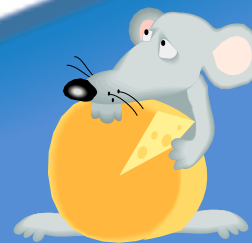


Septembre 2023



loic.yon@isima.fr
<https://perso.isima.fr/loic>



ISIMA 



Historique

1972 : inventé par Dennis Ritchie

Vient du B de K Thomson

Langage de niveau intermédiaire

Plus simple que l'assembleur

1973 : Noyau UNIX codé en C

1978 : K&R avec Brian Kernighan

1989 : ANSI C

1990 et plus : ISO C

C90, C99, C11, C17, C23

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

```
int main()
```

```
{
```

```
    printf("Bonjour les ZZ1!");
```

```
    return 0;
```

```
}
```

Programme C = une suite de fonctions dans un fichier texte

Au moins une fonction = main() point d'entrée du programme

Un élément doit être déclaré pour être utilisé !

Attention à l'**indentation** et à l'**alignement** des accolades





```
import math
```

```
def mafonction (p1, p2):  
    return p1+p2
```

```
if __name__ == "__main__":  
    print("execution")  
    i = "3"  
    j = 2  
    i = math.pi
```

```
    k = mafonction(i, j)  
    print('resultat{}'.format(k))
```

Langage interprété ?



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



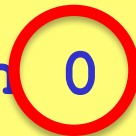
```
int main()  
{
```



```
    printf("Bonjour les ZZ1!");
```



```
    return 0
```



Exécution ligne par ligne

Découverte des erreurs au fil de l'eau

```
}
```



est interprété

Langage compilé !

```
#include <stdio.h>

int main()
{
    printf("Bonjour les ZZ1!");
    return 0;
}
```

ERREURS

COMPILATEUR

Exécution plus rapide

```
0e1f ba0e 0054 09cd 21b8 014c cd21 5468
6973 2070 726f 6772 616d 2063 616e 6e6f
7420 6265 2072 756e 2069 6e20 444f 5320
6d6f 6465 2e0d 0d0a 2400 0000 0000 0000
6e1f 984b 2a7e f618 2a7e f618 2a7e f618
3116 0221 0218 2f72 9618 287e f618
2f72 f918 317e f618 3976 ab18 287e f618
d05d ef18 2e7e f618 a976 ab18 3b7e f618
2a7e f718 1a7f f618 2f72 a918 967e f618
      c675 a818 2b7e f618 2f72 a
      8 2b7e f618
5269 6368 2a7e f618 0000 0000 0000 0000
```

Langage machine



CODER



COMPILER



EXÉCUTER

ERREURS



```
$ code ./zz1  
ou xemacs, vi
```

```
$ gcc zz1.c -o zz1
```

```
$ ls  
$ ./zz1
```



```
$ gcc zz1.c -o zz1 -Wall -Wextra
```

`/* Commentaire */`

`// Commentaire`

- Ce n'est PAS une perte de temps
 - Lisibilité pour vous et les autres
 - Quid dans 6 mois ?
- Deux formes
 - `/* Commentaires sur plusieurs lignes */`
 - Non imbriquables
 - `//` commentaire en fin de ligne
- Propositions de contenu vues plus tard

Bloc


- Ensemble séquentiel d'instructions
 - Accolades
 - Imbriquables

```
{  
    printf("Bonjour les ZZ1");  
    printf("suite du programme");  
}
```



l'indentation ne suffit pas ! Accolades obligatoires

Variable & type de variable

- Zone mémoire nommée
 - Garder une information
 - Valeur par défaut ?
- Toute variable a un **type** 
- Elle ne peut PAS changer de type à l'exécution
- Types standards = proches de la machine
- "Taille" du type = dépend des machines `sizeof`
- Caractère : `char`
- Entier : `int` `long`
- Réel : `float` `double`

Déclaration d'une variable

- Toujours déclarer avant d'utiliser
- Où ?
 - en début de bloc (Norme ANSI ou bonne pratique)
 - en début de programme

```
type nom;  
type nom = valeur;
```

```
int    i;  
int    j = 0, k;  
char   c = 'A';  
float  pi = 3.14;  
double d = 0.0;
```

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

```
int i = 0;   GLOBALE
```

```
int main()
```

```
{                   LOCALE
```

```
  float f = 3.0;
```

```
  printf("Bonjour les ZZ1!");
```

```
  return 0;
```

```
}
```

```
int autre()
```

```
  i? f?
```

```
{
```

```
  printf("autre fonction");
```

```
  return 1;
```

```
}
```

Variable locale ou globale ?

