

Les unités en Pascal

Christian Lasou, Nour-Eddine Oussous, Éric Wegrzynowski

Licence ST-A, USTL - API1

10 avril 2007

- 1** Introduction
- 2** La syntaxe Pascal
- 3** Exemple : U_CarresMagiques

1 Introduction

2 La syntaxe PASCAL

3 Exemple : U_CarresMagiques

Pourquoi des unités ?

- C'est la possibilité de mettre ensemble un certain nombre d'outils pour la résolution d'un problème

Pourquoi des unités ?

- C'est la possibilité de mettre ensemble un certain nombre d'outils pour la résolution d'un problème
- L'idée est de pouvoir réutiliser ces outils pour d'autres problèmes similaires

Pourquoi des unités ?

- C'est la possibilité de mettre ensemble un certain nombre d'outils pour la résolution d'un problème
- L'idée est de pouvoir réutiliser ces outils pour d'autres problèmes similaires
- Dans d'autres langages ces unités peuvent être génériques (utilisables pour différents types)

Qu'est-ce qu'une unité ?

- C'est un ensemble de
 - constantes
 - de types de données
 - de variables
 - de procédures et/ou fonctions

Qu'est-ce qu'une unité ?

- C'est un ensemble de
 - constantes
 - de types de données
 - de variables
 - de procédures et/ou fonctions
- Qui peuvent être partagés par plusieurs applications ou unités

Qu'est-ce qu'une unité ?

- C'est un ensemble de
 - constantes
 - de types de données
 - de variables
 - de procédures et/ou fonctions
- Qui peuvent être partagés par plusieurs applications ou unités
- PASCAL offre une gamme d'unités prédéfinies. **sysutils** et **math** en sont des exemples.

Qu'est-ce qu'une unité ?

- C'est un ensemble de
 - constantes
 - de types de données
 - de variables
 - de procédures et/ou fonctions
- Qui peuvent être partagés par plusieurs applications ou unités
- PASCAL offre une gamme d'unités prédéfinies. `sysutils` et `math` en sont des exemples.
- On peut définir ses propres unités comme `cartes`.

1 Introduction

2 La syntaxe Pascal

3 Exemple : U_CarresMagiques

Structure d'une unité

Listing

```
unit <identificateur> ;

interface
uses <liste des unités> ; {facultatif}
    {Déclarations publiques}

implementation
uses <liste des unités> ; {facultatif}
    {Déclarations privées}
    {implémentation des procédures / fonctions}

initialization {facultatif}
    {Code d'initialisation facultatif}

finalization {facultatif}
    {Code de finalisation facultatif}
end.
```

Structure d'une unité

- L'en-tête de l'unité commence par le mot réservé `unit` suivi du nom de l'unité

Structure d'une unité

- L'en-tête de l'unité commence par le mot réservé `unit` suivi du nom de l'unité
- Le mot réservé `interface` indique le début de la partie visible pour les autres unités ou applications

Structure d'une unité

- L'en-tête de l'unité commence par le mot réservé `unit` suivi du nom de l'unité
- Le mot réservé `interface` indique le début de la partie visible pour les autres unités ou applications
- Si l'unité utilise d'autres unités, on les liste après le mot réserve `uses`

Structure d'une unité

- L'en-tête de l'unité commence par le mot réservé `unit` suivi du nom de l'unité
- Le mot réservé `interface` indique le début de la partie visible pour les autres unités ou applications
- Si l'unité utilise d'autres unités, on les liste après le mot réserve `uses`
 - juste après le mot réservé `interface`

Structure d'une unité

- L'en-tête de l'unité commence par le mot réservé `unit` suivi du nom de l'unité
- Le mot réservé `interface` indique le début de la partie visible pour les autres unités ou applications
- Si l'unité utilise d'autres unités, on les liste après le mot réserve `uses`
 - juste après le mot réservé `interface`
 - ou juste après le mot réservé `implementation`

Structure d'une unité

- Partie `interface` :

Structure d'une unité

- Partie `interface` :
 - elle commence par le mot réservé `interface`

Structure d'une unité

- Partie `interface` :
 - elle commence par le mot réservé `interface`
 - elle définit ce qui est visible (accessible) à n'importe quelle application ou unité utilisant celle-ci

