

I. Prolégomènes

1 À l'aide d'un terminal de commandes, créer un nouveau répertoire nommé **TP2** dans votre répertoire **TP**, et se déplacer à l'intérieur. Ouvrir un nouveau fichier **tp2.c** avec l'IDE VSCodium.

2 Dans **tp2.c**, écrire le squelette de base d'un code C, en tentant de le faire de mémoire sans regarder.

Une première habitude à prendre

Dans ce TP et dans tous les suivants, il vous est demandé de **régulièrement sauvegarder votre code**, mais également de **régulièrement le compiler**, par exemple après chaque question. Cela est une bonne pratique qui vous permettra de détecter rapidement d'éventuels problèmes grâce aux erreurs ou warnings affichés par le compilateur.

II. Affichage en C

Intéressons-nous à l'instruction `printf("Hello CIV !\n")`. La fonction `printf` permet d'afficher dans le terminal la chaîne de caractères qu'elle a en argument. Le caractère spécial `'\n'` indique un retour à la ligne (ce n'est pas automatique contrairement à Python, c'est une bonne pratique de le mettre à chaque fois). Il est possible d'afficher des valeurs en C en les ajoutant comme arguments supplémentaires. Mais pour cela, il faut ajouter un spécificateur de format par argument supplémentaire dans la chaîne à afficher. Ces derniers indiquent où afficher la variable dans la chaîne, et comment interpréter l'argument pour correctement l'afficher (est-ce un entier naturel ? un flottant ? ...).

Type	`int`	`unsigned int`	`float`	`char`	`string`	`bool`
Spécificateur de format	`%d`	`%u`	`%f`	`%c`	`%s`	`%d` est acceptable

3 Dans la fonction `main`, ajouter la ligne suivante avant `return 0;` :

```
printf("Septembre a %d heures et pi vaut environ %f.\n", 30 * 24, 3.1416);
```

Compiler votre fichier, exécuter le fichier produit et admirer le résultat.

III. Variables et types simples en C

Pour **définir** une variable en C, il faut donner son **type**, son **identifiant** (son nom) et enfin une **expression** correspondant à sa valeur initiale après le symbole `=`. La variable est ainsi initialisée. Nous utiliserons des noms de variables en **snake_case**, c'est-à-dire en et avec les termes séparés par des underscores. Par exemple :

```
int ma_variable = 108;      int nb_heures = 30 * 24;      float pi = 3.1416;      char c = 'P';
```

4 Ajouter ces définitions de variables dans votre fonction `main`, ainsi qu'une ligne permettant de toutes les afficher avec la fonction `printf`. Attention aux caractères (type `char`) qui doivent être définis entre guillemets simples. Que remarquez-vous pour l'affichage de la variable `pi` ? Pourquoi ?

Pour **déclarer** une variable, on donne uniquement son type et son identifiant. Par exemple `int ma_variable;`. Il restera alors à initialiser sa valeur plus tard dans le code, en l'affectant.

5 Déclarer une nouvelle variable entière, mais sans la définir. Que se passe-t-il si l'on essaie de l'afficher avec la fonction `printf` ? À retenir : ne jamais utiliser ou afficher de variable non définie.

Une fois qu'une variable est définie ou déclarée, il est possible de lui affecter une nouvelle valeur avec l'opérateur `=`, par exemple `ma_variable = 108108;`. Il n'y a plus besoin de préciser le type pour affecter une nouvelle valeur, car la variable est déjà connue. En revanche, il est complètement interdit d'affecter une valeur à une variable qui n'aurait pas été déclarée ou définie plus tôt.

6 Toujours dans la fonction `main`, affecter de nouvelles valeurs à vos variables, et vérifier grâce à un affichage supplémentaire que le changement de valeur a bien eu lieu. Quelle erreur obtenez-vous si vous tentez d'affecter une valeur à une variable non déclarée et non définie ?

Lorsque l'on affecte une valeur à une variable, le terme de droite peut être n'importe quelle expression qui puisse être évaluée du bon type : cela peut-être une valeur brute (ex. un entier écrit littéralement), ou un calcul qui peut faire intervenir des variables déjà définies et des sorties de fonctions. On peut par exemple incrémenter une variable `x` avec l'affectation `x = x + 1;`. Le programme se charge d'évaluer combien vaut `x + 1`, puis l'affecte.

Il est également possible d'utiliser la notation binaire plutôt que décimale lors de l'affectation d'une variable entière, en rajoutant le préfixe `0b` immédiatement (sans espace) devant la séquence de bits en question.

7 Définir une variable `unsigned int v_binaire = 0b10010110;`. Prédire quelle valeur aura cette variable, et vérifier le résultat en l'affichant. En utilisant le type `int` cette fois-ci, pourquoi la variable est positive (et de même valeur), alors que la séquence binaire commence a priori par un 1 ?

8 Afficher votre variable `c` de type `char` en utilisant le spécificateur de format du type `int` (`%d`). Que représente l'affichage obtenu ? Réciproquement, définir une variable entière de valeur 70 et l'afficher comme un caractère.

9 Le mot-clef `sizeof` permet d'obtenir la taille (en octets) d'un type C donné en argument, par exemple `sizeof(int)` renvoie 4. Afficher les tailles des type `float` et `double` et vérifier que ces derniers sont corrects. Attention, le type renvoyé est `long unsigned int`, il faut utiliser `%lu`.

