Langage C

Cours Magistral

Licence Informatique & MIO

Pr Edouard Ngor SARR
Enseignant-Chercheur
Université Assane SECK de Ziguinchor (UASZ)
Ziguinchor-Sénégal
edouard-ngor.sarr@univ-zig.sn

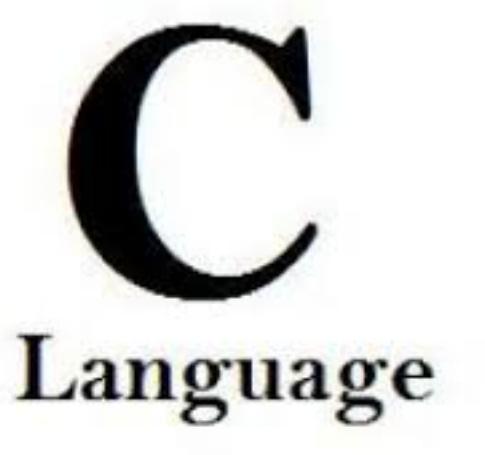
Oct 2025

Définitions

- **Programmation = Développement = Codage**
- **Instruction**
  - Ordre donné à l'ordinateur pour effectué une tache
  - Se termine toujours par un point virgule (;)
  - Exemple: Int x=5;
- **Programme**
  - Ensemble cohérant d'instructions pour résoudre un problème.
  - Chacune de ces instructions permet de faire une tache et la somme permet de résoudre le problème.
  - Exemple: Programme Somme de deux valeurs
  - Méthode : Fonction ou procédure
- Applications
  - Un ou plusieurs programme
  - Permet d'effectuer un travail (resoudre un ensemble de problemes)
  - Exemple: Application calculatrice

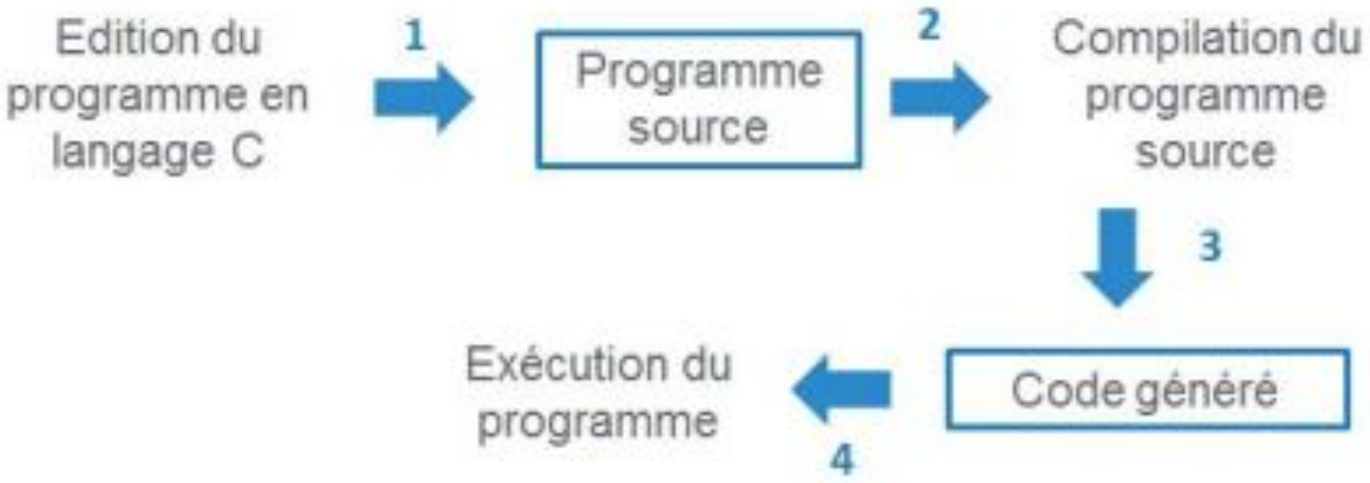
Le langage C

- Langage de programmation utilisé pour éditer et exécuter des programmes
- Conçu en 1972 par Dennis Richie et Ken Thompson
- Langage de Bas Niveau
- A inspiré beaucoup d'autres langages tels que le JAVA ou C++
- Langage Procédurale c'est-à-dire non orienté Objet
- Langage *compilé* (par opposition aux langages interprétés tels que le JAVA)
- Les fichiers source sont suffixés par .c



Exécution d'un programme en C

- **Cycle de vie d'un programme**



Structure d'un programme en C

- **04 grandes Parties**
  - Zone des directives ou déclaration des bibliothèques avec INCLUDE
  - Zone de déclaration des variables et constantes globales
  - Zone de déclaration (annonce) des fonctions
  - Corps du programme qui lui est composé:
    - Méthode principale main
      - Cette méthode est la partie du programme qui montre du point de vue séquentiel tout ce qui sera exécute.
      - Contient :
        - » les variables et constantes locales
        - » L'appel des autres méthodes du programme
    - Autres méthodes :
      - Procédure (void)
      - Ponctions

Structure d'un programme en C: Exemples

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

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Directives de préprocesseur

Instructions

Fonction

```c
#include <stdio.h>
#include "fonctions.h"
/* déclaration des fonctions */
main( )
{
/* déclaration des variables */
/* instructions; */
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int addition(int a, int b);
int soustraction(int a, int b);
int multiplication(int a, int b);
double division(int a, int b);
int main() {
   int x=12, y=4;
   printf("Soit les nombre %d et %d\n", x, y);
   printf("leur somme est: %d ", addition(x, y));
   printf("leur difference est: %d ", soustraction(x, y));
   printf("leur produit est: %d ", multiplication(x, y));
   printf("leur quotien est: %d ", division(x, y));
   return 0;
}
int addition(int a, int b) { return a + b;}
int soustraction(int a, int b) { return a - b; }
int multiplication(int a, int b){    return a*b; }
double division(int a, int b){  return a/b; }
```

Les identificateurs

- Une suite de caractères parmi : les lettres (minuscules ou majuscules, mais non accentuées),
- Le rôle d'un identificateur est de donner un nom à une entité du programme :un nom de variable, constantes ou de fonction
- **Règles**:
  - suite de caractères alphanumériques ou caractère espérés par un _ donc pas d'espace
  - premier caractère doit être alphabétique ou caractère surligné _
  - Longueur des identificateurs dépendante du compilateur
- **Mots réservés**
  - Un certain nombre de mots, appelés *mots-clefs*, sont réservés pour le langage lui-même donc non utilisables

auto	const	double	float	int	short	struct	unsigned
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static	union	while

Les types de données

- Un type est un ensemble de valeurs que peut prendre une variable.
- Il y a des types prédéfinis et des types qui peuvent être définis par le programmeur.

