Introduction au C/C++

Cours 4

Rémi Watrigant
(fortement inspiré du cours de V. Boudet, P. Giorgi et M. Joab de l'Université Montpellier 2)

Université de Nîmes

2013-2014

Plan

1 Structures de données

A quoi sert une structure de données

Les types scalaires :

- type de base : entier (`int, double,...`)
- type pointeur : `type *`

Les types composés :

- données homogènes (les tableaux) : `type[]`

A quoi sert une structure de données

Les types scalaires :

- type de base : entier (`int, double,...`)
- type pointeur : `type *`

Les types composés :

- données homogènes (les tableaux) : `type[]`
- données hétérogènes : les structures

Les structures de données

- étendent les types du langage pour une problématique donnée :
  nombre complexe, matrices, compte bancaire, voiture, ...

- regroupent des données dans une seule variable :
  (hétérogène) un compte bancaire= n° banque,n° compte, solde
  (homogène) un nbr complexe= partie imaginaire, partie réelle

Les structures de données

- étendent les types du langage pour une problématique donnée :
  nombre complexe, matrices, compte bancaire, voiture, ...

- regroupent des données dans une seule variable :
  (hétérogène) un compte bancaire= n° banque,n° compte, solde
  (homogène) un nbr complexe= partie imaginaire, partie réelle

Remarque

Cela facilite la manipulation des données et la structuration des programmes

Spécifications

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

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

Une variable structurée peut être manipulée :

- champ par champ (lecture, mise à jour)
- globalement (initialisation, copie, paramètre fonction)

Déclaration d'une structure en C

Syntaxe

```
struct NomStruct{
    type1 champ1;
    type2 champ2;
    ...
    typeN champN;
};
```

Cela déclare un nouveau type de données

- de nom `struct NomStruct`
- composé des champs `champ1, ..., champN`
- ayant respectivement pour type `type1,...,typeN`

Déclaration d'une structure en C

Syntaxe

```
struct NomStruct{
    type1 champ1;
    type2 champ2;
    ...
    typeN champN;
};
```

Attention

- `NomStruct` est un identificateur pas encore utilisé (ex. $\neq$ `int`).
- `type1,...,typeN` sont des types connus dont on connaît la taille mémoire (ex. $\neq$ `struct NomStruct`).

Déclaration d'une structure en C : Exemple

```
1 struct compteB {
2   int    banque;
3   int    compte;
4   float  solde
5 };
```

Attention

Le ; est obligatoire après la déclaration de la structure ! ! !

Déclaration d'une variable structurée

Syntaxe

```
struct NomStruct var;
```

`var` est une variable de type structure `NomStruct`.

