# COURS C++

## 1ère partie

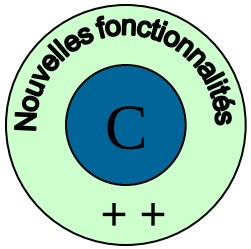
## Programmation structurée
## C++ de base

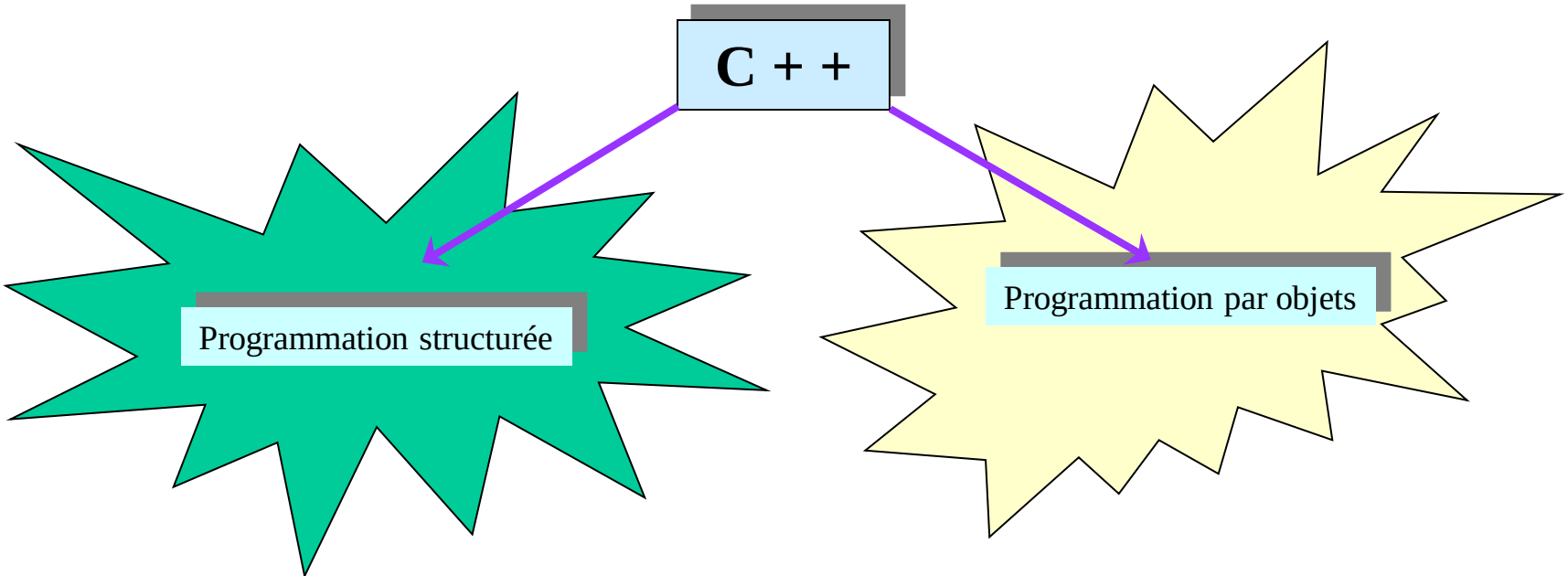
# Plan

- **Introduction**
- **Structure d'un programme C++**
- **Variables et Constantes**
- **Opérateurs et Instructions**
- **Tableaux et Enregistrements**
- **Sous-Programmes**
- **Exceptions**
- **Fonction surchargée et Fonction générique**

# Introduction

- **Origine**
  - **1970 : langage C (Dennis Ritchie)**
  - **1980 : langage C++ (Bjarne Stroustrup)**
- **Pourquoi C++ ?**
  - **Pour corriger les imperfections du C**
  - **Pour faire de la programmation par objets**



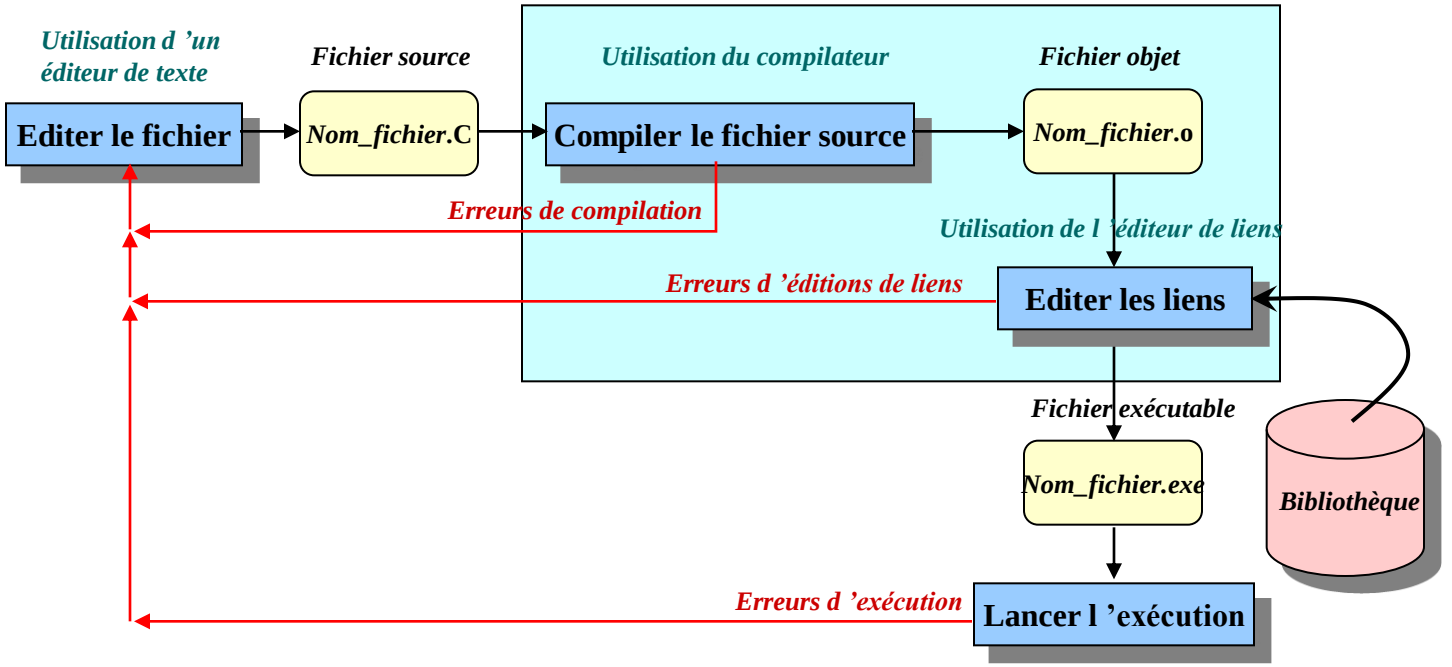



# Introduction

- **Pourquoi avoir choisi C++ ?**
  - **Moins d'inconvénients que C**
  - **Très utilisé dans l'industrie**
  - **Disponible sur tout type d'ordinateur**
  - **Support à la programmation par objets**

# Introduction

- **Cycle de mise au point d'un programme**



# Structure d'un programme simple

- **Algorithme**

```
importer entréeSortie;


-- détermine le plus grand des 2 entiers
-- a et b
programme calculerLePlusGrand

glossaire
   a <Entier> ; --1er ent
   b <Entier> ; -- 2ème e

début
   -- lire les 2 entiers
   écrire ("Entrez la pr
   lire (a) ;
   écrire ("Entrez la deuxième valeur : ") ;
   lire (b) ;
   -- calculer et écrire le plus grand des
   -- 2 entiers
   si a > b alors
      écrire (a) ;
   sinon
      écrire (b) ;
   fin si ;
fin
```

- **Programme C++ :** ***max2entiers.C***

```
#include "/usr/local/public/BIBLIOC++/entreeSortie.h"
#include "/usr/local/public/BIBLIOC++/chaine.h"

// determine le plus grand des deux entiers
// a et b
int main ( )
{

                    emier entier a comparer
                    uxieme entier a comparer


                     entiers a et b
                    aine ("Entrez la premiere valeur : ")) ;

   ecrire (uneChaine ("Entrez la deuxieme valeur : ")) ;
   lire (b) ;
   // calculer et ecrire le plus grand des
   // 2 entiers
   if  (a  >  b)
      ecrire (a) ;
   else
      ecrire (b) ;
}
```