Types simples prédéfinis en C

Type	Signification	Représentation système	
		Taille (bits)	Valeurs limites
int	Entier	16	-32768 à 32767
short (ou short int)	Entier	16	-32768 à 32767
long (ou long int)	Entier en double longueur	32	-2147483648 à 2147483647
char	Caractère	8	
float (ou short float)	Réel	32	$\pm 10^{-37}$ à $\pm 10^{38}$
double(ou long float)	Réel en double précision	64	$\pm 10^{-307}$ à $\pm 10^{308}$
long double	Réel en très grande précision	80	$\pm 10^{-4932}$ à $\pm 10^{4932}$
unsigned	Non signé (positif)	16	0 à 65535

Les commentaires

- Un texte en langage humaine dans un code source
- Permet de donner son opinion ou d'expliciter une partie du programme
- Débute par /* et se termine par */.
- Par exemple:

```
main( ) {
    printf("bonjour MIO 2"); /* ce programme affiche bonjour*/
}
```

/* commentaires */	/* commentaires */	/* * commentaires * commentaires */

Les Variables

- **Syntaxe de base :**

  type nomVariable ;

  – Exemple:

```
int i; /* i est une variable de type entier */
float j,k; /* j et k sont des variables de type réel */
char c; /* c est une variable de type caractère */
```

- Une valeur initiale peut être affectée à une variable dès la déclaration

```
int i, j=3, k; /* seul j est initialisé à 3*/
float f=1.2344 ; /* f est initialisé à 1,2344*/
```

- **On peut déclarer plusieurs variables d'un même type:**
  – Exemple:

```
int a, b, c;
```

- **On peut initialiser une variable lors de sa déclaration:**
  – Exemple:

```
float pi = 3.14;
```

Les Constantes

- Une constante est une donnée dont la valeur ne varie pas lors de l'exécution du programme.
- Syntaxe :

const Type Identificateur = Valeur ;

- Ou bien en utilisant la directive define avant d'entrer dans le main:

#define Identificateur Valeur

Déclaration de constantes

```
1-   main( )
     {
     const  float   pi=3.14;          /*déclare la constante pi avec const*/
     printf("pi égale à %f",pi);      /*affiche la valeur de pi*/
     }
2-   #define pi  3.14                 /*définit la constante pi avec define*/
     main( )
     {
     printf("pi égale à %f",pi);      /*affiche la valeur de pi*/
     }
```

Caractères spéciaux

- Des caractères spéciaux sont représentés à l'aide du méta-caractère \.

Liste des caractères spéciaux

Représentation	Signification
\0	Caractère NULL
\a	Bip (signal sonore)
\b	Espace arrière
\t	Tabulation
\n	Nouvelle ligne
\f	Nouvelle page
\r	Retour chariot
\"	Guillemet
\'	Apostrophe
\\	Antislash (\)
\ddd	Caractère ayant pour valeur ASCII octale ddd
\x hhh	Caractère ayant pour valeur ASCII hexadécimale ddd

Affichage avec printf

- L'instruction **printf** permet d'obtenir un affichage formaté à l'écran.
  Syntaxe :

  Printf ("texte et caractères de contrôle", , arg1) ;
- Chaque format d'affichage est introduit par le caractère % suivi d'un caractère qui indique le type de conversion.

Int x=5;

Printf ("X=%d", x) ;

Liste des formats d'affichage

Format d'affichage	Signification
%d	Conversion en décimal
%o	octal
%x	hexadécimal (0 à f)
%X	hexadécimal (0 à F)
%u	entier non signé
%c	caractère
%s	chaîne de caractères
%l	long ou double
%L	long double
%e	sous forme m.nnnexx
%E	sous forme m.nnnExx
%f	sous forme mm.nn
%g	Semblable à e ou f selon la valeur à afficher

Affichage avec printf

```
main( )
{

int i=2,k=5;                          /*i et k entiers initialisés à 2 et 5*/
float j=3.5;                          /*j réel initialisé à 3.5*/

printf("Donnez le prix unitaire");    /*le programme affiche
                                        Donnez le prix unitaire */

printf("Donnez le prix unitaire \n"); /*le programme affiche Donnez le prix
                                        unitaire et retourne à la ligne (\n)*/

printf("la valeur de i est %d\n ",i); /*le programme affiche la valeur
                                       de i est 2 et retourne à la ligne*/

printf("i=%d      j=%f",i,j);         /*le programme affiche
                                        i=2        j=3.5*/
```

Lecture de données avec scanf

- L'instruction scanf effectue la lecture des variables.
- Syntaxe :

  Scanf ("formats d'affichage", variable1, variable2,…,variablen) ;
- Remarque : Seules les variables scalaires (entiers, réels et caractères) doivent être précédées de &.

Lecture

```
#include <stdio.h>
main( )
{
int i;                  /*i entier*/
float k;                /*k réel*/
char m;                 /* m caractère*/
scanf("%d",&i);         /*le programme lit une valeur entière et l'affecte à i*/
scanf("%d%f",&i,&k);    /*le programme lit une valeur entière de i
                          puis une valeur réelle de k*/
scanf("%c",&m);         /*le programme lit un caractère et l'affecte à
                          la  variable m*/
}
```

Operateurs

- **Operateurs arithmétiques**

+ addition	x=y+z;
- soustraction	x=z-y;
* multiplication	t=vs* s;
/ division	v=a/u;
% reste de la division entière	z=e%b;

- Operateurs de pas

  += -= *= /=

- *Exemples*

i = i + 20;	i += 20;
i = i - 20;	i -= 20;
i = i * 20;	i *= 20;
i = i / 20;	i /= 20;

Operateurs

- **Operateurs logiques**
- Les opérateurs logiques sont, par ordre décroissant de priorité :

!	Non logique
> >= < <=	Test de supériorité et d'infériorité
== et !=	Test d'égalité et d'inégalité
&& et ||	ET et OU logique

Opérateurs logiques

```
#include <stdio.h>
main( )
{
int a,b,c;                          /*a,b et c entiers*/
printf ("Introduire a, b et c : ");
scanf ("%d%d%d",&a,&b,&c);

if (a==b && b!=c)                   /*si a=b et b≠c affiche le message suivant*/
            printf("a égale à b et b différent de c\n");

if (!(a<b) || a==0)                 /*si a>=b ou a=0 affiche le message suivant*/
            printf("a est supérieure ou égale à b ou a égale à 0\n");
}
```

Operateurs

- **Post (i++) et pré-incrémentation (++i)**
  - Ils permettent d'incrémenter ou de décrémenter une variable.
  - Dans une opération d'affectation qui met en jeu l'opérateur de :
    - pré-incrémentation (pré-décrémentation), la variable est d'abord incrémentée (décrémentée) de 1. L'affectation est ensuite effectuée.
    - post-incrémentation (post-décrémentation), L'affectation (sans les ++ (--)) est effectuée avant l'incrémentation (décrémentation).

Soient i=3 et j=5,

Instruction	Equivalent	Résultats
i++;	i=i+1;	i=4
++i;	i=i+1;	i=4
i--;	i=i-1;	i=2
--i;	i=i-1;	i=2
i= ++j;	j=j+1; i=j;	j=6 et i=6
j= ++i + 5;	i=i+1; j=i+5;	i=4 et j=9
j= i++ + 5;	j=i+5; i=i+1;	j=8 et i=4;

Instructions conditionnelles: IF

- L'instruction if sélectionne le traitement (bloc d'instructions) à faire si une condition est vérifiée.
- Syntaxe :

```
if(condition)                    si condition est vrai (≠0)
        {                        on exécute
        bloc d'instructions 1;   bloc d'instructions 1
        }
else                             sinon
        {                        on exécute
        bloc d'instructions 2;   bloc d'instructions 2
        }
```

- Si le traitement à effectuer est constitué d'une seule instruction, il est possible d'omettre les accolades.

