



Fenofitia Nomenjanahary

# Kajy University: Informatique

## Algorithmes et structures de données

**Author:** Dimby Rabearivony

**Date:** 19 avril 2024

**Version:** 1.0



*"Ny fahalalana no valamparihiko."*

# Table des matières

<b>1</b>	<b>Structures de données fondamentales</b>	<b>2</b>
1.1	Variables et langage de programmation . . . . .	2
1.2	Pointeurs . . . . .	3
1.3	Types de données . . . . .	5
1.4	Exercices . . . . .	7
1.5	Exercices . . . . .	7
<b>2</b>	<b>Algorithmes de recherche et de tri</b>	<b>8</b>
2.1	Qu'est-ce qu'un algorithme ? . . . . .	8
2.2	Tri . . . . .	8
2.3	Algorithmes de recherche . . . . .	8
2.4	Exercices . . . . .	8
<b>3</b>	<b>Complexité algorithmique</b>	<b>9</b>
3.1	Récursion . . . . .	9
3.2	La notation $O$ . . . . .	9
3.3	Les notations $\Omega$ et $\theta$ . . . . .	9
3.4	Complexité de certains algorithmes . . . . .	9
3.5	Exercices . . . . .	9
<b>4</b>	<b>Structures de données avancées</b>	<b>10</b>
4.1	Listes chaînées . . . . .	10
4.2	Piles . . . . .	10
4.3	Files . . . . .	10
4.4	Arbres binaires . . . . .	10
4.5	Arbres . . . . .	10
4.6	Graphes . . . . .	10
4.7	Exercices . . . . .	10
<b>5</b>	<b>Applications d'algorithmes en IA</b>	<b>11</b>
5.1	Greedy algorithm . . . . .	11

## Introduction

“En fait, je dirai que la différence entre un mauvais programmeur et un bon réside dans le fait qu’il considère son code ou ses structures de données comme plus importants. Les mauvais programmeurs se soucient du code. Les bons programmeurs se soucient des structures de données et de leurs relations.”

- Linus Torvalds

# Chapitre Structures de données fondamentales

## 1.1 Variables et langage de programmation

### 1.1.1 Déclaration de variables

Les variables constituent l'un des concepts les plus fondamentaux en programmation. En langage C, une variable est un espace de stockage nommé qui peut contenir une valeur modifiable. Les variables sont utilisées pour stocker des données telles que des nombres, des caractères et des adresses mémoire.

```
1  int age; // Declaration d'une variable de type entier appelee "age"
   "
2  float prix; // Declaration d'une variable de type flottant appelee
   "prix"
3  char lettre; // Declaration d'une variable de type caractere
   appelee "lettre"
```

### 1.1.2 Initialisation des variables

Les variables peuvent être initialisées lors de leur déclaration en leur attribuant une valeur initiale. Par exemple :

```
1  int nombre = 10; // Declaration et initialisation d'une variable
   de type entier avec la valeur 10
2  float pi = 3.14; // Declaration et initialisation d'une variable
   de type flottant avec la valeur 3.14
3  char grade = 'A'; // Declaration et initialisation d'une variable
   de type caractere avec la valeur 'A'
```

### 1.1.3 Utilisation des variables

Une fois déclarées et éventuellement initialisées, les variables peuvent être utilisées dans le programme pour stocker et manipuler des données. Par exemple :

```
1  #include <stdio.h>;
2  int x = 5;
3  int y = 10;
4  int somme = x + y; // Addition des valeurs des variables x et y
5  printf("La somme de %d et %d est %d\n", x, y, somme); // Affichage
   du resultat
```

### 1.1.4 Portée des variables

La portée d'une variable en C détermine où elle peut être utilisée dans le programme. Les variables peuvent être locales à une fonction, auquel cas elles ne sont accessibles que dans cette fonction, ou elles peuvent être globales, auquel cas elles sont accessibles dans tout le programme.

```
1  #include <stdio.h>;
2
3  int globalVar = 100; // Variable globale
4
5  void exampleFunction() {
6      int localVar = 50; // Variable locale a la fonction
7      printf("La variable globale est %d\n", globalVar); // Acces a la
8      printf("La variable locale est %d\n", localVar); // Acces a la
9  }
10
11 int main() {
12     printf("La variable globale est %d\n", globalVar); // Acces a la
13     // printf("La variable locale est %d\n", localVar); // Cela
14     // generera une erreur car localVar est locale a
15     // exampleFunction
16     exampleFunction();
17     return 0;
18 }
```

Les variables sont un élément essentiel en langage C et constituent la base de la manipulation des données dans les programmes. Il est crucial de comprendre leur déclaration, leur initialisation, leur utilisation et leur portée pour écrire des programmes efficaces et fonctionnels.

