# Chap0-Généralités

| **Licence informatique** | | |
|---|---|---|
| Deuxième année L2 | | |
| UEF 1 (fondamentale) = 15 crédits<br>Pour passer au L3 $\Rightarrow$ 90 Crédits+ Unités Fondamentales | | |
| Architecture des ordinateurs = 5 crédits | Algorithmique et structures de données (ASD) = 6 crédits | Logique Mathématique (LM) = 4 crédits |

| Unité d'Enseignement | VHS | V.H hebdomadaire | | | | Coeff | Crédits | Mode d'évaluation | |
|---|---|---|---|---|---|---|---|---|---|
| | 15 sem | C | TD | TP | Autres | | | Continu | Examen |
| **UE fondamentales** | | | | | | | | | |
| **UEF1** | 180h | 6h | 3h | 3h | 0h | 7 | 15 | | |
| **Architecture des Ordinateurs (AO)** | 45h | 1h30 | | 1h30 | | 2 | 5 | X | **X** |
| **Algorithmique et Structures de Données (ASD)** | 90h | 3h | 1h30 | 1h30 | | 3 | 6 | X | **X** |
| **Logique Mathématique (LM)** | 45h | 1h30 | 1h30 | | | 2 | 4 | X | **X** |
| **UEF2** | 157h30 | 4h30 | 4h30 | 1h30 | 0h | 8 | 13 | | |
| **Programmation orientée objet (POO)** | 67h30 | 1h30 | 1h30 | 1h30 | | 3 | 5 | X | **X** |
| **Systèmes d'Information** | 45h | 1h30 | 1h30 | | | 3 | 4 | X | **X** |
| **Option :**<br>**- Théorie des Langages**<br>**- Méthodes Numériques** | 45h | 1h30 | 1h30 | | | 2 | 4 | X | **X** |
| **UE méthodologie** | | | | | | | | | |
| **UEM1** | 22h30 | 0h | 1h30 | 0h | 0h | 1 | 2 | | |
| **Langue Anglaise 2** | 22h30 | | 1h30 | | | 1 | 2 | X | **X** |
| **Total Semestre3** | **360h** | **10h30** | **09h** | **4h30** | **0h** | **16** | **30** | | |

# Programme du S3

- Analyse d'algorithmes (complexité).
- Concepts de base de l'orienté objet.
- Concepts avancés : généricité, traitement d'exceptions, interfaces …
- Récursivité.
- Structures séquentielles: piles, files et listes.
- Structures hiérarchiques: arbres, arbres binaires, arbres de recherche, les tas et les files de priorité.
- Algorithmes de tri.
- Les ensembles.

# Bibliographie


- Mark Allen Weiss, Data Structures and Algorithm Analysis in Java, Pearson, Third Edition, 2012.
- William J. Collins, Data Structures and the Java Collections Framework, Wiley, 2011.
- Types de données et algorithmes (Livre non disponible à la BU)
- Introduction à l'algorithmique, 04-07-454
- Algorithmique, 04-04-55
- Algorithmes et structures de données, 04-07-145
- Algorithmique et programmation en PASCAL, 04-07-176 et 04-04-177

# Définition d'un algorithme

Définition retenue: Un algo décrit un traitement sur un certain nombre fini de données pour fournir des résultats.

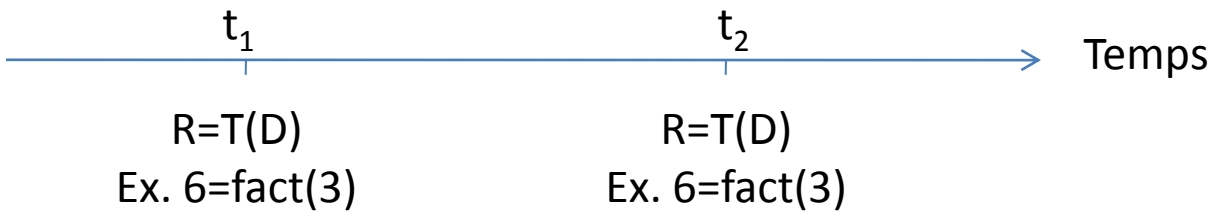

Traitement = { opérations rigoureuses et effectives }

Non ambigües
Ex. 2+3*4/5

Réalisables par machine
Ex. Faire du café ??