**Entrez la premiere valeur : 20**
**Entrez la deuxième valeur : 30**
**30**

# Structure d'un programme plus complexe

```
importer entréeSortie;

— retourne le plus grand entier des 2 entiers x et y
fonction max (entrée x <Entier>, entrée y <Entier>)
retourne <Entier>
début
    si x > y alors
        retourner (x) ;
    sinon
        retourner (y) ;
   fin si ;
fin

— détermine le plus gran
— entiers a, b et c
programme calculerLePlus
glossaire :
   a <Entier> ; — 1er en
   b <Entier> ; — 2ème entier à comparer
   c <Entier> ; — 3ème entier à comparer
début
   — lire les 3 entiers a, b et c
   écrire ("Entrez 3 entiers : ") ;
   lire (a) ; lire (b) ;  lire (c) ;
   — calculer et écrire le plus grand
   — des 3 entiers
   écrire  (max (a,  max (b, c))) ;
fin
```

```
Entrez 3 entiers : 20
40
10
40
```

```
#include "/usr/local/public/BIBLIOC++/entreeSortie.h "
#include "/usr/local/public/BIBLIOC++/chaine.h "

// retourne le plus grand entier des 2 entiers x et y
int max (const int x, const int y)
{
    if (x > y)
      return  (x) ;
    else
      return  (y) ;


ine le plus grand entier des 3
s a, b et c
()

   b, c ;        // entiers a comparer



   // lire les 3 entiers a, b et c
   ecrire (uneChaine ("Entrez 3 entiers : ")) ;
   lire (a) ; lire (b) ; lire (c) ;
   // calculer et ecrire le plus grand
   // des 3 entiers
   ecrire (max (a, max (b, c))) ;
}
```

# Appel de sous-programmes

*prototype*

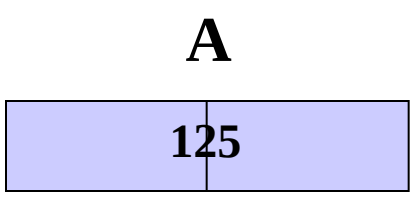
*définition*

```
#include "/usr/local/public/BLIBLIOC++/entreeSortie.h"
#include "/usr/local/public/BLIBLIOC++/chaine.h"

// prototype de la fonction max
int max(const int x , const int y);

// corps de la fonction main
int main()
{
   int a, b, c;          // entiers a comparer
   // lire les 3 entiers a, b et c
   ecrire(uneChaine("Entrez 3 entiers : "));
   lire(a) ; lire(b) ; lire(c);
   // calculer et ecrire le plus grand
   ecrire (max(a,  max( b, c)));
}

// corps de la fonction max
int max (const int x, const int y)
{
   if   (x > y)
      return (x);
   else
      return (y);
}
```

# Variable

- **Définition d'une variable**
  - **zone mémoire**
  - **caractérisée par :**
    - **un identificateur *(nom symbolique)***
    - **un type**
    - **une valeur**
- **Définition d'un type**
  - **caractérisé par un nom *(char, int, float …)***
  - **associe à tout variable :**
    - **sa taille en mémoire (nombre d'octets occupés)**
    - **son domaine de valeurs possibles**
    - **les opérations possibles**
- *Exemple*

**short A = 125;**




*Codage sur 2 octets*

# Identificateurs de variables

- **doit commencer par une lettre**
- **peut contenir des chiffres ou le symbole souligné « _ »**
- **ne doit pas correspondre à un mot réservé du langage** 
- *Exemples* **:**

  ***var***

  ***VAR***

  ***lettre1***

  ***lettre2***

  ***TRAIT1_FINI***

  ***finBoucle***

# Déclaration de variables

- *Exemples de déclarations simples*

```
int var;
long VAR;
char lettre1, lettre2, lettre3;
bool TRAIT1_FINI, finBoucle;
```

- *Exemples de déclarations avec initialisations*

```
int var = 1 ;
char lettre1 =  'A',  lettre2 = 'B', lettre3 ;
bool TRAIT1_FINI, finBoucle = true ;
```

# Constantes

- **Constantes littérales**

```
34.456          constante réelle
12.346 e2       constante réelle notation scientifique
123             constante entière
'A'             constante caractère
"cours d'info"  constante chaîne de caractères
```

- **Constantes symboliques**

```
const float PI = 3.14 ;        // constante décimale
const char ESPACE = ' ' ;      // constante caractère
float x = 5 * PI ;
```

# Opérateurs

- **Symboles permettant d'effectuer des opérations**
- **Pluralité :**
  - **nombre d'opérandes associés à l'opération**
- **Priorité :** 
  - **ordre dans lequel les opérateurs sont évalués dans une expression**
- *Exemple* **:**

```
A / B - C * D   <⟷>   (A / B) - (C * D)
```

# Opérateur d'affectation =

- *Exemples* :

```
int somme, a, b, numCompte = 1000;
bool test;
somme = 0 ;
somme = numCompte;
test = (a == b);
somme = somme + 1;
```

# Opérateurs arithmétiques

- **5 opérateurs : + - * / %**
- *Exemples* **:**

```
int somme = 0, resultat;
const int INCREMENT = 5;
somme = somme + 10;                          // somme = 10
somme = somme + 8 * 3  + INCREMENT;          // somme = 39
resultat = somme % 5;                        // resultat = 4
resultat = (resultat + 32 ) / 5;             // resultat = 7

float op, pi = 3.14;
op = 567.89 + 78.8 - pi;                     // op = 643,55

char c = 'A';                                // c = 65 (code ASCII)
c = c + 1;                                   // C = 66 = 'B'
c = 'D' + 2;                                 // C = 70 = 'F'
```

# Conversions

- **Conversions implicites**

```
int e ;
float r ;
char c ;
…
e = r * 5.7 ;                                   // attention
r = e ;                                         // oui
c = e ;                                         // attention
```

- **Conversions explicites**

```
int n = 100 ;
double racine ;
racine = sqrt (double (n * 10)) ;       // sqrt attend un type double

int nbVal = 4, somme = 50 ;
float moyenne ;
moyenne = somme / float (nbVal) ;     // nbVal est converti en flottant
```

# Autres opérateurs arithmétiques

- **Opérateur d'incrémentation : + +**
- **Opérateur de décrémentation : - -**
- *Exemple* **:**

```
int i  = 10 ;
i++ ;                                   // ⇔ i = i + 1
i-- ;                                   // ⇔ i = i  - 1
```

# Opérateurs relationnels

- `==  !=  <  <=  >  >=`
- **retournent une valeur booléenne**
  - **vrai (1)**
  - **faux (0)**
- *Exemple* **:**

```
int i  = 10 ;
ecrire  (i = = 10) ;                    // 1 (vrai)
ecrire  (i  !=  10) ;                   // 0 (faux)
ecrire  (i  <   10) ;                   // 0 (faux)
```

# Opérateurs logiques

- **&&  ||  !**
- **retournent une valeur booléenne**
  - **vrai (1)**
  - **faux (0)**
- *Exemple :*

```
int i  = 10, j = 5 ;
bool trouve = true ;
ecrire ((i = = 10) && (j = = 5)) ;    // 1 (vrai)
ecrire ((i = = 5) || (j= =5)) ;       // 1 (vrai)
ecrire (! (i  <  5)) ;                // 1 (vrai)
ecrire (! trouve) ;                   // 0 (faux)
```

# Instructions

- **Une instruction indique une action à réaliser**
- **Elle se termine par ;**
- **Le format est libre**
- **Plusieurs instructions peuvent être placées sur la même ligne**
- **Une même instruction peut faire l'objet de plusieurs de plusieurs lignes physiques**
- **C++ introduit différentes structures de contrôle**
- *Exemple :*

```
moyenne  = somme / nbVal ;
```

# La séquence

- **Enchaînement inconditionnel d'actions**
  - **introduit par la notion de bloc { }**
- *Exemple :*

```
{
    int a, b ;
    a = 6 ;
    b = 10 ;
    b = a * b + 100 ;
    ecrire (b) ;
    ecrire (a) ;
}
```

# La sélection

- **Choix entre 2 actions**
- **Syntaxe :**

```
if (condition)
      action1 ;
else
      action2 ;
```

- *Exemples* **:**

```
si a > b
alors
   a ← b ;
sinon
   b ← a ;
fin si ;
```

```
if (a > b)
   a = b ;
else
   b = a ;
```

```
si a > b
alors
   a ← b ;
fin si ;
```

```
if (a > b)
   a = b ;
```

# La sélection

- *Exemple* **:**

```
si a > b
alors

   a ← b ;
   b ← c ;
   f ← e ;

sinon
   b ← a ;
fin si ;
```

```
if (a > b)
{
   a = b ;
   b = c ;
   f = e ;
}
else
   b = a ;
```

# La sélection

- *Exemples de si imbriqués* :

```
si n = 0 alors
   si a > b alors
      z ← a ;
   sinon
      z ← b ;
   fin si ;
fin si ;
```

```
if (n = = 0)
   if (a > b)
      z = a ;
   else
      z = b ;
```

```
si n = 0 alors

   si a > b alors
      z ← a ;
   fin si ;
sinon
   z ← b ;
fin si ;
```

```
if (n = = 0)
{
   if (a > b)
      z = a ;
}
else
   z = b ;
```

# La sélection à choix multiples

- **Syntaxe :**

```
switch (expression)
{
     case constante1 : action1 ;
     case constante2 : action2 ;
        …
     default : actionN ;
}
```

