Algorithmes et programmation en Pascal

Faculté des Sciences de Luminy — Edouard Thiel

TD corrigés

Deug 1 Mass MA
Module de 75 heures
1997 à 2004

Table des matières

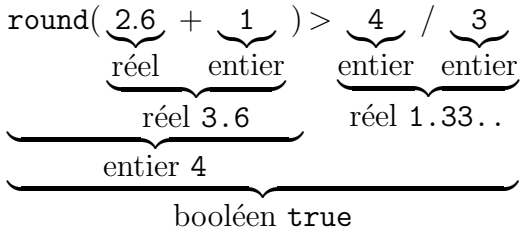
1. Expressions et affectations

1.1 Type d'une expression

Rappel Table des priorités classées par ordre décroissant, les opérateurs sur une même ligne ayant une priorité égale (on évalue alors de gauche à droite).

`() fonction()`	primaire
`+ - not`	unaire
`* / div mod and`	multiplicatif
`+ - or`	additif
`= <> < <= >= >`	relation

TD Donner le type et le résultat des expressions suivantes, ou dire si elles ne sont pas bien formées. Exemple :



1. `2 - 5 * 3 + 4` `(2 - 5) * (3 + 4)` `2 - (5 * 3 + 4)`
2. `12 / 3` `12 div 3` `11 div 3 div 2` `11 mod 3 + 5.2`
3. `1.0 * 2 + 3 - 4` `round (2 * 6.3) - 15 / 3` `(50 < 3 * 8)`
4. `false or not false and true` `(12 > 24) + (2 + 4 = 12)`
5. `(37 - 3 >= 14) - 'a' + 3` `pred ('b') > 'k'` `12 > 3 > 4`
6. `3.5 + 7 > 4 > false` `not (12 <> 3 * 16.8 / 4) and true`
7. `3 * cos (8.0 / (17 - (3 * 4) - 5))`

Correction

1. entier -9 entier -21 entier -17
2. réel 4.0 entier 4 entier 1 réel 7.2
3. réel 1.0 réel 8.0 booléen false
4. booléen true erreur : bool + bool
5. erreur : bool − car booléen false erreur : bool > 4 entier
6. booléen true booléen faux
7. erreur : division par 0

1.2 Année bissextile

Une année `a` est bissextile si elle est multiple de 4, et si elle est multiple de 100 elle doit aussi être multiple de 400. Par exemple 1996 oui, 1900 non, 2000 oui.

TD Écrire `b :=` l'expression.

TP Écrire un programme qui demande l'année, puis affiche si elle est bissextile.

Correction

`a` est bissextile si elle est multiple de 4 et pas de 100, ou multiple de 4 et de 400.

```
(1)   b := (a mod 4 = 0) and (a mod 100 <> 0)
        or (a mod 4 = 0) and (a mod 400 = 0);
(2)   b := (a mod 4 = 0) and (a mod 100 <> 0) or (a mod 400 = 0);
(3)   b := (a mod 4 = 0) and ((a mod 100 <> 0) or (a mod 400 = 0));
```

(1) $\longrightarrow$ (2) : `(a mod 4 = 0) and (a mod 400 = 0) = (a mod 400 = 0)`.
(1) $\longrightarrow$ (3) : `(x and y) or (x and z) = x and (y or z)`.

1.3 Boulangerie

Une boulangerie est ouverte de 7 heures à 13 heures et de 16 heures à 20 heures, sauf le lundi après-midi et le mardi toute la journée. On suppose que l'heure `h` est un entier entre 0 et 23. Le jour `j` code 0 pour lundi, 1 pour mardi, etc.

TD Écrire `b :=` l'expression, en essayant de trouver la plus courte.

TP Écrire un programme qui demande le jour et l'heure, puis affiche si la boulangerie est ouverte.

Correction

Il y a de multiples façons de procéder ; voici la plus courte.

```
b := (h >= 7) and (h <= 13) and (j <> 1) or
     (h >= 16) and (h <= 20) and (j > 1);
```

On peut se demander si il faut des inégalités strictes (l'énoncé ne précise rien).
$\longrightarrow$ Réfléchir chez soi au cas où les minutes sont fixées.

1.4 Logique de Boole

TD Simplifier les expressions :

```
p := (x < 7) and (y < 3) or not ((x >= 7) or (y < 3));
q := not (a and (not a or b)) or b;
```

Correction

On donne cette liste de propriétés (non vue en cours) avant de poser l'exercice :

```
  not (x and y) = (not x) or (not y)          {1}
  not (x or y) = (not x) and (not y)          {2}
  x and (y or z) = (x and y) or (x and z)     {3}
  x or (y and z) = (x or y) and (x or z)      {4}
  x and true = x                              {5}
  x or false = x                              {6}
  not x or x = true                           {7}
  not (u < v) = (u >= v)                      {8}
  not (u = v) = (u <> v)                      {9}
  x or y = y or x         (idem pour and)     {10}

p := (x < 7) and (y < 3) or not ((x >= 7) or (y < 3));
  := ((x < 7) and (y < 3)) or (not (x >= 7) and not (y < 3));  {2}
  := ((x < 7) and (y < 3)) or ((x < 7) and (y >= 3));
  := (x < 7) and ((y < 3) or (y >= 3));                        {3}
  := (x < 7) and true;                                         {7}
  := (x < 7);                                                  {5}

q := not (a and (not a or b)) or b;
  := (not a or not (not a or b)) or b;    {1}
  := not a or not (not a or b) or b;      {10}
  := (not a or b) or not (not a or b);    {10}
  := true;                                {7}
```

Remarque : $(x \Rightarrow y)$ s'écrit `not x or y`

1.5 Suppléments pour le TP

1) Codes ASCII

Faire un programme qui affiche la table des codes ASCII de 32 à 127 sur une colonne. Le modifier pour qu'il affiche sur 8 colonnes.

