

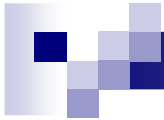
Cours de Langage C

Les bases du langage



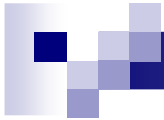
But des 1ères séances

- Préciser (un peu) ce qu'est l'informatique
- Bases d'un langage informatique : le langage C
- Bases du raisonnement informatique
- Prise en main d'un environnement de développement :
Code::Blocks
- Essais de 1ers programmes



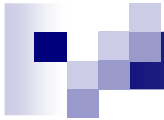
Le Langage C

- Créé au début des années 70 par ***Dennis Ritchie*** et ***Brian Kernighan***
- Langage multi plate-forme : Windows, Mac, Linux, nombreux micro-processeurs
- En respectant la norme ANSI/ISO, le même code va pouvoir être recompilé par plusieurs systèmes
- Reste encore aujourd'hui un langage très utilisé dans le monde (perfectionné depuis : la norme a évolué)



Le Langage C

- Langage évolué / langage de bas niveau
 - Évolué : « relativement » proche de l'humain
 - Bas niveau : proche de la machine
- Langage C :
 - Langage évolué
 - Mais aussi assez proche de la machine
 - Permettra par exemple de programmer des microprocesseurs.



1ère Approche de l'informatique

- Manipulation de données
 - Calculs, entrées, sorties – saisies, affichages – écriture
- et lecture dans des fichiers, ...
- Données : stockées dans des cases mémoires
 - Case mémoire : emplacement physique où est stockée (sous forme électrique) la donnée à traiter
 - La case mémoire élémentaire : c'est l'octet (on y reviendra)
 - Une case mémoire : connue par la machine grâce à son adresse = son emplacement physique
 - Connue par le programmeur par un nom (à choisir)



1ère Approche de l'informatique

- Traitements de base sur les mémoires
 - ☐ Initialisation d'une mémoire
 - ☐ Affectation du résultat d'un calcul ou d'un traitement à une mémoire
- Et aussi
 - ☐ Saisie d'une valeur au clavier et stockage dans une mémoire
 - ☐ Affichage sur l'écran du contenu de la mémoire



1ère Approche de l'informatique

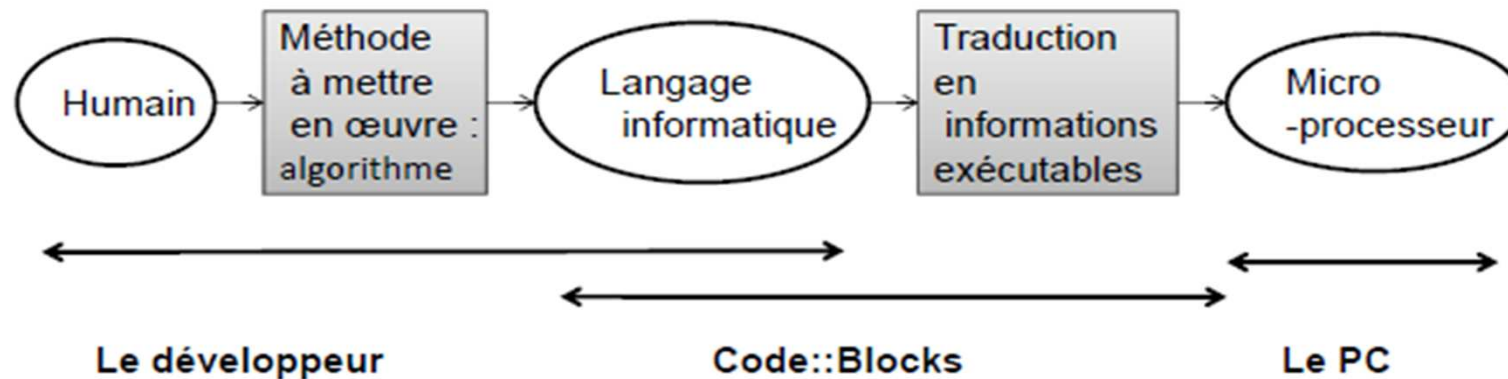
- **Un exemple d'algorithme** : Résolution d'une équation du 1^{er} degré à coefficients réels :

$$a.x + b = 0$$

```
Saisir a puis b
si a=0 faire
    si b=0 faire
        Afficher « Tout réel est solution »
    sinon Faire
        Afficher « Pas de solution réelle »
sinon Faire
    Calculer  $-b/a \rightarrow$  Solution
Afficher Solution
```