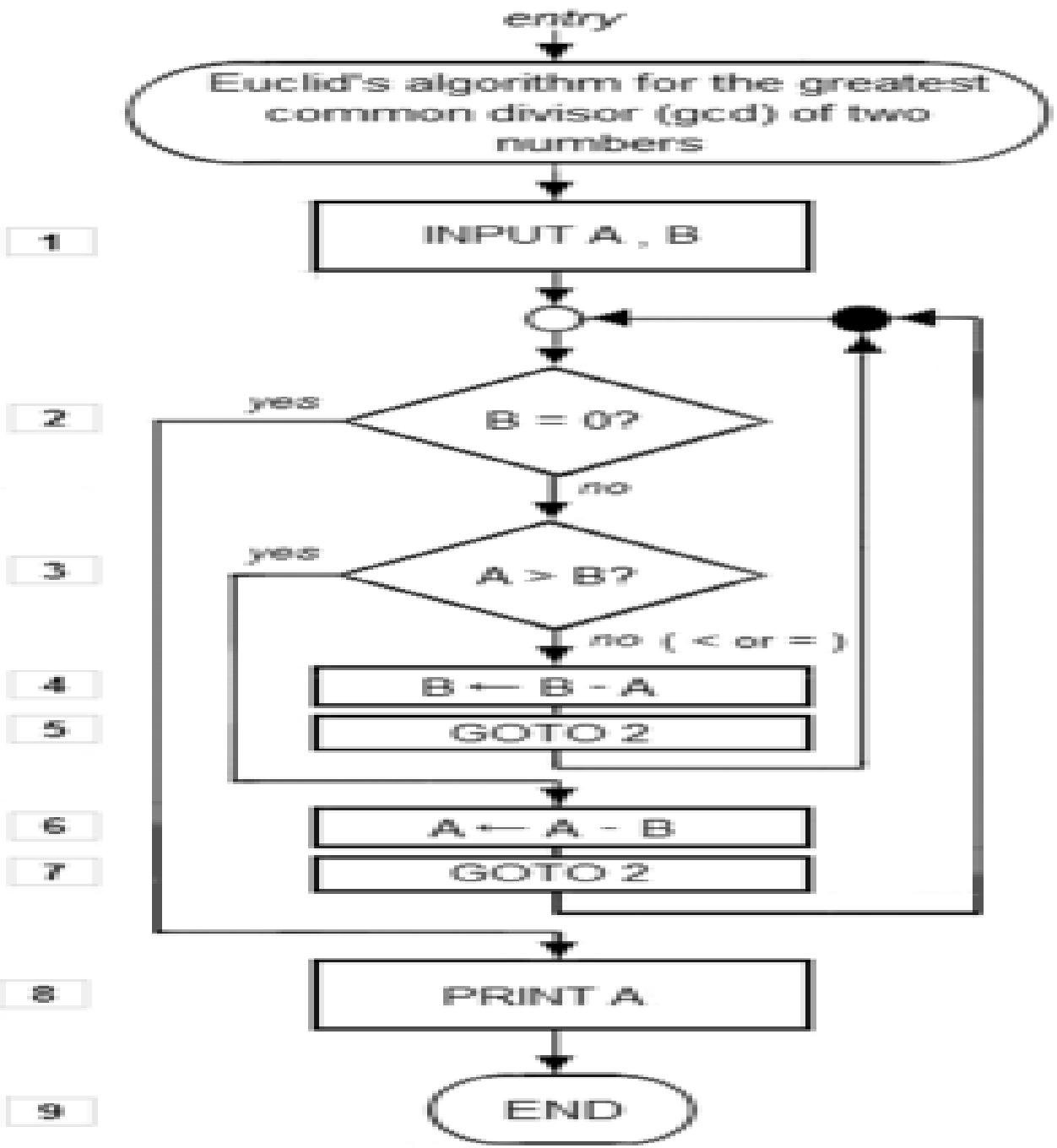
- Un algo doit se terminer après un **nbre fini d'opérations** et fournir un **résultat**.

- Les algos considérés sont **déterministes**




Expression des algos : dans un lgge de pgtion pour être compris et exécuté par ordinateur (Pascal, C, Java).

- Un algo est indépendant du lgge de pgtion utilisé. Exemple : Algo d'Euclide, Factoriel, PGCD . . . Etc.


entry
Euclid's algorithm for the greatest common divisor (gcd) of two numbers
1
INPUT A , B
2
yes
B = 0?
no
3
yes
A > B?
no ( < or = )
4
B ← B - A
5
GOTO 2
6
A ← A - B
7
GOTO 2
8
PRINT A
9
END

# Exercices de TD N° 00

## Exercice 01:

Ecrire un programme qui demande à l'utilisateur de taper 5 entiers et qui affiche leur moyenne. Le programme ne devra utiliser que 2 variables.

