Algorithmique et Structures de données

Thomas Lavergne & Florent Hivert

Mél : thomas.lavergne@lisn.upsaclay.fr

10 mars 2021

Structures

Quand plusieurs variables modélisent le même objet du monde réel, il est très pratique de les regrouper.

Retenir

Selon les langages de programmation, on appelle ces types ***structure, produit (product), enregistrement (record)****.*

Les ***composants*** *d'une structure s'appellent des* ***champs*** *et chaque champ possède un* ***nom et un type*** *(simple ou composé).*

Il peut y avoir autant de champs que l'on veut, et les types des différents champs peuvent être égaux ou différents.

Structures

Syntaxe

Définition d'un type structure :

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

Après une telle définition chaque variable de type `NomDuType` aura N champs, chacun du type indiqué.

Exemples de définition de type structure

Durée en heures, minutes et secondes :

```
struct dureeHMS {
  int heure;
  int minute;
  float seconde;
};
```

Nombre complexe :

```
  struct complexe {
    float part_re, part_im;
  };
```

Un premier LEGO®

Retenir

Un champ d'une structure peut lui aussi être une structure !

Type «date» composé d'un jour, un mois, une année

```
struct date {
  int jour, mois, annee;
};
```

Type «assuré social»

```
struct AssureSocial {
  string nom, prenom, Nsecu;
  date date_naissance;
};
```

Champs d'une structure...

Syntaxe

*La **notation pointée** permet d'accéder (en lecture et en écriture) aux **champs d'une struct** :*

```
nom_de_la_variable.nom_du_champ
```

Par exemple, pour leur affecter une valeur :

```
date aujourdhui;
aujourdhui.mois = 2;
aujourdhui.annee = 2020;
```

Structures imbriquées :

```
AssureSocial a;
if (a.date_naissance.annee >= 2003)
  cout << a.prenom << " " << a.nom << " est mineur";
```

Manipulations des structures

Retenir

À part l'extraction des champs, les ***seules manipulation possibles*** *sur une variable de type structurée sont*

- *l'****affectation à une structure du même type*** *(tous les champs sont recopiés)*
- *le* ***calcul de l'adresse*** *(adresse du premier champ)*

En particulier, on ne peut pas lire ou afficher une structure directement.

Manipulations des structures

Retenir

À part l'extraction des champs, les ***seules manipulation possibles*** *sur une variable de type structurée sont*

- *l'****affectation à une structure du même type*** *(tous les champs sont recopiés)*
- *le* ***calcul de l'adresse*** *(adresse du premier champ)*

En particulier, on ne peut pas lire ou afficher une structure directement.

Exemple : Affichage d'une date

date.cpp

```
 1  #include<iostream>
 2  #include<iomanip>
 3  using namespace std;
 4
 5  struct date {
 6    int jour, mois, annee;
 7  };
 8
 9  /** Affiche une date sous le format jj/mm/aaaa
10  * @param[in] d :  la date
11  **/
12  void afficheDate(date d) {
13    cout << setfill('0') << setw(2) << d.jour << "/"
14         << setfill('0') << setw(2) << d.mois << "/" << d.annee;
15  }
```

Exemple de saisie d'une date

date.cpp

```
1  /** Demande une date à l'utilisateur
2  *   Si la date n'est pas correcte, une nouvelle date est demandée
3  * @return une date
4  **/
5  date lireDate() {
6    date res;
7    bool erreur;
8    do {
9      cout << "jour ? "; cin >> res.jour;
10     cout << "mois ? "; cin >> res.mois;
11     cout << "annee ? "; cin >> res.annee;
12     erreur = not estCorrecteDate(res);
13     if (erreur) cout << "Date incorrecte !" << endl;
14   } while (erreur);
15   return res;
16 }
```

date.cpp

```
1   /** Teste si une année est bissextile
2   * @param[in] annee : un entier
3   * @return le booléen correspondant au test
4   **/
5   bool estBissextile(int annee) {
6     return (annee % 4 == 0 and annee % 100 != 0) or (annee % 400 == 0);
7   }
8
9   /** Le nombre de jours qu'il y a dans un mois
10  * @param[in] annee : un entier
11  * @param[in] mois : un entier entre 1 et 12
12  * @return le nombre de jours du mois
13  **/
14  int nbJourMois(int mois, int annee) {
15    switch (mois) {
16    case 1 : case 3 : case 5 :
17    case 7 : case 8 : case 10 : case 12 : return 31;
18    case 4 : case 6 : case 9 : case 11 : return 30;
19    case 2 :
20      if (estBissextile(annee)) return 29;
21      else                      return 28;
22    default return -1;
23    }
24  }
25
26  /** Teste si une date est correcte
27  * @param[in] d : une date
28  * @return le booléen correspondant au test
29  **/
30  bool estCorrecteDate(date d) {
31    if (d.mois<= 0 or d.mois > 12) return false;
32    if (d.jour<= 0) return false;
33    return d.jour<= nbJourMois(d.mois, d.annee);
34  }
```

