Présentation résumée du langage Pascal et de l'algorithmique

1998-1999

Professeur : Bertrand LIAUDET

Version du 30 septembre 1998

0. Sommaire

1. Présentation

1.1. Définition de la programmation

Le langage Pascal est un langage de programmation.

La programmation c'est l'élaboration et la codification de la suite des opérations formant un programme, c'est-à-dire l'ensemble ordonné et formalisé des opérations suffisantes (et si possible nécessaires) pour obtenir un résultat.

1.2. Quelques programmes simples

Le programme le plus simple c'est celui qui affiche du texte à l'écran

```
Program LePlusSimple;
begin
    writeln ('bonjour');
end.
```

Ce programme n'est pas très intéressant parce qu'il fait toujours la même chose.

Voyons un programme qui calcule de carré d'une valeur :

```
Program CalculCarre;
begin
    a : real;
    writeln ('entrez une valeur');
    readln (a);
    writeln ('carre de ', a, ' = ', a*a);
end.
```

1.3. Premières notions

1.3.1. Les Variables

Une variable est une mémoire, c'est-à-dire un dispositif capable de recevoir, de conserver et de fournir une information autant de fois souhaitées.

Les variables ont :

- un nom;.
- un type : il fixe les valeurs que peut prendre la variable et la manière dont l'information est codée;
- une adresse : c'est la localisation de la variable dans la mémoire (on ne s'y intéressera pas);
- une valeur.

On peut les représenter par un rectangle au-dessus duquel on écrit leur nom et leur type, dans lequel on écrit l'information conservée par la variable et en dessous duquel on écrit l'adresse de la variable.

Le nom, le type et l'adresse d'une variable sont fixés une fois pour toutes. Seule la valeur peut être modifiée.

1.3.2. Les déclarations

Dans le programme, on déclare les variables qu'on utilise. Déclarer signifie donner le nom et le type. L'adresse de la variable sera choisie par le programme au moment de son utilisation. Les valeurs de la variable sont définies dans le programme.

1.3.3. Readln et writeln : les procédures d'entrée-sortie

Les procédures d'entrées/sorties permettent au programme de communiquer avec l'extérieur, le plus souvent avec l'utilisateur, c'est-à-dire le plus souvent avec le clavier et l'écran.

1.3.3.1. Écrire à l'écran : Writeln

La procédure :

- **Writeln (liste d'expression)**

permet d'écrire à l'écran le contenu des expressions listées. L'affichage commence à partir de la position courante du curseur de l'écran puis fait passer le curseur de l'écran à la ligne (au début de la ligne suivante).

Toutes les expressions sont des paramètres en entrée.

Exemple Pascal

```
Writeln ('le carre de ');                    (* affiche : 'le carre de ' *)
Writeln (a);                                 (* affiche  la valeur de a)
Writeln (' = ');                             (* affiche : ' = ' *)
Writeln (a*a);          (* affiche le résultat de l'expression a * a *)

Writeln ('le carre de ', a, ' = ', a*a);          (* affiche :  1.234 *)
```

1.3.3.2. Lire au clavier : Read, Readln

La procédure :

- **Readln (liste de variables);**

permet de donner des valeurs à une liste de variables à partir du clavier. La valeur donnée au clavier est affichée à l'écran.

Exemples :

```
Readln (a);
Readln (a, b);
```

1.4. Structure du programme

Première présentation de la structure type d'un programme Pascal :

```
Program NomDuProgramme;

(* Programme principal *)
Begin
    (* zone des instructions du programme principal *)
end.
```

1.4.1. Autre exemple : programme de résolution d'une équation

Programme Pascal :

```
Program Equation;
Var
    A, B : real;
Begin
    writeln ('entrez l'équation, d'abord a, puis b');
    readln (a);
    readln (b);
    if A = 0 then begin
        if B = 0 then begin
            writeln ('il y a une infinité de solutions');
        end else begin
            writeln ('il n'y a pas de solution');
        end;
    end else begin
        writeln ('la solution est : ', -B/A);
    end;
end.
```

Dans cet exemple, on lit la valeur de deux variables, "a" et "b", par l'intermédiaire du clavier et de la procédure readln. Puis on traite les différents cas en fonction des valeurs de a et de b. Selon les cas, on affiche à l'écran le résultat avec la procédure writeln.

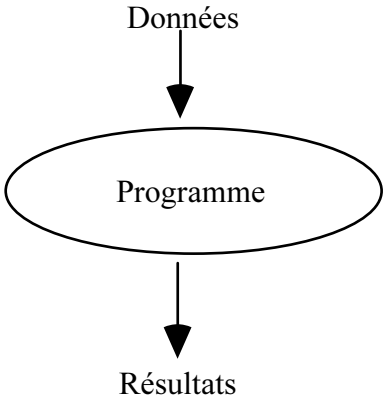
Avec cet exemple, on a le principe même de l'informatique. L'informatique est la science du traitement de l'information.