## Exercice 02:

Ecrire un programme qui demande à l'utilisateur de taper le prix HT d'un tél portable, le nombre de tél achetés, le taux de TVA (Exemple 7%, 10%,...). Ensuite, le programme affiche le prix TTC des tél achetés.

**Exercice 03:** Supposant x, y entiers.

Programme calculant y tel que :

$$Y = \begin{cases} x+8 \quad \text{Si} \quad x \leq 10 \\ x^2 + x + 10 \quad \text{Si } 10 < x < 100 \\ 0 \quad \text{Si} \quad x \geq 100 \end{cases}$$

**Exercice 04:**

Réécrire la suite de if en utilisant l'instruction switch:

```
if( x<=1) x=x+4;
        else if( x==2) x=x+8;
                else if( x==3) x=x+12;
                        else if( x==4) x=x+16;
                        else if( x>=5) x=x+20;
```

**Exercice 05:**

Programme résolvant l'équation $Ax^2+Bx+C=0$.

**Exercice 06:** boucle for

Somme des N premiers entiers à partir de 0.

**Exercice 07:** Boucle do while

Somme de N nombres entiers donnés.

**Exercice 08:** Boucle while

fonction indiquant si un entier donné N est premier.

**Exercice 09:** Ecrire une fonction qui indique si une chaîne de caractères est un palindrome ou non.

Un mot palindrome se lit de la même façon de gauche à droite et de droite à gauche.

Exemples: selles, radar, été, rêver, Laval

**Exercice 10 :** Ecrire un programme qui permet de créer une liste chaînée d'étudiants décrits par : Nom, Age, Moyenne. Trier la liste par moyenne.

**Exercice 11 :** Ecrire une fonction qui a en paramètre une chaîne de caractères (paramètre en entrée et en sortie) et qui supprime toutes les voyelles.

# Langages de programmation C/C++

## Éléments de base

# Structure d'un programme en C

```
#include <fichier-entête>
.   .   .

type1 fonction1()
{       .       .       .       }

type2 fonction2()
{       .       .       .       }

        .       .       .

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

# Déclaration des variables en C

type v1, v2, . . . ;  /* n'importe où dans le pgme */

Exemples:

```
int i, j;
int compte=0;
float pi=0.345E+1;          /*  Initialisation à la déclaration */
char c1='a';                /*un caractère*/
char  c2[10]="Bonjour" ;    /*  un tableau de caractères*/
string prenom("Mohamed");   /*  #include <string> */
int tabl[10];               /*  un tableau d'entiers */
float matrice[10][5];       /*  une matrice de réels  */
```

- Le nom d'une variable commence par une lettre ou un tiret de soulignement '_'

# Entrées/Sorties

```
#include <iostream>
using namespace std;

void main()
{
    int x;                                  // var x: integer;
    cout << "x = " << endl;                 //   writeln("x=");
    cin >> x;                               //  readln(x);
    float y, z;                             // var y, z: real;
    cout << "  Donner deux nombres réels :  "<<"\n"  ;
    cin >> y >> z;                          //  readln(y, z);

}
```

## Instructions

**Affectation :** vble=exp; exple: a=5;

**Instr composée ou Bloc:** { $inst_1$; $inst_2$; . . . ; $inst_n$;}

exple: { cin>> a; cin>>b; a:=a+b; cout << a;}

**Conditionnelle simple :** if (cond) inst; exple: if (a<b) cout << a;

**Alternative:** if (cond) inst; else inst;

Exple: if (a<b) cout <<a ; else cout << b;

**switch :**

```
switch(vble)
 {  case v1 : inst1; break;        /* break permet de sortir du switch */
    case v2 : inst2; break;
      . . .
    case vn : inst3; break;
    default : instDef;
 }
```

Exemple: int a, b; char c;

```
cin>>a; cin>>b;
c=getch();     // cin>>c;
switch (c)     { case '+' : cout << a+b; break;
                  case '*' : cout << a*b; break;
                  case '-' : cout << a-b; break;
                 default : cout << " erreur !! "; }
```

**Boucle while:** while (exp) inst;

Exemple: i=0;

```
while (i<10) { i=i+5; i=i+i/6;};
```

**Boucle do ..while:** do inst while (cond);

Exemple: i=0; do i=i+5;

```
while (i<5);
```

**Boucle for :** for (init; fin; increment) corps; /*Ordre !!! */

Exemple1:

