Structures et unions

types énumérés

Qu'est-ce qu'une structure

rôle et déclaration de types composés

utilisation de typedef

variables de types composé : déclaration, emploi, pointeurs

tableaux et structures

unions : rôle et déclarations

attention à ne pas économiser !

Types énumérés

Permet d'utiliser un ensemble de symboles à la place de valeurs entières : utile pour éviter de faire des codes peu lisibles.

Exemple : une fonction ou un programme peut avoir, à sa sortie, 5 valeurs différentes qui correspondent à l'apparition de certaines erreurs :

pas d'erreur, *erreur de saisie*, *erreur d'allocation*, *erreur de calcul*, *erreur de fichier*. Si on veut associer une valeur à ces erreurs, on peut leur donner une valeur entière.

Ex : 0 pour pas d'erreur, 1 pour erreur de saisie, etc…

problème : quand on relit le programme, on ne se rappelle pas forcément la signification de ces valeurs.

Types énumérés

On peut utiliser **enum** qui définit un type énuméré, c'est-à-dire un type défini par l'ensemble des valeurs que peut prendre une variable de ce type.

Syntaxe :

```
enum nom_d_enumeration
{
        symbole1,
        symbole2,
        …,
        symbole_n
};
```

Structures et unions

Qu'est-ce qu'une structure ?

Premier pas vers la programmation par objets.

Regroupement de valeurs ou variables de types qui peuvent différents pour décrire un objet.

Ensemble de propriétés ou d'informations relatives à quelque chose que l'on veut représenter de manière simplifiée pour un traitement informatique.

Structures et unions

Pour garder ces différentes informations ensemble : possibilité de définir de nouveaux types, dits composés, qui comporteront un ensemble de valeurs.

Exemple d'objet dont on veut garder les propriétés : une personne pour un logiciel administratif (impossible de décrire une personne complètement pour un ordinateur).

Le nom, le(s) prénom(s), l'age, le lieu de résidence, la nationalité.

Chacune de ces propriétés : représentée par une valeur dont on peut définir le type

Structures et unions

Le nom, le(s) prénom(s) : chaînes de caractère

l'age : entier non signé

le lieu de résidence: chaîne de caractère

la nationalité : chaîne de caractère

au total, pour une personne, on aura besoin de 5 variables : un entier, 4 chaînes de caractère.

Il serait utile de les regrouper pour en faire un nouveau type qui contiendrait toutes ces informations.

Sinon : déclarer 5 variables à chaque fois que l'on veut représenter une personne !

Structures et unions

Cela est possible en C : avec le mot réservé **`struct`**, qui définit un nouveau type.

Syntaxe

struct nom_du_type_composé

{

liste des variables regroupées avec leur type;

};

il ne s'agit pas d'une déclaration de variable !

Indique juste comment est composé le nouveau type !

Structures et unions

Exemple : le type personne déjà évoqué :

```
struct s_pers
{
        char nom[40];
        char prenoms[50];
        unsigned int age;
        char adresse[200];
        char nationalite[3];
};
```

le nouveau type s'appelle : struct s_pers qui contient des **champs**.

utilisable comme n'importe quel autre type, on peut déclarer des variables de ce nouveau type et les manipuler.

Structures et unions

Exemple de déclaration de variable dans un programme :

```
struct s_pers
{
	char nom[40];
	char prenoms[50];
	unsigned int age;
	char adresse[200];
	char nationalite[3];
};

void main()
{
	struct s_pers unePersonne;
	/* unePersonne est une variable de type s_pers */

	/* suite du programme */
}
```

Structures et unions

Emploi du mot réservé **`typedef`** :

contrairement à ce que l'on pourrait croire d'après le nom de ce mot réservé, typedef ne sert pas à définir un type. C'est le mot réservé struct qui a ce rôle.

Le type composé s'écrit struct nom_du_type :

peu pratique

pas cohérent avec les autres types simples, un seul mot :

possibilité de donner un nom plus court en utilisant un 'raccourci' analogue au #define : c'est le rôle de typedef.

