# Structures de données (LU2IN006)

Nawal Benabbou

Licence Informatique - Sorbonne Université

2024-2025

# Présentation et organisation de l'UE

## Contenu de l'UE

Présenter de manière synthétique les structures de données couramment utilisées en informatique afin de connaître les clefs permettant de choisir optimalement les structures les plus adaptées pour résoudre un problème donné.

**Quelques références bibliographiques :**


- C. Cormen, C. Leiserson, R. Rivest. "Introduction à l'algorithmique", Dunod, 1994.
- A. Aho, J. Ullman. "Concepts fondamentaux de l'informatique", Dunod, 1993.
- M. Divay. "Algorithmes et structures de données génériques", Dunod, 1999.
- R. Malgouyres, R. Zrour, F. Feschet. "Initiation à l'algorithmique et aux structures de données en C", 2008.


## Déroulement de l'UE

**Cours :** 11 séances de 1h45 (mercredi 16h-17h45)

**TD/TME :**

- 11 séances, en décalage d'une semaine avec le cours
- Chaque séance : TD de 1h45 et TME de 1h45 (en langage C)

**Documents et informations :** sur Moodle.

# Planning

| | 08h30 | 09h00 | 09h30 | 10h00 | 10h30 | 11h00 | 11h30 | 12h00 | 12h30 | 13h00 | 13h30 | 14h00 | 14h30 | 15h00 | 15h30 | 16h00 | 16h30 | 17h00 | 17h30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **lundi** | | | | | | LU2IN006 TD 6<br>L2 : mono6 | | | | | | LU2IN006 TME 6<br>L2 : mono6 | | | | | | | |
| | | | | | | LU2IN006 TD 7<br>L2 : maj1 | | | | | | | | | | | | | |
| | | | LU2IN006 TD 11<br>jour férié décalé<br>L2 : dant2 | | | LU2IN006 TME 11<br>jour férié décalé<br>L2 : dant2 | | | | | | | | | | | | | |
| | | | LU2IN006 TD 4<br>jour férié décalé<br>L2 : DM IP, mono4 | | | LU2IN006 TME 4<br>jour férié décalé<br>L2 : DM IP, mono4 | | | | | | | | | | | | | |
| **mardi** | | | LU2IN006 TD 2<br>L2 : DC idroit, mono2 | | | LU2IN006 TME 2<br>L2 : DC idroit, mono2 | | | | | | LU2IN006 TD 3<br>L2 : mono3 | | | | LU2IN006 TME 3<br>L2 : mono3 | | | |
| | | | LU2IN006 TD 5<br>L2 : mono5 | | | LU2IN006 TME 5<br>L2 : mono5 | | | | | | | | | | | | | |
| | | | LU2IN006 TME 7<br>L2 : maj1 | | | | | | | | | | | | | | | | |
| **mercredi** | | | | | | LU2IN006 TD 1<br>L2 : mono1 | | | | | | LU2IN006 TME 1<br>L2 : mono1 | | | | **LU2IN006 Cours 1**<br>L2 : dant1, dant2, DC idroit-li, DM IEEA-IM-IP, Maj, Mono | | | |
| | | | | | | LU2IN006 TD 8<br>L2 : DC li, maj2 | | | | | | LU2IN006 TME 8<br>L2 : DC li, maj2 | | | | | | | |
| | | | | | | LU2IN006 TD 9<br>L2 : DM IM | | | | | | LU2IN006 TME 9<br>L2 : DM IM | | | | | | | |
| **jeudi** | | | LU2IN006 TD 11<br>L2 : dant2 | | | LU2IN006 TME 11<br>L2 : dant2 | | | | | | | | | | | | | |
| | | | LU2IN006 TD 4<br>L2 : DM IP, mono4 | | | LU2IN006 TME 4<br>L2 : DM IP, mono4 | | | | | | | | | | | | | |
| **vendredi** | | | | | | | | | | | | LU2IN006 TD 10<br>L2 : dant1, DM IEEA | | | | LU2IN006 TME 10<br>L2 : dant1, DM IEEA | | | |