2) Erreurs

Faire de petits programmes qui illustrent les débordements arithmétiques et divisions par 0 vues en cours ; constater l'effet à l'exécution.

3) Portrait

Taper le programme `Portrait` vu en cours ; le modifier pour qu'il demande de rentrer au clavier les champs du type `personne_t`, puis qu'il les affiche.

2. Intervalles et enregistrements

2.1 Heures

1) Créer des types intervalles `heure_t`, `minute_t` et `seconde_t`, puis un type enregistrement `temps_t`.

Correction

```
TYPE
    heure_t   = 0..23;
    minute_t  = 0..59;
    seconde_t = 0..59;
    temps_t   = Record
                  h : heure_t;
                  m : minute_t;
                  s : seconde_t;
                End;
```

2) Soit `t1` et `t2` deux `temps_t`.
Faire un programme qui lit `t1` et `t2` et qui dit si ils sont égaux.

Correction

```
PROGRAM tp_egaux;
TYPE
    { cf 1) }
VAR
    t1, t2 : temps_t;
    egal   : boolean;
BEGIN
    write ('Temps1 (h m s) : ');
    readln (t1.h, t1.m, t1.s);
    write ('Temps2 (h m s) : ');
    readln (t2.h, t2.m, t2.s);

    { Teste si les temps sont egaux }
    egal := (t1.h = t2.h) and (t1.m = t2.m) and (t1.s = t2.s);

    writeln ('Egalite : ', egal);
END.
```

3) Modifier le programme pour qu'il dise si `t1 < t2`.
a) Avec une expression booléenne.
b) Avec des `if then else`.

Correction

Cela a été fait au premier semestre avec jour/mois/année, mais sans les record. C'est l'occasion de bien rappeler ces méthodes.

a) On déclare `inf : boolean`.

```
inf := (t1.h < t2.h)
    or (t1.h = t2.h) and (t1.m < t2.m)
    or (t1.h = t2.h) and (t1.m = t2.m) and (t1.s < t2.s);
```

On peut encore réduire l'expression :

```
inf := (t1.h < t2.h)
    or (t1.h = t2.h) and ( (t1.m < t2.m)
                        or (t1.m = t2.m) and (t1.s < t2.s) );
```

b) Autre méthode avec des `if then else` imbriqués : sera vu en cours prochainement.

```
inf := false;
if (t1.h < t2.h)
then inf := true
else if (t1.h = t2.h)
     then if (t1.m < t2.m)
          then inf := true
          else if (t1.m = t2.m)
               then if (t1.s < t2.s)
                    then inf := true;
```

4) Soit `t` un `temps_t`.

Faire un programme qui lit `t` et qui affiche le nombre de secondes `ns` écoulées depuis 0 :0 :0.

Correction

Le nombre de seconde écoulées est `ns := t.h * 3600 + t.m * 60 + t.s;`

On s'aperçoit que 23 :59 :59 compte 86399 secondes, on peut donc bien déclarer `ns` comme entier, car maxint $\simeq$ 2 milliards sous Delphi.

Sous Turbo Pascal, vu que maxint = 32767, on aurait dû déclarer `ns` comme un réel, puis faire `writeln ('ns : ', ns:5:0);` (5 chiffres en tout, dont 0 après la virgule).

```
PROGRAM nb_sec;
TYPE
    { cf 1) }
VAR
    t  : temps_t;
    ns : integer;
BEGIN
    write ('Temps (h m s) : ');
    readln (t.h, t.m, t.s);

    { calcule le nb de sec depuis 0:0:0 }
    ns := t.h * 3600 + t.m * 60 + t.s;

    writeln ('ns : ', ns);
END.
```

5) Faire un programme qui lit `t1` et `t2` et qui dit si `t1` est `<` ou `=` ou `>` à `t2`, en passant par la conversion en secondes.

Correction

```
PROGRAM tp_comp;
TYPE
    { cf 1) }
VAR
    t1, t2   : temps_t;
    ns1, ns2 : integer;
BEGIN
    { Lecture t1 et t2 ... }

    { calcule les nb de sec depuis 0:0:0 }
    ns1 := t1.h * 3600 + t1.m * 60 + t1.s;
    ns2 := t2.h * 3600 + t2.m * 60 + t2.s;

    if (ns1 < ns2)
    then writeln ('t1 < t2')
    else if (ns1 > ns2)
         then writeln ('t1 > t2')
         else writeln ('t1 = t2');
END.
```

6) Calculer la différence absolue en nombre de secondes entre `t1` et `t2`, puis en heures/minutes/secondes.