```
j=0;max=3; for (i=0; i<=max; i++) j++; cout << "j=="<<j;
```

La boucle while équivalente à cette boucle :

```
j=0;max=3; i=0;
while (i<=max) {j++; i++;};
cout << "j=="<<j;
```

Exemple2:

```
fin=10;   for  (i=0, x=1, y=2; i<=fin; i++, y++)  x++ ;
```

**En utilisant while :**

```
i=0; x=1; y=2; fin=10;
while (i<=fin) { x++; i++; y++;};
```

**<u>Définition des types structures :</u> (Record en pascal)**

```
struct typ_record
    { type1 champ1;
      type2 champ2;
      . . .
      typen champn;      } ;
struct   typ_record   vble_record;
```

ou bien

```
struct typ_record
        { type1 champ1;
          type2 champ2;
          . . .
          typen champn;
        }   vble_record;
```

L'accès aux champs se fait par le point '.'

<u>Exemple:</u>

```
    struct Etud
          { char nom[20];
            char pnom[20];
            int age;          };
    Etud E1;
```

Accès ensuite par :

```
E1.nom="mohamed";
cin >> E1.age;
```

**Incrémentation :**

```
int nombreJoueur(4);   //Il y a 4 joueurs dans la partie
nombreJoueur = nombreJoueur + 1;    //On en ajoute un
```

**Pré-incrémentation:**

```
int nombreJoueur(4);   //Il y a 4 joueurs dans la partie
++nombreJoueur;        //À partir d'ici, il y a 5 joueurs
```

**Post-incrémentation:**

```
nombreJoueur++;
```

**Décrémentation :**

```
-- nombreJoueur;        //pré-décrémentation
nombreJoueur--;         //post-décrémentation
```

## Les autres opérations:

double nombre(456);
nombre /= 3; // Raccourci de : nombre=nombre/3;

Il existe des raccourcis pour les 5 opérations de base : +=, -=, *=, /= et %=

## Exemple :

```
#include <iostream>
using namespace std;

int main()
{
  double nombre(5.3);
  nombre += 4.2;      //'nombre' vaut maintenant 9.5
  nombre *= 2.;       //'nombre' vaut maintenant 19
  nombre -= 1.;       //'nombre' vaut maintenant 18
  nombre /= 3.;       //'nombre' vaut maintenant 6
}
```

**Tableaux statiques:** type nom_tab[taille];

taille est une valeur constante.

L'indice du 1er élément est 0.

L'indice du dernier élément est taille-1.

Exemple : float notes[10]; int nombre[10];

**Tableaux dynamiques :** La taille peut changer

```
#include <vector>
vector<int> tableau(5);  // tabl dyn ent à 5 Elem
vector<int> tableau(5, 3);          //Crée un tableau de 5 entiers valant tous 3
vector<string> listeNoms(12, "blabla");
                    //Crée un tableau de 12 strings valant toutes "blabla"
vector<int> tableau; //Crée un tableau d'entiers  vide; on le remplit après.
```

**Changer la taille par ajout :**

```
vector<int> tab(3,2);  //Un tableau de 3 entiers valant tous 2
tab.push_back(8);
// ajoute une 4ème case au tableau qui contient la valeur 8
tableau.push_back(7);     // ajoute une 5ème case qui contient la valeur 7
tableau.push_back(14);   // ajoute une 6ème case qui contient la valeur 14
    //Le tableau contient maintenant les nombres : 2 2 2 8 7 14
```

**Changer la taille par suppression :**

```
vector<int> tableau(3,2);              //Un tableau de 3 entiers valant tous 2
tableau.pop_back();        // Supprimer la 3ème case
tableau.pop_back();        // Supprimer la 2ème case
```

tableau.size() retourne la taille du tableau.

