# INFO2009-Introduction à l'informatique

## Aide à la réussite - Session nº6 - Structures de données

Bertrand Alexis

Université de Liège

2023

# Rappel

Les structures de données sont un moyen pour rassembler des variables pouvant être hétérogènes (de types différents) dans un même champs. L'intérêt de faire ceci est de pouvoir manipuler les variables ensemble, comme une seule entité. C'est notamment très intéressant quand on veut désigner quelque chose qui possède plusieurs attributs. Par exemple, une personne est définie par son nom, son prénom, et son âge. Grâce à un type de donnée structuré, nous pouvons facilement créer cette représentation en gardant les champs ensembles. De cette manière, si une année passe, nous pouvons facilement modifier l'âge de la personne sans devoir réassocier le bon nom et prénom.

# Définir un type de données structuré (1/2)

En C, nous pouvons définir un type structuré de 2 manières.

## 1ère manière

```
struct nom_structure {
    type1 var1;
    type2 var2;
    ...
};
```

Dans ce cas, afin d'initier une variable à ce nouveau type, il est nécessaire de faire : `struct nom_structure var;`

# Définir un type de données structuré (2/2)

## 2ème manière

```
typedef struct {
    type1 var1;
    type2 var2;
    ...
} nom_structure;
```

Dans ce cas, afin d'initier une variable à ce nouveau type, il suffit de faire : `nom_structure var;`

Pourquoi ne mettons-nous plus le struct avant le nom de la structure ?

Tout simplement grâce au mot-clé `typedef` qui permet de donner un nom symbolique à un type. À ne pas confondre avec `#define` qui donne un alias autant à une valeur qu'un type ou une fonction !

# Comment utiliser un type de donnée structuré ?

Pour accéder à un une variable du type structuré, nous pouvons utiliser l'opérateur .

## Exemple

```
struct nom_struct var;
var.var1 = valeur;
```

# Fonction avec type de donnée structuré

Une variable d'un type structuré peut être donné en argument à une fonction. Imaginons que nous avons le type structuré ci-dessous. Regardons maintenant cette fonction ainsi que le main qui suit. Quel est le résultat affiché sur le terminal suite à l'exécution de la fonction main ?

```
struct nom {
    int n[2];
};
```

# Fonction avec type de donnée structuré (1/4)

```
void permutation(struct nom s){
    int t = s.n[0];
    s.n[0] = s.n[1];
    s.n[1] = tmp;
}
int main() {
    struct nom s;
    s.n[0] = 1;
    s.n[1] = 2;
    permutation(s);
    printf("%d%d", s.n[0], s.n[1]);
    return EXIT_SUCCESS;
}
```

# Fonction avec type de donnée structuré (2/4)

Contrairement à ce qu'on pourrait penser, le terminal nous affiche 12 et non 21.

Pourquoi ?

Il faut passer une variable d'un type structuré par pointeur en argument à une fonction, sinon le changement n'est que local à la fonction !

Lorsqu'on utilise un pointeur et qu'on souhaite accéder à une des variables de la structure, on peut utiliser l'opérateur `var->var1` au lieu de `*var.var1`

# Fonction avec type de donnée structuré (3/4)

```
void permutation(struct nom* s){
    int t = s->n[0];
    s->n[0] = s->n[1];
    s->n[1] = tmp;
}
int main() {
    struct nom s;
    s.n[0] = 1;
    s.n[1] = 2;
    permutation(&s);
    printf("%d%d", s.n[0], s.n[1]);
    return EXIT_SUCCESS;
}
```

# Fonction avec type de donnée structuré (4/4)

Le résultat affiché sur le terminal est désormais 21. Afin de modifier les variables d'une structure dans une fonction, il est donc requis de passer la variable par pointeur comme argument de la fonction.

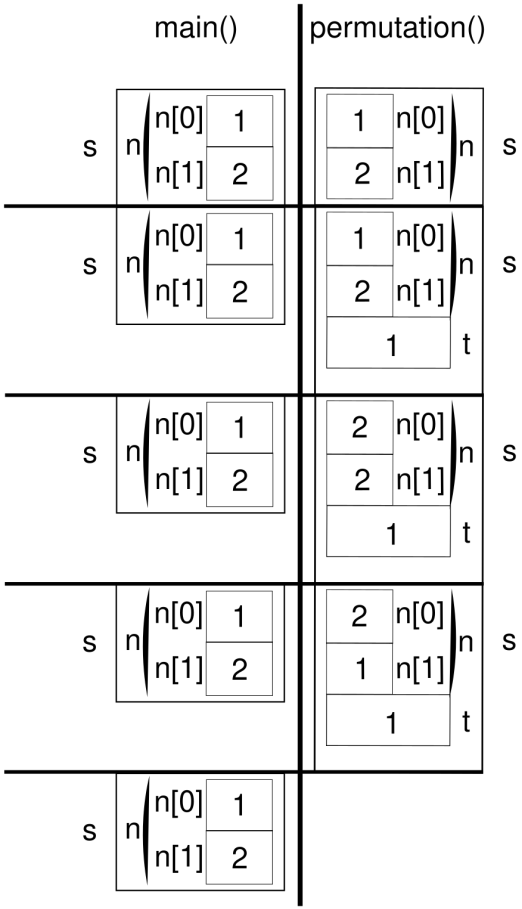
# Types de passage d'arguments (1/2)