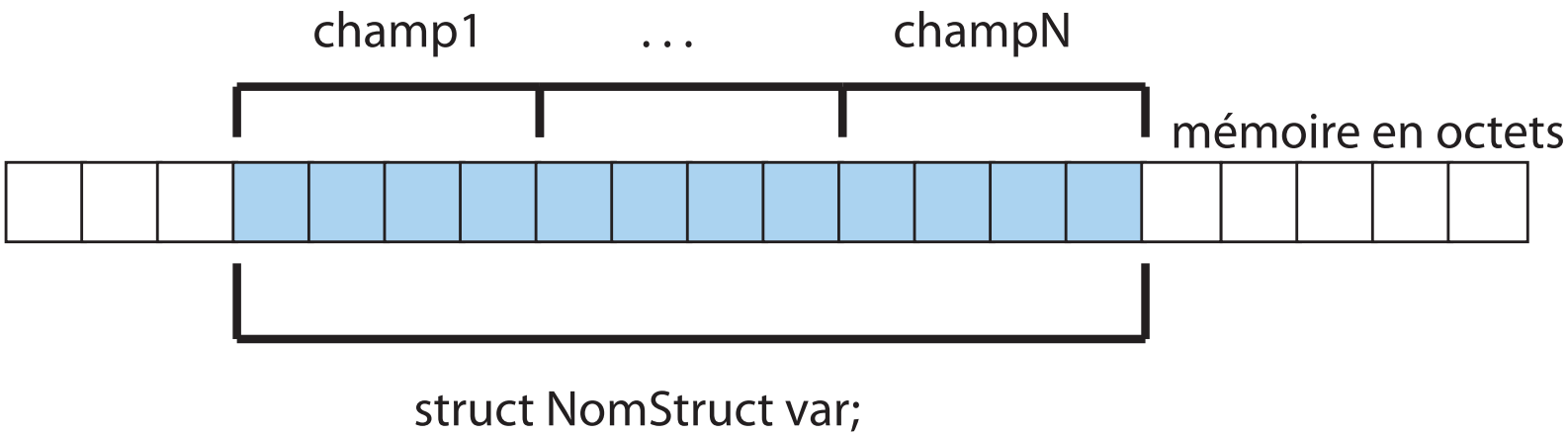
Exemple

```
struct compteB CB;
```

Une structure en mémoire ? ? ?

Syntaxe

```
struct NomStruct var;
```



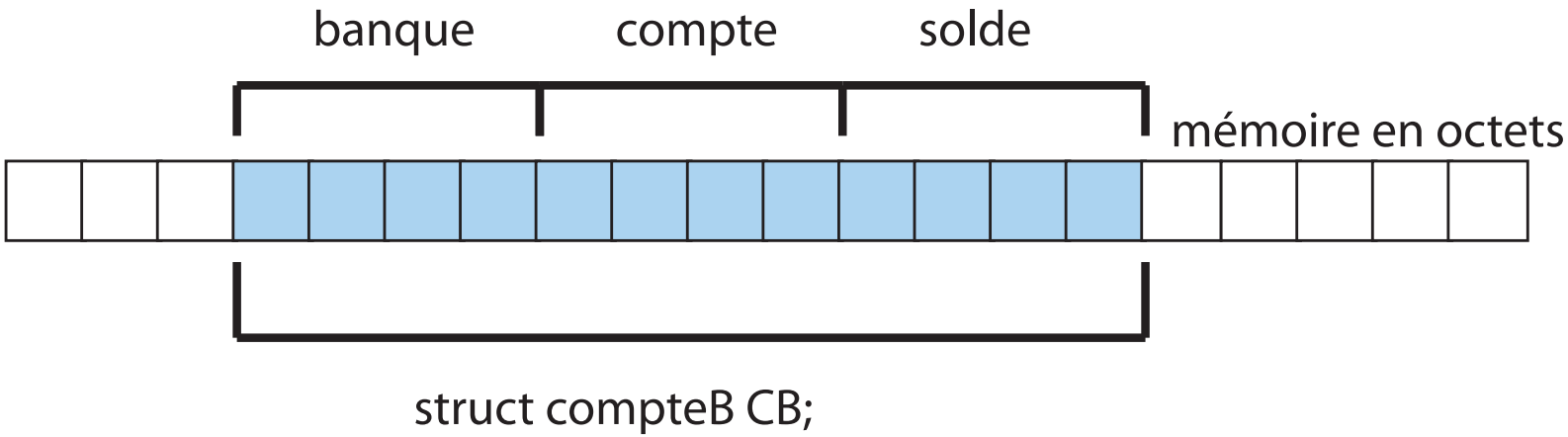
Remarque

Chaque donnée dans la variable est identifiée par l'identificateur du champ correspondant.

Une structure en mémoire ? ? ?

