

*à bien
retravailler!*

Variables

Temps d'étude conseillé : 4h30.

Ce nouveau TD vous permet de rentrer dans le vif du sujet, à savoir la programmation en C. Pour le moment, rien de vraiment excitant malheureusement mais ne vous inquiétez pas, les choses sérieuses vont commencer très bientôt !

1 Présentation du langage

Lorsque nous évoquons un langage¹, nous parlons naturellement d'un *langage de programmation*. Dans ce contexte, le Larousse nous donne la définition suivante pour le langage.

Définition 1 (Langage). Un *langage* est un ensemble de caractères, de symboles et de règles qui permettent de les assembler, utilisé pour donner des instructions à un ordinateur.

1.1 Généralités

Chaque langage de programmation possède des particularités qui en font un outil particulièrement adapté à une tâche plutôt qu'une autre. Le C possède les caractéristiques suivantes :

- impératif : la réalisation d'une instruction modifie directement l'état de la mémoire ;
- compilé : l'humain écrit un code C qui est traduit par un compilateur pour que la machine comprenne ce qu'il faut faire.

Sans rentrer dans les détails, sachez que chaque langage est créé pour répondre à un besoin spécifique. Par exemple, le C a été conçu pour donner des instructions directes à la machine, tout en conservant une compréhension simple pour un humain (au contraire de l'assembleur). Les codeurs n'ayant pas besoin d'instructions proches de la machine se tourneront par exemple vers des langages objets tels que le C++ ou le Java. Cette façon de rédiger des programmes s'appelle un *paradigme* de programmation.

Exemple 1. La programmation impérative, telle que le C, est similaire à une recette de cuisine. Une fois une instruction réalisée, on passe à la suivante. Elle est en opposition avec la programmation logique (implémentée par le Prolog) ou fonctionnelle (retrouvée dans le Lisp). Dans langages plus *haut niveau* (c'est à dire moins proches de la machine) sont retrouvés au travers de la programmation objet (Java, C++) ou Web (HTML, PHP) par exemple. ▲

En plus d'être impératif, le C est un langage *compilé*, c'est à dire qu'il nécessite d'être traduit entièrement en un exécutable que la machine pourra alors lancer. On parle alors de la réalisation d'une application. Cette mise en œuvre est en opposition avec les langages

1. Notez bien que l'orthographe anglaise de ce mot est particulièrement proche : *language*

interprétés tels que Bash (utilisé dans le terminal Linux que vous utilisez sous Ubuntu), Python, Ruby ou SQL. Dans ces derniers, chaque instruction est traduite *à la volée* pour être immédiatement exécutée. Ils sont en particulier utilisés pour l'administration d'un système d'information, même si le Python est particulièrement polyvalent.

1.2 Code minimal

Trêve de bavardages, regardons maintenant à quoi ressemble le plus petit programme C du monde².

```
void main()
{

}
```

On appelle ce bout de code le *point d'entrée du programme*. Tout programme C se doit de contenir au minimum ces quelques caractères. Ils permettent de définir l'emplacement de la première instruction. En effet, la lecture se fait ligne par ligne, de haut en bas. Nous le verrons plus tard dans le semestre, mais l'exécution d'un programme ne commence pas forcément en début de fichier. Il est également possible de trouver plusieurs fichiers C formant le programme complet. Il est donc nécessaire de définir ce point d'entrée. Ces quelques caractères doivent maintenant être rédigés dans un fichier avant de les compiler.

Un peu de pratique. Ouvrez un éditeur de texte tel que Gedit et rédigez ces quelques lignes dans un nouveau fichier. Sauvegardez-le sous le nom *prog.c*. ▲

L'extension *.c* permet à l'utilisateur de comprendre qu'il s'agit d'un *code source* dans lequel sont rédigées des *instructions* en vue de le *compiler* puis de lancer l'*exécutable* produit. Voyons dans un premier temps la compilation.

1.3 Compilation

Définition 2 (Compilateur). Un *compilateur* est un logiciel permettant de traduire un *code source* en *code machine* que cette dernière saura exécuter.

Le compilateur n'est donc pas relié directement au langage puisqu'il s'agit d'un logiciel. Celui que nous utiliserons au travers de ce module est GCC, le compilateur³ libre proposé par GNU. Sachez qu'il en existe d'autres, mais GCC est l'un des plus utilisés et l'un des plus fiables, d'où la raison de l'utiliser dans ce cours. Il est donc temps de l'utiliser afin de produire votre premier exécutable.