Instructions conditionnelles: IF

- **If ELSEIF ELSE**
  - En combinant plusieurs structures **if - else** en une expression nous obtenons une structure qui est très courante pour prendre des décisions entre plusieurs alternatives:

```
#include <stdio.h>
main()
{
 int A,B;
 printf("Entrez deux nombres entiers :");
 scanf("%i %i", &A, &B);
 if (A > B)
     printf("%i est plus grand que %i\n", A, B);
 else if (A < B)
     printf("%i est plus petit que %i\n", A, B);
 else
     printf("%i est égal à %i\n", A, B);
 return 0;
}
```

Instructions conditionnelles: IF

- **If imbriqués**
  - Il est possible d'imbriquer plusieurs structures if-else, cela permet de prendre des décisions entre plusieurs alternatives.
  - Mais, afin de gagner en lisibilité, on conseille d'adopter une écriture tabulée.

```
int A,B ;
printf("Entrer deux nombres entiers:\n") ;
scanf("%d %d",&A,&B);
if(A>B)
    printf("%d est plus grand que %d\n",A,B);
else
    if(A<B)
        printf("%d est plus petit que %d\n",A,B);
        else
            printf("%d est égal à %d\n",A,B);
```

Instructions conditionnelles: IF

- Exemple avec les IF

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

int main()
{
    char lettre;
    printf("Entrer une lettre de l'alphabet");
    scanf("%c", &lettre);

    if(lettre == 'a')
        printf("Première lettre de l'alphabet");
    else if(lettre = 'z')
        printf("Dernière lettre de l'alphabet")
    else
        printf("Ni la 1ère, ni la dernière");
}
```

Instructions conditionnelles: SWITCH

- **SELON en ALGO**
- S'il y a plus de deux choix possibles, l'instruction selon permet une facilité d'écriture

```
switch (expression )
  {case constante-1:
     liste d'instructions 1
     break;
   case constante-2:
     liste d'instructions 2
     break;
     ...
   case constante-n:
     liste d'instructions n
     break;
   default:
     liste d'instructions ¥
     break;
   }
```

Instructions Itératives: FOR

- Les *boucles* permettent de répéter une série d'instructions tant qu'une certaine condition n'est pas vérifiée.

```
for (initialisation ; condition ; incrémentation) {
    instructions à répéter
}
```

- Exemple

```
for (i = 0 ; i < 10 ; i = i + 1) {
   printf ("iteration %d\", i) ;
}
```

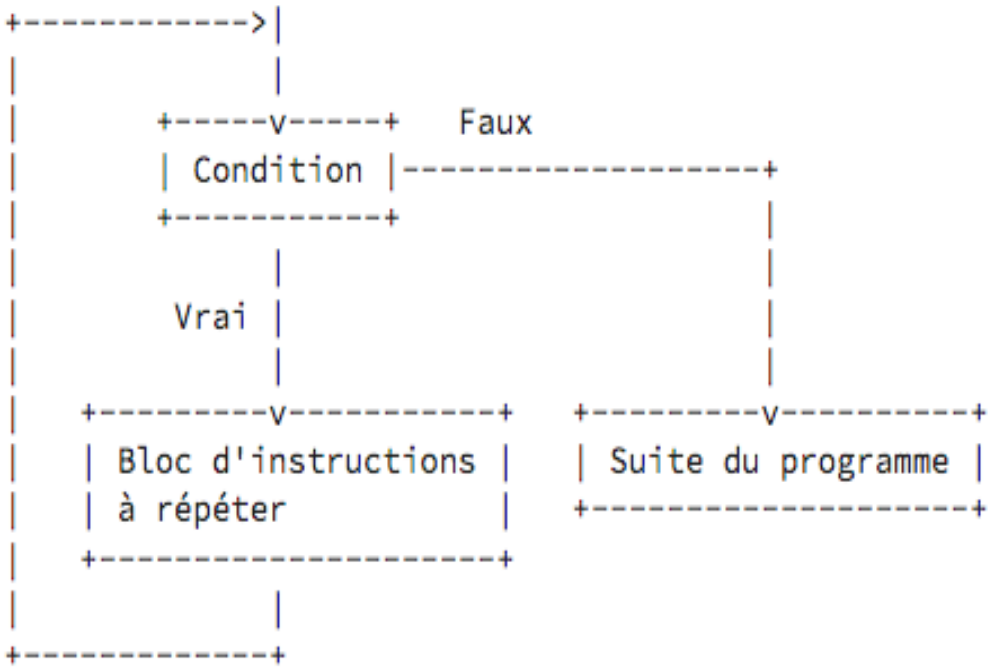
- Explications :
  - i = 0 initialise la variable i à 0. Il suppose l'existence préalable de la variable i (donc sa déclaration en amont).
  - i < 10 constitue la condition pour entrer dans la boucle (c'est la condition qui autorise l'exécution du bloc d'instructions de la boucle).
  - i = i + 1 correspond à l'incrémentation (on remplace parfois en i++ à la fin de chaque boucle).

Instructions Itératives: WHILE

- Tant que *expression* est vérifiée (*i.e.*, non nulle) alors *instruction* est exécutée.
- Si *expression* est nulle au départ, *instruction* ne sera jamais exécutée.

```
while (expression )
   instruction
```

```
i = 1;
while (i < 10)
   {
     printf("\n i = %d",i);
     i++;
   }
```



Instructions Itératives: DO WHILE

- Il peut arriver que l'on ne veuille effectuer le test de continuation qu'après avoir exécuté l'instruction.
- Dans ce cas, on utilise la boucle do---while.

```
do
  instruction
while (expression );
```

- Ici, *instruction* sera exécutée tant que *expression* est non nulle. Cela signifie donc que *instruction* est toujours exécutée au moins une fois.

```
int a;

do
  {
    printf("\n Entrez un entier entre 1 et 10 : ");
    scanf("%d",&a);
  }
while ((a <= 0) || (a > 10));
```

Instructions de branchement : BREAK

- break peut être employée à l'intérieur de n'importe quelle boucle.
  - Elle permet d'interrompre le déroulement de la boucle, et passe à la première instruction qui suit la boucle. En cas de boucles imbriquées, break fait sortir de la boucle la plus interne.

```
main()
{
  int i;
  for (i = 0; i < 5; i++)
    {
      printf("i = %d\n",i);
      if (i == 3)
        break;
    }
  printf("valeur de i a la sortie de la boucle = %d\n",i);
}
```

imprime à l'écran

