Initiation au C
cours n°7

Antoine Miné

École normale supérieure

5 avril 2007

Plan du cours

Les types de données structurées :

- `struct`,
- `union`.

Les *alias* de type : `typedef`.

Rappels sur les types C

Types scalaires :

- Types de base :
  - entiers : `int`, `unsigned`, `char`, `long`, etc.
  - flottants : `float`, `double`.
- Types pointeurs : `type *`.

Types composés :

- Types tableaux (homogènes) : `type []`.
- Types structures et unions (hétérogènes).

Les types `struct`

Notion de structure

Structure : ou “enregistrement”.

Permet de grouper plusieurs valeurs dans une seule variable.

Exemple : compte (Suisse) = n° banque + n° compte + solde.

Une structure est composée d'un nombre fixé de champs :

- nommés, (`banque`, `compte`, `solde`)
- typés. (`int` pour `banque` et `compte`, `float` pour `solde`)

Les champs peuvent être de type différent.

Une variable structurée peut être manipulée soit :

- champ par champ, (lecture, mise à jour)
- comme un tout. (initialisation, copie, passage à une fonction)

Déclarations de types structurés

Déclaration de type : obligatoire, on utilise le mot-clé `struct`.

Syntaxe

```
struct mastruct {
  type1   champ1 ;
    ⋮
  typeN   champN ;
} ;
```

Effet : déclare un nouveau type de structure

- de nom *mastruct*,
- de champs nommés *champ1* à *champN*,
- les champs ont pour type *type1* à *typeN*.

mastruct, *champ1* à *champN* doivent être des identificateurs.

Déclarations de types structurés

Exemple

```
struct compte {
  int    banque;
  int    compte;
  float  solde;
};
```

Sur un Intel 32 ou 64 bits,
un objet de type `compte` occupera (`sizeof`) 12 octets.

Déclaration de variables structurées

Le type associé à une structure est de la forme : `struct` *mastruct*.

Syntaxe

`struct` *mastruct variable*;

Effet : déclare une variable *variable* de type structure *mastruct* préalablement défini.

Exemple

```
struct compte cb, pel;
```

Déclare deux variable structurées de type `compte`.

Ne pas confondre : nom de type, de variable, de champ.

Accès aux champs

Opérateur point .

Syntaxe

variable.*champ*

Effet : référence le champ *champ* de la variable *variable*

- utilisable dans une expression,
  Exemple : `float fric = cb.solde + pel.solde;`
- modifiable (*lvalue*).
  Exemple : `cb.solde -= 200.25;`

Initialisation de structures

Initialisation à la déclaration : comme pour un tableau.

Syntaxe

```
struct mastruct variable = { expr1, ..., exprN } ;
```

- l'ordre des expressions est le même que celui des champs,
- les champs manquants sont initialisés à `0` ou `NULL`,
- on peut imbriquer les initialiseurs entre `{` et `}`.
  (structures et/ou tableaux imbriqués)

Exemple

```
struct compte cb = { 99, 1345, 0.1 };
```

Copies de structures

Copie : l'opérateur = peut être utilisé sur des variables structurées.

Effet : copie champ à champ.

Exemple

```
new_cb = cb ;
```

est équivalent à

```
new_cb.banque = cb.banque ;
new_cb.compte = cb.compte ;
new_cb.solde  = cb.solde ;
```

Initialisation par recopie : `struct compte visa = cb ;`

Ce n'était pas possible avec les tableaux !

Passage de structures par valeur

Appels de fonctions :

les variables structurées sont passées par valeur.

Exemple

```
void anniversaire(struct compte c)
{ c.solde += 10.; }

struct compte cb;
anniversaire(cb);
```

- une nouvelle variable c est créée,
- les champs de cb sont copiés dans ceux de c,
- une modification de c ne change pas cb !

Comportement très différent de celui des tableaux !

Retour de structures

Retour : une fonction peut retourner une structure.

Exemple

```
struct compte nouveau_compte(int banque)
{
  struct compte c = { banque, lrand48(), 0. };
  return c;
}
```

Applications :

- initialisation : `struct compte c = nouveau_compte(12);`
- copie : `cb = nouveau_compte(42);`

Là encore, un comportement très différent des tableaux.