1ère Approche de l'informatique

- Processus de développement :
 - A partir d'un problème donné





Les bases du Langage C

- Le programme « main »

```
main()  
{  
    .....  
}
```



Les bases du Langage C

- Une ligne de code ou instruction se termine par « ; »

```
main()
{
    .....
    mem3 = mem1 + mem2;
    .....
}
```



Affectation : avec le signe =

- Ne permet pas de tester l'égalité mathématique !
- Affecte l'expression de droite à la variable de gauche
 - ☐ Expressions non autorisées :
 - $b + c = 50 ;$
 - $3 = a ;$
 - ☐ Instructions autorisées :
 - $b = a + c ;$
 - $b = c + 77 ;$
 - $a = b = 44 ;$

Exemple 1

■ En pseudo-code

20 → mem1

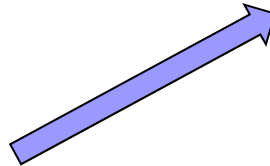
250 → mem2


mem1 + mem2 → mem3

Pour utiliser une variable, il faut, d'abord l'avoir déclarée en précisant son type

■ En Langage C

```
main()
{
  int mem1;
  int mem2;
  int mem3;
  mem1 = 20;
  mem2 = 250;
  mem3 = mem1 + mem2;
}
```





Exemple 1 (variantes)

■ *main()*

```
{  
/* Utilisation de la « , » */  
int mem1,mem2,mem3;  
mem1 = 20;  
mem2 = 250;  
mem3 = mem1 + mem2;  
}
```

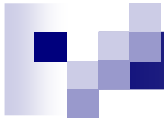
■ *main()*

```
{  
// Déclaration et Initialisation  
int mem1=20 , mem2=250 ,  
mem3;  
mem3 = mem1 + mem2;  
}
```



Commentaires du code

- Un code doit être commenté
 - Afin d'être lisible par d'autres (vos enseignants, votre binôme, vos futurs collègues, les personnes qui vont modifier votre code, ...)
 - Les commentaires sont compris entre `/*` et `*/` avec d'éventuels sauts de lignes
 - Ou encore `//` uniquement pour le reste de la ligne



Lisibilité du code

- Un code doit être lisible
 - ☐ Grâce aux commentaires
 - ☐ Avec des noms de variables ou de fonctions explicites
 - ☐ Avec une mise en forme des différents blocs
 - ☐ Grâce à l'indentation du code



Identificateurs des variables

- **Identificateur** = nom permettant de désigner une variable (ou une fonction)
- Caractères autorisés
 - a, ..., z, A, ..., Z et 0, ..., 9 et _
 - Un identificateur commence par une lettre ou par le caractère _
 - **Attention** : Le C différencie majuscules et minuscules



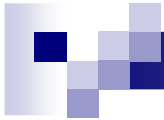
Identificateurs des variables

- Choix de nom explicite
 - Améliore la lisibilité du code
 - *Exemples :*
 - `ma_Variable` ou `maVariable`
 - `temperatureFahrenheit`
- Les noms explicites de variables permettent de ne pas surcommenter



Déclaration des variables en C

- Déclarer une variable c'est définir son nom et son type
`int mem1; /* mem1 est de type int */`
- En Langage C, on doit déclarer une variable avant de l'utiliser
- On effectue les déclarations de variables en début de bloc, c-à-d juste après « { »
- Pas d'importance dans l'ordre des déclarations
- **Attention** : ne pas mélanger ligne exécutables et déclarations



Quelques types de variables en C

- Type entier : `int`

- *Exemple* : `int` maVariable ;

- Type réel : `float`

- *Exemples* :

- `float` discriminant ;
 - `float` eps = 1e-6 ;
 - `float` tempe = 23.5 ;



