Chapitre I

Langages de programmation et langage C

... un chapitre où l'on situe le langage C dans la très nombreuse famille des langages de programmation et où l'on écrit son premier programme dans ce langage.

Le langage C est l'un des plus utilisés des très nombreux langages de programmation existant actuellement. Le paragraphe 1 brosse un panorama de ces langages, il en propose une classification et dresse une chronologie des plus marquants d'entre eux.

Avant d'en étudier séparément chaque constituant, il est nécessaire d'avoir une vision d'ensemble d'un programme C. Le paragraphe 2 présente un exemple de programme C, très simple, et le dissèque.

Un programme est un objet qui évolue dans le temps. Entre sa conception et son abandon, il peut être exécuté et modifié de nombreuses fois. Le paragraphe 3 décrit les quatre grandes étapes du cycle de vie d'un programme.

Ce n'est qu'au chapitre V que l'on apprendra à organiser, assembler et exécuter un programme C dans toute sa généralité. Afin que le lecteur puisse faire les exercices proposés dans les quatre premiers chapitres, le paragraphe 4 explique la marche à suivre pour saisir, compiler et exécuter un programme C simple.

1. Langages de programmation : un panorama

Le rôle d'un langage de programmation est de décrire des tâches à soumettre à un ordinateur. Il existe à l'heure actuelle des centaines de langages de programmation. Ils se distinguent les uns des autres par leur niveau, par leur degré de généralité ou de spécialisation et par leur paradigme de programmation.

Un langage de programmation est dit de bas niveau s'il est proche du langage machine de l'ordinateur sur lequel il est exécuté. C'est le cas, par exemple, des langages d'assemblage. Inversement un langage de programmation est dit de haut niveau s'il n'est pas lié à un ordinateur particulier et s'il permet au programmeur de décrire de façon aussi concise que possible ce que doit faire un programme sans décrire en détail comment il doit le faire. Les langages d'interrogation de bases de données, par exemple, sont des langages de haut niveau : l'utilisateur pose une question et le système de gestion de base de données établit la stratégie la plus efficace pour y répondre.

Un langage de programmation généraliste est conçu pour programmer tout type d'application. Le langage C, objet de ce livre, est un langage généraliste. Un langage de programmation spécialisé est conçu, au contraire, pour une classe d'application particulière : un langage pour la création de sites web, par exemple. Les langages spécialisés sont souvent des langages de plus haut niveau que les langages généralistes.

On distingue quatre grands paradigmes de programmation : la programmation impérative, la programmation fonctionnelle, la programmation logique et la programmation orientée objet. Dans sa version la plus simple, un programme impératif se

présente comme une suite d'instructions adressées à l'ordinateur et dont chacune a pour effet de changer l'état de la mémoire ou de communiquer avec les unités périphériques de l'ordinateur (clavier, écran ou imprimante, par exemple). La mémoire d'un programme impératif contient un ensemble de variables. Une variable est une case de la mémoire qui a un nom et qui contient une valeur : la valeur de cette variable. Cette valeur peut changer au cours de l'exécution du programme. Voici un exemple de programme impératif :

```
(1)  var x, y, z: entier;
(2)  x := 2;
(3)  y := 5;
(4)  z := x + y;
(5)  afficher z.
```

On notera que le texte de ce programme a été écrit en fonte non proportionnelle. C'est une convention classique, adoptée dans ce livre. La ligne 1 contient la définition de trois variables : `x`, `y` et `z` dont les valeurs sont des nombres entiers. Les lignes 2, 3 et 4 contiennent chacune une instruction d'affectation qui demande l'enregistrement de la valeur de l'expression qui suit le symbole := dans la case mémoire associée à la variable dont le nom précède ce symbole. La ligne 5 contient une instruction qui demande l'affichage à l'écran de la valeur de l'expression qui suit le mot-clé `afficher`. L'exécution d'un programme impératif consiste à exécuter les instructions les unes après les autres, ce qui, dans le cas du programme ci-dessus, produira l'affichage à l'écran du nombre 7.