# Équipe pédagogique

**Responsable de l'UE :** Nawal Benabbou
**Chargés de cours :** Nawal Benabbou (séances 1,2,7,8,9,10) et Pierre-Henri Wuillemin (séances 3,4,5,6,11)

| Groupe | Chargé de TD | Doublure(s) de TME |
|---|---|---|
| 1 | Arun Nadaradjane | Junwoo Park |
| 2 | Ana-Alexandra Brad | Ghassen Marrakchi |
| 3 | Tristan Bersoux | Mathilde Abrassart |
| 4 | Anissa Kheireddine | Junwoo Park |
| 5 | Jean-Baptiste Deloges | Sébastien Lallé |
| 6 | Sabrine Saouli | Aurélie Beynier |
| 7 | Mohamed Ouaguenouni | Guillaume Moinard |
| 8 | Amaury Curiel | Maxime Toquebiau |
| 9 | Manuel Amoussou | Alper Er |
| 10 | Ramy Iskander | Alper Er |
| 11 | Valentin Barbaza | Ghassen Marrakchi |

**Contacts :** nawal.benabbou@lip6.fr, pierre-henri.wuillemin@lip6.fr, amaury.curiel@lip6.fr, anissa.kheireddine@lip6.fr, mohamed.ouaguenouni@lip6.fr, tristan.bersoux@lip6.fr, manuel.amoussou@lip6.fr, valentin.barbaza@lip6.fr, arun.nadaradjane@lip6.fr, jean-baptiste.deloges@sorbonne-universite.fr, ana-alexandra.brad@sorbonne-universite.fr, sabrine.saouli@lip6.fr, alper.er@sorbonne-universite.fr, guillaume.moinard@lip6.fr, ghassen.marrakchi@lipn.univ-paris13.fr, junwoo.park@sorbonne-universite.fr, sebastien.lalle@lip6.fr, maxime.toquebiau@ece.fr, ramy.iskander@sorbonne-universite.fr, abrassart@ircam.fr, aurelie.beynier@lip6.fr

# Évaluation

## Contrôle des connaissances

- Examen final : 50%
- Un projet : 30%
- Contrôle continu : 20% (1 mini-projet à 10% + 1 interrogation à 10%)

## Sur le projet

Le projet est à réaliser pendant les séances 5 à 10 de TME, avec une soutenance prévue en séance 11.

**Attention :**

- La note du projet ne fait pas partie du contrôle continu, et est conservée en deuxième session.
- Le projet doit obligatoirement être fait **en binôme** (aucun monôme ne sera accepté).

## Sur l'examen et le contrôle continu

- Contrôle continu : 1 interrogation écrite réalisée pendant une séance de TD, et un mini-projet réalisé pendant les premières séances de TME.
- Examen final : aucun document autorisé, sauf une feuille A4 manuscrite.

# L'abstraction

Une structure de données est une mise en oeuvre concrète d'un type abstrait de données. Qu'est-ce que l'abstraction ?

## Définition générale (Larousse)

Opération intellectuelle qui consiste à isoler par la pensée l'un des caractères de quelque chose et à le considérer indépendamment de ses autres caractères.

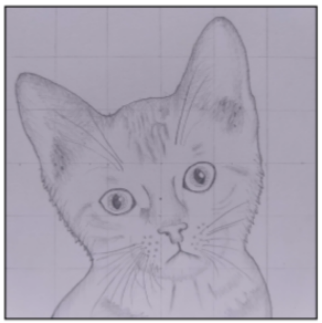
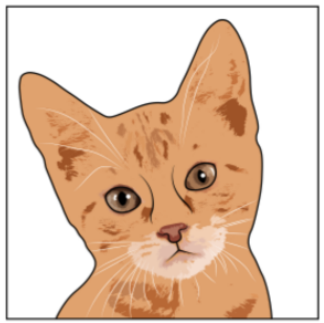
## Quelques types d'abstraction :

- **L'abstraction par simplification** permet de réduire la description d'un objet en ne conservant que les informations utiles ou importantes.