```
i = 0
i = 1
i = 2
i = 3
valeur de i a la sortie de la boucle = 3
```

Les bibliothèques

- La pratique du C exige l'utilisation de bibliothèques de fonctions.
- Sont disponibles sous forme précompilées (.lib).
- Afin de pouvoir les utiliser, il faut inclure des fichiers en-tête (.h) dans nos programmes.
- Pour inclure les fichiers en-tête: #include
- **Exemple**

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

Nom	Rôle
stdio.h	Gestion des E/S.
stdlib.h	Gestion de la mémoire, conversions et fonctions systèmes.
string.h	Gestion des chaînes de caractères.
conio.h	Gestion de l'écran.
ctype.h	Manipulation de caractères.
math.h	Fonctions mathématiques.

Les fonctions sur les caractères <ctype.h>

- isalnum() teste si le caractère est alphanumérique.
- isalpha() teste si le caractère est alphabétique.
- iscntrl() teste si l'argument est un caractère de contrôle.
- isdigit() teste si le caractère est numérique.
- isgraph() teste si le caractère est visible.
- islower() teste si le caractère représente une lettre minuscule.
- isprint() teste si le caractère est imprimable.
- ispunct() teste si le caractère est un signe de ponctuation.
- isspace() teste si le caractère est un espacement.
- isupper() teste si le caractère représente une lettre majuscule.
- isxdigit() teste si le caractère est un chiffre héxadécimal valide.
- tolower() convertit le caractère en sa représentation minuscule.
- toupper() convertit le caractère en sa représentation majuscule.

Les fonctions sur les chaines <string.h>

- strcat() concaténation de deux chaînes de caractères.
- strncat() concaténation limitée en longueur de deux chaînes.
- strchr() recherche la première occurence d'un caractère dans un chaîne.
- strrchr() recherche la dernière occurence d'un caractère dans une chaîne.
- strcmp() comparaison lexicographique de deux chaînes de caractères.
- strcpy() recopie d'une chaîne de caractère.
- strncpy() recopie limitée en longueur.
- strlen() calcule la longueur d'une chaîne.
- strstr() calcule la position d'une sous-chaîne dans une chaîne.
- strtok() découpe une chaîne en lexèmes.

Les fonctions sur les nombres <math.h>

- cos() cosinus.
- sin() sinus.
- tan() tangente.
- exp() exponentielle ($\mathbf{e^x}$).
- log() logarithme népérien.
- log10() logarithme décimal.
- modf() décomposition en partie entière et décimale d'un réél.
- pow() élévation à la puissance ($\mathbf{x^y}$).
- sqrt() extraction de racine carrée.
- ceil() calcul de l'entier inférieur le plus proche (fonction plancher).
- fabs() valeur absolue.
- floor() calcul de l'entier supérieur le plus proche (fonction plafond).
- fmod() reste de la division.

TD 1 & TP 1
Voir fichier joint au cours

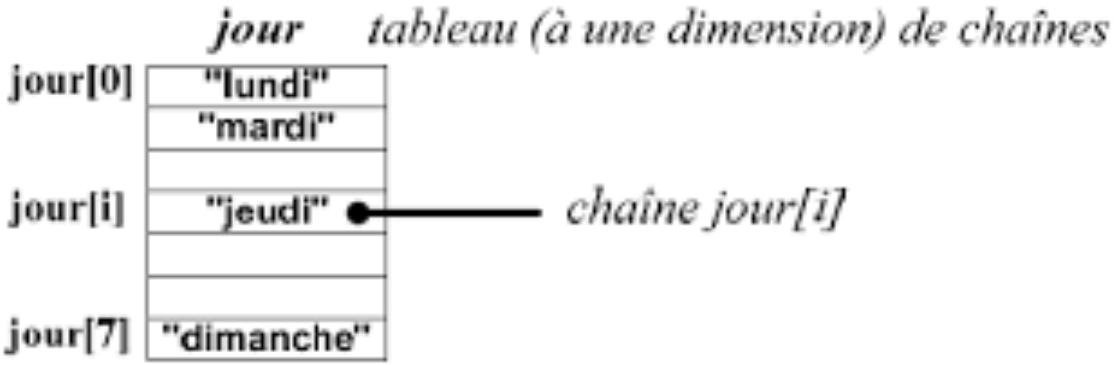
Chapitre 1
Les tableaux et les algorithmes de Tri

Introduction

- Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois.
- Or, pour de nombreuses données, comme cela est souvent le cas, des variables distinctes seraient beaucoup trop lourdes à gérer.
- Exemple:
  - Les notes de la MIO 2
  - La liste des étudiants de UASZ
  - La liste des employés de la SGBS
- C'est pourquoi:
  - Les langages de programmation tels que le langage C propose des structures de données permettant de stocker l'ensemble de ces données dans une unique « variable commune » : UN TABLEAU
- **Un tableau est une suite de cases (espace mémoire) de même taille destinée à stocker et manipuler un ensemble de valeurs de même type**

Introduction

- Deux formes de tableaux usuels
  - Tableaux à une dimension



  - Tableaux à N dimension dont N>=2

	colonne 0	colonne 1	colonne 2	colonne 3
ligne 0	tab[0][0]	tab[0][1]	tab[0][2]	tab[0][3]
ligne 1	tab[1][0]	tab[1][1]	tab[1][2]	tab[1][3]
ligne 2	tab[2][0]	tab[2][1]	tab[2][2]	tab[2][3]

Tableau à une dimension: Définitions

- Une variable structurée formée d'un nombre entier N de variables simples du même type
- 01 ligne + N colonnes ou vis versa (01 colonne + N lignes)
- Définitions
  - **La dimension ou la taille du tableau (N) :** Le nombre de cellules
  - **Indice i :** Le numéro de la cellule. Indice commence par 0 et s'arrête à (N-1).
    - Exemple : Un tableau de 10 elements (N=10):
      - Premier indice Tab[0] et dernier indice Tab[9]
  - **Tab[1]** désigne le contenu de la cellule numéro 1 c'est à dire la deuxièm

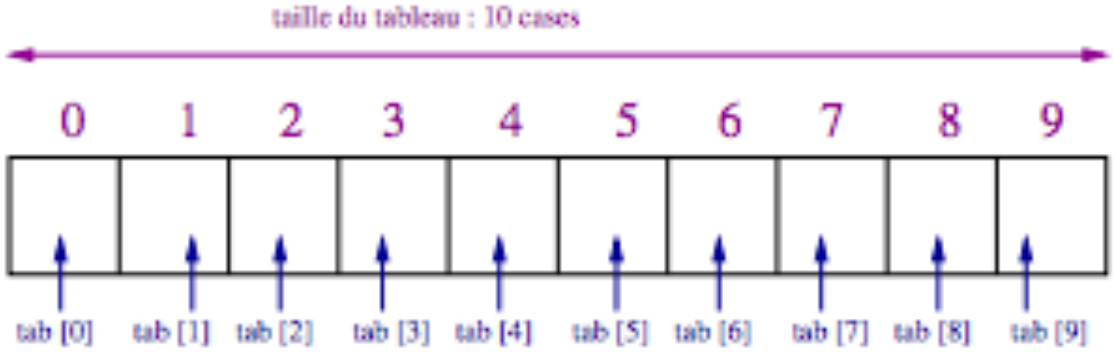


Tableau à une dimension: Déclaration

- **Déclaration nécessite trois informations :**
  - Le type des éléments du tableau ;
  - le nom du tableau (son identificateur) ;
  - la taille du tableau (le nombre d'éléments qui le composent)

type Nomtableau[Taille];

Exemple 1: on déclare un tableau nommé Tab1 de 04 entiers

```
int Tab1[4];
```

Exemple 2: on déclare un tableau TabP de 10 chaines de caractères

```
Char TabP[10];
```

- **Initialisation =** Action de déclarer un tableau et de lui donner ses premières valeurs

```
int tab[3] = { 12, 24, 73 };
```

Il est possible d'initialiser sans donner la taille du tableau

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

Tableau à une dimension: Accès aux éléments

- **Accéder aux éléments dans un tableau**
  - On utilise l'indice i
    - tab [i] avec i étant l'indice (numéro de la cellule)
  - Exemple d'utilisation :
    - tab [3] = 12;
      - Mettre la valeur 12 dans la case numéro 3 du tableau
    - printf ("%d", tab [3])
      - affiche la valeur numérique contenue dans la case numéro 3 du tableau
    - tab [3] = tab [3] + 2 ;
      - Ajoute 2 à la valeur contenue dans la case numéro 3 du tableau et le remttre dans la meme cellule tab [3] . Exemple: Si elle contenait auparavant la valeur 12, elle contiendra à présent la valeur 14.

Tableau à une dimension : Parcourir d'un tableaux

- On utilise une boucle FOR allant i=0 à M
  - 0 est le premier indice
  - M est l'indice maximal du tableau (M=N-1) ou N est le nombre d'éléments

Syntaxes:

for (i=0; i<=M; i++) { … }

for (i=0; i<N; i++) { … }