```
//Une fonction recevant un tableau d'entiers en argument
void fonction(vector<int>  a)
    {   //…   }
```

**Chaîne de caractères:** est un tableau de char. Toute chaîne se termine par le caractère nul '\0'

Opérations sur les chaînes:

strlen(s) = longueur de s.

**Exemple:** strlen("ABCDEFG")=7

/* Pour utiliser ces fonctions, il faut mettre #include **<cstring>** */

strcpy(s1,s2) : copie s2 dans s1

**Exemple:** s1="ABCDEFG" s2="XYZ"

Après strcpy(s1, s2) : s1="XYZ" s2="XYZ"

strcat(s1, s2) : s1=s1+s2

**Exemple:** s1="ABCDEFG" s2="XYZ"

Après strcat(s1, s2) : s1=" ABCDEFGXYZ" s2="XYZ"

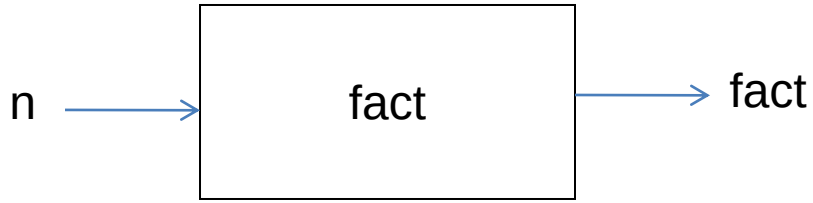
strncat (s1, s2, n) copie n chars de s2 dans s1

**Exemple:** s1="ABCDEFG" s2="XYZ"

Après strncat(s1, s2) : s1=" ABCDEFGXY" s2="XYZ"

# Définition de fonctions

```
int som(int a, int b)
    {
          return (a+b);
    }
int fact(int n)
    {  int f=1;
       for (int i=1; i<=n; i++) f *=i;
       return (f);
    }
```


**En lgge C, les définitions des fonctions ne peuvent pas être imbriquées.**

```
void trier( . . .  )
{   void permut ( . . . )          /* Interdit en C */
     . . .
}
```

# Passage des arguments par référence en C++:

```
#include <stdio.h>
void plus(int  &inval);
                // Préfixer le paramètre sortie  par &
main()
    {  int i=1;
       cout<< " \n valeur intiliale de i : \n"<< i;
       plus(i);
       cout<<" \n Après appel de plus, ";
       cout<<" la valeur de i devient : \n"<< i;
    }
void plus(int  &inval)   // inval est passée par référence
    { inval++;

      cout<<" \n Dans la fct plus, ";
       cout<<" la valeur de inval est : \n"<< inval ;
    }
```

**Résultat :**