Deux manières de renvoyer une date au programme

date.cpp

```
1  /** Le bug de l'an 2000
2  * @return le premier janvier 2000
3  **/
4  date bugday() {
5    return {1, 1, 2000};
6  }
7
8  /** Le bug de l'an 2000
9  * @param[out] d recoit le bug day
10 **/
11 void bugdayRef(date &d) {
12   d = {1, 1, 2000};
13 }
```

Manipulation d'une date

date.cpp

```
1  /** Augmente la date d'un jour
2  *  @param[in/out] d : une date
3  **/
4  void avanceDate(date &d) {
5    d.jour++;
6    if (d.jour > nbJourMois(d.mois, d.annee)) {
7      d.jour = 1;
8      d.mois++;
9      if (d.mois == 13) {
10       d.mois = 1;
11       d.annee++;
12     }
13   }
14 }
```

Stockage en mémoire d'une structure

Retenir

Le compilateur réserve les emplacements nécessaires pour stocker les différents champs de la structure. Ils correspondent à des emplacements mémoires consécutifs différents.

a	nom		"John"
	prenom		"Doe"
	Nsecu		"1 74 34 234 123"
	date_naissance	annee	1974
		mois	11
		jour	3

Note : dans la réalité, le type `string` est lui aussi composé et comporte plusieurs sous-variables (c'est un objet).

Initialisation d'une `struct`

Syntaxe

En C++, on peut initialiser une `struct` *avec les accolades :*

```
{val1, val2, ..., valN}
```

init-struct.cpp

```
1  struct date {
2    int jour, mois, annee;
3  };
4  date d = {3, 11, 1974};
5
6  // fonction qui retourne le 1er janvier 2000
7  date bugday() {
8    return {1, 1, 2000};
9  }
```

Passage d'une structure en paramètre par valeur

Retenir

Tous les champs *de la structure paramètre réel doivent être recopiés dans le paramètre formel :*

Avant l'appel ligne 6

```
1  void afficheDate(date d) {
2    cout << ...
3  }
4  [...]
5    date rdv;
6    rdv = {12, 3, 2019};
7    afficheDate(rdv);
```

Pile :

	14	?
	13	?
	12	?
	11	?

...

rdv.annee	6	**2019**
rdv.mois	5	**3**
rdv.jour	4	**12**

...

Passage d'une structure en paramètre par valeur

Retenir

Tous les champs *de la structure paramètre réel doivent être recopiés dans le paramètre formel :*

Appel de `afficheDate` ligne 7

```
1  void afficheDate(date d) {
2    cout << ...
3  }
4  [...]
5    date rdv;
6    rdv = {12, 3, 2019};
7    afficheDate(rdv);
```

Pile :

afficheDate	14	ADM
d.annee	13	2019
d.mois	12	3
d.jour	11	12

...

6	2019
5	3
4	12

...

Passage d'une structure en paramètre par référence

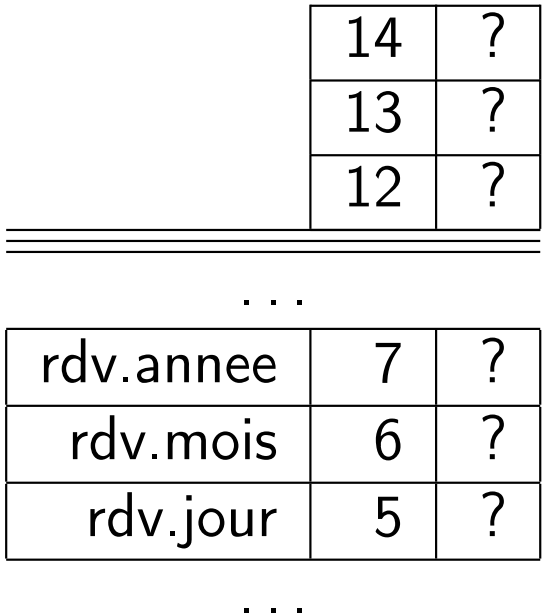
Retenir

*Comme on stocke les champs **toujours dans le même ordre**, il suffit de passer l'**adresse du premier champ**. Les autres seront dans les cases mémoires suivantes.*