Un peu de pratique. Dans votre terminal, placez-vous à l'endroit où vous avez sauvegardé votre code source.

- 1) Saisissez la commande suivante sans tenir compte des avertissements qui s'affichent :
`gcc prog.c`

2. En réalité, il s'agit du plus petit code C en comptant uniquement les caractères affichables, mais on n'est pas à ça près !

3. En pratique, GCC est une suite d'outils de compilation, mais par habitude de langage, on conservera le nom de compilateur.

- 2) À l'aide de la commande `ls`, vérifiez qu'un exécutable s'est bien créé. Quel est son nom ? Si ce n'est pas le cas, vérifiez que vous n'avez pas fait d'erreur lors de la copie du fichier C.

Si tout s'est bien passé, vous venez de voir apparaître un fichier nommé *a.out*. Il s'agit de l'exécutable produit par votre code source nommé *prog.c* et compilé en binaire avec GCC à l'aide la commande `gcc prog.c`. L'usage de GCC est donc relativement simple. Il est maintenant temps de lancer votre exécutable.⁴

Tout comme les commandes du terminal (`ls`, `cd`...) ou des programmes plus avancés (`gcc`), le nom de l'exécutable suffit à le lancer⁵. Notre objectif est donc d'exécuter le programme nommé *a.out*. Notre problème consiste à indiquer à l'ordinateur **quel** *a.out* nous souhaitons lancer. Notons que le problème se pose aussi bien pour les executables que vous avez créés que pour les commandes disponibles sur le système. Sans chemin particulier (par exemple `ls`), l'OS va chercher l'exécutable demandé dans quelques dossiers bien spécifiques (par exemple `/home/bin`). Notre exécutable *a.out* étant dans un répertoire du dossier personnel, il faut le signaler à l'OS.

Notation 1 (Dossier courant). Le dossier `.` représente le dossier courant. Pour rappel, le dossier `..` représente le dossier parent.

Un peu de pratique. Assurez-vous d'être placé dans le répertoire contenant l'exécutable fraîchement créé.

- 1) Quel message d'erreur obtenez-vous si vous lancez la commande *a.out* ?
- 2) Sachant que `.` représente votre dossier courant, lancez votre exécutable correctement à l'aide de son chemin relatif.
- 3) Que fait votre programme ?⁶
- 4) Déplacez-vous dans le dossier parent puis lancez à nouveau l'exécutable depuis ce dossier, toujours à l'aide de son chemin relatif.

Astuce. Si le nom *a.out* ne vous plaît pas particulièrement, utilisez l'option `-o` de `gcc` suivie de votre nouveau nom d'exécutable pour le changer.

Terminons cette section par une option qui n'en sera pas une pour vous : l'option `-Wall`. Elle permet au compilateur d'afficher tous les messages d'avertissement. Ces derniers préviennent que le comportement de votre programme risque d'être particulièrement inattendu.

En supposant que mon fichier C se nomme *prog.c* et que je veuille nommer mon exécutable *test.bin*, je réalise la compilation de mes sources à l'aide de la commande :

```
gcc -Wall prog.c -o test.bin
```

4. Pas par la fenêtre, évidemment ! LOL, MDR, PTDR, JPP!!!

5. Voir la note précédente (comique de répétition).


6. Ne soyez pas déçu du résultat, tout le monde est passé par là !

1.4 Un code moins minimal

Vous avez certainement été déçus de l'exécution de votre code minimal, et nous vous comprenons. Il est donc temps de réaliser un code moins minimal et de l'étudier dans le détail. Avant cela, reprenons une dernière fois votre code et passons-le à nouveau au compilateur afin de voir ce que ce dernier en pense.

```
bodin@TeamIbijau:~/Documents$gcc -Wall prog.c
prog.c:1:6: warning: return type of 'main' is not 'int' [-Wmain]
void main()
    ~
bodin@TeamIbijau:~/Documents$
```

Un peu de pratique. Analysez la capture d'écran ci-dessus pour répondre aux questions suivantes.

- 1) Quel est le nom de l'utilisateur ?
- 2) Quel est le nom de la machine ?
- 3) Dans quel dossier l'utilisateur est-il situé ?
- 4) Quel fichier est-il en train de compiler ? 