- *Exemples* **:**

```
si choix = 1 alors
   b ← a ;
 sinon
   si choix = 2 alors
      a ← b ;
   sinon
      c ← d ;
   fin si ;
fin si ;
```

```
int choix ;
…
switch (choix)
{
   case 1 :   b = a ; break ;
   case 2 :   a = b ; break ;
   default :  c = d ;
}
```

# La sélection à choix multiples

```
si c = 'A' alors
   x ← 0 ;
   y ← 0 ;
sinon
   si c = 'B' ou c = 'C' alors
      x <- 1 ;  y <- 1 ;
   sinon
      si c = 'D' ou c = 'E' alors
         x ← 2 ;  y ← 2 ;
      sinon
         si c = 'F' alors
            x ← 3 ;  y ← 3 ;
         fin si ;
      fin si ;
   fin si ;
fin si ;
```

```
char c ;
…
switch (c)
{
   case 'A' : x = 0 ; y = 0 ; break ;
   case 'B' :
   case 'C' : x = 1 ; y = 1 ; break ;
   case 'D' :
   case 'E' : x = 2; y = 2 ; break ;
   case 'F' : x = 3 ; y = 3 ;
}
```

# La sélection à choix multiples

```
si n = 4 alors
   écrire ("multiple de 4\n") ;
   écrire (" multiple de 2\n") ;
sinon
   si n = 2 alors
      écrire ("multiple de 2\n") ;
   sinon
      si n = 9  alors
         écrire ("multiple de 9\n") ;
         écrire ("multiple de 3\n") ;
      sinon
         si n = 3 alors
            écrire ("multiple de 3\n") ;
         fin si ;
      fin si ;
   fin si ;
fin si ;
```

```
int n ;
…
switch (n)
{
   case 4 : ecrire (uneChaine ("multiple de 4\n")) ;
   case 2 : ecrire (uneChaine ("multiple de 2\n")) ;
            break;
   case 9 : ecrire (uneChaine ("multiple de 9\n")) ;
   case 3 : ecrire (uneChaine ("multiple de 3\n")) ;
}
```

# La répétition

- **Action répétée tant qu'une condition reste vérifiée**
- **3 structures différentes pour traduire une répétition**
  - ***tantque faire …***
    - ***while***
  - ***répéter … tantque condition***
    - ***do … while***
  - ***pour***
    - ***for***

# Boucle *tantque ... faire*

- **Syntaxe :**

```
while (condition)
    action ;
```

- *Exemple* **:**

```
s ← 0 ;
i ← 1 ;
tantque i <= 10 faire

    s ← s + i ;
    i ← i + 1 ;
fin tantque ;
```

```
s = 0 ;
i = 1 ;
while (i <= 10)
{
    s = s + i ;
    i ++ ;
}
```

# Boucle *tantque ... faire*

- *Exemple* **:**

```
s ← 0 ;
lire (val) ;
tantque val /= 0 faire

   s ← s + val ;
   lire (val) ;
fin tantque ;
```

```
s = 0 ;
lire (val) ;
while (val != 0)
{
   s = s + val ;
   lire (val) ;
}
```

# Boucle *répéter ... tantque*

- **Syntaxe :**

```
do
        action ;
while (condition) ;
```

- *Exemple* **:**

```
écrire ("Entrez valeur : ") ;
lire (val) ;
tantque val <= 0 faire

   écrire ("Entrez valeur : ") ;
   lire (val) ;

fin tantque ;
```

```
do
{
   ecrire (uneChaine ("Entrez valeur : ") ;
   lire (val) ;
}
while (val <= 0) ;
```

# Boucle *pour*

- **Syntaxe :**

```
for (initialisation ; condition ; variation)
     action ;
```

- *Exemple* **:**

```
s ← 0 ;
i ← 1 ;
n ← 0 ;
tantque i <= 10 faire
   n ← s ;
   s ← s + i ;
   i ← i + 1 ;
fin tantque ;
```

```
s = 0 ;
n = 0 ;

for (i = 1 ; i <= 10 ; i ++)
{
   n = s ;
   s = s + i ;
}
```

# Boucle *pour*

- *Exemple* :

```
i ← 1 ;
j ← 10 ;
tantque i <= 10 et j > 5 faire
   écrire (i) ;
   i ← i + 3 ;
   j ← j - 1 ;
fin tantque ;
```

```
for (i=1 , j=10 ; i<=10 && j>5 ; i=i+3 , j--)
   ecrire (i) ;
```

# Tableau

- **Un tableau permet de stocker en mémoire des valeurs de même type**

- *Exemple* **:**

notes

| 12 |
|---|
| 5 |
| 12 |
| 13 |
| 4 |
| 15 |
| 16 |
| 11 |
| 10 |
| 20 |

# Tableau : définition de type

- **L'utilisation d'un tableau nécessite auparavant la définition de son type (introduit par *typedef*)**
  - **type des éléments**
  - **identificateur du type tableau**
  - **taille entre crochets**
- *Exemple* **:**

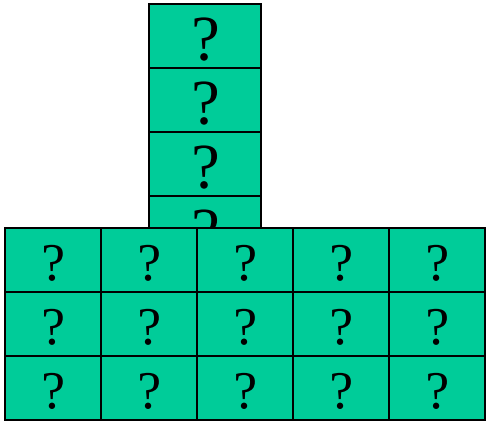
```
const int NBMAX = 100 ;

typedef int Tab1[6] ;          // définition tableau 1D
typedef float Tab2[NBMAX] ;    // définition tableau 1D
typedef int Tab3[3][5] ;       // définition tableau 2D
typedef char Tab4[2][2][3] ;   // définition tableau 3D
```

# Tableau : déclaration sans initialisation

```
const int NBMAX = 100 ;

typedef int Tab1[6] ;          // définition tableau 1D
typedef float Tab2[NBMAX] ;    // définition tableau 1D
typedef int Tab3[3][5] ;       // définition tableau 2D
typedef char Tab4[2][2][3] ;   // définition tableau 3D
```

- *Exemples de déclarations sans initialisation*

```
Tab1 a ;        // tableau 1D de 6 entiers
Tab3 d ;        // tableau 2D de 15 entiers
Tab2 b, c ;     // tableaux 1D de 100 réels
```

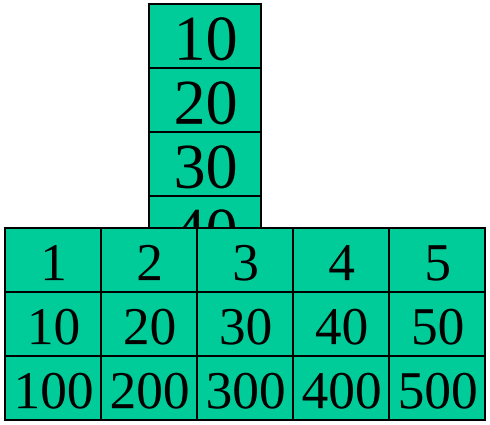


# Tableau : déclaration avec initialisation

```
const int NBMAX = 100 ;

typedef int Tab1[6] ;          // définition tableau 1D
typedef float Tab2[NBMAX] ;    // définition tableau 1D
typedef int Tab3[3][5] ;             // définition tableau 2D
typedef char Tab4[2][2][3] ;   // définition tableau 3D
```

- *Exemples de déclarations avec initialisation*

```
Tab1 x = {10, 20, 30, 40, 50, 60} ;
Tab3 y = {
              {1, 2, 3, 4, 5},
              {10, 20, 30, 40, 50},
              {100, 200, 300, 400, 500}
         } ;
```



# Tableau : opération d'accès à un élément

- **L'indice du premier élément d'un tableau est 0**

```
typedef int Tab1[6] ;
typedef int Tab3[3][5] ;
```

```
Tab1 x = {10, 20, 30, 40, 50, 60} ;
Tab3 y = {
            {1, 2, 3, 4, 5},
            {10, 20, 30, 40, 50},
            {100, 200, 300, 400, 500}
         } ;
```

```
x [1] = x [1] + 30 ;
y [2][3] = y [2][3] + y [0][0] ;
int i = 6 ;
x [2] = x [i-3] ;
y [2][2] = x [2] ;
```

y

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 10 | 20 | 30 | 40 | 50 |
| 100 | 200 | 40 | 401 | 500 |

x

| |
|---|
| 10 |
| 50 |
| 40 |
| 40 |
| 50 |
| 60 |

# Tableau : opérations globales

```
typedef int Tab1 [6] ;
```

```
Tab1 x = {10, 20, 30, 40, 50, 60} ;
Tab1 y = {2, 3, 4, 5, 6, 7} ;
```

- **Pas d'opérations globales possibles**

```
x  =  y ;                    // illégal
if (x = = y)…                // incorrect
if (x ! = y)  …              // incorrect
```