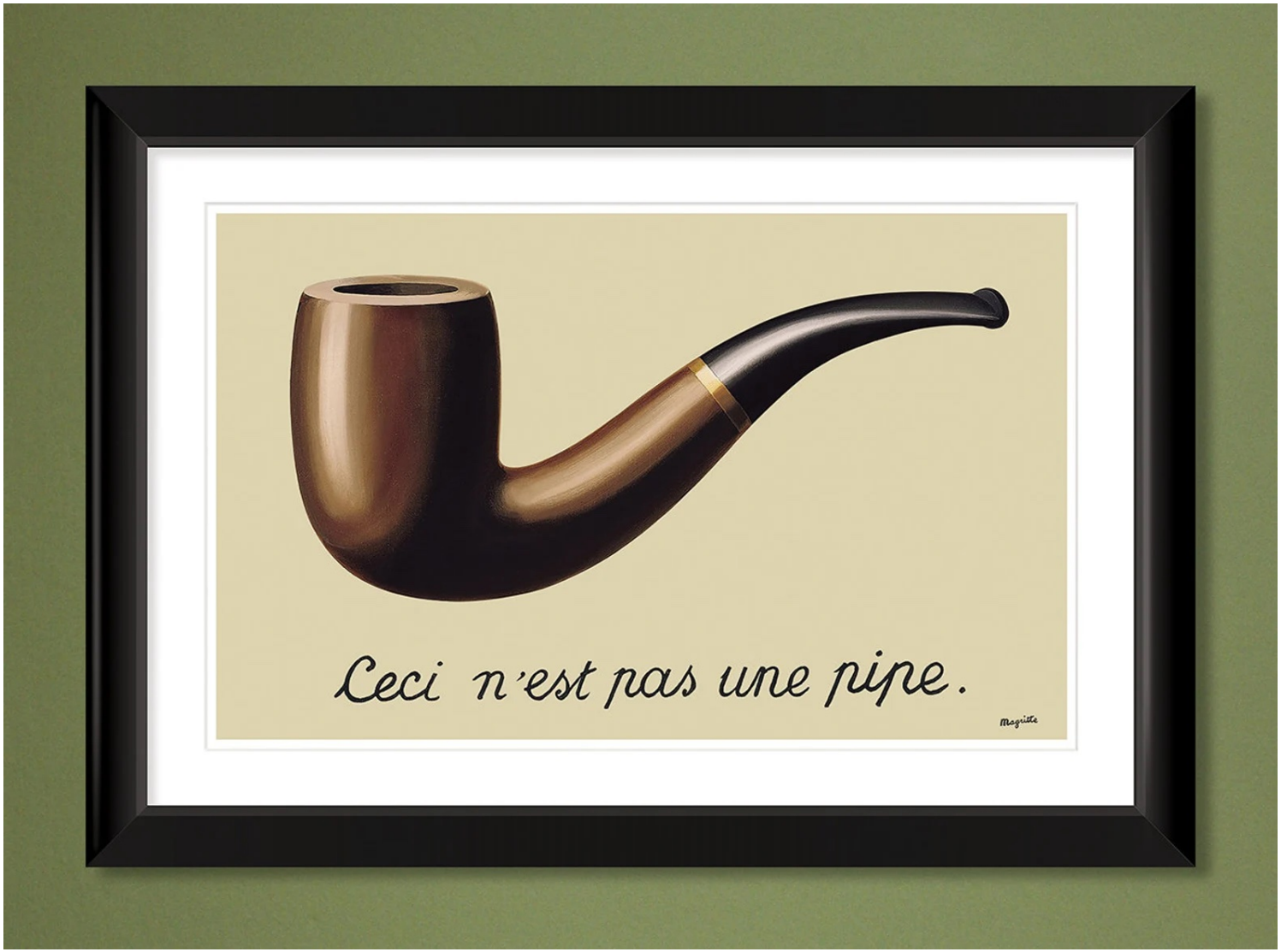
  *Exemples : un plan de métro, une fiche personnage.*

- **L'abstraction par généralisation** permet d'aller du particulier vers le général, d'un objet à une classe d'objets plus générale.

