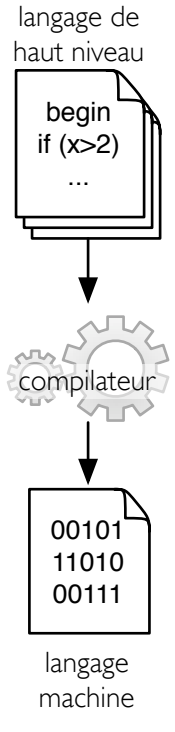
Initiation à la programmation impérative

1. Introduction

L'ordinateur est un outil qui permet de traiter l'information. Son intérêt repose sur son incroyable vitesse – un processeur est aujourd'hui capable de dépasser mille milliards d'opérations par seconde – et sur sa versatilité. En effet, contrairement à la plupart des objets courants (grille-pain, appareil photo, etc.), les fonctionnalités de l'ordinateur ne sont pas figées lors de sa fabrication. Il suffit d'installer des programmes pour étendre les possibilités de l'ordinateur : dessin, montage vidéo, retouche photo, etc.

L'ordinateur est donc polyvalent et rapide, mais il est complètement stupide. La machine ne sait qu'obéir à des ordres simples et précis, qu'elle applique à la lettre, sans aucun esprit critique, ni aucun esprit d'initiative. On dit qu'elle exécute un programme.

Comment créer soi-même un programme ? L'idéal serait de pouvoir parler à la machine, comme dans les films de science-fiction : "ordinateur, calcule-moi la surface d'un cercle de 8 cm de diamètre", mais c'est pour l'instant impossible. En fait l'ordinateur est composé de minuscules transistors (interrupteurs) qui ne ne comprennent que deux états : le courant passe (1), ou le courant ne passe pas (0). Ce langage binaire, ou langage machine, est très loin du langage naturel que nous utilisons. Il faut donc trouver un moyen intermédiaire pour se comprendre.



Les ingénieurs ont trouvé un compromis en 2 étapes : le programmeur utilise un langage simplifié proche de l'anglais, comprenant un vocabulaire très limité (instructions), puis utilise un compilateur qui traduit ce langage de haut niveau en langage machine.

Il existe des centaines de langages de programmation : *ADA, ASP, Basic, C, C++, Delphi, Eiffel, Fortran, Java, Lisp, ML, OCaml, Pascal, Perl, PHP, Prolog, Python, Rebol, Ruby, Smalltalk, Scheme, VHDL*, etc.

Chaque langage a ses particularités, ses domaines d'application favoris, ses forces et ses faiblesses. Il n'existe pas de langage parfait et universel.

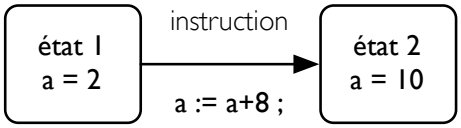
Méthodologie

Quel que soit le langage utilisé, la méthodologie est toujours la même :

1. Poser clairement le problème (objectif)
2. Trouver une méthode de résolution qui puisse s'exprimer en une suite d'instructions élémentaires (= recette de cuisine ou *algorithme*).
3. Transcrire cet algorithme dans le langage choisi
4. Compiler le programme
5. Exécuter le programme compilé (*application*)

La programmation impérative

"En informatique, la programmation impérative est un paradigme de programmation qui décrit les opérations en termes d'états du programme et de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme." (Wikipédia)



La plupart des langages de haut niveau comportent quatre types d'instructions principales : les assignations, les itérations (boucles), les branchement conditionnels (instruction *if*), et le branchement sans condition (instruction *goto* ou appel de procédure). La suite du cours détaille ces 4 types d'instructions et vous montre comment les combiner pour écrire de vrais programmes.

2. Pascal et l'environnement TopPascal

Pour ce cours, nous allons utiliser *Pascal*. C'est un langage simple, lisible et structuré, développé en 1970 par Niklaus Wirth à l'EPFZ. Il est toujours très utilisé à l'heure actuelle, en particulier à travers son successeur orienté objet appelé *Delphi*.

Pour écrire un nouveau programme, on ouvre d'abord une nouvelle fenêtre texte (*File / New*) de l'éditeur TopPascal (sorte de traitement de texte rudimentaire).

La disposition du texte dans cette fenêtre, l'utilisation de minuscules ou majuscules, le choix de la police et la taille des caractères ne changent rien au programme lui-même. Ils ne servent qu'à la lisibilité du code. Il est recommandé d'éviter les lettres accentuées.

