Septembre 2026

C1



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

CLERMONT AUVERGNE INP Isima

Plan



- Premier programme
- Variables
- Mémoire
- Fonctions prédéfinies : printf, scanf
- Fonctions, récursivité
- Tableaux
- Constante symbolique, compilation
- Chaines de caractères
- Fichiers texte


```c
#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


```python
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;
}
```

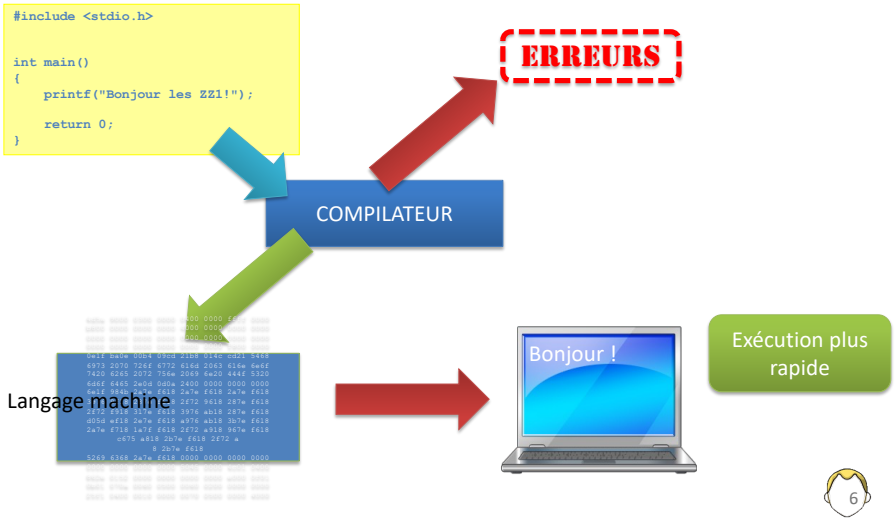
zz1.c

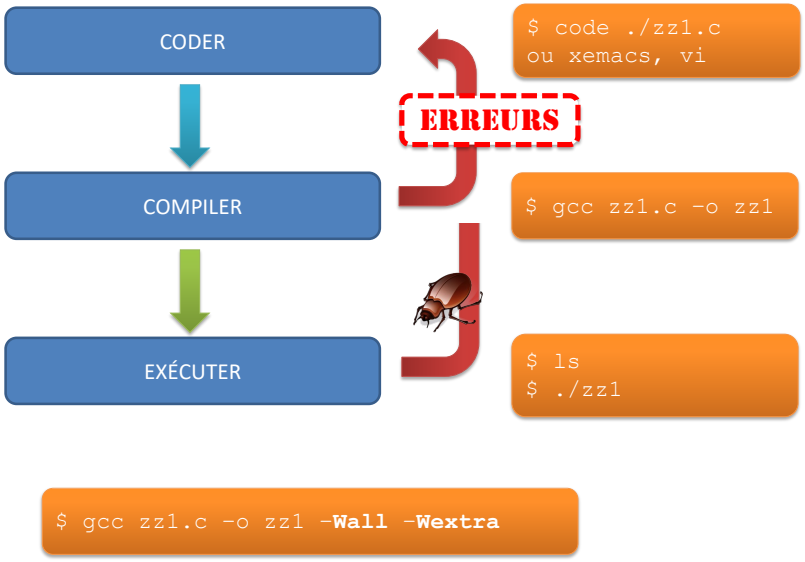
Exécution ligne par ligne

Découverte des erreurs au fil de l'eau

est interprété

Langage compilé !






```
$ 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

monoligne

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;
```


Variable locale ou globale ?

```
#include <stdio.h>

int i = 0;          GLOBALE

int main()
{                   LOCALE
   float f = 3.0;
   printf("Bonjour les ZZ1!");

   return 0;
}

int autre()                 ¡?f?
{
   printf("autre fonction");
   return 1;
}
```

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 : =, ++, --, +=, -=, *=, /=

Pas de x puissance y par ex

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 = $1 * 10^2 + 3 * 10 + 8$ (décimal)

= 1 0 0 0 1 0 1 0 (binaire)

= 8 * 16 + 10 = 8 A (hexadecimal)

- « 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

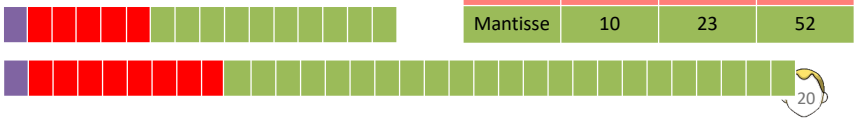
- 123,125 = $1,23125 * 10^2$

  = 1111011,001 = $1,111011001 * 2^6$

  = signe * mantisse * 2 exposant

- Codage : ±0 ±∞ 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
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*
- 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, …);
```

- Chaine
  - Chaine 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;
}
```



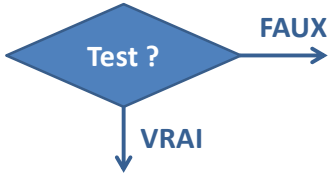
Formatage ?

