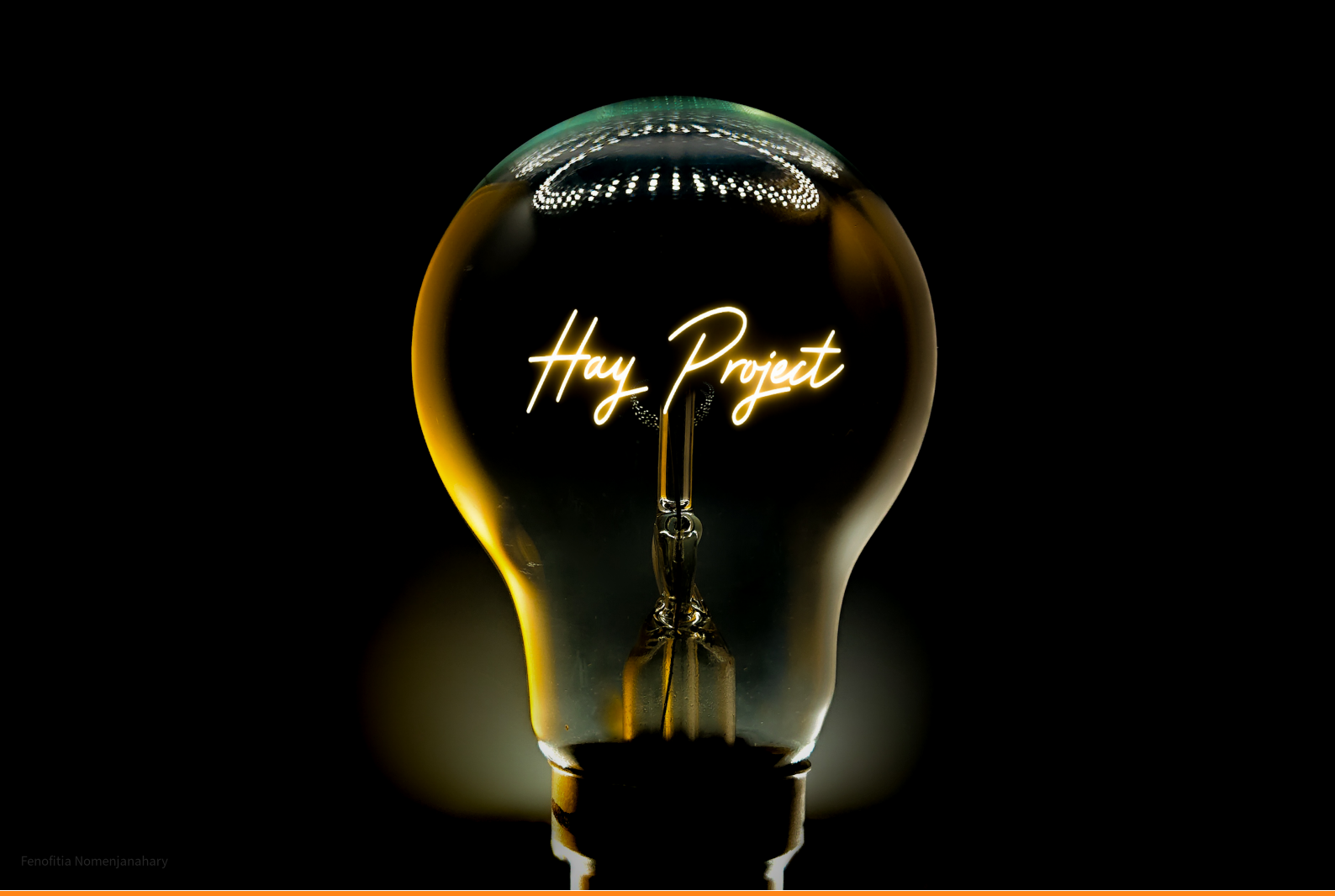

Kajy University: Informatique

Algorithmes et structures de données

Author: Dimby Rabearivony

Date: 10 mai 2024

Version: 1.0

"Ny fahalalana no valamparihiko."

Table des matières

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

————————-o0o————————-

Bienvenue dans ce cours sur les structures de données et les algorithmes à Kajy University, Antananarivo, Madagascar. Ce cours est conçu pour les étudiants de première année qui souhaitent acquérir une solide compréhension des concepts fondamentaux en informatique. Les structures de données et les algorithmes constituent le fondement de la programmation et du développement logiciel, essentiels pour toute carrière en informatique ou en ingénierie logicielle.

L’objectif principal de ce cours est de vous familiariser avec les différentes structures de données, telles que les tableaux, les listes chaînées, les piles, les files, les arbres binaires, et les graphes. Vous apprendrez également les algorithmes associés à ces structures, tels que les algorithmes de tri, de recherche, et de parcours. Le cours abordera également la complexité algorithmique, un élément crucial pour évaluer l’efficacité des algorithmes.

