**PROGRAMMER EN C**

**EXTRAITS LES PLUSINTERESSANT DU COURS OPENCLASSROOM**

**14189 Apprenez à programmer en c**

# La structure

Analysons le code minimal

#include <stdio.h>

#include <stdlib.h>

Ce sont des lignes spéciales que l'on ne voit qu'en haut des fichiers source. Ces lignes sont facilement reconnaissables car elles commencent par un dièse #. Ces lignes spéciales, on les appelle directives de préprocesseur (un nom compliqué, n'est-ce pas ?).

Ce sont des lignes qui seront lues par un programme appelé préprocesseur, un programme qui se lance au début de la compilation.

Oui : comme je vous l'ai dit plus tôt, ce qu'on a vu au début n'était qu'un schéma très simplifié de la compilation. Il se passe en réalité plusieurs choses pendant une compilation. On les détaillera plus tard : pour le moment, vous avez juste besoin d'insérer ces lignes en haut de chacun de vos fichiers.

Le mot include en anglais signifie « inclure » en français. Ces lignes demandent d'inclure des fichiers au projet, c'est-à-dire d'ajouter des fichiers pour la compilation.

Il y a deux lignes, donc deux fichiers inclus. Ces fichiers s 'appellent stdio.h et stdlib.h. Ces fichiers existent déjà, des fichiers source tout prêts. On verra plus tard qu'on les appelle des bibliothèques (certains parlent aussi de librairies mais c'est un anglicisme). En gros, ces fichiers contiennent du code tout prêt qui permet d'afficher du texte à l'écran.

Sans ces fichiers, écrire du texte à l'écran aurait été mission impossible. L'ordinateur à la base ne sait rien faire, il faut tout lui dire.

Vous voyez la galère dans laquelle on est !

Bref, en résumé les deux premières lignes incluent les bibliothèques qui vont nous permettre (entre autres) d'afficher du texte à l'écran assez « facilement ».

Passons à la suite. La suite, c'est tout ça :

int main()

{

printf("Hello world!\n");

return 0;

}

Ce que vous voyez là, c'est ce qu'on appelle une fonction. Un programme en langage C est constitué de fonctions, il ne contient quasiment que ça. Pour le moment, notre programme ne contient donc qu'une seule fonction.

Une fonction permet grosso modo de rassembler plusieurs commandes à l'ordinateur. Regroupées dans une fonction, les commandes permettent de faire quelque chose de précis. Par exemple, on peut créer une fonction ouvrir\_fichier qui contiendra une suite d'instructions pour l'ordinateur lui expliquant comment ouvrir un fichier.

L'avantage, c'est qu'une fois la fonction écrite, vous n'aurez plus qu'à dire ouvrir\_fichier, et votre ordinateur saura comment faire sans que vous ayez à tout répéter !

Sans rentrer dans les détails de la construction d'une fonction (il est trop tôt, on reparlera des fonctions plus tard), analysons quand même ses grandes parties. La première ligne contient le nom de la fonction, c'est le deuxième mot.

Oui : notre fonction s 'appelle donc main. C'est un nom de fonction particulier qui signifie « principal ». Main est la fonction principale de votre programme, c'est toujours par la fonction main que le programme commence.

Une fonction a un début et une fin, délimités par des accolades { et }. Toute la fonction main se trouve donc entre ces accolades. Si vous avez bien suivi, notre fonction main contient deux lignes :

printf("Hello world! \n");

return 0;

Ces lignes à l'intérieur d'une fonction ont un nom. On les appelle instructions (ça en fait du vocabulaire qu'il va falloir retenir).

Chaque instruction est une commande à l'ordinateur. Chacune de ces lignes demande à l'ordinateur de faire quelque chose de précis.

Comme je vous l'ai dit un peu plus haut, en regroupant intelligemment (c'est le travail du programmeur) les instructions dans des fonctions, on crée si on veut des « bouts de programmes tout prêts ». En utilisant les bonnes instructions, rien ne nous empêcherait donc de créer une fonction ouvrir\_fichier comme je vous l'ai expliqué tout à l'heure, ou encore une fonction avancer\_personnage dans un jeu vidéo, par exemple.

Un programme, ce n'est au bout du compte rien d'autre qu'une série d'instructions : « fais ceci », « fais cela ». Vous donnez des ordres à votre ordinateur et il les exécute. Du moins si vous l'avez bien dressé.

Très important : toute instruction se termine obligatoirement par un point-virgule « ; ». C'est d'ailleurs comme ça qu'on reconnaît ce qui est une instruction et ce qui n'en est pas une. Si vous oubliez de mettre un point-virgule à la fin d'une instruction, votre programme ne compilera pas !

La première ligne : printf("Hello world!\n"); demande à afficher le message « Hello world! » à l'écran. Quand votre programme arrivera à cette ligne, il va donc afficher un message à l'écran, puis passer à l'instruction suivante.

Passons à l'instruction suivante justement :

return 0;

Eh bien ça, en gros, ça veut dire que c'est fini (eh oui, déjà). Cette ligne indique qu'on arrive à la fin de notre fonction main et demande de renvoyer la valeur 0.

Pourquoi mon programme renverrait-il le nombre 0 ?

En fait, chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s 'est bien passé. En pratique, 0 signifie « tout s 'est bien passé » et n'importe quelle autre valeur signifie « erreur ». La plupart du temps, cette valeur n'est pas vraiment utilisée, mais il faut quand même en renvoyer une.

Votre programme aurait marché sans le return 0, mais on va dire que c'est plus propre et plus sérieux de le mettre, donc on le met.

Et voilà ! On vient de détailler un peu le fonctionnement du code minimal.

Certes, on n'a pas vraiment tout vu en profondeur, et vous devez avoir quelques questions en suspens. Soyez rassurés : toutes vos questions trouveront une réponse petit à petit. Je ne peux pas tout vous divulguer d'un coup, cela ferait trop de choses à assimiler.

Vous suivez toujours ? Si tel n'est pas le cas, rien ne presse. Ne vous forcez pas à lire la suite. Faites une pause et relisez ce début de chapitre à tête reposée. Tout ce que je viens de vous apprendre est fondamental, surtout si vous voulez être sûrs de pouvoir suivre après.