  *Exemples : les hommes sont des vertébrés, un bureau est un meuble*.

# L'abstraction




La trahison des images. René Magritte (1929)

# L'abstraction en informatique

## Abstraction informatique

L'abstraction en informatique consiste à synthétiser des caractéristiques et traitements communs, applicables à des entités ou concepts variés, afin de simplifier et d'unifier leur manipulation.

## Exemples

- Un ordinateur peut se définir comme un ensemble de composants électroniques, et ce sont les couches progressives d'abstraction (noyau, système d'exploitation...) qui permettent à l'utilisateur de ne pas avoir à gérer directement les aspects électroniques lorsqu'il manipule l'ordinateur.
- Les langages informatiques sont des abstractions, faisant correspondre le langage machine à un langage plus compréhensible par l'homme.
- Les variables sont des abstractions, permettant de représenter par un symbole la valeur qu'elle contient, qui peut être une structure complexe.
- Les procédures sont des abstractions, permettant de résumer en une instruction des choses qui peuvent être complexes à être réaliser.

**Attention :** l'abstraction est une bonne chose (codes généraux, faciles à lire), mais cela ne doit pas rendre le programme plus lent !

# Type abstrait de données

## Définition

Un *type abstrait de données* est une spécification mathématique d'un ensemble d'objets et d'un ensemble d'opérations applicables à ces objets.

## Exemples :

- les nombres entiers, munis des opérations $+, *$ etc.
- les chaînes de caractères, munis des opérations de concaténation, d'insertion, etc.
- les listes, munis des opérations d'insertion, d'accès à des éléments, etc.
- les ensembles, munis des opérations d'union, d'intersection, etc.

## Remarque :

Un type abstrait de données ne définit pas la façon dont les données sont stockées, ni la manière d'implémenter les méthodes. Définir un type abstrait permet de manipuler des objets sans connaître leur représentation dans la mémoire de l'ordinateur (codage, implémentation), et donc d'élaborer des algorithmes de manière abstraite, conduisant à des programmes compréhensibles et réutilisables.

# Structure de données

## Définition

Une structure de données est une représentation concrète dans la mémoire d'un ordinateur des objets décrits par un type abstrait. Elle est accompagnée de l'implémentation des opérations sur cette représentation.

**Exemple :** un booléen est représenté par 1 octet en C++ contre 1 bit en java...

## Coût associé au choix d'une structure de données

Le choix de la structure de données a plusieurs conséquences sur les performances des programmes :

- La déclaration d'un objet a un coût en mémoire (en octets). On parlera de complexité spatiale.
- Les opérations sur les objets ont un coût en temps de calcul. On parlera de complexité temporelle (mesurée en nombre d'opérations élémentaires).

$\Rightarrow$ Toutes les structures de données n'ont pas les mêmes performances, ni les mêmes cas d'utilisation.

Cette UE a pour but d'apporter une connaissance des principales structures de données, afin d'apprendre dans quel cas il convient d'utiliser une structure de données plutôt qu'une autre, et de savoir comment les implémenter.

# Qu'est-ce qu'un programme ?

## Un programme est composé de :

- un ensemble fini de variables, dont des variables d'entrée
- un ensemble fini d'opérations élémentaires
- un ensemble fini de structures de contrôle

## Variable

Une variable est une "boîte" ayant un nom et contenant une valeur appartenant à un domaine. Une variable peut correspondre à un type simple (entier, flottant), ou à une structure plus élaborée.

## Opération élémentaire

Une opération élémentaire est une commande informatique d'un langage portant sur les variables et réalisant une opération simple (arithmétique, logique, affectation, etc.) qu'un ordinateur réalise en un nombre connu d'appels au processeur (CPU). **Exemples :** `a = a + b ;` `i > j ;` `b = 2 ;`

## Structure de contrôle

Une structure de contrôle est une instruction pouvant dévier le flot de contrôle du programme lors de son exécution. En programmation impérative, elle est construite à partir de structures élémentaires (tests, boucles...).

# Qu'est-ce qu'un algorithme ?

Un algorithme est un programme associé à un problème et vérifiant certaines propriétés (terminaison, validité).