Les structures de données et les algorithmes sont au cœur de nombreux aspects de l’informatique. Ils permettent de stocker, organiser, et manipuler des données de manière efficace. Comprendre ces concepts vous aidera à résoudre des problèmes complexes, à développer des programmes performants, et à améliorer vos compétences en programmation.

Voici ce que vous apprendrez au cours de ce semestre :

- **Structures de données fondamentales** : Vous apprendrez les concepts de base comme les variables, les pointeurs, les types de données, les tableaux, et les chaînes de caractères. Cela servira de fondation pour comprendre les structures de données avancées et les algorithmes.
- **Algorithmes de recherche et de tri** : Cette partie couvre les algorithmes courants de tri et de recherche, y compris le tri à bulles, le tri par insertion, le tri rapide, la recherche linéaire, et la recherche binaire. Ces algorithmes sont essentiels pour manipuler et organiser des données.
- **Complexité algorithmique** : Vous découvrirez les concepts de complexité algorithmique, y compris la notation O, Ω et Θ. Vous étudierez également la récursion et comprendrez comment elle affecte la complexité algorithmique.
- **Structures de données avancées** : Cette section traite des structures de données avancées comme les listes chaînées, les piles, les files, les arbres binaires, et les graphes. Vous apprendrez également des algorithmes de parcours et leurs applications.

Ce cours adopte une approche pratique. Vous aurez des sessions théoriques pour comprendre les concepts de base, suivies de travaux pratiques pour appliquer ces concepts dans des exercices et des projets. Vous travaillerez avec le language C, pour mettre en œuvre des structures de données et des algorithmes, et développer des compétences en programmation.

À la fin du cours, vous aurez une compréhension solide des structures de données et des algorithmes, et vous serez en mesure de les appliquer dans des contextes réels. Ces compétences vous seront utiles tout au long de vos études et de votre carrière professionnelle.

Nous sommes impatients de vous accompagner dans ce voyage d'apprentissage et de vous voir progresser dans le domaine passionnant des structures de données et des algorithmes. Bon cours !

Chapitre Structures de données fondamentales

1.1 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.

1.1.1 Déclaration de variables

En language C, on déclare une variable en précédant son nom le type de donnée qu'elle contient.

```
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
         exampleFunction
7    printf("La variable globale est %d\n", globalVar); // Acces
         a la variable globale
8    printf("La variable locale est %d\n", localVar); // Acces a
         la variable locale
9  }
10
11 int main() {
12   printf("La variable globale est %d\n", globalVar); // Acces
         a la variable globale
13   // printf("La variable locale est %d\n", localVar); // Cela
         generera une erreur car localVar est locale a
         exampleFunction
14   exampleFunction();
15   return 0;
16 }
```

Les variables sont un element essentiel en langage C et constituent la base de la manipulation des donnees dans les programmes. Il est crucial de comprendre leur declaration, leur initialisation, leur utilisation et leur portee pour ecrire 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 *, 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.

1.2.4 Double pointeurs

Les double pointeurs (ou pointeurs vers des pointeurs) contiennent l'adresse mémoire d'un autre pointeur. Ils sont utiles pour passer des pointeurs par référence à des fonctions, pour des structures de données complexes comme des tableaux de tableaux, ou pour manipuler des listes chaînées.

Voici un exemple de déclaration et d'utilisation d'un double pointeur :

```
1  int a = 5;
2  int *ptr = &a; // Pointeur vers int
3  int **double_ptr = &ptr; // Double pointeur vers pointeur int
4
5  printf("La valeur de a est %d\\n", a); // Affichage de la
       valeur de a
6  printf("La valeur pointe par ptr est %d\\n", *ptr); //
       Affichage de la valeur pointee par ptr
7  printf("La valeur pointee par double_ptr est %d\\n", **
       double_ptr); // Affichage de la valeur pointee par
       double_ptr
```

Les double pointeurs sont également utilisés pour retourner des pointeurs depuis des fonctions ou pour allouer dynamiquement de la mémoire pour des structures comme des tableaux 2D.

1.2.5 Utilisation avancée des pointeurs

Les pointeurs permettent des opérations avancées comme l'arithmétique des pointeurs, où vous pouvez ajouter ou soustraire des valeurs pour déplacer le pointeur en mémoire. Voici un exemple :

```
1  int tableau[5] = {1, 2, 3, 4, 5};
2  int *ptr = tableau;
3
```

```
4  printf("Premier␣element:␣%d\\n", *ptr); // Affichage du
       premier element
5  ptr++; // Deplacement du pointeur vers le prochain element
6  printf("Deuxieme␣element:␣%d\\n", *ptr); // Affichage du
       deuxieme element
```

Soyez prudent avec l'arithmétique des pointeurs, car elle peut conduire à des erreurs de segmentation si vous sortez des limites de la mémoire allouée.