```
for (i = 0 ; i < 4 ; i++)
{
    printf("%d\n", tableau[i]);
}
```

```
int tableau[4], i = 0;

tableau[0] = 10;
tableau[1] = 23;
tableau[2] = 505;
tableau[3] = 8;

for (i = 0 ; i < 4 ; i++)
{
    printf("%d\n", tableau[i]);
}

return 0;
```

Tableau à une dimension: Exemple

```
/* Exemple pour tester l'utilisation des tableaux
*/
#include <stdio.h>

int main () {
    int tab [10] ; /* un tableau de 10 entiers est initialisee */
    int i ;

    /* On demande a l'utilisateur de remplir les 10 cases du tableau */
    for (i = 0 ; i < 10 ; i = i + 1) {
        printf ("Quelle valeur pour la case %d ?\n", i) ;
        scanf ("%d", &tab [i]) ;
    }

    /* On affiche a present le tableau complet */
    printf ("Voici le tableau que vous avez rempli :\n") ;
    for (i = 0 ; i < 10 ; i = i + 1) {
        printf ("%d ", tab [i]) ;
    }
    printf ("\n") ;

    return 0 ;
}
```

Tableau à 02 dimensions: Définitions

- **Ensemble de lignes et de colonnes**
- Un tableau à deux dimensions contient donc ***L*C composantes***.
  - L le ***nombre de lignes***
  - C le ***nombre de colonnes***.
  - L et C sont alors les deux ***dimensions***.

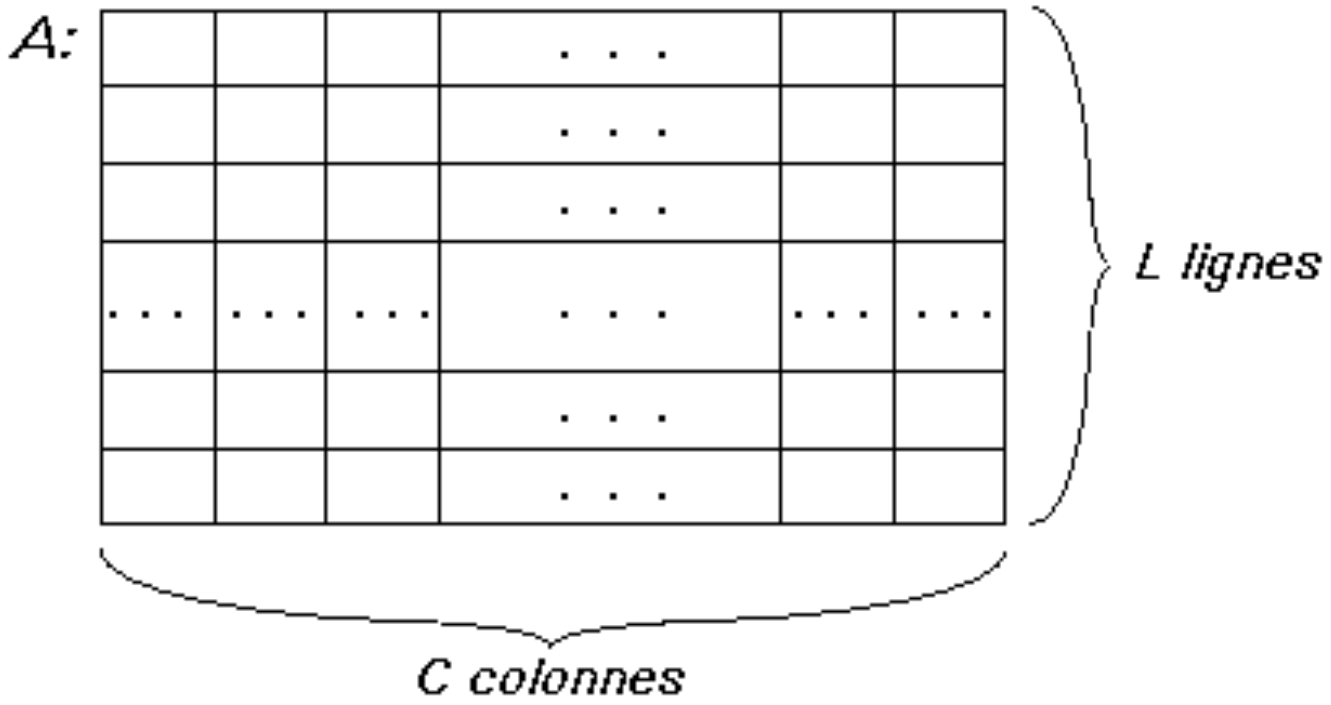


Tableau à 02 dimensions: Définitions

- Exemple : La Liste (prénom, nom, Téléphone et adresse) de vos camarades de classe)

Pape Ousmane	DIOP	77 876 76 76	Dakar
Aliou	FAYE	76 555 44 22	Ziguinchor
Fatou	FALL	78 763 44 22	Touba
Jean Alain	DIATTA	33 875 55 11	THIES
Amie	Ka	77 765 77 11	Matam
Paul	SARR	76 776 32 44	Mbour

- On dit qu'un tableau à deux dimensions est ***carré***, si L est égal à C.

Tableau à 02 dimensions: Déclaration

- **Déclaration nécessite trois informations :**
  - Le type des éléments du tableau ;
  - le nom du tableau (son identificateur) ;
  - La taille des différentes dimensions (Nombre de lignes et Nombre de colonnes).

Type NomduTableau [Nb lignes] [Nb colonnes];

  - **Exemples**
    - int tab[20][35];
      - On déclare un tableau de 20 lignes et 35 colonnes
    - Char TabEtudiants [3][15];
      - On déclare un tableau de 3 lignes et 15 colonnes

Tableau à 02 dimensions: Initialisation

- **Initialiser un tableau 2D**
  - int Tab[3][2] = {{ 51, 23 }, { 11, 32 }, { 83, 14 } };
  - Tab[0][0] =51;
  - Tab[1][1] =32;
  - Tab[0][0] = Tab[2][1] ;

51	23
11	32
83	**51**

  - Char tabE[2][2] = {{ 'Fatou', 'BA' }, { 'Amie', 'FAYE' }, { 'Jean", 'SARR' } };

Fatou	BA
Amie	FAYE
Jean	SARR

Tableau à 02 dimensions: Initialisation

- **Initialiser un tableau 2D**

