Programmation en C — Cours 1

6–17 septembre 2021

1 Un premier programme en C

Le programme suivant affiche la chaîne « `Bonjour` » :

```c
/* Un programme qui affiche bonjour. */

#include <stdio.h>

int
main(void)
{
    printf("Bonjour.\n");
    return 0;
}
```

La première ligne de ce programme est un *commentaire*, uniquement destiné au lecteur humain : le compilateur ignore tout ce qui se trouve entre `/*` et `*/`. La troisième ligne est une *directive du préprocesseur*, et indique au compilateur que nous nous servirons des interfaces de la bibliothèque d'entrées-sorties standard (la *standard input/output library*, ou `stdio`).

Le reste du programme est constitué de la *définition de fonction*, une suite d'instructions encadrée par des accolades « `{` » et « `}` » et ayant un nom (ici `main`). Vous étudierez les fonctions en détail plus tard dans le cours ; dans cette introduction, nous n'utiliserons que la fonction `main`.

Le corps de la fonction `main` du programme contient deux instructions. La première écrit sur la sortie standard la chaîne « `Bonjour.` » suivie d'un caractère de fin de ligne (« `\n` » pour *new line*). Comme toute instruction en C, elle est terminée par un point-virgule « `;` ».

La dernière instruction termine l'exécution de la fonction `main`.

2 Variables

Le langage C permet de définir des *variables*, qui sont des emplacements de la mémoire principale nommées et qui peuvent contenir des données. En C, en outre de son nom, toute variable a un *type*, qui spécifie la taille et l'interprétation de la mémoire associée à la variable. À tout moment de l'exécution du programme, une variable a une *valeur* qui est un élément du type de la variable.

```c
#include <stdio.h>

int
main(void)
{
    int i;
    i = 1;
    printf("La variable i vaut %d.\n", i);
    i = 2;
    printf("La variable i vaut maintenant %d.\n", i);
    i = i + 1;
    printf("La variable i vaut enfin %d.\n", i);
    return 0;
}
```

La fonction `main` du programme ci-dessus commence par la déclaration d'une variable `i` de type entier (« `int` », pour *integer*).

L'instruction `i = 1` est une instruction d'*affectation* qui stocke dans la variable `i` la valeur 1 ; la valeur de `i` restera 1 jusqu'à ce qu'on lui affecte une nouvelle valeur. Attention : malgré la notation utilisée, il ne s'agit pas d'une égalité, qui est une relation symétrique, mais d'une affectation, qui se lit de droite à gauche.

Par la suite, le programme fait deux autres affectations. La première, `i = 2`, est analogue à celle que nous avons déjà vue. La dernière, `i = i + 1`, est plus intéressante : elle affecte à `i` la valeur courante de `i` incrémentée (augmentée) de 1. Cette affectation se lit de droite à gauche tandis que l'égalité mathématique $i = i + 1$ est équivalente à $i + 1 = i$ et serait impossible à satisfaire.

Sortie formatée

L'instruction[1] `printf` permet de formater et d'écrire des données sur la sortie standard. Le premier paramètre de `printf` est une *chaîne de formatage* qui indique comment afficher les données ; l'endroit où la valeur du second paramètre doit être insérée est indiqué par « `%d` » (s'il s'agit d'un entier).

Il peut y avoir plusieurs occurrences de `%d` dans le format, et il faut alors passer le même nombre de paramètres supplémentaires à `printf`. Par exemple, on aurait pu écrire :

[1] En fait, c'est une invocation de fonction.

```
    i = 2;
    printf("Le carré de %d est %d.\n", i, i * i);
```

3 Entrées

```
#include <stdio.h>

int
main(void)
{
    int i;
    printf("Entrez un nombre: ");
    scanf("%d", &i);
    printf("Vous avez entré %d dont le carré est %d.\n", i, i * i);
    return 0;
}
```

L'instruction `scanf` permet de lire une valeur à partir de l'entrée standard. Son premier argument est similaire à celui de `printf`, et spécifie le format de l'entrée à laquelle on s'attend. Suivent les noms des variables à lire, qui cette fois sont précédés d'un signe « & » qui vous sera expliqué plus tard dans le cours.

4 Nombres à virgule flottante

Dans la partie ci-dessus, nous avons manipulé des *nombres entiers* codés sur 4 octets : nos variables étaient déclarées comme ayant le type `int`, nos constantes ne contenaient pas de point décimal, et nous utilisions le descripteur de format `%d` dans les formats passés à `printf` et `scanf`.

Le langage C permet aussi de manipuler des *nombres à virgule flottante*, codés sur 8 octets. Une variable contenant un nombre à virgule flottante doit être déclarée comme ayant le type `double` ; une constante de type `double` doit contenir un point décimal ; et le descripteur de format correspondant est `%lf`.

Surcharge des opérateurs Les opérateurs arithmétiques, « + », « − », etc., sont *surchargés* : leur opération dépend du type de leurs opérandes. Lorsqu'ils sont appliqués à des opérandes de types différents, l'opérande du type le plus petit est automatiquement converti dans le type du plus grand.

La surcharge est particulièrement notable dans le cas de l'opérateur « / », qui effectue une division entière (euclidienne) sur les entiers, et une division en virgule flottante sur les valeurs à virgule flottante.

Exercice Quel est le type et la valeur des expressions suivantes ?

1. `5 / 2;`

2. `5.0 / 2.0;`

3. `5 / 2.0;`

4. `(1 / 3) * 3;`

5. `(1.0 / 3.0) * 3.0`.