



# Essentiel d'informatique

2023/2024

Victor Sarrazin



*Bienvenue dans l'essentiel d'informatique de mes cours de prépa. Ce document a pour objectif de contenir l'intégralité des cours d'informatique afin de les condenser et de les adapter.*

*Bonne lecture...*

# Sommaire

## Introduction aux langages :

 I Introduction au C	3
 II Introduction au OCaml	5

## Structures de données :

 I Structures de données	6
 II Piles, files, dictionnaires	6
 III Arbres	6
 IV Graphes	6

## Informatique théorique :

 I Bases	7
 II Récursion	7
 III Stratégies algorithmiques	7
 IV SQL	7

# Introduction

## I Introduction au C

### I.1 Variables

Pour définir une variable en C on a la syntaxe suivante : type nom

```
1 int mango = 0;
```



Il est possible de définir plusieurs variables en même temps :

```
1 int banana = apple = 12;
```



### I.2 Opérateurs

On a les opérations arithmétiques suivantes :

Opération	En C
Addition	a + b
Soustraction	a - b
Multiplication	a * b
Division	a / b
Modulo	a % b

On peut utiliser +=, -=, \*=, /= et %= pour faire des opérations arithmétiques et des assignations

De plus on peut utiliser ++ et -- pour incrémenter/décrémenter

Les comparaisons se font avec >, >=, <=, < et ==.

On a des opérateurs binaires && (et logique), || (ou logique) et ! (négation de l'expression suivante)



#### Attention :

Le && est prioritaire sur le ||

### I.3 Structures de contrôle

Pour exécuter de manière conditionnelle, on utilise if (cond) {...} else if (...) {} ... {} else {}

Ainsi le code suivant est valide :

```

1  if (x == 1) {
2      // Do code
3  } else if (x > 12) {
4      // Do code bis
5  } else {
6      // Do code ter
7  }

```



#### Attention :

En C un 0 est considéré comme false et toute autre valeur numérique true

Pour faire une boucle on peut utiliser un while (cond) {} qui exécute le code tant que la condition est valide

On peut utiliser do {} while (cond) qui exécute une fois puis tant que la condition est vérifiée

Il est aussi possible d'utiliser for (...) {}, de la manière suivante :

```

1  // De 0 à n - 1
2  for (int i = 0; i < n; i++) {
3
4  }
5
6  // De 0 à n - 1 tant que cond
7  for (int i = 0; i < n && cond; i++) {
8
9  }

```

A noter qu'en C il est possible de modifier la valeur de i et donc de sortir plus tôt de la boucle

Il est possible de sortir d'une boucle avec break, ou de passer à l'itération suivante avec continue



## I.4 Fonctions

Pour définir une fonction on écrit :

```

1  int my_func(int a, int b) {
2      // Do code
3      return 1;
4  }

```

Si on ne prend pas d'arguments on écrit int my\_func(void) {} et si on ne veut rien renvoyer on utilise void my\_func(...) {}

Ainsi pour appeler une fonction on fait :

```

1  int resp = my_func(12, 14);

```

On peut déclarer une fonction avant de donner son code mais juste sa signature avec :

```

1  int my_func(int);

```

## I.5 Tableaux en C

Le type d'un tableau en C est `type[]` ou `* type`

Pour initialiser un tableau on a les manières suivantes :

```
1  int[4] test = {0, 1, 2, 3}; // Initialise un tableau de taille 4 avec 0,1,2,3
2  int[] test = {0, 1, 2, 3}; // Initialise un tableau avec 0,1,2,3 (avec 4
3  éléments)
   int[4] test = {0}; // Initialise un tableau de taille 4 avec 0,0,0,0
```

Il n'est pas obligé de donner la taille d'un tableau elle sera déterminée au moment de l'exécution



### Attention :

Si on dépasse du tableau C ne prévient pas mais s'autorise à faire n'importe quoi

Pour affecter dans une case de tableau on fait :

```
1  test[1] = test[2] // On met dans la case 1 la valeur de la case 2
```

