Université A. Mira de Béjaia
Faculté des Sciences Exactes
Département d'Informatique

Chapitre 8 partie 1 : Les pointeurs

Contenu du chapitre :

1. Notion d'adresse

La mémoire centrale (RAM) utilisée par les programmes est découpée en octets, dont chacun est identifié par un numéro séquentiel appelé **adresse**. Par convention, une adresse est notée en hexadécimal et précédée par 0x.

Adresse	**Case memoire**
0x3fffd10	
0x3fffd11	
0x3fffd12	
0x3fffd13	

Déclarer une variable c'est attribuer un *nom* (*identificateur*) à une zone de la mémoire centrale. Cette zone est définie par :

- Sa position : l'adresse de de son premier octet.
- Sa taille : le nombre d'octets.

Exemple :

X : entier ;
X ← 18 ;

Adresse	**Case memoire**	**Zone**
0x3fffd10		
0x3fffd11	**18**	X
0x3fffd12		
0x3fffd13		
0x3fffd14		
0x3fffd15		
0x3fffd13		
0x3fffd16		
0x3fffd17		

2. Définition d'un pointeur

Pour accéder à une valeur d'une variable, il suffit d'utiliser son nom. D'une manière analogue, pour accéder à l'adresse d'une variable, on utilise un pointeur. Par conséquent, **un pointeur est une variable qui contient l'adresse d'une autre variable**.

Le but d'utilisation d'un pointeur est d'accéder directement à la mémoire, de faire référence à une entité, sans la nommer.

3. Déclaration d'un pointeur

On déclare un pointeur de la manière suivante :

Nom : ^ Type pointé ; //Pointeur sur type pointé

Par exemple, on déclare un pointeur et un entier :

```
Var
    P : ^ entier ;    //Pointeur sur entier
    X : entier ;
```

4. Manipulation des pointeurs

4.1 Initialisation

Il existe 3 manières pour initialiser une variable de type pointeur :

- Initialisation avec la constante **NIL** (le pointeur ne pointe sur rien).
- Initialisation avec l'adresse d'une donnée existante (adresse d'une variable ou valeur d'un autre pointeur).
- Initialisation par allocation dynamique de mémoire (**allouer**).

Il est à noter que pour une variable donnée, seule une des trois manières doit être utilisée.

4.1.1 Initialisation par NIL

Un pointeur qui n'est pas initialisé s'appelle un **pointeur pendant**. C'est un pointeur qui pointe n'importe où. Si on déréférence ce pointeur et on lui affecte une nouvelle valeur, on va écraser un emplacement mémoire quelconque et on risque de faire planter le programme. Si on veut que le pointeur pointe nul part, il faut l'initialiser à **NIL** (Not Identified Link, NULL en C).

Exemple :

```
P : ^ entier ;
P ← NIL ; // P pointe nulle part
```

4.1.2 Initialisation par des données

On affecte l'adresse d'une variable en utilisant l'opérateur d'adresse **&**.

Exemple :

```
Var
    P : ^entier ;
    X : entier ;
Début
    X ← 18 ; //X est placée dans les quatre cases qui suivent la case à l'adresse 0x3fffd11
    P ←&X ;   /*P contient l'adresse de la première case de la zone attribuée à X. on dit dans ce cas-là : P pointe sur X). */
```

Adresse	Case memoire	Zone
0x3fffd10		
0x3fffd11	**18**	X
0x3fffd12		
0x3fffd13		
0x3fffd14		
0x3fffd15		
0x3fffd13	0x3fffd11	P

Remarque :

- Un pointeur pointe sur la variable dont il contient l'adresse.
- Un pointeur est associé à un type de variable sur lequel il peut pointer. Par exemple, un pointeur sur entier ne peut pointer que sur des variables entières.

Il est possible d'extraire la valeur de la variable pointée. Cette opération s'appelle **déréférenciation** d'un pointeur. La déréférenciation est possible grâce à l'opérateur ^.