# Cas particulier : chaîne de caractères

- **Le type chaîne de caractères n'existe pas**
- **On peut manipuler des tableaux de caractères**

# Chaîne : opérations globales

```
typedef char Chaine20 [20] ;
```

```
Chaine20 ch1, ch2;
```

- **Pas d'opérations globales possibles avec les opérateurs**

```
ch1   =   ch2 ;                    // illégal
if (ch1 = = ch2)…                  // incorrect
if (ch1 ! = ch2)      …            // incorrect
```

# Chaîne : utilisation d'une bibliothèque

- **Pour manipuler des chaînes de caractères**

```
#include    "/usr/local/public/BIBLIOC++/chaine.h"
```

- *Exemple* **:**

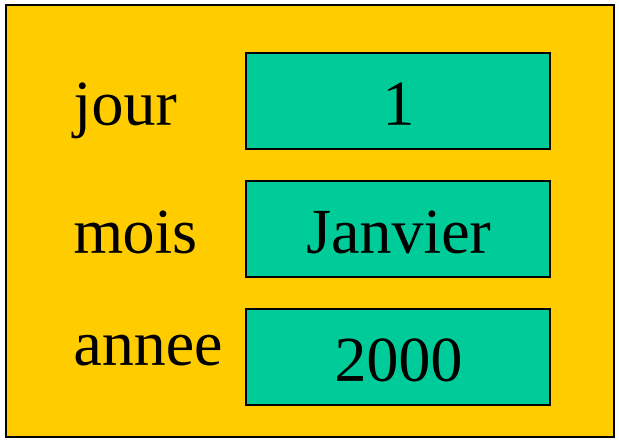
```
Chaine c1, c2 ;
c1 = uneChaine ("toto") ;
c2 = c1 ;
c1 = concatener (c1, uneChaine (" ")) ;
c1 = concatener (c1, c2) ;
if (diff (c1, c2))
{
   ecrire (uneChaine ("chaines differentes\n")) ;
}
for (int i = longueur (c2) - 1 ; i >= 0 ; i --)
{
   ecrire (ieme (c2, i)) ;
}
```

***otot***

# Enregistrement

- **Un enregistrement désigne une variable décomposée en plusieurs champs**

- *Exemple* **:**



# Enregistrement : définition de type

- **L'utilisation d'un enregistrement nécessite auparavant la définition de son type (introduit par *struct*)**
  - **identificateur du type enregistrement**
  - **liste des champs**

```
struct nom_du_type_enregistrement
{
    type1   variable1 ;
    type2   variable2 ;
    …
    typen   variableN ;
} ;
```

# Enregistrement : définition de type

- *Exemples de définitions de types enregistrements* **:**

```
struct Date
{
    int jour ;
    Chaine mois ;
    int annee ;
} ;
```

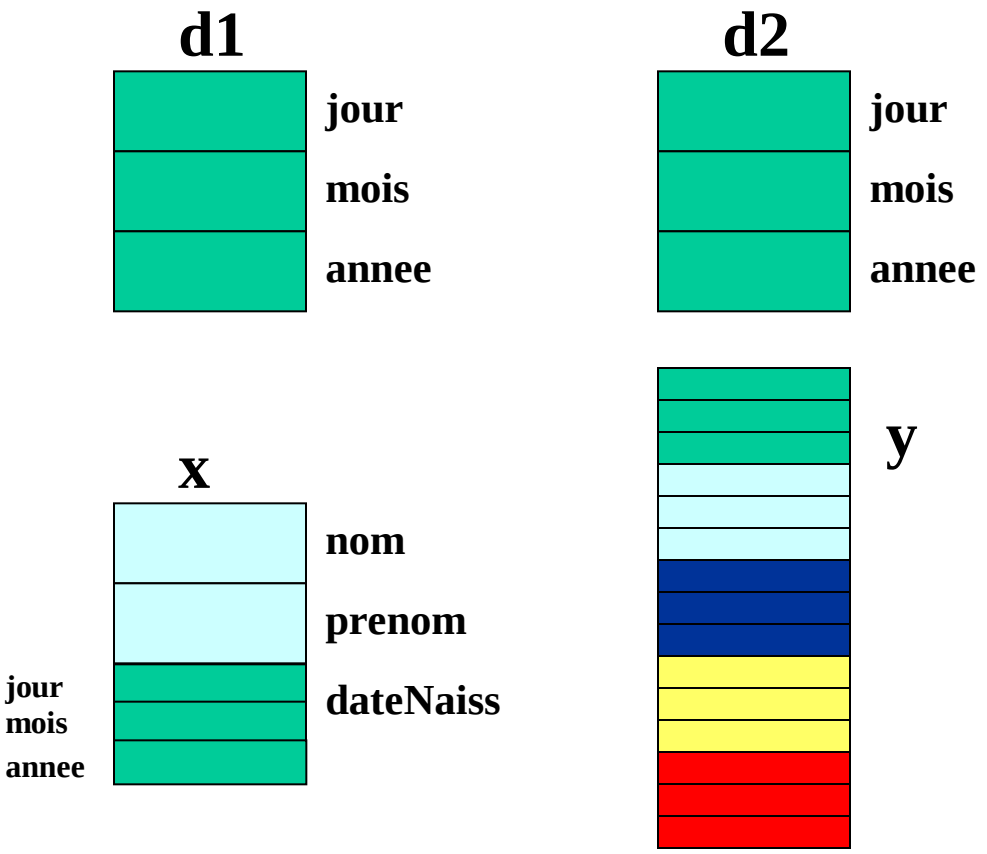
```
struct Individu
{
    Chaine nom ;
    Chaine prenom ;
    Date dateNaiss ;
} ;
```

# Enregistrement : déclaration sans initialisation

- **Déclarations d'enregistrements**

```
struct Date
{
    int jour ;
    Chaine mois ;
    int annee ;
} ;

struct Individu
{
    Chaine nom ;
    Chaine prenom ;
    Date dateNaiss ;
} ;
```

```
Date d1, d2 ;
Individu x ;

typedef Date TabDate [5] ;
TabDate y ;
```

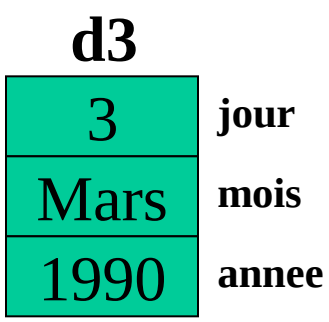


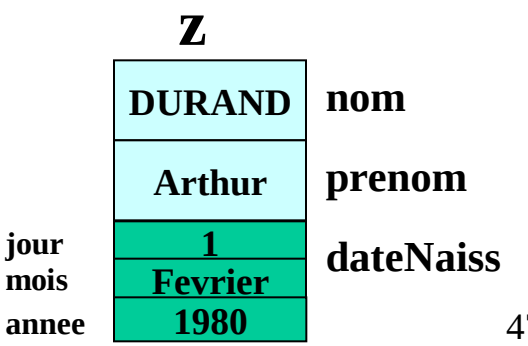
# Enregistrement : déclaration avec initialisation

- **Déclarations d'enregistrements**

```
struct Date
{
   int jour ;
   Chaine mois ;
   int annee ;
} ;

struct Individu
{
   Chaine nom ;
   Chaine prenom ;
   Date dateNaiss ;
} ;
```

```
Date d3 = {3, uneChaine ("Mars"), 1990} ;

Individu z = {uneChaine ("DURAND"),
              uneChaine ("Arthur"),
              {1, uneChaine ("Fevrier"), 1980}}  ;
```

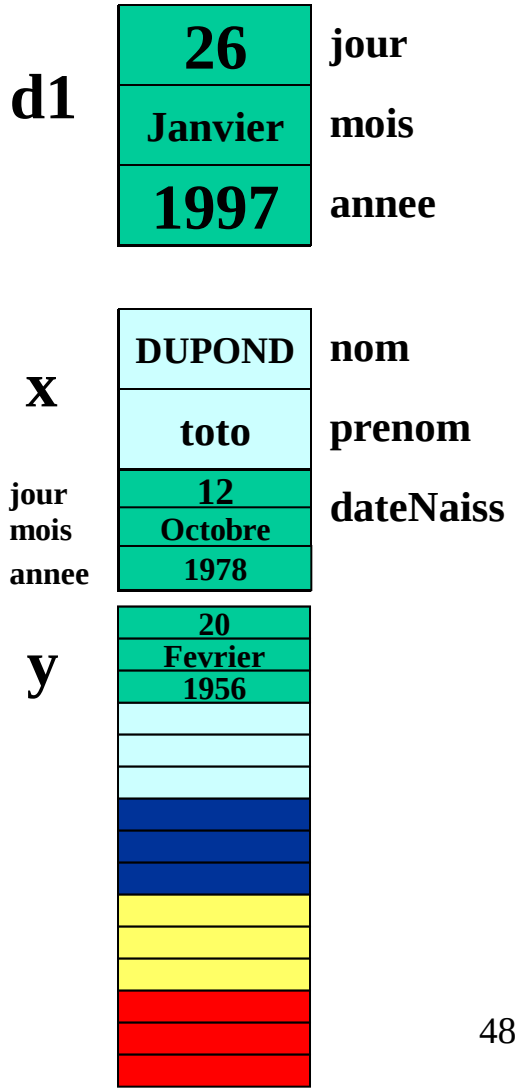


# Enregistrement : opération d'accès à un champ

- **L'accès à un champ d'un enregistrement se fait par l'intermédiaire de l'opérateur .**

```
d1.jour = 26 ;
d1.mois = uneChaine ("Janvier");
d1.annee = 1997 ;

x.nom = uneChaine ("DUPOND") ;
x.prenom = uneChaine ("toto") ;
x.dateNaiss.jour = 12 ;
x.dateNaiss.mois = uneChaine ("Octobre") ;
x.dateNaiss.annee = 1978 ;

y[0].jour = 20 ;
y[0].mois = uneChaine ("Fevrier") ;
y[0].annee = 1956 ;
```