Manipulation de variables

```
int main()
{
    int    i = 0, reste, j=3;
    float d = 1.03, rayon;

    i = i+1;           i=j-1;
    i++;              j--;
    ++i;              --j;
    i+=1;             j-=2;
    i+=3;

    rayon  = 10.0 / ( 3.14 * 2);
    rayon *= 2.0;
    reste  = 5 % 2;

    return 0;
}
```

Quelques opérateurs

- Numérique : $+$, $-$, $*$, $/$, $\%$
- Comparaison : $<$, $>$, $<=$, $>=$, $==$, $!=$
- Logique : $\&\&$, $||$, $!$
- Binaire : $\&$, $|$, $<<$, $>>$
- Affectation : $=$, $++$, $--$, $+=$, $-=$, $*=$, $/=$

Changer de type

- Type fixé à la compilation mais c'est possible
 - *Cast*
 - Opérateur de transtypage
 - Opérateur de coercition
- Transformer une valeur réelle en une valeur entière avec perte éventuelle d'informations

```
double d1 = 10.2;  
double d2 = 7.9;  
int     i  = (int) d1;
```

Mémoire : 0 | | 1

- Binaire
- Décalage ? Interprétation ?
- Programme
 - Langage machine
- Mémoire
 - Variable avec des types différents
 - Interprétation / Normes

Codage d'un entier sur 8 bits

7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0

$$1*2^7+0*2^6+0*2^5+0*2^4+1*2^3+0*2^2+1*2^1+0*2^0$$

138

- 10 (-118)

Codage d'un nombre positif de 0 à 255 = 2^8-1

Codage d'un nombre négatif de -128 à 127

Plusieurs bases

$$138 = 1 * 10^2 + 3 * 10 + 8 \quad (\text{décimal})$$

$$= 10001010 \quad (\text{binaire})$$

$$= 8 * 16 + 10 = 8A \quad (\text{hexadécimal})$$

- « Chiffres » en hexa

0 1 2 3 4 5 6 7 8 9 A B C D E F

- $255 = 15 * 16 + 15 = FF$

- $4095 = 15 * 16^2 + 15 * 16 + 15 = FFF$

Codage d'entier

	Non signé	Signé
1 bit	0 -> 1	Positif ou négatif
1 octet	0 -> 255	- 128 -> 127
2 octets	0 -> 65 535	- 32768 -> 32767
4 octets	0 -> 4294967295	- 2147483648 -> $2^{31}-1$
8 octets	0 -> $2^{64}-1$	- 2^{63} -> $2^{63}-1$
Plus ?		

- Choix de la taille
 - Besoins
 - Machine (processeur)

Codage d'un nombre réel

- $123,125 = 1,23125 * 10^2$
 $= 1111011,001 = 1,111011001 * 2^6$
 $= \text{signe} * \text{mantisse} * 2^{\text{exposant}}$

- Codage : $\pm 0 \pm \infty$ NaN
- Valeurs non représentables



	16 bits	32 bits	64 bits
Signe	1	1	1
Exposant	5	8	11
Mantisse	10	23	52

Opérateurs logiques

A	OU	B
0		0
1		0
0		1
1		1
10		5
5		0

A	ET	B
0	&&	0
1	&&	0
0	&&	1
1	&&	1
10	&&	5
5	&&	0

NON	A
!	0
!	1
!	5

0 est FAUX

Toute valeur NON NULLE est VRAIE

Opérateurs binaires (1)

	7	6	5	4	3	2	1	0	
138	1	0	0	0	1	0	1	0	
& 147	1	0	0	1	0	0	1	1	
<hr/>									
130	1	0	0	0	0	0	1	0	&
155	1	0	0	1	1	0	1	1	

Opérateurs binaires (2)

	7	6	5	4	3	2	1	0
138	1	0	0	0	1	0	1	0


>> 1 = 69

0	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

<< 2 = 552

1	0	0	0	1	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---

Fonctions prédéfinies

- Bibliothèque *library*  Module
- Fichier entête *header* extension `.h`
 - Ne contient que les déclarations
 - Pas le code exécutable correspondant
 - Il faut parfois modifier la ligne de compilation (pour la bibliothèque `math` par ex)
- Exemples de `/usr/include`
 - `stdio.h`
 - `stdlib.h`

printf

- Écrire sur la **sortie standard**
 - Nombre variable d'arguments

```
printf(chaine, arg1, arg2, ...);
```

- Chaîne
 - Chaîne de caractères
 - Instruction(s) de formatage

Exemples

```
int main()
{
    float x =1.0, y=.0;
    int i = 7;
    /* Quelques exemples */
    printf("Un mot\n");
    printf("un nombre : %d\n", 3);
    printf("Solution x:%f, y:%f\n", x, y);

    printf("%d\n", ++i); /* a essayer */
    printf("%d\n", i++);

    printf("%d\n", 3.5);
    printf("\n");
    printf("\t\t %d\n", i);

    return 0;
}
```