Variables en C

- On va rencontrer les variables à **3 occasions** :
 - Déclaration → traité ici
 - Initialisation → **Souvent indispensable** ... pour éviter des programmes faux (même s'ils compilent)
 - Utilisation → traité à plusieurs reprises
- Initialisation : de 2 façons

<pre>int maVariable ; ...//déclaration ou code maVariable = 2 ; // Init</pre>	<pre>int maVariable = 2 ; // Déclaration & Init simultanées</pre>
--	--

Fonctions d'affichage (1ère approche)

- Affichage d'une phrase, d'un entier, d'un réel :

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

```
main()
```

```
{
```

```
    int mem1 = 10 ;
```

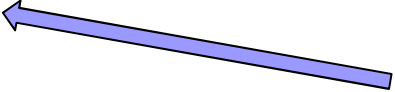
```
    float mem2 = 20.0 ;
```

```
    printf("Bonjour, ça va bien ?\n");
```

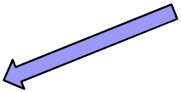
```
    printf("%d\t", mem1);
```

```
    printf("%f\n", mem2);
```

```
}
```



Nécessaire en début de code
pour utiliser la fonction
d'affichage



3 appels à la fonction
d'affichage



Fichiers d'en tête : .h

- « Stdio » : *standard input output*
- Fichier « .h » (comme header) ou fichier d'en-tête
- Contient la déclaration de la fonction utilisée
- Permet de s'en servir en sachant :
 - ☐ Ce qu'elle fait
 - ☐ La syntaxe à utiliser
 - ☐ Mais sans connaître le code de la fonction
- Il en existe de nombreux autres :
 - ☐ math.h, stdlib.h, string.h, ctype.h, time.h

Fonctions de saisie au clavier

(1ère approche)

- Saisie d'un entier ou d'un réel :

```
#include <stdio.h>
main()
{
    int mem1 ;
    float mem2 ;
    ...
    scanf("%d", &mem1);
    scanf("%f" , &mem2);
    ...
}
```

Attention à bien
mettre le « & » avant le
nom de la variable
(mem1 ici)

Attention : pas de
« \n » du type :
'scanf('%d\n ');



Structure d'un programme simple

```
#include <stdio.h>
main()
{
    int mem1 ;
    float mem2 ;
    ...
    scanf("%d",&mem1);
    scanf("%f" &mem2);
    ...
}
```

En-tête

Corps du programme : main()

Accolade de début

Déclarations des variables

Instructions du programme

Accolade de fin



Les structures de contrôle

- Quelques problèmes simples :
 - ☐ Convertir les Fahrenheit en Celsius et afficher un message si la température (en °C) est négative
 - ☐ Déterminer le min et le max parmi 2 nombres
 - ☐ Afficher les nombres impairs entre 0 et 100 (inclus)
- Nécessitent des tests ou des boucles :
ce sont les structures de contrôle



Structure de contrôle : Les boucles

- 3 types de boucles
 - Boucles *for*
 - De $i = 0$ à N , **faire** instructions
 - Boucles *while*
 - **Tant que** (condition vraie), **faire** instructions
 - Boucles *do while*
 - **faire** instructions **Tant que** (condition vraie)



La boucle « for »

■ Exemples de boucles *for* :

```
for (i=0;i<100;i++)  
    Nombre = Nombre *2; /* Fin de la boucle for : i s'incrémente s'il  
                           n'est pas égal à 100 et la ligne d'instruction est ré-exécutée */
```

```
for (i=0;i<=100;i=i+10)  
{  
    Nombre = Nombre *2; /* Plusieurs instructions : donc on met des  
                           accolades */  
    Nombre2 = Nombre2 *3;  
} /* Fin de la boucle for : i s'incrémente etc... */
```

```
for (i=0;;i++) { ... } /* La condition de terminaison est facultative  
                           mais pas le ; */
```

```
for (;) { ... } /* Une boucle for particulière !! */
```



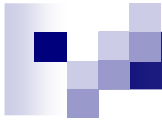
La boucle « while »