Avant l'appel ligne 5

```
1  void lireDate(date &d) {
2    [...]
3    cin >> d.annee;
4    [...]
5  }
6  [...]
7    date rdv;
8    lireDate(rdv);
```

Pile :



Passage d'une structure en paramètre par référence

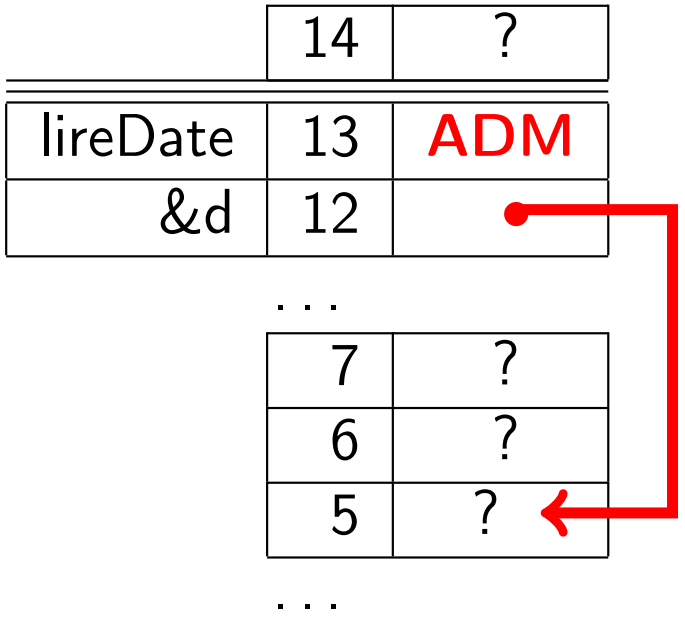
Retenir

*Comme on stocke les champs **toujours dans le même ordre**, il suffit de passer l'**adresse du premier champ**. Les autres seront dans les cases mémoires suivantes.*

Appel de `lireDate` ligne 6

```
1  void lireDate(date &d) {
2    [...]
3    cin >> d.annee;
4    [...]
5  }
6  [...]
7    date rdv;
8    lireDate(rdv);
```

Pile :



Passage d'une structure en paramètre par référence

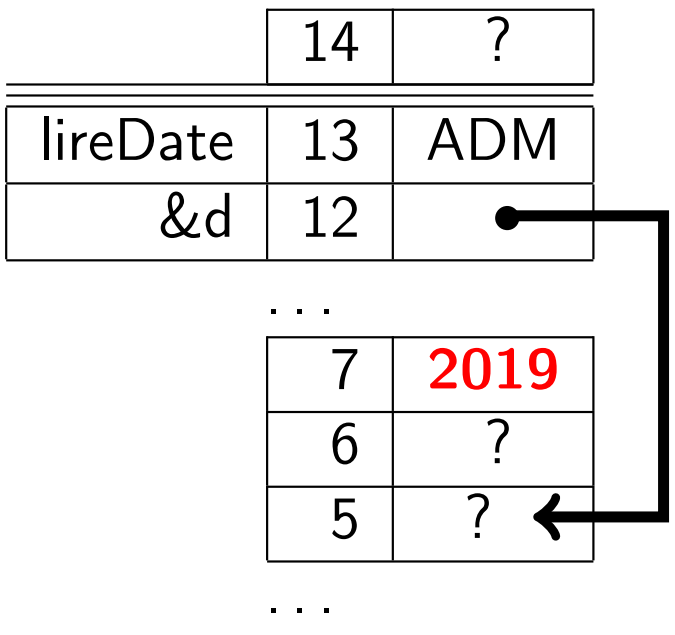
Retenir

*Comme on stocke les champs **toujours dans le même ordre**, il suffit de passer l'**adresse du premier champ**. Les autres seront dans les cases mémoires suivantes.*

Affectation de `d.jour` ligne 1

```
1  void lireDate(date &d) {
2    [...]
3    cin >> d.annee; // -> 2019
4    [...]
5  }
6  [...]
7    date rdv;
8    lireDate(rdv);
```

Pile :



Une structure comme résultat

L'affectation des structures est possible ! Une fonction peut donc retourner une structure.