Formatage ?

```
float pi=3.14;  
printf("Pi vaut %f", pi);
```

%d : un entier

%x : un entier en hexadécimal

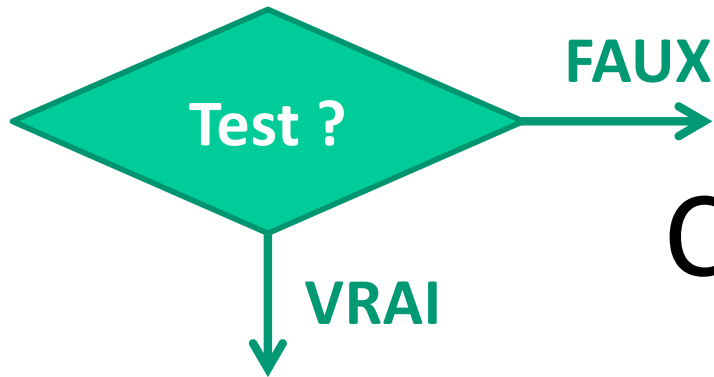
%tailled

%e ou **%f** ou **%g** ou **%lf** : un nb réel

%taille.précisione

%c : un caractère

%s : une chaîne de caractères



Condition (1)

```
if (i>0)
{
    printf("positif");
}
```

```
if (nb1>nb2)
    printf("trop grand");
else printf("trop petit");
```

```
if (a<0)
{
    printf("0");
}
else
{
    printf("1");
}
```

- Accolades ou non ? => Guide de style
- Parenthèses **obligatoires** autour du test

Condition (2)

- Tests simples

```
if (i==-1) printf("i vaut -1");  
if (i!=-1) printf("i différent de -1");  
if (i!=0)   printf("i non nul");  
if (i==0)   printf("i nul");
```

- Tout test non nul est VRAI
Tout test nul est FAUX

```
if (i)      printf("i non nul");  
if (!i)     printf("i nul");
```

Boucles conditionnelles (1)

```
while (test)  
{  
    ...  
}
```

```
int i = 17;  
while (i>0)  
{  
    printf("%d", i);  
    i /=2 ;  
}
```

```
do  
{  
    ...  
} while (test);
```

```
int i = 50;  
do  
{  
    printf("%d", i);  
    --i;  
} while (i>=0);
```

Condition (3)

- ET (ALORS)

```
if ((i>=0) && (i<=10))  
    printf("i compris entre 0 et 10");
```

- OU

```
if ((i>3) || (j!=18))  
    printf("i est plus grand que 3  
           ou j est différent de 18");
```

- Évaluation partielle possible
 - Tester un indice de tableau et un élément de tableau

Condition (4)

- Opérateur ternaire

```
printf("%s", (i==0) ? "ZERO" : "NON NUL");
```

```
(test) ? VRAI : FAUX
```

- Affectation dans un test

```
if (a=b) printf("a=b et b non nuls");
```

```
if ((a=b)) printf("a=b et b non nuls");
```


Boucles conditionnelles (2)

```
for (initialisation; test; incrémentation)
{
    ...
}
```

```
int i = 5;
...
for (i=0; i<10; i++)
{
    printf("%d\n", i);
}
```

scanf (1)

- Lire sur **l'entrée standard**
- Nombre variable d'arguments

```
scanf (chaine, [&]arg1, [&]arg2, ...) ;
```

- Chaine
 - Chaine de caractères
 - Instruction de formatage
 - Présence de l'**esperluette**



scanf (2)

```
int main()  
{  
    char c;  
    int i;  
  
    printf("Taper 0 pour continuer");  
    scanf("%c", &c);  
    printf("Saisir un nombre entre 0 et 10");  
    scanf("%d", &i);  
  
    return 0;  
}
```

Valider les saisies par la touche ENTREE



On précisera plus tard quand mettre l'esperluette ou non

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

```
int main()
```

```
{
```

```
    int    i, j;
```

```
    float  f;
```

```
    printf("taper entier <ENT> reel <ENT>");
```

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

```
    /* Boucle qui occupe le processeur */
```

```
    printf("la saisie est mémorisée");
```

```
    scanf("%f", &f);
```

```
    printf("%d %f", i, f);
```

```
    return 0;
```

```
}
```

Saisie mémorisée

mémoire tampon

(buffer)

Vider cette mémoire

```
int main()
{
    int i;
    char c;

    printf("Saisir i\n");
    scanf("%d", &i);
    printf("Saisir c\n");
    scanf("%c", &c);

    printf("%d %c\n", i, c);
    return 0;
}
```

On saisit i

On passe directement...

```
scanf("%d%c", &i);
```

`%*c` est un caractère qui n'est pas stocké dans une variable

Selon

```
switch (variable)
{
    case valeur1 :
        instructions;
        break;
    case valeur2 :
        instructions;
        break;
    default:
        instructions;
        break;
}
```

```
char c;
scanf("%c%c", &c);
switch (c)
{
    case 'Q' :
    case 'q' :
        printf("Quitter");
        break;
    case '1' :
        printf("Action 1");
        break;
    default:
        printf("Pas reconnu");
}
```