Traiter l'information c'est :

- récupérer les données à traiter dans un format adéquat
- traiter les données
- renvoyer les résultats dans un format adéquat

Un programme fait toujours cela. Analyser un programme, consiste à déterminer quelles sont les données à traiter, quels sont les traitements qu'on leur applique, quels sont les résultats obtenus.

Schématiquement, un programme c'est :



2. La notion de procédure

Un programme, quelle que soit sa complexité ou sa simplicité, est un système de traitement de l'information, c'est-à-dire un système traitant des informations fournies **en entrée** et produisant des informations **en sortie**.

Ce système, on va l'appeler **procédure**.

Le concept algorithmique de procédure regroupe à la fois les concepts de programme, de procédure et de fonction du langage pascal.

2.1. Readln et writeln sont des procédures.

On voit sur ces exemples qu'une procédure à un nom et des paramètres qui sont en entrée et/ou en sortie.

Dans le cas du Readln, les paramètres sont en sortie. Quand j'écris Readln (a), je produis la variable "a", avec sa valeur, en sortie. En entrée, il y a le clavier et l'utilisateur. Le clavier est un périphérique d'entrée.

Dans le cas du Writeln, les paramètres sont en entrée. Quand j'écris Writeln ('valeur de a = ', a), je fournis deux informations : le texte 'valeur de a = ' et la variable 'a'. En sortie, il y a l'écran et l'utilisateur. L'écran est un périphérique de sortie.

2.2. Cas du programme principal

Dans les exemples traités, calcul du carré et résolution d'une équation du premier degré, quelles sont les entrées et les sorties des programmes principaux ? Dans les deux cas, l'entrée c'est le clavier, la sortie c'est l'écran. Les informations en entrée (les données) sont fournies au clavier, les informations en sortie (les résultats) sont fournies à l'écran.

2.3. Transformation des exemples avec procédure

Dans les deux exemples traités, on a tendance à se dire que l'entrée du programme "carré" c'est un réel quelconque et la sortie c'est le carré de ce réel, et que l'entrée du programme "équation" c'est le couple de réels (a, b) et le résultat c'est le nombre de solution et la valeur des solutions.

2.3.1. Procédure équation

```
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
Begin
    if A = 0 then begin                                    (* test : si *)
```

```
            if B = 0 then begin                          (* test : si *)
                NbSol := -1;
            end else begin                               (* test : sinon *)
                NbSol := 0 ;
            end;                                         (* test : fin de si *)
        end else begin
            NbSol := 1;                                  (* affectation *)
            X := -B/A;                                   (* affectation *)
        end;                                             (* fin de si *)
    end;            (* fin de procédure *)
```

La première ligne :

```
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
```

est appelée : entête de la procédure.

A et B sont des paramètres en entrée.

X et NbSol sont des paramètres en sortie. Ils sont précédés du mot-clé : "var".

Entre le "Begin" du début de la procédure et le "end;" de la fin de la procédure, se situe ce qu'on appelle le corps de la procédure.

2.3.2. Procédure carré

```
Procedure CalculCarre (A : real; Carre : real);
begin
    Carre := A * A;
end;
```

Remarquons que dans ce cas, on n'a qu'un seul paramètre en sortie.

Quand on a qu'un seul paramètre en sortie, le Pascal permet de renvoyer la valeur de ce paramètre dans ce qu'on appelle une fonction :

```
Function Carre (A : real) : real;
begin
    return A * A;
end;
```

Notez qu'on précise alors le type de l'entête de la fonction. Dans notre exemple, c'est un réel.