Retenir

*La case mémoire `return` du tableau d'activation est en fait composée de **plusieurs cases mémoires**.*

Mais, cela peut forcer (pas toujours...) le compilateur à faire une copie de la structure qui peut être coûteuse.

Retenir

*Il peut être plus efficace de **passer le résultat dans un paramètre par référence** (mais attention à la lisibilité du code)...*

Plutôt que : `AssureSocial LitAssureSocial();`

Écrire : `void LitAssureSocial(AssureSocial &A);`

Une structure comme résultat

L'affectation des structures est possible ! Une fonction peut donc retourner une structure.

Retenir

*La case mémoire `return` du tableau d'activation est en fait composée de **plusieurs cases mémoires**.*

Mais, cela peut forcer (pas toujours...) le compilateur à faire une copie de la structure qui peut être coûteuse.

Retenir

*Il peut être plus efficace de **passer le résultat dans un paramètre par référence** (mais attention à la lisibilité du code)...*

Plutôt que : `AssureSocial LitAssureSocial();`

Écrire : `void LitAssureSocial(AssureSocial &A);`

Les tableaux

Le **type tableau** est un type prédéfini qui permet de créer un grand nombre de variables du même type à des emplacements consécutifs de la mémoire.

Syntaxe

Pour définir une variable de type tableau, il faut préciser sa taille ainsi que le type de ses éléments :

```
type_des_elements nom_du_tableau[taille];
```

On peut aussi donner un nom au type tableau :

def-tableau.cpp

```
1 const int taille=10;
2 using tab = int[taille];
3 tab t;
```

Tableau, C et C++

Retenir

Les tableaux définis ici sont les ***tableaux bas niveau*** *hérités du langage C.*
En C++ moderne, on utilise plutôt les classes `array` *pour les tableaux de taille fixe et* `vector` *pour les tableaux de taille variable.*
Ces classes sont construites en utilisant les tableaux du C.

def-tableau.cpp

```
1
2  using namespace std;
3  #include<array>
4  using tabcpp = array<int, taille>;
5
6  #include<vector>
7  using vectcpp = vector<int>;
```

Type tableau...

Syntaxe

On accède, en lecture comme en écriture, à la valeur de chaque élément d'un tableau en utilisant la ***notation indicée*** *:*

```
t[indice]
```

où `indice` *est une expression (constante, variable, calcul...).*

Retenir

*L'accès n'est possible que si l'****indice est compris entre 0 et la taille du tableau - 1****. Le comportement d'un* ***accès en dehors des bornes du tableau*** *est* ***INDÉFINI*** *(segfault, rien, modification d'une autre variable, accès super utilisateur...).*

Type tableau...

Syntaxe

On accède, en lecture comme en écriture, à la valeur de chaque élément d'un tableau en utilisant la ***notation indicée*** *:*

```
t[indice]
```

où `indice` *est une expression (constante, variable, calcul...).*

Retenir

*L'accès n'est possible que si l'****indice est compris entre 0 et la taille du tableau - 1****. Le comportement d'un* ***accès en dehors des bornes du tableau*** *est* ***INDÉFINI*** *(segfault, rien, modification d'une autre variable, accès super utilisateur...).*

Tableau et mémoire

Retenir

Pour stocker un tableau, le compilateur réserve des ***emplacements mémoire consécutifs****.*

Le calcul de l'adresse du i-ème élément du tableau peut donc se faire avec :

$$\text{Adresse}(T[i]) = \text{Adresse}(T[0]) + i * \textit{Taille d'un élément}$$

Accès très rapide : ne demande pas de parcourir le tableau.

```
int t[10];
```

	0	1	2	3	4	5	6	7	8	9	
?	1	8	5	13	9	11	13	1	19	2	?

Tableau et mémoire

Retenir

Pour stocker un tableau, le compilateur réserve des ***emplacements mémoire consécutifs****.*
Le calcul de l'adresse du i-ème élément du tableau peut donc se faire avec :

$$\text{Adresse}(T[i]) = \text{Adresse}(T[0]) + i * \textit{Taille d'un élément}$$

Accès très rapide : ***ne demande pas de parcourir le tableau****.*

```
int t[10];
```

	0	1	2	3	4	5	6	7	8	9	
?	1	8	5	13	9	11	13	1	19	2	?