```
valeur initiale de i est :  1
Dans la fct plus, la valeur de inval est : 2
Après appel de plus, la valeur de i devient : 2
```

La valeur de i a changé, au niveau de main(); après l'appel de la fct plus; c'est le passage par référence.

La fct plus travaille sur l'argument passé par référence.

# Pointeurs

- *Un pointeur est une variable qui contient l'adresse d'une autre variable.*

**int *pointeur;**

//déclare un pointeur qui contient l'adresse d'une variable de type int.

**float *pointeurA;**

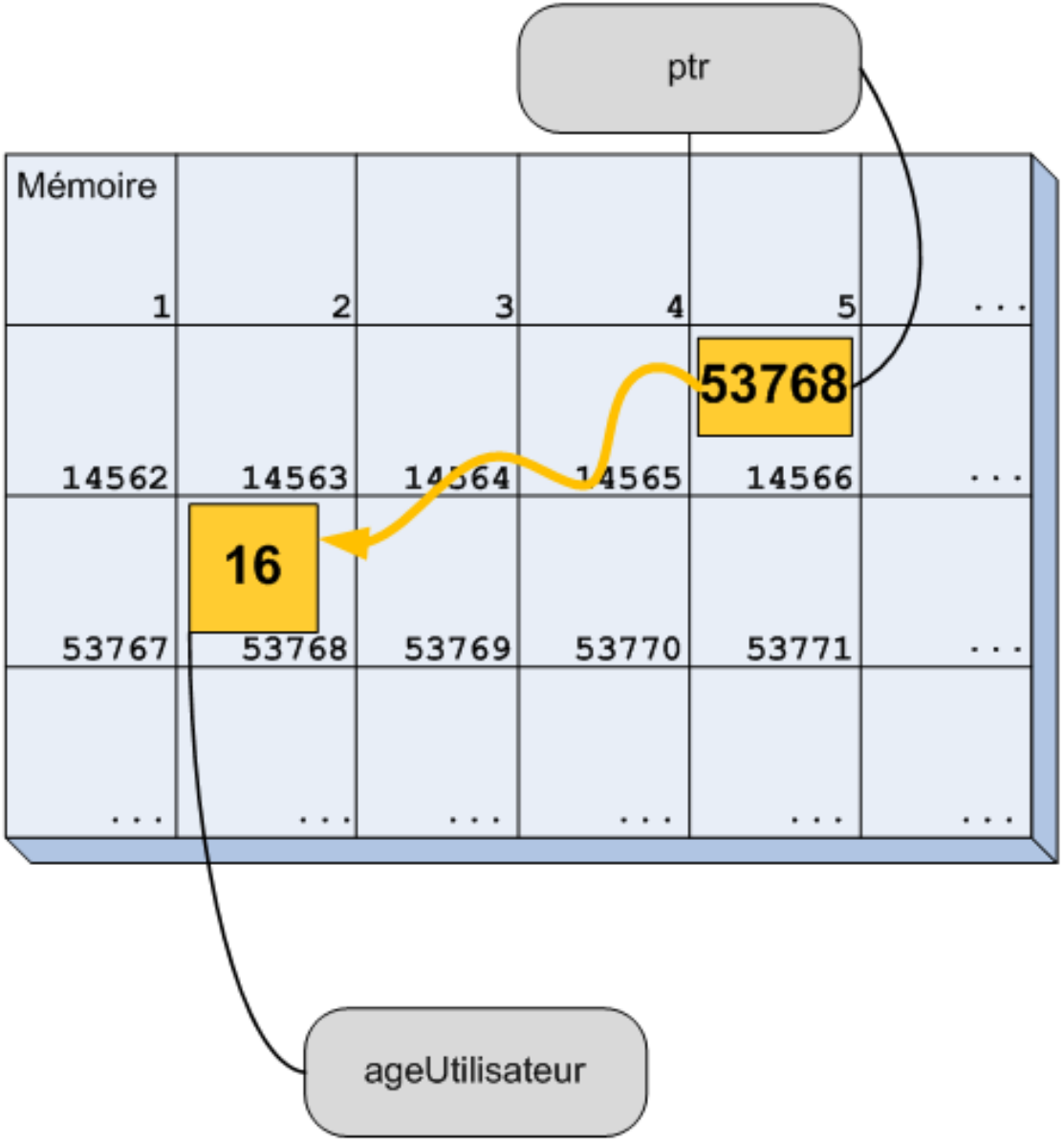
//Un pointeur qui peut contenir l'adresse d'un nombre réel

**int *pointeur(0);** // déclare un pointeur entier et l'initialise à 0

**double *pointeurA(0);** // déclare un pointeur réel et l'initialise à 0

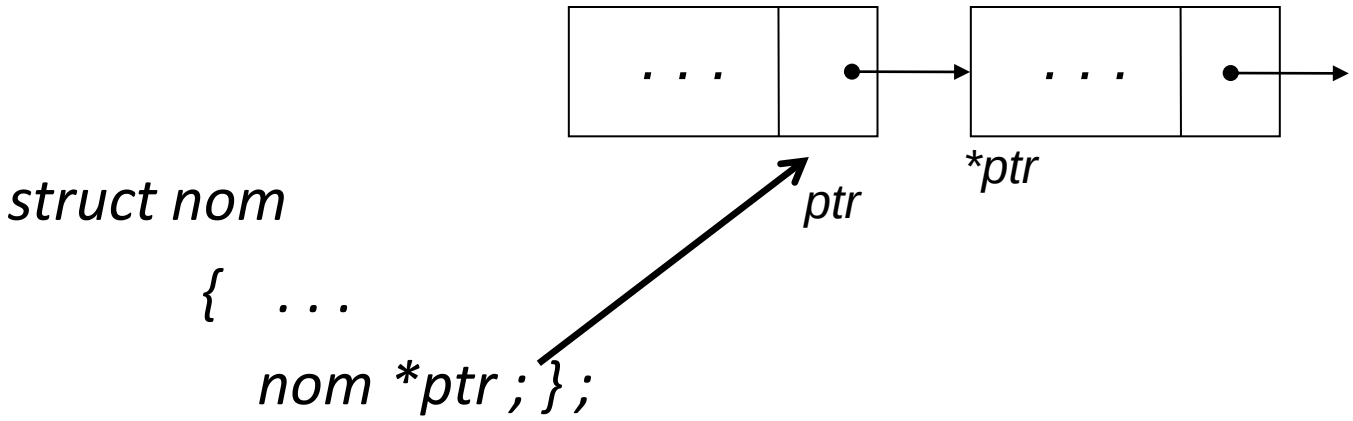
```
int main()
{    int ageUtilisateur(16);      //Une variable de type int
    int *ptr(0);
 //Un pointeur pouvant contenir l'adresse d'un nombre entier, initialisé à 0
    ptr = &ageUtilisateur;
   //On met l'adresse de 'ageUtilisateur' dans le pointeur 'ptr'
}
```