Exemple

```
struct compteB CB;
```



Remarque

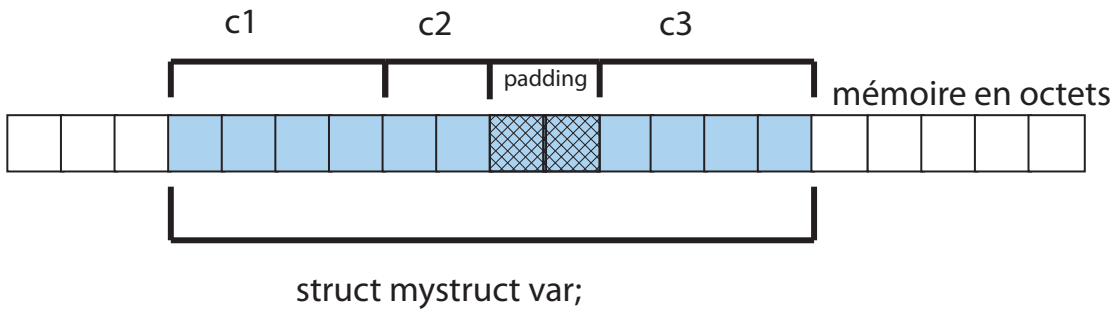
Les champs sont ordonnés en mémoire dans l'ordre de leur déclaration dans la structure.

Une structure en mémoire ? ? ?

Attention

Les champs des structures sont alignés suivant l'alignement des types de base en mémoire (32 ou 64 bits).

```
struct mystruct {
   int c1;
   short int c2;
   float c3;
};
```



- on peut obtenir l'occupation mémoire en utilisation `sizeof`
  exemple : `sizeof(struct mystruct)` ou `sizeof(var)`
- les règles de padding sont complexes (au delà de ce cours)

Accès aux champs d'une donnée structurée

On utilise la notation pointée :

Syntaxe

```
var.champ
```

Cela permet de récupérer la partie `champ` de la variable `var`.

Accès aux champs d'une donnée structurée

On utilise la notation pointée :

Syntaxe

`var.champ`

Cela permet de récupérer la partie `champ` de la variable `var`.

Remarque

Cela définit un identificateur sur la donnée concernée

- possédant une adresse : `&(var.champ)`
- accessible en lecture/écriture : `var.champ=...`

Accès aux champs d'une donnée structurée

```
1  #include <stdio.h>
2  struct compteB {
3     int banque;
4     int compte;
5     float solde;
6  };
7
8  int main(){
9     struct compteB CB;
10    CB.solde=1000;
11    printf("le solde de CB est de %d\n",CB.solde);
12    return 0;
13 }
```

Initialisation d'une structure

Initialisation lors de la déclaration (*comme les tableaux statiques*)

Syntaxe

struct NomStruct var= {exp1,...,expN} ;

équivalent à :

- `var.champ1=exp1;`
- ...
- `var.champN=expN;`

Initialisation d'une structure

Initialisation lors de la déclaration (*comme les tableaux statiques*)

Syntaxe

struct NomStruct var= {exp1,...,expN} ;

Remarque

- l'ordre des expressions est identique au champs
- les champs manquants sont initialisé à 0 ou `NULL`
- on peut imbriquer les initialisations `{..}` (e.g. struct, tableau)

Initialisation d'une structure : Exemple

```c
#include <stdio.h>
struct compteB {
   int banque;
   int compte;
   float solde;
};

int main(){
   struct compteB CB={1234, 04834567, 1760.50};

   printf("le compte %d ",CB.compte);
   printf("de la banque %d "CB.banque);
   print("a un solde de %d\n",CB.solde);
   return 0;
}
```

Copie des structures

- A l'inverse des tableaux statiques la copie de structure fonctionne via l'opérateur $=$

Remarque

la copie se fait champ par champ

Exemple

```
struct compteB cb1, cb2;
cb1=cb2;
```

est équivalent à

```
cb1.banque=cb2.banque;
cb1.compte=cb2.compte;
cb1.solde=cb2.solde
```

Copie des structures

Remarque

L'initialisation par recopie fonctionne également

Exemple

```
struct compteB cb1;
...
struct compteB cb2=cb1;
```

Variable structuré : généralités

- Bien que comprenant plusieurs données, une variable structurée est considérée comme une seule donné au sens des variables.
  - copie/initialisation possible
  - paramètre/retour de fonctions

Variable structuré : généralités

- Bien que comprenant plusieurs données, une variable structurée est considérée comme une seule donné au sens des variables.
  - copie/initialisation possible
  - paramètre/retour de fonctions

- A l'inverse des variables tableaux qui correspondent à une adresse mémoire référençant les données.

