Les bases de Delphi

Programmation procédurale en Pascal

Les bases de l'algorithmie : types, variables, branchements et boucles

Ricco Rakotomalala
Université Lumière Lyon 2

Généralités sur la programmation

Algorithmie

• Solution « informatique » relative à un problème

• Suite d'actions (instructions) appliquées sur des données

• 3 étapes principales :

1. saisie (réception) des données
2. Traitements
3. restitution (application) des résultats

Programme

• Transcription d'un algorithme avec une syntaxe prédéfinie

• Delphi – Version Objet du Pascal

• Même principes fondamentaux que les autres langages à objets (Delphi, Java, etc.)

• Delphi introduit des fonctionnalités supplémentaires : la programmation visuelle et la programmation événementielle

Mode d'exécution d'un programme

Langage interprété : + portabilité application ; - lenteur (R, VBA, …)

Langage compilé : + rapidité ; - pas portable

(solution possible : write once, compile anywhere ; ex. Lazarus)

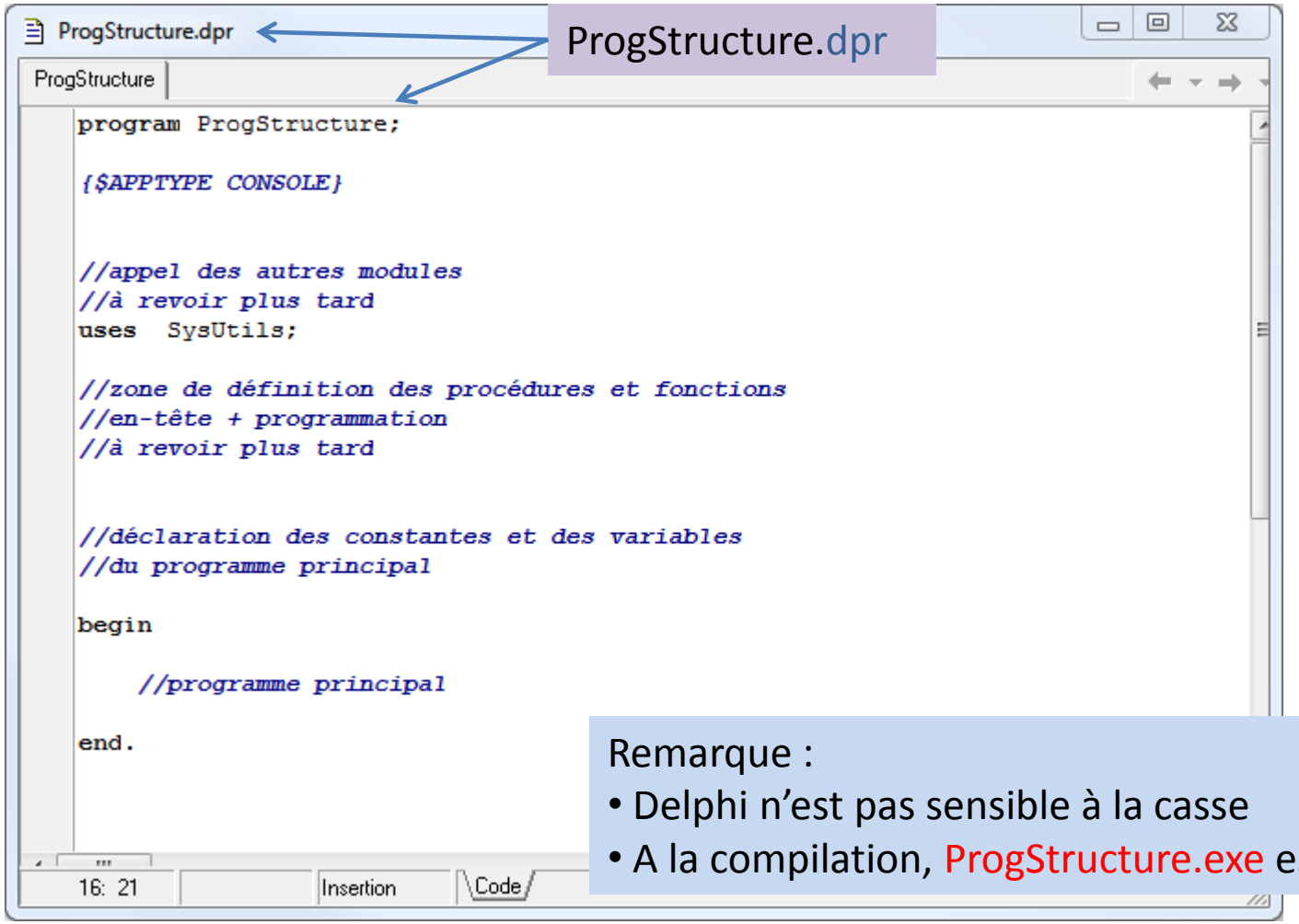
Langage pseudo-compilé : + portabilité plate-forme ; - lenteur (?)

