Langage et programmation

ISN - Chapitre 2

Table des matières

I Type de données

Dans de nombreux langages de programmation, les expressions sont classées en fonction de leur type.

I 1 Les types de base

On distingue par exemple pour les nombres :

Le type **int** (integer : entier) pour définir des entiers relatifs.

Le type **float** (pour virgule flottante) pour définir des nombres réels.

On retrouve fréquemment d'autres types de données dans les programmes, comme les **booléens** (égal à VRAI ou FAUX). qui permettent de réaliser des tests en vu d'appliquer telle ou telle instruction.

I 2 Les tableaux

Un **tableau** est une structure pouvant contenir plusieurs valeurs du même type. On utilise dans de nombreux programmes les crochets [et] pour appeler un élément particulier de ce tableau.

Il est possible de représenter des tableaux bidimensionnels (plusieurs lignes, plusieurs colonnes) comme un « tableau de variables de type tableau », chacun de ces tableaux représentant une ligne. On utilise généralement la notation à double paire de crochets : *nom*[][].

I 3 Les chaines de caractères

On utilise le type **char** pour représenter un (unique) caractère.

On utilise le type **string** pour représenter une chaîne de caractères, généralement notée entre guillemets. Une variable de type *string* peut-être vu comme un tableau de variables de type *char*.

II Les fonctions

II 1 Notion de fonction

Il arrive d'écrire des programmes dans lesquels un bloc d'instructions revient plusieurs fois, parce que nous voulons faire plusieurs fois la même chose (par exemple, l'affichage d'un écran de fin de partie).

Définition d'une fonction

Dans les langages de programmation, une **fonction** est un bloc d'instructions isolé du reste du programme, qui possède un nom, et qui peut être appelée par ce nom à n'importe quel endroit du programme et autant de fois qu'on le veut.

Remarque : on appelle « appel d'une fonction » le fait d'exécuter le bloc d'instructions composant la fonction.

Utiliser des fonctions à plusieurs avantages :

- Éviter les répétitions dans le programme.
- Rendre les programmes plus courts.
- Rendre les programmes plus faciles à comprendre.
- Améliorer l'organisation du travail de développement, notamment à plusieurs.
- Faciliter les améliorations futures.

II 2 Portée des variables et passage d'arguments

Variables

Lorsque l'on définit des variables dans un programme, il faut se poser la question de leurs portées.

Une **variable globale** est une variable déclarée en début de programme, en dehors de toute fonction. Sa portée est le programme entier.

Une **variable locale** est une variable déclarée à l'intérieur d'une fonction. Elle n'est pas visible depuis les autres fonctions.

Remarque :
Il faut éviter d'utiliser le même nom pour une variable globale et une variable locale. Cependant, si cela se produit, à l'intérieur de la portée de la variable locale, c'est celle-ci qui est visible et non plus la variable globale, qui ne sera à nouveau accessible que lorsque l'on sera sorti de la portée de la variable locale.

Arguments

On appelle argument d'une fonction une variable particulière, utilisée dans le corps de la fonction, et dont la valeur est donnée dans le programme principal au moment où la fonction est appelée.

En programmation, le passage d'arguments permet de communiquer des informations depuis le programme principal vers une fonction.

Par exemple, une fonction **TracerPoint(x,y)** qui prend en *arguments* deux variables de type *int* x et y, définis dans le programme, et qui utilise ces arguments pour réaliser une action (par exemple tracer le point de coordonnées (x ;y)).

II 3 Retour d'une valeur

On veut aussi souvent communiquer des informations dans l'autre sens : depuis une fonction, vers le programme principal.

Les *fonctions* dans un programme permettent de renvoyer ce genre d'informations. La valeur produite par une fonction à partir de ses arguments, s'il en existe une, est appelée **valeur de retour**.

II 4 Fonctions récursives

Définition

Une fonction qui s'appelle elle-même est appelée une **fonction récursive**.

Ce point sera développé dans un chapitre dédié plus tard dans l'année.

III Programme et instructions

Lorsqu'on réalise un programme, il faut savoir prendre le temps, régulièrement, de le tester, pour éviter les bugs graves, et le corriger à temps.

III 1 Jeu de tests

Pour vérifier si un programme ne produit pas d'erreur au cours de son exécution, et s'il effectue la tâche attendue, une première méthode consiste à exécuter plusieurs fois ce programme, en lui fournissant des entrées, appelées tests, qui permettront de détecter les erreurs éventuelles.

Pour qu'elles jouent leur rôle, il faut choisir ces entrées de sorte que :

- On sache quelle doit être la sortie correcte du programme avec chacune de ces entrées.
- Chaque cas distinct d'exécution du programme soit parcouru avec au moins un choix d'entrées.
- Les cas limites soient essayés : les valeurs au bord d'une condition d'inégalité par exemple.

Lorsqu'on réalise un dossier-projet, on peut mettre dans ce dossier le jeu de tests à réaliser pour tester un programme.

III 2 Optimisation

En programmation, il est nécessaire d'essayer d'optimiser son programme, en le faisant régulièrement pour éviter d'avoir à faire des modifications trop conséquentes une fois le programme terminé.

Pour cela, on peut essayer de :

- Éviter les répétitions inutiles de bloc d'instructions.
- Éviter l'accumulation de variables inutiles.
- Eviter les tests (booléens) dont le résultat peut être anticipé.