## II Introduction au OCaml

# Structures de données

 I Structures de données

 II Piles, files, dictionnaires

 III Arbres

 IV Graphes



# Informatique théorique



## I Bases



### I.1 Fonctions

On dit qu'une fonction a des **effets de bord** si son exécution a des conséquences sur d'autres choses que ses variables locales

Une fonction est **déterministe** si le résultat est toujours le même avec les mêmes arguments

Une fonction est dite **pure** lorsqu'elle est déterministe et sans effets de bord



### I.2 Algorithmes de tri



## II Récursion



## III Stratégies algorithmiques



## IV SQL



### IV.1 Généralités

En SQL on stocke des entités avec des attributs et à chaque attribut on lui associe un type

On peut définir des relations entre les différentes entités

On stocke ces entités dans des tables : dans chaque table on stocke une entité

Il est possible de garder une case vide en plaçant un NULL dans la case



### IV.2 Requêtes

Pour récupérer des données (projections) dans une table on a :

```
1  # Seulement les colonnes spécifiées
2  SELECT coll, ..., coln FROM table
3
4  # Toutes les colonnes
5  SELECT * FROM table
6
7  # Toutes les colonnes mais sans doublon
8  SELECT DISTINCT * FROM table
```



Ainsi on récupère toutes les lignes de la table avec ces projections

On peut aussi faire une sélection sur un critère :

```
1 SELECT * FROM table WHERE bool
```



Les opérations booléennes sont les suivantes :

- `col > a/col < a/col = a` pour faire des comparaisons
- `col IN (a, b, c)` pour savoir si la cellule est dans un ensemble de valeur
- `col IS NULL/IS NOT NULL` pour savoir si la cellule est nulle ou non
- `col LIKE '% Text %'` pour regarder si Text est dans la chaîne de caractère de la cellule

On peut combiner les critères avec AND/OR/NOT

Il est possible de sélectionner un attribut non projeté

Pour ordonner les résultats on ordonne en utilisant

```
1 # Triés par valeur croissante
2 SELECT * FROM table ORDER BY col
3
4 # Triés par valeur décroissante
5 SELECT * FROM table ORDER BY col DESC
```



Pour limiter le nombre de valeurs on utilise

```
1 # On prend au maximum 3 éléments
2 SELECT * FROM table LIMIT 3
3
4 # On prend au maximum 3 éléments mais sans les 2 premiers
5 SELECT * FROM table LIMIT 3 OFFSET 2
```



### IV.3 Fonctions

On peut compter le nombre d'entités qui vont être renvoyées

```
1 # Nombre d'éléments dans la table
2 SELECT COUNT(*) FROM table
```



On peut compter sur une colonne spécifique avec `COUNT(col1, ..., col2)`, les cases ne sont pas comptées si NULL,

Il est aussi possible de compter le nombre de valeur distinctes pour une colonne :

```
1 SELECT COUNT(DISTINCT col) FROM table
```



On peut utiliser MAX, MIN, SUM et AVG pour avoir du préprocessing, il est aussi possible d'avoir la moyenne en faisant `SUM(col)/COUNT(*)`



#### Attention :

On ne peut mélanger une colonne et une fonction dans la projection

Il est possible de grouper les valeurs



```
1 # Renvoie des groupes des valeurs de col
2 SELECT col FROM table GROUP BY col
```



#### Attention :

Il n'est pas possible d'utiliser GROUP BY sur des colonnes non groupées

Par contre les fonctions agissent sur chaque groupe, ainsi il est possible d'écrire

```
1 # Renvoie des groupes des valeurs de col avec le nombre d'occurrence de cette
  valeur dans la table
2 SELECT col, COUNT(*) FROM table GROUP BY col
```

Pour sélectionner des groupes on peut utiliser :

```
1 # Renvoie des groupes des valeurs de col si la valeur minimale du groupe dans la
  colonne col2 est supérieure à x avec la valeur minimale de col2 de ce groupe dans
  la table
2 SELECT col1, MIN(col2) FROM table GROUP BY col1 HAVING MIN(col2) > x
```