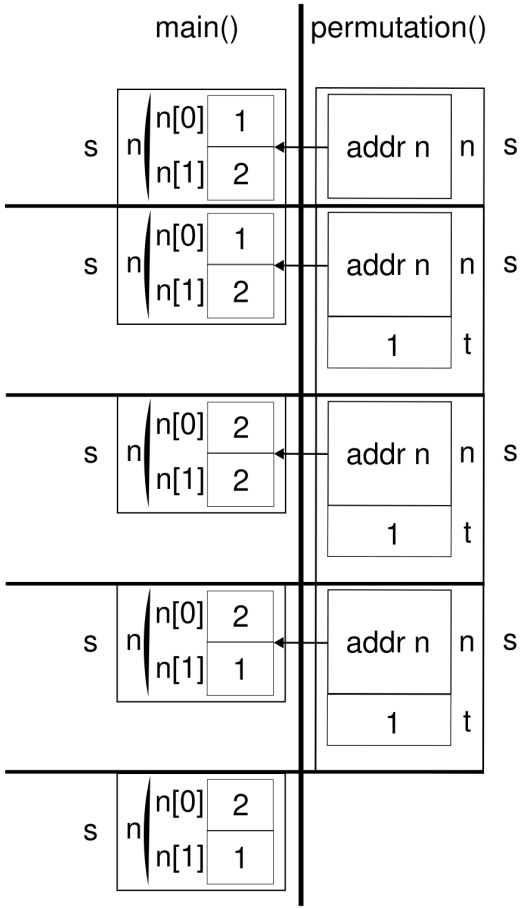
## Pourquoi cette différence entre les 2 exemples ?

Dans le premier exemple, on fait ce qu'on appelle un "passage d'argument par variable". C'est comme si on recopiait dans la fonction la valeur de la variable. Quand on modifie la variable dans la fonction, on modifie en fait la copie et donc pas l'originale. Dans le deuxième exemple, celui qui nous affiche le résultat escompté, nous avons fait un "passage d'argument par pointeur". À la place de recopier la valeur, on va donner l'adresse de la variable. On peut alors accéder à la variable d'origine grâce à un pointeur sur son adresse, c'est pour ça que la variable présente dans le main n'a pas seulement été modifiée dans la variable locale à la fonction.

# Types de passage d'arguments (2/2)

Nous allons essayer de comprendre ce qu'il se passe par un dessin représentant les 2 types de passage d'arguments :



# Exercices

1. Une entreprise vous engage afin de construire un moyen de représenter ses employés.
   - 1.1 Définissez la structure. Cette dernière doit contenir le nom et le prénom de l'employé (max 32 caractères) en plus de son salaire mensuel.
   - 1.2 L'entreprise contient 3 employés, dans une fonction `main`, initialisez un tableau pour contenir ces 3 employés et remplissez les champs adéquats avec :
     - Doe John 1900
     - Renaud Alice 2200
     - Herbert Bob 2300
   - 1.3 John a reçu une promotion de 200. Écrivez une fonction qui permet d'ajouter un certain montant à son salaire mensuel. De plus, montrez comment appeler cette fonction à partir de la fonction `main`.

# Exercices

2. (Examen de janvier 2019)
   - 2.1 Écrire un fragment de code C définissant les deux types structurés suivants :
     - Un type point représentant les coordonnées cartésiennes (x, y) d'un point dans le plan, où x et y sont des valeurs réelles.
     - Un type segment représentant un segment de droite caractérisé par son origine et sa destination (qui sont des points).
   - 2.2 Écrire une fonction C prenant en arguments quatre réels $x_1$, $y_1$, $x_2$ et $y_2$ définissant les extrémités $(x_1, y_1)$ et $(x_2, y_2)$ d'un segment de droite, et retournant un pointeur vers une représentation de ce segment, nouvellement allouée.
   - 2.3 Écrire une fonction C permettant de libérer une représentation d'un segment créée par la fonction obtenue au point (2).

# Exercices

3. 3.1 Définissez une structure de donnée représentant un élément d'une liste liée contenant un entier. C'est à dire une structure contenant un champs `valeur` de type entier et un champs pointant vers le prochain élément nommé `suivant`. Attention que `suivant` doit pointer vers `NULL` par défaut.
   3.2 Ensuite, créez une fonction prenant en argument un pointeur vers le premier élément de la liste, et un nouvel entier qui allouera un nouvel élément et l'insérera à la fin de la boucle.
   3.3 Enfin, simulez un appel

# Exercices

4. (Examen de janvier 2023)
   - 4.1 Écrire un fragment de code définissant un type structuré capable de représenter une séquence de mots. Une instance de ce type est composée d'un champ `nb_mots` représentant le nombre de mots qui appartiennent à la séquence, et d'un champ mots pointant vers un vecteur de `nb_mots` pointeurs vers des chaînes de caractères, correspondant chacune à un mot de la séquence.

# Exercices

4.2 Écrire une fonction prenant en argument une chaîne de caractères, et retournant un pointeur vers une représentation nouvellement allouée de la séquence des mots qui composent cette chaîne. On considère que ces mots sont délimités dans la chaîne par le caractère d'espacement ' '. Par exemple, la chaîne de caractères contenant "Introduction à l'informatique" est composée des mots "Introduction", "à" et "l'informatique". Le nombre de mots et leur longueur ne sont pas connus à l'avance, et peuvent grandement varier d'une chaîne à une autre. En cas d'erreur, la fonction doit retourner un pointeur vide.

4.3 Écrire une fonction permettant de libérer une représentation d'une séquence de mots créée par la fonction obtenue au point précédent.