```
int main()
{    int ageUtilisateur(16);
//Une variable de type int

    int *ptr(0);
 //Un pointeur pouvant contenir l'adresse d'un nombre entier

    ptr = &ageUtilisateur;
   //On met l'adresse de 'ageUtilisateur' dans le pointeur 'ptr'
}
```



# Utilisation de pointeurs de structures

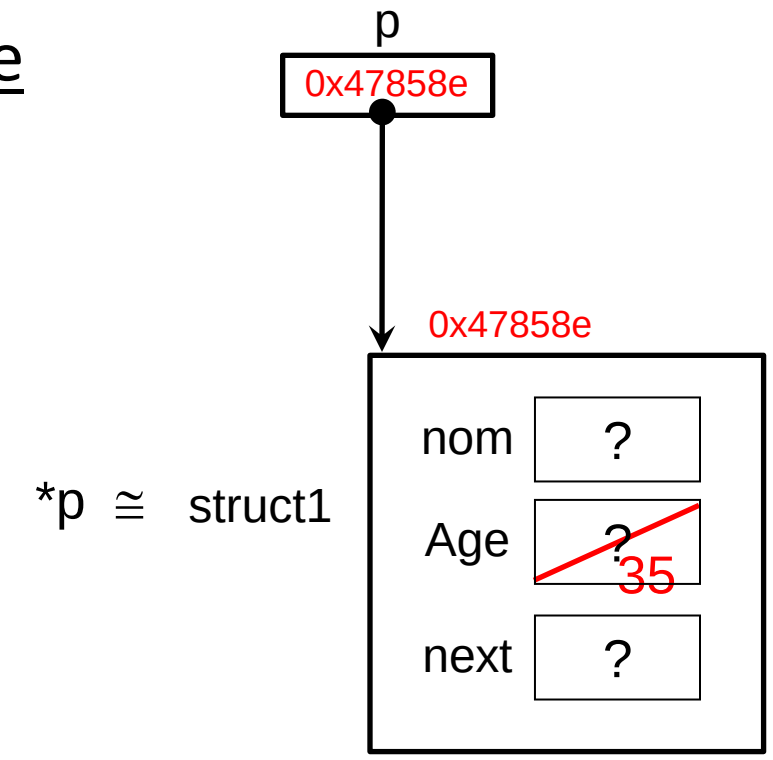
- Il est possible de faire un pointeur sur une structure dans une structure en indiquant le nom de la structure comme type du pointeur :




- Ce type de construction permet de créer des listes chaînées, dans lesquelles chaque structure contient l'adresse de la structure suivante dans la liste.

# Exemple

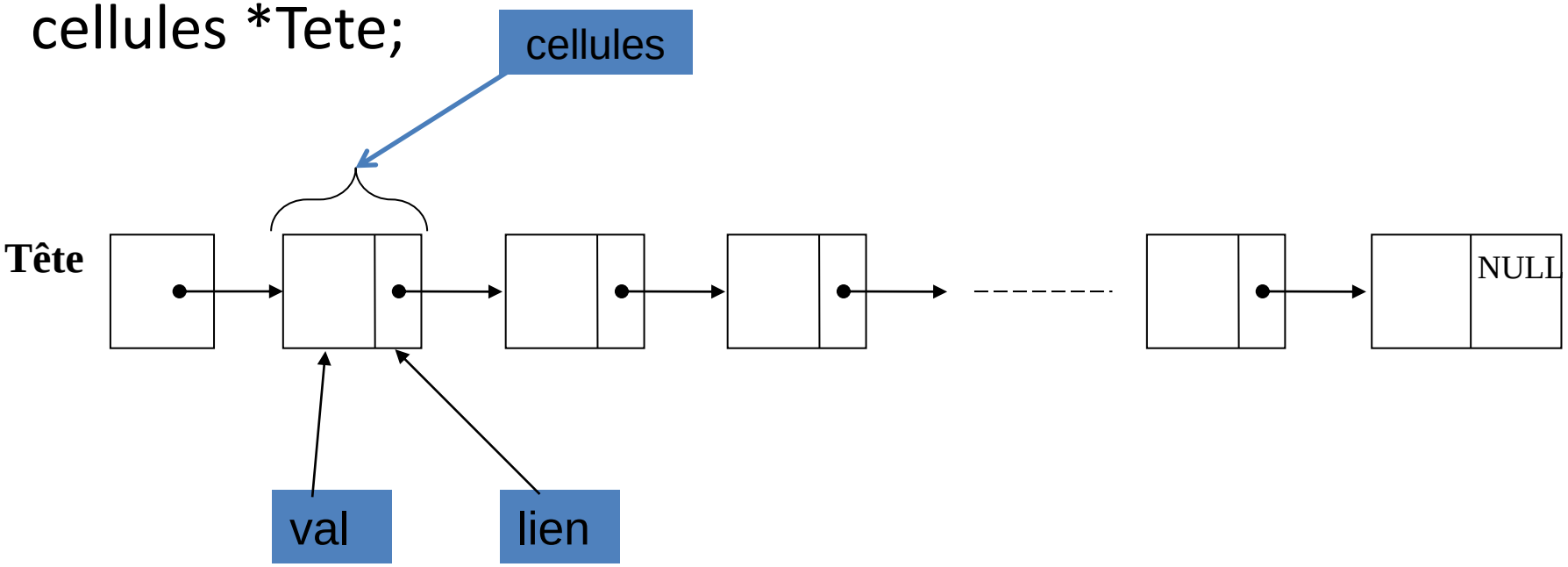
```
struct Client
   {    char nom[10];
        int Age;
        Client* next;
   };
Client struct1;
Client *p = &struct1;
p->Age = 35;
      /* ou bien   */
(*p).Age=35;
/* ou bien  */  struct1.Age=35;
```




*p ≅ struct1

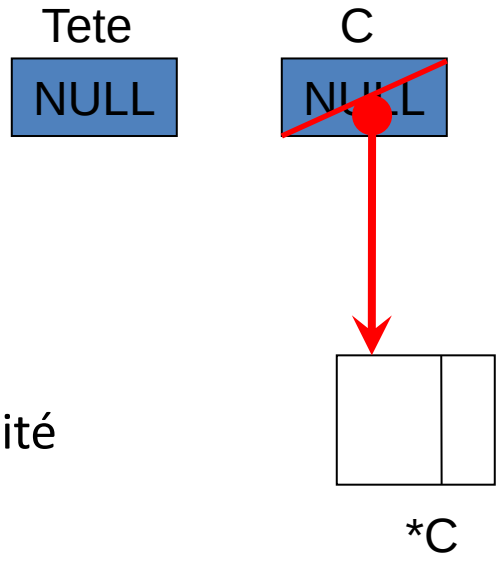
# Listes chaînées