Un programme fonctionnel est une suite de définitions de fonctions. Par exemple :

```
(1)  pi = 3.14;
(2)  carre(x) = x * x;
(3)  aire_disque(r) = pi * carre(r);
```

La ligne 1 définit `pi` comme étant la fonction constante 3,14. La ligne 2 définit `carre` comme étant la fonction qui appliquée à une valeur `x` retourne cette valeur multipliée par elle-même. La ligne 3 définit `aire_disque` comme étant la fonction qui appliquée à une valeur `r` retourne le produit de `pi` par le carré de la valeur de `r`. L'exécution d'un programme fonctionnel consiste à poser une question sous la forme d'une application de fonction. La réponse est la valeur retournée par cette application. Par exemple, pour connaître l'aire d'un disque de rayon 5, l'utilisateur posera la question :

```
aire_disque(5);
```

La réponse sera 78,5 car :

```
aire_disque(5) = pi * carre(5) = 3.14 * 5 * 5 = 15.7 * 5 = 78.5
```

Un programme logique est constitué d'un ensemble de relations décrivant des faits et des règles. Par exemple :

```
(1)  pere("Alain", "Jean");
(2)  mere("Nicole", "Jean");
(3)  pere("Paul", "Marie");
(4)  mere("Claire", "Marie");
(5)  pere("Jean", "François");
(6)  mere("Marie", "François");
(7)  parent(x, y) si pere(x, y);
(8)  parent(x, y) si mere(x, y);
(9)  grand-parent(x, z) si parent(x, y) et parent(y, z);
```

Les lignes 1 à 6 contiennent des relations qui sont des faits : Alain est le père de Jean, Nicole est la mère de Jean, etc. La ligne 7 contient une relation qui est une règle stipulant que *x* est un parent de *y* si *x* est le père de *y*. La ligne 8 contient une relation qui est une

règle stipulant que *x* est un parent de *y* si *x* est la mère de *y*. La ligne 9 contient une relation qui est une règle stipulant que *x* est un grand-parent de *z* s'il existe un *y* tel que *x* est le parent de *y* et *y* est le parent de *z*. L'exécution d'un programme logique est lancée en posant une question sous la forme d'une relation à vérifier. Par exemple, pour connaître les grands-parents de François, l'utilisateur posera la question :

```
grand-parent(x, "François");
```

qui signifie : « Quelles sont les valeurs de `x` pour lesquelles la relation `grand-parent(x, "François")` est vraie ? ». La réponse sera :

```
{x = "Alain", x = "Nicole", x = "Paul", x = "Claire"}.
```

Un programme orienté objet est caractérisé par la façon dont sont représentées les données : un ensemble de classes dont les instances sont des objets. Une classe définit les propriétés des objets de cette classe et les opérations que l'on peut leur appliquer. Par exemple, la classe `Personne` possédant les propriétés `nom` et `annee_de_naissance` et l'opération `age` définie par `age(p) = annee_courante – annee_de_naissance(p)` où `p` est un objet de la classe `Personne`. Un objet est défini par un identificateur unique et invariable et par les valeurs de ses propriétés qui, elles, sont variables. Une classe peut être fille d'une autre classe. En ce cas, les objets de la classe fille héritent de toutes les propriétés et de toutes les opérations de la classe mère. Par exemple, si la classe `Etudiant` est une sous-classe de la classe `Personne`, alors un étudiant est aussi une personne qui a un nom et une année de naissance et dont l'âge peut être calculé par l'opération `âge` définie dans la classe `Personne`. Par ailleurs, un étudiant peut avoir des propriétés supplémentaires telle que `numero_étudiant`. La manipulation des données peut, elle, être réalisée selon le paradigme impératif ou fonctionnel.

Les langages de programmation mêlent en général plusieurs de ces quatre paradigmes. La possibilité de définir et d'appliquer des fonctions, notamment, est présente dans pratiquement tous les langages de programmation.