1.2.6 Pointeurs et mémoire dynamique

Les pointeurs sont essentiels pour la gestion de la mémoire dynamique en langage C. Avec des fonctions comme 'malloc' et 'free', vous pouvez allouer et libérer de la mémoire au moment de l'exécution. Voici un exemple :

```
1  int *array = (int *)malloc(10 * sizeof(int)); // Allouer de la
       memoire pour 10 entiers
2  if (array == NULL) {
3    printf("Echec␣de␣l'allocation␣de␣memoire!\\n");
4    return 1;
5  }
6
7  for (int i = 0; i < 10; i++) {
8    array[i] = i * 2; // Initialisation de la memoire allouee
9  }
10
11 free(array); // Liberer la memoire
```

La gestion de la mémoire est cruciale lors de l'utilisation des pointeurs. Assurez-vous de toujours libérer la mémoire allouée pour éviter 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, 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, avec leurs tailles habituelles :

a) **int** : Pour les entiers signés (généralement 4 octets).

b) **float** : Pour les nombres à virgule flottante (généralement 4 octets).

c) **double** : Pour les nombres à virgule flottante double précision (généralement 8 octets).

d) **char** : Pour les caractères ASCII (généralement 1 octet).

Voici des exemples de déclaration de variables pour ces types de données :

```
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

Outre les types de données de base, le langage C offre la possibilité de créer des types de données dérivés, comme les tableaux, les structures, les unions et les énumérations.

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 d'un tableau et d'opérations de base :

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

b) **Structures** :

Les structures permettent de regrouper des variables de types différents. Elles permettent de définir des types de données personnalisés. Voici un exemple d'utilisation des structures :

```
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) **Unions** :

Les unions permettent de partager la même zone mémoire entre différents types de données. Voici un exemple de déclaration d'une union :

```
1     union Data {
2       int entier;
3       float flottant;
4       char caractere;
5     };
6
7     union Data d;
8     d.entier = 10; // Utilisation de l'union avec un entier
9     d.flottant = 5.5; // Utilisation avec un flottant (
          ecrase l'entier)
```

d) **Énumérations** :

Les énumérations permettent de définir un ensemble de valeurs nommées. Elles sont utiles pour créer des listes de constantes. Voici un exemple d'utilisation des énumérations :

```
1     enum Jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,
          SAMEDI, DIMANCHE};
2
3     enum Jour aujourdhui = MERCREDI; // Declaration d'une
          enumeration et initialisation
```

1.3.3 Typedef

Le mot-clé *typedef* permet de créer des alias pour des types de données existants. Cela peut simplifier des déclarations complexes, rendre le code plus lisible, et améliorer l'abstraction des types. 'typedef' est couramment utilisé pour simplifier des déclarations de pointeurs, de structures, d'unions, et de fonctions.

Alias pour des types de base

Voici un exemple d'utilisation de 'typedef' pour créer un alias pour un type de base :

```
1 typedef int Entier; // Creation d'un alias pour int
2 Entier age = 30; // Utilisation du typedef pour creer un alias
```

Cet exemple montre comment 'typedef' peut simplifier des déclarations. Utiliser des alias descriptifs peut rendre le code plus lisible et compréhensible, en particulier lorsque les types de base ont des applications spécifiques.

Utilisation avec des pointeurs

Les pointeurs en C peuvent être complexes à déclarer, surtout pour les pointeurs de fonctions ou les structures dérivées. 'typedef' simplifie les déclarations de pointeurs en créant des alias :

```
1 typedef int* PointeurEntier; // Alias pour un pointeur vers
      int
2 PointeurEntier ptr = NULL; // Declaration d'un pointeur vers
      int
```

Vous pouvez également utiliser 'typedef' pour simplifier des déclarations de pointeurs de fonctions, rendant le code plus lisible :

```
1 typedef int (*Fonction)(int, int); // Alias pour un pointeur
      vers une fonction
```

Utilisation avec des structures

'typedef' est largement utilisé pour simplifier les déclarations de structures complexes. Cela facilite l'utilisation de structures pour regrouper des variables de types différents :

```
1 typedef struct {
2   char nom[50];
3   int age;
4   float taille;
5 } Personne;
6
7 Personne p1; // Utilisation du typedef pour nommer une
      structure
```

Avec 'typedef', les déclarations de structures deviennent plus concises et lisibles, ce qui peut simplifier le code et faciliter les modifications futures.