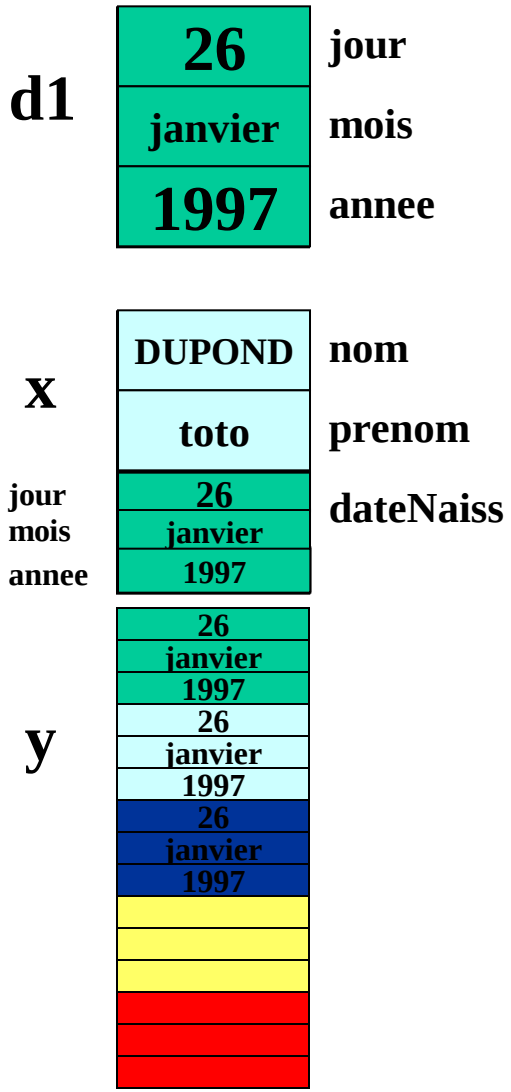


# Enregistrement : opérations globales

- **Affectation**

```
Date d1, d2 ;
Individu x ;

typedef Date TabDate [5] ;
TabDate y ;
```

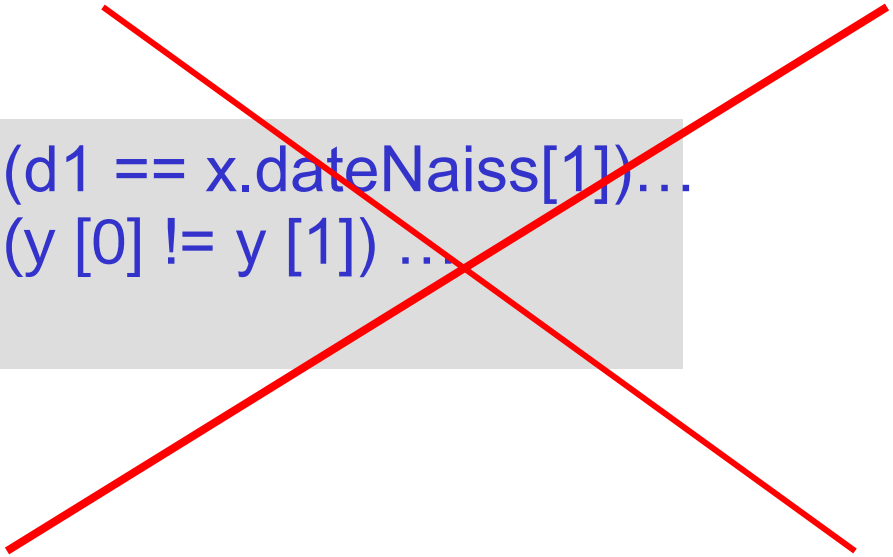
```
x.dateNaiss = d1 ;

y[0] = d1 ;
y[1] = x.dateNaiss ;
y[2] = y[1] ;
```

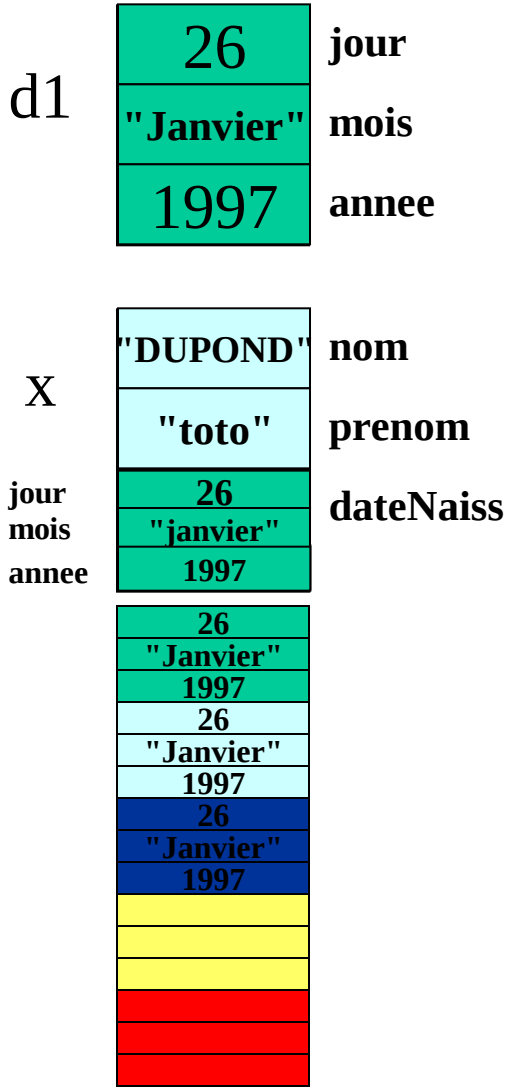


# Enregistrement : opérations globales

- **Comparaisons impossibles**

```
if (d1 == x.dateNaiss[1])…
if (y [0] != y [1]) …
```





# Sous-programmes

- **Pas de distinction entre procédures et fonctions :**
  - **Tout est fonction**
  - **Une procédure est une fonction particulière qui ne retourne aucune valeur**
  - **Le programme principal est une fonction particulière s'appelant *main***
- *Exemple* **:**

```
#include "/usr/local/public/BIBLIOC++/entreeSortie.h"
int main( )
{
        int a = 5 ;
        int b = 10 ;
        ecrire (a) ;
        ecrire (b) ;
        ecrire (a / b) ;
}
```

# Procédures

- **Syntaxe :**

```
void nom (paramètres_formels)
{
      corps
}
```

  - ***nom*** **: nom symbolique de la procédure**
  - ***paramètres_formels*** **: liste des paramètres formels**
  - ***corps*** **: déclaration des variables locales + instructions**

# Procédures

```
-- calcule et affiche s/n
procédure moyenne (entrée s <Réel>, entrée n <Entier>)

glossaire
   m <Réel> ; -- moyenne des valeurs

début
   si n /= 0 alors

      m ← s / n ;
      écrire ("moyenne : ") ;
      écrire (m) ;

   sinon
      écrire ("impossible à calculer") ;
   fin si ;
fin
```

```
// calcule et affiche la moyenne s/n
void moyenne (const float s, const int n)
{

   float m ; // moyenne des valeurs
   if (n != 0)
   {
      m = s / n ;
      ecrire (uneChaine("moyenne : ")) ;
      ecrire (m) ;
   }
   else
      ecrire (uneChaine("impossible à calculer")) ;
}
```

```
float a ;
int b ;
…
moyenne  (a, b) ;
```

# Fonctions

- **Syntaxe :**

```
type nom (paramètres_formels)
{
      corps
      return (expression) ;
}
```

  - *nom* **: nom symbolique de la fonction**
  - *paramètres_formels* **: liste des paramètres formels**
  - *corps* **: déclaration des variables locales + instructions**
  - *expression* **: valeur retournée par la fonction**