2.4. Procédure et programme principal

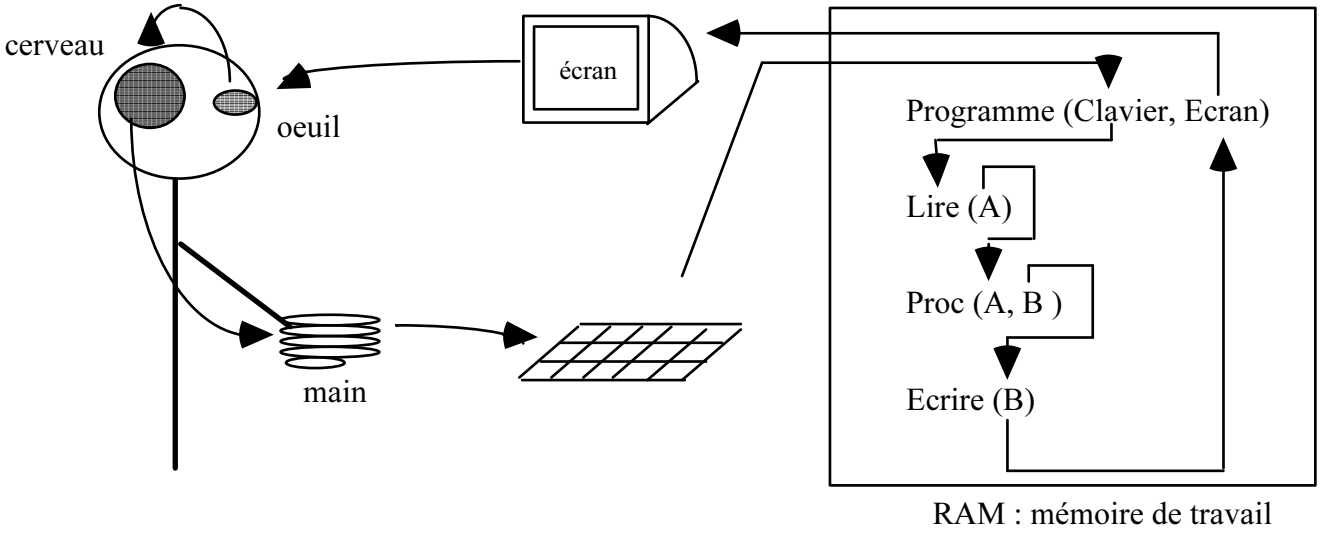
Comment maintenant intégrer une procédure dans un programme principal?

On a déjà dit que le programme principal est une sorte de procédure qui traite des informations fournies en entrée, les données, qui proviennent en général du clavier, pour aboutir à des informations en sortie, les résultats, qu'on affiche en général à l'écran.

Un programme ressemble donc schématiquement à cette suite d'instructions :

```
Programme (Clavier, Ecran)
Début
    Lire (a);
    Procedure (a, b);
    Ecrire (b);
Fin.
```

On peut schématiser ainsi la circulation de l'information :



Le programme Pascal appelant les procédures Equation et Calcul carré devient :

```
Program Equation;
Var
    a, b : real;

(* déclaration des procédures *)
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
Begin
    if A = 0 then begin                              (* test : si *)
        if B = 0 then begin                          (* test : si *)
            NbSol := -1;
        end else begin                               (* test : sinon *)
            NbSol := 0 ;
        end;                                         (* test : fin de si *)
    end else begin
        NbSol := 1;                                  (* affectation *)
        X := -B/A;                                   (* affectation *)
    end;                                             (* fin de si *)
end;            (* fin de procédure *)
```

```
(* programme principal *)
Begin
    writeln ('entrez l'équation, d'abord a, puis b');
    readln (a);
    readln (b);
    Equation (X, Y, N, Res);
    if N = -1 then begin
        writeln ('il y a une infinité de solutions');
    end else begin
        if N = 0 then begin
            writeln ('il n'y a pas de solution');
        end else begin
            writeln ('la solution est : ', -X/Y);
        end;
    end;
end.
```

Remarquons les blocs d'instructions.
Cas du programme de calcul du carré :

```
Program CalculCarreAvecProc; (* attention, le programme doit avoir
Var                 un nom différent de celui des procédures *)
    a : real;

(* déclaration des procédures *)
Procedure CalculCarre (A : real; Carre : real);
begin
    Carre := A * A;
end;

(* programme principal *)
Begin
    Readln (X);
    CalculCarre (X, Y);
    Writeln ('carre de ', X, ' = ', Y);
end.
```

Cas d'une fonction :

```
Program CalculCarre;
Var
    X : real;
```

```
(* déclaration des procédures *)
Function Carre (A : real) : real;
begin
    return A * A;
end;

(* programme principal *)
Begin
    Readln (X);
    Writeln ('carre de ', X, ' = ', Carre (X));
end.
```

2.5. Structure type du programme Pascal

Première présentation de la structure type d'un programme Pascal :

```
Program NomDuProgramme;


Var
    (* zone pour déclarer les variables globales *)

(* Procédures et fonctions *)
    (* zone pour déclarer les procédures et les fonctions *)

(* Programme principal *)
Begin
    (* zone des instructions du programme principal *)
end.
```

3. Langage algorithmique

On n'écrira pas directement les programmes, les procédures et les fonctions en Pascal mais on réfléchira à l'aide d'un langage algorithmique.