Correction

Soit `diff` un entier ; la différence est `diff := abs(ns1 - ns2);`
Idée : faire la conversion de `diff` en heures/minutes/secondes.
On place le résultat dans `t3 : temps_t`.

```
diff := abs(ns1 - ns2);
t3.h := diff div 3600;
diff := diff mod 3600;
t3.m := diff div 60;
diff := diff mod 60;
t3.s := diff;
```

Attention on ne peut employer `div` et `mod` que sur des entiers ! Dans le cas de Turbo Pascal où maxint = 32768, on aurait dû prendre `diff` réel, et il aurait fallu écrire :

```
diff := abs(ns1 - ns2);
t3.h := trunc (diff / 3600); { div 3600 }
diff := diff - t3.h * 3600;  { mod 3600 }
t3.m := trunc (diff / 60);   { div 60 }
diff := diff - t3.m * 60;    { mod 60 }
t3.s := diff;
```

Autre méthode : chercher d'abord les secondes puis les minutes puis les heures.

2.2 Suppléments pour le TP

1) Procédures

Taper tous les exemples du cours sur les procédures, et regarder ce qui se passe quand on met ou on enlève le `var` devant un paramètre.

2) Programme mystère

Que fait ce programme ?

```
PROGRAM abyz;
VAR d : char;
PROCEDURE change (var c : char);
BEGIN
    if (c >= 'a') and (c <= 'z')
    then c := chr (ord('a') + ord('z') - ord(c));
END;
BEGIN
    read (d);
    while d <> '.' do
    begin change (d); write (d); read (d); end;
    readln;
END.
```

3. Procédures

Toutes les procédures seront totalement paramétrées.

3.1 Min et Max

1) Procédure `Min` de 2 entiers.

2) Procédure `Max` de 2 entiers.

3) Procédure `MinMax` de 2 entiers, qui appelle les procédures `Min` et `Max`.

4) Programme qui lit 2 entiers, appelle `MinMax` et affiche les résultats.

Correction

Penser à utiliser des identificateurs différents à chaque niveau, pour l'exemple.

```
1)  PROCEDURE Min (    a, b : integer ;
                    var inf  : integer );
    BEGIN
        if a < b then inf := a
                 else inf := b;
    END;


2)  PROCEDURE Max (    a, b : integer ;
                    var sup  : integer );
    BEGIN
        if a > b then sup := a
                 else sup := b;
    END;


3)  PROCEDURE MinMax (    x, y : integer ;
                       var p, g : integer );
    BEGIN
        Min (x, y, p);
        Max (x, y, g);
    END;


4)  PROGRAM exo1;
    VAR u, v, pp, pg : integer;   { var globales }

    { ici procédures Min, Max et MinMax }

    BEGIN
        write ('u v ? '); readln (u, v);
        MinMax (u, v, pp, pg);
        writeln ('min ', pp, ' max ', pg);
    END.
```

3.2 Échange dans l'ordre croissant

1) Procédure `Echange2` sur 2 réels a, b qui échange éventuellement a et b, pour que l'état de sortie soit $a \leq b$. Écrire un programme d'appel.

2) Procédure `Echange3` sur 3 réels a, b, c qui appelle `Echange2`. L'état de sortie est $a \leq b \leq c$. Écrire plusieurs versions.

3) *En TP uniquement.* Procédure `Echange4` sur 4 réels a, b, c, d qui appelle `Echange2`. L'état de sortie est $a \leq b \leq c \leq d$. Écrire plusieurs versions.

Correction

```
1)  PROCEDURE Echange2 (var a, b : real);
    VAR t : real;
    BEGIN
        if a > b then
        begin t := a; a := b; b := t; end;
    END;

    PROGRAM Tri1;
    VAR u, v : real;
    { Mettre ici la procedure }
    BEGIN
        write ('u v ? '); readln (u, v);
        Echange2 (u, v);
        writeln (u, ' <= ', v);
    END.
```

On se pose la question : `var` ou pas `var` ?

- ▷ Pas de `var` : `a` et `b` sont des variables locales, initialisées par l'appel, et perdues après.
- ▷ Avec un `var` : `a` et `b` sont des *alias* de `u` et `v`.
  ⟶ Manipuler `a` et `b` revient à manipuler *directement* `u` et `v`.

```
2)  PROCEDURE Echange3 (var a, b, c : real);
    BEGIN
        Echange2 (a, b);  { on met le plus petit dans a }
        Echange2 (a, c);
        Echange2 (b, c);  { a etant le plus petit, on trie b,c }
    END;
```

Il y a d'autres solutions possibles. Exemple :

```
        Echange2 (a, b);  { on met le plus grand dans b }
        Echange2 (b, c);  { on met le plus grand dans c }
        Echange2 (a, b);  { c etant le plus grand, on trie a,b }
```

3.3 Passage d'enregistrements

Il s'agit de refaire (rapidement) une partie du §2 en insistant sur le passage de paramètres.