Rappel : '1' est un caractère et non un entier



Vider la mémoire tampon en sortie

- Ecriture "*bufferisée*" pour la performance
- Pas d'affichage à l'écran ou affichage tardif
 - Gênant pour le débogage

```
POUR i DE 0 A BEAUCOUP {  
    /* quelquechose */  
    /* forcer l'affichage ou pas */  
}  
/* affichage */
```

```
fflush(stdout);
```

```
printf("\n");
```

Les fonctions (1)

- Sous-programme
 - Réutilisable
- Identificateur
 - Unique
 - Non réservé
 - Significatif
- Liste de paramètres
 - Nom du paramètre et type (variable)
 - Séparation par une virgule
- Type de retour

Les fonctions (2)

```
double carre (double c)
{
    double val = c*c;
    return val;
}
```

```
void fonction_qui_ne_renvoie_rien (int a)
{
    ...
}
```

```
type_de_retour nom (type1 nom1, type2 nom2, ...)
{
    ...
    [return [valeur/variable];]
}
```

Max (v1, v2) ?

Prototype / signature : `int max(int v1, int v2)`

```
int max(int v1, int v2)
```

```
{  
    if (v1 > v2)  
        return v1;  
    return v2;  
} /* C à l'ancienne */
```

```
int max(int v1, int v2)  
{  
    return (v1>v2)?v1:v2;  
} /* minimaliste */
```

```
int max(int v1, int v2)  
{  
    int r = v2;  
  
    if (v1 > v2)  
        r = v1;  
  
    return r;  
} /* GL compliant */
```

Exemple d'utilisation

```
#include <stdio.h>

int max(int v1, int v2)
{
    ↔ return (v1>v2)?v1:v2;
}

int main()
{
    ↔ int x = max (10, 20);
    ↔ printf("le max est %d", x);
    ↔ return 0;
}
```

Alignement des crochets de bloc



Indentation

Local ou global (1) ?

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

```
int a;
```

```
void f1()  
{  
    printf("%d", a);  
}
```

```
int main()  
{  
    scanf("%d", &a);  
    f1();  
    return 0;  
}
```

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

```
void f1()  
{  
    int a;  
    scanf("%d", &a);  
    printf("%d", a);  
}
```

```
int main()  
{  
    f1();  
    printf("%d", a);  
    return 0;  
}
```

Local ou global (2) ?

```
#include <stdio.h>

void f1()
{
    printf("%d", a);
}

int main()
{
    int a;
    scanf("%d", &a);
    f1();
    return 0;
}
```

```
#include <stdio.h>

void f1(int b)
{
    printf("%d", b);
}

int main()
{
    int a;
    scanf("%d", &a);
    f1(a);
    return 0;
}
```

Récurtivité

$$\left\{ \begin{array}{l} n! = n * (n-1)! \\ 0! = 1 \end{array} \right.$$

```
int fact(int n);
```

Prototype

```
int fact(int n) {  
    return n * fact(n-1);  
}
```

Toujours
vérifier les
paramètres

```
int fact(int n) {  
    int r = 1;  
  
    if (n>1)  
        r = n * fact(n-1);  
    return r;  
}
```

Constante symbolique (1)

```
#define N 100
```

- Mot-clé `#define`
- Fausse "constante"
 - Remplacée en début de compilation (préprocesseur)
 - Pas de type
- Pas de point-virgule en général
- Convention : identificateur en majuscules

Constante symbolique (2)

```
#define N 100
```

```
int main()  
{  
    int i;  
    for ( i = 0 ; i < N ; ++i)  
        printf("merci %d fois\n", i);  
  
    return 0;  
}
```

```
int main()  
{  
    int i;  
    for ( i = 0; i < 100 ; ++i)  
        printf("merci %d fois\n", i);  
  
    return 0;  
}
```


Tableau statique

- Taille fixée à la compilation
- 1^{er} indice : 0
- Dernier indice : taille -1

```
type    identificateur[entier_pos];
```

```
#define N 100  
int     tableau1[1000];  
float   tableau2[N];  
int     tableau3[] = {0,1,2,3};  
int     tableau4[10] = { 0 };
```

```
void afficher(float tab[] int taille)
{
    int i;
    for (i=0;i<taille;++i)
        printf("%f ", tab[i]);
}
```

Tableau et fonction

```
int main()
{
    float tableau[100];

    printf("%f", tableau[0]);
    scanf ("%f", &tableau[5]);

    afficher(tableau, 100);

    return 0;
}
```

Tableau de taille variable ?

- Tableau dont la taille varie pendant l'exécution

```
int   taille = 50;  
float tableau[taille];
```



Ne marche pas !

