Les paires, tuples et structures

Conteneur = collection d'objets de même type. Par exemple, dans bibliothèque, livre, mais également des dvd

Pair = 2 éléments (first et second), de type différent. Création avec make_pair

Tuple = N éléments, création avec make_tuple.

Inconvénient : non nommé, pas de sens sémantique. Par exemple pair<string, string> peut représenter un nom et un prénom, un titre de livre et un auteur, une marque de voiture et un modèle, etc. Cela dit juste que l'on a 2 string, mais ne donne pas de sens à ces string.

Structure = N éléments de type différent, avec des noms. Agrégation de variables (avec nom, type et valeur). Nom de structure.

Par exemple :

```
struct Personne {
    string nom {};
    string prenom {};
};

struct Livre {
    string titre {};
    string auteur {};
};

struct Voiture {
    string marque {};
    string modele {};
};
```

N'importe quel type :

```
struct Personne {
    string nom {};
    int age {}
    double taille {};
    bool masculin { true };
};
```

Voir même contenir d'autre structure :

```
struct Identite {
    string nom {};
    string prenom {};
};

struct Personne {
    Identite identite {};
    int age {}
    double taille {};
    bool masculin { true };
};

Personne personne {};
personne.age = 24;
personne.identite.nom = "toto";
```

Permet de créer de nouveaux types.

Sémantique liée = mieux que tuple/pair (moins de risque d'erreur, meilleur compréhension du code)

Syntaxe pour déclarer : struct, bloc de déclaration, ne pas oublier le ;

Initialisation des membres : par défaut ou avec une valeur (si cela à un sens)

Accéder aux variable membres. En lecture et en écriture

Généricité ? Ajouter un type template, permet de modifier la structure de données. Par exemple :

```
template<typename T>
```

```
struct Personne {
    T identifiant {};
    string nom {};
    string prenom {};
};
```

Utilisation :

```
Personne<int> personne_avec_numero_secu {};
personne_avec_numero_secu.identifiant = 12234567897;


Personne<string> personne_avec_autre {};
personne_avec_autre.identifiant = "1a4slsn";

// utilisation possible avec aussi des structures
Personne<int> personne_avec_identifiant {};
personne_avec_identifiant.identifiant.nom = "toto";
personne_avec_identifiant.identifiant.prenom = "dupont";
```

Remarque : déjà rencontré des types génériques : vector et array. Même principe, mais avec plus de fonctionnalités. On verra par la suite en détail comment ajouter encore plus de fonctionnalités.

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++