Soit le type `temps_t` défini au §2, soit `t` un `temps_t`, et `ns` un entier.

1) Écrire une procédure `ecriture(t)` qui affiche à l'écran les valeurs des champs de `t` sous la forme `h:m:s` sans retour à la ligne.

2) Écrire une procédure `lecture(t)` qui lit au clavier les valeurs à mémoriser dans les champs de `t`.

3) Écrire une procédure `calc_ns(t,ns)` qui renvoie le nombre de secondes `ns` écoulées depuis 0 :0 :0.

4) Faire un programme qui lit `t1` et `t2` et qui dit si `t1` est < ou = ou > à `t2`, en passant par la conversion en secondes. Le programme utilise les procédures précédentes.

Correction

```
    PROGRAM temps;

    TYPE
        heure_t   = 0..23;
        minute_t  = 0..59;
        seconde_t = 0..59;
        temps_t   = Record
                      h : heure_t;
                      m : minute_t;
                      s : seconde_t;
                    End;

1)  PROCEDURE ecriture (t : temps_t);
    BEGIN
        write (t.h, ':', t.m, ':', t.s);
    END;

2)  PROCEDURE lecture (var t : temps_t);
    BEGIN
        write ('Temps (h m s) : ');
        readln (t.h, t.m, t.s);
    END;

3)  PROCEDURE calc_ns (t : temps_t; var ns : integer);
    BEGIN
        ns := t.h * 3600 + t.m * 60 + t.s;
    END;

4)  VAR t1, t2 : temps_t;
        n1, n2 : integer;
        c      : char;

    BEGIN
        lecture (t1); lecture (t2);
        calc_ns (t1, n1); calc_ns (t2, n2);
```

```
    if n1 < n2 then                 c := '<'
               else if n1 = n2 then c := '='
                                   else c := '>';

    ecriture (t1) ; write (' ', c, ' '); ecriture (t2); writeln;
END.
```

3.4 Suppléments pour le TP

1) Programme mystère

La balle est sous quel gobelet ?

```
PROGRAM balle;

TYPE gobelet_t = record
                   balle   : boolean;
                   couleur : (Rouge, Vert, Bleu);
                 end;
VAR g1,g2,g3 : gobelet_t;

PROCEDURE abra ( var x, y : gobelet_t );
VAR  b : boolean;
BEGIN
    b := x.balle; x.balle := y.balle; y.balle := b;
END;

PROCEDURE cad ( x, y : gobelet_t );
VAR  b : boolean;
BEGIN
    b := x.balle; x.balle := y.balle; y.balle := b;
END;

BEGIN
    g1.balle := TRUE; g2.balle := FALSE; g3.balle := FALSE;

    abra (g1,g2); cad (g2,g3); abra (g3,g1);

    if g1.balle then
            writeln('la balle est sous le gobelet 1');
    if g2.balle then
            writeln('la balle est sous le gobelet 2');
    if g3.balle then
            writeln('la balle est sous le gobelet 3');
END.
```

4. Boucles et fonctions

4.1 Fonctions numériques

1) Factorielle
Faire une fonction `facto(n)` qui renvoie $n!$.

2) Puissance
Faire une fonction `puiss(x,n)` qui renvoie x^n.

3) Exponentielle $$e^x \simeq 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

Faire une fonction `expn(x,n)` qui calcule la valeur approchée de e^x en faisant appel aux fonctions `facto` et `puiss`.

Correction

```
1)  FUNCTION facto (n : integer) : integer;
    VAR i, r : integer;
    BEGIN
        r := 1; { init pour 0! }
        for i := 1 to n do r := r * i;  { r = i! }
        facto := r;
    END;
```

– La fonction est juste également pour `n` $= 0$.
– On peut aussi commencer le `for` à 2.

```
2)  FUNCTION puiss (x : real; n : integer) : real;
    VAR i : integer;
        r : real;
    BEGIN
        r := 1.0; { init pour x^0 }
        for i := 1 to n do r := r * x;  { r = x^i }
        puiss := r;
    END;
```

– 0^0 n'est mathématiquement pas défini.

```
3)  FUNCTION expn (x : real; n : integer) : real;
    VAR i : integer;
        r : real;
    BEGIN
        r := 1.0; { init }
        for i := 1 to n do r := r + puiss(x,i) / facto(i);
        expn := r;
    END;
```

– Cette fonction n'est pas efficace à cause des appels à `puiss` et à `facto` qui sont de plus en plus lents.

4.2 Lecture au clavier

1) Compter des caractères

Faire une fonction qui lit au clavier une suite de caractères terminée par `'.'` et qui renvoie le nombre de caractères lus avant le `'.'`.

Ne pas les laisser chercher trop longtemps, mais leur expliquer l'algorithme.

2) Compter les L

Faire une fonction qui lit au clavier une suite de caractères terminée par `'.'` et qui renvoie le nombre d'occurences du caractère `L`.