Tenez : comme je suis de bonne humeur, je vous fais un schéma qui récapitule le vocabulaire qu'on vient d'apprendre (fig. suivante)



Testons notre programme

Tester devrait aller vite. Tout ce que vous avez à faire c'est compiler le projet, puis l'exécuter (cliquez sur l'icône Build & Run sous Code::Blocks).

Si vous ne l'avez pas encore fait, on vous demandera d'enregistrer les fichiers. Faites-le.

Si la compilation ne fonctionne pas et que vous avez une erreur de ce type : "My-program - Release" uses an invalid compiler. Skipping... Nothing to be done... Cela signifie que vous avez téléchargé la version de Code::Blocks sans mingw (le compilateur). Retournez sur le site de Code::Blocks pour télécharger la version avec mingw.

Après un temps d'attente insupportable (la compilation), votre premier programme va apparaître sous vos yeux totalement envahis de bonheur.

## Les caractères spéciaux

Les caractères spéciaux sont des lettres spéciales qui permettent d'indiquer qu'on veut aller à la ligne, faire une tabulation, etc.

Ils sont faciles à reconnaître : c'est un ensemble de deux caractères. Le premier d'entre eux est toujours un anti-slash ( \ ), et le second un nombre ou une lettre. Voici deux caractères spéciaux courants que vous aurez probablement besoin d'utiliser, ainsi que leur signification :

\n : retour à la ligne (= « Entrée ») ;

\t : tabulation.

Dans notre cas, pour faire une entrée, il suffit de taper \n pour créer un retour à la ligne.

Si je veux donc faire un retour à la ligne juste après le mot « Bonjour », je devrais taper :

printf("Bonjour\n");

Votre ordinateur comprend qu'il doit afficher « Bonjour » suivi d'un retour à la ligne.

# Les variables

## Déclarer une variable

Alors une variable, c'est quoi ?

Eh bien c'est une petite information temporaire qu'on stocke dans la RAM. Tout simplement.

On dit qu'elle est « variable » car c'est une valeur qui peut changer pendant le déroulement du programme. Par exemple, notre nombre 5 de tout à l'heure (le nombre de vies restant au joueur) risque de diminuer au fil du temps. Si ce nombre atteint 0, on saura que le joueur a perdu.

Nos programmes, vous allez le voir, sont remplis de variables. Vous allez en voir partout, à toutes les sauces.

En langage C, une variable est constituée de deux choses :

* une **valeur** : c'est le nombre qu'elle stocke, par exemple 5 ;
* un **nom** : c'est ce qui permet de la reconnaître. En programmant en C, on n'aura pas à retenir l'adresse mémoire (ouf !) : à la place, on va juste indiquer des noms de variables. C'est le compilateur qui fera la conversion entre le nom et l'adresse.

## Donner un nom à ses variables

En langage C, chaque variable doit donc avoir un nom. Pour notre fameuse variable qui retient le nombre de vies, on aimerait bien l'appeler « Nombre de vies » ou quelque chose du genre.

Hélas, il y a quelques contraintes. Vous ne pouvez pas appeler une variable n'importe comment :

* il ne peut y avoir que des minuscules, majuscules et des chiffres (abcABC012) ;
* votre nom de variable doit commencer par une lettre ;
* les espaces sont interdits. À la place, on peut utiliser le caractère « underscore » \_ (qui ressemble à un trait de soulignement). C'est le seul caractère différent des lettres et chiffres autorisé ;
* vous n'avez pas le droit d'utiliser des accents (éàê etc.).

Enfin, et c'est très important à savoir, le langage C fait la différence entre les majuscules et les minuscules. Pour votre culture, sachez qu'on dit que c'est un langage qui « respecte la casse ».

Donc, du coup, les variables largeur, LARGEUR ou encore LArgEuR sont trois variables différentes en langage C, même si pour nous ça a l'air de signifier la même chose !

Voici quelques exemples de noms de variables corrects : nombreDeVies, prenom, nom, numero\_de\_telephone, numeroDeTelephone.

Chaque programmeur a sa propre façon de nommer des variables. Pendant ce cours, je vais vous montrer ma manière de faire :

* je commence tous mes noms de variables par une lettre minuscule ;
* s 'il y a plusieurs mots dans mon nom de variable, je mets une lettre majuscule au début de chaque nouveau mot.

Je vais vous demander de faire de la même manière que moi, ça nous permettra d'être sur la même longueur d'ondes.

Quoi que vous fassiez, faites-en sorte de donner des noms clairs à vos variables. On aurait pu abréger nombreDeVies, en l'écrivant par exemple ndv. C'est peut-être plus court, mais c'est beaucoup moins clair pour vous quand vous relisez votre code. N'ayez donc pas peur de donner des noms un peu plus longs pour que ça reste compréhensible.

## Les types de variables

Notre ordinateur, vous pourrez le constater, n'est en fait rien d'autre qu'une (très grosse) machine à calculer. Il ne sait traiter que des nombres.

Oui mais voilà, j'ai un scoop ! Il existe plusieurs types de nombres ! Par exemple, il y a les nombres entiers positifs :

* 45 ;
* 398 ;
* 7650.

Mais il y a aussi des nombres décimaux, c'est-à-dire des nombres à virgule :

* 75,909 ;
* 1,7741 ;
* 9810,7.

En plus de ça, il y a aussi des nombres entiers négatifs :

* -87 ;
* -916.

… Et des nombres négatifs décimaux !

* -76,9 ;
* -100,11.

Votre pauvre ordinateur a besoin d'aide ! Lorsque vous lui demandez de stocker un nombre, vous devez dire de quel type il est.

Ce n'est pas vraiment qu'il ne soit pas capable de le reconnaître tout seul, mais… ça l'aide beaucoup à s 'organiser, et à faire en sorte de ne pas prendre trop de mémoire pour rien.

Lorsque vous créez une variable, vous allez donc devoir **indiquer son type**.

Voici les principaux types de variables existant en langage C :

|  |  |  |
| --- | --- | --- |
| Nom du type | Minimum | Maximum |
| signed char | -127 | 127 |
| int | -32 767 | 32 767 |
| long | -2 147 483 647 | 2 147 483 647 |
| float | -1 x1037 | 1 x1037 |
| double | -1 x1037 | 1 x1037 |

Les valeurs présentées ci-dessus sont les minimums garantis par le langage. En réalité, il est fort probable que vous puissiez stocker des valeurs plus élevées que celles-ci. Cependant, veillez à garder ces valeurs en tête lorsque vous choisissez un type, c'est important.