Structures et unions

Emploi du mot réservé **`typedef`** :

syntaxe :
typedef struct nom_du_type
{
liste des variables regroupées avec leur type;
} nom_court_du_type;

ainsi on peut employer, à la place de struct nom_du_type, le nom_court_du_type.

Exemple : on veut créer un nouveau type pour travailler avec les nombres complexes. Un complexe est défini par deux valeurs réelles, sa partie entière et sa partie imaginaire

Structures et unions

définition de type sans typedef : on écrit :

```
struct s_complex
{
        float re;
        float im;
};
```

avec typedef :

```
typedef struct s_complex
{
        float re;
        float im;
}       complex;
```

maintenant, `complex` est un nouveau type utilisable (presque) comme n'importe quel autre type. On peut continuer à écrire struct s_complex.

Structures et unions

Manipulation des variables dont le type est composé.

Déclarer une variable d'un type composé : utilisation de plusieurs valeurs, pas d'une seule.

Chaque variable ou **champ** dans le type composé, porte un **nom** grâce auquel on peut accéder au champ.

C'est ce **nom** qui va être utile pour manipuler individuellement les champs.

Avec une variable dont le type est composé, on ne travaille qu'avec les champs individuels.

Structures et unions

Soit une variable dont le type est composé. On accède aux champs en écrivant le nom de cette variable (pas le nom du type), suivi d'un point '.' puis du nom du champ (pas son type) concerné.

Nom_de_variable.nom_du_champ

exemple avec le type complex (on ne réécrit pas sa définition)

```
void main()
{
	complex z_1;  /* aurait pu s'appeler a, x, toto  */

	z_1.re = 1.5;
	z_1.im = -1.;
}
```

Structures et unions

Retour sur cet exemple :

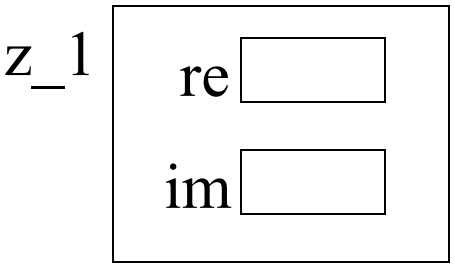
```
void main()
{
	complex z_1; /* aurait pu s'appeler a, x, toto  */

	z_1.re = 1.5;
	z_1.im = -1.;
}
```

z_1.re permet d'accèder au champ nommé re dans le type complex. Ce champ est de type float (choisi ainsi). z_1.re se lit : 'le champ re de la variable z_', qui est de type float.
On peut donc manipuler cette expression (z_1.re est une expression) comme une valeur de type float.
Même chose pour z_1.im !

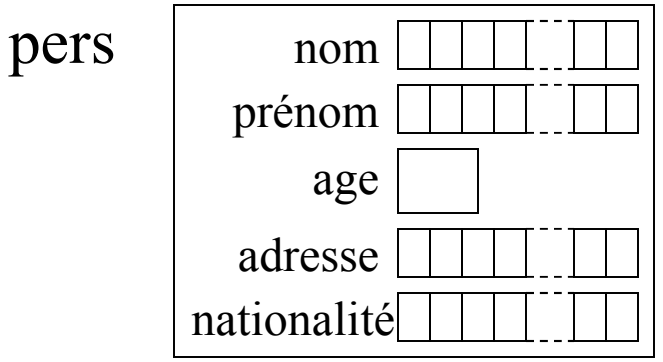
Structures

Schéma pour les types composés : utiliser une boîte pour représenter une variable.

Exemple de la variable *z_1* précédente



Autre exemple : soit *pers* une variable du type personne déjà vu



Application : saisie des champs

Écrire un programme qui définisse un type composé représentant une voiture : les valeurs intéressantes sont : nom du constructeur, nom du modèle, année de fabrication, nombre kilomètres, montant de la prime d'assurance.