Correction

```
FUNCTION nb_single (C1, CFin : char) : integer;
VAR res : integer;
    c   : char;
BEGIN
    res := 0; { init }
    read(c);  { lit le 1er caractere }
    while c <> CFin do
    begin
        if c = C1 then res := res+1;
        read(c); { lit le prochain caractere }
    end;
    readln; { vide le buffer apres le CFin         }
            { et SURTOUT absorbe le retour chariot }
    nb_single := res;  { resultat de la fonction }
END;
```

On peut aussi écrire un `repeat until` avec un seul `read` au début (équivalent), ou un `while` avec un seul `read` au début (à proscrire).

3) Compter les LE *Uniquement en TP*

Faire une fonction qui lit au clavier une suite de caractères terminée par `'.'`, en mémorisant à chaque lecture le caractère précédent, puis renvoie le nombre d'occurences de couples de caractères consécutifs `LE`.

Correction

```
FUNCTION nb_couple (C1, C2, CFin : char) : integer;
VAR res  : integer;
    c, d : char;
BEGIN
    res := 0;  { init }
    d := CFin; { precedent car. lu, init <> C1 et C2 }
    read(c);
    while c <> CFin do
    begin
        if (d = C1) and (c = C2) then res := res+1;
        d := c; read(c);
    end;
    readln;
    nb_couple := res;
END;
```

4.3 Intervalles de temps

Il s'agit d'insister sur le passage de paramètres avec des fonctions.

Soit le type `temps_t` défini au §2.

1) Écrire un type enregistrement `inter_t` qui mémorise le début et la fin d'une période de temps.

2) Écrire une fonction `calc_ns(t)` qui renvoie le nombre de secondes `ns` écoulées depuis 0 :0 :0.

3) Écrire une fonction `convertir(ns)` qui convertit `ns` en heure/minute/seconde et renvoie le résultat sous forme d'un `temps_t`.

4) Écrire une fonction `intertemps(i)` où `i` est un `inter_t`. La fonction renvoie dans un `temps_t` la durée séparant le début et la fin. La fonction se sert de `calc_ns` et de `convertir`.

Correction

```
1)  TYPE
        heure_t   = 0..23;
        minute_t  = 0..59;
        seconde_t = 0..59;
        temps_t   = Record
                      h : heure_t;
                      m : minute_t;
                      s : seconde_t;
                    End;
        inter_t =   Record
                      debut, fin : temps_t;
                    End;

2)  FUNCTION calc_ns (t : temps_t) : integer;
    BEGIN
        calc_ns := t.h * 3600 + t.m * 60 + t.s;
    END;

3)  FUNCTION convertir (ns : integer) : temps_t;
    VAR res : temps_t;
    BEGIN
        res.h := ns div 3600;
        ns    := ns mod 3600;
        res.m := ns div 60;
        ns    := ns mod 60;
        res.s := ns;
        convertir := res;
    END;

4)  FUNCTION intertemps (i : inter_t) : temps_t;
    BEGIN
        intertemps := convertir(calc_ns(i.fin) - calc_ns(i.debut));
    END;
```

4.4 Suppléments pour le TP

1) Calcul de pi $\dfrac{\pi}{4} = 1 - \dfrac{1}{3} + \dfrac{1}{5} - \dfrac{1}{7} + \dfrac{1}{9} \cdots$

Faire une fonction `calcpi` qui calcule la valeur approchée de π en s'arrêtant lorsque le terme $\frac{1}{x}$ est plus petit que ε.

Faire un programme qui lit ε réel, appelle `calcpi` puis affiche le résultat.

Correction

```
PROGRAM calculpi;

FUNCTION calcpi (e : real) : real;
VAR s, q, r, t : real;
BEGIN
    s := 1.0; { signe }
    q := 1.0; { diviseur }
    r := 1.0; { somme partielle }

    repeat
      s := -s;     { inverse le signe }
      q := q + 2;  { diviseur de 2 en 2 }
      t := s / q;  { terme }
      r := r + t;  { somme partielle }
    until abs(t) < e;

    calcpi = r*4;
END;

VAR e : real;
BEGIN
    write ('epsilon = '); readln (e);
    writeln ('pi = ', calcpi(e));
END.
```

5. Tableaux et indices

5.1 Le programme `AppaLet`

On dispose des types suivants.

```
CONST MaxTabCar = 4096;
      CMin = 'a';
      CMax = 'z';
TYPE  TabCar_t = array [1 .. MaxTabCar] of char;
      TabPourcent_t = array [CMin .. CMax] of real;
```

1) Lecture au clavier

Faire une procédure `LectureLettres (tabCar, nbCar, CarFin)` qui lit au clavier une suite de caractères terminée par `CarFin:char` et mémorise au fur et à mesure la suite dans le vecteur `tabCar:TabCar_t` ainsi que le nombre de caractères lus dans `nbCar`. On considère que `CarFin` ne fait pas partie de la suite.

2) Affichage à l'envers

Faire une procédure `AfficheEnvers (tabCar, nbCar)` qui affiche à l'envers le contenu de `tabCar:TabCar_t`.

3) Calcul de pourcentage

Faire une procédure `CalcPourcent (tabCar, nbCar, tabP)` qui reçoit en paramètres le tableau de caractères `tabCar:TabCar_t` et sa taille courante `nbCar`, puis stocke dans `tabP:TabPourcent_t` le pourcentage d'apparition de chaque caractères de `tabCar`.