(principe : write once, run anywhere ; ex. Java)

Etapes de la conception d'un programme (Génie Logiciel)

1. Fixer les objectifs et détermination des besoins : que doit faire le logiciel, dans quel cadre va-t-il servir, quels seront les utilisateurs types ? On le rédige souvent avec le commanditaire du logiciel (Remarque : commanditaire = maître d'ouvrage ; réalisateur = maître d'œuvre)
2. Conception et spécifications : quels sont les fonctionnalités du logiciel, avec quelle interface ?
3. Programmation : modélisation et codage
4. Tests : obtient-on les résultats attendus, les calculs sont corrects, y a-t-il plantage et dans quelles circonstances ? (tests unitaires, tests d'intégration, etc.)
5. Déploiement : installer le chez le client (vérification des configurations, installation de l'exécutable et des fichiers annexes, etc.)
6. Maintenance : corrective, traquer les bugs et les corriger (patches) ; évolutive (ajouter des fonctionnalités nouvelles au logiciel : soit sur l'ergonomie, soit en ajoutant de nouvelles procédures)

Organisation d'un programme
(mode console)



Remarque :
- Delphi n'est pas sensible à la casse
- A la compilation, ProgStructure.exe est généré

TYPES DE DONNÉES ET INSTANCIATION

Le type de données définit le type d'opération qui leurs sont applicables

Types simples et transtypage

Entier : integer ; opérateurs : +, - , *, div , mod (ex. 5 div 2 → 2 ; 5 mod 2 → 1 ; etc.)

Réel : double ; opérateurs : +, - , *, / (ex. 5 / 2 → 2.5)

Booléen : boolean (true, false) ; avec : not, and, or (ex. true and not(false) → true)

Chaîne de caractères : string (valeurs entre quotes '…') ; op : méthodes associées au type String (ex. concaténation, pos, copy, etc.)

Transtypage

Réel → chaîne : floattostr() (ex. floattostr(10) → '10 ')

Chaîne → réel : strtofloat() (ex. floattostr('5') → 5 ; floattostr('toto') → ERREUR)

Inttostr() et StrToInt() : entier ↔ chaîne)

Variables et opérateur d'affectation

var permet de déclarer et utiliser une variable
➔var identifiant_variable : type de la variable;

N.B. le contenu d'une variable peut être modifié durant l'exécution du programme

:= permet d'affecter une valeur à une variable
➔ identifiant_variable := valeur;

N.B. : les types doivent être cohérents

valeur peut être le fruit d'une opération

A droite et à gauche de := , une variable ne se comporte pas de la même manière

➔ var pht : double;

➔
```
var pht : double;
    code : integer;
```

➔ pht := 10.5;

➔
```
var pht, pttc : double;
...
pht:= 10.5;
pttc:= pht * 1.196;
```

➔ pttc := 2.0 * pttc;

Constantes

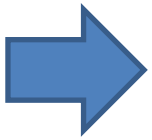
Une constante est une « sorte » de variable dont la valeur est fixée une fois pour toutes à la compilation du programme.

const nom_constante : type de données = valeur;

```
const tva : double = 0.196;
```

Opérateur de comparaison

Un opérateur de comparaison compare des valeurs (variables) de mêmes types, le résultat est un booléen.



```
var test : boolean;

test:= (5 > 3); //→ true
```

= ; <> ; > ; >= ; < ; <=

N.B.