Notez que tous les types n'ont pas été présentés, seul les principaux ont été conservés.

Les trois premiers types (signed char, int, long) permettent de stocker des nombres entiers : 1, 2, 3, 4…

Les deux derniers (float, double) permettent de stocker des nombres décimaux (appelés nombres flottants) : 13.8, 16.911…

Vous verrez que la plupart du temps on manipule des nombres entiers (tant mieux, parce que c'est plus facile à utiliser).

Attention avec les nombres décimaux ! Votre ordinateur ne connaît pas la virgule, il utilise le point. Vous ne devez donc pas écrire 54,9 mais plutôt 54.9 !

Ce n'est pas tout ! Pour les types entiers (signed char, int, long…), il existe d'autres types dits unsigned (non signés) qui eux ne peuvent stocker que des nombres positifs. Pour les utiliser, il suffit d'écrire le mot unsigned devant le type

Comme vous le voyez, les unsigned sont des types qui ont le défaut de ne pas pouvoir stocker de nombres négatifs, mais l'avantage de pouvoir stocker des nombres deux fois plus grands (signed char s 'arrête à 127, tandis que unsigned char s 'arrête à 255 par exemple).

Vous remarquerez que le type char a été présenté soit avec le mot-clé signed, soit avec le mot-clé unsigned, mais jamais seul. La raison en est toute simple : le type peut être signé ou non signé suivant les machines. Veillez donc à spécifier lequel des deux vous souhaitez utiliser suivant le type de valeur que vous désirez stocker.



Pourquoi avoir créé trois types pour les nombres entiers ? Un seul type aurait été suffisant, non ?

Oui, mais on a créé à l'origine plusieurs types pour économiser de la mémoire. Ainsi, quand on dit à l'ordinateur qu'on a besoin d'une variable de type char, on prend moins d'espace en mémoire que si on avait demandé une variable de type int.

Toutefois, c'était utile surtout à l'époque où la mémoire était limitée. Aujourd'hui, nos ordinateurs ont largement assez de mémoire vive pour que ça ne soit plus vraiment un problème. Il ne sera donc pas utile de se prendre la tête pendant des heures sur le choix d'un type. Si vous ne savez pas si votre variable risque de prendre une grosse valeur, mettez int (ou double pour un flottant).

En résumé, on fera surtout la distinction entre nombres entiers et flottants :

* pour un nombre **entier**, on utilisera le plus souvent int ;
* pour un nombre **flottant**, on utilisera généralement double.

## Déclarer une variable

On y arrive. Maintenant, créez un nouveau projet console que vous appellerez « variables ».

On va voir comment déclarer une variable, c'est-à-dire demander à l'ordinateur la permission d'utiliser un peu de mémoire.

Une déclaration de variable, c'est très simple maintenant que vous savez tout ce qu'il faut. Il suffit dans l'ordre :

1. d'indiquer le type de la variable que l'on veut créer ;
2. d'insérer un espace ;
3. d'indiquer le nom que vous voulez donner à la variable ;
4. et enfin, de ne pas oublier le point-virgule.

Par exemple, si je veux créer ma variable nombreDeVies de type int, je dois taper la ligne suivante :

int nombreDeVies;

Et c'est tout ! Quelques autres exemples stupides pour la forme :

int noteDeMaths;

double sommeArgentRecue;

unsigned int nombreDeLecteursEnTrainDeLireUnNomDeVariableUnPeuLong;

Bon bref, vous avez compris le principe je pense !

Ce qu'on fait là s 'appelle une **déclaration de variable** (un vocabulaire à retenir). Vous devez faire les déclarations de variables au début des fonctions. Comme pour le moment on n'a qu'une seule fonction (la fonction main), vous allez déclarer la variable comme ceci :

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) // Équivalent de int main()

{

int nombreDeVies;

return 0;

}

Si vous lancez le programme ci-dessus, vous constaterez avec stupeur… qu'il ne fait rien.

Quelques explications

Alors, avant que vous ne m'étrangliez en croyant que je vous mène en bateau depuis tout à l'heure, laissez-moi juste dire une chose pour ma défense.

En fait, il se passe des choses, mais vous ne les voyez pas. Lorsque le programme arrive à la ligne de la déclaration de variable, il demande bien gentiment à l'ordinateur s 'il peut utiliser un peu d'espace dans la mémoire vive.

Si tout va bien, l'ordinateur répond « Oui bien sûr, fais comme chez toi ». Généralement, cela se passe sans problème. Le seul souci qu'il pourrait y avoir, c'est qu'il n'y ait plus de place en mémoire…Mais heureusement cela arrive rarement, car pour remplir toute la mémoire rien qu'avec des int il faut vraiment le vouloir !

Soyez donc sans crainte, vos variables devraient normalement être créées sans souci.

Une petite astuce à connaître : si vous avez plusieurs variables du même type à déclarer, inutile de faire une ligne pour chaque variable. Il vous suffit de séparer les différents noms de variables par des virgules sur la même ligne : int nombreDeVies, niveau, ageDuJoueur. Cela créera trois variables int appelées nombreDeVies, niveau et ageDuJoueur.

Et maintenant ?

Maintenant qu'on a créé notre variable, on va pouvoir lui donner une valeur.

## Affecter une valeur à une variable

C'est tout ce qu'il y a de plus bête. Si vous voulez donner une valeur à la variable nombreDeVies, il suffit de procéder comme ceci :

nombreDeVies = 5;

Rien de plus à faire. Vous indiquez le nom de la variable, un signe égal, puis la valeur que vous voulez y mettre.

Ici, on vient de donner la valeur 5 à la variable nombreDeVies.

Notre programme complet ressemble donc à ceci :

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

int nombreDeVies;

nombreDeVies = 5;

return 0;

}

Là encore, rien ne s 'affiche à l'écran, tout se passe dans la mémoire.

Quelque part dans les tréfonds de votre ordinateur, une petite case de mémoire vient de prendre la valeur 5. N'est-ce pas magnifique ?

