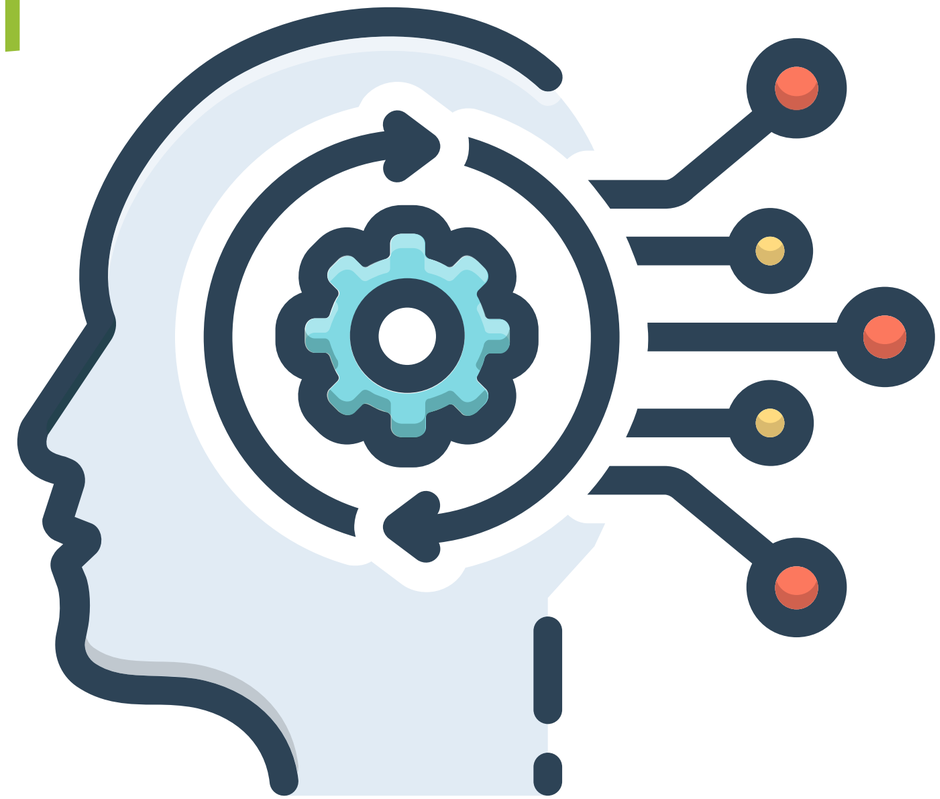
DR B. DIOP
babacar.diop@ussein.edu.sn

Algorithmes et Programmation II

Structures



INFO – AGROTIC

Last update: oct. 2025

Les Structures : définition

Définition

- Une structure est un type de données qui permet de regrouper plusieurs variables (même de types différents) sous un même nom.
- Utile pour représenter des objets complexes : étudiants, livres, véhicules, etc.

Syntaxe de base :

```
struct Etudiant {
    char nom[50];
    int age;
    float moyenne;
};
```

Définit un enregistrement **Etudiant** composé de :

- une chaîne de caractères (nom)
- un entier (âge)
- un réel (moyenne)

Les Structures : utilisation

Déclaration et utilisation

- Déclaration d'une variable de type **struct Etudiant** :

```
struct Etudiant etu1;
```

- Accès aux membres de la variable **etu1** après création :

```
strcpy(etu1.nom, "Babacar");
etu1.age = 25;
etu1.moyenne = 14.5;
```

- Affichage

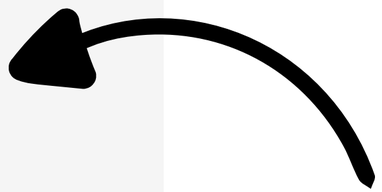
```
printf("Nom: %s, Age: %d, Moyenne: %.2f\n", etu1.nom, etu1.age, etu1.moyenne);
```

Structures imbriquées

Structures à l'intérieur de structures

- Une structure peut contenir d'autres structures
- **Exemple** de structures imbriquées :

```c
struct Adresse {
    char ville[30];
    char rue[50];
};

struct Etudiant {
    char nom[50];
    int age;
    struct Adresse adr;
};
```



Définir un premier enregistrement **Adresse** composé de :

- deux chaîne de caractères
  - **ville** et **rue**

Ensuite, définir un second enregistrement **Etudiant** composé de :

- une chaîne de caractères (**nom**)
- un entier (**age**)
- un réel (**moyenne**)

Structures imbriquées

Accès aux membres imbriqués

- L’accès aux membres imbriqués s’effectue par une combinaison de niveaux de points (.)

```
strcpy(etu1.adr.ville, "Dakar");
strcpy(etu1.adr.rue, "Rue de l’Université");
```

Pointeurs sur structures

Manipuler les structures avec des pointeurs

- Déclaration d'un pointeur sur une structure :

```
struct Etudiant *ptr;
ptr = &etu1;
```

- Accès aux membres via le pointeur

```
printf("Nom: %s\n", ptr->nom);
ptr->age = 26;
```

-> est l'opérateur utilisé pour accéder aux membres via un pointeur.