Plan

1 Structures de données
- Structures et fonctions
- Structures et tableaux
- Structure et pointeur
- Utilisation des alias de type
- Structure et allocation dynamique

Passage des structures comme paramètre

Comme tous les paramètres de fonction, les variables structurées sont passées par copie/valeur.

Syntaxe

```
type_r mafonction(struct nomStruct var,...) {...}
```

- `var` est un paramètre de type `struct nomStruct`
- lors de l'appel, une copie du paramètre structuré sera utilisé

Passage des structures comme paramètre

Exemple :

```
1 void interet(struct compteB cb, float taux){
2   c.solde*=(1.+taux);
3 }
4 int main(){
5   compteB CB;
6   interet(CB, 0.025);//NE MODIFIE PAS CB
7   return 0;
8 }
```

Attention

c'est une copie de `CB` qui est manipulée par la fonction `interet(...)`

Retour d'une structure

Comme toutes les variables scalaires, une structure peut être retournée par une fonction.

Syntaxe

```
struct nomStruct mafonction(...) {
...
return exp;
}
```

- `exp` doit être une variable ou une constante de type `struct nomStruct`
- c'est une copie de `exp` qui est renvoyée

Retour d'une structure : exemple

```
1 struct compteB creation_compte(int banq){
2   strut compteB c={banq, 0, 0.);
3     return c;
4 }
```

- init. : `struc compteB c = creation_compte(1234);`
- copie : `cb = creation_compte(4321);`

Attention

Dans ce cas d'utilisation, le nombre de copie de la structure est de deux $\rightarrow$ (coût mémoire potentiellement prohibitif)

Passage de structures par adresse

On utilise un pointeur sur la structure

Syntaxe

```
type_r mafonction(struct nomStruct *var,...) {...}
```

- `var` contient l'adresse mémoire d'une variable de type `struct nomStruct`
- aucune copie mémoire lors de l'appel de la fonction

Passage de structures par adresse : Exemple

```
1 void interet(struct compteB *cb, float taux){
2   (*cb).solde*=(1.+taux);
3 }
4 int main(){
5   compteB CB;
6   interet(&CB, 0.025);//MODIFIE BIEN CB
7   return 0;
8 }
```

Attention

La priorité des opérateurs est très importante :
`*var.champ` correspond à `*(var.champ)` et non à `(*var).champ`

Simplification d'accès aux champs via pointeur

Comme on a souvent besoin de la construction `(*var).champ` le langage C propose un opérateur spécial : `->`

Syntaxe

```
struct nomStruct *var= ...;
var->champ=...;
```

- strictement équivalent à `(*var).champ=...;`
- les champs sont accessibles en lecture et en écriture (bien sûr)

Exemple

L'affichage d'une structure nécessite souvent une fonction particulière

```
1  void affiche(struct compteB *CB){
2     printf("le compte %d ",CB->compte);
3     printf("de la banque %d "CB->banque);
4     printf("a un solde de %d\n",CB->solde);
5  }
6  int main(){
7     struct compteB cb={1234,0482671234,1270.50};
8     affiche(&cb); passage par adresse -> evite la copie
9     return 0;
10 }
```

Plan

1 Structures de données
- Structures et fonctions
- Structures et tableaux
- Structure et pointeur
- Utilisation des alias de type
- Structure et allocation dynamique

Champ de type tableau dans une structure

Les champs d'une structure peuvent être des tableaux statiques

Syntaxe

```
struct nomStruct{
    type1 champ1[10];
    type2 champ2;
};
```

- `champ1` contient un tableau de 10 `type1`

Attention

La structure ne contient pas que l'adresse du 1er élément du tableau mais bien l'ensemble des éléments du tableau.