```
int NOTE[10][20] = {{45, 34, ... , 50, 48},
                    {39, 24, ... , 49, 45},
                      ...    ...     ...
                    {40, 40, ... , 54, 44}};
```

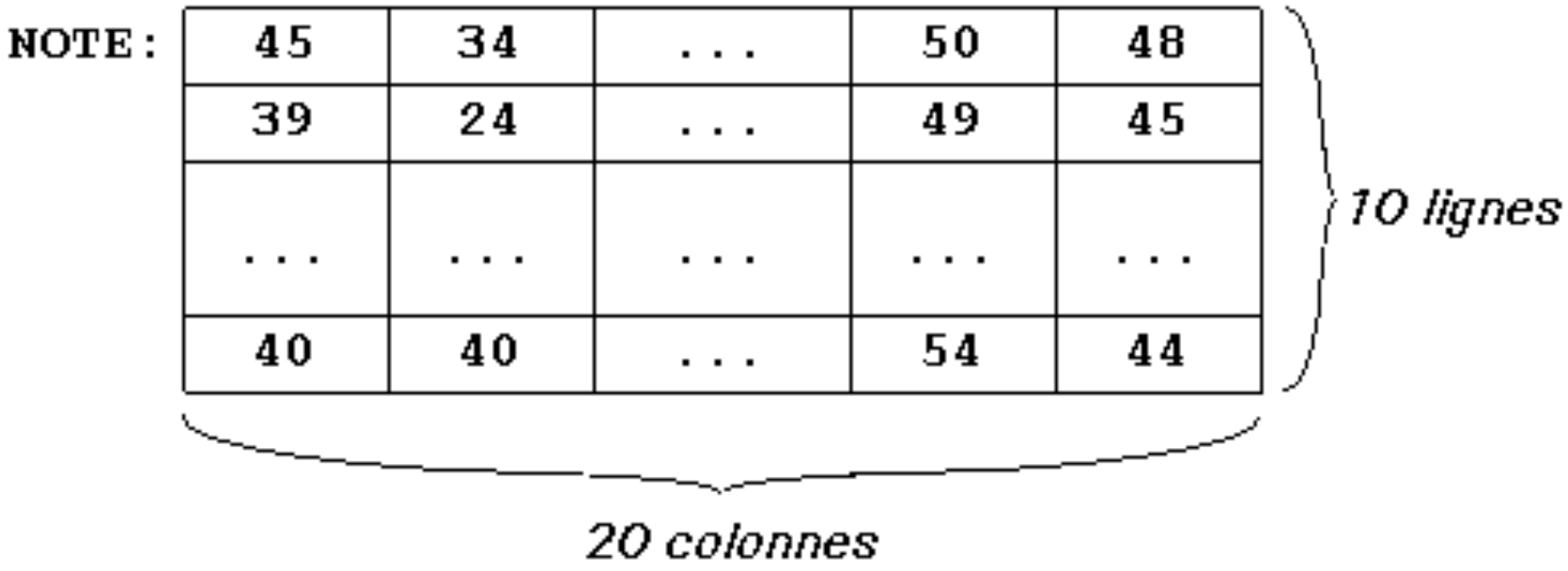


Tableau à 02 dimensions: Accès au éléments

- **Utiliser les deux indices i et j**
  - i pour les lignes 0 à L
  - j pour les colonnes 0 à C

Les éléments d'un tableau de dimensions L et C se présentent de la façon suivante:

```
/                                                                  \
|  A[0][0]    A[0][1]    A[0][2]    . . .    A[0][C-1]   |
|  A[1][0]    A[1][1]    A[1][2]    . . .    A[1][C-1]   |
|  A[2][0]    A[2][1]    A[2][2]    . . .    A[2][C-1]   |
|    . . .      . . .      . . .    . . .      . . .     |
| A[L-1][0]  A[L-1][1]  A[L-1][2]  . . .  A[L-1][C-1]  |
\                                                                  /
```

Tableau à 02 dimensions: Parcourir

- On utilise deux boucles FOR imbriquées
  - La première boucle i pour les lignes for (i=0;i<L; i++)
  - La seconde boucle j pour les colonnes for (i=0;i<C; i++)
- Principe: On se position à i=0 puis on parcourt tous les j (0 à C) puis on passe à i=1 ainsi de suite jusqu'à i<L

```
int main(void) {
    int Tab[3][2] = {{ 51, 23 }, { 11, 32 }, { 83, 14 } };
    int i=0, j=0;
    for (i = 0; i < 3; ++i) {
        for (j = 0; j < 2; ++j) {
            printf("tab[%d][%d] = %d\n", i, j, tab[i][j]);
        }
    }
    return 0;
}
```

51	23
11	32
83	**14**

Introduction

- **Trier = ordonner suivant un ou plusieurs critères**

- Existe 6 types de tri:
  1. tri par sélection
  2. tri a bulle
  3. tri par permutation
  4. tri par insertion
  5. tri par fusion
  6. tri rapide

  NB: Nous intéresserons juste aux trois premiers algorithmes

Tri par sélection

- L'idée est simple :

1. Rechercher le plus grand élément (ou le plus petit)
2. Le placer en fin de tableau (ou en début)
3. Recommencer avec le second plus grand (ou le second plus petit)
4. Le placer en avant-dernière position (ou en seconde position)
5. Et ainsi de suite jusqu'à avoir parcouru la totalité du tableau.

- Ce tri ce fait par 2 boucles for

```
int i,j,c;
for(i=0;i<N-1;i++)
    for(j=i+1;j<N;j++)
        if ( T[i] > T[j] ) {
            c = T[i];
            T[i] = T[j];
            T[j] = c;
        }
```

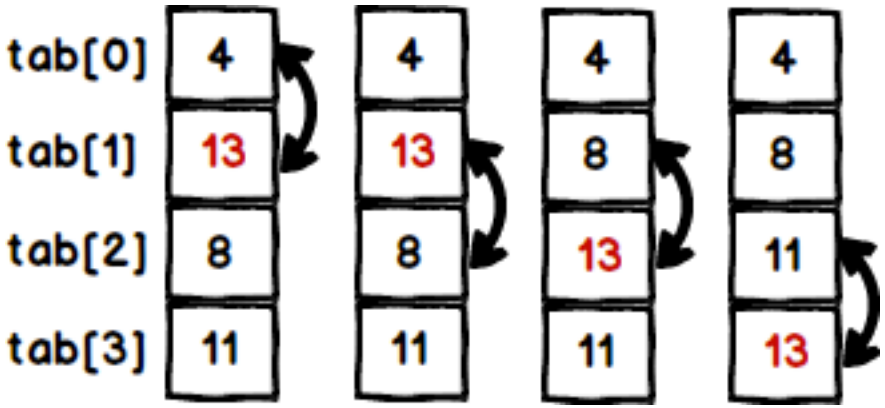
Tri par sélection

6	2	8	1	5	3	7	9	4	0

6	2	4	1	5	3	0	**7**	**8**	**9**
0	2	4	1	5	3	**6**	**7**	**8**	**9**
0	2	4	1	3	**5**	**6**	**7**	**8**	**9**
0	2	3	1	**4**	**5**	**6**	**7**	**8**	**9**
0	2	1	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Tri à bulle ou «*Bubble Sort*»

- Cet algorithme parcourt le tableau en comparant 2 cases successives, lorsqu'il trouve qu'elles ne sont pas dans l'ordre souhaité (croissant dans ce cas) , il permute ces 2 cases
  - les valeurs les plus petites se déplacent progressivement vers le haut, comme une bulle d'air dans l'eau, et les valeurs les plus grandes descendent vers le bas du tableau.
- A la fin d'un parcours complet on aura le déplacement du minimum a la fin du tableau.
- En faisant cet opération N fois , le tableau serait donc trié.



Tri à bulle

- Dans chaque boucle, les paires successives des éléments sont comparées et permutées si nécessaire.
- Si la paire a la même valeur ou est en ordre croissant, on la garde elle-même.
- S'il y a **N** éléments à trier, le tri à bulles fait **N-1** pour traverser le tableau.

```
int i,j,c;

for(j=1;j<=N;j++) // pour faire l'operation N fois
    for(i=0;i<N-1;i++)
        if ( T[i] > T[i+1] ) {
                c = T[i];
                T[i] = T[i+1];
                T[i+1] = c;
        }
```

Tri par permutation

- Cet algorithme consiste a parcourir le tableau jusqu'à ce qu'il trouve un élément inférieur que le précédent (mal placé),il prend cet élément et il le range à sa place dans le tableau , et il continue le parcours jusqu'à la fin. Et affin de ne pas écraser les valeurs du tableau il faut réaliser une translation des valeurs a l'aide d'une boucle .