■ Exemples de boucles *while* :

```
while (i<100)
    i++;    /* Fin de la boucle while: i s'incrémente s'il n'est pas égal
            à 100 et la ligne d'instruction est ré-exécutée*/
```

```
while (i<100)
{
    i++;
    Nombre = Nombre *i;    /* Plusieurs instructions : donc on met
                           des accolades */
    Nombre2 = Nombre2 *2*i;
};                          /* Fin de la boucle while*/
```

```
while (1) { ... }          /* Une boucle while particulière !! */
```



La boucle « do »

■ Les boucles *do ... while* :

- Syntaxe proche des *while*
- Mais ce type de boucle sera exécutée au moins une fois (même si la condition est fausse d'emblée)

```
do {  
    i++;  
}  
while (i<100) ;
```

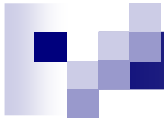


Les conditions d'arrêt des boucles

■ Condition d'arrêt : expression logiques

□ Opérateurs :

Opérateurs	Signification	
<	inférieur à	}
<=	inférieur ou égal à	
>	supérieur ou égal à	
>=	supérieur ou égal à	
==	égal (ne pas confondre avec = qui correspond à l'affectation)	}
!=	différent de	
		Même priorité
		Priorité inférieure



Les conditions d'arrêt

- Condition d'arrêt : expression logique

- Évaluation d'expression logique :

- A l'aide de : `||` (Ou) `&&` (Et) `!` (négation logique)
 - Exemple : `(i >= 100) || (j == 10) && !(k > 0)`
 - `&&` prioritaire sur `||`
 - 0 correspond à Faux
 - Toute valeur non nulle correspond à Vrai
 - Exemple : `if (correct)` ou bien `if (!correct)`



Structure de contrôle : Les tests

■ 2 types de tests

- Les *if ... else*

- Les *switch*

■ Les *if ... else*

```
if (Condition) instruction1;
```

```
if (Condition) instruction1;  
else instruction2;
```

```
if (Condition)  
{  
    instruction1;  
    instruction2;  
}  
else { instruction2 ;}
```




Les tests : les if imbriqués

- Exemple

```
if (choix1 < 0)
    if (choix2 > 0)
        instruction1;
else
    instruction2;
```

- Le ***else*** se rapporte toujours au dernier ***if*** sans ***else***
- L'indentation n'a pas de conséquence sur la compilation ...
- ... mais sur la lisibilité: Pensez donc à indenter !
 - Code::Blocks propose d'ailleurs une mise en forme via *Plugins*
> *Source code formatter*



Les tests : les if successifs

- Application : choix successifs

if (choix1 < 0) instruction1;

else if (choix2 > 0) instruction2;

else if (choix3 > 0) instruction3;

else if (choix4 > 0) instruction4;

[***else*** instruction5;]



Les tests

■ Le *switch*

```
Switch (Choix)
{
  case 1 : instruction1 ; break;
  case 2 : instruction2 ; break;
  case 3 : instruction3 ; break;
  ...
  default : instruction_default ; break;
}
```

- ☐ Le break permet de sortir de la boucle lorsque la condition est vérifiée
- ☐ La variable de choix doit être de type entier (ou caractère)
- ☐ Le default n'est exécuté que si aucun des autres cas n'est vrai : il est facultatif.
- ☐ Le break après default est facultatif mais est une bonne convention d'écriture



Tableau à 1 dimension

- Comme pour les variables, on va rencontrer les tableaux dans 3 types de circonstance : **Déclaration – Initialisation – Utilisation**

- **Déclaration :**

`int tab[10] ;`

Attention : `int tab[n]` est interdit en C

- La numérotation des composantes commence à 0 et se termine à N-1 ; ex : `tab[0]` à `tab[9]`

- **Utilisation :**

```
x = tab[2] + tab[4];  
tab[2] = tab[4]*2 ;  
tab[i] = i + j ;
```

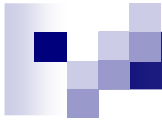


Tableau à 1 dimension

■ Attention :

Si on dépasse la taille du tableau : le programme compile mais l'exécution « plantera »

■ Initialisation :

- ☐ Fortement conseillé
- ☐ 2 façons : soit à la déclaration, soit dans le code

```
int Montab[5] = {1,3,17,-54,2};
```

```
int Montab[5];
```

```
... bout de code
```

```
// boucle remplissant les cases tab[i]
```



Tableau à 2 dimensions

- Pratiquement comme pour les tableaux 1D.