On peut s 'amuser si on veut à changer la valeur par la suite :

int nombreDeVies;

nombreDeVies = 5;

nombreDeVies = 4;

nombreDeVies = 3;

Dans cet exemple, la variable va prendre d'abord la valeur 5, puis 4, et enfin 3. Comme votre ordinateur est très rapide, tout cela se passe extrêmement vite. Vous n'avez pas le temps de cligner des yeux que votre variable vient de prendre les valeurs 5, 4 et 3… et ça y est, votre programme est fini.

## La valeur d'une nouvelle variable

Voici une question très importante que je veux vous soumettre :

Quand on déclare une variable, quelle valeur a-t-elle au départ ?

En effet, quand l'ordinateur lit cette ligne :

int nombreDeVies;

Il réserve un petit emplacement en mémoire, d'accord. Mais quelle est la valeur de la variable à ce moment-là ? Y a-t-il une valeur par défaut (par exemple 0) ?

Eh bien, accrochez-vous : la réponse est non. Non, non et non, il n'y a pas de valeur par défaut. En fait, l'emplacement est réservé mais la valeur ne change pas. On n'efface pas ce qui se trouve dans la « case mémoire ». Du coup, votre variable prend la valeur qui se trouvait là avant dans la mémoire, et **cette valeur peut être n'importe quoi !**

Si cette zone de la mémoire n'a jamais été modifiée, la valeur est peut-être 0. Mais vous n'en êtes pas sûrs, il pourrait très bien y avoir le nombre 363 ou 18 à la place, c'est-à-dire un reste d'un vieux programme qui est passé par là avant !

Il faut donc faire très attention à ça si on veut éviter des problèmes par la suite. Le mieux est d'initialiser la variable dès qu'on la déclare. En C, c'est tout à fait possible. En gros, ça consiste à combiner la déclaration et l'affectation d'une variable dans la même instruction :

int nombreDeVies = 5;

Ici, la variable nombreDeVies est déclarée et prend tout de suite la valeur 5.

L'avantage, c'est que vous êtes sûrs après que cette variable contient une valeur correcte, et pas du n'importe quoi.

## Les constantes

Il arrive parfois que l'on ait besoin d'utiliser une variable dont on voudrait qu'elle garde la même valeur pendant toute la durée du programme. C'est-à-dire qu'une fois déclarée, vous voudriez que votre variable conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.

Ces variables particulières sont appelées **constantes**, justement parce que leur valeur reste constante.

Pour déclarer une constante, c'est en fait très simple : il faut utiliser le mot **const** juste devant le type quand vous déclarez votre variable. Par ailleurs, il faut obligatoirement lui donner une valeur au moment de sa déclaration comme on vient d'apprendre à le faire. Après, il sera trop tard : vous ne pourrez plus changer la valeur de la constante.

Exemple de déclaration de constante :

const int NOMBRE\_DE\_VIES\_INITIALES = 5;

Ce n'est pas une obligation, mais par convention on écrit les noms des constantes entièrement en majuscules comme je viens de le faire là. Cela nous permet ainsi de distinguer facilement les constantes des variables. Notez qu'on utilise l'underscore \_ à la place de l'espace.

À part ça, une constante s 'utilise comme une variable normale, vous pouvez afficher sa valeur si vous le désirez.

La seule chose qui change, c'est que si vous essayez de modifier la valeur de la constante plus loin dans le programme, le compilateur vous indiquera qu'il y a une erreur avec cette constante.

Les erreurs de compilation sont affichées en bas de l'écran (dans ce que j'appelle la « zone de la mort », vous vous souvenez ?).

Dans un tel cas, le compilateur vous afficherait un mot doux du genre : [Warning] assignment of read-only variable 'NOMBRE\_DE\_VIES\_INITIALES' (traduction : « Triple idiot, pourquoi tu essaies de modifier la valeur d'une constante ? »).

## Afficher le contenu d'une variable

On sait afficher du texte à l'écran avec la fonction printf.

Maintenant, on va voir comment afficher la valeur d'une variable avec cette même fonction.

On utilise en fait printf de la même manière, sauf que l'on rajoute un symbole spécial à l'endroit où l'on veut afficher la valeur de la variable. Par exemple :

printf("Il vous reste %d vies");

Ce « symbole spécial » dont je viens de vous parler est en fait un '% suivi d'une lettre (dans mon exemple, la lettre 'd'). Cette lettre permet d'indiquer ce que l'on doit afficher. 'd' signifie que l'on souhaite afficher un int.

Il existe plusieurs autres possibilités, mais pour des raisons de simplicité on va se contenter de retenir celles-ci :

|  |  |
| --- | --- |
| Format | Type attendu |
| "%d" | int |
| "%ld" | long |
| "%f" | float |
| "%f" | double |

Remarquez que le format utilisé pour afficher un float et un double est identique.

Je vous parlerai des autres symboles en temps voulu. Pour le moment, retenez uniquement ceux-ci

On a presque fini. On a indiqué qu'à un endroit précis on voulait afficher un nombre entier, mais on n'a pas précisé lequel ! Il faut donc indiquer à la fonction printf quelle est la variable dont on veut afficher la valeur.

Pour ce faire, vous devez taper le nom de la variable après les guillemets et après avoir rajouté une virgule, comme ceci :

printf("Il vous reste %d vies", nombreDeVies);

Le %d sera remplacé par la variable indiquée après la virgule, à savoir nombreDeVies.

On se teste ça dans un programme ?

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[])

{

int nombreDeVies = 5; // Au départ, le joueur a 5 vies

printf("Vous avez %d vies\n", nombreDeVies);

printf("\*\*\*\* B A M \*\*\*\*\n"); // Là il se prend un grand coup sur la tête

nombreDeVies = 4; // Il vient de perdre une vie !

printf("Il ne vous reste plus que %d vies maintenant!\n\n", nombreDeVies);

return 0;

}

Ça pourrait presque être un jeu vidéo (il faut juste beaucoup d'imagination).

Ce programme affiche ceci à l'écran :

Vous avez 5 vies

\*\*\*\* B A M \*\*\*\*

Il ne vous reste plus que 4 vies maintenant !

Vous devriez reconnaître ce qui se passe dans votre programme.