```
struct cellules
    { Element val;
      struct cellules *lien;
    } ;
cellules *Tete;
```

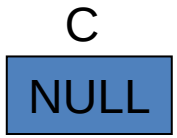


# Allocation dynamique

```
struct cellules
    { Element val;
      struct cellules *lien;
    } ;
cellules *Tete(0), *C(0);
```




```
 C = new cellules;        //  En C++

    cellules *C = new cellules ;                    // autre possibilité
          //  équivalent de New(C);  en Pascal

free (C) ;          //  équivalent de dispose(C);  en Pascal

delete C;           /* En  C++  */
```



# Fichiers

- La première chose à faire quand on veut manipuler des fichiers, c'est de les ouvrir. En C++, c'est la même chose. Une fois le fichier ouvert, tout se passe comme pour *cout* et *cin*.

- On parle de **flux** pour désigner les moyens de communication d'un programme avec l'extérieur. Dans ce chapitre, nous allons donc parler des **flux vers les fichiers**.

# Ouvrir un fichier en écriture

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{           string const nomFichier("C:\Users\dell\Desktop\Exples_TP\scores.txt");
            //ou bien           string const nomFichier("scores.txt");
            ofstream monFlux(nomFichier.c_str());
   if(monFlux)
   {
      monFlux << "Bonjour, je suis une phrase écrite dans un fichier." << endl;
      monFlux << 42.1337 << endl;
      int age(23);
      monFlux << "J'ai " << age << " ans." << endl;
   }
   else
   {
      cout << "ERREUR: Impossible d'ouvrir le fichier." << endl;
   }
monFlux.close();  //On ferme le fichier
system("pause");
}
```

# Ouvrir un fichier en lecture

```
ifstream fichier(nomFichier.c_str());

   if(fichier)
   {
      //L'ouverture s'est bien passée, on peut donc lire
      string ligne; //Une variable pour stocker les lignes lues

      while(getline(fichier, ligne)) //Tant qu'on n'est pas à la fin, on lit
      {
         cout << ligne << endl;
         //Et on l'affiche dans la console
         //Ou alors on fait quelque chose avec cette ligne
         //A vous de voir
      }
   }
   else
   {
      cout << "ERREUR: Impossible d'ouvrir le fichier en lecture." << endl;
   }
monFlux.close();  //On ferme le fichier
system("pause");
```