# Fonctions

- *Exemple* **:**

```
-- calcule et retourne s/n
fonction moyenne (entrée s <Réel>,
    entrée n <Entier>)
retourne <Réel>

glossaire :
   m <Réel> ; -- moyenne des valeurs

début
   m ← s / n ;
   retourner (m) ;
fin
```

```
// calcule et retourne s/n
float moyenne (const float s, const int n)
{


   float m ;   // moyenne des valeurs
```

```
float a ;
int b ;
float c ;
…
ecrire (moyenne (a, b)) ;
c = moyenne (a, b) ;
```

# Modes de transmission des paramètres

- **Paramètres de types prédéfinis ou de types enregistrements**

  - **2 modes de transmission**
    - **par valeur :** ***type paramètre***
    - **par référence :** ***type &paramètre***

  - **Règles de transmission**

| | |
|---|---|
| entrée | **const *valeur*** |
| sortie | ***référence*** |
| en mise-à-jour | ***référence*** |

# Modes de transmission des paramètres

- *Exemple* **:**

```
-- échange le contenu des deux variables
-- entières x et y
procédure échanger ( maj x <Entier>,
                     maj y <Entier>)

glossaire
   z <Entier> ; -- variable servant à l'échange

début
   z ← x ;
   x ← y ;
   y ← z ;
fin
```

```
// echange le contenu des deux variables
// entieres x et y
void echanger (int &x, int &y)

{

   int z ;  // variable servant a l'echange


   z = x ;
   x = y ;
   y = z ;
```

```
int a = 5 ;
int b = 8 ;
echanger (a, b) ;
ecrire (a) ;
ecrire (b) ;
```

# Modes de transmission des paramètres

- **Paramètres de types tableaux**

**nom d'un tableau ≡ adresse de son premier élément**

– **1 mode de transmission**

- **par adresse :** ***type paramètre***

– **Règles de transmission**

| | |
|---|---|
| entrée | ***adresse précédé de* const** |
| sortie | ***adresse*** |
| mise-à-jour | ***adresse*** |

# Modes de transmission des paramètres

- *Exemple* **:**

```
typedef int Tableau [30] ;
```

```
-- incrémente de 1 les n éléments
-- d'un tableau t
procédure incrémenterDeUn
   (mise à jour t <Tableau>,  entrée n <Entier>)
glossaire
   i <Entier> ; -- indice de parcours du tableau
début
   i ← 1 ;
   tantque i <= n faire
      t [i] ←  t [i] + 1 ;
      i ←  i + 1 ;
   fin tantque ;
fin
```

```
// incremente de 1 les n elements
// d'un tableau t
void incrementerDeUn
            (Tableau t, const int n)
{




      for (int i = 0 ; i < n ; i ++)
         t [i] = t [i] + 1 ;
}
```

```
tableau a ;
int b = 10 ;
incrementerDeUn (a, b) ;
```

# Modes de transmission des paramètres

- *Exemple* **:**

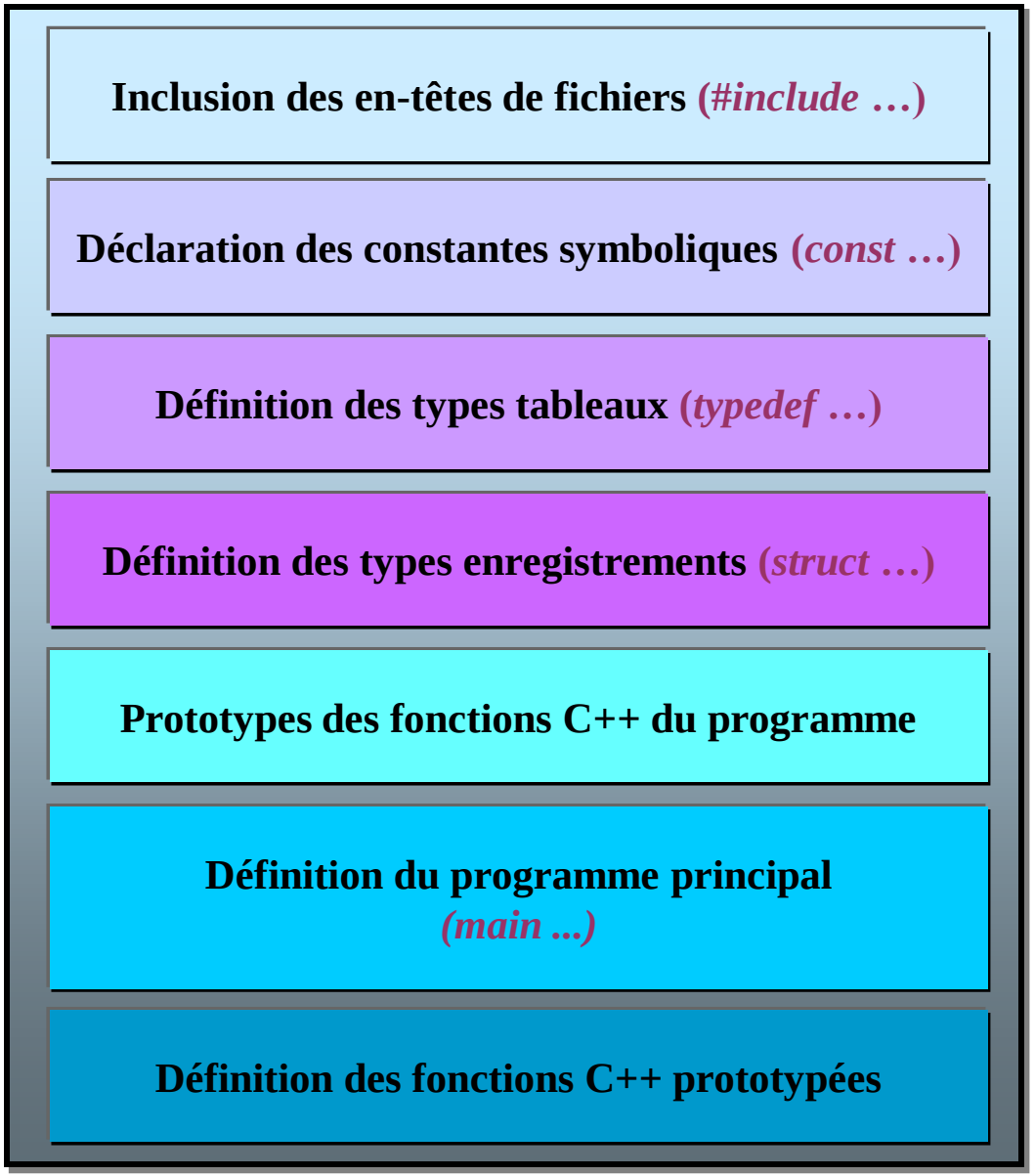
```
typedef int Tableau [5] ;
```

```
-- affiche un tableau t de n entiers
procédure afficher (entrée t <Tableau>,
 entrée n <Entier>)

glossaire
   i <Entier> ; -- indice de parcours du tableau

début
   i ← 1 ;
   tantque i <= n faire
      écrire (t [i]) ;
      i ← i + 1 ;
   fin tantque ;
fin
```

```
// affiche un tableau t de n entiers
void afficher
(const Tableau t, const int n)
{
      for (int i = 0 ; i < n ; i ++)
      ecrire (t [i]) ;
}
```

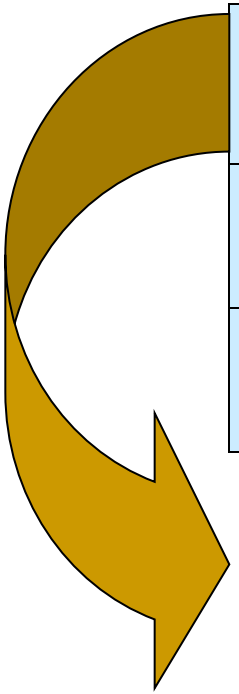
```
tableau a = {1,2,3,4,5} ;
int b = 5 ;
afficher (a ,b) ;
```

# Structure d'un programme C++



# Exceptions

- **Situation exceptionnelle à laquelle un sous-programme ne peut répondre**
  - **déroutement conduisant à l'arrêt du programme**
- **C++ possède son propre mécanisme pour signaler et traiter les exceptions**



| | |
|---|---|
| throw | ***Signale la levée d'une exception (dans l'entête et dans le corps)*** |
| try | ***Déclare l'intention de prendre en charge une exception levée*** |
| catch | ***Définit le traitement d'une exception*** |

On se limitera à lever une exception

# Levée d'une exception

- **Syntaxe :**

```
throw expression
```

```
typedef int Tableau [10] ;
```

```
-- calcule la moyenne des nbNotes notes
fonction moyenneDesNotes
    (entrée notes <Tableau>, entrée nbNotes <Enti
retourne <Réel> déclenche (aucuneNote)
glossaire
   i <Entier> : compteur des notes ;
   somme <Réel> : somme des notes ;
début
   somme ← 0 ;
   i ← 1 ;
   tantque   i <= nbNotes faire
      somme ← somme + notes [i] ;
      i ← i + 1 ;
   fin tantque ;
   si nbNotes  = 0 alors
     déclencher (aucuneNote) ;
   fin si ;
   retourner (somme  /  nbNotes) ;
fin
```

```
// calcule la moyenne des nbNotes notes
float moyenneDesNotes
    (const Tableau notes, const  int nbNotes)
 throw (Chaine)
{
   float somme = 0 ; // somme des notes



   for (int  i = 0 ; i < nbNotes ; i ++)
      somme = somme + notes [i] ;


   if (nbNotes = = 0)
      throw uneChaine ("AUCUNE NOTE") ;

   return (somme / nbNotes) ;
}
```

