Haute École de Bruxelles
École Supérieure d'Informatique

TD Variables structurées

Résumé

Ce TD a pour but de rappeler les variables structurées.

1 Les variables structurées

Dans les chapitres précédents, nous avons utilisé des variables de types dits «simples »(entier, réel, booléen, caractère, chaine) ne pouvant contenir qu'une seule valeur à la fois. Cependant, certains types d'information consistent en un regroupement de plusieurs données élémentaires. Quelques exemples :

- Une date est composée de trois éléments (le jour, le mois, l'année). Le mois peut s'exprimer par un entier (15/10/2014) ou par une chaine (15 octobre 2014).
- Un moment de la journée est un triple d'entiers (heures, minutes, secondes).
- La localisation d'un point dans un plan nécessite la connaissance de deux coordonnées cartésiennes (l'abscisse x et l'ordonnée y) ou polaires (le rayon et l'angle).
- Une adresse est composée de plusieurs données : un nom de rue, un numéro de maison, parfois un numéro de boite postale, un code postal, le nom de la localité, un nom ou code de pays pour une adresse à l'étranger. . .

Pour stocker et manipuler de tels ensembles de données, nous utiliserons des **types structurés** ou **structures**. Une structure est donc un **ensemble fini d'éléments pouvant être de types distincts**. Chaque élément de cet ensemble, appelé **champ** de la structure, possède un nom unique.

Notez qu'un champ d'une structure peut lui-même être une structure. Par exemple, une carte d'identité inclut parmi ses informations une date de naissance, l'adresse de son propriétaire (cachée dans la puce électronique !). . .

1.1 Définition d'une structure

La définition d'un type structuré adoptera le modèle suivant :

```
structure NomDeLaStructure
    nomChamp1 : type1
    nomChamp2 : type2
    ...
    nomChampN : typeN
fin structure
```

`nomChamp1`, . . ., `nomChampN` sont les noms des différents champs de la structure, et `type1`, . . ., `typeN` les types de ces champs. Ces types sont soit

les types «simples »étudiés précédemment (entier, réel, booléen, caractère, chaine) soit d'autres types structurés dont la structure aura été préalablement définie.

Pour exemple, nous définissons ci-dessous trois types structurés que nous utiliserons souvent par la suite :

```
structure Date
    jour : entier
    mois : entier
    année : entier
fin structure


structure Moment
    heure : entier
    minute : entier
    seconde : entier
fin structure


structure Point
    x : réel
    y : réel
fin structure
```

1.2 Déclaration d'une variable de type structuré

Une fois un type structuré défini, la déclaration de variables de ce type est identique à celle des variables simples. Par exemple :

```
anniversaire, jourJ : Date
départ, arrivée, unMoment : Moment
a, b, centreGravité : Point
```

1.3 Utilisation des variables de type structuré

En général, pour manipuler une variable structurée ou en modifier le contenu, il faut agir au niveau de ses champs en utilisant les opérations permises selon leur type. Pour accéder à l'un des champs d'une variable structurée, il faut mentionner le nom de ce champ ainsi que celui de la variable dont il fait partie. Nous utiliserons la notation «pointée » :

```
nomVariable.nomChamp
```

Exemple d'instructions utilisant les variables déclarées au paragraphe précédent :

```
anniversaire.jour ← 15
anniversaire.mois ← 10
anniversaire.année ← 2014
arrivée.heure ← départ.heure + 2
centreGravité.x ← (a.x + b.x) / 2
```

On peut cependant, dans certains cas, utiliser une variable structurée de manière globale (c'est-à-dire d'une façon qui agit simultanément sur chacun de ses champs). Le cas le plus courant est l'affectation interne entre deux variables structurées de même type, par exemple :

```
arrivée ← départ
```

qui résume les trois instructions suivantes :

```
arrivée.heure ← départ.heure
arrivée.minute ← départ.minute
arrivée.seconde ← départ.seconde
```

Une variable structurée peut aussi être le paramètre d'un module, et un module peut également renvoyer une «valeur »de type structuré. Par exemple, l'entête d'un module renvoyant le nombre de secondes écoulées entre une heure de départ et d'arrivée serait :

```
module nbSecondesEcoulées( départ↓, arrivée↓ : Moment) → entier
```

On pourra aussi lire ou afficher une variable structurée :

```
lire unMoment
afficher unMoment
```

Par contre, il n'est pas autorisé d'utiliser les opérateurs de comparaison ($<$, $>$) pour comparer des variables structurées (même de même type), car une relation d'ordre n'accompagne pas toujours les structures utilisées. En effet, s'il est naturel de vouloir comparer des dates ou des moments, comment définir une relation d'ordre avec les points du plan ou avec des cartes d'identités ?