```
int i,j,k,c;

for(i=1;i<N;i++) {

    if ( T[i] < T[i-1] ) { // si l'élément est mal placé

        j = 0;

        while ( T[j] < T[i] ) j++; // cette boucle sort par j = la nouvelle position de l'élément

 c = T[i]; // ces 2 lignes servent a translater les cases en avant pour pouvoir insérer l'élément
        for( k = i-1 ; k >= j ; k-- ) T[k+1] = T[k];
 T[j] = c; // l'insertion
    }
}
```

TD 2 & TP 2:
Les tableaux et algorithmes de tri

Voir fichier joint au cours

Chapitre 3
Programmation modulaire et récursivité

La modularité des programmes

- La programmation modulaire est le fait de structurer le code source d'un programme informatique en plusieurs fichiers.
- Un programme dépassant une ou deux pages est difficile à com prendre
- Une écriture modulaire permet de scinder le programme en plusieurs parties et sous - parties
- En C, le module se nomme la « fonction ».
- Le programme programme principal décrit essentiellement essentiellement les enchaînements des fonctions
- Objectifs
  - Lisibilité du code
  - Réutilisation de la fonction
  - Tests facilités
  - Évolutivité du code
  - Travail en equipe

Les méthodes : fonctions et procédures

- Une méthode est une suite d'instructions séparée permettant d'exécuter un traitement bien défini dans mon programme.
- Exemple: Pour un programme Calcul.C nous pouvons avoir en plus de la méthode principale main() les méthodes suivantes:
    - addition()
    - soustraction()
    - multiplication()
    - division ()
- Les procédures et les fonctions ont pour objectif d'éviter les répétitions de suite d'instructions
- Nous avons deux forme de méthodes en programmation :
  - **Les fonctions:** retournent toujours un résultat après exécution
  - **Les procédures:** ne retournent pas un résultat après exécution

  - **NB: Officiellement en langage C nous n'avons que des procédures c.-à-d. que même les procédures sont nommées fonction**

Les méthodes : fonctions et procédures

- **Les paramètres / Arguments**
  - Un paramètres est une information sous la forme d'une variable qui est passée à la fonction lors de l'appel
  - Elle peut subir des modification dans celle-ci
  - Une méthode peut disposer de plusieurs paramètres

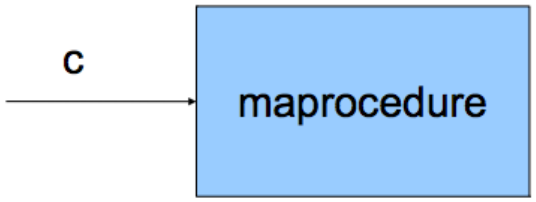
    Type1 Parametre1, Type2 Parametre2 …
  - Exemple

    int Addition (int x, int y) { …. }
  - Pas de déclaration multiple pour les paramètres
  - Dans le cas où une fonction n'a pas de paramètres formels, le mot clé **void** est mis en tant que *liste-de-déclarations-de-paramètres*.

```
double pi(void) {
    return(3.14159);
}
```

Les méthodes : fonctions et procédures

- **Une procédure**
  - **Une méthode qui** ne retourne pas un résultat après exécution dont pas d'instruction **return**

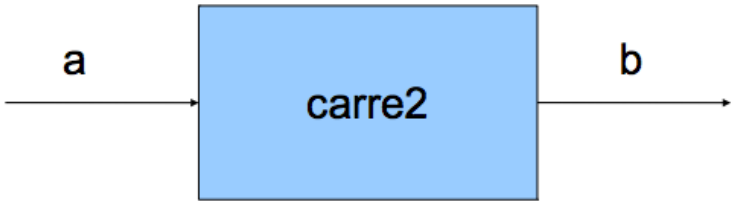



```
void nomdeprocédure(paramètres) {
   instruction_exécuté1;
   instruction_exécuté2;
   instruction_exécuté3;
   ...;
}
```

  - Déclarée avec le mot-clé **VOID**
  - Peut prendre des paramètres
  - Peut déclarée ses propres variables et constantes (locales)

Les méthodes : fonctions et les procédures

- **Une fonction**
  - **Une méthode qui** retourne toujours un résultat après exécution dont dispose toujours de l'instruction **return**



  - Déclarée en utilisant le type de la valeur de retour
    - Exemple: Si une fonction doit retourner un entier alors elle sera déclarer avec le mot int.
  - Peut prendre des paramètres
  - Peut déclarée ses propres variables et constantes (locales)

Type-retourné **NOM-FONCTION (type1 paramètre1, type2 paramètre2, ...) ;**

```
double CalculePrixNet(double prix, double tauxTVA) ;
```

Les méthodes : fonctions et les procédures

- **Une fonction**
  - Déclaration

```
typederetour nomdefonction(paramètres) {
  instruction_exécutél;
  ...;
  return valeurderetour
}
```

  - Exemple

```
int mafonction(int x) {
  int a; ...
  return(a);
}
```

- NB
  - En C, une fonction ne peut retourner qu'une valeur au plus grâce à la commande return
  - Le type de la fonction doit être le même que celui de la valeur retournée

Variables locales / Variables Globales

- **Variables locales**
  - Un bloc est la partie de code compris entre {}
  - Une variable créee dans un bloc n'existe que dans ce bloc
  - C'est une variable variable locale au bloc
  - Elle ne sera pas connue en dehors
  - Sa valeur est perdue à la sortie du bloc
  - « Sa durée de vie est celle du bloc »

```
int triple (int x)
{
int y ;
y = 3 * x ;
return (y) ;
}
```

- **Une variable globale**
  - Existe en dehors de tout bloc
  - A sa mémoire réservée pour toute l'exécution du programme
  - « Sa durée de vie est celle du programme »

```
int i ;
main()
{ i=2;
printf(''%d'',i);
}
```

Variables locales / Variables Globales

Si une variable est utilisée uniquement de manière locale, sa déclaration se fait dans la fonction où elle est utilisée.

Exemple:

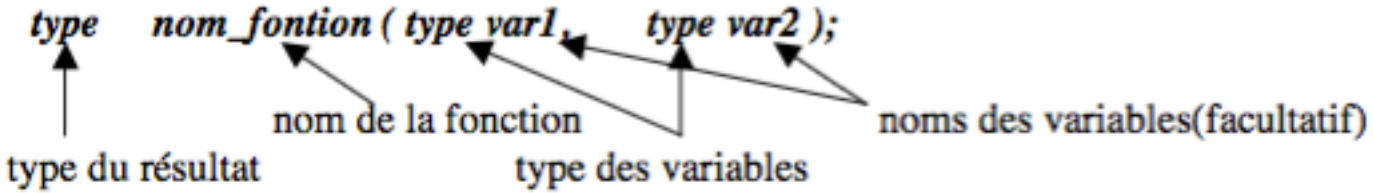
```
main()                          void fonction()
        {                               {
        int A;                          int A;
        }                               }
```

Si une variable doit être disponible pour plusieurs fonctions du programme, elle doit être déclarée de manière globale, c'est à dire juste après les include (et donc aussi avant le main).