IV. Conditionnelles en C

Graphe de flot de contrôle

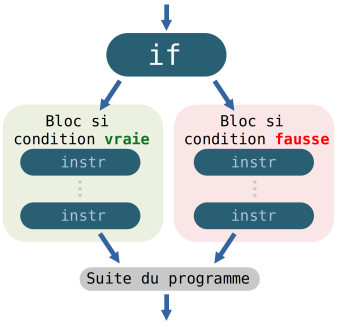


Schéma général

```c
if (condition) {
    instr1;
    ...
} else {
    instr2;
    ...
}
// Suite du programme
```

Exemple

```c
if (ma_variable == 108) {
    printf("Bonne valeur.\n");

} else {
    ma_variable = 108;
    printf("Valeur corrigée.\n");
}
// Suite du programme
```

Un branchement conditionnel s'écrit avec le mot-clef `if` suivi d'une condition entre parenthèses. Cette condition peut s'écrire à l'aide de l'opérateur d'égalité `==` entre deux valeurs, des comparaisons `>`, `<`, `>=`, `<=`, ou de manière générale via l'évaluation d'une fonction renvoyant un booléen. Il est possible de former des conditions plus complexes en combinant ces prédicats à l'aide de l'opérateur de négation, de l'opérateur de conjonction (vérifie si les deux prédicats sont vrais) ou l'opérateur de disjonction (vérifie si au moins un des deux prédicats est vrai).

Symbole	`==`	`!=`	`>` ou `<`	`>=` ou `<=`	`!`	`&&`	`||`
Signification	Égalité	Inégalité	Comparaison stricte	Comparaison large	Négation (NON)	Conjonction (ET)	Disjonction (OU)

Les instructions à exécuter si la condition est vraie ou si elle est fausse sont encadrées par des blocs délimités par des accolades { et }. Il est possible de ne pas ajouter le mot-clef `else`, ce qui revient au fait de laisser le deuxième bloc d'instructions vide : si la condition n'est pas vérifiée on passera directement à la suite du programme.

10 Dans la fonction `main`, ajouter une conditionnelle qui affiche "Égalité" si 2 + 2 est égal à 4, et affiche "Pas d'égalité" sinon. Faire de même pour vérifier si 2 milliards et 2 milliards font 4 milliards. Que vous dit alors le compilateur et quel résultat obtenez-vous ? Pourquoi ?

11 De la même manière, utiliser une conditionnelle et l'affichage pour vérifier si 0.1 + 0.2 est égal à 0.3. Que penser du résultat ?

12 L'addition de flottants n'est pas associative et dépend de l'ordre d'exécution des opérations. Comparer le résultat de ces deux sommes de trois flottants réalisées dans des ordres différents : `float f1 = 1e30 + (-1e30 + 0.1);` et `float f2 = (1e30 + -1e30) + 0.1;`. Expliquer.

On retiendra qu'il faut faire particulièrement attention aux valeurs des entiers manipulés, car C n'aura pas toujours de garde-fou pour vous prémunir des résultats aberrants obtenus suite à un dépassement d'entier. De même, lorsqu'il s'agira de comparer des flottants entre eux, il est nécessaire d'ajouter une petite marge de sécurité en réalisant par exemple une égalité à ε près pour lutter contre les erreurs d'arrondis.

13 Prédire puis vérifier expérimentalement si les conditions suivantes sont vraies ou fausses :

- `0 + 0 == 0`
- `(0 > 1) && (7 <= 7)`
- `!(a > a) && (1 + 1 == 2)` pour `a` de type `int`
- `(0 == 1) || ((3 < 7) && !(2 + 2 == 5))`

V. Prototypes de fonctions en C

Toute fonction en C possède un prototype, qui décrit le nom de la fonction, son type de sortie ainsi que tous ses paramètres et leurs types. Par exemple le prototype `int addition(int a, int b)` nous décrit une fonction nommée `addition` qui prend en argument deux entiers `a` et `b` et renvoie un entier.

14 Noter le prototype d'une fonction `multiplication` qui renvoie la multiplication de deux entiers, d'une fonction `discriminant` qui renvoie le discriminant d'un polynôme du second degré à partir de ses coefficients, et le prototype d'une fonction `nb_racines` qui renvoie son nombre de racines.

VI. Fonctions en C

Pour **définir une fonction** en C, il faut écrire son prototype (en dehors de la fonction `main`) puis son code à l'intérieur d'accolades. Tout comme les variables, il est possible de **déclarer une fonction** sans la définir en n'écrivant pas le bloc d'instructions délimité par les accolades. De même, il ne faut pas appeler une fonction déclarée mais non définie.

15 Implémenter les fonctions `multiplication`, `discriminant` et `nb_racines` avant votre fonction `main`. Appeler vos fonctions et afficher ce qu'elles renvoient sur quelques valeurs de votre choix.

VII. Assert

Certaines fonctions ne renvoient rien, par exemple des fonctions de test ou d'affichage. Pourtant, il faut bien leur assigner un type lors de leur déclaration. On utilise alors le type `void`, pour "vide".

assert

Le mot-clef `assert` est fondamental en C : il permet de s'assurer qu'une condition est vraie durant l'exécution du programme. Par exemple, si une variable `x` est censé être non nulle, on peut ajouter la ligne `assert(x != 0);`. Si la condition est effectivement vraie comme attendue, l'exécution continue sans changement : on passe à la suite du programme. En revanche si la condition est fausse, c'est qu'on est tombé dans un cas inattendu : autant s'arrêter immédiatement avant de provoquer de plus gros dégâts plus loin.

16 Implémenter une fonction de prototype `void test_mult()`, qui ne prend aucun argument et ne renvoie rien (d'où son type de sortie `void`), et qui réalise 5 tests sur la valeur de sortie de la fonction `multiplication`, de la forme `assert(multiplication(12, 9) == 108);`.

17 Appeler cette fonction de tests dans votre fonction `main`, et vérifier que l'exécution se déroule bien. Modifier ensuite un de vos `assert` avec un test incorrect et observer ce qu'il se passe à l'exécution.