Les premiers langages de programmation pour ordinateur datent du début des années 1950 et plusieurs centaines ont vu le jour depuis. En voici une brève chronologie. Le premier est Fortran (Formula translator) apparu en 1954 et principalement orienté vers les applications numériques. Le premier langage de programmation fonctionnel, Lisp (*List processing*) est apparu en 1958. Il est principalement utilisé dans le domaine de l'intelligence artificielle. Le langage Algol (*Algorithmic oriented language*) dont la première version date de 1958 doit, bien que n'ayant pas dépassé le cadre universitaire, son importance à l'introduction de concepts qui ont ensuite été repris dans la plupart des langages de programmation modernes. En 1959 est apparu le langage Cobol (*Common business oriented language*) orienté vers un autre grand domaine d'application de l'informatique : la gestion. L'avènement, au début des années 1980, des micro-ordinateurs a popularisé deux langages à cause de leur simplicité : le langage Basic (*Beginner's all-purpose symbolic instruction code*) apparu en 1964 et le langage Pascal (du nom du mathématicien Blaise Pascal) dérivé d'Algol, apparu en 1971, qui a connu un succès immédiat à cause de la clarté de sa syntaxe et de sa portabilité. En 1972 est apparu le premier langage de programmation logique, Prolog, qui a ensuite été étendu à la programmation par contraintes. Comme Lisp, il est principalement utilisé dans le domaine de l'intelligence artificielle.

Le langage C est apparu au début des années 1970. Il est l'œuvre de trois chercheurs de la compagnie Bell aux USA : Brian Kernighan, Dennis Ritchie et Kenneth Thomson. Conçu à l'origine pour programmer le système d'exploitation UNIX, il est devenu l'un des langages de programmation les plus utilisés au monde. Sa syntaxe a de plus été reprise

dans d'autres langages eux-mêmes très utilisés tels que C++, PHP ou Java. C++, apparu en 1983, est une extension orientée objet de C. PHP, apparu en 1994, est destiné à la production de pages web dynamiques. Java, apparu en 1995, est un langage orienté objet qui permet l'écriture de logiciels portables sur la plupart des systèmes d'exploitation.

2. **Anatomie d'un programme C**

Un programme C manipule des valeurs (on dit aussi des données) classées par types : les nombres entiers ou les nombres décimaux, par exemple. Une valeur peut être affectée à une variable. Une variable est une case de la mémoire, repérée par son adresse, qui contient la valeur affectée à cette variable et qui peut être nommée.

Un programme C est composé d'un ensemble de fonctions. Ces fonctions s'appellent les unes les autres en se transmettant des valeurs. La définition d'une fonction spécifie son nom, le type des valeurs qui peuvent lui être transmises, le type de la valeur qu'elle retourne et son corps qui est une suite d'instructions dont l'exécution, retourne une valeur à la fonction appelante et éventuellement produit des effets de bord tels que des entrées/sorties. Tout programme C doit comporter au moins une fonction : la fonction `main` (fonction principale). L'exécution d'un programme C commence par l'appel de cette fonction.

Une instruction est un ordre donné à l'ordinateur de réaliser une suite d'actions dont chacune a pour effet de changer l'état de la mémoire ou le déroulement du programme ou bien de communiquer avec les unités d'entrées-sorties qui peuvent être un clavier, un écran, une imprimante ou un disque pour ne citer que les plus classiques. On dit qu'un programme écrit sur une unité de sortie (affichage à l'écran, impression sur papier ou enregistrement sur un disque) et qu'il lit sur une unité d'entrée (caractères saisis au clavier ou enregistrés dans un fichier).

Un programme C peut appeler des fonctions prédéfinies telles que les fonctions d'entrées-sorties ou les fonctions mathématiques usuelles. Les définitions de ces fonctions sont contenues dans la bibliothèque standard de C.

Un programme C est exécuté sous le contrôle du système d'exploitation qui est le logiciel qui assure la gestion de toutes les tâches soumises à l'ordinateur, par exemple : UNIX, Windows de Microsoft ou MacOS d'Apple.