- Solution à la main
- Solution élégante : plus tard...
- Solution à la main
 - Définir un tableau de grande taille (capacité)
 - Définir une taille effective
 - Manipuler le tableau en vérifiant que la taille effective du tableau est inférieure à la capacité.

```
#define TAILLE_MAX 1000

float notes[TAILLE_MAX];
int  taille;

int main()
{
    do
    {
        printf("Nombre de note(s) ?\n");
        scanf("%d%c", &taille);
    }
    while (taille >= TAILLE_MAX);

    return 0;
}
```

```
#define TAILLE_MAX 100

int tableau[TAILLE_MAX];

int rechercher_g(int v)
{
    int i=0;           Exemple avec && "intelligent"
    while ((i<TAILLE_MAX) && (tableau[i]!=v))
    {
        ++i;
    }
    return i;
}

int main()
{
    printf("%d", rechercher_g(3));
    return 0;
}
```

```
#define TAILLE_MAX 100

int rechercher(int tab[], int v)
{
    int i=0;
    while ((i<TAILLE_MAX) && (tab[i]!=v))
    {
        Exemple avec && "intelligent"
        ++i;
    }
    return i;
}

int main()
{
    int tableau[TAILLE_MAX];
    printf("%d", rechercher(tableau, 3));
    return 0;
}
```

Effets de bord ?

```
/* float tab[50]; */  
  
printf("Saisir un indice et sa valeur");  
scanf("%d %f", &i, &tab[i]);
```

```
int rechercher(int tab[], int v)  
{  
    int i=0;  
  
    while ((i<TAILLE_MAX) && (tab[i++]!=v));  
  
    return i;  
}
```

Fonction au même comportement
que celle du transparent précédent ?

Le hasard

- Pas de hasard, mais du pseudo-aléatoire
- Fichier `stdlib.h`
- `rand()` renvoie un entier compris entre 0 et `RAND_MAX`
- `srand()` permet d'initialiser le germe ou graine (*seed*) du générateur de nombres pseudo-aléatoires
- Il est **normal** d'avoir la même suite de nombres si l'on fixe le germe (débuguage).


```
/* afficher 10 nombres entre 0 et 1 */  
int i;  
srand(time(0));  
for(i=0; i<10; ++i)  
    printf("%f", rand() / (float)RAND_MAX);
```

```
/* afficher 10 nombres entre 0 et 99 */  
int i;  
srand(time(0));  
for(i=0; i<10; ++i)  
    printf("%d", rand() % 100);
```

Introduction d'un biais statistique

Avoir des nombres différents à chaque exécution ?

```
#define GERME -1 // a changer au cas où

int main()
{
    int germe = GERME;

    germe = (germe==-1)?time(0):germe;
    // utiliser qqch qui change a l execution
    srand(germe);
    printf("Pour debug, germe :%d\n");

    return 0;
}
```

Type du résultat ?

```
rand() / RAND_MAX  
rand() / (float) RAND_MAX;  
(float) rand() / (float) RAND_MAX;  
((float) rand()) / (float) RAND_MAX;  
((float) rand()) / RAND_MAX;  
(float) rand() / RAND_MAX;  
(float) (rand() / RAND_MAX) ;  
rand() / (RAND_MAX + .0) ;  
(rand() + .0) / RAND_MAX;
```

Fonctions mathématiques

- Bibliothèque standard `math.h`
 - `#include <math.h>`
- Edition des liens
 - Fichier `libm.a` ou `libm.so` ou `libm-XX-YY.sp`
- Modifier la ligne de compilation
 - Options `-lm`

`cos()` `sin()` `tan()` `acos()` `asin()` `atan()`

`exp()` `log()` `pow()` `sqrt()`

`fabs()` `floor()` `ceil()`

Table ASCII

- *American Standard Code for Information Interchange*
- 1 caractère = 1 code
- Codes inférieurs à 128 normalisés

	BC	TAB		CR		ESC		ESP	!
	8	9		13		27		32	33

0	1		9	:					@
48	49		57	58					64

A	B	C						Y	Z
65	66	67						89	90

a	b	c						y	z
97	98	99						121	122

Chaîne de caractères (1)

1. Tableau de caractères

- Taille maximale
- Taille effective `strlen()`
- Table `ASCII`

`char s[8]`

2. Caractère de fin de chaîne

- Caractère `'\0'` – Zéro numérique

0	1	2	3	4	5	6	7
Z	Z	1	\0				
90	90	49	0	?	?	?	?