```
#include <stdio.h>
int A;
void main()
        {
        …
        }
```

Les fonction dans le programme: Prototype

- En général, le nom d'une fonction apparait trois fois dans un programme:
  - Lors de sa déclaration
    - En C, il faut déclarer une fonction avant de pouvoir l'utiliser.
    - Seule la fonction main n'a pas besoin d'être déclarée.



  - Lors de sa définition
    - Si on définit la fonction (dans le programme) avant de l'appeler, la déclaration est inutile, mais: afin de faciliter la lecture d'un programme, on conseille de définir les fonctions après le main(), il faut donc les déclarer avant (le main), juste après les include.
  - Lors de son appel dans une autre methode

Les fonction dans le programme: Prototype

- Une fonction doit toujours être déclarée sous forme de prototype avant d'être implémenter
- Se fait après les #include

```
#include <stdio.h>
#include <stdlib.h>
int addition(int a, int b);
int soustraction(int a, int b);
int multiplication(int a, int b);
double division(int a, int b);
```

```
#include <stdio.h>

int triple(int x) ; //prototype

int main()
{
int a=2 ;
int b ;
triple(2) ; //appels
triple(a) ;
b = triple(a) ;
a = triple(a) ;
return 0;
}

int triple(int x) //définition
{
return (3*x) ;
}
```

Structure générale d'un programme avec méthodes

#include ...
#define ...

Déclarations des fonctions (prototypes)

main() {
...
appels aux fonctions
...}

Définitions des fonctions

```
#include <stdio.h>
#include <stdlib.h>
int addition(int a, int b);
int soustraction(int a, int b);
int multiplication(int a, int b);
double division(int a, int b);
int main() {
    int x=12, y=4;
    printf("Soit les nombre %d et %d\n", x, y);
    printf("leur somme est: %d ", addition(x, y));
    printf("leur difference est: %d ", soustraction(x, y));
    printf("leur produit est: %d ", multiplication(x, y));
    printf("leur quotien est: %d ", division(x, y));
    return 0;
}
int addition(int a, int b) { return a + b;}
int soustraction(int a, int b) { return a - b; }
int multiplication(int a, int b){    return a*b; }
double division(int a, int b){  return a/b; }
```

Appel de fonction externe (Dans un autre fichier)

- Dans le cas de la programmation modulaire faisant appel à une fonction se trouvant dans un autre fichier, nous remarquons l'utilisation de 3 fichiers:

.c

Contient les définitions de nos fonctions on remarquera la directive d'inclusion #include qui permet de prendre en compte les entêtes définies dans le .h

.h

Contient les entêtes de nos fonctions (en langage c il est possible de séparer les définitions des entêtes de fonctions des définitions des fonctions elle-même)

main.c

Contient le programme principal, on remarquera que le fichier main.c peut utiliser les fonctions d'opération grâce à la directive de compilation #include.

Le fichier header (.h)

- Si un programme main.c désire utiliser une (ou plusieurs) des fonctions disponibles dans un autre fichier par exemple calcul.c, il doit accéder à leurs déclarations pour pouvoir se compiler.
- C'est le rôle des fichiers en-tête (header) avec l'extension .h.
- On aura donc un couple de fichiers :
  - un .h (pour les déclarations)
  - un .c (pour les définitions) portant le même nom.

- Exemple de fichier calcul.h

```
int addition(int a, int b);
int soustraction(int a, int b);
int multiplication(int a, int b);
double division(int a, int b);
```

Le fichiers module (.c) et main.c

- Le module contient l'ensemble des traitements avec leurs prototypes+ implémentations
  - Les procédures
  - Les fonctions :

- Exemple de fichier calcul.c

- Dans le fichier main.c
  - Utiliser un INCLUDE du fichier calcul.h
  - Faire appel aux fonctions de clacul.c

```c
#include <stdio.h> /* pour la déclaration de printf */
#include "calcul.h" /* pour la déclaration de multiplier */
int main() {
   printf("2*2 = %d\n", multiplier(2, 2));
   return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
int addition(int a, int b);
int soustraction(int a, int b);
int multiplication(int a, int b);
double division(int a, int b);
int addition(int a, int b) {
                    return a + b;
}
int soustraction(int a, int b) {
                    return a - b;
}
int multiplication(int a, int b){
                    return a*b;
}
double division(int a, int b){
                    return a/b;
}
```

La récursivité

- **La récursivité**
  - Une notion est dite récursive lorsqu'elle se contient elle-même en partie ou si elle est partiellement définie à partir d'elle-même.
  - Typiquement, il s'agit d'une suite dont le terme général s'exprime à partir de termes qui le précèdent.

- La forme récursive est généralement plus simple à comprendre et plus élégante, elle peut être séduisante dans sa conception intellectuelle.

- **Une fonction récursive**
  - Une fonction qui s'appelle elle-même.

```
void fct() {
    ... fct();
}
```

Récursive / Non Récursive

- la factorielle d'un nombre N donné est le produit des nombres entiers inférieurs ou égaux à ce nombre N.
  - Exemple: 5! =1*2*3*4*5 ou bien 5! =5*4!

Forme Normale

N! = 1*2*3 ... *(N-1)*N

```
int factorielle (int N) {
      int i, fact=1;
      for (i=2;i<=N;i++) {
            fact*=i;
      }
 return fact;
}
```

Forme Recursive

N! = N*(N-1)!

```
int factorielle (int N) {
     if (N<=1)
        return 1;
     else
       return N*Factorielle(N-1);
}
```

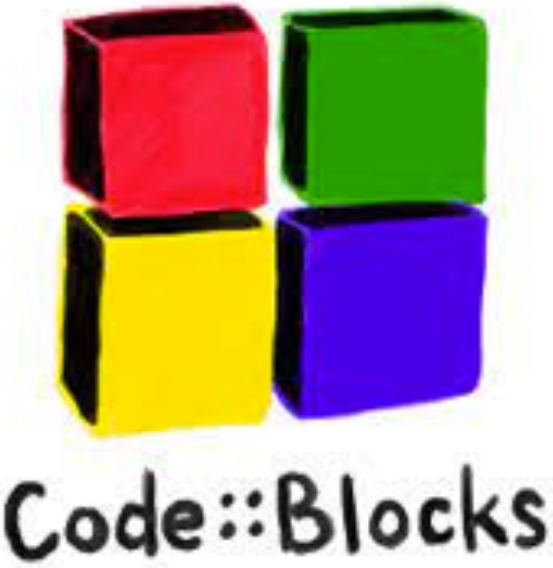
Récursive / Non Récursive

- Les **entrées/sorties** (**E/S**) ne font pas partie du langage C, car ces opérations sont dépendantes du système. Néanmoins puisqu'il s'agit de tâches habituelles, sa bibliothèque standard est fournie avec des fonctions permettant de réaliser ces opérations de manière portable. Ces fonctions sont principalement déclarées dans le fichier **stdio.h**.
- Les entrées/sorties en langage C se font par l'intermédiaire d'entités logiques, appelées **flux**
- Un flux de texte est organisé en **lignes**. En langage C, une ligne est une suite de caractères terminée par le **caractère de fin de ligne** (inclus) : '\n'.

Références

- Bac, C. (2004). Support de Cours de Langage C. *Institut National de Télécommunications*.
- Klaudel, H. Langage C: notes du cours.
- http://cours.pise.info/algo/introduction.htm
- https://www.i3s.unice.fr/~map/Cours/DUT_API_TD_TP/C1-2_APIStructuresAlgorithmiquesdeBase.pdf
- https://www.lri.fr/~hivert/COURS/CFA-L3/00-Intro.pdf
- https://openclassrooms.com/fr/courses/4366701-decouvrez-le-fonctionnement-des-algorithmes/4384756-tirez-pleinement-parti-de-ce-cours
- https://www.gladir.com/CODER/C/deffonction.htm
- https://melem.developpez.com/tutoriels/langage-c/fichiers/?page=cours

Logiciel Recommandé pour les TPs



Langage C

Cours Magistral

Licence Informatique & MIO

Pr Edouard Ngor SARR
Enseignant-Chercheur
Université Assane SECK de Ziguinchor (UASZ)
Ziguinchor-Sénégal
edouard-ngor.sarr@univ-zig.sn

Oct 2025