- **Déclaration :**

`int tab[10][2] ; // 10 lignes, 2 col.`

- **Utilisation :**

`x = tab[2][3] + tab[2][4];`

`tab[2][4] = tab[1][4]*2 ;`

`tab[i][j] = i + j ;`



Le binaire : la base 2

- En informatique, les informations sont stockées dans des dispositifs électroniques ne prenant que 2 états électriques : un état correspond à 0 et l'autre à 1.
- On parle alors de bits (« **binary digit** »)
- Ca impose de représenter les nombres en base 2 (ou binaire)
- On peut presque s'en affranchir pour programmer en Langage évolué.
- Voyons en ce pendant le principe



Le binaire : la base 2

- En base 10 : on exprime les nombres avec 10 chiffres de 0 à 9 :

$$1984 = 1.10^3 + 9.10^2 + 8.10^1 + 4.10^0$$

- En base 2: on a seulement 2 chiffres 0 et 1

$$1011 = 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0$$

soit 11



Le binaire : la base 2

- En informatique : la « case mémoire » élémentaire est l'octet (groupement de 8 bits) ou byte en anglais
- Une donnée peut être mémorisée sur plusieurs octets (voir juste après)
- Un octet permet de coder :
 - De 0000 0000 à 1111 1111
 - Soit de 0 à 255 en non-signé
 - ou de – 128 à +127 en signé

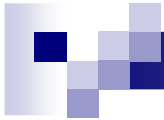


Types des variables

■ *Pourquoi déclarer le type des variables ?*

□ Pour que l'ordinateur puisse traiter une mémoire il doit connaître :

- Son emplacement physique
- La taille occupée par cette mémoire (en nombre d'octets)
- Emplacement physique ? Rôle de la chaîne de développement (Code::Blocks)
- La taille occupée ? le programmeur le fait en indiquant le type



Types des variables : les entiers

- Types entiers :

- ☐ char : 1 octet
- ☐ short: 2 octets
- ☐ long: 4 octets
- ☐ int: 2 ou 4 octets car ça dépend du compilateur : type le plus employé. Ici 4 octets.

- Et aussi

- ☐ Types « *unsigned* » : unsigned short int, unsigned long int.



Opérations sur les entiers

■ 5 opérations :

☐ + (addition)

☐ - (soustraction)

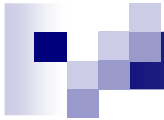
☐ * (produit)

☐ / (division)

☐ % (reste de la division entière).

■ Évaluation de la gauche vers la droite

■ / et * : même niveau de priorité et priorité sur + et -



Opérations sur les entiers

- Opérations unaires :
 - ☐ `i++` post incrémentation
 - ☐ `++i` pré incrémentation
 - ☐ `i--` post décrémentation
 - ☐ `--i` pré décrémentation
 - ☐ `l += 5` signifie `i = i+5`
 - ☐ Idem avec `i *= a` etc.



Types des variables : les réels

- Types réels :

- ☐ float: réel en simple précision codé sur 4 octets
- ☐ double: réel en double précision codé sur 8 octets

- 4 opérations :

- ☐ Les mêmes que pour les entiers, mais sans le modulo (%)



En résumé

Type en C	Nom Français	Taille (octets)	Valeurs permises
unsigned short int	entier court non signé	2	0 à 65 536
short	entier court	2	-32 768 à 32 767
unsigned long int	entier long non signé	4	0 à 4 294 967 295
long int	entier long	4	-2 147 483 648 à 2 147 483 647
int	entier (16 bits)	2	-32 768 à 32 767
int	entier (32 bits)	4	-2 147 483 648 à 2 147 483 647
unsigned int	entier non signé (16 bits)	2	0 à 65 536
unsigned int	entier non signé (32 bits)	4	0 à 4 294 967 295
char	texte	1	256 caractères (0 à 255)
float	nombre décimal	4	1.2e-38 à 3.4e38
double	nombre double	8	de 2 2e 308 à 1 8e308