Comparer des chaînes de caractères

Ordre lexicographique

- Ordre d'affichage des résultats d'un *ls* ?
 - "Abc" < > "abc"
 - "abc" < > "aBc"
 - "abc" < > "abcd"
 - "abc" < > "Abcd"

Fonction de comparaison de chaînes ?

```
#include<string.h>
```

Chaîne de caractères (2)

- **Copier** deux chaînes de caractères

```
strcpy(destination, source);
```

⇒ **Affectation (=) ne marche pas**

- **Comparer** deux chaînes de caractères

```
strcmp(chaine1, chaine2);
```

⇒ **Comparaison (==) ne marche pas**

⇒ Renvoie la première différence

- **Concaténer** deux chaînes de caractères

```
strcat(destination, source);
```



```
int main()
{
    char chaine[255];
    char sauve [255];

    printf("saisir du texte");
    scanf("%s", chaine);

    printf("longueur de la chaine: %lu",
           strlen(chaine));

    strcpy(sauve, chaine);

    if (strcmp(chaine, "fin")==0)
        printf("Voulez-vous quitter ?");
    return 0;
}
```

Il existe beaucoup d'autres
fonctions sur les chaines de
caractères
puts, gets ...

%u ou %lu
long unsigned int

Chaines vs tableaux

```
int main()  
{  
    char s1[10];  
    char s2[10];  
  
    s1[0] = s2[0] = 'Z';  
    s1[1] = s2[1] = 'z';  
    s1[2] = s2[2] = '1';  
    s2[3] = '\\0';          /* ou = 0 */  
  
}
```

```
printf("%s", s1);
```

```
printf("%s", s2);
```

```
printf("%u", strlen(s1));
```

```
printf("%u", strlen(s2));
```

```

int main()
{
    char s[100];

    scanf("%s", s);    /* "Biere" */
    printf("%s", s);

    s[0] = 'F';
    printf("%s", s);
    s[4] = s[4] - 'a' + 'A';
    printf("%s", s);
    s[5] = 's';
    printf("%s", s);
    s[6] = 0;
    printf("%s", s);
    s[4] = 0;
    printf("%s", s);
}

```

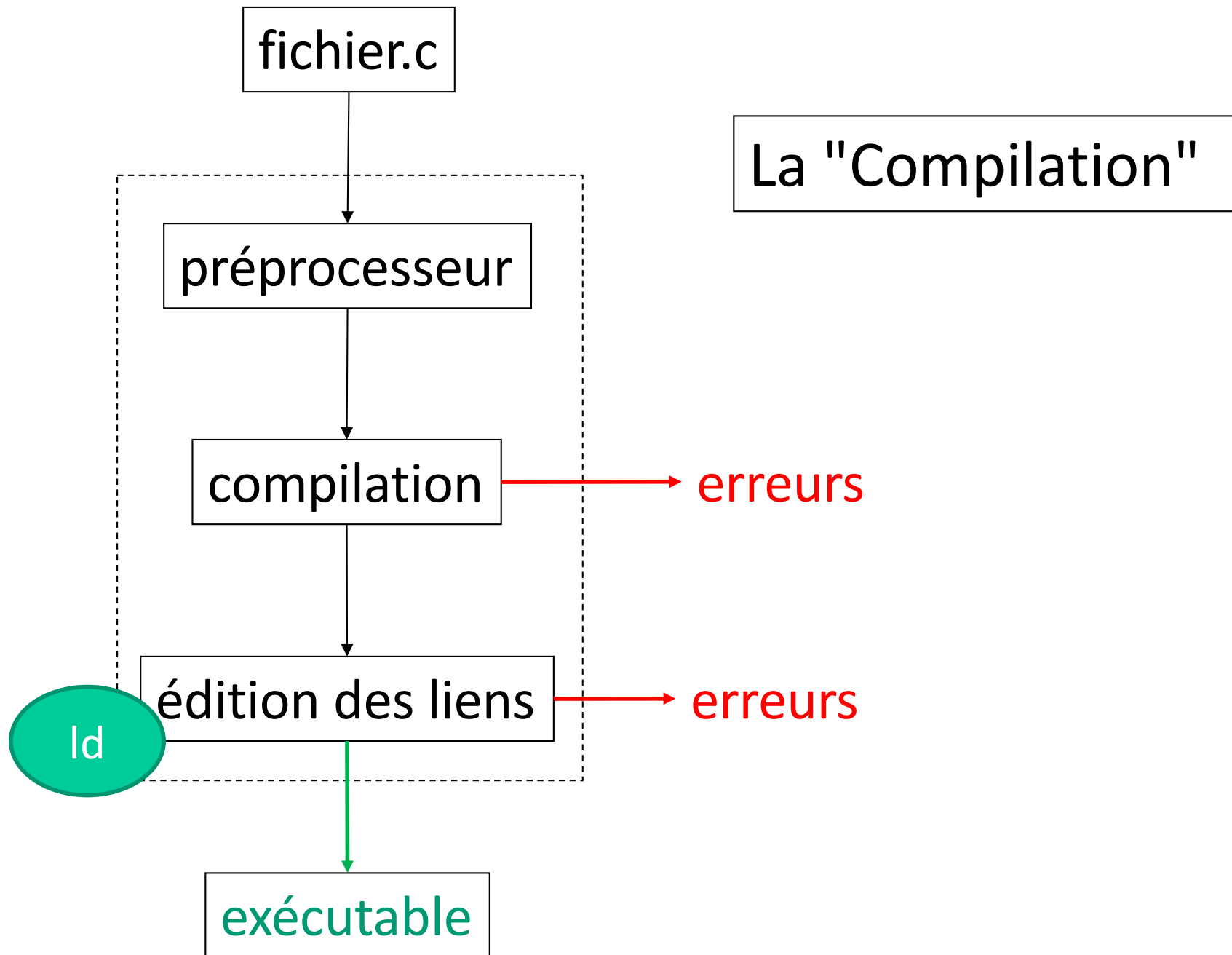
Calculer la taille d'une
chaîne de caractères?