# Interception d ’une exception

- **Traitement par défaut :**
  - **arrêt du programme**
- *Exemple* **:**

```
typedef int tableau [10] ;
```

```
float moyenneDesNotes
    (const Tableau notes, const int nbNotes)
throw (Chaine)
{
   float somme = 0 ; // somme des notes
   for (int  i = 0 ; i < nbNotes ; i ++)
      somme = somme + notes [i] ;

   if (nbNotes == 0)
      throw uneChaine ("AUCUNE NOTE") ;

   return (somme / nbNotes) ;
}
```

*terminate ()*

```
// saisie et affiche la moyenne des notes
int main()
{
   tableau notes ;                  // tableau des notes
   int nbNotes ;                    // nombre de notes
   float uneNote ;                  // note saisie
   int i = 0;                       // compteur de notes
   // saisie des notes
   ecrire ("Entrez la premiere note ") ;
   lire (uneNote) ;
   while (uneNote <= 0 && uneNote <= 20)
   {
      notes [i] = uneNote ;
      i ++ ;
      ecrire("Entrez la note suivante ") ;
      lire (uneNote) ;
   }
   ecrire ("Moyenne : ") ;
   ecrire (moyenneDesNotes (notes, nbNotes)) ;
}
```

# Fonction surchargée

- **C++ autorise la surcharge de fonctions**
- **2 fonctions sont surchargées si :**
  - **Elles ont le même nom symbolique**
  - **Elles n'ont pas les mêmes paramètres**
- **Avantage :**
  - **garder la même sémantique pour des concepts identiques**

```
-- calcule la somme de 2 entiers
int somme (const int x, const int y) ;
```

```
-- calcule la somme de 2 réels
float somme (const float x, const float y) ;
```

```
void main ()
{
    int a = 3, b = 4 ;
    float c = 2.5, d = 1.0 ;
    ecrireNL (somme (a, b)) ;
    ecrireNL (somme (c, d), 2) ;
}
```

# Fonction générique

- **Exemple :**

```
-- permute 2 entiers
void permuter (int & a, int & b)
{
    int c ;
    c = a ;
    a = b ;
    b = c ;
}
```

```
void main ()
{
    int x = 5, y = 8 ;
    float z = 2.5, t = 8.9 ;
    permuter (x, y) ;
    permuter (z, t) ;
}
```

```
-- permute 2 réels
void permuter (float & a, float & b)
{
    float c ;
    c = a ;
    a = b ;
    b = c ;
}
```

# Fonction générique

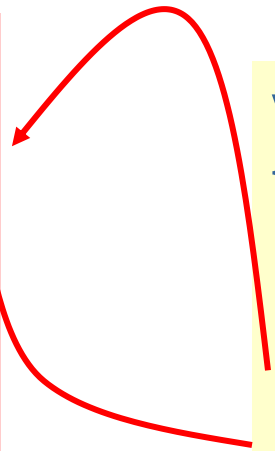
- **Fonction indépendante du type de ses arguments**
- **Syntaxe :**

```
template <class identificateur>
```

- **Exemple :**

```
-- permute 2 valeurs
template <class T>
void permuter (T & a, T & b)
{
   T c ;
   c = a ;
   a = b ;
   b = c ;
}
```

```
void main ()
{
   int x = 5, y = 8 ;
   float z = 2.5, t = 8.9 ;
   permuter (x, y) ;
   permuter (z, t) ;
}
```



# Exemple de programme C++

```
//*****************************************************************************//
// ROLE DU PROGRAMME      | rembourse plusieurs dettes a partir de billets    //
//                        | disponible en caisse                              //
//------------------------|-------------------------------------------------//
// FICHIER SOURCE         | /usr/public/mesProgrammes/dette.C                 //
//------------------------|-------------------------------------------------//
// LANGAGE                | C++                                               //
//------------------------|-------------------------------------------------//
// AUTEUR(S)              | Christine JULIEN                                  //
//------------------------|-------------------------------------------------//
// DATE CREATION          | 10/10/1998                                        //
// DATE MISE A JOUR       | 24/10/1999                                        //
//*****************************************************************************//


//-----------------------------------------------------------------------------
// IMPORTATION DES BIBLIOTHEQUES UTILISEES
//-----------------------------------------------------------------------------
#include "/usr/local/public/BIBLIOC++/entreeSortie.h"
#include "/usr/local/public/BIBLIOC++/chaine.h"

//-----------------------------------------------------------------------------
// DEFINITION DES TYPES ENREGISTREMENT
//-----------------------------------------------------------------------------
// definition du type Caisse
struct Caisse
{
    int n50;        // nombre de billets de 50 euros disponibles
    int n20;        // nombre de billets de 20 euros disponibles
    int n10;        // nombre de billets de 10 euros disponibles
};
```

# Exemple de programme C++

```
//-------------------------------------------------------------------------
// SPECIFICATION DES SOUS-PROGRAMMMES
//-------------------------------------------------------------------------
// lit 3 entiers representant respectivement le nombre de billets de
// 50, 20 et 10 euros et les range dans caisse
void lireCaisse(Caisse &caisse);

// calcule et retourne la somme disponible dans caisse
int sommeCaisse(const Caisse caisse);

// calcule la nouvelle dette d restant a payer en fonction du nombre n de
// billets disponibles en caisse d'une valeur donnee ;
// determine le nombre nb de billets delivres de cette valeur
// en remboursement de tout ou partie de d
void calculerDetteRestante(int &d, const int n, const int valeur, int &nb);

// rembourse une dette d en delivrant nb100, nb50, nb10 billets
// de valeurs respectives 50, 20 et 10 euros
// a partir de caisse ayant n50, n20 et n10 billets
// de valeurs respectives 50, 20 et 10 euros
void rembourserUneDette
    (int &d, const Caisse caisse, int &nb50, int &nb20, int &nb10);

// met a jour le montant caisse du debiteur en fonction du nombre de
// billets de 50 (nb50), 20 (nb20) et 10 (nb10) euros delivres
// pour rembourser une dette
void mettreAJourCaisse
    (Caisse &caisse, const int nb50, const int nb20, const int nb10);
...
```

# Exemple de programme C++

```
//-------------------------------------------------------------------------
// PROGRAMME PRINCIPAL
//-------------------------------------------------------------------------
int main()
{
    Caisse maCaisse;// enregistrement correspondant a la caisse
    int uneDette;   // montant d'une dette
    int x50;        // nombre de billets de 50 euros delivres pour une dette
    int x20;        // nombre de billets de 20 euros delivres pour une dette
    int x10;        // nombre de billets de 10 euros delivres pour une dette
    int nbDettes;   // nombre de dettes a rembourser
    int cptDettes;  // nombre de dettes examinees

    effacer();
    ecrireNL(uneChaine("   **** REMBOURSEMENT DE PLUSIEURS DETTES ****"));
    ecrireNL();
    ecrireNL();
    // lire le nombre de billets disponible en caisse
    lireCaisse(uneCaisse);
    // calculer et ecrire la somme disponible en caisse
    ecrire(uneChaine("Montant de la somme disponible en caisse : "));
    ecrireNL(sommeCaisse (uneCaisse));
    ecrireNL();
    // lire le nombre de dettes a rembourser
    ecrire(uneChaine("Entrer le nombre de dettes a rembourser : "));
    lire(nbDettes);
    ...
    ecrireNL(uneChaine("Fin de remboursement des dettes"));
}
```

# Exemple de programme C++

```
//-------------------------------------------------------------------------
// DEFINITION DES SOUS-PROGRAMMMES
//-------------------------------------------------------------------------

void lireCaisse(Caisse &caisse)
{
    ecrire (uneChaine ("Entrer le nombre de billets de 50 euros disponibles : "));
    lire (caisse.n50);
    ecrire (uneChaine ("Entrer le nombre de billets de 20 euros disponibles : "));
    lire (caisse.n20);
    ecrire (uneChaine ("Entrer le nombre de billets de 10 euros disponibles : "));
    lire (caisse.n10);
}

int sommeCaisse(const Caisse caisse)
{
    return (caisse.n50 * 50 + caisse.n20 * 20 + caisse.n10 * 10);
}

void initialiserCompteur(int &compteur, const int valeur)
{
    compteur = valeur;
}

void augmenterDeUn (int &compteur)
{
    compteur++;
}

...
```