4) Recherche du maximum

Faire une fonction `MaxPourcent (tabP)` qui renvoie le caractère ayant le plus grand pourcentage d'apparition dans `tabP:TabPourcent_t`.

5) Recherche des plus fréquents

Faire une procédure `PlusFrequents (tabP, tabF, n)` qui recherche les `n` caractères les plus fréquents à partir des pourcentages stockés dans `tabP:TabPourcent_t` et qui les mémorise dans le vecteur `tabF:TabCar_t`, le caractère le plus fréquent en tête.

6) Programme principal

Faire un programme principal `AppaLet` qui lit au clavier une suite de caractères terminée par `'.'`, recherche les 5 lettres les plus fréquentes dans le texte, puis les affiche dans l'ordre croissant de fréquence.

Correction

```
1)  PROCEDURE LectureLettres (var tabCar : TabCar_t;
                              var nbCar  : integer;
                                  CarFin : char );
    VAR c : char;
    BEGIN
        nbCar := 0;
        read (c);              { lecture premier caractère }
        while c <> CarFin do
        begin
          nbCar := nbCar + 1;
          tabCar[nbCar] := c;
          read (c);            { lecture du caractère suivant }
        end;
        readln;                { absorbe le retour chariot }
    END;

2)  PROCEDURE AfficheEnvers (tabCar : TabCar_t;
                             nbCar  : integer );
    VAR i : integer;
    BEGIN
        for i := nbCar downto 1 do write(tabCar[i]);
        writeln;
    END;


3)  PROCEDURE CalcPourcent (    tabCar : TabCar_t;
                                nbCar  : integer;
                            var tabP   : TabPourcent_t);
    VAR i, n : integer;
        c    : char;
    BEGIN
        { initialisation }
        for c := CMin to CMax do tabP[c] := 0.0;
        n := 0;   { nb de char dans CMin..CMax }

        { comptage }
        for i := 1 to nbCar do
        begin
          c := tabCar[i];
          if (c >= CMin) and (c <= CMax)
          then begin
                 tabP[c] := tabP[c] + 1.0;
                 n := n+1;
               end;
        end;

        { calcul pourcentages }
        for c := CMin to CMax do tabP[c] := tabP[c] * 100.0 / n;
    END;
```

```
4)  FUNCTION MaxPourcent (tabP : TabPourcent_t) : char;
    VAR c, m : char;
    BEGIN
        { recherche de l'indice m du max }
        m := CMin;
        for c := succ(CMin) to CMax do
          if tabP[c] > tabP[m] then m := c;
        MaxPourcent := m;
    END;

5)  PROCEDURE PlusFrequents (    tabP : TabPourcent_t;
                             var tabF : TabCar_t;
                                 n    : integer);
    VAR i : integer;
        c : char;
    BEGIN
        for i := 1 to n do
        begin
          c := MaxPourcent (tabP);
          tabF[i] := c;   { c est le caractère le plus fréquent.   }
          tabP[c] := 0.0; { on met la fréquence de c à 0, pour que }
        end;              {   le max suivant ne soit plus ce c     }
    END;

6)  PROGRAM AppaLet;

    { Ici : types et constantes, procédures et fonctions }

    VAR
        tc, tf : TabCar_t;
        tp     : TabPourcent_t;
        nc     : integer;
    BEGIN
        LectureLettres (tc, nc, '.');
        CalcPourcent   (tc, nc, tp);
        PlusFrequents  (tp, tf, 5);
        AfficheEnvers  (tf, 5);
    END.
```

6. Tableaux, record et string

6.1 Le programme `RendezV`

Soit le type `temps_t` défini au §2, les procédures `ecriture (t : temps_t)` et `lecture (t : temps_t)`, et la fonction `calc_ns (t : temps_t) : integer`. On introduit les types suivants :

```
CONST MaxRdv = 20;
TYPE  Rdv_t = Record
                titre      : string[63];
                debut, fin : temps_t;
              End;
      TabRdv_t = array [1..MaxRdv] of Rdv_t;
```

1) Saisie

Faire une procédure `SaisieRdv (r)` qui lit au clavier les champs de `r:Rdv_t`, au besoin en appelant la procédure `lecture()`.

2) Affichage

Faire une procédure `AffiRdv (r)` qui affiche à l'écran les champs de `r:Rdv_t`, au besoin en appelant la procédure `ecriture()`.

3) Chevauchement

Faire une fonction booléenne `Chevauche (r1,r2)` qui renvoie `TRUE` si les rendez-vous `r1,r2:Rdv_t` se chevauchent.

4) Test journée

Faire une procédure `TestJournee (j,n)` qui détecte et affiche les rendez-vous qui se chevauchent dans la journée, parmi les `n` rendez-vous stockés dans `j:TabRdv_t`.

5) Programme principal

Le programme principal `RendezV` demande un nombre de rendez-vous, lit ces rendez-vous au clavier, puis affiche la liste des rendez-vous qui se chevauchent.