3.1. Structure Algo du Writeln

```
afficher (x);
```

ou encore :

```
écran <- (x);
```

3.2. Structure Algo du Readln

```
lire (x);
```

ou encore :

```
x <- clavier;
```

3.3. Structure Algo du test :

```
condition ? instructions | instructions ¿
```

3.4. Structure Algo de l'affectation :

```
var <- expression;
```

3.5. Structure Algo des procédures et des fonctions

3.6. Exemples d'algorithmes :

3.6.1. Procedure Carre

```
CalculCarre (x<-, carre ->) [ carre <- x*x ]
```

3.6.2. Fonction Carre

```
Carre (x<-) [ carre <- x*x ]
```

3.6.3. Procedure Equation

Equation (a<-, b<-, x->, nbsol ->) **[** a=0? b=0? nbsol = -1 / nbsol <- 0 d / x <- -b/a; nbsol <- 0 d **]**

3.6.4. Programme principal

Programme (clavier<-, ecran->) **[** lire(a); lire(b), Equation(a, b, x, ns); affichez(ns); ns=1? affichez(x) /d **]**

3.6.5. Fonction qui calcule la surface d'un cercle

SurfCercle (R<-) **[** SurfCercle <- Pi * R * R **]**

3.6.6. Fonction qui calcule la surface d'un rectangle

SurfRectangle (Long <-, Larg <-) **[** SurfRectangle <- Long * Larg **]**

3.6.7. Procedure de résolution d'une équation du second degré

Equation2 (a<-, b<-, c<-, x1->, x2 ->, nbsol ->) **[** a=0? equation (b, c, x1, nbsol); x2 <- x1/ delta <- b*b -(4*a*c); delta > 0 ? nbsol = 2; x1 <- (-b -racine(delta))/ (2*a); x2 <- (-b + racine (delta)) / (2 * a) ; / delta = 0 ? nbsol <- 1; x1 <- -b / (2*a); x2 <- x1; / nbsol <- 0 d d **]**

4. La boucle

La boucle permet de répéter plusieurs fois une même série d'instructions.

Il y a 3 sortes de boucle en Pascal:

- La boucle avec compteur
- La boucle tant que
- La boucle jusqu'à ce que

4.1. La boucle avec compteur : FOR

4.1.1. Somme des N premiers nombres

```
Program BoucleFor;
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    for i := 0 to N do begin
        Som := Som + i;
    end;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

ou alors :

```
Program BoucleForDownto
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    for i := N downto 0 do begin
        Som := Som + i;
    end;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

4.1.2. Structure Pascal

```
for i := début (to ou downto) fin do begin
    instructions;
end;
```

"début" et "fin" sont des expressions contenant des valeurs entières.

4.1.3. Structure Algo

```
  fin
{i:début  instructions }
```

pour des raisons typographiques, on écrira aussi dans ce polycopié :

```
{(i:début à fin) instructions }
```

4.1.4. Procédure Pascal

```
Function Somme (N : integer) : integer;
(* calcul de la somme des N premiers nombres*)
Var
    i, Som : integer;
Begin
    Som := 0;
    for i := 0 to N do begin
        Som := Som + i;
    end;
    return Som;
end.
```

```
Somme (N<-) [ som<-0; { (i:0 à N) som <- som + 1 } ]
```

4.2. La boucle tant que : WHILE

4.2.1. Exemple Pascal

```
Program BoucleWhile;
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    i:=0
    while i < N do begin
        i := i + 1;
        Som := Som + i;
    end;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

4.2.2. Structure Pascal

```
while condition do begin
    instructions;
end;
```

Une condition est une expression dont la valeur est de type booléen. Le chapitre 3 décrit la structure générale d'une expression.

4.2.3. Structure Algo

```
{/condition instructions }
```

pour des raisons typographiques, on écrira aussi dans ce polycopié :

```
{(/condition) instructions }
```

4.3. La boucle jusqu'à ce que : REPEAT

4.3.1. Exemple Pascal

```
Program BoucleRepeat;
(* calcul de la somme des N premiers nombres*)
Var
    i, N, Som : integer;
Begin
    write ('entrez un entier : ');
    readln (N);
    Som := 0;
    i:=0
    repeat
        SOM := SOM + i;
        i := i + 1;
    until  i > N;
    writeln ('la somme des ', N,' premiers nombres vaut : ', Som );
end.
```

4.3.2. Structure Pascal

```
repeat
    instructions;
```

```
    until condition ;
```

4.3.3. Structure Algo

```
{ instructions }/condition
```

pour des raisons typographiques, on écrira aussi dans ce polycopié :

```
{ instructions (/condition)}
```

4.4. Exemple classique : le PGCD

```
Function PGCD (A, B : integer) : integer;
(* calcul du PGCD de deux nombres *)
Var
    i, Som : integer;
Begin
    while A != B do begin
        if (A > B) then begin
            A := A - B;
        end else begin
            B := B - A;
        end;
    end;
    return A;
end.
```