Les opérations sont exécutées dans cet ordre :

- WHERE
- GROUP BY
- HAVING
- ORDER BY
- LIMIT/OFFSET
- SELECT à la fin bien qu'on le mette en tête de la requête

Ainsi une clause valide est

```
1 SELECT * WHERE cond GROUP BY col HAVING cond2 ORDER BY col2 LIMIT 3 OFFSET 2
```



## IV.4 Sous requêtes

Il est possible d'écrire une sous requête :

```
1 # Ici on sélectionne seulement les éléments donc la valeur col est supérieure à
  la valeur moyenne de col
2 SELECT * FROM table WHERE col > (SELECT AVG(col) FROM table)
```

Il est donc aussi possible d'utiliser cette syntaxe avec des IN

```
1 # Ici on va sélectionner seulement les lignes dont la valeur de col correspond à
  la condition cond
2 SELECT * FROM table WHERE col IN (SELECT DISTINCT col FROM * WHERE cond)
```

Le col AS nameBis permet de renommer une colonne

Si on reçoit un tableau, on peut sélectionner dans les réponses

```

1 # Ainsi on renvoie la moyenne d'une colonne col2 telle que ses éléments vérifient la condition
2 SELECT AVG(colName) FROM (SELECT col1, col2 AS colName FROM table WHERE cond)

```

## IV.5 Combiner les tables

Il est possible de combiner des tables

```

1 # Sélectionne dans le produit cartésien des deux tables
2 SELECT * FROM table1, table2

```

Mais en faisant ça on va avoir plein de lignes qui n'ont pas de sens, ainsi si on veut garder seulement les lignes qui nous intéressent

```

1 # Sélectionne dans le produit cartésien des deux tables seulement les éléments donc la col1 de la table 1 est le même que celui de la col 2 de la table 2
2 SELECT * FROM table1, table2 WHERE table1.col1 = table2.col2

```

Mais pour éviter ça on peut aussi de manière équivalente écrire :

```

1 # On sélectionne les éléments de la table1 en ajoutant la table2 si la condition est vérifiée, le ON est donc un WHERE
2 SELECT * FROM table1 JOIN table2 ON table1.col1 = table2.col2

```

Le produit cartésien n'est donc qu'une manière de jointure

On peut aussi utiliser le LEFT JOIN qui permet de garder un élément de la première table même si il n'a pas d'équivalent dans la seconde table

```

1 # On sélectionne les éléments de la table1 en concaténant les éléments dont la condition est vérifiée, et rien si il n'y a pas d'équivalent
2 SELECT * FROM table1 LEFT JOIN table2 ON table1.col1 = table2.col2

```

On peut faire l'union de deux requêtes

```

1 # On a les éléments qui vérifient la cond1 ou cond2
2 SELECT * FROM table WHERE cond1 UNION SELECT * FROM table WHERE cond2

```



### Attention :

Pour utiliser l'union il faut juste que les types sont compatibles mais pas les noms de colonne

On peut aussi faire l'intersection de deux requêtes























```

1 # On a les éléments qui vérifient la cond1 et cond2
2 SELECT * FROM table WHERE cond1 INTERSECT SELECT * FROM table WHERE cond2

```

On peut faire des différences ensemblistes avec MINUS ou EXCEPT

# Table des matières

 I Introduction au C	3
 I.1 Variables	3
 I.2 Opérateurs	3
 I.3 Structures de contrôle	3
 I.4 Fonctions	4
 I.5 Tableaux en C	5
 II Introduction au OCaml	5
 I Structures de données	6
 II Piles, files, dictionnaires	6
 III Arbres	6
 IV Graphes	6
 I Bases	7
 I.1 Fonctions	7
 I.2 Algorithmes de tri	7
 II Récursion	7
 III Stratégies algorithmiques	7
 IV SQL	7
 IV.1 Généralités	7
 IV.2 Requêtes	7
 IV.3 Fonctions	8
 IV.4 Sous requêtes	9
 IV.5 Combiner les tables	10
Table des matières	11