1. Souvent utilisé dans les branchements conditionnels.
2. Permet de comparer des chaînes de caractères

ENTRÉES ET SORTIES

Application console

Saisie et affichage à la console

Read()

Readln ()

Lecture à la « console » : clavier
Tout type

Write()

Writeln()

Affichage à la « console » : écran
Tout type

Un premier programme (pht → pttc)

ProjetTTC.dpr

ProjetTTC

```
program ProjetTTC;

{$APPTYPE CONSOLE}

uses
  SysUtils;

//déclaration de la constante
const tva: double = 0.196;

//variables manipulées dans le programme
var pht, pttc: double;

begin
  //affichage texte d'invitation
  write('entrer pht = ');
  //saisie du prix hors taxe
  readln(pht);
  //calcul du prix ttc
  pttc:= pht * (1 + tva);
  //affichage du prix ttc
  writeln('pttc = ', pttc);
  //blocage de la console
  readln;
end.
```

23: 26 Insertion Code

Saisie des valeurs à manipuler

Calculs

Restitution des résultats

STRUCTURES ALGORITHMIQUES

Branchements et boucles

Branchement conditionnel « if »

Condition : booléen
Opération de comparaison
Peut être complexe (combinaison de conditions)

```
if (condition) then
  instruction si condition vraie
else
  instruction si condition fausse;
```

La partie « else » est facultative
dans ce cas, il faut mettre ; après l'instruction vraie

```
if (condition) then
  begin
    bloc d'instructions si condition vraie;
    …
  end
else
  begin
    bloc d'instructions si condition fausse;
    …
  end;
```

Exemple 1

```
if (codevoiture = 2) then
 begin
  prix:= 60.0 * jour + 1.2 * km;
  writeln('prix = ',prix);
 end;
```

Exemple 2

```
if (codevoiture = 1) or (codevoiture = 2) then
 begin
  if (codevoiture = 1)
   then prix:= 70.0 * jour + 0.5 * km
   else prix:= 60.0 * jour + 1.2 * km;
 end
else erreur:= true;
```

Branchement multiple « case of »

(utilisation d'une variable de contrôle)

Entier ou caractère (type ordinal)

```
case (variable) of
  valeur_1:
            begin
            instruction(s) pour valeur_1;
            end;
  valeur_2:
            begin
            instructions(s) pour valeur_2;
            end;
  …
else
            begin
            instruction(s) pour cas par défaut;
            end;
end;
```

La partie « else » est facultative.
Si une seule instruction, begin … end n'est pas nécessaire

Exemple

```
case code of
 1,3: prix:= 10 * jour + 0.3 * km;
 2: prix:= 25 * jour + 0.1 * km;
 4: prix:= 11 * jour + 0.8 * km;
 else
 erreur:= true;
end;
```

Boucle indicée « for »

Entier (type ordinal)

```
for indice:= val.départ to val.fin do
 begin
    instruction(s);
 end;
```