Le programme final sera compilé (*Comp+Run* ou *F9*). Si aucune erreur de syntaxe n'est trouvée par le compilateur, ce programme sera immédiatement exécuté. Une nouvelle fenêtre contiendra le résultat de cette exécution.

Les fenêtres texte et résultat peuvent être sauvegardées ou imprimées.

3. Les opérateurs mathématiques

Les opérateurs mathématiques permettent d'effectuer des calculs : addition (+), soustraction (-), multiplication (*), division (/), division entière (div), reste d'une division entière (mod). Par exemple :

```
1+2 ;  1024-256 ;  43 div 4 ;  21 mod 2 ;  etc.
```

Nous verrons plus loin les opérateurs logiques (*not*, *or*, *and*) et les opérateurs relationnels (=, <, >, etc.).

4. Les variables

Dans la plupart des programmes, il est nécessaire de mémoriser des résultats intermédiaires. On utilise pour cela des *variables*.

On peut voir une variable comme un tiroir dans la mémoire de l'ordinateur, dans lequel on peut mettre provisoirement des valeurs. Le nom de la variable est comme l'étiquette sur le tiroir. Il permet de retrouver la valeur.

L'instruction d'affectation permet de placer une valeur dans une variable. En Pascal, la syntaxe de cette opération est la suivante :

```
variable := expression ;
```

Une *expression* est formée à partir d'opérandes (variables, constantes, fonctions) et d'opérateurs. Nous pouvons déjà écrire de simples instructions :

```
base            := 12;
hauteur         :=  6;
aire_triangle   := base * hauteur / 2 ;

message         := 'une erreur s''est produite ' ;
```

Les textes doivent être entourés d'apostrophes. Pour écrire une apostrophe, il suffit d'en taper deux.

5. Les identificateurs

Un identificateur est un mot qui sert à nommer les variables, les constantes, les procédures, les fonctions, et les programmes. Il commence par une lettre suivie d'une combinaison quelconque de lettres, de chiffres et de soulignés. Les mots clés réservés (angl. *reserved keywords*) tels que *Program*, *Var*, *Const*, *While*, *Write*, *Integer*, etc. ne peuvent pas être utilisés comme identificateurs.

```
volume := 3 ;
classe_5c := 24 ;
note_maximum := 6 ;
move_to_trash() ;
```

6. Les commentaires

Des commentaires peuvent être ajoutés entre accolades dans le code. Ils seront ignorés lors de la compilation. Le meilleur moyen pour éviter d'avoir à écrire des commentaires inutiles consiste à utiliser des identificateurs auto-explicites. Les deux exemples suivants produisent le même résultat, mais la lisibilité diffère :

```
q := 3 ;   d := 4 ;
w := d*q/2 ;  {Note : w correspond à l'aire du triangle}

base := 3 ;   hauteur := 4 ;
aire_triangle :=  base*hauteur / 2 ;
```

7. Les instructions de lecture / écriture

Il est souvent nécessaire de communiquer des résultats à l'utilisateur, ou de prendre en paramètre des valeurs rentrées par celui-ci. Il existe pour cela des instructions spéciales de lecture et d'écriture.

La procédure Write() permet d'afficher à l'écran des valeurs numériques ou du texte. La procédure Writeln() est analogue, mais effectue en plus un retour à la ligne. La syntaxe est la suivante :

```
Write (v1, v2, v3, ... , vn)
Writeln (v1, v2, v3, ... , vn)
```

où vi sont des constantes, variables ou des expressions.

Par exemple :

```
Writeln ('La base du triangle vaut ', base) ;
Writeln ('L'aire du triangle vaut ', base*hauteur/2);
```

En précisant *vi : n*, l'affichage se fait dans un champ de n espaces. Pour les valeurs réelles, la syntaxe *vi : n : m* permet de préciser le nombre *m* de chiffres après la virgule que l'on veut afficher. Par exemple :

```
hauteur := 12.34 ;
Writeln ('La hauteur vaut ', hauteur:4:1);
Writeln ('La hauteur vaut environ ', hauteur:3:0);
```

Pour la lecture de données, on utilise la procédure *Read()*:

```
Read (liste_de_variables) ;
```

Les variables sont séparées par des virgules.

Lors de l'exécution de cette instruction, le programme sera interrompu pour permettre à l'utilisateur d'entrer au clavier des valeurs pour les différentes variables mentionnées dans *liste_de_variables*. Les valeurs numériques doivent être séparées par un espace.