Passage de structures par référence

Passage par référence : peut être simulé grâce aux pointeurs.

Exemple

```
void loyer(struct compte* c)
{ (*c).solde -= 999.9; }

struct compte cb;
loyer(&cb);
```

- il n'y a pas d'allocation ou de copie de structure,
  $\Longrightarrow$ coût faible en mémoire et en temps,
- `*c` et `cb` référencent le même objet en mémoire,
  $\Longrightarrow$ `(*c).solde -= 999.9;` modifie `cb.solde`.

L'opérateur ->

Attention à la priorité des opérateurs

`*x.y` signifie `*(x.y)` et pas `(*x).y`

Comme on a souvent besoin de la construction `(*x).y`, le C propose un opérateur spécial : `->`. (tiret, supérieur)

Syntaxe

`variable->champ`

- strictement équivalent à `(*variable).champ`.

Application : affichage d'une structure

Affichage : il faut afficher les champs un par un, à la main !

$\Longrightarrow$ on définit souvent une fonction auxiliaire.

Fonction d'affichage

```
void affiche_compte(const struct compte* c)
{
  printf("Compte\n");
  printf("------\n");
  printf("Banque : %i\n", c->banque);
  printf("Numéro : %i\n", c->numero);
  printf("SOLDE  : %f\n", c->solde);
}
```

- on opte pour un passage par référence pour éviter la copie,
- `*c` n'est pas modifié $\Rightarrow$ on l'indique par `const`.

Tableaux dans les structures

Des champs tableaux peuvent apparaître dans une structure :

Exemple

```
struct id {
  char nom[30];
  int  naissance[3];
};
```

Notes :

- une variable de type `struct id` occupe 42 octets,
- on peut imbriquer les initialiseurs :
  Ex. : `struct id u = { "Antoine", { 11, 10, 1977 }};`
- on peut accéder à un élément par un chemin d'accès :
  Exemple : `u.naissance[2]` : année de naissance.

Tableaux dans les structures

- L'affectation =, l'initialisation, le passage en argument ou en retour de fonction copient récursivement les champs.

Exemple

```
v = u;
```

est équivalent à

```
v.nom[0] = u.nom[0];
    ⋮
v.nom[29] = u.nom[29];
v.naissance[0] = u.naissance[0];
v.naissance[1] = u.naissance[1];
v.naissance[2] = u.naissance[2];
```

Structures imbriquées

Des champs structures peuvent apparaître dans une structure :

Exemple

```
struct fiche {
  struct id     ident;
  struct compte cb, visa;
};
```

Notes : (similaires aux tableaux dans les structures)

- on peut imbriquer les initialiseurs,
- les sous-structures sont copiées récursivement,
- on accède à un champ par un chemin d'accès :
  Exemple : `u.cb.numero` : n° de compte CB.

Exemples de structures imbriquées

Exemple

```
void affiche_compte(const struct compte* c);

void affiche(const struct fiche* f)
{
  printf( "fiche de %s\n", f->ident.nom );
  affiche_compte( &f->cb );
  affiche_compte( &f->visa );
}
```

Les structures permettent :

- d'organiser ses données de manière hiérarchique,
- de réutiliser des fonctions.

Tableaux de structures

On peut aussi faire des tableaux de structures. . .

Exemple

```
struct entree {
  chat mot[TAILLE_MOTS];
  int  nombre_ok, nombre_spam;
};

struct entree corpus[NB_MOTS];
```

Quizz :

- comment accéder au *i*-ème caractère du *j*-ème mot ?
- `corpus[i]` est-il passé par valeur ou référence ?
- `corpus` est-il passé par valeur ou référence ?

Tableaux de structures

On peut aussi faire des tableaux de structures. . .

Exemple

```
struct entree {
  chat mot[TAILLE_MOTS];
  int  nombre_ok, nombre_spam;
};

struct entree corpus[NB_MOTS];
```

Quizz :

- accès au i-ème caractère du j-ème mot : `corpus[j].mot[i]`,
- `corpus[i]` est passé par valeur et peut être copié,
- `corpus` est passé par référence et ne peut pas être copié.

Alias de type avec `typedef`

Alias de type avec typedef

Syntaxe

`typedef` *type monalias* ;

Effet : *monalias* devient un *alias* de *type*.