Passage d'un tableau en paramètre

Retenir (**Attention !**)

*Les tableaux ont un comportement particulier : lors d'un passage d'un tableau en paramètre, on ne passe que l'**adresse du premier élément**.*

En quelque sorte, le tableau est passé par référence. Il n'est pas possible de passer un tableau simple par valeur. En pratique, on veut rarement passer un tableau par valeur car la recopie est coûteuse. Solutions (voir `exemple/tab-const.cpp`) :

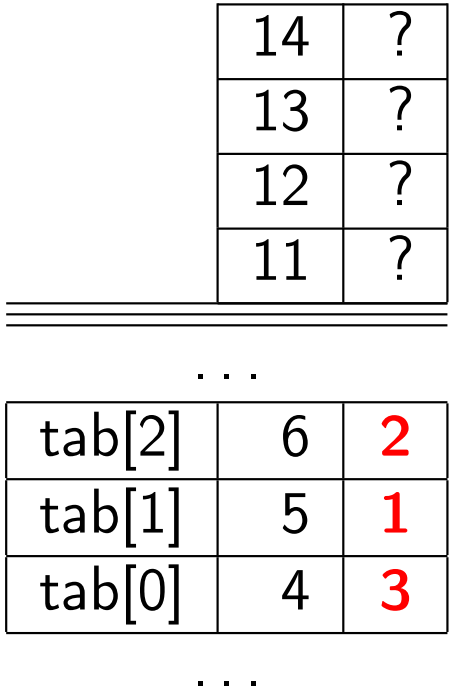
- Utiliser un tableau constant : `const int t[3]`
- Utiliser les `array` du `C++` : `array<int, 3> t`
- Mettre le tableau à l'intérieur d'une structure (voir plus loin)

Mécanisme du passage en paramètre d'un tableau

Avant l'appel ligne 6

```
1  void afficheTab(int t[3]) {
2    for (int i = 0; i < 3; i++)
3        cout << t[i] << " ";
4  }
5  [...]
6    int tab[3] = {3,1,2};
7    afficheTab(tab);
```

Pile :

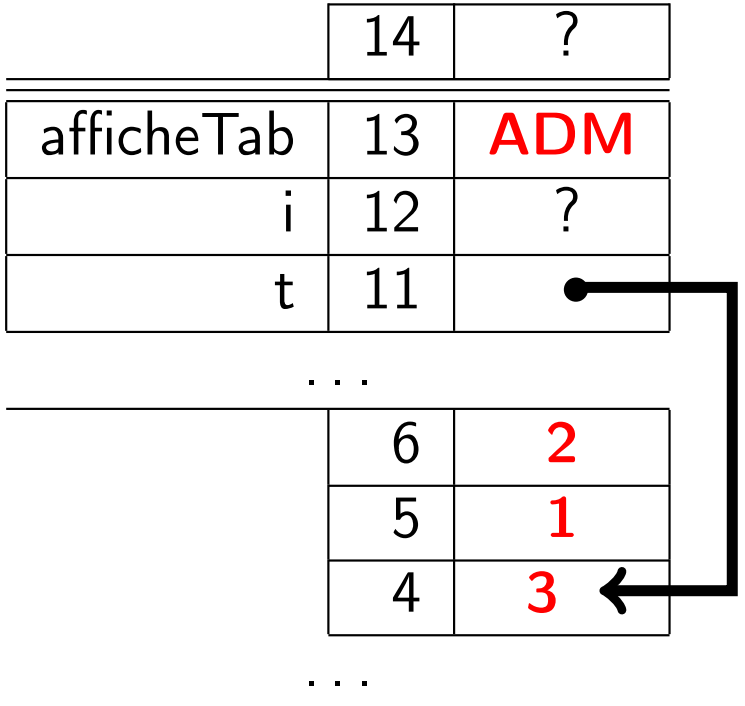


Mécanisme du passage en paramètre d'un tableau

Appel ligne 7

```
1  void afficheTab(int t[3]) {
2    for (int i = 0; i < 3; i++)
3        cout << t[i] << " ";
4  }
5  [...]
6    int tab[3] = {3,1,2};
7    afficheTab(tab);
```

Pile :



