Pointeurs et Références [pn]
Support de Cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Université Haute-Alsace

Version 20 mai 2018

Table des matières

alg - Pointeurs et Références

Mots-Clés Pointeurs, Références ■
Requis Structuration de l'information ■
Difficulté ●●○ (1 h) ■

Introduction

Ce module introduit les variables de type **pointeur vers**, aussi appelées **variables pointeurs**. Après un rappel général sur l'allocation de la place mémoire aux variables classiques lors des exécutions, il présente le mécanisme des variables pointeurs puis décrit les instructions qui permettent de les manipuler.

1 Type Pointeur

1.1 Qu'est-ce qu'un pointeur ?

Question
Que faites-vous quand vous déménagez ?

Réponse
Vous communiquez la nouvelle adresse à tous vos amis. Ce serait mieux si vous aviez une **adresse universelle** connue de tous et gérée par « La Poste » : vous n'auriez plus qu'à prévenir une unique personne du changement : « La Poste ».

Question
Comment gardez-vous un site intéressant sur Internet ?

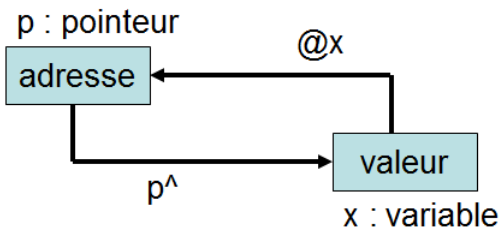
Réponse
Via un « bookmark/signet » dans votre navigateur : c'est un lien vers l'information qui vous intéresse.

Question
Et comment fait-on cela en programmation ?

Réponse
Pour faire la même chose et garder un lien vers une entité (une variable, fonction, etc.), c.-à-d. en avoir une **référence universelle**, on utilise des **pointeurs**.

Pointeur
Élément informatique qui contient l'**adresse** d'un autre élément informatique. Il peut être vu comme un « localisateur » : par son contenu le pointeur désigne (pointe sur) l'emplacement physique d'une autre valeur (pointeur de donnée) ou d'une fonction (pointeur de code). La valeur d'un *pointeur* est une **adresse virtuelle**.

Représentation graphique
Si `x` désigne une variable, l'expression `@x`, qui fournit l'adresse de `x`, est un pointeur ; ici `@x` pointe vers `x`. Si `f` désigne une procédure ou fonction, l'expression `@f` pointe vers le début du code de la procédure ou fonction `f`.



Sous-type du pointeur

Un pointeur est typé : il est lié au type de l'entité vers laquelle il pointe. Le type de la variable pointée est appelé le **sous-type du pointeur**.

A quoi ça sert ?

En programmation, les pointeurs servent essentiellement à :

- **Partager** des objets (données, fonctions...) entre plusieurs portions de code **sans** les **dupliquer** par la notion de *référence*.
- **Choisir** des éléments non connus à priori (au moment de la programmation) par la notion de *généricité*.
- Manipuler des objets dont la **portée** (c.-à-d. la durée de vie) **dépasse les blocs** dans lesquels ils sont déclarés via l'*allocation dynamique* de mémoire.

Remarque

Les pointeurs n'existent pas dans tous les langages (e.g. Java et Python).

1.2 Déclaration d'un pointeur

((alg)) **Déclaration d'un pointeur**

```
Variable p : Pointeur T
```

Explication

Déclare une variable `Pointeur` de nom `p` vers une valeur de type `T`. On dit que `p` est un pointeur de type `T`.

Taille d'un type pointeur

Le nombre d'octets de l'adresse est **indépendante** du type de l'entité référencée.

```
Taille(Pointeur Caractère) = Taille(Pointeur Réel) = 4
(Taille(Caractère) = 1) <> (Taille(Réel) = 8)
```

Attention

La déclaration d'une variable pointeur réserve les octets nécessaire au codage d'une adresse mémoire mais ne réserve aucune mémoire pour la donnée pointée.

1.3 La valeur Nil

La valeur Nil

Not Identified Link, elle désigne la valeur associée au **pointeur nul** (dit aussi *nul*). Elle peut être mémorisée dans toute variable pointeur, quel que soit son sous-type.

La valeur Nil

```
Nil // ou NULL
```

Utilité

Il sert généralement à signaler l'échec d'une opération, l'apparition d'une erreur ou une fonction non définie, etc.

Représentation graphique

La valeur `Nil` est souvent représentée par un trait barrant en diagonale, de bas en haut et de gauche à droite, la case représentant la variable pointeur. 

1.4 L'opérateur d'adresse

Opérateur d'adresse
Désigne **l'adresse mémoire de l'entité** (variable, constante, fonction...).