La première ligne en réponse à notre sollicitation à GCC nous indique un avertissement sur la colonne 6 de la ligne 1 du fichier *prog.c*. Le compilateur indique de modifier le type de retour de la fonction `main()` qui doit être `int` et non `void`. Sans trop chercher à comprendre, il semble que nous pouvons très certainement lui faire plaisir en remplaçant le mot `void` par `int`. Sauvegardez le code et compilez à nouveau le fichier. Observons le résultat.

```
bodin@TeamIbijau:~/Documents$gcc -Wall prog.c
bodin@TeamIbijau:~/Documents$
```

Plus rien ne s'affiche, comme le dit le proverbe : pas de nouvelles, bonnes nouvelles. Le compilateur est donc content, il est temps d'exécuter à nouveau votre programme. Pas de surprise de ce côté, il est toujours aussi triste. Nous allons donc ajouter quelques lignes de code afin de réaliser quelque chose de plus visuel : un « Hello World » customisé. Pour cela, remplacez votre code par le suivant.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf ("Hello Potoo !\n");
6      return 0;
7  }
```

La première ligne est une *instruction pré-processeur*. Elle se repère grâce au caractère dièse⁷ débutant la ligne. Elle est interprétée par le *pré-compilateur* et permet de faire appel à un ensemble d'outils présents dans la *bibliothèque* `stdio.h`. Nous verrons d'autres instructions pré-processeur au fil de ce cours.

7. Ou croisillon, même si la mode semble être de dire hashtag...

Vous reconnaissez en ligne 3 le point d'entrée du programme qui contient maintenant deux instructions. Notez que ces dernières, en lignes 5 et 6, sont décalés sur la droite. Cela est fait par convention, car une accolade ouvrante est présente sur la ligne 4. La ligne 5 permet d'afficher le texte entre guillemets sur la console. Notez que cette ligne se termine par un point-virgule qui permet de clore l'instruction. Enfin, la ligne 6 permet de retourner le code 0 à la console afin de lui signaler que le programme s'est exécuté correctement. Sauvegardez ce code, compilez-le, exécutez-le. Votre premier programme C est fait !

Un `main()` est obligatoire ! Sans lui, pas de point de départ pour votre programme. Vous devrez donc pour le moment tout rédiger à l'intérieur du `main()`.

1.5 La syntaxe

Il est temps d'étudier plus en détail le langage C. Comme tout langage, l'apprentissage passe par les étapes de la découverte de l'alphabet et du vocabulaire de base. À ce titre, la liste suivante regroupe l'ensemble des symboles autorisés dans le langage C ainsi qu'une courte description.

- les 26 lettres majuscules et minuscules de l'alphabet latin : A, a, B, b, ..., Z, z ;
- les 10 chiffres arabes : 0, 1, ..., 9 ;
- les symboles d'opération arithmétique : + - * / ^ ;
- les symboles d'opérations binaires et logiques : & | && || ;
- les symboles de relation : < > = == != ;
- les séparateurs simples et en tandem : , ; . : () { } [] ' " ;
- d'autres symboles d'enrichissement : % # ? _ \

Cette liste indique que le compilateur n'est pas tenu d'accepter d'autres symboles. L'évolution des compilateurs rend possible, pour la plupart, l'utilisation des lettres accentuées. Afin de conserver une compatibilité maximale de votre code, **nous vous demandons de vous restreindre à l'usage strict des symboles listés ci-dessus**⁸.

Maintenant que l'alphabet est connu, il est temps de former des mots. À la différence des langages parlés, le C est un langage qui vous permet de créer vos propres mots, notamment au travers des noms de variables. Néanmoins, afin que le compilateur puisse faire son travail correctement, une série de mots-clés a été définis. Ces mots ont un sens très particulier, et vous ne pourrez les utiliser que dans des contextes précis. Vous devez donc les connaître afin de les éviter lorsque vous nommerez vos variables.

- types de données : `char const double float int long short signed unsigned void volatile` ;
- allocation : `auto extern register static` ;
- constructeurs : `enum struct typedef union` ;
- boucle : `do for while` ;
- sélection : `case default else if switch` ;
- rupture de séquence : `break continue goto return` ;
- autres : `sizeof inline restrict`.

8. De toute façon vous coderez en anglais, donc les accents...

2 Les variables

Il est grand temps de manipuler un peu le C! Mais que manipuler exactement? On sait que le C est un langage impératif, c'est à dire qu'il modifie l'état de la mémoire à chaque action. La mémoire est en fait un grand espace de stockage (quelques mega ou giga octets disponibles dans la RAM), dans lequel on vient placer des éléments. Dans le contexte du langage C, un élément est représenté à l'aide d'un nom et associé à une valeur.

Définition 3 (Variable). Une variable est un identificateur évoluant au cours du temps et dont le nom est formé de lettres et/ou chiffres et/ou du symbole `_`.

On peut faire le parallèle entre les variables et les conserves alimentaires. Afin de savoir ce que contient une boîte, il est nécessaire de l'étiqueter. Tout comme les conserves de tata Paulette, une variable contient *quelque chose* et est repérée par un nom. En plus d'avoir un nom, le langage C impose aux variables d'avoir un *type de donnée*. Puisque que le C est un langage très proche de la machine, le fait de donner un type particulier aux variables permet de réaliser plus facilement des optimisations et donc faire tourner les programmes plus rapidement tout en prenant moins de mémoire.

Il existe 6 types de données élémentaires en C, chacun permettant de stocker des valeurs numériques particulières :

- les types `char`, `short`, `int` et `long` permettent de stocker des entiers,
- les types `float` et `double` permettent de stocker des nombres décimaux.

2.1 Déclaration et modification

L'utilisation d'une variable passe par deux phases : sa *déclaration* puis la *modification de sa valeur*. La déclaration d'une variable équivaut à prendre un bocal et coller une étiquette dessus. La modification équivaut à changer le contenu du bocal. Afin de mieux saisir le fonctionnement de ces dernières, étudions le court extrait de code suivant.