Le programme doit réaliser la saisie de toutes ces valeurs (puis les afficher éventuellement). Pour l'instant, pas de fonction.

```
typedef struct s_car
{
	char *constru;
	char *modele;
	unsigned int anneeFab;
	unsigned long int nbKilom;
	double montantPrim;
} t_voiture;
```

Pointeurs et structures

Comme pour n'importe quel autre type, on peut utiliser des pointeurs sur des types composés: exemples de déclaration de variables associés :

complex *pt_z;

ou

voiture *p_vroum;

tout ce qui a été vu à propos des pointeurs reste valable.

On peut accéder directement aux champs du type composé en utilisant une variable de type 'pointeur sur type composé' en utilisant non plus le '.' mais la notation flèche '->'.

Pointeurs et structures

Un exemple pour illustrer : on utilise le type voiture déjà défini.

```
/* définition du type voiture à remettre */
void main()
{
	voiture unVehicule;
	voiture *p_vroum;

	/* schema pointeur non initalise */
	p_vroum = &unVehicule;

	/* schema lien pointeur/variable */
	p_vroum->anneeFab = 2001;
	unVehicule.nbKilom = 53201;
}
```

Pointeurs et structures

Allocation dynamique possible en utilisant l'opérateur sizeof; qui donne la taille du type composé en ajoutant les tailles des éléments :

voiture *pt_voit;

pt_voit=(voiture *)malloc(sizeof(voiture));

on peut donc aussi utiliser des tableaux dont les éléments sont d'un type composé, en utilisant la même syntaxe (les []).

Un exemple à traiter : tableau contenant 5 éléments de type 'voiture', que l'on classe par année de fabrication décroissante : écrire le programme corrspondant.

Structures et paramètres

On peut aussi utiliser les variables de type composé comme paramètres ou arguments de fonction, tout ce qui a été vu concernant les fonctions reste valable pour ces types.

Passage par valeur ou par adresse possible. Cependant, les structures sont en général assez grandes (en taille), c'est pourquoi un passage par valeur n'est pas économique :

rappel : passage de paramètres par valeur : recopie de l'argument dans le paramètre, 2 versions en mémoire. Si l'argument est grand en taille : le paramètre aussi, consommation de mémoire a priori pas utile.

Systématiquement faire un passage par adresse lorsque l'on utilise un paramètre de type composé, sans se soucier du fait qu'il soit en entrée ou en sortie.

Structures et paramètres

On le manipulera, dans la fonction, à l'aide de '->' plutôt qu'avec le '.'

pour les paramètres de type composé qui sont uniquement en entrée, il suffit d'ajouter le modificateur const qui empêchera la modification.

Un exemple : fonctions d'affichage et de saisie d'une variable de type voiture.

Structures imbriquées

Parmi les champs que l'on trouve dans un type composé, nous avons pour l'instant vu que l'on utilisait des types de base du C :

on peut utiliser d'autres types composés, à condition qu'ils soient définis avant. Accès par l'application successive des opérateurs '.' ou '->'

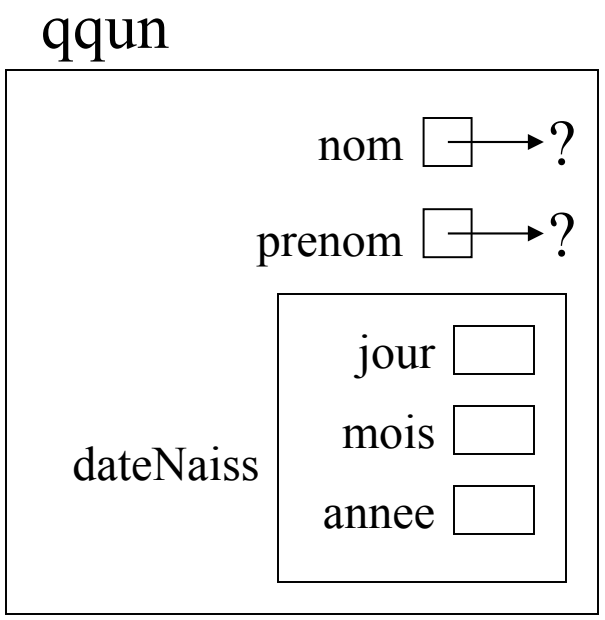
exemple : définit un type *date* utilisé dans un type *personne*.

```
typedef struct s-date
{
	unsigned int jour, mois, annee;
}t_date;
typedef struct s_pers
{
	char *nom;
	char *prenom;
	t_date dateNaiss; /*utilisable car déjà défini*/
} t_personnne;
```