1. Au départ le joueur a 5 vies, on affiche ça dans un printf.
2. Ensuite, le joueur prend un coup sur la tête (d'où le BAM).
3. Finalement il n'a plus que 4 vies, on affiche ça aussi avec un printf.

Bref, c'est plutôt simple.

## Afficher plusieurs variables dans un même printf

Il est possible d'afficher la valeur de plusieurs variables dans un seul printf. Il vous suffit pour cela d'indiquer des %d ou des %f là où vous voulez, puis d'indiquer les variables correspondantes dans le même ordre, séparées par des virgules.

Par exemple :

printf("Vous avez %d vies et vous etes au niveau n° %d",nombreDeVies, niveau);

Veillez à bien indiquer vos variables dans le bon ordre. Le premier %d sera remplacé par la première variable (nombreDeVies), et le second %d par la seconde variable (niveau). Si vous vous trompez d'ordre, votre phrase ne voudra plus rien dire.

Allez, un petit test maintenant. Notez que j'enlève les lignes tout en haut (les directives de préprocesseur commençant par un #), je vais supposer que vous les mettez à chaque fois maintenant :

int main(int argc, char \*argv[])

{

int nombreDeVies = 5, niveau = 1;

printf("Vous avez %d vies et vous etes au niveau n° %d\n",

nombreDeVies, niveau);

return 0;

}

Ce qui affichera :

Vous avez 5 vies et vous etes au niveau n° 1

## Récupérer une saisie

Les variables vont en fait commencer à devenir intéressantes maintenant. On va apprendre à demander à l'utilisateur de taper un nombre dans la console. Ce nombre, on va le récupérer et le stocker dans une variable.

Une fois que ça sera fait, on pourra faire tout un tas de choses avec, vous verrez.

Pour demander à l'utilisateur d'entrer quelque chose dans la console, on va utiliser une autre fonction toute prête : **scanf**.

Cette fonction ressemble beaucoup à printf. Vous devez mettre un format pour indiquer ce que l'utilisateur doit entrer (un int, un float,...). Puis vous devez ensuite indiquer le nom de la variable qui va recevoir le nombre.

Voici comment faire par exemple :

int age = 0;

scanf("%d", &age);

On doit mettre le %d entre guillemets.

Par ailleurs, il faut mettre le symbole & devant le nom de la variable qui va recevoir la valeur.

Euh, pourquoi mettre un & devant le nom de la variable ?

Là, il va falloir que vous me fassiez confiance. Si je dois vous expliquer ça tout de suite, on n'est pas sortis de l'auberge, croyez-moi !

Que je vous rassure quand même : je vous expliquerai un peu plus tard ce que signifie ce symbole. Pour le moment, je choisis de ne pas vous l'expliquer pour ne pas vous embrouiller, c'est donc plutôt un service que je vous rends là !

Attention, il y a une petite divergence de format entre printf et scanf ! Pour récupérer un float, c'est le format "%f" qu'il faut utiliser, mais pour le type double c'est le format "%lf".

double poids = 0;

scanf("%lf", &poids);

Revenons à notre programme. Lorsque celui-ci arrive à un scanf, il se met en pause et attend que l'utilisateur entre un nombre.

Ce nombre sera stocké dans la variable age.

Voici un petit programme simple qui demande l'âge de l'utilisateur et qui le lui affiche ensuite :

int main(int argc, char \*argv[])

{

int age = 0; // On initialise la variable à 0

printf("Quel age avez-vous ? ");

scanf("%d", &age); // On demande d'entrer l'âge avec scanf

printf("Ah ! Vous avez donc %d ans !\n\n", age);

return 0;

}

Quel age avez-vous ? 20

Ah ! Vous avez donc 20 ans !

Le programme se met donc en pause après avoir affiché la question « Quel age avez-vous ? ». Le curseur apparaît à l'écran, vous devez taper un nombre entier (votre âge). Tapez ensuite sur « Entrée » pour valider, et le programme continuera à s 'exécuter.

Ici, tout ce qu'il fait après c'est afficher la valeur de la variable age à l'écran (« Ah ! Vous avez donc 20 ans ! »).

Voilà, vous avez compris le principe. Grâce à la fonction scanf, on peut donc commencer à interagir avec l'utilisateur.

Notez que rien ne vous empêche de taper autre chose qu'un nombre entier :

* si vous rentrez un nombre décimal, comme 2.9, il sera automatiquement tronqué, c'est-à-dire que seule la partie entière sera conservée. Dans ce cas, c'est le nombre 2 qui aurait été stocké dans la variable ;
* si vous tapez des lettres au hasard (« éèydf »), la variable ne changera pas de valeur. Ce qui est bien ici, c'est qu'on avait initialisé notre variable à 0 au début. De ce fait, le programme affichera « 0 ans » si ça n'a pas marché. Si on n'avait pas initialisé la variable, le programme aurait pu afficher n'importe quoi !

**En résumé**

* Nos ordinateurs possèdent plusieurs types de mémoire. De la plus rapide à la plus lente : les registres, la mémoire cache, la mémoire vive et le disque dur.
* Pour « retenir » des informations, notre programme a besoin de stocker des données dans la mémoire. Il utilise pour cela la mémoire vive. Les registres et la mémoire cache sont aussi utilisés pour augmenter les performances, mais cela fonctionne automatiquement, nous n'avons pas à nous en préoccuper.
* Dans notre code source, les variables sont des données stockées temporairement en mémoire vive. La valeur de ces données peut changer au cours du programme.
* À l'opposé, on parle de constantes pour des données stockées en mémoire vive. La valeur de ces données ne peut pas changer.
* Il existe plusieurs types de variables, qui occupent plus ou moins d'espace en mémoire. Certains types comme int sont prévus pour stocker des nombres entiers, tandis que d'autres comme double stockent des nombres décimaux.
* La fonction scanf permet de demander à l'utilisateur de saisir un nombre.