Exemple

```
1 struct Etudiant {
2   char nom[32];
3   int  numero;
4   int naissance[3];
5 };
```

- tous les éléments du tableau sont intégrés dans la structure
  taille de la structure `Etudiant` : 48 octets = 32 char + 4 int
- ils sont accessibles par l'accès au champ puis au tableau

Exemple : `struct Etudiant etud; etud.naissance[2]=1984;`

Exemple

On peut imbriquer les initialisations lors de la construction d'une donnée structurée :

```
1 struct Etudiant {
2   char nom[32];
3   int  numero;
4   int naissance[3];
5 };
6 ...
7 struct Etudiant e={"Beri",2010003111,{12,4,1987}};
8 ...
```

Tableau dans les structures

Remarque

L'affectation, l'initialisation, le passage en argument et le retour de fonction d'une structure copie récursivement chacun des champs ...

Tous les tableaux statiques dans les structures seront donc copié élément par élément lors

- de l'affectation de la structure
- du passage en argument d'une fonction
- du retour de la structure par une fonction

Tableau dans les structures : Exemple

```
1  void anniversaire(struct Etudiant e){
2     printf("%s est ne le ", e.nom);
3     printf("%d/",e.naissance[0]);
4     printf("%d/",e.naissance[1]);
5     printf("%d/n",e.naissance[2]);
6  }
7
8  int main(){
9     struct Etudiant e1,e2={"Beri",2010003111,{12,4,1987}};
10    e1=e2;   // copie des tableaux
11    anniversaire(e2);   // copie des tableaux
12    return 0;
13 }
```

Champs de type structure dans une structure

Les champs d'une structure peuvent être des structures

Syntaxe

```
struct nomStruct{
    struct otherStruct champ1;
    type2 champ2;
};
```

- `champ1` est un type structurée de type `otherStruct`
- le comportement est similaire aux tableaux statiques (copie)

Champs de type structure dans une structure

Les champs d'une structure peuvent être des structures

Syntaxe

```
struct nomStruct{
    struct otherStruct champ1;
    type2 champ2;
};
```

- `champ1` est un type structurée de type `otherStruct`
- le comportement est similaire aux tableaux statiques (copie)

Attention

La taille de la structure imbriquée doit être connue ...

$\rightarrow$ les structures récursives sont impossibles

Exemple

```
1 struct Date {
2   int    jj ,mm, aaaa;
3 };
4 struct Etudiant {
5   char nom[32];
6   int   numero;
7   struct Date naissance;
8 };
```

- tous les champs des structures sont intégrés
  taille de la structure `Etudiant` : 48 octets = 32 char + 4 int
- ils sont accessibles par les appels imbriqués des champs

Exemple : `struct Etudiant e; e.naissance.aaaa=1984;`

Exemple

On peut imbriquer les initialisations lors de la construction d'une structure imbriquée :

```
1  struct Date {
2    int    jj ,mm, aaaa;
3  };
4
5  struct Etudiant {
6    char nom[32];
7    int   numero;
8    struct Date naissance;
9  };
10 ...
11 struct Etudiant e={"Beri",2010003111,{12,4,1987}};
```

Structure dans les structures

Remarque

L'affectation, l'initialisation, le passage en argument et le retour de fonction d'une structure copie récursivement chacun des champs ...

Comme avec les tableaux statiques, les structures imbriquées seront donc copié champs par champs lors

- de l'affectation de la structure
- du passage en argument d'une fonction
- du retour de la structure par une fonction

Structure dans les structures : Exemple

```
1  void anniversaire(struct Etudiant e){
2     printf("%s est ne le ", e.nom);
3     printf("%d/", e.naissance.jj);
4     printf("%d/", e.naissance.mm);
5     printf("%d/n",e.naissance.aaaa);
6  }
7
8  int main(){
9     struct Etudiant e1,e2={"Beri",2010003111,{12,4,1987}};
10    e1=e2; // copie tableau et structure
11    anniversaire(e2); // copie tableau et structure
12    return 0;
13 }
```

Tableau statique de structure

Les structures étant un type de données, on peut les mettre dans un tableau

Syntaxe

```
struct nomStruct var[N];
```

- `var` est un tableau contenant `N` données structurées
- `N` doit être une constante connue

Attention

`var` reste un tableau et contient l'adresse mémoire ou se trouve les `N` données structurées.