Exemple I.1. Le programme suivant, appelé *Maximum*, est écrit en C. Il calcule le plus grand de deux nombres saisis au clavier par l'utilisateur et affiche le résultat à l'écran.

```
(1)   /*
(2)    * Calcul du maximum de deux nombres entiers
(3)    */
(4)   #include <stdio.h>
(5)   int max(int x, int y)
(6)      {
(7)      if (x > y)
(8)         return x;
(9)      else
(10)        return y;
(11)     }
```

```
(12)  int main(void)
(13)      {
(14)      int a, b, m;
(15)      printf("Saisir deux nombres entiers ? ");
(16)      scanf("%d%d", &a, &b);
(17)      m = max(a, b);
(18)      printf("Le maximum de %d et %d est %d.", a, b, m);
(19)      return 0;
(20)      }
```

Ce programme est construit de la façon suivante :

- Les lignes 1 à 3 contiennent un commentaire qui indique ce qu'effectue le programme.
- La ligne 4 indique que le programme appellera des fonctions d'entrée-sortie de la bibliothèque standard : la fonction `scanf` pour lire des nombres saisis au clavier (ligne 16) et la fonction `printf` pour afficher un message à l'écran (ligne 15 et 18).
- Les lignes 5 à 11 constituent la définition de la fonction `max` qui calcule et retourne le maximum des deux nombres qui lui sont transmis. La ligne 5 est l'en-tête de cette fonction. Elle indique que cette fonction retourne un nombre entier (type `int`), a pour nom `max` et est appelée avec deux nombres entiers qui seront affectées aux variables `x` et `y` de type `int` (nombre entier).
- Le corps de la fonction `max` est le bloc qui commence à la ligne 6 et se termine à la ligne 11. L'instruction `if` qui commence à la ligne 7 et se termine à la ligne 10 demande la comparaison des valeurs des variables `x` et `y`. Si la valeur de `x` est supérieure à celle de `y`, alors (instruction `return` de la ligne 8) la valeur de `x` sera retournée à la fonction qui a appelé la fonction `max`. Sinon (ligne 9), c'est la valeur de `y` qui sera retournée (instruction `return` de la ligne 10).
- Les lignes 12 à 20 constituent la définition de la fonction `main`. Le corps de cette fonction est le bloc qui commence à la ligne 13 et se termine à la ligne 20.
- La ligne 14 contient la définition de trois variables `a`, `b` et `m` de type `int` (nombre entier) auxquelles seront respectivement affectées les deux nombres saisis par l'utilisateur et le maximum de ces deux nombres.
- L'instruction de la ligne 15 appelle la fonction `printf` pour afficher à l'écran le message `Saisir deux nombres entiers ?` invitant l'utilisateur à saisir les deux nombres dont il recherche le maximum.
- L'instruction de la ligne 16 appelle la fonction `scanf` pour lire les deux nombres saisis par l'utilisateur et les affecter respectivement aux variables `a` et `b`.
- L'instruction de la ligne 17 appelle la fonction `max` pour calculer le maximum des valeurs des variables `a` et `b`. La valeur retournée par cet appel sera affectée à la variable `m` (opérateur =).
- L'instruction de la ligne 18 appelle la fonction `printf` pour afficher à l'écran le message `Le maximum de` *val*(`a`) `et` *val*(`b`) `est` *val*(`m`)`.` où *val*(`a`) est la valeur de la variable `a`, *val*(`b`) celle de la variable `b` et *val*(`m`) celle de la variable `m`.
- L'instruction de la ligne 19 rend la main au système d'exploitation en lui indiquant que le programme s'est bien déroulé.

L'exécution de ce programme se déroule de la façon suivante :

- La fonction `main` est appelée par le système d'exploitation.
- L'utilisateur est invité à saisir deux nombres (ligne 15).