## Conclusion

* Un ordinateur n'est en fait qu'une **calculatrice géante** : tout ce qu'il sait faire, ce sont des opérations.
* Les opérations connues par votre ordinateur sont très **basiques** : l'addition, la soustraction, la multiplication, la division et le modulo (il s 'agit du reste de la division).
* Il est possible **d'effectuer des calculs entre des variables**. C'est d'ailleurs ce qu'un ordinateur sait faire de mieux : il le fait bien et vite.
* **L'incrémentation** est l'opération qui consiste à ajouter 1 à une variable. On écrit variable++.
* La **décrémentation** est l'opération inverse : on retire 1 à une variable. On écrit donc variable--.
* Pour augmenter le nombre d'opérations connues par votre ordinateur, il faut charger la **bibliothèque mathématique** (c'est-à-dire #include <math.h>).
* Cette bibliothèque contient des **fonctions mathématiques plus avan**cées, telles que la puissance, la racine carrée, l'arrondi, l'exponentielle, le logarithme, etc.

# Les conditions

## Quelques symboles à connaître

Voici un petit tableau de symboles du langage C à connaître par cœur :

|  |  |
| --- | --- |
| Symbole | Signification |
| == | est égal à |
| > | est supérieur à |
| < | est inférieur à |
| >= | est supérieur ou égal à |
| <= | est inférieur ou égal à |
| != | est différent de |

Faites très attention, il y a bien deux symboles == pour tester l'égalité. Une erreur courante que font les débutants et de ne mettre qu'un symbole =, ce qui n'a pas la même signification en C. Je vous en reparlerai un peu plus bas.

S'il n'y a qu'une instruction entre les accolades, alors celles-ci deviennent facultatives. Je recommande néanmoins de toujours mettre des accolades pour des raisons de clarté

## Les conditions

SI la variable vaut ça,

ALORS fais ceci,

SINON fais cela.

if (age >= 18) // Si l'âge est supérieur ou égal à 18

{

printf ("Vous etes majeur !");

}

else if ( age > 4 ) // Sinon, si l'âge est au moins supérieur à 4

{

printf ("Bon t'es pas trop jeune quand même...");

}

else // Sinon...

{

printf ("Aga gaa aga gaaa"); // Langage bébé, vous pouvez pas comprendre

}

## Test ET

Si on veut faire le test que j'ai mentionné plus haut, il faudra écrire :

if (age > 18 && age < 25)

Les deux symboles && signifient ET. Notre condition se dirait en français : « si l'âge est supérieur à 18 ET si l'âge est inférieur à 25 ».

## Test OU

Pour faire un OU, on utilise les deux signes ||. Je dois avouer que ce signe n'est pas facilement accessible sur nos claviers. Pour le taper sur un clavier AZERTY français, il faudra faire Alt Gr + 6. Sur un clavier belge, il faudra faire Alt Gr + &.

Imaginons pour l'exemple un programme stupide qui décide si une personne a le droit d'ouvrir un compte en banque. C'est bien connu, pour ouvrir un compte en banque il vaut mieux ne pas être trop jeune (on va dire arbitrairement qu'il faut avoir au moins 30 ans) ou bien avoir beaucoup d'argent (parce que là, même à 10 ans on vous acceptera à bras ouverts !). Notre test pour savoir

si le client a le droit d'ouvrir un compte en banque pourrait être :

if (age > 30 || argent > 100000)

{

printf("Bienvenue chez PicsouBanque !");

}

else

{

printf("Hors de ma vue, miserable !");

}

Ce test n'est valide que si la personne a plus de 30 ans ou si elle possède plus de 100 000 euros !

## Test NON

Le dernier symbole qu'il nous reste à tester est le point d'exclamation. En informatique, le point d'exclamation signifie « non ».

Vous devez mettre ce signe avant votre condition pour dire « si cela n'est pas vrai » :

if (!(age < 18))

Cela pourrait se traduire par « si la personne n'est pas mineure ». Si on avait enlevé le ! devant, cela aurait signifié l'inverse : « si la personne est mineure ».

## La condition switch

La condition if... else que l'on vient de voir est le type de condition le plus souvent utilisé.

En fait, il n'y a pas 36 façons de faire une condition en C. Le if... else permet de gérer tous les cas.

Toutefois, le if... else peut s 'avérer quelque peu… répétitif. Prenons cet exemple :

switch (age)

{

case 2:

printf("Salut bebe !");

break;

case 6:

printf("Salut gamin !");

break;

case 12:

printf("Salut jeune !");

break;

case 16:

printf("Salut ado !");

break;

case 18:

printf("Salut adulte !");

break;

case 68:

printf("Salut papy !");

break;

default:

printf("Je n'ai aucune phrase de prête pour ton âge ");

break;

}

Vous devez mettre une instruction break; obligatoirement à la fin de chaque cas. Si vous ne le faites pas, alors l'ordinateur ira lire les instructions en dessous censées être réservées aux autres cas ! L'instruction break; commande en fait à l'ordinateur de « sortir » des accolades.

## Les ternaires : des conditions condensées

Il existe une troisième façon de faire des conditions, plus rare.

On appelle cela des express ions ternaires.

Concrètement, c'est comme un if... else, sauf qu'on fait tout tenir sur une seule ligne !

Comme un exemple vaut mieux qu'un long discours, je vais vous donner deux fois la même condition : la première avec un if... else, et la seconde, identique, mais sous forme d'une expression ternaire.

**Une condition if... else bien connue**

Supposons qu'on ait une variable booléenne majeur qui vaut vrai (1) si on est majeur, et faux (0) si on est mineur.

On veut changer la valeur de la variable age en fonction du booléen, pour mettre "18" si on est majeur, "17" si on est mineur.

C'est un exemple complètement stupide je suis d'accord, mais ça me permet de vous montrer comment on peut se servir des expressions ternaires.

if (majeur)

age = 18;

else

age = 17;

**La même condition en ternaire**

Voici un code qui fait exactement la même chose que le code précédent, mais écrit cette fois sous forme ternaire :

age = (majeur) ? 18 : 17;

Les ternaires permettent, sur une seule ligne, de changer la valeur d'une variable en fonction d'une condition. Ici la condition est tout simplement majeur, mais ça pourrait être n'importe quelle condition plus longue bien entendu. Un autre exemple ?

autorisation = (age >= 18) ? 1 : 0;

Le point d'interrogation permet de dire « est-ce que tu es majeur ? ». Si oui, alors on met la valeur 18 dans age. Sinon (le deux-points : signifie else ici), on met la valeur 17.