Exemple

On peut imbriquer les initialisations lors de la construction d'un tableau statique de structure :

```
1 struct Date {
2   int    jj ,mm, aaaa;
3 };
4 ...
5 struct Date D[3]={{2,1,1923},{23,12,2004},{13,01,2011}};
```

- on ne peut pas copier le tableau par l'opérateur d'affectation : `Date T[3]=D;` (INTERDIT)

Exemple

On peut imbriquer les initialisations lors de la construction d'un tableau statique de structure :

```
1 struct Date {
2   int    jj ,mm, aaaa;
3 };
4 ...
5 struct Date D[3]={{2,1,1923},{23,12,2004},{13,01,2011}};
```

- on ne peut pas copier le tableau par l'opérateur d'affectation : `Date T[3]=D;` (INTERDIT)
- lors de passage du tableau en paramètre d'une fonction, les données structurées ne sont pas copiées

Plan

1 Structures de données
- Structures et fonctions
- Structures et tableaux
- Structure et pointeur
- Utilisation des alias de type
- Structure et allocation dynamique

Champs pointeur dans une structure

Les champs pointeurs dans une structure permettent de stocker l'adresse de n'importe quelle donnée (scalaire, tableau, structure)

Syntaxe

```
struct nomStruct {
   type1 *champ1;
   type2 champ2;
};
```

- `champ1` contient l'adresse d'une donnée de type `type1`
- `type1` est n'importe quel type connu (`struct nomStruct` compris)

Exemple

```
1  struct Etudiant {
2    char *nom;
3    struct Date anniversaire;
4    struct Etudiant *binome;
5  };
6  ...
7  struct Etudiant e1={"berio",{1.3.1987}}
8  struct Etudiant e2={"lefort",{12,1,1988}};
9  e1.binome=&e2;
10 e2.binome=&e1;
```

Plan

1 Structures de données
- Structures et fonctions
- Structures et tableaux
- Structure et pointeur
- Utilisation des alias de type
- Structure et allocation dynamique

Alias de type avec typedef

Les alias de type permette de renommer un type C

Syntaxe

```
typedef type nomalias;
```

- `nomalias var;` permet de déclarer une variable de type `type`
- permet de rendre un programme plus concis (ex. `uint`)
- utile avec les structures pour supprimer le mot clé `struct` dans le type de données

Utilisation de typedef dans les structures

On peut utiliser le typedef de deux manières avec les structures :

- renommer une structure existante

Syntaxe

```
struct nomStruct {... };
typedef struct nomStruct nomStruct_t
nomStruc_t var;
```

- renommer une structure anonyme lors de sa déclaration

Syntaxe

```
typedef struct {... } nomStruct_t;
nomStruc_t var;
```

Exemple 2

Renommage d'une structure définie :

```
1 struct Etudiant {
2   char *nom;
3   struct Date anniversaire;
4 };
5 typedef struct Etudiant Etudiant;
6 ...
7 Etudiant e1={"berio",{1.3.1987}}
```

Exemple 2

Renommage d'une structure anonyme :

```
1 typedef struct {
2   char *nom;
3   struct Date anniversaire;
4 } Etudiant;
5 ...
6 Etudiant e1={"berio",{1.3.1987}}
```

Plan

1 Structures de données
- Structures et fonctions
- Structures et tableaux
- Structure et pointeur
- Utilisation des alias de type
- Structure et allocation dynamique

Allocation dynamique de structure

Comme pour les types de base (int, double, ...) il est possible de faire de l'allocation dynamique de données structurées.

Remarque

Il suffit d'utiliser l'allocation avec `malloc` et la libération avec `free`.

Allocation dynamique de structure

Comme pour les types de base (int, double, ...) il est possible de faire de l'allocation dynamique de données structurées.

Remarque

Il suffit d'utiliser l'allocation avec `malloc` et la libération avec `free`.

Exemple

```
typedef struct {int jj,mm,aaaa;} Date;
Date *T= malloc(3*sizeof(Date);
...
free(T);
```

On utilise la fonction `sizeof` pour récupérer la taille de la `struct`