```
1  #define CST 3.73
2
3  int main ()
4  {
5      int a;
6      int b = 10;
7      char c = 'A';
8      float d, e = 1.6;
9      a = b;
10     c = c + 1;
11     c += 1;
12     c++;
13     d = e - 0.6;
14
15     return 0;
16 }
```

La toute première ligne permet de définir une *constante*, c'est à dire une variable qui ne change pas de valeur au cours du temps. Elle s'appelle `CST` et vaut le nombre flottant 3.73.

Il est possible d'identifier facilement les déclarations (lignes 5 à 8) et les affectations (lignes 9 à 13). Remarquons que les lignes 6 à 8 montrent comment réaliser une déclaration suivie immédiatement d'une affectation. On appelle cela une *initialisation*.

Concentrons-nous pour le moment sur les déclarations. La ligne 5 permet de définir une nouvelle variable nommée `a` et de type `int`. On ne peut pour le moment pas connaître la valeur de la variable `a` car on ne lui a pas donné de valeur, au contraire des lignes 6 et 7. Dans tous les cas, `a` ne vaut pas « rien »! Sa valeur dépend du contenu de la mémoire à l'endroit où la variable est.

La ligne 7 permet de définir un `char`, c'est à dire une valeur représentant un caractère. La table d'association valeur - caractère est référencée dans la table ASCII. Nous vous laissons jeter un œil sur Internet pour regarder à quoi ressemble cette table. Vous vous rendrez compte que le caractère 'A' vaut la valeur 65. La ligne 8 indique comment déclarer deux variables du même type. Ainsi, les variables `d` et `e` sont toutes les deux de type `float`. On ne connaît pas la valeur de `d`, et celle de `e` sera de 1.6 après initialisation.

Étudions maintenant les lignes d'affectation. La ligne 9 transfère la valeur de `b` dans la variable `a`. Ainsi, après cette ligne, `a` vaut la même valeur que `b` soit 10. Les lignes 10 à 12 ont exactement la même utilité. Elles permettent d'*incrémenter* la variable `c`, c'est à dire, ajouter 1 à sa valeur. Enfin, la ligne 11 permet de modifier la valeur de la variable `d` en lui donnant la valeur `e - 6`, c'est à dire $1.6 - 0.6$.

TD Exercices 1 et 2.

Dans tous les cas, retenez que le même type de données doit se trouver de part et d'autre du symbole `=`.

2.2 Affichage et saisie

Manipuler des variables, c'est bien ; vérifier le résultat d'un calcul, c'est mieux. Pour afficher des éléments sur la console, vous savez déjà que l'on peut rédiger une ligne de code contenant le mot `printf`. Il s'agit en réalité d'une *fonction* ; nous verrons dans un prochain chapitre ce que c'est exactement et comment en définir de nouvelles. L'utilisation de cette dernière n'est pas particulièrement intuitive mais relativement puissante puisqu'elle permet d'afficher n'importe quelle variable, de n'importe quel type, sous n'importe quel format. Vous allez déduire vous-même la façon d'utiliser cette fonction au travers des exercices suivants.

TD Exercices 3 à 6.

L'affichage étant normalement clair, il est temps d'apprendre à saisir une valeur dans notre programme. Nous verrons au cours du temps d'autres méthodes plus efficaces, mais l'avantage de la saisie formatée est de directement interpréter la valeur saisie et de la placer dans le type de donnée approprié. La fonction utilisée ici est la fonction `scanf()`. Son fonctionnement est particulièrement similaire à celui de l'affichage. Détaillons un extrait de code pour comprendre son fonctionnement.