Structure d'une unité

■ Partie `interface` :

- elle commence par le mot réservé `interface`
- elle définit ce qui est visible (accessible) à n'importe quelle application ou unité utilisant celle-ci
- on peut y déclarer des constantes, des types de données, des variables, des procédures et des fonctions

Structure d'une unité

- Partie `implementation` :

Structure d'une unité

- Partie `implementation` :
 - elle commence par le mot réservé `implementation`

Structure d'une unité

- Partie `implementation` :
 - elle commence par le mot réservé `implementation`
 - tout ce qui est déclaré dans la partie interface est accessible au code de la partie implémentation

Structure d'une unité

- Partie `implementation` :
 - elle commence par le mot réservé `implementation`
 - tout ce qui est déclaré dans la partie interface est accessible au code de la partie implémentation
 - elle peut avoir ses propres déclarations supplémentaires, mais celles-ci ne sont pas accessibles aux programmes ou unités utilisant cette unité

Structure d'une unité

- Partie `implementation` :
 - elle commence par le mot réservé `implementation`
 - tout ce qui est déclaré dans la partie interface est accessible au code de la partie implémentation
 - elle peut avoir ses propres déclarations supplémentaires, mais celles-ci ne sont pas accessibles aux programmes ou unités utilisant cette unité
 - elle peut avoir une clause `uses`

Structure d'une unité

- Partie `implementation` :
 - elle commence par le mot réservé `implementation`
 - tout ce qui est déclaré dans la partie interface est accessible au code de la partie implémentation
 - elle peut avoir ses propres déclarations supplémentaires, mais celles-ci ne sont pas accessibles aux programmes ou unités utilisant cette unité
 - elle peut avoir une clause `uses`
 - Le corps des routines déclarées dans la partie `interface` doit apparaître dans la partie `implementation`

Structure d'une unité

- Partie `initialization` :

Structure d'une unité

- Partie `initialization` :
 - elle commence par le mot réservé `initialization`

Structure d'une unité

- Partie `initialization` :
 - elle commence par le mot réservé `initialization`
 - elle permet d'initialiser des données que l'unité utilise ou rend accessibles au moyen de la partie `interface`

Structure d'une unité

- Partie `initialization` :
 - elle commence par le mot réservé `initialization`
 - elle permet d'initialiser des données que l'unité utilise ou rend accessibles au moyen de la partie `interface`
 - lorsqu'une application utilise une unité, le code de la partie initialisation est exécuté avant toute autre partie du code.

Structure d'une unité

- Partie `finalization` :

Structure d'une unité

- Partie `finalization` :
 - elle commence par le mot réservé `finalization`

Structure d'une unité

- Partie `finalization` :
 - elle commence par le mot réservé `finalization`
 - elle permet de faire le ménage avant la fin de l'application qui utilise l'unité

Structure d'une unité

- Partie `finalization` :
 - elle commence par le mot réservé `finalization`
 - elle permet de faire le ménage avant la fin de l'application qui utilise l'unité
 - lorsqu'une application utilise une unité, le code de la partie finalisation est exécuté avant la fin de l'application.

Utilisation des unités

- Pour utiliser une unité, il suffit de mettre son nom à la suite de la clause `uses` du programme ou de l'unité qui fait appel à elle.

Utilisation des unités

- Pour utiliser une unité, il suffit de mettre son nom à la suite de la clause `uses` du programme ou de l'unité qui fait appel à elle.
- Pour utiliser une entité (constante, type, variable, procédure, fonction, ...) déclarée dans une autre unité, deux possibilités :

Utilisation des unités

- Pour utiliser une unité, il suffit de mettre son nom à la suite de la clause `uses` du programme ou de l'unité qui fait appel à elle.
- Pour utiliser une entité (constante, type, variable, procédure, fonction, ...) déclarée dans une autre unité, deux possibilités :
 - 1 utilisation du nom pleinement qualifié sous la forme
`unite.entite`

Utilisation des unités

- Pour utiliser une unité, il suffit de mettre son nom à la suite de la clause `uses` du programme ou de l'unité qui fait appel à elle.
- Pour utiliser une entité (constante, type, variable, procédure, fonction, ...) déclarée dans une autre unité, deux possibilités :
 - 1 utilisation du nom pleinement qualifié sous la forme `unite.entite`
 - 2 utilisation du seul nom de l'entité sous la forme `entite`, possible uniquement si l'entité de l'unité n'est pas masquée par une autre entité définie ailleurs et portant le même nom.