Structures imbriquées

Déclaration d'une variable de type t_pers et initialisation de sa date de naissance :

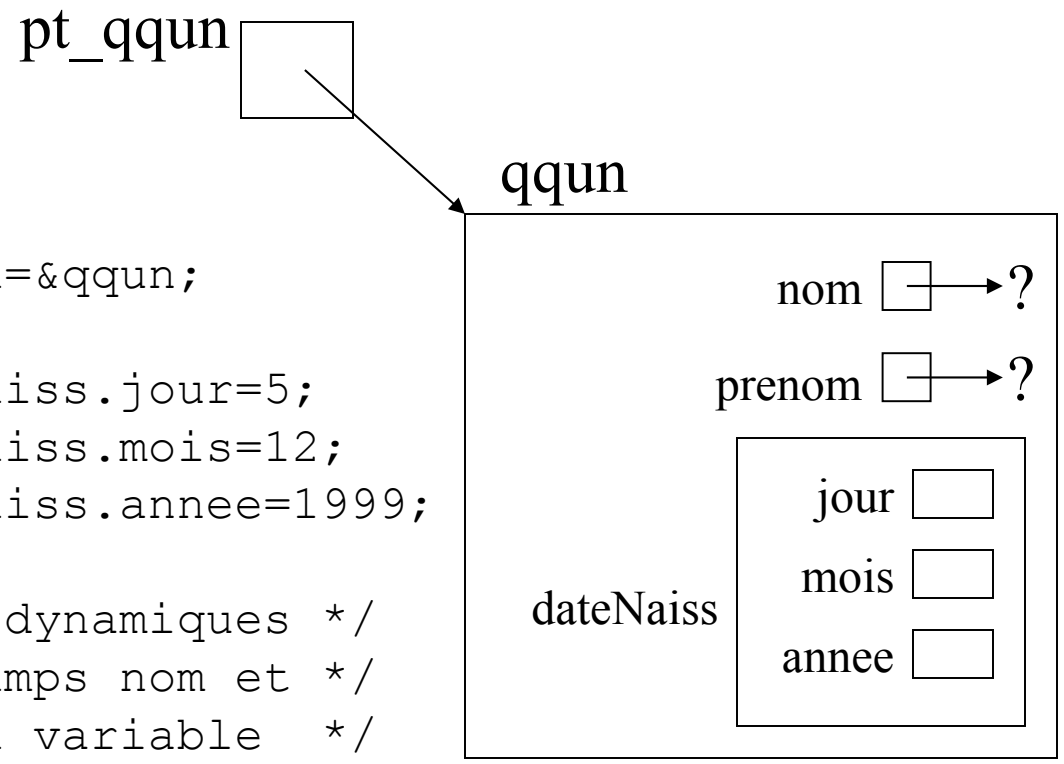
```
void main()
{
        t_pers qqun;

        qqun.dateNaiss.jour=5;
        qqun.dateNaiss.mois=12;
        qqun.dateNaiss.annee=1999;

        /* allocations dynamiques */
        /* pour les champs nom et */
        /* prenom de la variable  */
}
```



Structures imbriquées

Déclaration d'une variable de type t_pers et initialisation de sa date de naissance :

```
void main()
{
        t_pers qqun;
        t_pers *pt_qqun=&qqun;

        pt_qqun->dateNaiss.jour=5;
        pt_qqun->dateNaiss.mois=12;
        pt_qqun->dateNaiss.annee=1999;

        /* allocations dynamiques */
        /* pour les champs nom et */
        /* prenom de la variable  */
}
```



Unions

Les **unions** permettent l'utilisation d'un même espace mémoire par des données de types différents à des moments différents. Différent des structures qui utilisent toujours toutes les variables définies dans le type complexe !