- Il les saisit, par exemple, 5 et 19 (ligne 16).
- La fonction `max` est appelée avec les valeurs 5 et 19 (ligne 17) qui sont affectées respectivement aux variables `x` et `y` de la fonction `max`.
- La condition `x > y` est testée (ligne 7) : elle est fausse car la valeur 5 de `x` n'est pas supérieure à la valeur 19 de `y` ; l'instruction (ligne 10) introduite par le mot-clé `else` (ligne 9) est donc exécutée et la valeur 19 est retournée à la fonction `main`.
- La valeur 19 est affectée à la variable `m` (ligne 17).
- Le message `Le maximum de 5 et 19 est 19.` est affiché (ligne 18).
- L'exécution de l'instruction `return 0;` (ligne 19) rend la main au système d'exploitation en lui indiquant (retour de la valeur 0) que le programme s'est bien déroulé.

Cette exécution déclenche le dialogue suivant sur l'écran du poste de travail :

```
Saisir deux nombres entiers ? 5 19↵
Le maximum de 5 et 19 est 19.
```

où les caractères saisis par l'utilisateur sont en caractères gras et le caractère ↵ symbolise la frappe de la touche Entrée. □

3. Cycle de vie d'un programme

La vie d'un programme comporte quatre grandes étapes qui s'enchaînent cycliquement, jusqu'à l'abandon de ce programme :

1. **Conception du programme.** Il s'agit d'analyser l'application à programmer pour mettre en évidence les données à manipuler et les opérations à réaliser, choisir les meilleurs algorithmes pour réaliser ces opérations, décider d'une présentation des résultats adaptée aux utilisateurs du programme, etc. C'est une étape cruciale mais difficile lorsque l'application est complexe. Heureusement, il existe maintenant des méthodes et des outils sophistiqués pour aider à cette tâche.
2. **Ecriture du programme**. Il s'agit de traduire le résultat de la conception en un programme écrit dans le langage de programmation choisi. C'est ce que l'on apprend à faire dans ce livre avec le langage C.
3. **Compilation.** Pour qu'un programme puisse être exécuté par un ordinateur, il faut le traduire dans le langage machine de cet ordinateur. C'est le rôle du compilateur, qui lit le programme, vérifie s'il est lexicalement, syntaxiquement et sémantiquement correct et si c'est le cas, le traduit en langage machine produisant le code exécutable du programme. Si le programme n'est pas correct, le compilateur signale les erreurs en les localisant aussi précisément que possible dans le texte du programme. Il faut alors retourner à l'étape 2.
4. **Exécution.** Le code exécutable produit à l'issue de la compilation est soumis au système d'exploitation qui l'exécute. Dans la plupart des cas, un programme est fait pour être exécuté plusieurs fois. Tant qu'aucune modification n'a été apportée à ce programme, il n'est pas nécessaire de le recompiler avant chaque exécution. On conserve le code exécutable et on le soumet au système d'exploitation chaque fois que nécessaire. Il se peut que l'exécution du programme déclenche des erreurs non détectées à la compilation ou ne satisfasse pas ses utilisateurs, à cause d'une mauvaise écriture ou d'une mauvaise conception du programme. Il faut alors retourner à l'étape 2 ou à l'étape 1.

4. Compilation et exécution d'un programme C simple

Ce n'est qu'au chapitre V que l'on apprendra à organiser et écrire un programme C complet. En attendant, afin de pouvoir faire les exercices proposés dans les quatre premiers chapitres, voici la marche à suivre pour compiler et exécuter, dans un environnement Unix, avec le compilateur GCC[1], un programme dont le texte est enregistré dans un fichier unique :

1. Saisir le texte du programme sous un éditeur de texte et le sauver dans le fichier *prog*`.c` du répertoire courant (*prog* est le nom du programme).
2. Compiler le programme en soumettant la commande :

   `gcc -Wall` *prog*`.c -o` *prog*
3. Si le compilateur détecte des erreurs, elles seront affichées et il faudra les corriger. S'il n'y pas d'erreurs, le fichier exécutable *prog* est généré. L'option `-Wall`, bien que facultative, est recommandée car elle génère une liste très complète d'avertissements (*warnings*) sur des incorrections éventuelles ou des oublis dans le texte du programme qui pourraient provoquer des problèmes lors de l'exécution.
4. Exécuter le programme en soumettant la commande :

   prog
5. Le système d'exploitation appelle alors la fonction `main`. Cet appel déclenche l'exécution du programme.