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`
- On peut ajouter un mot réservé `Initialization` au dessus du mot réservé `end` pour créer une partie `Initialization`

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`
- On peut ajouter un mot réservé `Initialization` au dessus du mot réservé `end` pour créer une partie `Initialization`
- Le nom du fichier contenant l'unité doit être identique à l'identificateur qui suit le mot réservé `unit`

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`
- On peut ajouter un mot réservé `Initialization` au dessus du mot réservé `end` pour créer une partie `Initialization`
- Le nom du fichier contenant l'unité doit être identique à l'identificateur qui suit le mot réservé `unit`
- Avec FREEPASCAL, la compilation d'une unité se fait par la commande `fpc`. Le compilateur produit deux fichiers :

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`
- On peut ajouter un mot réservé `Initialization` au dessus du mot réservé `end` pour créer une partie `Initialization`
- Le nom du fichier contenant l'unité doit être identique à l'identificateur qui suit le mot réservé `unit`
- Avec FREEPASCAL, la compilation d'une unité se fait par la commande `fpc`. Le compilateur produit deux fichiers :
 - `---.ppu` : fichier de description de l'unité

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`
- On peut ajouter un mot réservé `Initialization` au dessus du mot réservé `end` pour créer une partie `Initialization`
- Le nom du fichier contenant l'unité doit être identique à l'identificateur qui suit le mot réservé `unit`
- Avec FREEPASCAL, la compilation d'une unité se fait par la commande `fpc`. Le compilateur produit deux fichiers :
 - `---.ppu` : fichier de description de l'unité
 - `---.o` : contient le code de l'unité

Remarques

- Le mot réservé `end` n'est pas associé à un `begin`
- On peut ajouter un mot réservé `Initialization` au dessus du mot réservé `end` pour créer une partie `Initialization`
- Le nom du fichier contenant l'unité doit être identique à l'identificateur qui suit le mot réservé `unit`
- Avec FREEPASCAL, la compilation d'une unité se fait par la commande `fpc`. Le compilateur produit deux fichiers :
 - `---.ppu` : fichier de description de l'unité
 - `---.o` : contient le code de l'unité
- Les deux fichiers sont nécessaires si l'on veut utiliser l'unité dans un programme

1 Introduction

2 La syntaxe PASCAL

3 Exemple : U_CarresMagiques

Exemple : Interface (1/3)

Listing

```
unit U_CarresMagiques ;

interface

const
    MAX      = 31; // taille maximale des carrés
type
    INDICE = 0..MAX-1;
    TAILLE = 3..MAX;
    CARRE = record
        taille : TAILLE;
        contenu : array[INDICE,INDICE] of INTEGER;
    end;
```



Exemple : Interface (2/3)

Listing

```
// lireCarre(c,n) initialise un carré c de taille n
// avec des données lues depuis l'entrée standard
procedure lireCarre(out c: CARRE; const n: TAILLE);

// ecrireCarre(c) écrit le carré c sur la sortie
// standard
procedure ecrireCarre(const c : CARRE);

// estCarreMagique(c)=vrai si c est un carré magique
//                               faux sinon
function estCarreMagique(c : CARRE) : BOOLEAN;
```

Exemple : Interface (3/3)

Listing

```
// estCarreNormal(c) = vrai si c est un carré normal
//                  = faux sinon
function estCarreNormal(c : CARRE) : BOOLEAN;

// construireCarrePair(c,n) construit un carré
// magique d'ordre pair n dans c
// CU : n <= MAX, n pair
procedure construireCarrePair(out c : CARRE;
                             const n : TAILLE);

// construireCarreImpair(c,n) construit un carré
// magique d'ordre impair n dans c
// CU : n <= MAX, n impair
procedure construireCarreImpair(out c : CARRE;
                                const n : TAILLE);
```



Exemple : Implémentation (1/5)

