Cours d'informatique II

2ème partie

Luka Nerima

semestre d'été 2005

"Structures de données"

PLAN

Bibliographie

Ouvrages conseillés:

MÖSSENBÖCK H., *Object-Oriented Programming in Oberon-2*, Springer, 1994.

Abstraction de données. Concepts de base de la programmation Objet. Applications typique des objets: types abstraits, composantes génériques, structure de données hétérogènes, etc. Conception orientée objet. Le modèle MVC (modèle - vue contrôleur). Etude de cas: système de fenêtres.

WEISS, M. A., *Data Structures & Problem Solving Using Java,* , Addison-Wesley, 1998.

Survol de Java. Objets et classes. Analyse de complexité des algorithmes. Structures de données: pile, queue, liste chaînée, arbre, arbre de recherche binaire, table de hachage, queue prioritaire, graphe. Algorithme sur les structures de données. Applications.

Chapitre 0. Notion de type de données

Langage de programmation typé:

- Impose que les variables du progamme soient explicitement définies avec un type de données
- Le typage des variables garantit que, pendant l'exécution du programme, les variables prendront uniquement des valeurs appartenant au domaine caractérisé par le type

Les langages typés proposent:

- des types de bases, prédéfinit dans le langage
- des constructeurs syntaxiques (ARRAY, RECORD, POINTER) qui permettent à l'utilisateur de définir ses propres types par combinaison des types de base et/ou des types définit par l'utilisateur

Les types de base (ou types primitifs) en Oberon

Il y a 10 types de base en Oberon (BB):

Type	Domaine de valeurs	Opérations
INTEGER	-2147483648..2147483647	+ - * DIV MOD DEC INC
BYTE	-128..127	+ - * DIV MOD DEC INC
SHORTINT	-32768..32767	+ - * DIV MOD DEC INC
LONGINT	-9.2E18..9.2E18	+ - * DIV MOD DEC INC
SHORTREAL	-3.4E38..3.4E38	op. relationnels: = # < <= >= >
REAL	-1.8E308..1.8E308	op. relationnels: = # < <= >= >
BOOLEAN	FALSE, TRUE	& OR ~ op. relationnels: = #
CHAR	0X..FFFFX	CAP op. conversion: ORD CHR
SHORTCHAR	0X...FFX	CAP op. conversion: ORD CHR
SET	set of 0..31	+ - * / IN INCL EXCL

Remarque:

- ce tableau n’est pas exhaustif en ce qui concerne les opérations (il manque notamment les procedures de conversion de type)

A quoi servent les types de données ?

En résumé, un type de données définit:

(1) le domaine des valeurs possibles

(2) les opérations applicables à ces valeurs

Type de base <-> type structuré:

- type de base: une valeur par variable
- type structuré une **collection** de valeurs par variable

Exemples:

```
VAR k: INTEGER; (* k peut contenir 1 valeur entière *)

    tab: ARRAY 10 OF INTEGER; (*tab peut contenir 10
                                 valeurs entières *)

    rec: RECORD                (* rec peut contenir
             r: REAL;          une valeur de type réel
             i: INTEGER;       et une valeur de type entier *)
         END
```

Type statique <-> type dynamique

- statique: la place mémoire est allouée à l'initialisation.
- dynamique (pointeur): la mémoire est allouée à la demande, en cours d'exécution du programme (avec l'opération NEW)
- une structure de type dynamique peut croître "infiniment" pendant l'exécution du programme; elle peut se réduire aussi
- chaque variable allouée dynamiquement est référencée par une variable de type pointeur

Déclaration d'une variable de type pointeur et création de la variable dynamique

- la déclaration d'une variable de type pointeur indique de quel type sera la variable allouée dynamiquement (appelé "type de base"). Exemple:

```
TYPE String = ARRAY 32 OF CHAR;
        PERSONNE = RECORD nom, prénom: String; END;

VAR p: POINTER TO PERSONNE;
```

- La déclaration ne crée pas la variable dynamique elle-même. La création doit se faire explicitement avec l'instruction

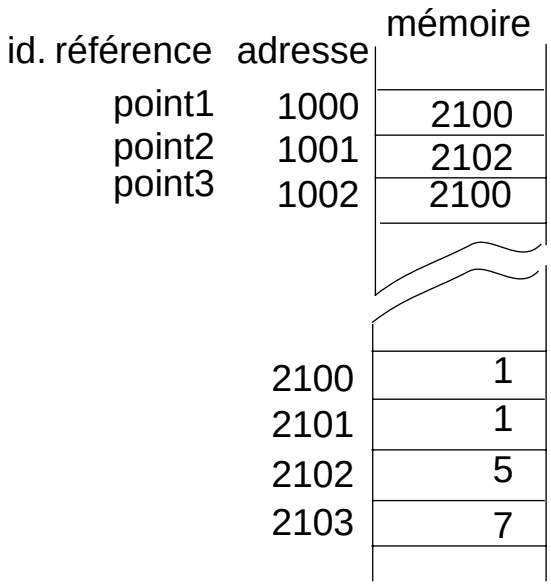
```
NEW(p)
```

- La variable ainsi crée n'est pas désignée par un identificateur: elle est anonyme. Pour y accéder, on utilisera la variable de type pointeur (p)
- On appelles plus simplement les variables de type pointeur "pointeurs" ou "références"