## Problème

Un problème se définit à partir de paramètres, qui prennent des valeurs dans un domaine, et d'une question.

**Exemples :**

- À partir d'un tableau, construire un tableau trié.
- Trouver le plus court chemin dans une carte routière.
- Découper du tissu pour faire des vêtements en minimisant les chutes.
- Identifier les contours d'un visage sur une photo.

## Définition : algorithme

Un algorithme est un programme résolvant un problème donné

- en un nombre fini d'opérations élémentaires (terminaison).
- en donnant une réponse juste à la question du problème, quelque soit les valeurs des variables d'entrée (validité).

# Performances d'un algorithme

## Complexité temporelle

La complexité temporelle d'un algorithme décrit le temps nécessaire à son exécution. Elle est estimée en comptant le nombre d'opérations élémentaires effectuées lors d'une exécution de l'algorithme.

Pour la plupart des algorithmes, le nombre d'opérations élémentaires exécutées dépend de la valeur des paramètres d'entrée.

## "Pire des cas" et "Meilleur des cas"

Le cas où un algorithme effectue le nombre maximum d'opérations élémentaires possibles est communément appelé le "pire des cas". Le "meilleur des cas" correspond quant à lui au cas où il effectue le moins d'opérations élémentaires possibles.

# Illustration sur un exemple

Prenons l'exemple de la recherche d'une valeur dans un tableau.

```
1  int recherche_tab(int val, int* tab, int n){
2      int i=0;
3      while ((i<n) && (tab[i]!=val)){
4          i=i+1; //ou encore i++;
5      }
6      if (i<n){
7          return 1;
8      }else{
9          return 0;
10     }
11 }
```

## Quel est le meilleur des cas ?

Le meilleur des cas correspond au cas où l'élément recherché est le premier élément du tableau. On réalise alors une initialisation (ligne 2), deux tests (lignes 3 et 6) et le retour de la fonction (ligne 7) $\Rightarrow$ 4 opérations élémentaires.

## Quel est le pire des cas ?

Le pire des cas correspond à la recherche d'un entier non présent dans le tableau. Dans ce cas, on réalise une initialisation (ligne 2), $n+2$ tests (lignes 3 et 6), $n$ additions (ligne 4) et le retour de la fonction (ligne 9) $\Rightarrow$ $2n+4$ opérations élémentaires.

# Complexité temporelle et notations de Landau

Ce n'est pas toujours possible de donner précisément la complexité d'un algorithme (car le nombre d'opérations effectuées peut dépendre de beaucoup de paramètres). Dans ce cas, on cherche à fournir des bornes sur la complexité temporelle, et on utilise les notations de Landau.

## Définition : notation de Landau $O$ (borne supérieure)

Soit $f(n)$ et $g(n)$ deux fonctions positives définies sur $\mathbb{N}$. On dit que $f(n)$ est en $O(g(n))$ si la fonction $f$ est asymptotiquement bornée supérieurement par la fonction $g$ à un facteur près. Plus formellement, $f(n)$ est en $O(g(n))$ s'il existe un entier $n_0$ et un réel $k > 0$ tels que pour tout $n \geq n_0$ :

$$f(n) \leq kg(n)$$

## Retour à l'exemple de recherche dans un tableau

Pour cet algorithme, notons $f(n)$ le nombre d'opérations effectuées pour un tableau de taille $n$. Nous avons vu que, dans le pire des cas, l'algorithme effectue (environ) $2n+4$ opérations élémentaires. Donc, en posant $g(n) = n$, et par exemple $k = 6$ et $n_0 = 1$, on a bien $f(n) \leq k \times g(n)$ pour tout $n \geq n_0$. On en conclut que l'algorithme a une complexité temporelle en $O(n)$.