Rappel (Manipulation de tableau)

tab-acc.cpp

```cpp
#include<iostream>
#include<iomanip>
using namespace std;

const int taille = 10;
using tab = int[taille];

void lireTab(tab t) {
 for (int i=0; i<taille; i++)
    cin >> t[i];
}

void afficheTab(const tab t) {
  for (int i=0; i<taille; i++)
    cout << setw(3) << i;
  cout << endl;
  for (int i=0; i<taille; i++)
    cout << setw(3) << t[i];
  cout << endl;
}

float moyenneTab(const tab t) {
  int somme = 0;
  for (int i=0; i<taille; i++)
    somme += t[i];
  return float(somme) / float(taille);
}
```

tab-acc.cpp

```cpp

int maxTab(const tab t) {
  int maxi = 0;
  for (int i=0; i<taille; i++)
    maxi = max(maxi, t[i]);
  return maxi;
}

int main() {
  tab notes;

  lireTab(notes);
  cout << endl << endl << endl;

  afficheTab(notes);

  cout << "moyenne = " << moyenneTab(notes) << endl;

  cout << "maxi = " << maxTab(notes) << endl;

}
```

Passer un tableau de taille inconnue

Retenir

Dans le mécanisme de passage de paramètres précédent, on peut passer un tableau dont la ***fonction ne connaît pas la taille à la compilation****.*

tab-ukn-size.cpp

```
1  void afficheTab(int taille, const int t[]) {
2    for (int i = 0; i<taille; i++)
3      cout << t[i] << " ";
4  }
5
6  int main() {
7    int tab[3] = {12,43,13};
8    afficheTab(3, tab); cout << endl;
9    int tab2[4] = {1,34,54,12};
10   afficheTab(4, tab2); cout << endl;
11 }
```

Différences tableau/vecteur

- La taille d'un tableau est **fixée statiquement** et ne peut pas évoluer, alors que celle d'un vecteur peut évoluer en fonction des besoins
- Contrairement aux vecteurs, un tableau n'a pas de **méthode prédéfinie** :
  - calcul du nombre d'éléments : `v.size()`
  - ajout d'un élément à la fin : `v.push_back(n)`
- Il faut représenter, dans une variable annexe, le nombre d'éléments présents dans le tableau
- Les tableaux sont **toujours passés par un pointeur sur le premier élément** alors que les vecteurs sont passés normalement (par valeur, ou par référence avec `&`)
- Pas de distinction entre les phases de **déclaration et d'allocation** initiales comme pour les vecteurs
- **Pas d'affectation directe possible** d'un tableau dans un autre

Le principe de l'idée récursive

«Quand on a une bonne idée, en l'appliquant récursivement on obtient très souvent une bien meilleure idée.»

- Les structures et les tableaux sont de bonnes idées !
- En **combinant** les deux, on a encore beaucoup plus de possibilités.

Retenir

- *Il est possible qu'**un champ d'une structure soit un tableau***
- *On peut faire des **tableaux dont les éléments sont des structures***

Exemple de tableaux de structures

carnet.cpp

```
1  struct personne {
2    string nom, prenom;
3    date naissance;
4    long int telephone;
5    // int numero de rue; string nom de rue ...
6  };
7
8  const int taille = 1000;
9  using carnet = personne[taille];
10
11 void lirePersonne(personne &p) {
12   cin >> p.nom >> p.prenom;
13   p.naissance = lireDate();
14   cin >> p.telephone;
15 }
16
17
18 void affichePersonne(personne p) {
19   cout << p.nom << " " << p.prenom << " ";
20   afficheDate(p.naissance);
21   cout << " " << p.telephone;
22 }
23
24 void initCarnet(carnet c) {
25   for (int i=0; i< taille; i++)
26     c[i].nom = "";
27 }
```

carnet.cpp

```
1  void afficheCarnet(carnet c) {
2    for (int i=0; i< taille; i++)
3      if (c[i].nom != "") {
4        cout << i << " : ";
5        affichePersonne(c[i]);
6        cout << endl;
7      }
8    cout << endl;
9  }
10
11
12 int main() {
13   carnet addr;
14
15   initCarnet(addr);
16
17   addr[5] = {"toto", "dupont", {3, 12, 1998}, 1345782348};
18   afficheCarnet(addr);
19
20   int i;
21   cout << "Case a modifier ? ";
22   cin >> i;
23   lirePersonne(addr[i]);
24   cout << endl;
25
26   afficheCarnet(addr);
27 }
```