Transformer une chaîne
en majuscules
ou en minuscules ?

Coder strcmp ?

Détails sur la compilation

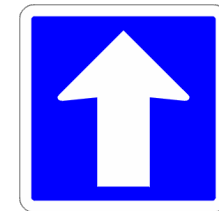
- Préprocesseur
 - Directives # (macro, tests, include)
 - Substitution de texte
- Compilation
 - Liste des erreurs
 - Vérification des syntaxes et des déclarations
- Edition des liens
 - Recherche des symboles



Paramètre de main()

```
int main(int argc, char ** argv);
```

`argv` contient les différents éléments donnés par la ligne de commande (revu plus tard)



```
int main(int argc, char * argv[]);
```

```
printf("Nom du programme : %s", argv[0]);  
if (argc > 1)  
    printf("Parametre 1 : %s", argv[1]);
```

```
$ ./prog seed=3527
```

```
int main(int argc, char *argv[])
{
    int germe = time(0);

    if (argc>1)
        if (strncmp(argv[1], "seed=", 5) ==0)
            germe = atoi(argv[1]+5);

    srand(germe);

    return 0;
}
```

Les fichiers

- Fichiers textes
 - Compréhensible (ou presque) par l'humain
 - `more, vi`
- Fichiers binaires
 - Le reste ;-)

Un fichier...

- Informations
 - Nom et type du fichier
 - Mode d'ouverture (Lecture/Ecriture)
 - Emplacement sur le disque

```
FILE * fopen(char[] nom, char[] mode);  
void fclose(FILE *);  
int fscanf(FILE * fic, char[] format, ...);  
int fprintf(FILE * fic, char[] format, ...);
```

- Fichiers standards prédéfinis
 - stdin
 - stdout
 - stderr

```

int main()
{
    FILE * fichier; /* désolé !!! */
    char nom[20] = "toto.txt";

    fichier = fopen(nom, "w");
    /* ouverture en écriture */
    if (fichier==0)
        printf("Problème à l'ouverture\n");
    else
    {
        fprintf(fichier, "%s\n", nom);
        /* meme utilisation que printf */
        fclose(fichier);
    }
    return 0;
}

```

Ouverture en écriture ...

```

int main() {
    FILE * fichier;
    char nom[20] = "toto.txt";
    char temp[255];

    fichier = fopen(nom, "r"); /* lecture */
    if (fichier==0)
        printf("Problème à l'ouverture");
    else {
        fscanf(fichier, "%s", temp);
        while(!feof(fichier)) {
            printf("%s", temp);
            fscanf(fichier, "%s", temp);
        }
        fclose(fichier);
    }
    return 0;
}

```

Utilisation d'un
fichier en lecture

Est-ce que le fichier de
texte se termine par une
ligne vide ou non ?

Kecepatan tildonc ? (1)

```
int main()  
{  
    int i;  
  
    for(i=0; i<10; ++i);  
    {  
        printf("%d", i);  
    }  
    return 0;  
}
```

Kecepatas tildonc ? (2)

```
/* afficher 10 nombres entre 0 et 1 */  
int i;  
srand(time(0));  
for(i=0; i<10; ++i)  
    printf("%f", rand()/RAND_MAX);
```

```
/* saisir 10 nombres */  
int i;  
float tab[5];  
  
for(i=0; i<10; ++i)  
    scanf("%d", tab[i]);
```

Kecepatas tildonc ? (3)

```
/* afficher 10 nombres entre 0 et 1 */  
int i;  
srand(time(0));  
for(i=0; i<10; ++i)  
    printf("%d", rand() / (float)RAND_MAX);
```

Documenter ?