## Conclusion

* Les conditions sont à la base de tous les programmes. C'est un moyen pour l'ordinateur de prendre une décision en fonction de la valeur d'une variable.
* Les mots-clés if, else if, else signifient respectivement « si », « sinon si », « sinon ». On peut écrire autant de else if que l'on désire.
* Un booléen est une variable qui peut avoir deux états : vrai (1) ou faux (0) (toute valeur différente de 0 est en fait considérée comme « vraie »). On utilise des int pour stocker des booléens car ce ne sont en fait rien d'autre que des nombres.
* Le switch est une alternative au if quand il s 'agit d'analyser la valeur d'une variable. Il permet de rendre un code source plus clair si vous vous apprêtiez à tester de nombreux cas. Si vous utilisez de nombreux else if c'est en général le signe qu'un switch serait plus adapté pour rendre le code source plus lisible.
* Les ternaires sont des conditions très concises qui permettent d'affecter rapidement une valeur à une variable en fonction du résultat d'un test. On les utilise avec parcimonie car le code source a tendance à devenir moins lisible avec elles.

# Les boucles

* while
* do… while
* for

Dans tous les cas, le schéma est le même

Les boucles sont des structures qui nous permettent de répéter une série d'instructions plusieurs fois.

Il existe plusieurs types de boucles : while, do… while et for. Certaines sont plus adaptées que d'autres selon les cas.

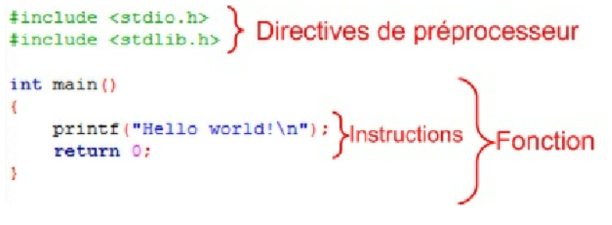
La boucle for est probablement celle qu'on utilise le plus dans la pratique. On y fait très souvent des incrémentations ou des décrémentations de variables.

Si on connait le nombre d’itérations, on utilise le FOR,

sinon, s’il faut au moins faire une fois les instructions, on utilise Do while,

sinon While

# Les fonctions



## Schéma d'une fonction

Vous avez déjà eu un aperçu de la façon dont est faite une fonction avec la fonction main.

Cependant pour bien que vous compreniez il va falloir que je vous montre quand même comment on construit une fonction.

Le code suivant représente une fonction schématiquement. C'est un modèle à connaître :

type nomFonction(parametres)

{

// Insérez vos instructions ici

}

Vous reconnaissez la forme de la fonction main.

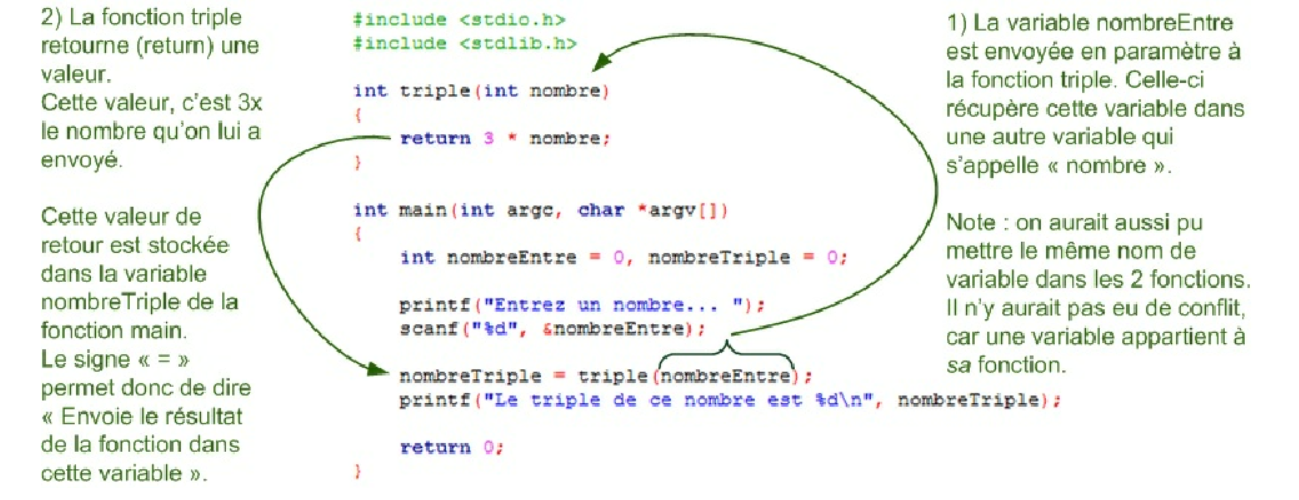
Voici ce qu'il faut savoir sur ce schéma.

* type (correspond à la sortie) : c'est le type de la fonction. Comme les variables, les fonctions ont un type. Ce type dépend du résultat que la fonction renvoie : si la fonction renvoie un nombre décimal, vous mettrez sûrement double, si elle renvoie un entier vous mettrez int ou long par exemple. Mais il est aussi possible de créer des fonctions qui ne renvoient rien !

Il y a donc deux sortes de fonctions :

* + les fonctions qui renvoient une valeur : on leur met un des types que l'on connaît (char, int, double, etc.) ;
  + les fonctions qui ne renvoient pas de valeur : on leur met un type spécial void (qui signifie « vide »).
* nomFonction : c'est le nom de votre fonction. Vous pouvez appeler votre fonction comme vous voulez, du temps que vous respectez les mêmes règles que pour les variables (pas d'accents, pas d'espaces, etc.).
* parametres (correspond à l'entrée) : entre parenthèses , vous pouvez envoyer des paramètres à la fonction. Ce sont des valeurs avec lesquelles la fonction va travailler.

Vous pouvez envoyer autant de paramètres que vous le voulez. Vous pouvez aussi n'envoyer aucun paramètre à la fonction, mais ça se fait plus rarement.



## Le prototype pour annoncer une fonction

Jusqu'ici, je vous ai demandé de placer votre fonction avant la fonction main.

Pourquoi ?

Parce que l'ordre a une réelle importance ici : si vous mettez votre fonction avant le main dans votre code source, votre ordinateur l'aura lue et la connaîtra. Lorsque vous ferez un appel à la fonction dans le main, l'ordinateur connaîtra la fonction et saura où aller la chercher.