```
1  int a;  
2  float b;  
3  char c;  
4  scanf ("%d", &a);  
5  scanf ("%f %c", &b, &c);
```

Nous voyons à la ligne 4 comment récupérer la valeur d'un entier de la part de l'utilisateur. Le formatage est plus simple que le `printf` puisqu'il n'est pas nécessaire de placer le symbole `\n` après le `%d`. En revanche, il est **obligatoire** de placer un symbole `&` avant le nom de la variable à remplir. Nous verrons pourquoi dans le cours sur les pointeurs. La ligne 5 détaille sans trop de surprise comment récupérer 2 valeurs sur une même ligne d'entrée.

TD Exercices 7 et 8.

2.3 Détail de l'état de la mémoire

Au niveau de l'ordinateur, qui contient une mémoire appelée *mémoire vive* ou RAM (*Random Access Memory*⁹), on dispose d'*octets*. Rappelons que c'est dans cette mémoire que viennent se charger les programmes afin de s'exécuter.

Définition 4 (Octet). Un *octet* est un espace de stockage de 8 bits.

Définition 5 (Bit). Un *bit* est un espace de stockage permettant de stocker la valeur 0 ou la valeur 1.

Adresse	Valeur
...	...
0x0162	17
0x0163	32
0x0164	0
0x0165	0
0x0166	0
0x0167	0
0x0168	255
0x0169	12
0x016A	64
0x016B	3
...	...

Tout d'abord, jetons un œil à cette fameuse mémoire que le langage C permet de modifier. Il ne s'agit de rien de plus qu'un ensemble de cases collées les unes aux autres, chacune permettant de stocker une valeur sur un octet. Pour repérer ces cases facilement dans la machine, on leur attribue une adresse. L'usage veut qu'une adresse soit un nombre entier sur 32 ou 64 bits. Leur représentation est ici simplifiée sous la forme de valeur hexadécimale. Chaque case possède ainsi une adresse unique et une valeur sur 8 bits. Il est impossible de ne rien avoir dans une case. En effet, pour stocker une valeur sur un bit, on utilise des composants proches d'un interrupteur : s'il y a de l'électricité à l'intérieur, la valeur est à 1, sinon elle est à 0.

On observe donc que les adresses se suivent¹⁰, même si elles sont notées en hexa, et qu'à chaque adresse correspond une valeur. Notons qu'elles sont toutes positives, et pourtant, vous apprendrez à former des entiers négatifs avec, et même des décimaux ! Mais ceci est une autre histoire.

Il est maintenant temps de comprendre comment la manipulation des variables affecte l'état de la mémoire. L'exemple ci-dessous présente quelques déclarations de variables ainsi que l'état de la mémoire. La partie gauche présente les lignes de code réalisées, la partie droite donne l'état de la mémoire à la fin de l'exécution du code.

