

Définition

Une valeur, c'est une information courte qui est stockée dans la mémoire vive. Elle est "variable" car elle peut changer durant le déroulement du programme. Tous nos programmes sont remplis de variables.

En C, une variable de constitue de **deux choses** :

- Une **valeur**, c'est à dire le nombre qu'elle stocke ;
- Un **nom**, qui permet de la reconnaître et de l'appeler. Son nom permet au compilateur de la reconnaître et de la convertir depuis son "adresse" (chemin d'accès.)

De l'importance de nommer ses variables

Il faut donc nommer chaque variable. Il est pertinent de nommer sa variable d'une façon facile à reconnaître - par exemple, dans un jeu, la variable qui se souvient du nombre de vie devrait sûrement s'appeler `>| nb_vies` ou `>| count_life`. Cela permet d'augmenter la lisibilité du code (par un humain...) et facilite le travail à plusieurs.

Il y a cependant quelques contraintes à retenir :

- la variable se déclare en minuscule, majuscules ou chiffres uniquement, sans caractères spéciaux ;
- elle commence **obligatoirement** par une lettre
- on utilise pas d'espaces, mais on peut le remplacer par un `>| _`. C'est le seul caractère non alphabétique autorisé.

⚠ Le C est sensible à la casse, il faut donc être vigilant sur les majuscules et minuscules et prendre garde à harmoniser ses variables.

Les différents types de variables

L'ordinateur **ne sait traiter que des nombres**. Il faut donc lui parler en chiffres, et lui indiquer de quel type ils sont.

Pour rappel, les nombres peuvent être :

- Entiers et positifs ou négatifs (5, -87)
- Décimaux et positifs ou négatifs (75.476, -13.9)

Quand on déclare la valeur, il faudra préciser à la machine de quel type est le nombre.

Voici les types de variables existant en C, divisées en deux catégories :

- Pour stocker des nombres entiers : `>| signed char`, `>| int`, `>| long`
- Pour stocker des nombres décimaux, aussi appellés **nombre flottants** : `>| float`, `>| double`

⚠ Pour déclarer des décimales, il faut utiliser le point et non pas la virgule ; on écrit donc `>| 53.4` et pas `>| 53,4`.

Chaque type possède une fourchette maximum et minimum stockable par le langage. En voilà ci-après un tableau récapitulatif :

Nom du type	Minimum	Maximum
<code>signed char</code>	-127	127
<code>int</code>	-32 767	32 767
<code>long</code>	-2 147 483 647	2 147 483 647
<code>float</code>	-1×10^{37}	1×10^{37}
<code>double</code>	-1×10^{37}	1×10^{37}

Si on utilise que des nombres entier, on peut ajouter `>| unsigned` devant certains types. Ils ont le défaut de ne pas pouvoir stocker de nombres négatifs, mais ils permettent de stocker des nombres bien plus élevés, comme on peut voir sur le tableau suivant :

<code>unsigned char</code>	0 à 255
<code>unsigned int</code>	0 à 65 535
<code>unsigned long</code>	0 à 4 294 967 295

⚠ Le type `char` ne se présente jamais seul, il doit toujours être spécifié `signed` ou `unsigned`. Cela car il peut être signé ou non selon la machine ; il faut donc toujours préciser lequel on veut, en fonction du type de valeur qu'on veut utiliser (entier ou flottant)

Le saviez-vous ?

Il existe autant de types différents car à l'origine, il était important d'économiser la RAM. Une variable `char` (max 127) prend moins de place à stocker qu'une `int` (max 32767).

Aujourd'hui, ça n'a plus tellement d'importance ; on peut utiliser `int` ou `double` sans trop réfléchir dans la plupart des cas.



En bref :

→ On distingue principalement **nombres entiers** avec `>| int` de **nombres flottants** avec `>| double`

Déclarer une variable

→ On crée une valeur, et ce faisant on crée son petit espace de mémoire associée, qui sera rempli avec la valeur :

```

#include <stdio.h> // On inclut les bibliothèques pertinentes
#include <stdlib.h>

int main(int argc, char *argv[])
/* Équivalent de int main() - on déclare int argc et chat
* argv[] pour permettre au programme de recevoir des commandes inline.*/
{
    int nombre_de_vies; // on crée la variable "Nombre de vies"

    return 0; // commande de conclusion
}

```

→ Ensuite, on assigne une valeur à la fonction. Pour se faire, deux techniques existent :

