléatoires. La modélisation (physique, géographique, économique, etc.) a parfois recours à des modèles dits stochastiques (chouette, encore un nouveau mot savant !). Ce sont des modèles dans lesquels les variables se déduisent les unes des autres par des relations déterministes (autrement dit des calculs), mais où l'on simule la part d'incertitude par une « fourchette » de hasard.

Par exemple, un modèle démographique supposera qu'une femme a en moyenne x enfants au cours de sa vie, mettons 1,5. Mais il supposera aussi que sur une population donnée, ce chiffre peut fluctuer entre 1,35 et 1,65 (si on laisse une part d'incertitude de 10%). Chaque année, c'est-à-dire chaque série de calcul des valeurs du modèle, on aura ainsi besoin de faire choisir à la machine un nombre au hasard compris entre 1,35 et 1,65.

Dans tous les langages, cette fonction existe et produit le résultat suivant :

```
Après :   Toto ← Alea()          On a :    0 =< Toto < 1
```

En fait, on se rend compte avec un tout petit peu de pratique que cette fonction Aléa peut nous servir pour générer n'importe quel nombre compris dans n'importe quelle fourchette. Je sais bien que mes lecteurs ne sont guère matheux, mais là, on reste franchement en deçà du niveau de feu le BEPC :

- Si Alea génère un nombre compris entre 0 et 1, Alea multiplié par z produit un nombre entre 0 et z. Donc, il faut estimer la « largeur » de la fourchette voulue et multiplier Alea par cette « largeur » désirée.
- Ensuite, si la fourchette ne commence pas à zéro, il va suffire d'ajouter ou de retrancher quelque chose pour « caler » la fourchette au bon endroit.

Par exemple, si je veux générer un nombre entre 1,35 et 1,65 ; la « fourchette » mesure 0,30 de large. Donc :

```
0 =< Alea()*0,30 < 0,30
```

Il suffit d'ajouter 1,35 pour obtenir la fourchette voulue. Si j'écris que :

```
Toto ← Alea()*0,30 + 1,35
```

Toto aura bien une valeur comprise entre 1,35 et 1,65. Et le tour est joué !

Bon, il est grand temps que vous montriez ce que vous avez appris…

Exercice 9.7

Exercice 9.8

Exercice 9.9

Exercice 9.10

Exercice 9.11

4. Les fonctions de conversion

Dernière grande catégorie de fonctions, là aussi disponibles dans tous les langages, car leur rôle est parfois incontournable, les fonctions dites de conversion.

Rappelez-vous ce que nous avons vu dans les premières pages de ce cours : il existe différents types de variables, qui déterminent notamment le type de codage qui sera utilisé. Prenons le chiffre 3. Si je le stocke dans une variable de type alphanumérique, il sera codé en tant que caractère, sur un octet. Si en revanche je le stocke dans une variable de type entier, il sera codé sur deux octets. Et la configuration des bits sera complètement différente dans les deux cas.

Une conclusion évidente, et sur laquelle on a déjà eu l'occasion d'insister, c'est qu'on ne peut pas faire n'importe quoi avec n'importe quoi, et qu'on ne peut pas par exemple multiplier "3" et "5", si 3 et 5 sont stockés dans des variables de type caractère. Jusque là, pas de scoop me direz-vous, à juste titre vous répondrai-je, mais attendez donc la suite.

Pourquoi ne pas en tirer les conséquences, et stocker convenablement les nombres dans des variables numériques, les caractères dans des variables alphanumériques, comme nous l'avons toujours fait ?

Parce qu'il y a des situations où on n'a pas le choix ! Nous allons voir dès le chapitre suivant un mode de stockage (les fichiers textes) où toutes les informations, quelles qu'elles soient, sont obligatoirement stockées sous forme de caractères. Dès lors, si l'on veut pouvoir récupérer des nombres et faire des opérations dessus, il va bien falloir être capable de convertir ces chaînes en numériques.

Aussi, tous les langages proposent-ils une palette de fonctions destinées à opérer de telles conversions. On trouvera au moins une fonction destinée à convertir une chaîne en numérique (appelons-la Cnum en pseudo-code), et une convertissant un nombre en caractère (Ccar).