Labo d'introduction à l'informatique

pour les mathématiques

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Semaine 8

Dictionnaires (Maps)

Structures

La fin de la programmation ?

Avec les structures de contrôle que nous avons vues, ainsi que le fait de pouvoir définir différents types de variables dont les tableau, nous pouvons en théorie aborder n’importe quel problème de programmation !

La fin de la programmation ?

Avec les structures de contrôle que nous avons vues, ainsi que le fait de pouvoir définir différents types de variables dont les tableau, nous pouvons en théorie aborder n'importe quel problème de programmation !

Nous avons même en général plusieurs façons de résoudre un problème donné.

Aujourd'hui, nous allons voir des concepts nouveaux qui nous **simplifient** la vie ou bien rendent nos programmes plus **efficaces**.

Un nouveau problème

Supposons que l'on veuille écrire une fonction `prix(x)` qui retourne, grâce au nom du produit `x` d'un magasin, le prix de ce produit.

Par exemple : `prix("jus d'orange")` retourne la valeur `1.5`.

À quoi devrait ressembler le code d'une telle fonction ?

On suppose qu'il n'y a que 5 produits possibles pour l'exemple : *jus d'orange, eau minérale, soda, chips, chocolat*… mais il faut s'en imaginer davantage pour saisir l'essence du problème !

Solution à base de conditions `if/elseif/else`

```
function p = prix(x)
    if strcmp(x, "jus d'orange")
        p = 1.5;
    elseif strcmp(x, "eau minérale")
        p = 0.75;
    elseif strcmp(x, "soda")
        p = 2.5;
    elseif strcmp(x, "chips")
        p = 3.5;
    elseif strcmp(x, "chocolat")
        p = 2;
    else
        p = 0;
        disp("Erreur : produit non disponible")
    end
end
```

Fastidieux, répétitif !

Solution à base de tableaux

```
function p = prix(x)
    cles = {'jus d''orange', 'eau minérale', 'soda', 'chips', 'chocolat'};
    valeurs = [1.5, 0.75, 2.5, 3.5, 2];
    p = 0;
    for i=1:length(x)
        if strcmp(cles{i}, x)
            p = valeurs(i);
        end
    end
    if p == 0
        disp("Erreur : produit non disponible")
    end
end
```

Nous verrons en cours pourquoi cette solution n'est pas performante en général.

Solution à base d'une structure `switch/case`

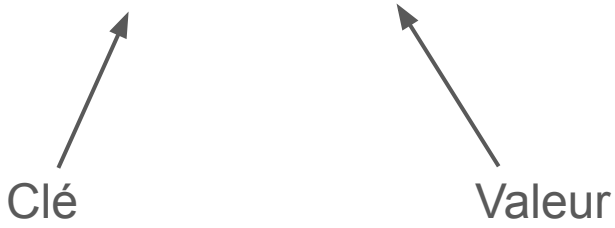
```
function p = prix(x)
    switch x                    % Voici un nouveau mot clé pour nous !
        case "jus d'orange"
            p = 1.5;
        case "eau minérale"
            p = 0.75;
        case "soda"
            p = 2.5;
        case "chips"
            p = 3.5;
        case "chocolat"
            p = 2;
        otherwise
            p = 0;
            disp("Erreur : produit non disponible");
    end
end
```

Permet d'éviter la répétition explicite des comparaisons

Un nouveau type de donnée

Le plus pratique serait de pouvoir définir un “catalogue” des prix.

Un tel catalogue lierait des textes à des prix.



Une structure de donnée permettant de définir des **couples clé-valeur** est le **dictionnaire** (ou map).

Solution à base d'un dictionnaire

```
function p = prix(x)
    d = containers.Map();
    d("jus d'orange") = 1.5;
    d("eau minérale") = 0.75;
    d("soda") = 2.5;
    d("chips") = 3.5;
    d("chocolat") = 2;
    if isKey(d, x)
        p = d(x);
    else
        p = 0;
        disp("Erreur : produit non disponible")
    end
end
```

On lie des chaînes de caractères à des nombres

Solution à base d'un dictionnaire

```
function p = prix(x)
    d = containers.Map();
    d("jus d'orange") = 1.5;
    d("eau minérale") = 0.75;
    d("soda") = 2.5;
    d("chips") = 3.5;
    d("chocolat") = 2;
    if isKey(d, x)
        p = d(x);
    else
        p = 0;
        disp("Erreur : produit non disponible")
    end
end
```

Nous verrons en cours pourquoi cette solution est performante

Solution à base d'un dictionnaire (alternative)

```
function p = prix(x)
    cles = {'jus d''orange', 'eau minérale', 'soda', 'chips', 'chocolat'};
    valeurs = [1.5, 0.75, 2.5, 3.5, 2];
    d = containers.Map(cles, valeurs);
    if isKey(d, x)
        p = d(x);
    else
        p = 0;
        disp("Erreur : produit non disponible")
    end
end
```

Itérer sur les éléments d’un dictionnaire

Il est important de noter qu’un dictionnaire n’est **pas un groupement ordonné** des valeurs comme un tableau.