Dans le cas où l'option `-o` *prog* n'est pas présente, le fichier exécutable généré a pour nom `a.out` et la commande à soumettre pour exécuter le programme est `a.out`. Sous MinGW, c'est un fichier exécutable Windows `a.exe` qui est généré. La commande à soumettre est tout simplement `a`.

Par exemple, si le texte du programme *Maximum* du paragraphe I.3 est enregistré dans le fichier source `maximum.c`, sa compilation sera lancée par la commande :

```
gcc -Wall maximum.c -o maximum
```

et son exécution par la commande :

```
maximum
```

Il est conseillé de regrouper les programmes correspondant aux exercices proposés en fin de chaque chapitre dans un même répertoire et de donner au fichier source un nom composé de la façon suivante : `exo.`*c*`.`*n*`.c` où *c* est le numéro du chapitre et *n* le numéro de l'exercice dans le chapitre.

5. Exercices

Exercice I.1. Saisir et exécuter le programme *Maximum*.

---

[1] GCC (*GNU Compiler Collection*) est une collection de compilateurs initialement écrits pour le système d'exploitation GNU, un système d'exploitation compatible avec UNIX, composé uniquement de logiciels libres et soutenu par la *Free Software Foundation* (http://www.fsf.org/). Sous Windows, on pourra opérer dans l'environnement MinGW (*Minimalist GNU for Windows*, http://www.mingw.org/) qui donne accès à GCC.

Chapitre II

Données

... un chapitre où l'on apprend à définir les données d'un programme C et comment est organisée sa mémoire.

Un programme C manipule des valeurs (on dit aussi des données). Dans la mémoire d'un ordinateur, une valeur est représentée sous forme binaire. Dans un programme, une valeur est :

- soit désignée par une constante qui peut être l'écriture littérale de cette valeur ou bien un nom donné à cette valeur ;
- soit affectée à une case de la mémoire appelée variable et accédée à partir du nom de cette variable ou de son l'adresse (sa position dans la mémoire).

Supposons, par exemple, que l'on veuille écrire un programme qui calcule puis affiche le volume d'une sphère dont le rayon est saisi par l'utilisateur du programme. La valeur approchée 3,14 de π, utilisée pour calculer ce volume pourra être désignée par la constante littérale `3.14` ou bien par la constante symbolique `PI` dont on aura déclaré qu'elle désigne cette valeur. Le rayon de la sphère saisi par l'utilisateur pourra être affecté à la variable `rayon` et le volume de cette sphère, une fois calculé pourra être affecté à la variable `volume`.

Les valeurs sont classées par types. Un type est un ensemble nommé de valeurs. L'ensemble des valeurs d'un type constitue l'extension de ce type. Par exemple, le type `unsigned char` a pour extension, sur la plupart des implantations de C, l'ensemble des nombres entiers appartenant à l'intervalle [0 ; 255]. Dire qu'une valeur est de type *T*, c'est dire que cette valeur appartient à l'extension de *T*. Dire qu'une constante est de type *T*, c'est dire que la valeur que désigne cette constante est de type *T*. Dire qu'une variable est de type *T*, c'est dire que les valeurs qui sont affectées à cette variable sont de type *T*.

Une valeur peut être simple ou composée. Une valeur simple peut être :

- un nombre entier ;
- un nombre décimal ;
- une adresse.

Une valeur composée peut être :

- une structure composée d'une suite de variables qui peuvent être de types différents (par exemple, une structure représentant un point d'un plan, composée du nom, de l'abscisse et de l'ordonnée de ce point) ;
- une union qui est une variable qui peut être d'un type à choisir parmi un ensemble de types possibles (par exemple, une union représentant un point décrit soit par ses coordonnées cartésiennes, soit par ses coordonnées polaires).

Il est de plus possible de définir des tableaux qui sont des suites de variables de même type rangées de façon contigüe dans la mémoire mais qui, attention, ne sont pas des