```
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 O 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

Saisie mémorisée

mémoire tampon (*buffer*)

```
#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;
}
```

Autre exemple disponible sur https://perso.isima.fr/loic

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)
{
  int r = v2;

  if (v1 > v2)
    r = v1;

  return r;
} /* GL compliant */
```

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


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;
}
```

max est connue avant d'être utilisée

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) ?

```c
#include <stdio.h>

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

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

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

Passage du paramètre par valeur

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

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


Récursivité

$$\begin{cases} n! = n * (n-1)! \\ 0! = 1 \end{cases}$$

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

Prototype

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

Toujours vérifier les paramètres des fonctions en début de fonction

```c
int fact(int n) {
   int r = 1;

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

Tableau statique à une dimension

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

```c
float   tableau1[1000];
```

1000 éléments non initialisé

```c
int     tableau2[]   = {0,1,2,3};
```

Dimension 4, initialisé

```c
int     tableau3[10] = { 0 };
```

Dimension 10, initialisé

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


Utilisation d'un tableau

```c
int main()
{
  float tableau[100] = { 0.0 };
  int i;

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

  // affichage du tableau ?

  for(i=0; i< 100; ++i)
     printf("%f\n",  tableau[i]);
  return 0;
}
```

Pas de fonction prédéfinie

Pas de vérification des bornes

Tableau et fonction

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

Taille du tableau

```
int main()
{
  float tableau[100] = { 0.0 };

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

  afficher(tableau, 100);

  return 0;
}
```

Seule valeur utilisable

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;
}
```

Constante symbolique et tableau

- Taille du tableau fixée à la compilation donnée par une constante
- Changement de taille => recompilation

```
#define N 10

float tableau4[N];
```

```
#define N 100

float tableau4[N];
```

```
int i;
for(i=0; i< N; ++i)
     printf("%d",tableau4[i] = 1.0);
```

Tableau de taille variable ?

- Tableau dont la taille varie pendant l'exécution
  - *Variable Length Array*

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



On n'en veut 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;
   while ((i<TAILLE_MAX) && (tableau[i]!=v))
   {
      ++i;
   }
   return i;
}
```

Exemple avec && "intelligent"

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

```
#define TAILLE_MAX 100

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

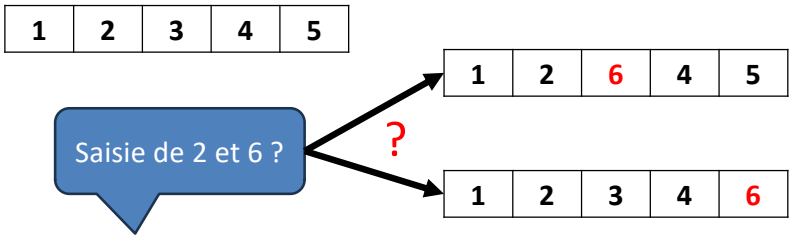
Exemple avec && "intelligent"

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

Effets de bord ? (1)

```
int   tab[5] = { 1, 2, 3, 4, 5 };
int   i      = 4;

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



Effets de bord ? (2)

```
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 56 ?

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 (déboguage).


```
/* 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

```c
int main()
{
  // srand(10);
  printf("Alea :%d\n", rand());

  return 0;
}
```

```c
int main()
{
  srand(time(0));
  printf("Alea :%d\n", rand());

  return 0;
}
```


Avoir des nombres différents à chaque exécution ?

```c
#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;
}
```

Fonctions mathématiques

- Bibliothèque standard `math.h`
  - `#include <math.h>`
- Edition des liens
  - Fichier lib**m**.a ou lib**m**.so ou lib**m**-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. Tableau de caractères
   - Taille maximale
   - Taille effective `strlen()`
   - Table `ASCII`
2. Caractère de fin de chaine
   - Caractère '\0' – Zéro numérique

```
char s[8]
```

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


```
int main()
{
  char vide   [] =  "";
  char un     [] = "un";
  char deux [40] = "deux\0trois";
  char quoi [30];

  printf("Chaine 1 : %s\n", vide);
  printf("Chaine 2 : %s\n", un);
  printf("Chaine 3 : %s\n", deux);
  printf("Chaine 4 : %s\n", quoi);

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

  return 0;
}
```

Taille réservée en mémoire ?


