

I. Les structures

Une structure est un type que l'on déclare et qui permet de stocker dans une seule entité plusieurs éléments de type éventuellement différents.

1. Déclaration et utilisation d'une structure

Pseudo-code	Pascal
<pre>Type <i>NomStructure</i> = Structure <i>attribut₁</i> : <i>TypeAttribut₁</i> <i>attribut₂</i> : <i>TypeAttribut₂</i> ... <i>attribut_n</i> : <i>TypeAttribut_n</i> finstructure <i>variable</i> : <i>NomStructure</i></pre>	<pre>type <i>NomStructure</i> = record <i>attribut₁</i> : <i>TypeAttribut₁</i>; <i>attribut₂</i> : <i>TypeAttribut₂</i>; ... <i>attribut_n</i> : <i>TypeAttribut_n</i> end; <i>variable</i> : <i>NomStructure</i>;</pre>
{ accès à un attribut d'une variable de type structure }	
<pre><i>variable</i>.<i>attribut₁</i> ← valeur</pre>	<pre><i>variable</i>.<i>attribut₁</i> := valeur</pre>

2. Opérations et instructions

- $\leftarrow$: copie toute la structure
- $=$: vrai si tous les attributs identiques
- $\neq$: vrai si un des attributs différent

3. Bonne pratique : principe d'encapsulation

La bonne pratique consiste à créer des accesseurs pour lire (get / obtenir) ou écrire (set / fixer) les données contenues dans la structure. Si « l'architecture » de la structure venait à être modifiée, les accesseurs devraient être modifiés aussi pour que l'accès soit garanti de façon transparente pour l'utilisateur de la structure via ces accesseurs.

L'utilisateur de la structure ne doit pas avoir à écrire `variable.attribut1`, mais plutôt `obtenirAttribut1(variable)`.

4. Particularité des enregistrement en Pascal

a. L'instruction with ... do

```
with var do
begin
    { code }
end;
```

Dans un programme on a une variable `var` de type structure `NomStructure`. On peut écrire dans le programme le code ci-contre.

Dans la zone { code }, l'accès aux attributs de `var` se fait directement. On tape `attribut1` au lieu de `var.attribut1`.

b. Enregistrement conditionnel

Dans un enregistrement, le nom et le type du dernier attribut peut être conditionné par la valeur de l'avant dernier. Pour ce faire, on utilise le code ci-dessous.

```
type   TDenombrable = (valeur1, ..., valeurk);

NomStructure = record
    attribut1 : TypeAttribut1;
    attribut2 : TypeAttribut2;
    ...
    case attributn-1 : TDenombrable of
        valeur1 : (attributn,1 : TypeAttributn,1);
        valeur2 : (attributn,2 : TypeAttributn,2);
        ...
        valeurk : (attributn,k : TypeAttributn,k)
    end; { ce end ferme à la fois le case et le record }
```

II. Les types fonction et procédure (hors programme)

On peut déclarer des types fonction et procédure permettant de passer en argument des fonctions ou des procédures. Par exemple, ci-dessous, depuis le programme principal, on passe uneFonction puis uneAutreFonction en argument d'uneProcédure. uneProcédure utilisant la fonction passée en argument, son « résultat » devra logiquement en dépendre.

```
type
    TFonction = function(p1 : Type; ...; pn : Type) : Type;
    TProcédure = procedure([var] p1 : Type; ...; [var] pn : Type);

function uneFonction(p1 : Type; ...; pn : Type) : Type;
begin
    ...
end;

function uneAutreFonction(p1 : Type; ...; pn : Type) : Type;
begin
    ...
end;

procedure uneProcédure([var] p1 : Type; ...; fonctionParametre : TFonction)
    var uneVar : Type;
begin
    ...
    uneVar := fonctionParametre(p1, ..., pn);
    ...
end;

begin
    ...
    uneProcédure(p1, ..., @uneFonction);
    ...
    uneProcédure(p1, ..., @uneFonction);
    ...
end.
```