Ou, dans l'autre sens…

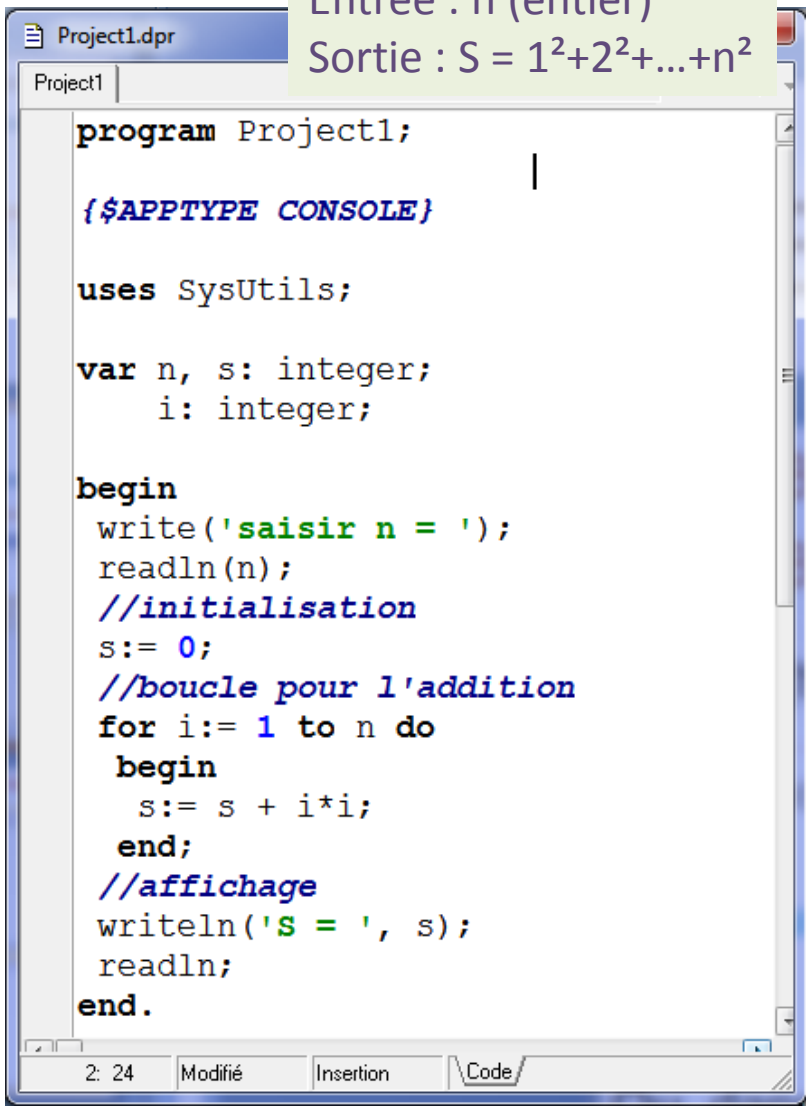
```
for indice:= val.départ downto val.fin do
 begin
    instruction(s);
 end;
```

L'instruction break provoque la sortie de la boucle

Exemple

Entrée : n (entier)