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

strlen()

- Longueur d'une **chaîne de caractère**

```
strlen(chaine);
```

Un entier positif size_t
(%u ou %lu) [long] unsigned int

Parcours de la chaine de caractères jusqu'à trouver le caractère de fin

- Pas de vérification de la présence du caractère de fin
- Pas de test sur le taille "réelle" de la chaine

Comparer des chaines de caractères
Ordre lexicographique

- Ordre d'affichage des résultats d'un *ls* ?

- `"Abc" < > "abc"`
- `"abc" < > "aBc"`
- `"abc" < > "abcd"`
- `"abc" < > "Abcd"`

Fonction de comparaison de chaines ?

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

Comparaison de chaînes

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

- Comparaison des deux **chaines de caractères**
  - Se termine au premier caractère différent
  - Renvoie 0 si les chaînes sont égales,
    Une valeur négative ou positive sinon
- L'opérateur == n'est pas utilisable


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

Copie de chaîne

- Copie caractère par caractère de la chaine *source* vers la chaine *destination*

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

  - Source est une chaine de caractères
  - La destination doit avoir une taille suffisante
- L'opérateur = n'est pas utilisable


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

Concaténation de chaînes

- Ajout de la chaîne *source* à la fin de la chaine *destination*

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

  - Source est une chaine de caractères
  - La destination doit avoir une taille suffisante
- L'opérateur + n'est pas utilisable


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

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

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

  strcpy(sauve, chaine);

  if (strcmp(chaine, "fin")==0)
     printf("Vous avez tape -fin-");
  return 0;
}
```

%u ou %lu

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

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("%lu", strlen(s1));
```

```
printf("%lu", 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);
}
```


Que faut-il savoir faire ?

- Savoir coder *strlen* pour être capable de s'en passer
  - Algorithmique
- Traduire une chaine minuscule en majuscule
- Savoir coder *strcmp* et *strcpy*


```
int compter_a(char s[])
{
  int r = 0;

  for (int i = 0; s[i] != '\n'; ++i)
  {
     if (s[i] == 'a') r+=1;
  }

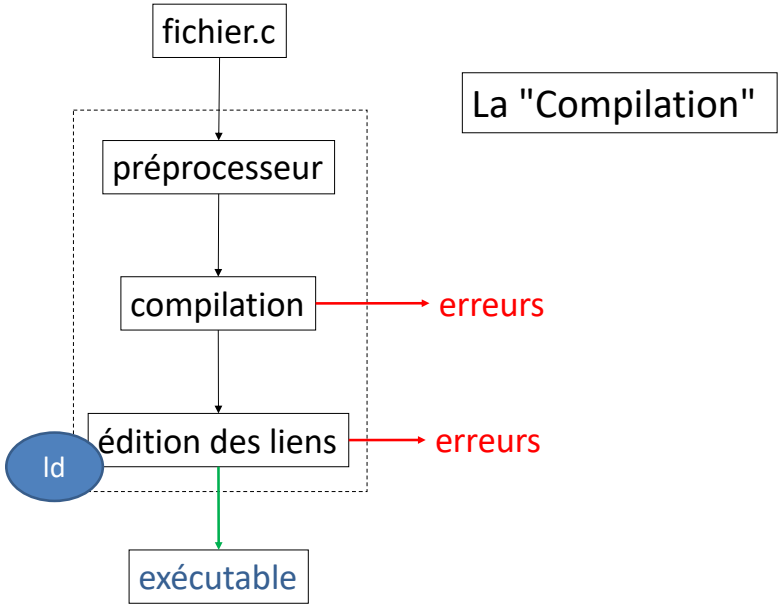
  return r;
}
```

Fin de chaine ?

```
for (int i = 0; s[i] != '\0'; ++i)
{
   if (s[i] == 'a') r+=1;
}
```

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 ?

Documenter ?

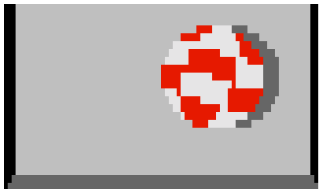
- Les fichiers .c ou .h
  - Ce que fait le fichier en général, la date, l'auteur, les maj
  - Les options de compilation
- Les fonctions
  - Entrées/sorties
  - Ce qui est fait
- Les variables
  - Obligatoire pour les variables globales
- Les entrées/sorties
- Les astuces de codes
  - Mettre éventuellement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


```
#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=(l-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=(l+=
         n)<i||l>e-i?p=-p
             ,-n:n;}}
```

Ce qu'il ne faut pas faire ….

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



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
- La norme ISO …