```
PGCD (A<-, B<-) [ {/(A!=B)  A >B ? A<-A-B | B<B-A ¿ }PGCD<-A]
```

5. Les débranchements

5.1. présentation

Les "goto" sont interdits.

On n'utilisera que des débranchements dits "structurés".

Il y a 5 débranchements structurés :

- **break** : fait quitter la boucle
- **next, suivant** : fait passer au suivant de la boucle (saut à la fin de la boucle)
- **reloop** : fait reprendre la boucle au départ.
- **return** : fait quitter la procédure
- **exit** : fait quitter le programme

5.2. utilisation

5.2.1. La boucle sans fin

La boucle sans fin est une quatrième sorte de boucle qui n'existe pas directement en Pascal.

Dans une boucle sans fin, il y a forcément un débranchement pour pouvoir sortir de la boucle.

Sa structure algo est la suivante :

5.2.1.1. Structure Algo

```
{ … ? ! /d … }
```

son meilleur équivalent Pascal est :

5.2.1.2. Structure Pascal

```
while TRUE do begin
    …
    if …
        break
    end;
    …
end;
```

5.2.2. Gestion des cas particuliers

On utilise les débranchements pour traiter les cas particuliers. Ensuite on peut passer au cas général sans avoir à rester imbriqué dans un test. Les débranchements permettent de désimbriquer les tests.

```
Procedure Equa1 (A, B: real; var X : real; var NbSol : integer);
Begin
    if A = 0 and B = 0 then begin
        NbSol := -1;
        return;
    end;
    if A = 0 and B != 0 then begin
        NbSol := 0 ;
        return;
```

```
        end;
        NbSol := 1;
        X := -B/A;
    end;
```

6. Les tableaux

6.1. Définition

Un type tableau est un type regroupant plusieurs variables de même type, ordonnées et accessibles individuellement par leurs indices.

Il existe des tableaux à un indice (on dit encore une dimension) et des tableaux à deux indices (deux dimensions).

6.1.1. Exemple Pascal

```
Type
    Tableau = Array [1..10] of integer;
    Matrice = Array [1.10,1..10] of real;
Var
    tab : Tableau;
    mat : Matrice;
Begin
    tab [1] := 5;
    tab [2] := tab [1];
    mat[1,1] := 4.3;
```

6.1.2. Exemple Pascal

```
Program TesteTableau;
    Tableau = Array [1..10] of integer;
Var
    tab : Tableau;
Begin
    For i := 1 to 10 do begin
        write ('entrez tab de ', i);
        readln (tab[i]);
    end;
    For i := 1 to 10 do begin
        writeln ('tab de ', i, ' = ', tab[i]);
    end;
```

6.2. exercices

6.2.1. somme des éléments d'un tableau à une dimension

Somme (tab (in), n (in)) :
[SOMME **<-** tab1 **;** {(i:2àn) SOMME **<-** SOMME+tab(i)} **]**

6.2.2. min et max d'un tableau à une dimension

MinMax (tab (in), n (in), min (out), max (out)) :
[min <- tab(1) ; max <- tab1 ; {(i:2àn) tab(i) < min ? min <- tab(i) /d tab(i) > max ? max **<-** tab(i) /d **}** **]**

6.2.3. indice du min et max d'un tableau à une dimension

NlMinMax (tab (in), n (in), nlmin (out), nlmax (out)) :
[nlmin **<-** 1 **;** max **<-** 1 **;** **{**(i:2àn) tab(i) < tab(nlmin) ? nlmin **<-** i /d tab(i) > tab(nlmax) ? nlmax **<-** nlmax /d **}** **]**

6.2.4. trier un tableau à une dimension

TriCroissant (tab<->, n<-) :
[**{**(i:1 à N-1) nlmin <- i {(j:i à N) taj(j)<tab(nlmin) ? nlmin <- j /d } tmp<-tab(i); tab(i)<-tab(nlmin); tab(nlmin)<-tmp **}** **]**

6.2.5. somme des éléments d'un tableau à deux dimensions

Somme (mat (in), n (in)) :
[SOMME **<-** 0 **;** **{**(i:1 à nl) **{**(j:1 à nc) SOMME **<-** SOMME+mat(i,j)} **}]**

6.2.6. max d'un tableau à deux dimensions

Max (mat (in), nl (in), nc (in), max (out)) :
[max <- mat(1,1) ; {(i:2àn) mat(i,j) > max ? max **<-** tab(i,j) /d **}** **]**

6.2.7. indices du min d'un tableau à deux dimensions

NlMinMax (mat (in), nl (in), nc (in), nlmin (out), ncmin (out)) :
[nlmin **<-** 1 **;** ncmin <- 1 **;** **{**(i:1àn) **{**(j:1àn) mat(i,j) < tab(nlmin, ncmin) ? nlmin **<-** i ; ncmin <- j /d **}** **]**

6.2.8. trier un tableau à deux dimensions, selon la x ième colonne

TriCroissant (mat<->, nl<-, nc <-, x <-) :
[**{**(i:1 à nl-1) nlmin <- i {(j:i à nl) mat(j,x) < mat(nlmin, x) ? nlmin <- j /d } {(j:1 à nc) tmp(j)<-tab(i,j); tab(i,j)<-tab(nlmin,j)); tab(nlmin,j)<-tmp(j) **}** **]**

7. Les chaînes de caractères

7.1. Définition

Une chaîne de caractère est une suite ordonnée de caractères.

7.1.1. Exemple Pascal