Correction

```
1)  PROCEDURE SaisieRdv (var r : Rdv_t);
    BEGIN
        write ('Titre : '); readln (r.titre);
        lecture (r.debut);
        lecture (r.fin);
    END;

2)  PROCEDURE AffiRdv (r : Rdv_t);
    BEGIN
        writeln ('Titre : ', r.titre);
        ecriture (r.debut);
        ecriture (r.fin);
    END;

3)  FUNCTION Chevauche (r1, r2 : Rdv_t) : boolean;
    VAR deb1, deb2, fin1, fin2;
    BEGIN
        deb1 := calc_ns(r1.debut); fin1 := calc_ns(r1.fin);
        deb2 := calc_ns(r2.debut); fin2 := calc_ns(r2.fin);
        Chevauche := (deb1 < fin2) and (deb2 < fin1);
    END;

4)  PROCEDURE TestJournee (j : TabRdv_t; n : integer);
    VAR i, k : integer;
    BEGIN
        for i := 1 to n-1 do
          for k := i+1 to n do
            if Chevauche (j[i], j[k])
              then begin
                     writeln ('Ces RDV se chevauchent :');
                     AffiRdv (j[i]);
                     AffiRdv (j[k]);
                   end;
    END;

5)  PROGRAM RendezV;

    { Ici : types et constantes,                              }
    {       procedures lecture et ecriture, fonction calc_ns }

    VAR i, nb : integer; j : TabRdv_t;
    BEGIN
        write ('Nb rdv : '); readln (nb);
        for i := 1 to nb do SaisieRdv (j[i]);
        TestJournee (j, nb);
    END.
```

6.2 Cryptage

TD Faire le tableau de sortie du programme mystère (= écrire le tableau des valeurs successives des variables et expressions).

TP Faire tracer le programme mystère (= mettre des `writeln` affichant les valeurs successives des variables et expressions).

```
PROGRAM mystere;          { Partiel juin 1997 }
TYPE t = string[127];
PROCEDURE p (var m : t; c : t);
VAR i : integer;
BEGIN
    for i := 1 to length(m) do m[i] :=
      chr( ord(m[i]) + ord(c[ (i-1) mod length(c) +1]) - ord('a'));
END;
VAR m, c : t;
BEGIN
    m := 'on' + 'claff'; c := 'ieg'; p (m,c);
    writeln (m);
END.
```

Correction

La procédure reçoit les paramètres `m = 'onclaff'` et `c = 'ieg'`.

Dans la boucle on pause `j := (i-1) mod length(c) +1;`

L'expression à évaluer est `chr( ord(m[i])+ord(c[j])-ord('a') )`.

i	ord(m[i])	j	ord(c[j])	ord(c[j])-ord('a') =?	chr()
1	ord('o')	1	ord('i')	ord('i')-ord('a') = 8	'w'
2	ord('n')	2	ord('e')	ord('e')-ord('a') = 4	'r'
3	ord('c')	3	ord('g')	ord('g')-ord('a') = 6	'i'
4	ord('l')	1	ord('i')	ord('i')-ord('a') = 8	't'
5	ord('a')	2	ord('e')	ord('e')-ord('a') = 4	'e'
6	ord('f')	3	ord('g')	ord('g')-ord('a') = 6	'l'
7	ord('f')	1	ord('i')	ord('i')-ord('a') = 8	'n'

7. Détection de mots

7.1 Le programme `ColEcran`

On dispose du type `ligne_t = string[255]`. Soit `s` une `ligne_t`.

1) Début avec des `'a'`

On considère que `s` ne comprend que des `'a'` et des espaces, et que les groupes de `a` forment des « mots ». On veut détecter le début et la fin des mots. Écrire :

```
Function DetecteDebut (s : ligne_t; i : integer) : integer;
```

La fonction renvoie la position du début du prochain mot à partir de `i` dans `s`, renvoie `i` si `i` est déjà dans un mot, ou renvoie 0 s'il n'y a plus de mot à partir de `i` ou si `i` n'est pas dans l'intervalle 1..255.

2) Fin avec des `'a'`

Avec les mêmes hypothèses sur le contenu de `s`, on considère que `i` est *dans* un mot. Écrire :

```
Function DetecteFin (s : ligne_t; i : integer) : integer;
```

La fonction renvoie la position du dernier caractère du mot à partir de `i` dans `s`.

3) Test de lettre

Écrire la `Function EstLettre (c : char) : boolean;` qui reçoit un caractère `c` quelconque, puis renvoie `TRUE` si `c` est une lettre (i.e un caractère minuscule ou majuscule ou souligné).

4) Début et fin avec des lettres

On considère maintenant que `s` contient des caractères quelconques, qu'un mot est composé de lettres, et que les mots sont séparés par des caractères qui ne sont pas des lettres.

Réécrire les fonctions `DetecteDebut` et `DetecteFin`.

5) Affichage d'un mot

Faire la `Procedure AffiMot (s : ligne_t; debut, fin : integer);` qui affiche un mot extrait de `s` à partir de son `debut` et de sa `fin`.

6) Programme principal

Faire le programme principal `ColEcran` qui lit au clavier une ligne de caractères quelconques, puis affiche tous les mots de la ligne les uns en dessous des autres.