Exemples

```
typedef unsigned int uint;    uint i;
typedef int compte_t;         compte_t numero;
```

Applications : permet de rendre un programme

- plus concis, (`uint` plus court que `unsigned int`)
- paramétrique, par abstraction du type réellement utilisé.
  (on peut facilement changer le type des numéros de compte)

Application de typedef aux structures

On peut se servir de typedef pour éviter le mot-clé struct.

Exemple

```
struct compte { ... };
typedef struct compte compte_t;

compte_t c = { ... };
void affiche_compte(const compte_t* c);
```

Note : on peut utiliser un *alias* dans un *alias* de type.

Exemple

```
typedef const compte_t* compte_constptr;
void affiche_compte(compte_constptr c);
```

typedef alternatifs

Exemple

```
struct compte { ... };
typedef struct compte compte;
```

On donne le même nom au type structuré et à l'*alias*.

Il n'y a pas d'ambiguïté entre `struct compte` et `compte`.

typedef alternatifs

Exemple

```
typedef struct compte { ... } compte;
```

Définit à la fois `struct compte` et `compte`.

On peut utiliser les deux types.

typedef alternatifs

Exemple

```
typedef struct { ... } compte;
```

Définit seulement `compte`, pas `struct compte`.

Le type structuré est anonyme.

typedef dans la bibliothèque C

On a déjà vu des exemples de `typedef`. . . dans la bibliothèque C !

- `size_t`

  utilisé, par exemple, comme type de retour de `strlen`,
  défini par la bibliothèque C comme un type entier,
  sur mon Linux 64-bit : `typedef unsigned long size_t`.

- `FILE`

  type des flux ouverts par `fopen`,
  sur mon Linux 64-bit :

```
typedef struct _IO_FILE FILE;
struct _IO_FILE {
  int _flags;
  char* _IO_read_ptr;
  ...
};
```

Les types `union`

Notion d'union

Union C = alternative.

Exemple : nombre = entier ou flottant.

Une union est composée de champs :

- nommés,
- typés.

Les champs partagent le même emplacement en mémoire :

- on ne peut utiliser qu'un champ à la fois,
- l'union occupe la place nécessaire au plus gros champ.

Déclarations de types unions

Déclaration de type :

similaire à une structure, mais avec le mot-clé `union`.

Syntaxe

```
union monunion {
   type1   champ1 ;
     ⋮
   typeN   champN ;
} ;
```

Effet : déclare un nouveau type d'union.

Exemple d'union

Exemple

```
union nombre {
  int     entier;
  double  flottant;
};
```

Occupation en mémoire : 8 octets.

Variables unions

Les unions se déclarent et s'utilisent comme des structures. . .

Exemple

```
union nombre nb;
nb.flottant = 12;
nb.flottant *= 2;
```

. . . avec une différence importante :

Attention

Écrire dans un champ rend invalide les autres champs.

Exemple d'utilisation invalide

On ne doit pas écrire dans un champ puis lire depuis un autre !

Exemple faux

```
union nombre nb;

nb.flottant = 12.;  /* OK */
nb.entier   = 42;   /* OK */
x = nb.flottant;    /* erreur! */
```

$\Longrightarrow$ il faut se souvenir du champ actif !

Utilisation pratique des unions

En pratique, on se sert d'un discriminant pour se souvenir du champ actif.

Exemple

```
typedef union  { int ent; double flot; } val_nb;
typedef struct { int type; val_nb val; } nombre;
nombre nb;
```

Mode d'emploi : le champ entier `type` sert de discriminant

- si `nb.type==0`, alors on utilise `nb.val.ent`,
- si `nb.type==1`, alors on utilise `nb.val.flot`.

Unions de structures

Autre utilisation courante : champs structures.

Exemple

```
typedef struct { int type; int x; ... } truc;
typedef struct { int type; int y; ... } machin;
typedef union {
  int     type;
  truc    truc;
  machin  machin;
} machintruc;
machintruc b;
```

- on se sert de `b.type` pour indiquer si on a affaire à un `truc` ou un `machin`,
- `b.type`, `b.truc.type` et `b.machin.type` sont interchangeables et représentent le même objet mémoire.