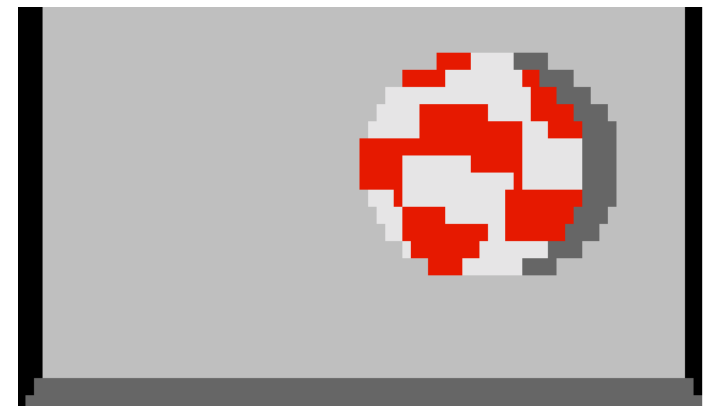
- Les fichiers .c ou .h
 - Ce que fait le fichier en général, la date, l'auteur, les maj
 - Les options de compilation
- Les fonctions
 - Entrées/sorties
 - Ce qui est fait
- Les variables
 - Obligatoire pour les variables globales
- Les entrées/sorties
- Les astuces de codes
 - Mettre éventuellement des "références" sur des algos
 - Commenter les effets de bord si besoin

Ce qu'il ne faut pas faire

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <sys/ioctl.h>

main() {
    short a[4];ioctl
    (0,TIOCGWINSZ,&a);int
    b,c,d=*a,e=a[1];float f,g,
    h,i=d/2+d%2+1,j=d/5-1,k=0,l=e/
    2,m=d/4,n=.01*e,o=0,p=.1;while (
    printf("\x1b[H\x1B[?25l"),!usleep(
    79383)){for (b=c=0;h=2*(m-c)/i,f=-
    .3*(g=(1-b)/i)+.954*h,c<d;c+=(b==
    b%e)==0)printf("\x1B[%dm ",g*g>1-h
    *h?c>d-j?b<d-c||d-c>e-b?40:100:b<j
    ||b>e-j?40:g*(g+.6)+.09+h*h<1?100:
    47:((int)(9-k+(.954*g+.3*h)/sqrt
    (1-f*f))+(int)(2+f*2))%2==0?107
    :101);k+=p,m+=o,o=m>d-2*j?
    -.04*d:o+.002*d;n=(1+=
    n)<i||l>e-i?p=-p
    ,-n:n;}}
```

Peter Eastman
IOCCC 2011




```
/* PROGRAMME prog.c
   AUTEUR : loic
   VERSION/DATE : 2020
   COMPILATION : -O2

   Ce fichier contient l'implémentation de ...

*/

/* taille maximale d'un tableau statique */
#define TAILLE_MAX 1000

/* variable globale erreur
   retourne un code d'erreur si différent de 0 */
int erreur;

/* suite sur le transparent suivant */
```

```

/*
FONCTION rechercher

renvoie l'indice du tableau tab qui contient
valeur sinon renvoie TAILLE_MAX si non trouvé
utilise l'algorithme capillotracté vu en cours

ENTREE : tab - tableau d'entier dans lequel on
recherche valeur
ENTREE : valeur - valeur à rechercher dans le
tableau
RETOURNE : l'indice du tableau

*/
int rechercher(int tab[], int valeur)
{
    ...

    ++i; // pas commenter ça quand même
    ...
}

```

Guide de bonnes pratiques

- Respecter le guide de style
 - Indentation
 - Alignement des accolades
 - Noms homogènes + conventions
- Suivre les recommandations de l'ANSSI
- Commenter (+ *doxygen*)
- Éviter les effets de bord
- Respecter les règles de Génie Logiciel
 - Return unique
 - Maintenance facilitée
- Tester au fur et à mesure
- Diviser pour mieux régner

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

```
int a;
```

```
void f1(float a)
{
    printf("%f\n", a);
}
```

Masquage de variable

```
void main()
{
    a = 3;
    f1(2.000);
    printf("%d\n", a);
}
```

Kecepasstildonc ? (4)

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

```
int a;
```

```
void f1(int a)
```

```
{
```

```
    a=2;
```

```
}
```

```
void main()
```

```
{
```

```
    a = 3;
```

```
    f1(a);
```

```
    printf("%d", a);
```

```
}
```

Passage des paramètres par valeur
Cet exercice marcherait avec un tableau

Kecepasstildonc ? (5)



Bibliographie

- La programmation sous UNIX, Jean-Marie RIFFLET, Ediscience, 2003
- The C Programming Language, KERNIGHAN, RITCHIE, Prentice-Hall, 1989
- La programmation sécurisée an C, ANSSI, 2020

- Exemples PYTHON
- Compléter les dessins de la compilation ?
- Les options
- Les fonctions de caractères
- Exemple et codage de strcmp à mettre
- Le passage de min en maj
- La merde avec strlen et %u %lu

Tableaux et chaînes de caractères

- Opérations basiques
- Passer en minuscules ou en majuscules
- Longueur de la chaîne
- Copie ?
- Comparaison ?
- Ordre lexicographique
- Longueur pas importante


```
for (i=0; i<10; ++i)
    printf("i");
    printf("%d, i");
```

```
char c = "A";
```