En revanche, si vous mettez votre fonction après le main, ça ne marchera pas car l'ordinateur ne connaîtra pas encore la fonction. Essayez, vous verrez !

Mais… c'est un peu mal fait, non ?

Tout à fait d'accord avec vous ! Mais rassurez-vous, les programmeurs s 'en sont rendu compte avant vous et ont prévu le coup.

Grâce à ce que je vais vous apprendre maintenant, vous pourrez positionner vos fonctions dans n'importe quel ordre dans le code source. C'est mieux de ne pas avoir à s 'en soucier, croyez-moi.

Nous allons « annoncer » nos fonctions à l'ordinateur en écrivant ce qu'on appelle des prototypes. Ne soyez pas intimidés par ce nom high-tech, ça cache en fait quelque chose de très simple.

Copiez la première ligne de la fonction (double aireRectangle…) tout en haut de votre fichier source (jus te après les #include).

Rajoutez un point-virgule à la fin de cette nouvelle ligne.

Et voilà ! Maintenant, vous pouvez placer votre fonction aireRectangle après la fonction main si vous le voulez !

N'oubliez JAMAIS de mettre un point-virgule à la fin d'un prototype. C'est ce qui permet à l'ordinateur de différencier un prototype du véritable début d'une fonction.

Si vous ne le faites pas, vous risquez d'avoir des erreurs incompréhensibles lors de la compilation.

# A l’assaut des pointeurs

## Adresse et valeur

* age : désigne la valeur de la variable ;
* &age : désigne l'adresse de la variable

## Utiliser des pointeurs

Jusqu'ici, nous avons uniquement créé des variables faites pour contenir des nombres. Maintenant, nous allons apprendre à créer des variables faites pour contenir des adresses : ce sont justement ce qu'on appelle des pointeurs.

## Créer un pointeur

Pour créer une variable de type pointeur, on doit rajouter le symbole \* devant le nom de la variable.

int \*monPointeur;

## Conclusion

Voici ce qu'il faut avoir compris et ce qu'il faut retenir pour la suite de ce chapitre :

* sur une variable, comme la variable age : (déclaration : int age)
  + age signifie : « Je veux la valeur de la variable age »,
  + &age signifie : « Je veux l'adresse à laquelle se trouve la variable age » ;
* sur un pointeur, comme pointeurSurAge : (déclaration : int \*pointeurSurAge)
  + pointeurSurAge signifie : « Je veux la valeur de pointeurSurAge » (cette valeur étant une adresse),
  + \*pointeurSurAge signifie : « Je veux la valeur de la variable qui se trouve à l'adresse contenue dans pointeurSurAge ».

void triplePointeur(int \*pointeurSurNombre);

int main(int argc, char \*argv[])

{

int nombre = 5;

triplePointeur(&nombre); // On envoie l'adresse de nombre à la fonction

printf("%d", nombre); // On affiche la variable nombre. La fonction a directement modifié la valeur de la variable car elle connaissait son adresse

return 0;

}

void triplePointeur(int \*pointeurSurNombre)

{

\*pointeurSurNombre \*= 3; // On multiplie par 3 la valeur de nombre

}

En résumé

* Chaque variable est stockée à une adresse précise en mémoire.
* Les pointeurs sont semblables aux variables, à ceci près qu'au lieu de stocker un nombre ils stockent l'adresse à laquelle se trouve une variable en mémoire.
* Si on place un symbole & devant un nom de variable, on obtient son adresse au lieu de sa valeur (ex. : &age).
* Si on place un symbole \* devant un nom de pointeur, on obtient la valeur de la variable stockée à l'adresse indiquée par le pointeur.
* Les pointeurs constituent une notion essentielle du langage C, mais néanmoins un peu complexe au début. Il faut prendre le temps de bien comprendre comment ils fonctionnent car beaucoup d'autres notions sont basées dessus.

# Les tableaux

## Définir un tableau

int tableau[4];

Attention : un tableau commence à l'indice n° 0 ! Notre tableau de 4 int a donc les indices 0, 1, 2 et 3. Il n'y a pas d'indice 4 dans un tableau de 4 cases ! C'est une source d'erreurs très courantes, souvenez-vous-en.

## Initialiser un tableau

int tableau[4] = {10, 23}; // Valeurs insérées : 10, 23, 0, 0

## Passage de tableaux à une fonction

Vous aurez à coup sûr souvent besoin d'afficher tout le contenu de votre tableau. Pourquoi ne pas écrire une fonction qui fait ça ? Ça va nous permettre de découvrir comment on envoie un tableau à une fonction (ce qui m'arrange).

Il va falloir envoyer deux informations à la fonction : le tableau (enfin, l'adresse du tableau) et aussi et surtout sa taille !

En effet, notre fonction doit être capable d'initialiser un tableau de n'importe quelle taille. Or, dans votre fonction, vous ne connaissez pas la taille de votre tableau. C'est pour cela qu'il faut envoyer en plus une variable que vous appellerez par exemple tailleTableau.

Comme je vous l'ai dit, tableau peut être considéré comme un pointeur. On peut donc l'envoyer à la fonction comme on l'aurait fait avec un vulgaire pointeur :

// Prototype de la fonction d'affichage

void affiche(int \*tableau, int tailleTableau);

int main(int argc, char \*argv[])

{

int tableau[4] = {10, 15, 3};

// On affiche le contenu du tableau affiche(tableau, 4);

return 0;

}

void affiche(int \*tableau, int tailleTableau)

{

int i;

for (i = 0 ; i < tailleTableau ; i++)

{

printf("%d\n", tableau[i]);

}

}

Notez qu'il existe une autre façon d'indiquer que la fonction reçoit un tableau. Plutôt que d'indiquer que la fonction attend un int \*tableau, mettez ceci :

void affiche(int tableau[], int tailleTableau)

## En résumé

* Les tableaux sont des ensembles de variables du même type stockées côte à côte en mémoire.
* La taille d'un tableau doit être déterminée avant la compilation, elle ne peut pas dépendre d'une variable.
* Chaque case d'un tableau de type int contient une variable de type int.
* Les cases sont numérotées via des indices commençant à 0 : tableau[0], tableau[1], tableau[2], etc.