# Complexité temporelle et notations de Landau

## Définition : notation de Landau $\Omega$ (borne inférieure)

Soit $f(n)$ et $g(n)$ deux fonctions positives définies sur $\mathbb{N}$. On dit que $f(n)$ est en $\Omega(g(n))$ si la fonction $f$ est asymptotiquement bornée inférieurement par la fonction $g$ à un facteur près. Plus formellement, $f(n)$ est en $\Omega(g(n))$ s'il existe un entier $n_0$ et un réel $k > 0$ tels que pour tout $n \geq n_0$ :

$$kg(n) \leq f(n)$$

## Retour à l'exemple de recherche dans un tableau

Notons $f(n)$ le nombre d'opérations élémentaires effectuées pour un tableau de taille $n$. Nous avons vu que, dans le meilleur des cas, l'algorithme effectue (environ) 4 opérations élémentaires. Donc, en posant $g(n) = 1$, $k = 1$ et $n_0 = 0$, on a $k \times g(n) \leq f(n)$ pour tout $n \geq n_0$, ce qui signifie que la complexité temporelle de cet algorithme est en $\Omega(1)$.

# Complexité temporelle et notations de Landau

## Définition : notation de Landau $\Theta$ (borne inférieure et supérieure)

Soit $f(n)$ et $g(n)$ deux fonctions positives définies sur $\mathbb{N}$. On dit que $f(n)$ est en $\Theta(g(n))$ si la fonction $f$ est dominée et soumise à $g$ asymptotiquement. Plus formellement, $f(n)$ est en $\Theta(g(n))$ s'il existe un entier $n_0$ et deux réels $k_1, k_2 > 0$ tels que pour tout $n \geq n_0$ :

$$k_1 g(n) \leq f(n) \leq k_2 g(n)$$

## Remarque :

Quand on évalue la complexité temporelle d'un algorithme avec des notations de Landau, on peut en pratique ignorer les :

- opérations élémentaires hors des boucles car elles sont en nombre fini,
- tests de boucle car ils ajoutent un nombre fini d'opérations par tour.

# Exemple

```
1  double mystere(int n){
2      int i,j;
3      double f,s;
4      s=0;
5      for (i=1;i<=n;i++){
6          f=1;
7          for (j=1;j<=i;j++){
8                  f=f*j;
9          }
10         s=s+f;
11     }
12     return s;
13 }
```

## Que fait cet algorithme ?

Il permet de calculer la somme des factorielles, c'est-à-dire : $\sum_{i=1}^{n} i!$

## Quelle est sa complexité temporelle ?

La complexité de cet algorithme est $\Theta(n^2)$ car, dans tous les cas, le nombre d'opérations élémentaires réalisées est de l'ordre de :

$$\sum_{i=1}^{n}\left(2+\sum_{j=1}^{i}1\right)=\sum_{i=1}^{n}\left(2+i\right)=2n+\frac{n\times(n+1)}{2}=\frac{1}{2}n^2+\frac{5}{2}n \Rightarrow \Theta(n^2)$$

# Une autre exemple

```
1  double mystere2(int n){
2      int j;
3      double f,s;
4      s=0;f=1;
5
6      for (i=1;i<=n;i++){
7          f=f*i;
8          s=s+f;
9      }
10     return s;
11 }
```

**Que fait cet algorithme ?**

Cet algorithme permet lui aussi de calculer la somme des factorielles.

**Quelle est sa complexité temporelle ?**

Sa complexité est en $\Theta(n)$ car son nombre d'opérations élémentaires est dans tous les cas de l'ordre de $2n$ (2 opérations élémentaires par tour de boucle).

$\Rightarrow$ Comme ces deux algorithmes permettent de résoudre le même problème, on préférera le deuxième au premier car il a une meilleure complexité temporelle.

# Performances d'un algorithme

Pourquoi est-ce que la complexité temporelle est importante ?