```
Type
    Chaine10 : String[10];
Var
    ch1, mot : chaine10;
    ch2 : string;
```

Par défaut, une chaîne de caractères contient au maximum 255 caractères. Elle est codée sur 256 octets car un octet est utilisé pour garder la longueur de la chaîne.

Une chaîne déclarée String[10] contient 10 caractères maximum et est représentée sur 11 octets, le onzième octet étant utilisé pour garder la longueur de la chaîne.

7.2. Algèbre

Théorie des opérations portant sur les chaînes de caractères: quels traitements peut-on faire subir à une variable de type chaîne de caractères ?

7.2.1. Affectations

Si la variable "Mot" est une variable de type chaîne de caractères, on peut lui affecter une expression de type chaîne de caractères ou de type caractère :

- mot := 'bonjour';
- mot := mot1; avec "mot1" de type chaîne de caractères.
- mot := C; avec "C" de type caractère.
- mot := expression; avec une expression de type caractère ou chaîne de caractères.

7.2.2. Comparaisons

On peut comparer les caractères et les chaînes de caractères. Le résultat de la comparaison c'est le résultat de la comparaison des valeurs ASCII correspondantes :

- 'Arrive' < 'arrive' est vrai (Code ASCII de 'A' = 65, de 'a' = 97);
- 'ARRIVE' > 'ARRIERE' est vrai (Code ASCII de 'V' = 86, de 'E' = 69);

C'est une comparaison par ordre alphabétique : le type est ordonné.

7.3. Fonctions applicables aux chaînes de caractères

7.3.1. Connaître sa longueur

La fonction **lgr** en algo, **length** en Pascal, permet de connaître la longueur d'une chaîne de caractère :

```
l <- lgr(ch) ;
```

7.3.2. Extraire un morceau

La fonction **ssch** (prononcer sous chaîne) en algo, **copy** en Pascal, permet de récupérer un morceau de chaîne de caractère (une sous-chaîne) démarrant, dans la **chaîne** fournie, à la position **début** et de longueur **lgr**.

ch <- **ssch** (chaîne, début , lgr),

7.3.3. Coller plusieurs morceaux

La fonction **concat** en algo et en Pascal permet de mettre bout à bout une série de chaînes de caractères (ou de caractères) produisant ainsi une nouvelle chaîne de caractères.

ch <- **Concat**(ch1, ch2, ch3, etc)

L'opérateur "+" est applicable aux chaînes de caractères. Cet opérateur est équivalent à la fonction concat.

Nom	Description	Type des entrées	Type du résultat
Concat	Concat (Ch1, Ch2, Ch3,...) concatène les chaînes en entrée	chaînes ou caractères	chaîne
Copy	Copy (Chaine, Position, Longueur) extrait une partie (sous-chaîne) d'une chaîne, d'une longueur donnée, à partir d'une position donnée.	une chaîne 2 entiers	chaîne
Length	Length (Chaine) fournit la longueur de la chaîne.	chaîne	entier

7.4. Exemples

7.4.1. position d'un caractère dans une chaîne

POS (chaîne, cara, début) :

[POS <- 0 **;** début <1? début <- 1 /d début >lgr(chaîne)?!* /d **{**(i:début à lgr(chaîne)) ssch(chaîne, i, 1)=cara ? POS <- i **;** !* /d **}** **]**

7.4.2. position d'une chaîne dans une chaîne

POS (chaîne, ch, début) :

[POS <- 0 **;** début <1? début <- 1 /d début >lgr(chaîne)-lgr(ch)+1?!* /d **{**(i:début à lgr(chaîne)-lgr(ch)+1) ssch(chaîne, i, lgr(ch))=ch? POS <- i **;** !* /d **}** **]**

7.4.3. nombre d'occurrences d'un caractère dans une chaîne

NbOccur (chaîne, cara) :

[NbOccur <- 0 **;** **{**(i:1 à lgr(chaine)) sschaine(chaine, i, 1)=cara ? NbOccur ++ /d **}** **]**

8. Conversion de type : chaîne vers entier ou reel

Il est nécessaire d'avoir des procédures de conversion de type pour éviter que les programmes "plantent" à chaque fois que l'utilisateur donne une valeur de type incompatible, par exemple une chaîne de caractère ("toto") à la place d'un entier.

Les fonction "**Val**" et "**Str**" jouent ce rôle.

Str	Str (RéelOuEntier, Chaine) transforme un réel ou un entier en chaîne.	un entier ou un réel une chaîne	chaîne
Val	Val (Chaine, RéelOuEntier, OK) transforme une chaîne en entier ou en réel et ressort un paramètre qui vaut 0 si la transformation s'est bien passée.	une chaîne un entier ou un réel un entier	entier ou réel

On peut toujours transformer un entier ou un réel en la chaîne de caractères correspondante. Par contre, on ne peut pas toujours transformer une chaîne de caractères en entier ou en réel. C'est pourquoi, la fonction "**Val**" ressort un paramètre (ok) qui dit si la transformation s'est correctement passée.

8.1. Exemples :