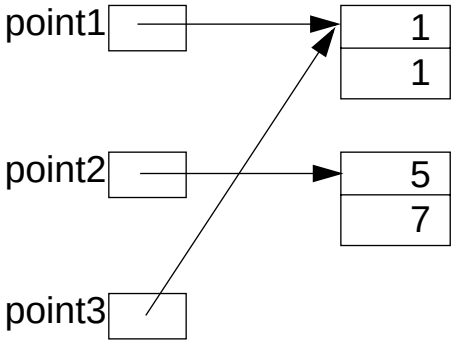
Chapitre 1. La référence

- Référence ≡ variable de type pointeur (POINTER)
- La référence contient l'adresse mémoire où réside un enregistrement ou, en toute généralité, une unité d'information
- Exemple: deux enregistrements représentant chacun une coordonnée (x,y) de l'écran; ils sont référencés par trois références: point1, point2 et point3:

Représentation physique:



Représentation conceptuelle:

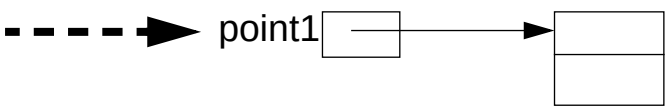


- Notion importante car elle concerne:
  - les variables de type dynamique POINTER
  - les objets (que nous verrons au chapitre 3)

Déclaration de la référence, création et suppression des enregistrements référencés

- La déclaration d'une référence se fait de la même manière que celle des autres types de variable
- Pour indiquer qu'il s'agit d'une référence on utilise le mot réservé POINTER
- Aprés la déclaration de la référence, sa valeur est NIL
- La création (allocation de la mémoire) d'un enregistrement référencé se fait à l'aide de la procédure NEW; la valeur de la référence devient#NIL
- Exemple:

```
TYPE Point = POINTER TO  RECORD x, y: INTEGER END;

VAR point1: Point;
BEGIN
    ...
    NEW(point1);
    ...
```



- Lorsqu'un enregistrement n'est plus référencé, il n'y a plus moyen d'y accéder; l'enregistrement est alors candidat à la suppression automatique par le ramasseur de miette (garbage collection)

Remarque:

- En Oberon il est également possible de référencer des tableaux

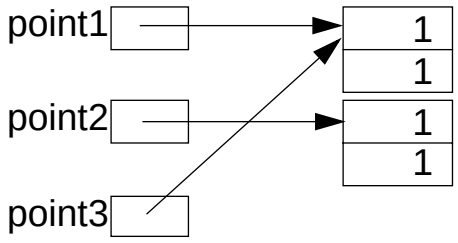
L’opérateur flèche (^) et l’opérateur point (.)

- L’opérateur flèche (^) sert à “déréférencer” la référence, càd à accéder à l’enregistrement référencé. Attention: la référence doit être # NIL
- L’opérateur point (.) sert à accéder aux champs de l’enregistrement; nous verrons aux chap. 2 et 3 que les champs peuvent aussi représenter des méthodes
- Pour alléger l’écriture, l’opérateur (^) est facultatif lorsqu’on accède aux champs d’un enregistrement référencé. On peut écrire point1.x au lieu de point1^.x

Exemple:

```
TYPE Point = POINTER TO RECORD x, y: INTEGER END;
VAR point1, point2, point3: Point;
BEGIN
    ...
    NEW(point1);
    point1.x:=1; point1.y:=1;
    point3:=point1; (* affectation de la référence -> point3 se
                       réfère au même enregistrement que point1*)
    NEW(point2);
    point2^:=point1^; (* affectation de l’enregistrement *)
    ...
```

Après l’exécution:

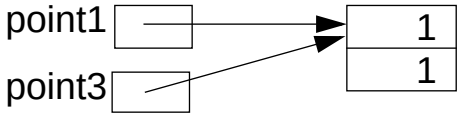


La signification de :=

Soit l'instruction d'affectation g:=d

- L'affectation (:=) entre deux variables primitives, disons de type INTEGER, a pour effet de copier la valeur de la variable de droite dans la variable de gauche.
- L'affectation entre deux références a exactement le même effet: la valeur de la variable de droite est copiée dans la variable gauche. Comme la valeur est une adresse mémoire, après l'affectation la variable de gauche va se référer au même enregistrement (ou objet) que la variable de droite.

P.e. après l'affectation p3:=p1



Les affectations ci-dessous sont-elles légales ?

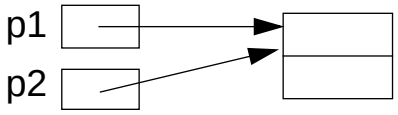
```
TYPE String = ARRAY 32 OF CHAR;
       NoeudListe = POINTER TO Noeud;
       Noeud = RECORD clé: String; suivant: NoeudListe; END;

VAR liste: NoeudListe; noeud: Noeud;
BEGIN ...
    NEW(liste);
    noeud := liste;
    noeud := liste^;
    noeud^.suivant := liste;
    noeud.suivant := liste;
    noeud.suivant := liste.suivant;
    ...
```

La signification de =

- Deux références sont égale au sens de l'opérateur (=) si elles se réfèrent au même enregistrement (ou objet)

Exemple:

p2:=p1

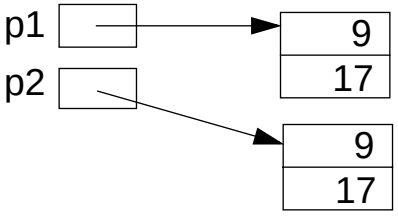


(* après cette affectation, l'expression p1 = p2 est **vraie** *)

- Mais deux références p1 et p2 qui se réfèrent à deux enregistrements différents ne sont pas égales, même si tous les champs des enregistrements sont égaux deux à deux

Exemple:

NEW(p1); NEW(p2);
p1.x:=9; p1.y:=17;
p2.x:=9; p2.y:=17;



(* après ces instructions, p1 = p2 est **fausse***)

(* mais p1.x=p2.x & p1.y=p2.y est vraie (appelée"égalité profonde")*)