Sortie : $S = 1^2+2^2+\ldots+n^2$

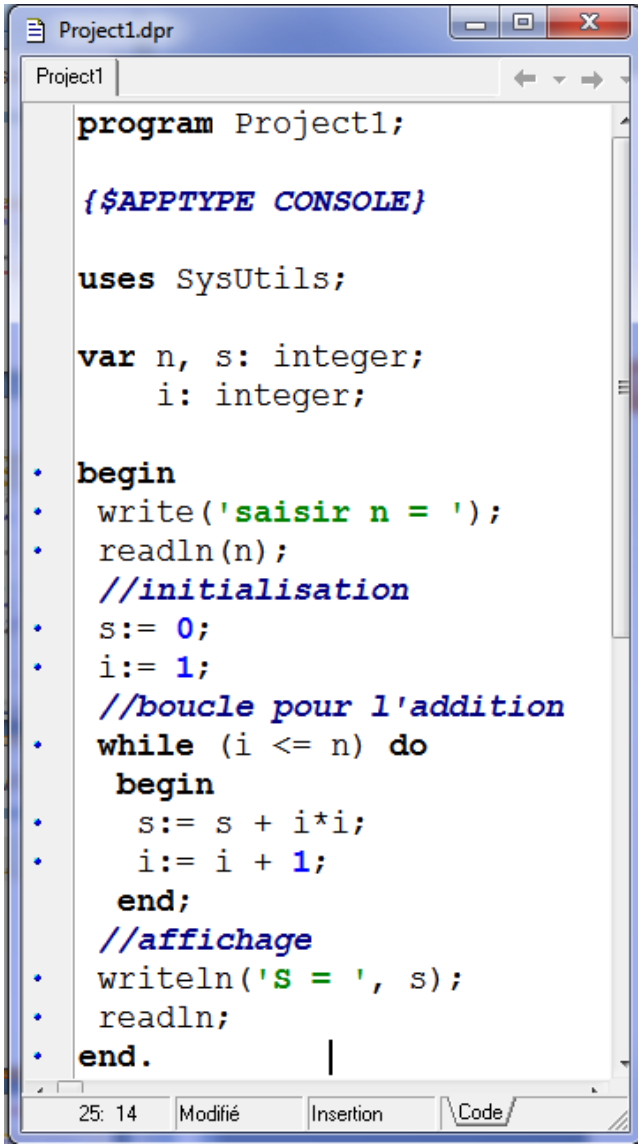


Boucle conditionnelle « while »

Danger ! Boucle infinie

```
while (condition vraie) do
 begin
  instruction(s);
 end;
```

Exemple
Entrée : n (entier)
Sortie : $S = 1^2+2^2+...+n^2$



Boucle conditionnelle « repeat … until »

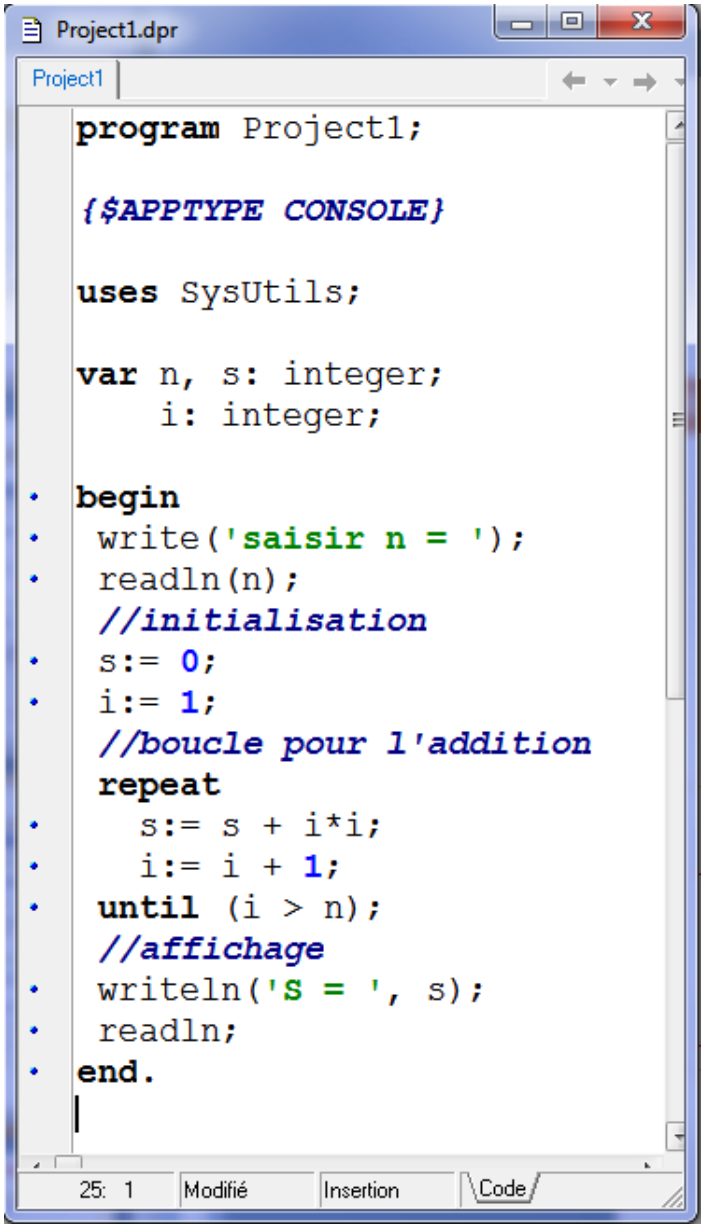
Danger ! Boucle infinie

```
repeat
   instruction(s);
until (condition vraie);
```

Exemple

Entrée : n (entier)

Sortie : S = $1^2+2^2+\ldots+n^2$



FIN…

Les mêmes concepts sont – à peu de choses près – présents dans tous les langages de programmation…