Passing Struct Pointers to Functions

Pourquoi utiliser des pointeurs sur les structures ?

- Passer une structure entière à une fonction copie tout son contenu, ce qui peut être coûteux en mémoire.

- Passer un pointeur permet :
  - d'éviter la copie complète,
  - et de modifier directement la structure originale.

Exemple 1 : passage par valeur

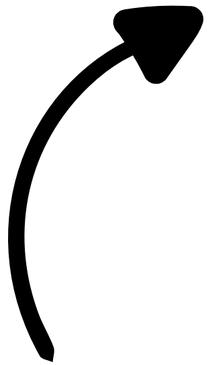
```c
#include <stdio.h>

struct Etudiant {
    char nom[50];
    int age;
};

void afficher(struct Etudiant e) {
    printf("Nom: %s, Age: %d\n", e.nom, e.age);
}

int main() {
    struct Etudiant etu = {"Babacar", 25};
    afficher(etu); // Copie de la structure
    return 0;
}
```

Ici, la fonction reçoit une copie : toute modification ne change pas la variable originale.

Passing Struct Pointers to Functions

Avantages

- Évite la copie (plus rapide pour de grandes structures).
- Permet à la fonction de modifier les valeurs réelles.



Exemple 2: passage par adresse

```c
#include <stdio.h>

struct Etudiant {
    char nom[50];
    int age;
};

void modifier(struct Etudiant *e) {
    e->age = 30; // Modifie directement la variable
}

int main() {
    struct Etudiant etu = {"Babacar", 25};
    modifier(&etu); // On passe l'adresse
    printf("Age après modification: %d\n", etu.age);
    return 0;
}
```

Tableaux de pointeurs sur structures en C

Concept

- Un tableau stocke des éléments du même type en mémoire contiguë.
- On peut créer des tableaux de :
  - types primitifs (int, char, float),
  - structures (types définis par l'utilisateur),
  - pointeurs vers des structures.
- Objectif : gérer plusieurs objets dynamiques de même type.
  -

Tableau statique 1D de pointeurs sur structures

```c
typedef struct node {
    int number;
    char character;
} node;

node *p1 = malloc(sizeof(node));
node *p2 = malloc(sizeof(node));

node *arr[2];
arr[0] = p1;
arr[1] = p2;

arr[0]->number = 100;
arr[0]->character = 'a';
```

Sortie

```
100 and a
200 and b
```

Remarque :

1. Chaque élément du tableau est un pointeur vers une structure stockée ailleurs.
2. Le pointeur sur la structure nécessite une allocation mémoire.

Tableau statique 2D de pointeurs sur structures

```
<struct_name> * <array_name>[rows][cols];
<array_name>[i][j] = <pointer_to_structure>;
```

```
node *arr[2][2];
arr[0][0] = p1;
arr[0][1] = p2;
arr[1][0] = p3;
arr[1][1] = p4;

printf("%c - %d", arr[i][j]->character, arr[i][j]->number);
```

Sortie

```
a - 100   b - 200
c - 300   d - 400
```

Remarque :

Le type du tableau doit être struct_name * (pointeur vers structure).

Tableaux dynamiques 1D de pointeurs sur structures

```
<struct_name> **array =
    (<struct_name **>) malloc(sizeof(<struct_name *>) * size);
```

```
node **arr = malloc(sizeof(node *) * 4);
arr[0] = p1;
arr[1] = p2;
arr[2] = p3;
arr[3] = p4;
```

Sortie

```
a - 100
b - 200
c - 300
d - 400
```

Différence avec statique :

- Alloué dans le heap, non sur la pile.
- Peut être redimensionné.
- Nécessite un double pointeur (node **).

Tableaux dynamiques 2D (triple pointeurs)

```
// Initialisation des pointeurs sur les structure
st_ptr1->number = 100;
st_ptr1->character = 'a';

st_ptr2->number = 200;
st_ptr2->character = 'b';

st_ptr3->number = 300;
st_ptr3->character = 'c';

st_ptr4->number = 400;
st_ptr4->character = 'd';

// Declarer le tableau dynamiquement pour la structure
// pointers    Step - 1
node*** structure_array
    = (node***)malloc(sizeof(node***) * 2);

// Double pointeurs utilisé à chaque indice pour pointer vers
// tableau de poinbteurs de structure - Step 2
for (int i = 0; i < 2; i++) {
    structure_array[i] = (node**)malloc(sizeof(node**));
}

// Initialisation du tableau de pointeurs de structure - Step 3
structure_array[0][0] = st_ptr1;
structure_array[0][1] = st_ptr2;
structure_array[1][0] = st_ptr3;
structure_array[1][1] = st_ptr4;
```

Principe

- Un tableau de tableaux contenant des pointeurs vers structures.
- Utilise un triple pointeur (node ***).

Etape 1/

- Déclaration du tableau dynamique.

Etape 2/

- Parcours le tableau et alloue la mémoire

Etape 3/

- Initialisation du tableau de pointeurs