## Un petit exemple

Considérons un ordinateur capable de réaliser $10^9$ opérations élémentaires par seconde (processeur avec une fréquence de $1$ GigaHertz). Notons $f(n)$ le nombre d'opérations élémentaires réalisées par un programme, où $n$ est la taille de l'instance (par exemple, la taille d'un tableau). Quand $n = 1000$, combien de temps faut-il pour exécuter un programme de complexité :

- $f(n) = \Theta(n^2)$ ? $1 \times 10^{-3}$ seconde
- $f(n) = \Theta(n^3)$ ? $1$ seconde
- $f(n) = \Theta(n^4)$ ? $16.7$ minutes
- $f(n) = \Theta(n^5)$ ? $11.6$ jours
- $f(n) = \Theta(2^n)$ ? $3.39 \times 10^{282}$ siècles

Quand on effectue des opérations sur des données, le choix des structures de données a un impact sur le nombre d'opérations élémentaires effectuées. Il est primordial de savoir comment choisir les structures de données selon l'application considérée, pour réduire les temps de calcul des programmes.

# Une autre mesure de performance : la complexité spatiale

On peut aussi vouloir évaluer l'efficacité d'un algorithme selon son occupation mémoire. En effet, il faut pouvoir déterminer une implémentation permettant de ne pas dépasser l'espace mémoire d'un ordinateur.

**Définition : complexité spatiale**

La complexité spatiale d'un algorithme est une estimation de l'espace mémoire occupé lors de son exécution.

$\Rightarrow$ La complexité spatiale peut être indiquée avec les notations de Landau par rapport à la valeur des variables d'entrée (comme la complexité temporelle).

**Remarque**

Les complexités temporelles et spatiales sont fréquemment en opposition, et dans ce cas, il faut déterminer un compromis entre les deux.

# Exemple

```java
double mystere3(int n){
    int f=1;
    for (i=2;i<=n;i++){
        f=f*i;
    }
    return f;
}
```

```java
double mystere4(int n){
    if (n <= 1){
        return 1;
    }else{
        return n*mystere4(n-1);
    }
}
```

## Que font ces algorithmes ?

Ces deux algorithmes permettent de calculer la factorielle d'un nombre ($n!$). Le premier algorithme (`mystere3`) donne une version itérative, alors que le deuxième (`mystere4`) donne une version récursive.

## Quelles sont leurs complexités temporelles et spatiales ?

Ces deux algorithmes ont la même complexité temporelle : $\Theta(n)$. Cependant, leur complexité spatiale est différente :

- La version itérative utilise un nombre constant de cases mémoire (au plus 4) $\Rightarrow$ Complexité spatiale : $\Theta(1)$
- La version récursive a besoin de 2 cases mémoire par appel, donc $2n$ au total $\Rightarrow$ Complexité spatiale : $\Theta(n)$.

# Compilation et programmation modulaire

## Programmation modulaire

Pour un programme, la modularité est le fait d'être composé en plusieurs parties, relativement indépendantes les unes des autres. La programmation modulaire offre la possibilité de découper un gros programme en modules (réunissant des fonctionnalités particulières) répartis dans différents fichiers.

## Avantages

La programmation modulaire :

- rend le code plus compréhensible,
- permet la réutilisation de code,
- supprime les risques d'erreurs en reprogrammant la même chose,
- permet de développer et d'améliorer des parties de code indépendamment,
- facilite le travail collaboratif,
- simplifie la compilation en se concentrant sur les fichiers modifiés.

$\Rightarrow$ Nécessite une réflexion sur le découpage du code avant de se lancer.

# Module en langage C et exemple

## Module

Un module est composé de :

- un fichier d'en-tête (extension ".h"), décrivant l'interface du module . Il contient des déclarations de fonctions, de type (`struct`), de variables et éventuellement des inclusions de bibliothèques.
- un fichier source (extension ".c") contenant le code des fonctions déclarées dans le fichier d'en-tête.

```c
/* Fichier tableau.h */
#ifndef TABLEAU_H
#define TABLEAU_H

typedef struct tableau {
    char* tab;
    int taille;
} Tableau;

int recherche(int x, Tableau* t);

#endif
```

```c
/* Fichier tableau.c */
#include "tableau.h"

int recherche(int x, Tableau* t){
    int i=0;
    while (i<t->taille){
        if (t->tab[i]==x){
            return i;
        }else{
            i=i+1;
        }
    }
    return -1;
}
```