```

1 // Déclaration de la variable a
2 int a;
3 // Déclaration et initialisation de b
4 short b = -2;
5 // Déclaration et initialisation de c
6 char c = 'z';

```

Adresse	Nom	Valeur
0x0162	a	?
0x0163		
0x0164		
0x0165		
0x0166	b	-2
0x0167		
0x0168	c	122
0x0169	?	?
0x016A	?	?
0x016B	?	?

Commentons ces résultats. La déclaration de la variable *a* est associée à la ligne de code

9. Oui, comme l'album des Daft Punk, enfin presque.

10. En réalité, les adresses ne sont pas croissantes mais décroissantes, un détail qu'un cours de compilation pourrait vous expliquer.

n°2 : `int a;`. Elle permet de réserver de la place pour 4 octets dans la mémoire entre les adresses 0x0162 et 0x0165. Cet emplacement est alors réservé uniquement pour la variable `a`. La surprise consiste en ce point d'interrogation dans la case *Valeur*. Souvenez-vous qu'il est impossible de ne pas remplir la mémoire. Lorsqu'on déclare une variable, sans lui donner de valeur, cette dernière prend alors la valeur de la mémoire lors de la déclaration. On ne sait donc pas ce qu'elle vaut tant qu'on ne l'a pas affichée ou modifiée.

La ligne n° 4 (`short b = -2;`) permet de réserver deux octets en mémoire et de leur attribuer la valeur -2. Pour le moment, on ignore la façon dont l'ordinateur peut stocker des valeurs négatives sur un ou plusieurs octets, mais ne vous inquiétez pas, vous en saurez plus dans un prochain chapitre. De la même façon, la ligne 6 déclare et initialise un caractère nommé `c` à la valeur 'z'. Notez que dans la mémoire, la valeur 'z' est représentée par l'entier 122. Ceci est dû à la table ASCII qui permet d'attribuer une valeur entière à n'importe quel caractère.

2.4 Taille des types standards

Afin de parfaire la compréhension des variables, il est nécessaire de connaître la quantité de mémoire réservée pour chaque type de donnée. La première colonne du tableau suivant vous donne les différents type de données utilisables. La seconde vous permet de savoir quoi mettre dans ces types. La troisième colonne vous indique la quantité de mémoire que tout système doit au minimum réserver pour chaque type. Enfin, la dernière colonne vous indique la quantité de mémoire couramment allouée sur des systèmes modernes.

Type	Données	Taille minimale	Taille courante	Valeur maximale Nombre de valeurs
<code>char</code>	caractères	1 octet	1 octet	256
<code>short</code>	petits entiers	2 octets	2 octets	65536
<code>int</code>	entiers	2 octets	4 octets	$\approx 4.3 \cdot 10^9$
<code>long</code>	entiers	4 octets	4 octets	$\approx 4.3 \cdot 10^9$
<code>float</code>	décimaux	4 octets	4 octets	$3,4 \times 10^{38}$
<code>double</code>	décimaux précis	8 octets	8 octets	$1,7 \times 10^{308}$

Valeur maximale

TD Exercices 9 à 11.

2.5 Forçage de type

Quelques fois, nous sommes obligés de convertir une valeur en un autre type. Par exemple la fonction mathématique partie entière accepte en entrée un flottant et peut retourner un entier. Dans ce cas, il existera nécessairement une ligne de code avec de part et d'autre du symbole `=` des types de données différents. Il est alors nécessaire de forcer le type de l'opérande de droite pour le convertir dans le type donné par l'opérande de gauche. Par exemple, si l'on souhaite récupérer la partie entière d'un flottant `f` facilement, on écrit `int a=(int)f;`

Le nouveau type de donnée est placé entre parenthèses avant le nom de la variable à *caster*. Lorsqu'on caste un flottant en entier, seule la partie entière du flottant est conservée. Si le type de conversion n'est pas précisé, le cast est dit alors *implicite*. On distingue alors deux cas :

- le cast est effectué au travers d'une affectation : la valeur de droite est castée à partir du type de la variable de gauche ;

- le cast est effectué autour d'une opération arithmétique : l'opération s'effectue dans le type de plus grande précision.

Résumons tout cela au travers de quelques exemples.

```
int a;  
short b = 5;  
float f = 3.14;  
a = 0.9;      // Cast implicite: conversion de 0.9 en entier  
b = a;        // Cast implicite: conversion de l'entier a en short  
a = f + b;    // Cast implicite (addition) puis implicite  
              ↪ (affectation)  
b = (short)f; // Cast explicite
```

TD Exercice 12.

3 Pour aller plus loin

Avant de conclure ce chapitre concernant les variables, abordons la notion importante du calcul flottant et de précision au travers des exercices suivants.

TD Exercices 13 et 14.

Résumons maintenant tout ce chapitre concernant les variables à l'aide d'une fiche mémoire avant de passer au chapitre suivant.

Fiche de synthèse : variables.

Variable : Espace de stockage d'une valeur d'une certaine taille associée à un nom.

- Une variable ressemble à une boîte de conserve dont on choisit la taille, le nom et le contenu.
- La boîte de conserve, la variable, n'est jamais vide.
- La taille est en octets (8 bits).
- Le nom est formé de lettres, chiffres ou _.
- Une variable possède une adresse et un type.
- Un entier est stocké dans un `char`, `short`, `int` ou `long`.
- Un décimal est stocké dans un `float` ou `double`.

Déclaration : `type nom;`

Déclaration et affectation : `type nom = valeur;`