```
X:=45;
Str (X, Mot);     (* Mot vaut '45' *)
Val (Mot, Y, ok);                          (* Y vaut 45 et ok vaut 0 *)
Mot := 'toto';
Val (Mot, Y, ok);  (* Y vaut n'importe quoi et ok est différent de 0 *)
```

9. Les fichiers

9.1. Définition

Un fichier est une variable différente de toutes les autres variables utilisées jusqu'à présent car elle est conservée durablement indépendamment du programme.

Du point de vue le plus général, c'est-à-dire du point de vue du système d'exploitation de l'ordinateur, un fichier est une suite d'octets regroupés sous un même nom et stockés durablement.

Pour le langage Pascal, un fichier est une suite de variables de mêmes types conservées durablement.

9.1.1. Exemples Pascal :

```
Type
    FichierDeReels = File of real;
    FichierDEntiers = File of integer;
    FichierDeTexte = TEXT;
```

Le type TEXT est celui d'un fichier de chaînes de caractères de longueurs variables terminées par les deux caractères spéciaux : retour chariot (ASCII = 13) et saut de ligne (ASCII = 10) qui codent la fin de la chaîne et le passage à la ligne.

Le type TEXT est donc un cas particulier de File of Char.

Une variable de type fichier peut être comprise comme un enregistrement qui contient deux informations :

- le nom du fichier;
- l'adresse d'un élément du fichier;

Cette adresse permet de circuler dans le fichier, c'est pourquoi on choisit ici de préfixer les variables de type fichier par "Pt" pour signifier que ce sont des pointeurs.

9.1.2. Exemples Pascal :