# Exemple

```c
/* Fichier lecture.h */
#ifndef LECTURE_H
#define LECTURE_H
#include "tableau.h"
#include <stdio.h>

Tableau* lire (int n, FILE* f);

#endif
```

```c
/* Fichier lecture.c */
#include "lecture.h"
#include <stdlib.h>

Tableau* lire (int n, FILE* f){
    Tableau* p = (Tableau*)malloc(sizeof(Tableau));
    p->tab = (char*)malloc(sizeof(char)*n);
    p->taille=n;
    if (fgets(p->tab,n,f)!= NULL){
        return p;
    }else{
        printf("Probleme de lecture");
        return NULL;
    }
}
```

# Exemple

```c
/* Fichier main.c */
#include "tableau.h"
#include "lecture.h"
#include <stdio.h>

void main(){
    FILE* f = fopen("nomFic.txt", "r");
    if (f!=NULL){
        Tableau* p = lire(256, f);
        printf("%s", p->tab);
    }
}
```

## Compilation séparée

On peut maintenant compiler les fichiers sources séparément, puis créer l'exécutable à partir des fichiers compilés. Sur l'exemple, on peut faire :

- `gcc -c tableau.c` (crée le fichier "tableau.o")
- `gcc -c lecture.c` (crée le fichier "lecture.o")
- `gcc -c main.c` (crée le fichier "main.o")
- `gcc -o main main.o tableau.o lecture.o` (crée l'exécutable `main`)

# Makefile

## L'outil `make`

L'outil `make` est un programme permettant d'automatiser la génération de fichiers à partir de code source, en exécutant un certain nombre de commandes `shell`. Contrairement à un simple script `shell`, `make` exécute les commandes uniquement si elles sont nécessaires. Dans notre cas, cela permet de recompiler uniquement les fichiers qui ont été modifiés depuis la dernière compilation.

## Makefile

L'outil `make` utilise des fichiers, appelés `Makefile`, qui spécifient les règles à suivre pour créer les fichiers désirés. Les règles sont sous la forme :

```
1 nom_cible : dependances
2     commandes
```

où `nom_cible` est le nom du fichier à créer, `dependances` est l'ensemble des fichiers nécessaires à sa création, et `commandes` donne les commandes à exécuter (précédées d'une tabulation).

# Fonctionnement de l'outil `make`

Quand on tape la commande `make nom_cible` dans le terminal, l'outil `make` évalue la règle permettant de créer `nom_cible`. Si `nom_cible` n'est pas donné en argument de `make`, alors l'outil évalue la première règle rencontrée dans le `makefile`.

## Fonctionnement : une évaluation récursive

Évaluer une règle $R$ se fait récursivement en analysant ses dépendances de la manière suivante :

- Toute dépendance qui est la cible d'une autre règle doit être évaluée avant de pouvoir exécuter les commandes de la règle $R$.
- Une fois que toutes les dépendances de $R$ ont été évaluées, les commandes de la règle $R$ sont exécutées uniquement si la cible est plus ancienne que l'une de ses dépendances (au moins).

# Retour sur l'exemple

```
1  all : main //pas necessaire ici car un seul executable
2
3  main : main.o lecture.o tableau.o
4      gcc -o main main.o lecture.o tableau.o
5
6  main.o :  main.c lecture.h tableau.h
7      gcc -c main.c
8
9  lecture.o : lecture.c lecture.h tableau.h
10     gcc -c lecture.c
11
12 tableau.o : tableau.c tableau.h
13     gcc -c tableau.c
14
15 clean :
16     rm -f *.o main
```

# Pour résumer

## Objectif de l'UE

L'objectif de ce module est tout à la fois :

- Étudier les différents types abstraits et structures de données.
- Comprendre leur vitesse et leur occupation mémoire.
- Savoir identifier dans quelles situations une structure de données est préférable à une autre.
- Apprendre à les implémenter correctement (en langage C).

## Remarque

Cette étude est réalisée avec le langage C, mais elle pourrait être faite dans d'autres langages, et elle vous sera utile quel que soit le langage !

$\Rightarrow$ Avant la première séance de TD/TME, lisez les rappels de C mis en ligne sur moodle !