# Le langage Pascal

EISTI


18 octobre 2012

# Présentation

## Historique

- langage de haut niveau, créé dans les années 1970 par Niklaus Wirth ;
- conçu à l'origine pour un but éducatif, proche de l'algorithmique ;
- à l'origine du compilateur TeX de Donald Knuth.

# Présentation

## Caractéristiques

- Format libre : permet une lisibilité du programme.
- Modulaire : peut être découpé en modules séparés, réunissables sous forme de librairie.
- Portable.
- Fortement typé : chaque variable doit être typée, les types sont restreints, proches de la représentation interne du processeur.
- Statiquement typé : la taille des variables ne varie pas pendant l'exécution du programme.

# Techniques de base

## Présentation

- Un **fichier source** Pascal est un fichier texte ayant une extension *".pas"*.
- Le fichier source contenant le programme principal et des fichiers de bibliothèque contenant des fonctionnalités réutilisables.
- Le compilateur (**fpc**) s'occupe de traduire le langage Pascal en langage machine. Il produit un **fichier objet** *".o"* et un exécutable, sans extension.

# Techniques de base

## Compilation

La compilation de déroule en deux étapes

- compilation
- édition de lien

## ligne de compilation

fpc fichier.pas

# Techniques de base

## Compilation

- analyse lexicale : reconnaissance des mots clés du langage
- analyse syntaxique : analyse la structure du programme et sa conformité avec la norme

# Techniques de base

## Edition de liens

- Réunir l'ensemble des fichiers afin de créer un exécutable

# Compilation

## Compilation

```
ls
---> hello.pas
fpc hello.pas
ls
---> hello hello.o hello.pas
```

# Structure d'un programme

## Structure

Un fichier source se compose :

- Une section "program" avec le nom du programme principal
- déclaration des fonctions et procédures
- déclaration des variables du programme principal
- code du programme principal
- des commentaires, évidemment !

# Exemple de programme

```
program First;
uses crt;
var
  nb : integer;
begin
  clrScr();

  write('Saisir un nb : ');
  readln(nb);
  writeln('vous avez saisi ... ', nb, '!!');
  writeln('Au revoir !');
end.
```

# Syntaxe

## Quelques règles...

- Une instruction Pascal se termine par « ; »
- `(* un commentaire *)`

## nomenclature de nommage

- maVariable, procedureCalculIntegral(), MonFichier.pas
- ne pas commencer par des chiffres
- pas de caractères spéciaux (" ", "-", " :", "@"...)
- attention aux mots clés (integer, else, file, case, function, var...)

# Bloc d'exécution

- Un bloc d'exécution (dans une fonction, une condition, une boucle) est encadré par les instruction **begin** et **end ;**
- L'instruction **end** finale (fin programme) s'écrit **end.**

# Présentation

## Types

Le Pascal est un langage typé où l'on doit spécifier un type pour chaque variable, permettant de réserver l'espace mémoire nécessaire :

- entier : integer (2o), shortint (1o), longint (4o), byte (1o)
- booléen : boolean (1bit)
- caractère : char (1o)
- réel : real (6o)
- chaîne : string...
- tableau : array... of...
- structure : record
- ensemble : set
- ...

# Présentation

## Déclaration - Affectation

- Déclaration

```
x : real;
n : integer;
carac : char;
```

- initialisation

```
n := 12;
x := 3.12e8;
carac := 'a';
```

# Présentation

## Constantes

On peut définir des constantes dans un programme : **const**

ex : `const vrai = true;`

Il existe des constantes prédéfinies :

- true, false
- pi
- nil
- maxint (32767)

# Présentation

## Constantes de caractère

| Constante | Description |
|---|---|
| #13#10 | nouvelle ligne |
| #9 | tabulation |
| #11 | tabulation verticale |
| ' '' ' | ' |
| ''''' | '' |

# Présentation

## Fonctions sur les caractères

- ord(*car*) : donne le code ascii de *car*
- chr(*car*) : donne le caractère du code ascii *n*
- succ(*car*) / pred(*car*) : donne le caractère suivant/précédent

# Présentation

## Chaînes de caractères

'ceci est une chaine de caractères'

## Déclaration

Une chaine de caractères est un tableau de caractères, la notion sera abordée plus tard