# Les entrées/sorties

```
//-------------------------------------------------------------------------
// SPECIFICATION DES SOUS-PROGRAMMES DE LECTURE SUR L'ENTREE STANDARD
// CHAQUE LECTURE DOIT ETRE VALIDEE PAR L'APPUI SUR LA TOUCHE <ENTREE>
//-------------------------------------------------------------------------

// attend l'appui sur la touche <ENTREE>
void lire();

// lit un booleen v
void lire(bool &v);

// lit un caractere v
void lire(char &v);

// lit un entier court v
void lire(short &v);

// lit un entier court non signe v
void lire(unsigned short & v);

// lit un entier v
void lire(int &v);

// lit un entier non signe
void lire(unsigned int &v);

// lit un entier long v
void lire (long &v);

// lit un entier long non signe v
void lire(unsigned long &v);

// lit un reel v
void lire(float &v);

// lit un reel double precision v
void lire(double &v);
```

# Les entrées/sorties

```
//-------------------------------------------------------------------------
// SPECIFICATION DES SOUS-PROGRAMMES D'ECRITURE SUR LA SORTIE STANDARD
// LES IDENTIFICATEURS DE SOUS-PROGRAMMES SE TERMINANT PAR NL POSITIONNENT
// LE CURSEUR AU DEBUT DE LA LIGNE SUIVANTE APRES AVOIR EFFECTUE L'ECRITURE
//-------------------------------------------------------------------------

// positionne le curseur en debut de ligne suivante
void ecrireNL();

// ecrit un booleen v
void ecrire(const bool v);
void ecrireNL(const bool v);

// ecrit un caractere v
void ecrire(const char v);
void ecrireNL(const char v);

// ecrit un entier long v
void ecrire(const long v);
void ecrireNL(const long v);

…

// ecrit un entier long non signe v
void ecrire (const unsigned long v);
void ecrireNL(const unsigned long v);

// ecrit un reel v avec nb chiffres apres le point decimal
void ecrire(const float v, const int nb);
void ecrireNL(const float v, const int nb);

// ecrit un reel double precision v avec nb chiffres apres le point decimal
void ecrire(const double v, const int nb);
void ecrireNL(const double v, const int nb);
```

# Les entrées/sorties

```
//-----------------------------------------------------------------------
// SPECIFICATION DES SOUS-PROGRAMMES DE GESTION DE L'AFFICHAGE ECRAN
//-----------------------------------------------------------------------

// remet a blanc l'ecran et positionne le curseur en haut a gauche de l'ecran
void effacer();

// positionne l'affichage en mode video inverse
void inverser();

// positionne l'affichage en mode normal
void normal();

// positionne l'affichage en gras
void gras();

// emet un beep sonore
void beep();

// positionne le curseur en ligne l et colonne c
void allerLC(const int l, const int c);
```

# Les chaînes de caractères

```
//-------------------------------------------------------------------------
// SPECIFICATION DES SOUS-PROGRAMMES DE MANIPULATION DU TYPE ABSTRAIT DE
// DONNEE Chaine
//-------------------------------------------------------------------------

// constante designant une chaine vide
Chaine CHAINE_VIDE();

// construit une chaine a partir d'un tableau de caracteres terminé par '\n'
// leve l'exception "LONGUEUR INVALIDE" si la chaine resultat depasse
// 255 caracteres
Chaine uneChaine(const TabCar t);

// renvoie le nombre de caracteres effectif d'une chaine ch
int longueur(const Chaine ch);

// renvoie une chaine resultant de la concatenation de ch2 a ch1
// leve l'exception "LONGUEUR INVALIDE" si la chaine resultat
// depasse 255 caracteres
Chaine concatener(const Chaine ch1, const Chaine ch2);

// renvoie le resultat de l'expression ch1 == ch2
bool egal(const Chaine ch1, const Chaine ch2);

// renvoie le resultat de l'expression ch1 < ch2 (ordre lexicographique)
bool inf(const Chaine ch1, const Chaine ch2);
```

# Les chaînes de caractères

```
// renvoie le resultat de l'expression ch1 > ch2 (ordre lexicographique)
bool sup(const Chaine ch1, const Chaine ch2);

// renvoie le resultat de l'expression ch1 != ch2
bool diff(const Chaine ch1, const Chaine ch2);

// renvoie le resultat de l'expression ch1 <= ch2 (ordre lexicographique)
bool infEgal(const Chaine ch1, const Chaine ch2);

// renvoie le resultat de l'expression ch1 >= ch2 (ordre lexicographique)
bool supEgal(const Chaine ch1, const Chaine ch2);

// lit une chaine ch sur l'entree standard
void lire(Chaine & ch);

// ecrit une chaine ch sur la sortie standard
void ecrire(const Chaine ch);
```

# Mots réservés

| | | | | |
|---|---|---|---|---|
| and | and_eq | asm | auto | bitand |
| bitor | bool | break | case | catch |
| char | class | compl | const | const_cast |
| continue | default | delete | do | double |
| dynamic_cast | else | enum | explicit | extern |
| false | float | for | friend | goto |
| if | inline | int | long | mutable |
| namespace | new | not | not_eq | operator |
| or | or_eq | private | protected | public |
| register | reinterpret_cast | return | short | signed |
| sizeof | static | static_cast | struct | switch |
| template | this | throw | true | try |
| typedef | typeid | typename | union | unsigned |
| using | virtual | void | volatile | wchar_t |
| while | xor | xor_eq | | |

# Types prédéfinis par le langage

| *Type C++* | *Signification* | *Taille* | *Valeurs* |
|---|---|---|---|
| **bool** | Booléen | *1 octet* | true (= 1) ou<br>false (= 0) |
| **char** | Caractère | *1 octet* | 256 caractères<br>code ASCII |
| **short** | Entier court | *2 octets* | De -32 768<br>à 32 767 |
| **unsigned short** | Entier court non signé | *2 octets* | De 0<br>à 65 535 |
| **int** | Entier court ou long | *2 octets*<br>*ou 4 octets* | |
| **unsigned int** | Entier court ou long non signé | *2 octets*<br>*ou 4 octets* | |
| **long** | Entier long | *4 octets* | De -2 147 483 648<br>à 2 147 483 647 |
| **unsigned long** | Entier long non signé | *4 octets* | De 0<br>à 4 294 967 295 |
| **float** | Réel | *4 octets* | De 1,2 e-38<br>à 3,4 e38 |
| **double** | Réel double précision | *8 octets* | De 2,2 e-308<br>à 1,8 e308 |

# Priorités des opérateurs

| *Priorité* | *Opérateurs* |
|---|---|
| 8 | `++  --  -(unaire)  +(unaire)` |
| 7 | `*  /  %` |
| 6 | `+(binaire)  -(binaire)` |
| 5 | `<  >  <=  >=` |
| 4 | `==  !=` |
| 3 | `&&` |
| 2 | `\|\|` |
| 1 | `=` |

# Caractères spéciaux

| *Caractère* | *Signification* |
|---|---|
| **\n** | Nouvelle ligne (retour chariot et saut de ligne) |
| **\b** | Retour arrière |
| **\t** | Tabulation horizontale |
| **\f** | Saut de page |
| **\a** | Signal sonore |
| **\"** | Guillemets |
| **\'** | Quote |
| **\0** | Caractère nul |
| **** | antislash |

# Table des codes ASCII

| *Caractère* | *Valeur Décimale* | *Caractère* | *Valeur Décimale* | *Caractère* | *Valeur Décimale* | *Caractère* | *Valeur Décimale* |
|---|---|---|---|---|---|---|---|
| Ctrl @ | 0 | | 32 | @ | 64 | ’ | 96 |
| Ctrl A | 1 | ! | 33 | A | 65 | a | 97 |
| Ctrl B | 2 | " | 34 | B | 66 | b | 98 |
| Ctrl C | 3 | # | 35 | C | 67 | c | 99 |
| Ctrl D | 4 | $ | 36 | D | 68 | d | 100 |
| Ctrl E | 5 | % | 37 | E | 69 | e | 101 |
| Ctrl F | 6 | & | 38 | F | 70 | f | 102 |
| \a | 7 | ‘ | 39 | G | 71 | g | 103 |
| \b | 8 | ( | 40 | H | 72 | h | 104 |
| \t | 9 | ) | 41 | I | 73 | i | 105 |
| \n | 10 | * | 42 | J | 74 | j | 106 |
| \v | 11 | + | 43 | K | 75 | k | 107 |
| \f | 12 | , | 44 | L | 76 | l | 108 |
| \r | 13 | - | 45 | M | 77 | m | 109 |

# Conventions d'appellations

| | |
|---|---|
| Identificateur de type | `typedef char Chaine[100+1];`<br><br>`struct Date`<br>`{`<br>`  int jour;`<br>`  int mois;`<br>`  int annee;`<br>`};` |
| Identificateur de variable | `int moyenne;`<br><br>`Date dateDuJour;` |
| Identificateur de fonction C++ | `float somme(const float x, const float y);`<br><br>`Date calculerDateDuJour();` |
| Identificateur de constante symbolique | `const float PI = 3.14;`<br><br>`const int NB_MAX_TAB = 100;` |