((alg)) **Opérateur d'adresse**

```
@ x
```

1.5 L'opérateur d'indirection

Opérateur d'indirection

Dit aussi *opérateur de déréférence*, il désigne le contenu de la **valeur pointée** par le pointeur. On dit que la variable pointeur subit un **déréférencement** par analogie au fait que le pointeur **référence** une variable : il en contient l'adresse mémoire.

((alg)) **Opérateur d'indirection**

```
p^ # avec p pointeur
```

Explication

Renvoie la **valeur pointée par** le pointeur `p`. L'expression `p^` est appelée **pointeur déréférencé**. Dit autrement, `p^` est une variable du type de la zone pointée.

Attention

Lorsqu'un pointeur vaut `Nil`, il n'y a aucun sens de tenter d'accéder à la zone pointée. Ainsi l'instruction suivante produira un comportement imprévisible :

```
x <- p^ // OUPS si p vaut Nil
```

1.6 Exemples

alg : Exemple Opérateur d'adresse

```
Variable x : Entier
Variable p : Pointeur Entier
Début
| x <- 3
| p <- @x // p pointe vers x
| p^ <- p^ + 1 // incrémente x via p
Fin
```

2 Pointeur : Compléments

Déplacer dans la partie 1 et mettre ici la partie qui concerne les structures. ■

2.1 Affectation d'un autre pointeur

L'affectation entre pointeurs est licite si les deux pointeurs ont le **même** sous-type.

Exemple Affectation

```
Variable p, q : Pointeur T
Début
| p <- q // p pointe vers la même zone que q
Fin
```

Attention !

Il n'y a aucune différence entre une adresse et une adresse ! Ainsi il est possible d'affecter l'adresse d'une variable de type `T2` à un pointeur vers une donnée de type `T1`. Dans ce cas les résultats sont imprévisibles !

```
Variable x : Réel
Variable p : Pointeur Entier
Début
| x <- 3.141592
| p <- @x // autorisé -- mais que pointe p ?
Fin
```

2.2 Initialisation de pointeurs

Un pointeur est un type élémentaire. Par conséquent les éléments de type pointeur ne sont pas initialisés. L'initialisation pourra s'effectuer par l'affectation de :

- L'adresse d'une entité `@x`.
- L'adresse d'une procédure ou fonction `@f`.
- Un autre pointeur du même type.
- La valeur `Nil` du pointeur nul.

Pointeur pendant

Les pointeurs sont la source principale d'erreurs durant l'exécution des programmes. Le problème le plus fréquent, souvent dû à l'absence d'initialisation, est un pointeur qui ne désigne rien d'utilisable : on parle de **pointeur pendant**. Il est vital encore plus que pour les autres types, que les pointeurs soient initialisés. Faites-en une règle.

Toute déclaration de pointeur doit être associée à une initialisation, ou à une affectation proche. Utilisez `Nil` si vous ne connaissez pas encore la mémoire pointée au moment de l'initialisation.

2.3 Comparaisons entre pointeurs

Les comparaisons concernant les pointeurs sont réduites. Ainsi :

- Deux variables pointeurs peuvent être comparées entre elles si et seulement si elles sont du même sous-type. Dans ce cas, seules les opérateurs de comparaison d'**égalité** = et d'**inégalité** <> ont un sens.
- La valeur `Nil` peut être comparée à toutes les variables pointeurs quels que soient leurs sous-types.

Conclusion

Le test entre variables pointeurs ne permet de savoir que deux choses :

- Si deux variables pointeurs pointent sur la même variable pointée.
- Si une variable pointeur mémorise `Nil`.

3 Exemple : Manipulation de pointeurs

((alg)) **Exemple Pseudo-code**

```
Algorithme pg_pnExemple
Type EnrClient
|  nom : Chaîne
|  total : Entier
FinType
Variable valeur : Entier
Variable pnVal : Pointeur Entier
Variable client : EnrClient
Variable pnClient : Pointeur EnrClient
Début
|  pnVal <- @valeur
|  pnClient <- @client
|  Saisir(pnVal^)
|  Saisir(pnClient^.nom)
|  Saisir(pnClient^.total)
|  pnClient^.total <- pnClient^.total + pnVal^
|  Afficher("Client ", pnClient^.nom, " : Total vaut ", pnClient^.total)
Fin
```

Explication

L'algorithme déclare deux variables pointeurs `pnVal` (vers entier) et `pnClient` (vers un `EnrClient`) aux contenus indéterminés.

L'exécution des deux premières instructions a pour effet d'initialiser les variables pointeur : la première récupère l'adresse de l'entier `valeur`, la deuxième celle de la structure `client`.

Il saisit ensuite un entier ainsi que les informations d'une structure `EnrClient` via les variables pointeurs `pnVal` et `pnClient`. Il cumule le total du client avec la `valeur` en utilisant les variables pointeurs puis affiche la structure `EnrClient`.