Utilisation avec des unions

Les unions permettent de partager la même zone mémoire entre différents types de données. ‘typedef‘ peut être utilisé pour nommer des unions, ce qui simplifie l’utilisation des structures complexes :

```
1  typedef union {
2    int entier;
3    float flottant;
4    char caractere;
5  } Donnees;
6
7  Donnees d; // Utilisation du typedef pour nommer une union
```

Utilisation avec des énumérations

Les énumérations permettent de définir des ensembles de valeurs nommées. ‘typedef‘ facilite la déclaration et l’utilisation des énumérations :

```
1  typedef enum {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI,
       DIMANCHE} Jour;
2
3  Jour aujourdhui = MERCREDI; // Utilisation de l'enumeration
       avec typedef
```

Avantages de ‘typedef‘

‘typedef‘ offre plusieurs avantages pour le code C, notamment :

- **Simplification des déclarations complexes** : ‘typedef‘ permet de rendre le code plus concis en créant des alias pour des déclarations compliquées. - **Lisibilité accrue** : Les alias créés avec ‘typedef‘ rendent le code plus facile à lire et à comprendre. - **Abstraction des types** : ‘typedef‘ permet de créer des abstractions pour des types de données, ce qui facilite les modifications futures. - **Réutilisation du code** : Les alias créés avec ‘typedef‘ peuvent être réutilisés dans différents contextes, ce qui améliore la maintenabilité du code.

Dans l’ensemble, ‘typedef‘ est un outil puissant en langage C qui permet de simplifier les déclarations complexes, de rendre le code plus lisible, et de faciliter la création de types de données personnalisés. Que ce soit pour des pointeurs, des structures, des unions, ou des fonctions, ‘typedef‘ est un outil précieux pour les développeurs C.

1.4 Exercices

1.4.1 Variables et types de données

1. Quelle est la différence entre les types de données suivants :
   (a). **int**
   (b). **float**
   (c). **char**
2. Écrivez un programme C qui déclare des variables pour chaque type de données mentionné et affiche leurs valeurs.
3. Écrivez un programme C qui utilise des variables de différents types de données pour calculer la somme d'un entier et d'un flottant, puis affichez le résultat.
4. Quelles sont les limites de valeurs pour les types de données suivants :
   (a). **int**
   (b). **float**
   (c). **double**

   Indication : Considérer le nombre de bits et octets. Par exemple, un caractère ASCII est représenté par un octet, c'est-à-dire 8 bits. Le caractère "A" est représenté par 01000001. Avec huit chiffres de 0 et 1, on peut representer $2^8 = 256$ valeurs. Pour les nombres signés (int, float, double, ...), l'opposé est representé différemment.

1.4.2 Pointeurs

1. Quelle est la différence entre une variable et un pointeur ?
2. Écrivez un programme C qui utilise des pointeurs pour modifier la valeur d'une variable.
3. Écrivez un programme C qui utilise des pointeurs pour échanger les valeurs de deux variables.
4. Expliquez ce qu'est un pointeur nul (null pointer) et ce qui se passe si vous tentez d'utiliser un pointeur nul.

1.4.3 Tableaux

1. Déclarez un tableau de 5 entiers et initialisez-le avec des valeurs. Écrivez un programme C pour afficher les valeurs du tableau.
2. Modifiez le programme pour inverser les éléments du tableau et afficher le tableau inversé.
3. Écrivez un programme C qui calcule la somme de tous les éléments d'un tableau d'entiers de longueur 'n'.
4. Que se passe-t-il si vous essayez d'accéder à un index hors limites (out of bounds) dans un tableau C ? Écrivez un programme pour le démontrer.

1.4.4 Chaînes de caractères

1. Quelle est la particularité des chaînes de caractères en C par rapport aux autres types de données ?
2. Écrivez un programme C qui déclare une chaîne de caractères et utilise la bibliothèque ‘string.h‘ pour copier du texte dans la chaîne.
3. Écrivez un programme C qui concatène deux chaînes de caractères en utilisant ‘strcat‘.
4. Comment compare-t-on deux chaînes de caractères en C ? Écrivez un programme qui compare deux chaînes et indique si elles sont identiques ou non.

1.4.5 Structures

1. Déclarez une structure en C pour représenter une personne avec un nom, un âge, et une taille.
2. Écrivez un programme C qui utilise cette structure pour créer une personne, assigner des valeurs, et afficher les informations de la personne.
3. Écrivez un programme C qui utilise des structures pour représenter un point 3D avec des coordonnées x, y, z. Ensuite, écrivez une fonction qui calcule la distance entre deux points 3D.
4. Créez une structure pour représenter une voiture avec un nom, une année, et un prix. Écrivez un programme C qui utilise cette structure pour créer une voiture et afficher ses propriétés.