Si le besoin de comparer des variables structurées se fait sentir, il faudra dans ce cas écrire des modules de comparaison adaptés aux structures utilisées.

Par facilité d'écriture, on peut assigner tous les champs en une fois via des « ». Exemple :

```
anniversaire ← {1, 9, 1989}
```

1.4 Exemple d'algorithme

Le module ci-dessous reçoit en paramètre deux dates ; la valeur renvoyée est -1 si la première date est antérieure à la deuxième, 0 si les deux dates sont identiques ou 1 si la première date est postérieure à la deuxième.

```
// Reçoit 2 dates en paramètres et retourne la valeur
// -1 si la première date est antérieure à la deuxième
// 0 si les deux dates sont identiques
// 1 si la première date est postérieure à la deuxième.
module comparerDates(date1↓, date2↓ : Date) → entier
    résultat : entier
    résultat ← -1 // valeur choisie par défaut
    si date1.année ≥ date2.année alors
      si date1.année > date2.année alors
        résultat ← 1
      sinon // les années sont identiques
        si date1.mois ≥ date2.mois alors
          si date1.mois > date2.mois alors
            résultat ← 1
          sinon // les années et les mois sont identiques
            si date1.jour ≥ date2.jour alors
              si date1.jour > date2.jour alors
                résultat ← 1
              sinon // les années, les mois et les jours sont identiques
                résultat ← 0
              fin si
            fin si
```

```
          fin si
        fin si
      fin si
    fin si
    retourner résultat
fin module
```

2 Exercices

Maintenant, mettons tout ça en pratique.

2.1 À vous de jouer...

N'oubliez pas nos quelques conseils pour vous guider dans la résolution de tels problèmes :

- il convient d'abord de bien comprendre le problème posé ; assurez-vous qu'il est parfaitement spécifié ;
- résolvez le problème via quelques exemples précis ;
- mettez en évidence les variables **« données »**, les variables **« résultats »** et les variables de travail ;
- n'hésitez pas à faire une ébauche de résolution en français avant d'élaborer l'algorithme définitif pseudo-codé ;
- déclarez ensuite les variables (et leur type) qui interviennent dans chaque module ; les noms des variables risquant de ne pas être suffisamment explicites.
- Écrivez la partie algorithmique **AVANT** de vous lancer dans la programmation en Java.
- Demandez-vous si vous avez besoin de parcourir tout le tableau ou de sortir prématurément (si on a trouvé ce qu'on cherche par exemple).

Écrivez un algorithme qui

1. reçoit en paramètre un moment d'une journée et qui retourne le nombre de secondes écoulées depuis minuit jusqu'à ce moment.
2. reçoit en paramètre un nombre de secondes écoulées dans une journée à partir de minuit et qui retourne le moment correspondant de la journée.
3. reçoit en paramètres deux moments d'une journée et qui retourne le nombre de secondes séparant ces deux moments.
4. recevant les coordonnées de deux points distincts du plan et qui retourne les coordonnées du point situé au milieu des deux.

5. recevant les coordonnées de deux points distincts du plan et qui retourne la distance entre ces deux points.

Cercle

Définir un type Cercle pouvant décrire de façon commode un cercle quelconque dans un espace à deux dimensions. Écrire ensuite

- un module calculant la surface du cercle reçu en paramètre ;
- un module recevant 2 points en paramètre et retournant le cercle dont le diamètre est le segment reliant ces 2 points ;
- un module qui détermine si un point donné est dans un cercle ;
- un module qui indique si 2 cercles ont une intersection.