Par exemple, le code suivant itère clés-valeurs, qui n’apparaissent pas forcément dans le même ordre que celui où on les a insérés :

```
% On suppose que d existe

cles = keys(d);  % Récupère toutes les clés du dictionnaire
for i = 1:length(cles)
    cle = cles{i};  % Accès à chaque clé
    valeur = d(cle);  % Accès à la valeur associée à la clé
    fprintf('Le produit "%s" coûte %.2f CHF.\n', cle, valeur);
end
```

Les avantages de la solution avec le dictionnaire

- **Clarté** du code.
- **Performance** du code pour l'accès aux éléments.
- Une variable unique pour tout **regrouper**.
- Permet de facilement **insérer** ou **retirer** des valeurs, y compris durant l'exécution :

  Insertion d'un couple clé-valeur supplémentaire : `d('riz') = 3.5;`

  Suppression d'un couple clé-valeur existant : `remove(d, 'soda');`

Types de clés-valeurs

- Dans les diapos précédentes, nous nous sommes limités à des clés textuelles et à des valeurs numériques.
- Les dictionnaires (Maps) peuvent associer des nombres ou des chaînes de caractères à toutes sortes de valeurs différentes.

Exercice rapide

Considérons un dictionnaire au sein duquel on lie des mots à une valeur booléenne (1 s'il y a au moins une fois la lettre e dans le mot, 0 sinon).

La fonction suivante permet d'ajouter automatiquement un mot au dictionnaire.

```
function dico=ajout_au_dico(dico, mot)
    dico(mot) = 0; % a priori, pas de 'e' dans le mot...
    for i=1:length(mot)
        if mot(i) == 'e'
            % à compléter ! Ici, on ajoute le mot à dico avec la valeur 1.
        end
    end
end
```

Exercice rapide - Correction

Considérons un dictionnaire au sein duquel on lie des mots à une valeur booléenne (1 s'il y a au moins une fois la lettre e dans le mot, 0 sinon).

La fonction suivante permet d'ajouter automatiquement un mot au dictionnaire.

```
function dico=ajout_au_dico(dico, mot)
    dico(mot) = 0; % a priori, pas de 'e' dans le mot...
    for i=1:length(mot)
        if mot(i) == 'e'
            dico(mot) = 1;
            break;
        end
    end
end
```

Quelle solution choisir ?

- Dépend du problème.
- Toutes les solutions sont fonctionnelles, mais toutes ne sont pas efficaces ou simples.
- Savoir choisir la solution pertinente fait partie du travail de programmation.
- Dans certaines situations, plusieurs solutions se valent.

Exemples de solutions naturelles

- Stocker les coefficients d'un polynôme : tableau de nombres.
  coeffs = [3, 0, -6, 5];

- Stocker différents mots : tableau de chaînes de caractères.
  mots = {'Ballon', 'Balle', 'Frisbee'};

- Stocker l'altitude de différentes villes : dictionnaire.

```
a = containers.Map();
a('Genève') = 400;
a('Barcelone') = 0;
```

Encore un nouveau problème

- Stocker les coefficients d'un polynôme : tableau de nombres.
  coeffs = [3, 0, -6, 5];

- Stocker différents mots : tableau de chaînes de caractères.
  mots = {'Ballon', 'Balle', 'Frisbee'};

- Stocker l'altitude de différentes villes : dictionnaire.

```
a = containers.Map();
a('Genève') = 400;
a('Barcelone') = 0;
```

- **Stocker l'altitude, population, pays, indicatif téléphonique de différents villes ?**

Encore un nouveau problème

- Stocker l'altitude, population, pays, indicatif téléphonique de différents villes ?
  ⇒ On peut imaginer toutes sortes de solutions.

- Exemple de solution possible :

```
nom = {'Genève', 'Barcelone', 'New York', 'Tokyo'};
pop = [5e5, 2e6, 8e6, 14e6];
pays = {'Suisse', 'Espagne', 'USA', 'Japon'};
etc…
```

  ⇒ mais dans tous les cas, cela constitue une solution *ad hoc*.
  Il est désirable de tout **regrouper** au sein d'une **variable unique** !

- Dans ce genre de situation, déclarer une nouvelle **structure** peut être utile.

Structures

- Permet d’associer un nombre arbitraire de champs à une variable unique.

- Par exemple :

```
geneve.nom = 'Genève';
geneve.pop = 5e5;
geneve.pays = 'Suisse';
geneve.indicatif = '+41 22';
geneve.communes = {'Meyrin', 'Vernier', . . .};
```

Tableaux de structures

La fonction `repmat` permet de créer une matrice d'objets structurés de la même façon qu'un modèle :

```
ville.nom = ' ';
ville.pop = 0;
ville.pays = ' ';
mes_villes = repmat(ville, 1, 100) % tableau de cent villes
mes_villes(8).nom = 'Berlin'; % ici, on définit le nom de la ville 8
```

Tableaux cellulaires de structures

On peut aussi stocker des structures dans des tableaux cellulaires :

```
a.nom = 'Alice';
a.age = 34;
b.nom = 'Bob';
b.age =27;

personnes = {a, b};
personnes{2}.age = 32; % On modifie l'âge de Bob
```

Notes finale

- Les tableaux cellulaires et les matrices sont deux possibilités distinctes pour stocker des structures.

- Erreur commune : quand une fonction reçoit une structure en entrée, il n'y a pas besoin de définir ses membres pour y faire référence ! Il suffit de les utiliser en supposant qu'ils existent.
  Exemple de fonction qui utilise une structure passée en argument :

```
func = affiche_personne(x)
    fprintf('La personne %s possède %d ans.\n', x.nom, x.age)
end
```