Exemple :

```
Var
        X : entier ;
        P : ^ entier ;
 Début
        X ←33 ;
        P ← &X ;      //P pointe sur X
        Écrire (p^) ;     // la valeur 33 est affichée sur l'écran
        P^ ← 34 ;      // X vaut maintenant 34
```

Exemple récapitulatif : Donnez la trace d'exécution des instructions suivantes :

```
Var
        X, N : entier ;
        P : ^ entier ;
 Début
        N ← 20 ;
        P ← &N ;
        Écrire (P^) ;
        X ← P^ ;
        P^ ← 30 ;
        Écrire (P^) ;
        P ← &X ;
        Écrire (P^) ;
```

4.1.3 Allocation dynamique

La RAM d'un ordinateur est composée de plusieurs types de mémoires, essentiellement :

- **Mémoire statique :** zone de la mémoire où sont stockés les données qui ont la même durée de vie que le programme (variables globales).
- **Mémoire automatique :** zone de la mémoire appelée Pile d'exécution où sont stockés les paramètres, les adresses et valeurs du retour et variables locales des fonctions empilées. Cette mémoire est gérée automatiquement par le compilateur (réservation et libération).
- **Mémoire dynamique :** zone de la mémoire appelée **TAS** dans laquelle le programmeur peut réserver explicitement de la place. Il devra la libérer explicitement.

L'allocation dynamique permet d'allouer un emplacement dans la mémoire appelée **TAS**. Les variables du **TAS** ne sont pas déclarées. Elles sont créées par une instruction spécifique. Elles ne portent pas de nom, mais on peut y accéder par leur adresse. Elles peuvent être utilisées partout jusqu'à leur destruction.

Pour créer une variable dans le tas, on utilise l'instructuin **ALLOUER** suivi du type de la variable qu'on veut créer.

Exemple :

Var

P : ^ entier ; // P est une variable de type pointeur

Début

Allouer (P) ;

/* Allouer réserve de la mémoire dans le TAS. P contient alors l'adresse de la mémoire allouée (NIL si aucun espace mémoire n'est disponible pour allocation*/

Adresse	Case memoire	Zone
0x3fffd10		
0x3fffd11	0x3fffd15	P
0x3fffd12		
0x3fffd13		
0x3fffd14		
0x3fffd15		Réservée pour P
0x3fffd13		
0x3fffd16		
0x3fffd17		

Remarque : Allouer réserve de la mémoire, mais ne l'utilise pas.

Si on ne libère pas la mémoire, elle peut être saturée et le programme peut se planter.
Pour éviter la saturation de la mémoire, il est donc nécessaire de **désallouer** les emplacements alloués dans le TAS. On utilise donc l'instruction **LIBÉRER**.

Attention : il est interdit de :

- Libérer un pointeur de valeur NIL.
- Libérer une zone déjà libérée.
- Accéder à une zone mémoire libérée.
- Libérer de la mémoire allouée automatiquement par le compilateur (variable nommée)).

Exemple :

Var

P : ^ entier ;

Début

Allouer (P) ; // Réservation de la mémoire dans le TAS pour P
… // Utilisation du pointeur P
Libérer (P) ; // Récupération de la mémoire réservée pour P dans le TAS

5. Arithmétique des pointeurs

Les seules opérations arithmétiques valides sur les pointeurs (de type entier par exemple) sont :

- L'addition d'un entier à un pointeur. Le résultat est un pointeur de même type que le pointeur de départ.
- La soustraction d'un entier à un pointeur. Le résultat est un pointeur de même type que le pointeur de départ.
- La différence de deux pointeurs pointant sur des objets de même type. Le résultat est entier.

Si **X** est un entier et **P** un pointeur sur un objet de type **TYPE1** donnée, l'expression **P+X** désigne un pointeur sur un objet de type **TYPE1** dont la valeur de l'adresse égale à l'adresse de **P** incrémentée de **X** fois taille du type **TYPE1**.

6. Pointeur vers un enregistrement

Nous pouvons utiliser un pointeur pour mémoriser l'adresse d'un enregistrement.

Exemple :

```
Type Personne = enregistrement
        Nom : chaine de caractères ;
        Num : entier ;
 Fin ;
Var
        Pointpers: ^ Personne ;
        Pers : Personne ;
```

On peut accéder aux champs de l'enregistrement **pers** en utilisant l'opérateur ^.

```
Début
        Pers.Nom ← "Ali" ;
        Pers.Num ← 12 ;
        Pointpers ← &Pers ; // Pointpers reçoit l'adresse de l'enregistement Pers
        Écrire (pointpers ^.Nom) ;
        Écrire (pointpers ^.Num) ;
Fin ;
```

7. Pointeurs et tableaux

Tout tableau, en algorithmique est en fait un pointeur constant.

Exemple :

```
    Var
            T: tableau [1..10] d'entier ;
```

T est un pointeur constant dont la valeur est l'adresse du premier élément du tableau. De ce fait :

- **T^** contient la valeur de T[1].
- **(T+1)^** contient la valeur de T[2].
- **(T+2)^** contient la valeur de T[3].
- Etc ;

En générale, (T+i)^ contient la valeur T[i+1].