```
Var
    PtFichTexte : FichierDeTexte ;
    PtFichReels : FichierDeReels;
    PtFichEntiers : FichierDEntiers;
    PtFichTexte2 : TEXT;
    PtFichReels2 : File of real;
```

9.2. Algèbre

9.2.1. Ouverture d'un fichier existant déjà :

Pour accéder au contenu d'un fichier qui existe déjà, on utilise les deux procédures suivantes :

Assign (Var PtFich : File; NomEtCheminFichier : String);

Reset (Var PtFich : File);

- NomEtCheminFichier : contient le nom du fichier et éventuellement le répertoire du fichier (sous la forme DOS standard : C: \ ... \ chezmoi \ monfichier). Par défaut, le fichier est cherché dans le répertoire courant.
- PtFich : contient l'adresse d'un élément du fichier.

Assign assigne à la variable de type fichier la chaîne de caractères fournie et qui est le nom du fichier. La variable de type fichier est modifiée : elle est en sortie. Le nom du fichier est en entrée.

Reset donne à l'adresse fournie par la variable de type fichier l'adresse du premier élément du fichier dont le nom a été assigné précédemment dans la variable de type fichier. Autrement dit Reset fait pointer le pointeur de fichier sur le premier élément du fichier. La variable de type fichier est donc utilisée (le nom du fichier) et modifiée (le pointeur) : elle est en entrée et en sortie.

9.2.2. Création d'un fichier

Pour créer un nouveau fichier, on utilise les deux procédures suivantes :

Assign (Var PtFich : File; NomEtCheminFichier : String);

Rewrite (Var PtFich : File);

Rewrite crée un fichier dont le nom a été assigné précédemment dans la variable de type fichier. Rewrite donne à l'adresse fournie par la variable de type fichier l'adresse du début du fichier, vide à ce moment. La variable de type fichier est donc utilisée (le nom du fichier) et modifiée (le pointeur) : elle est en entrée et en sortie.

Attention : si le fichier existait déjà, alors il est détruit.

9.2.3. Lire un élément dans le fichier et passer au suivant

Pour lire un élément du fichier on utilise les procédures suivantes :

Read (Var PtFich : File; Var v1, v2...);

Readln (Var PtFich : File; Var Ligne : String);

La procédure Read est utilisée pour tous les types de fichiers à l'exception des fichiers de type TEXT.

La procédure Readln est utilisée uniquement pour les fichiers de type TEXT.

Read et Readln recopient à partir de l'adresse fournie par la variable de type fichier (PtFich) les octets du fichier dans les variables passées en paramètre (v1, v2, etc.). L'adresse fournie par la variable de type fichier est alors modifiée : c'est celle de l'élément du fichier suivant la dernière lecture (éventuellement la fin du fichier). L'adresse fournie par la variable de type fichier est donc utilisée et modifiée : la variable de type fichier est en entrée et en sortie.

Dans le cas d'un fichier TEXT, Readln lit dans le fichier : soit sur toute la longueur de la chaîne en sortie, soit jusqu'à trouver dans le fichier un retour chariot (ASCII = 13) et saut de ligne (ASCII = 10) , c'est-à-dire les deux caractères de fin de ligne.

9.2.4. Écrire un élément dans le fichier et passer au suivant

Pour écrire un élément dans un fichier on utilise les procédures suivantes :

Write (Var PtFich : File; v1, v2...);

Writeln (Var PtFich : File; Ligne : String);

La procédure Write est utilisée pour tous les types de fichiers à l'exception des fichiers de type TEXT.

La procédure Writeln est utilisée uniquement pour les fichiers de type TEXT.

Write et Writeln écrivent le contenu des variables passées en paramètre (v1, v2, etc.) dans le fichier à l'adresse fournie par la variable de type fichier (PtFich). L'adresse fournie par la variable de type fichier est alors modifiée : c'est celle de l'élément du fichier suivant la dernière écriture (éventuellement la fin du fichier). L'adresse fournie par la variable de type fichier est donc utilisée et modifiée : la variable de type fichier est en entrée et en sortie.

Dans le cas d'un fichier TEXT, le Writeln écrit dans le fichier toute la chaîne de caractères en entrée (Ligne) puis les deux caractères spéciaux de fin de ligne : retour chariot (ASCII = 13) et saut de ligne (ASCII = 10).

9.2.5. Pointer directement sur un élément du fichier

Il est possible de donner à l'adresse fournie par la variable de type fichier l'adresse de n'importe quel élément du fichier en donnant son numéro d'ordre dans le fichier (on démarre la comptabilité à 0). Autrement dit, il est possible de pointer directement sur un élément d'un fichier :

Seek (Var PtFich : File; v1, Rang : Longint);

Seek donne à l'adresse fournie par la variable de type fichier la valeur de l'adresse de l'élément dont le rang est passé en entrée (cette adresse est calculée à partir de l'adresse du début du fichier, du numéro de rang et de la taille des éléments du fichier définie par leur type). L'adresse fournie par la variable de type fichier est modifiée : la variable de type fichier est en sortie, tandis que le numéro de rang est en entrée.

La procédure Seek peut être utilisée pour tous les types de fichiers, sauf les fichiers de type TEXT.

9.2.6. Fermeture d'un fichier

Pour fermer un fichier on utilise la procédure suivante :

Close (Var PtFich : File);

A la sortie de la procédure, l'adresse fournie par la variable de type fichier est nulle ainsi que le nom du fichier de la variable de type fichier.

Après un Close, la variable de type fichier peut être réutilisée pour une nouvelle assignation.

9.2.7. Détection de la fin d'un fichier

EOF (PtFich : File) : boolean;

La fonction EOF est Vrai quand l'adresse fournie par la variable de type fichier est celle de la fin du fichier.

9.2.7.1. Exemples Pascal

```
Program AfficherFichierTexte;
Var
    PtFich : Text;
    NomFich , ligne : String;
Begin
    write ('entrer le nom du fichier à afficher');
    readln (NomFich);
    Assign (PtFich, NomFich);
    Reset (PtFich);
    While not EOF (PtFich) do begin
        readln (PtFich, Ligne);
        writeln (Ligne);
    end;
    Close (PtFich);
end.
```

```
Program EcrireFichier;
Var
    X : Real;
    PtFich : File of Real;
    q : char;
Begin
    write ('entrer le nom du fichier à créer : ');
    readln (NomFich);
    Assign (PtFich, NomFich);
    Rewrite (PtFich);
    Repeat
        Write ('entrer un reel');
        Readln (X);
        Write (PtFich, X);
        Write ('entrer q pour arrêter le remplissage du fichier : ');
        readln (q);
    Until q = 'q';
    Close (PtFich);
```

```
end.
```

10. Trace d'un programme

exemple : tri d'un tableau

N=5						tab(1)	tab(2)	tab(3)	tab(4)	tab(5)
i	nlmax	j	tab(j)	tab(nlmax)	tmp	5	8	12	1	3
1										
	1	2	8							
				5						
	2									
		3	12	8						
	3									
		4	1	12						
		5	3	12						
					5	12		5		
2	etc.									