La procédure *Readln()* est identique à *Read()*, excepté qu'après la lecture de la dernière variable, le reste de la ligne est ignoré. La touche *Return* doit être appuyée et l'ordinateur effectue alors un retour à la ligne. Par exemple:

```
Write ('Veuillez entrer trois nombres :') ;
Read (x, y, z) ;

Write ('Largeur du rectangle');
Readln(largeur);
```

8. Les blocs d'instructions

Il est possible de regrouper des instructions dans un *bloc*. Les instructions sont séparées par un point-virgule, et le bloc est délimité par les mots réservés *Begin* et *End*.

```
begin
   volume := x*x*x ;
   aire := x*x
end
```

L'indentation (retrait à gauche) du code améliore la lisibilité. Elle se fait en ajoutant 3 espaces en début de ligne à l'intérieur des blocs.

9. La structure d'un programme

Tout programme *Pascal* est composé d'un en-tête, d'une partie déclarations et d'un bloc d'instructions.

```
Program nom_du_programme ;
  { déclarations }
Begin
  { instructions }
End .
```

L'en-tête est composé du mot clé réservé *Program* suivi d'un identificateur.

Dans la partie déclarations, on indique les types, les variables, les fonctions et les procédures que l'on entend utiliser dans le programme.

Les déclarations et type des variables

En *Pascal*, une variable ne peut pas contenir n'importe quoi. On doit préciser le genre d'informations qu'elle va contenir : nombre entier (angl. *Integer*), nombre à virgule (angl. *Real*), booléen (angl. *Boolean*) caractère (angl. *Char*), chaîne de caractères (angl. *String*). C'est ce qu'on appelle le *type* de la variable.

Les types prédéfinis les plus souvent utilisés sont :

Integer	[-32768 .. 32767]	16 bits
LongInt	[-2147483648 .. 2147483647]	32 bits
Real	mantisse d'env. 16 chiffres, |exp.| < 4930	
Boolean	*True* ou *False*	1 bit
Char	256 caractères possibles	8 bits
String	texte de 255 caractères	2048 bits

La déclaration des variables se fait selon le format :

```
Var   liste_de_variables_1 : type_1 ;
      liste_de_variables_2 : type_2 ;
```

Par exemple :

```
Var   base, hauteur :     Integer ;
      rayon, moyenne :    Real ;
      message_erreur :    String ;
```

Les constantes doivent aussi être déclarées comme suit :

```
Const   nom_1 = valeur_1 ;
        nom_2 = valeur_2 ;
```

Le nom est un identificateur. Par exemple :

```
Const   note_max = 6 ;
        g = 9.81 ;
        message = 'Bienvenue dans Pascal' ;
```

Certaines constantes sont prédéfinies et il n'est donc pas nécessaire de les déclarer :

Pi	=	3.141592653589793239
MaxInt	=	32767 (= 2^{15}-1)
MaxLong	=	2147483647 (= 2^{31}-1)

Les instructions

Les instructions sont séparées par des points-virgules et sont exécutées de manière séquentielle.

```
Program exemple ;
Var x : Integer ;
Begin
  volume := x*x*x ;
  aire := x*x
End .
```

Le point après le *End* termine le programme.

10. Les structures de contrôle

Les structures de contrôle permettent de modifier la séquence des instructions. On distingue les instructions conditionnelles et les boucles (itérations).

Les instructions conditionnelles

Dans la vie de tous les jours on utilise des actions conditionnelles en permanence. Par exemple : "*Si il fait beau je vais me promener, sinon je lis un livre à la maison.*".

L'instruction *if* permet d'exécuter un énoncé seulement si une certaine condition est vraie (angl. *true*). Si la condition est fausse (angl. *false*), alors rien n'est exécuté, ou alors l'instruction qui suit le mot réservé *else* est exécutée. En Pascal la syntaxe est la suivante :

```
If condition Then énoncé ;
If condition Then énoncé_si_vrai Else énoncé_si_faux ;
```

Les conditions s'écrivent au moyen d'opérateurs logiques et relationnels.

Opérateurs logiques :

Not	négation (non)
And	conjonction (et)
Or	disjonction (ou)

Opérateurs relationnels :

=	égalité
<>	inégalité
<	infériorité stricte
<=	infériorité
>	supériorité stricte
>=	supériorité

Par exemple :

```
If (temps = beau) Then
  aller_se_promener() ;
Else
  lire_un_livre() ;
```

Si une condition contient plusieurs instructions, il faut rajouter *Begin* et *End* :

```
If (x>0) Then
  Begin
    volume := x*x*x ;
    aire := x*x
  End
Else
  Writeln ('Erreur :  valeur nulle ou négative');
```

Les instructions *If* peuvent être imbriqueés. En cas d'ambiguïté, *Else* se rapporte au dernier *If*.