# Opérateurs et symboles

## Opérateurs et symboles

```
+ - * / div mod
:=
< > <= >= <> (nombres et chaines)
+ (concatenation)
+ * - = <= in (ensembles)
and or xor not
nil
```

'a'+1 a un sens, cela renvoie le caractère qui suit le 'a' dans le codage machine

# Structures conditionnelles, boucles

## Si...Alors...Sinon

```
if (<expression booléenne>) then
begin
...
end
else
begin
...
end;
```

# Structures conditionnelles, boucles

## case

permet de faire plusieurs tests de valeurs d'une même variable.

```
case (<variable>) of
<valeur 1>:
begin
...
end;
<valeur 2>, <valeur 3>, <valeur 7>:
begin
...
end;
<valeur n>:
begin
...
end;
else
begin
...
end;
```

# Structures conditionnelles, boucles

## pour i de 1 a n faire

permet de répéter une série d'instruction un nombre défini de fois. Ce nombre est géré par un compteur qui est incrémenté (ou décrémenté) de 1 à chaque tour de boucle

```
for <init cpt> to (ou downto) <fin> do
begin
...
end;
```

# Structures conditionnelles, boucles

**Exemple :**

```
for i:=1 to 12 do
begin
 writeln(i);
end;
```

# Structures conditionnelles, boucles

## tant que ... faire

```
while (<expression booléenne>) do
begin
...
end;
```

## répéter ... jusqu'à ...

```
repeat
...
until (<expression booléenne>);
```

# Présentation

## C'est du propre !

Il est nécessaire d'organiser les programmes en sous parties et sous-pogramme afin

- d'avoir une meilleure lisibilité
- de limiter les erreurs
- d'organiser des tests ciblés
- de pouvoir réutiliser du code
- de faciliter les corrections/évolutions

# Fonctions

## Fonctions

Une fonction est un sous-programme permettant l'exécution d'une suite d'instructions retournant un résultat. Elle est définie par :

- sa signature (type de retour, nom et paramètres d'entrée)
- une liste d'instructions à l'intérieur de laquelle se trouve l'instruction `nomFonction := résultat`

# Fonctions

## Déclaration

**function** nomFonction(arg1 : type1 ; arg2 : type2 ; ...) : **type retourné ;**
**begin**
liste d'instructions
**end ;**

```
function calculAire(rayon : real) : real;
 var aire : real
begin
 aire := 0;
...
 calculAire := aire;
end;
```

**L'appel se fait alors par appel du nom de la fonction :**

```
resultat := calculAire(12);
```

# Fonctions

## Exemple

Écrire la fonction qui réalise l'exposant entier d'un entier.

# Fonctions

## Exemple

Écrire la fonction qui réalise l'exposant entier d'un entier.

```
function expo(a, n : integer) : integer ;
var   result, i : integer ;
begin
  result := 1 ;
  for i := 1 to n do
  begin
    result := result * a ;
  end ;
  expo := result ;
end ;
```

# Fonctions

## Procédures

Une procédure est utilisée pour faire un traitement sans calcul ou pour retourner plusieurs valeurs calculées :

## Exemple

```
procedure affMenu();
begin
writeln('Faites votre choix...')
writeln('1 - Travailler');
...
writeln('0 - Quitter');
end;
```

**Appel de la procédure :**

```
...
affMenu();
```

# Fonctions

## Exemple

```
procedure aireSurf(rayon : real; var aire : real; var surf : real);
...
end;
```

**Appel de la procédure :**

```
a, s : integer ;
...
aireSurf(12.4, a, s) ;
writeln('aire = ', a, ', surface = ', b) ;
```

# Fonctions prédéfinies

## Lire et écrire

Les fonctions permettant la lecture et l'écriture de données sont :

- `readln` : fait une lecture clavier (la validation de la saisie se fait par la touche entrée)
- `write, writeln` : permettent l'écriture de différents types de données

# Fonctions prédéfinies

## Fonctions numériques standards

- exp(x), ln(x)
- sqr(x), sqrt(x)
- abs(x), odd(x)
- round(x), int(x), frac(x)
- sin(x), cos(x), tan(x)
- ....

# Fonctions prédéfinies

## Fonctions caractères standards

- Char(n)
- concat(ch1, ch2,..., chn)
- copy(ch, deb, n)
- length(ch), pos(mot,ch)
- upCase(ch)
- ....