1. On la déclare après la fonction, en respecifiant la fonction à laquelle on attribue la valeur :

```

#include <stdio.h> // On inclut les bibliothèques pertinentes
#include <stdlib.h>

int main(int argc, char *argv[])
/* Équivalent de int main() - on déclare int argc et chat
* argv[] pour permettre au programme de recevoir des commandes inline.*/
{
    int nombre_de_vies; // on crée la variable "Nombre de vies" et
    on lui assigne la valeur "5"
    nombre_de_vies = 5; // on attribue la valeur "5" à la fonction
    "nombre_de_vies"
    return 0; // commande de conclusion
}

```

⚠ Il n'y a pas de valeur par défaut pour les fonctions. En utilisant cette méthode, on risque donc de voir notre emplacement de mémoire remplie par une valeur complètement aléatoire. Pour éviter les soucis, on privilégie la méthode 2 :

2. On déclare immédiatement la valeur, en même temps que la fonction. Ca permet de s'assurer l'emplacement de mémoire dédié à la variable stocke la bonne information, et pas une vieille information d'un autre programme random :

```

#include <stdio.h> // On inclut les bibliothèques pertinentes
#include <stdlib.h>

int main(int argc, char *argv[])
/* Équivalent de int main() - on déclare int argc et chat
* argv[] pour permettre au programme de recevoir des commandes inline.*/
{
    int nombre_de_vies = 5; // on crée la variable "Nombre de vies" et on lui assigne la valeur "5"
    return 0; // commande de conclusion
}

```

Et voilà ! On a déclaré une variable. Celle ci peut être modifiée, durant l'exécution du programme ou en dehors de celle-ci.

Par exemple, si on veut que la valeur initiale de la fonction `>| int nombre_de_vies` soit 5, puis qu'elle se réduise jusqu'à arriver par exemple à 2, on peut dire :

```

int main(int argc, char *argv[])
/* Équivalent de int main() - on déclare int argc et chat
* argv[] pour permettre au programme de recevoir des commandes inline.*/
{
    int nombre_de_vies; // on crée la variable "Nombre de vies" et7
    on lui assigne la valeur "5"
    nombre_de_vies = 4; // la valeur de nombre_de_vies passe à 4
    nombre_de_vies = 3;
    nombre_de_vies = 2;
    return 0; // commande de conclusion
}

```

Déclarer une constante

La constance est déclarée `>| const`.

Par opposition à la variable, la valeur de la constante **ne peut pas être modifiée**. Elle est déclarée une fois lors de sa création, puis elle reste, par définition, **constante**.

Si on voulait, par exemple, déterminer que le nombre de vie maximales autorisées dans un jeu est 5, on pourrait déclarer :

```
>| const int NOMBRE_DE_VIES_MAX = 5;
```

Ainsi, le joueur ne pourra jamais avoir plus de 5 vies, même s'il parvient à obtenir le code du programme et qu'il tente de le modifier.

Afficher le contenu d'une variable

On a déjà utilisé `>| printf` pour afficher du texte dans la console. On peut utiliser cette fonction pour afficher la valeur d'une variable ; pour se faire, on utilise différents symboles en fonction de la fonction qu'on appelle.

Format	Type attendu
"%d"	int
"%u"	unsigned int
"%ld"	long
"%f"	float
"%f"	double

En voici un résumé (non exhaustif) :

En application, ça ressemble à ceci :

```

int main(int argc, char *argv[])
{
    int nombre_de_vies=5; // on crée la variable "Nombre de vies" et on lui assigne la valeur "5"

    printf ("Vous avez %d vies\n\n", nombre_de_vies); // On indique à la machine de récupérer la valeur (%) de
la int (d), nombre_de_vies
    printf ("GRAOUUUU !! Un orc vous transperce avec son épée !!!\n\n"); // Aïe ! Il perd une vie !
    nombre_de_vies = 4; // on déclare le nouveau nombre de vies
    printf ("Vous n'avez plus que %d vies :\n", nombre_de_vies); // On indique à la machine de récupérer la
valeur (%) de la int (d), nombre_de_vies
    return 0; // commande de conclusion
}