Les itérations

Les itérations, aussi appelées boucles (angl. *loops*), ou instructions répétitives, permettent de répéter une ou plusieurs instructions.

L'instruction *for* permet d'exécuter un énoncé un nombre de fois déterminé à l'avance. La syntaxe est la suivante :

```
For variable := v1 To v2 Do énoncé ;
For variable := v1 Downto v2 Do énoncé ;
```

La variable (de type entier) sert de compteur. Elle prend la valeur initiale *v1*. A chaque passage, elle est incrémentée (+1) / décrémentée (-1) d'une unité. L'énoncé est réexécuté jusqu'à ce que la valeur de la variable soit égale à la valeur *v2*.

Par exemple: "Je fais 10 fois le tour de la piste avant de me reposer.".

```
For n := 1 To 10 Do
  Writeln ('Il reste encore ', 10-n, ' tours à faire.') ;
```

L'instruction *While* (tant que) permet de faire des itérations sans en connaître le nombre à l'avance. La syntaxe est la suivante :

```
While condition Do énoncé ;
```

Par exemple : "Tant que le stock n'est pas vide, prendre un objet."

```
While (stock > 0) Do
 stock := stock-1;
```

Pour éviter des boucles infinies, il faut veiller à ce que la condition puisse prendre la valeur *false* lors de l'exécution de l'énoncé.

L'instruction *Repeat* permet d'exécuter une instruction ou un bloc d'instructions jusqu'à ce que la condition soit vraie. La boucle est donc obligatoirement exécutée une fois. La syntaxe est la suivante :

```
Repeat instructions Until condition ;
```

Par exemple : "J'emprunte la voiture des mes parents sans permission jusqu'à ce que je me fasse attraper.".

```
Repeat emprunter_voiture() Until parents_au_courant ;
```

11. Procédures et fonctions

Pascal permet de définir des *procédures* et des *fonctions*. (Les fonctions sont des procédures qui retournent une valeur.) Ceci dépasse le cadre de ce cours, mais voici quand même deux exemples :

```
Procedure print_value(var compteur:  Integer);
begin
  Writeln ('Le compteur vaut :', compteur) ;
end;

Function get_next(compteur : Integer): Integer;
begin
  {instructions}
end;
```

C'est grâce à ces fonctions que l'on peut créer des programmes avec interface graphique : lorsque l'utilisateur clique sur un bouton, la fonction correspondante est lancée, puis le résultat est retourné à l'utilisateur.

Annexe A : Fonctions mathématiques standards

Abs (x)	valeur absolue de x
Sqr (x)	carré de x
Sqrt (x)	racine carrée de x
Trunc (x)	partie tronquée de x
Round (x)	valeur arrondie de x
uRandom (n)	val. entière aléatoire entre 0 et n-1
uRandom1n (n)	val. entière aléatoire entre 1 et n
uRandomReal (a,b)	valeur réelle aléatoire entre a et b
Odd (n)	*True* si n est impair, *False* si n est pair

Par exemple :

```
y : = -13 ;
x : = Abs(y) ;
If Odd (x) Then
  Sqrt (x+1) ;
Else
  Sqrt (x) ;
```

Annexe B : Autres fonctions utiles

ReadChar	lit un caractère au clavier
KeyPressed (c)	*True* si une touche du clavier a été appuyée (c est du type Char)
Button	*True* si le bouton de la souris a été pressé
TickCount	retourne le temps qui s'est écoulé depuis le démarrage du système (en 1/1000 de secondes)
ClearText	efface la fenêtre texte

Par exemple :

```
Program demo ;
Var   time_start, time_end, time_elapsed : LongInt ;
      lettre : Char ;

Begin

  ClearText ;
  time_start := TickCount ;

  Writeln ('Tapez la lette c pour continuer.') ;
  Repeat Until KeyPressed (lettre) = 'c' ;

  Writeln ('Veuillez appuyer un bouton de la souris') ;
  Repeat until Button ;

  time_end := TickCount ;
  time_elapsed := (time_end - time_start) / 1000 ;
  Write ('L'exécution de ce programme a duré ') ;
  Writeln (time_elapsed:0:3,' secondes') ;

End .
```