Très peu ou pas utilisées puisque la mémoire n'a pas forcément besoins d'être économisée sur les types.

Syntaxe : proche de celle des struct :

```
union nom_de_union
{
        liste des variables;
};
```

Unions

Au contraire d'une structure, une union ne gardera pas toutes les variables listées en même temps en mémoire. Le compilateur va rechercher, parmi les variables listées dans l'union, celle dont la taille en octets est la plus grande : ce sera la taille de l'union.

On accède aux champs d'une union de la même manière qu'aux champs d'une struct : avec la notation '.'

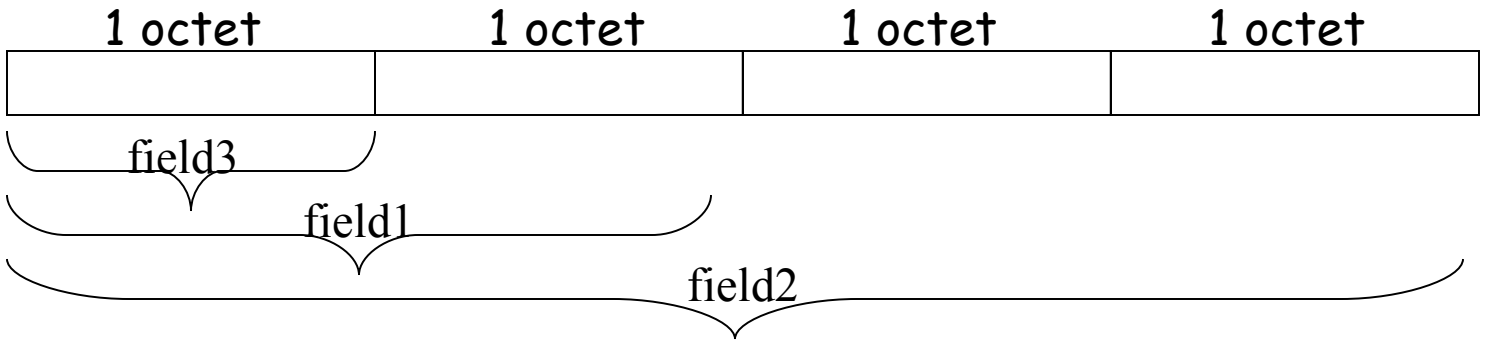
représentation différente d'une struct, donnée sur un exemple

```
typedef union s_test
{
      short int field1;   /* taille : 2 octets */
      long  int field2;   /* taille : 4 octets */
      char field3;        /* taille : 1 octet  */
} t_test;
```

Unions

La taille de cette union sera donc de 4 octets, on pourra accéder, selon le champ, a une partie ou à une autre de l'union

```
typedef union s_test
{
        short int field1;    /* taille : 2 octets */
        long  int field2;    /* taille : 4 octets */
        char field3;         /* taille : 1 octet  */
} t_test;
```



Tableaux de structures

Une structure est un type comme un autre, on peut regrouper des variables de ce type dans un tableau sans aucun problème !

Utilisation de la notation [].

Exemple : constitution d'un répertoire personnel, où l'on stocke, pour chaque contact :

le nom, le prénom, l'adresse, le numéro de téléphone.

Nom, prénom et adresse sont des chaînes de caractère;

le numéro de téléphone est un entier long non signé (on gérera le préfixe 0)

définissons donc le type :

Tableaux de structures

```
typedef struct s_pers
{
        char *nom;
        char *prenom;
        char *adresse;
        unsigned long int tel;
}       t_pers;
```

comment déclarer un tableau contenant des cases de type t_pers ?

```
t_pers Repertoire[50];
```

Repertoire est un tableau contenant jusqu'à 50 cases de type t_pers.

Pour éviter de se perdre dans ces écritures 'compliquées' : regarder le type des expressions employées.