Priorité des opérateurs

- L'expression **la plus à droite** est d'abord évaluée puis stockée dans la variable immédiatement à gauche du signe d'affectation « = »
- Dans une expression, la règle d'associativité est de gauche à droite
- Avec certaines règles de priorité :

++	--
* / %	
+ -	

(du plus au moins prioritaire)



Choix des types

- **Attention** : Le compilateur ne fait pas le traitement des dépassements de capacité.
- C'est au développeur de typer ses variables de façon correcte

- Exemple :

```
char a,b,c;
```

```
...
```

```
a = b + c ;
```

Ce programme va être compilé, s'exécuter
(apparemment) normalement, mais est faux !



Conversion de types

- **Attention** : les opérateurs sont conçus pour 2 opérandes de même type et produisent un résultat du même type.

`int a, b;`

`a + b` est de type `int`

`5 / 2` est donc `int` et vaut 2 !!

`32 / 100 * 100` vaut 0

- Quand 2 opérandes de types différents, le compilateur convertit dans le type dominant avec la convention :

`char < short int < long int < float < double`

- Forçage de type :

`5 / 2.0` Le résultat est un `float` valant 2.5

`32 / 100.0 * 100` Le résultat est un `float` valant 32.0



Conversion de types

■ Exemples :

`int a = 64, b = 2;`

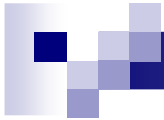
`float x = 1, y = 2;`

`b = 7 / b;` ***/* En premier : calcul de l'argument à droite du = donc 7 (entier) / 2 (entier) donne 3 (entier, reste 1, que l'on obtiendrait par 5%2). donc b=3 */***

`x = 7 / b;` ***/* 7 et b entiers => passage en réel inutile, calcul de 7/2 donne 3 (entier, reste 1) puis opérateur = (transformation du 3 en 3.0 puis transfert dans X qui vaut donc 3.0) */***

`x = 7 / y;` ***/* un int et un float autour de / : transformation implicite de 7 en réel (7.0), division des deux réel (3.5), puis transfert dans x */***

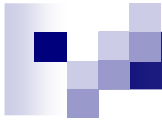
`x = 7.0 / b;` ***/* un int et un float autour de / : transformation implicite de b en réel, division des deux réel (3.5), puis transfert dans x */***



Conversion de types

■ Attention :

- ☐ Le compilateur fait une conversion de type sans tenir compte du membre de gauche de l'affectation
- ☐ Le résultat peut donc dépasser le type prévu
- ☐ D'où perte d'informations !



Constantes symboliques

- Définies en début de programme (au même niveau que les `#include`) par :

`#define` PI 3.14159

`#define` TAUX_TVA 0.196

- Règle de style : constantes symboliques en majuscules
- Règle de syntaxe : pas de signe = et pas de ;
- Intérêt :
 - ☐ Lisibilité du code
 - ☐ 1 seule modification de la valeur même s'il y a plusieurs occurrences dans le code source



Constantes symboliques

■ Exemples d'utilisation

```
#define PI 3.14159
#define TAUX_TVA 0.196
main()
{
    double x, omega=2,5;
    double prix, cout;
    ...
    x = sin(2*PI*omega*t) ;
    prix = cout*(1 + TAUX_TVA) ;
}
```

■ Les constantes symboliques sont très utilisées



Affichage console avec printf

- Affichage de variables

int x;

printf("x =%d\n", x);

Affiche l'*int* x et saute à la ligne suivante

Les codes de format d'affichage sont :

%d	int
%c	char
%f	float
%lf	double ...



Affichage console avec printf

- Affichage de variables et de chaînes de caractères

```
int x = 2 ;
```

```
double y = 2.5 ;
```

```
printf("le produit de %d par %lf vaut  
      %lf \n" , x , y, x*y);
```

Affiche : le produit de 2 par 2.5 vaut 5



Saisie clavier avec scanf et printf

- Saisies de valeur de variables

```
int x ;
```

```
printf("Entrez x = \n");
```

```
scanf("%d", &x);
```

Stocke nombre tapé au clavier dans la variable x

Attention :

au signe & devant la variable x

pas de \n dans le scanf