## 1.2 Pointeurs

Les pointeurs sont un concept essentiel en langage C. Un pointeur est une variable qui contient l'adresse mémoire d'une autre variable. En d'autres termes, un pointeur pointe vers l'emplacement en mémoire où une valeur est stockée.

### 1.2.1 Déclaration de pointeurs

En langage C, un pointeur est déclaré en précédant le nom de la variable avec l'opérateur d'indirection \*, qui indique que la variable est un pointeur. Voici un exemple de déclaration de pointeur :

```
1  int *ptr; // Declaration d'un pointeur vers un entier
2  float *ptr_float; // Declaration d'un pointeur vers un flottant
3  char *ptr_char; // Declaration d'un pointeur vers un caractere
```

### 1.2.2 Initialisation de pointeurs

Les pointeurs peuvent être initialisés avec l'adresse mémoire d'une variable existante à l'aide de l'opérateur d'adresse &. Voici un exemple d'initialisation de pointeur :

```
1  int var = 10; // Declaration et initialisation d'une variable
2  int *ptr; // Declaration d'un pointeur
3  ptr = &var; // Initialisation du pointeur avec l'adresse de la
    variable var
```

### 1.2.3 Utilisation de pointeurs

Une fois qu'un pointeur est initialisé, il peut être utilisé pour accéder à la valeur à laquelle il pointe ou pour modifier cette valeur. Voici quelques exemples :

```
1  #include <stdio.h>;
2  int var = 10; // Declaration et initialisation d'une variable
3  int *ptr; // Declaration d'un pointeur
4  ptr = &var; // Initialisation du pointeur avec l'adresse de la
    variable var
5  printf("La_valeur_de_var_est_%d\n", var); // Affichage de la
    valeur de var
6  printf("L'adresse_de_var_est_%p\n", &var); // Affichage de l'
    adresse de var
7  printf("La_valeur_pointee_par_le_pointeur_est_%d\n", *ptr); //
    Affichage de la valeur pointee par le pointeur
8  *ptr = 20; // Modification de la valeur pointee par le pointeur
9  printf("La_nouvelle_valeur_de_var_est_%d\n", var); // Affichage de
    la nouvelle valeur de var
```

Les pointeurs sont un concept puissant en langage C, mais ils nécessitent une manipulation prudente pour éviter les erreurs de segmentation et les fuites de mémoire.

## 1.3 Types de données

Les types de données en langage C déterminent la nature des valeurs qu'une variable peut contenir. Le langage C prend en charge plusieurs types de données de base, notamment les entiers, les flottants et les caractères, ainsi que des types de données dérivés tels que les tableaux et les structures.

### 1.3.1 Types de données de base

Les types de données de base définissent les valeurs simples que peuvent contenir les variables en langage C. Voici quelques-uns des types de données de base les plus couramment utilisés :

- a) **int** : Pour les entiers signés.
- b) **float** : Pour les nombres à virgule flottante.
- c) **double** : Pour les nombres à virgule flottante doubles précision.
- d) **char** : Pour les caractères ASCII.

Voici comment ces types de données peuvent être utilisés dans des déclarations de variables :

```
1  int age = 30; // Declaration d'une variable de type entier
2  float poids = 75.5; // Declaration d'une variable de type flottant
3  double prix = 99.99; // Declaration d'une variable de type double
4  char grade = 'A'; // Declaration d'une variable de type caractere
```

### 1.3.2 Types de données dérivés<sup>1</sup>

Outre les types de données de base, le langage C offre la possibilité de créer des types de données dérivés, tels que les tableaux, les structures et les pointeurs. Ces types de données permettent de regrouper des valeurs connexes ou d'adresser des données de manière plus complexe.

- a) **Tableaux** :

Un tableau est une collection ordonnée d'éléments du même type. Les éléments d'un tableau sont accessibles via un index numérique. Voici un exemple de déclaration et d'utilisation d'un tableau en langage C :

```
1  int tableau[5]; // Declaration d'un tableau d'entiers de
    taille 5
2  tableau[0] = 10; // Attribution de la valeur 10 au premier
    element du tableau
3  tableau[1] = 20; // Attribution de la valeur 20 au deuxieme
    element du tableau
```

1. Refer back to [1] for this section. Check well that all is correct.

b) **Structures :**

Une structure est une collection de variables de types différents regroupées sous un seul nom. Elle permet de définir des types de données personnalisés. Voici un exemple de déclaration et d'utilisation d'une structure en langage C :

```

1      #include <stdio.h>;
2      #include <string.h>;
3
4      struct Personne {
5          char nom[50];
6          int age;
7          float taille;
8      };
9
10     struct Personne p1; // Declaration d'une structure de type
        Personne
11     strcpy(p1.nom, "John_Doe"); // Attribution d'une valeur au
        champ nom
12     p1.age = 30; // Attribution d'une valeur au champ age
13     p1.taille = 1.75; // Attribution d'une valeur au champ
        taille