Tableaux de structures

```
t_pers repertoire[50];
```

repertoire est un tableau contenant jusqu'à 50 cases de type t_pers.

On ne peut pas lui appliquer le '.' car ce n'est pas une structure

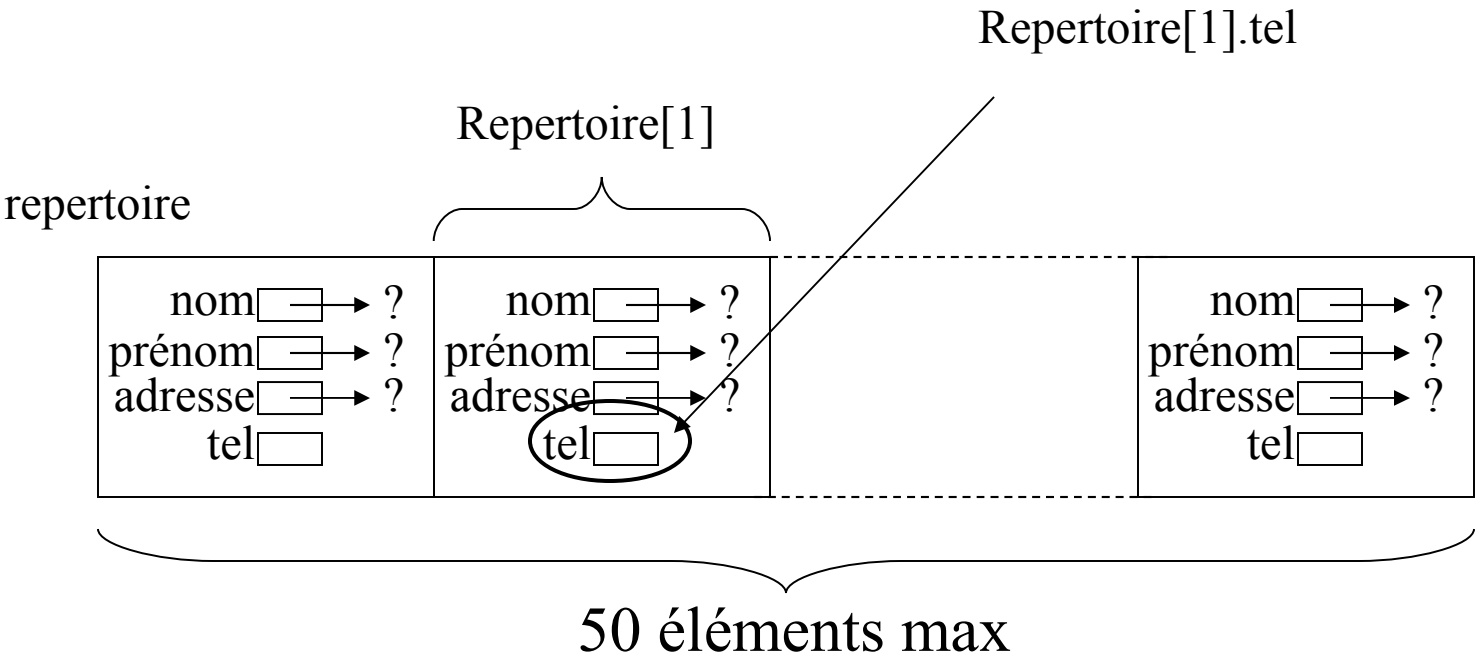
Soit une variable entière **num** servant d'indice pour le tableau :

repertoire[num] est une case du tableau. Or chaque case du tableau est de type t_pers. Donc repertoire[num] est de type t_pers : c'est une structure. On peut lui appliquer l'opérateur '.' pour accéder aux différents champs.

Repertoire[num].tel est le champ **tel** de la structure rangée à l'indice **num** dans le tableau **repertoire**. Cette expression est de type entier (unsigned long int).

Tableaux de structures

Illustration :



Tableaux de structures

Application : écrire une fonction qui collecte les données relatives à une personne et les range dans le tableau **repertoire**.