```

On peut également indiquer au programme de chercher les valeurs de plusieurs variables :

```

int main(int argc, char *argv[])
{
    int nombre_de_vies = 5, niveau = 1; // on crée la variable "Nombre de vies" et la variable Niveau et on
leur assigne leurs valeurs.
    printf ("Vous avez %d vies et vous êtes niveau %d.\n\n", nombre_de_vies, niveau); // On indique le nombre
de vies et le niveau du joueur. Attention à bien appeler les variables dans le bon ordre !
    printf ("GRAOUUUU !! Un orc vous transperce avec son épée !!!\n\n"); // Aïe ! Il perd une vie !
    nombre_de_vies = 4; // on déclare le nouveau nombre de vies
    niveau = 2; // on déclare le nouveau niveau
    printf ("Vous n'avez plus que %d vies, mais vous êtes passé niveau %d ! \n", nombre_de_vies, niveau); // Le
joueur n'a plus que 4 vies, mais il a gagné un niveau !
    return 0; // commande de conclusion
}

```

Et voilà le résultat :

```

[Running] cd "c:\Users\coral\Desktop\School\Code\" && gcc variables.c -o variables && "c:\Users\coral\Desktop\School\Code\"variables
Vous avez 1 vies et vous êtes niveau 5.

GRAOUUUU !! Un orc vous transperce avec son épée !!!

Vous n'avez plus que 4 vies, mais vous êtes passé niveau 2 !

[Done] exited with code=0 in 0.179 seconds

```

Récupérer une saisie

On va maintenant apprendre à la machine à récupérer la saisie de l'utilisateur ; c'est à dire qu'on va demander à l'utilisateur de saisir une valeur dans la console en réponse au programme, et indiquer au programme de récupérer cette saisie pour la stocker.

Lorsque le programme arrive à un `>| scanf`, il se met en pause et il attend que l'utilisateur entre un nombre. Celui ci sera stocké dans la variable spécifiée.

Pour ce faire, on utilise `>| scanf`, qui ressemble à `>| printf`. A bien noter que :

- On doit indiquer le format dans pour indiquer si l'utilisateur doit entrer une valeur entière ou décimale (`>| int` ou `>| float`)
- On doit ensuite indiquer la variable qui reçoit la valeur.

Exemple :

```

int age = 0;
scanf("%d", &age);

```

- ⚠ Points de vigilance :
- On note `%d` entre `""`
 - On ajoute `&` devant la variable qui accueille la valeur.
 - On ne récupère pas les valeurs de float et double de la même façon : pour le premier, il faudra utiliser %f, pour le second, %lf.

Compiler et exécuter avec une variable > | scanf

L'utilisateur doit utiliser la **console** ou le **terminal**

```
gcc dungeon_crawler.c -o dungeon_crawler
```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int groupe = 0, PV = 20, niveau = 1, orc = 0; // déclare le nombre de membres dans le groupe

    printf ("Vous entrez dans la tour du Mage Noir. Combien d'aventuriers y a-t-il dans votre groupe ?\n\n");
    scanf ("%d", &groupe); // on demande à l'utilisateur de saisir le nombre d'aventuriers dans son groupe.
    printf ("\n Les %d aventuriers de niveau %d s'enfoncent dans le couloir sombre et humide...\n\n", groupe,
niveau);
    printf ("Plic, plic, plic... les %d aventuriers entendent le bruit de gouttes tombant d'un plafond, quelque
part dans le donjon.\n\n");
    printf ("ROAR !!! Un rugissement assourdit le groupe. Des orcs !!! Combien en voyez-vous ?\n\n");
    scanf ("%d", &orc);
    printf ("\n Les %d orcs vous foncent dessus, haches sorties !!!\n", orc);

    getchar(); // permet d'afficher les messages d'erreur.

    return 0;
}
```

En résumé :

- Nos ordinateurs possèdent plusieurs types de mémoire ; pour fonctionner, notre programme stocke les données dans la **mémoire vive** .
- Les données de nos **variables** sont stockées temporairement dans la mémoire vive. Leurs valeurs peuvent changer au cours de l'exécution du programme.
- A l'inverse, les données de nos **constantes** ne peuvent pas être modifiées. Elles sont tout de même stockées dans la mémoire vive (?)
- Il existe différents types de variables, qui occupent plus ou moins d'espace mémoire. Elles ont différents usages : `int` permet de stocker des nombres entiers, et d'autres comme `double` stockent des nombres décimaux.
- On utilise `printf` pour afficher des informations à l'écran, et `scanf` pour demander à l'utilisateur de saisir une information dans la console.