```

c) **Chaînes de caractères :**

Les chaînes de caractères en C sont des tableaux de caractères terminés par un caractère nul ('\0'). Elles sont utilisées pour représenter et manipuler du texte. Voici un exemple de déclaration et d'utilisation de chaînes de caractères en langage C :

```

1      #include <stdio.h>;
2      #include <string.h>;
3
4      char chaine[50]; // Declaration d'une chaine de caracteres
5      strcpy(chaine, "Bonjour"); // Attribution d'une valeur a la
        chaine
6      printf("%s\n", chaine); // Affichage de la chaine

```

Les types de données en langage C offrent une flexibilité et une puissance considérables pour la manipulation des données. Il est essentiel de comprendre ces types de données et leurs utilisations pour écrire des programmes efficaces et fonctionnels.



## 1.4 Exercices

### Types de données :

1. Écrivez un programme C pour déclarer une variable de chaque type de données de base et initialisez-les avec des valeurs.
2. Déclarez un tableau d'entiers de taille 10 et initialisez-le avec des valeurs de votre choix. Affichez ensuite ces valeurs à l'écran.
3. Créez une structure `Personne` avec des champs pour le nom, l'âge et la taille. Déclarez une variable de type `Personne` et initialisez-la avec des valeurs fictives. Affichez ensuite ces valeurs à l'écran.
4. Écrivez une fonction en C pour inverser une chaîne de caractères donnée.
5. Écrivez une fonction en C pour trier un tableau d'entiers en utilisant l'algorithme de tri à bulles.

### Types de données de base :

1. Écrivez un programme C pour convertir un nombre entier en binaire.
2. Écrivez une fonction en C pour calculer la somme des chiffres d'un nombre entier.
3. Déclarez une variable de type `char` et utilisez-la pour stocker une lettre majuscule. Ensuite, utilisez une opération pour convertir cette lettre en minuscule.
4. Écrivez un programme C pour vérifier si un nombre donné est premier ou non.
5. Écrivez une fonction récursive en C pour calculer le factoriel d'un nombre entier.

### Types de données dérivés :

1. Écrivez une fonction en C pour concaténer deux chaînes de caractères données.
2. Déclarez un tableau de structures `Etudiant` avec des champs pour le nom, l'âge et la moyenne. Initialisez-le avec des valeurs fictives et affichez ensuite ces valeurs à l'écran.
3. Écrivez une fonction en C pour ajouter un élément à une liste chaînée.
4. Implémentez une file (queue) en utilisant des listes chaînées en C.
5. Écrivez une fonction en C pour supprimer un élément d'un arbre binaire de recherche.

# Chapitre Algorithmes de recherche et de tri

## 2.1 Qu'est-ce qu'un algorithme ?

### Définition 2.1. Algorithme

*Un algorithme est constitué d'instructions étape par étape sans ambiguïté pour résoudre un problème donné.*



**Exemple 2.1** Considérons le problème de la préparation d'une omelette. Pour préparer une omelette, on suit les étapes indiquées ci-dessous :

1. Récupérez la poêle.
2. Récupérez l'huile.
  - (a). Avons-nous de l'huile ?
    - I. Si oui, mettez-le dans la poêle.
    - II. Si non, voulons-nous acheter de l'huile ?
      - A. Si oui, sortez et achetez.
      - B. Si non, on termine.
3. Allumez la cuisinière, etc...

Ce que nous faisons, c'est que, pour un problème donné (préparer une omelette), nous proposons une procédure étape par étape pour le résoudre.

## 2.2 Tri

### 2.2.1 Tri à bulle

### 2.2.2 Tri par tas

## 2.3 Algorithmes de recherche

### 2.3.1 Recherche systématique

### 2.3.2 Recherche par

## 2.4 Exercices

# Chapitre Complexité algorithmique

## 3.1 Récursion

## 3.2 La notation $O$

## 3.3 Les notations $\Omega$ et $\theta$

## 3.4 Complexité de certains algorithmes

## 3.5 Exercices

## **Chapitre Structures de données avancées**

### **4.1 Listes chaînées**

### **4.2 Piles**

### **4.3 Files**

### **4.4 Arbres binaires**

### **4.5 Arbres**

### **4.6 Graphes**

### **4.7 Exercices**

# Chapitre Applications d'algorithmes en IA

## 5.1 Greedy algorithm

## Bibliographie

- [1] Karumanchi, N. (2017). Data structures and algorithms made easy.
- [2] Knuth, D. E. (1986). *The T<sub>E</sub>X Book*. Addison-Wesley Professional.