Listing

```
implementation

  procedure lireCarre(out c : CARRE ;
                     const n : TAILLE );

  var
    i, j : INDICE;
  begin
    c.taille := n ;
    for i := 0 to n-1 do begin
      for j := 0 to n-1 do
        read(c.contenu[i,j]);
      end {for i};
    end {lireCarre};
  end {lireCarre};
```

Exemple : Implémentation (2/5)

Listing

```
// la variable format permet d'assurer que les
// entiers d'une même ligne seront écrits séparés.
procedure ecrireCarre(const c : CARRE);
var
    i,j : INDICE;
    format : CARDINAL;
begin
    format := 2 + floor(ln(c.taille*c.taille)/ln(10));
    for i := 0 to c.taille - 1 do begin
        for j := 0 to c.taille - 1 do
            write(c.contenu[i,j]:format);
            writeln();
        end {for i};
    end {ecrireCarre};
```

Exemple : Implémentation (3/5)

Listing

```
// complementaire(k,a,b) = a+b-k
// CU : a<= k <=b
function complementaire(k,a,b : INTEGER) : INTEGER;
begin
    complementaire := a+b-k;
end {complementaire};

// on suppose écrites les fonctions sommeDiagonale,
// sommeDiagonale2, lignesCorrectes et colonnesCorrectes
function estCarreMagique(c : CARRE) : BOOLEAN;
var
    v: INTEGER;
begin
    v := sommeDiagonale(c);
    estCarreMagique := (v = sommeDiagonale2(c))
        and lignesCorrectes(c,v)
        and colonnesCorrectes(c,v);
end {estCarreMagique};
```



Exemple : Implémentation (4/5)

Listing

```
procedure construireCarreImpair(out c : CARRE; const n : TAILLE);
var
  ligne, colonne : INDICE;
  x : INTEGER;
begin
  c.taille := n;
  colonne := n div 2;
  ligne := colonne + 1;
  c.contenu[ligne,colonne] := 1;
  for x := 2 to n*n do begin
    if (x-1) mod n <> 0 then begin
      ligne := (ligne+1) mod n;
      colonne := (colonne+1) mod n;
    end else begin
      ligne := (ligne+2) mod n;
    end {if};
    c.contenu[ligne,colonne] := x;
  end {for};
end {construireCarreImpair};
```

Exemple : Implémentation (5/5)

Listing

```
procedure construireCarrePair(out c : CARRE; const n : TAILLE);
var
  i,j : INDICE;
  c1, c2 : CARRE;
begin
  // construction du premier carre auxiliaire
  construireC1(c1,n);

  // construction du second carré auxiliaire
  construireC2(c2,c1,n);

  // construction du carré voulu
  c.taille := n;
  for i := 0 to n-1 do
    for j := 0 to n-1 do
      c.contenu[i,j] := 1 + c1.contenu[i,j] + n*c2.contenu[i,j];
    end {construireCarrePair};
  end {construireCarrePair};
```

Exemple : Remarque

Remarque

Noter que plusieurs fonctions/procédures sont définies dans la partie **implementation** mais, pas dans la partie **interface**. Cela signifie que ces fonctions/procédures ne sont pas visibles à l'extérieur de l'unité. C'est le cas par exemple de `complementaire`, `construireC1`, `construireC2`,...

Exemple : Initialisation et finalisation

Listing

initialization

```
writeln(stderr,'*****');  
writeln(stderr,'* [API1] Carrés Magiques (2007) *');  
writeln(stderr,'*****');  
writeln(stderr);
```

finalization

```
writeln(stderr);  
writeln(stderr,'*****');  
writeln(stderr,'* [API1] Merci d''avoir utilisé *');  
writeln(stderr,'* Carrés Magiques (2007) *');  
writeln(stderr,'*****');
```

```
end.
```

Exemple : Utilisation

Listing

```
program verifierCarre;  
uses U_CarresMagiques;  
  
var  
  n : TAILLE;  
  c : CARRE;  
  
begin  
  // lecture de l'ordre du carré  
  read(n);  
  // lecture du carré  
  lireCarre(c,n);  
  ecrireCarre(c);  
  if estCarreMagique(c) then  
    writeln('Carré magique')  
  else  
    writeln('Carré non magique');  
end.
```