Cours 1 : Les conditionnelles

Guillaume Aubian

Ce document a été grandement inspiré par – et largement copié sur – le document correspondant du responsable précédent de ce cours, Juliusz Chroboczek, avec son accord.

1 Un premier programme en C

Le programme suivant affiche « Hello World ».

```
#include <stdio.h>

int main() {

    /* Ceci va afficher Hello World. */
    printf("Hello World.\n");

    return 0;
}
```

Les deux premières lignes sont, pour le moment, de la magie : ne cherchons pas à comprendre leur signification pour le moment (il s'agit d'une directive de préprocesseur et d'une définition de fonction). Pour l'instant vous écrirez simplement entre les accolades qui suivent.

La troisième ligne est un commentaire, uniquement destiné au lecteur humain : le compilateur ignore ce qui se trouve entre /* et */. La ligne suivante appelle la fonction printf qui affiche alors Hello World. Enfin, l'instruction suivante termine l'exécution de main. Notez que ces deux instructions finissent par un point-virgule : c'est la règle en C.

2 Les variables

Une variable a un *nom* et un *type*, qui ne change pas. À tout moment, elle a aussi une *valeur*, qui peut changer.

```
#include <stdio.h>

int main() {

    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 déclaration d'une variable prend la forme « `typedelavariable nomdelavariable;` ». Ici « `int i;` ».

Pour assigner une valeur a une variable, on écrit « `nomdelavariable = valeuraaffecter;` ». Ici « `i = 1;` », « `i = 2;` » ou « `i = i + 1;` ». **Attention : il ne s'agit pas d'une égalité, qui est une opération symétrique, mais d'une affectation, qui ne l'est pas. Ainsi, « `1 = i;` » ou « `i + 1 = i;` » n'ont aucun sens.**

Sortie formatée

La fonction `printf` permet de formater et d'afficher des données sur la sortie standard. Le premier paramètre est une *chaîne de formatage*, une sorte de chaîne de caractères sous stéroïdes. Sans instruction supplémentaire, il s'agit simplement d'un texte à afficher. Néanmoins, `%d` permet par exemple d'indiquer l'emplacement d'un entier, dont la valeur est précisée dans les arguments ultérieurs de `printf`.

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

3 Entrée

```
#include <stdio.h>

int main() {

    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'une esperluette (symbole « `&` »). Vous verrez plus tard son utilité.

4 Le type des booléens

La plupart des langages actuels ont un type pour les booléens, i.e. ce qui est vrai (on écrit alors `true`) ou faux (`false`). Ce n'est malheureusement pas le cas en C. Par convention, pour représenter que quelque chose (notamment, une inégalité) est vrai ou faux, C utilise la convention suivante : 0 pour dire que c'est faux, n'importe quel autre entier pour dire que c'est vrai.

Ainsi $1337 < 42$ renvoie 0.

```
#include <stdio.h>

int main() {

    printf("%d", 1337 < 42);

    return 0;
}
```

« > », « <= », « >= » ou « == » (on utilise bien un double signe « = » ici, le signe seul étant réservé aux affectations) fonctionnent de façon similaire.

5 Conditionnelle

```
#include <stdio.h>

int main() {

    int i;
    scanf("%d", &i);
    int b;
    b = i > 2;
    if(b) {
        printf("Ici, i est strictement supérieur à 2.\n");
    }
    else {
        printf("Là, i est inférieur ou égal à 2.\n");
    }

    return 0;
}
```

La conditionnelle if consiste du mot clé `if` suivi d'un booléen entre parenthèses et de deux *branches* séparées par le mot clé else. Si la condition est vraie, c'est la première branche qui est exécutée ; sinon, c'est la deuxième. La deuxième branche peut être omise lorsqu'elle est vide

Souvent, on ne s'embête pas à exprimer la condition dans un booléen, mais on la met directement entre les parenthèse suivant le `if`.

```
#include <stdio.h>

int main() {

    int i;
    scanf("%d", &i);
    if(i > 2) {
        printf("Ici, i est strictement supérieur à 2.\n");
    }
    else {
        printf("Là, i est inférieur ou égal à 2.\n");
    }
    return 0;

}
```

Insistons sur le fait que la condition est un booléen. Notamment, il existe des outils bien pratiques pour manipuler les booléens : les opérateurs booléens. On pourra citer « `&&` » (« et ») vrai si ses deux opérandes sont tous deux vrais, faux sinon ; « `||` » (« ou »), faux si ses deux opérandes sont tous deux faux, vrai sinon ; et enfin « `!` » (« non »), vrai si son opérande est faux, faux sinon.

6 Conditionnelles

On peut mettre ce qu'on veut dans les branches d'une conditionnelle, notamment une autre conditionnelle. On dit alors que c'est une *conditionnelle imbriquée*.

```
#include <stdio.h>

int main() {

    int i, j;
    scanf("%d", &i);
    scanf("%d", &j);
    if(i == j) {
        printf("Match nul.\n");
    } else {
```

```
        if(i > j) {
            printf("Le premier joueur a gagné.\n");
        }
        else {
            printf("Le deuxième joueur a gagné.\n");
        }
    }

    return 0;
}
```

Une suite de conditionnelles imbriquées augmente l'indentation du code. On peut éviter ce problème en omettant les accolades de chaque `else`, et en mettant chaque `if` sur la même ligne que le `else` qui précède. Une telle suite de conditionnelles s'appelle une *conditionnelle filée*.

```
#include <stdio.h>

int main() {
    int i, j;
    scanf("%d", &i);
    scanf("%d", &j);
    if(i == j) {
        printf("Match nul.\n");
    } else if(i > j) {
        printf("Le premier joueur a gagné.\n");
    } else {
        printf("Le deuxième joueur a gagné.\n");
    }

    return 0;
}
```