Correction

1) Attention dans le `while` : en Pascal les expressions sont toujours évaluées complètement, et on n'a pas le droit d'écrire `s[256]`. C'est pourquoi on teste l'inférieur **strict** sur `i` dans le `while`. On peut éviter ce problème avec un booléen `continuer`.

```
Function DetecteDebut (s : ligne_t; i : integer) : integer;
Begin
    DetecteDebut := 0;
    if (i >= 1) and (i <= length(s))
    then begin
           while (s[i] <> 'a') and (i < length(s)) do
               i := i+1;
           if s[i] = 'a' then DetecteDebut := i;
         end;
End;
```

2)
```
Function DetecteFin (s : ligne_t; i : integer) : integer;
Begin
    while (s[i] = 'a') and (i < length(s)) do
        i := i+1;
    if s[i] = 'a' then DetecteFin := i
                  else DetecteFin := i-1;
End;
```

3)
```
Function EstLettre (c : char) : boolean;
Begin
    EstLettre := (c >= 'a') and (c <= 'z') or
                 (c >= 'A') and (c <= 'Z') or
                 (c  = '_');
End;
```

4) Il suffit de remplacer `(s[i] = 'a')` par `EstLettre (s[i])`.

```
Function DetecteDebut (s : ligne_t; i : integer) : integer;
Begin
    DetecteDebut := 0;
    if (i >= 1) and (i <= length(s))
    then begin
           while not EstLettre (s[i]) and (i < length(s)) do
               i := i+1;
           if EstLettre (s[i]) then DetecteDebut := i;
         end;
End;

Function DetecteFin (s : ligne_t; i : integer) : integer;
Begin
    while EstLettre (s[i]) and (i < length(s)) do
        i := i+1;
    if EstLettre (s[i]) then DetecteFin := i
                        else DetecteFin := i-1;
End;
```

5)
```
Procedure AffiMot (s : ligne_t; debut, fin : integer);
Var i : integer;
Begin
    for i := debut to fin do write (s[i]);
End;
```

6)
```
PROGRAM ColEcran;

{ ici : Types et constantes, procédures et fonctions }

Var s    : ligne_t;
    i, j : integer;
Begin
    writeln ('Tapez une ligne :'); readln (s);

    i := DetecteDebut (s, 1);
    while i <> 0 do
    begin
      j := DetecteFin (s, i);
      AffiMot (s, i, j);
      writeln;
      i := DetecteDebut (s, j+1);
    end;
End.
```

8. Fichiers texte

Rappel Sous Delphi, il faut mettre un `var` devant les paramètres de type `text`.

8.1 Le programme `MinusCol`

On dispose des types et des fonctions définis au §7.

1) Caractère minuscule
Écrire la `function MinusCar (c : char) : char` qui renvoie `c` en minuscule si `c` est une lettre, sinon renvoie `c` inchangé.

2) Ligne minuscule
Écrire la `procedure MinusLig (var s : ligne_t)` qui met tous les caractères de `s` en minuscules.

3) Ouverture et fermeture
On suppose que l'on dispose de la `procedure CasseFi (var f1, f2 : text)`. Écrire une `procedure CasseNom (nom1, nom2 : ligne_t)` qui ouvre en lecture `f1:text`, ouvre en écriture `f2:text`, appelle `CasseFi (f1, f2)` puis ferme ces fichiers.

4) Gestion des erreurs *En TP uniquement*
Réécrire la procédure `CasseNom` en une fonction qui renvoie 0 si tout s'est bien passé, 1 si il y a une erreur d'ouverture sur `nom1`, 2 si il y a une erreur d'ouverture sur `nom2`.

5) Caractère par caractère
Faire la `procedure CasseFi (var f1, f2 : text)` qui lit `f1` caractère par caractère, et recopie ces caractères en minuscule dans `f2`.

6) Ligne par ligne
Refaire la procédure `CasseFi` en procédant cette fois ligne par ligne.

7) Un mot par ligne
On suppose que l'on dispose de la `procedure ColNom (nom1, nom2 : ligne_t)` qui fait la même chose que `CasseNom`, mais appelle `ColFi` à la place de `CasseFi`.
Écrire cette `procedure ColFi (var f1, f2 : text)`, qui recopie en minuscule dans `f2` les mots de `f1`, en écrivant un mot par ligne.

8) Programme principal
Faire le programme principal `MinusCol` qui lit au clavier 2 noms de fichiers, puis recopie dans le second fichier tous les mots du premier fichier, en minuscule les uns en dessous des autres.
En TP, afficher en cas d'erreur un message selon la nature de l'erreur.

Correction

```
1)  Function MinusCar (c : char) : char;
    Begin
        if (c >= 'A') and (c <= 'Z')
        then MinusCar := chr(ord(c) - ord('A') + ord ('a'))
        else MinusCar := c;
    End;

2)  Procedure MinusLig (var s : ligne_t);
    Var i : integer;
    Begin
        for i := 1 to length(s) do
            s[i] := MinusCar(s[i]);
    End;

3)  Procedure CasseNom (nom1, nom2 : ligne_t);
    Var f1, f2 : text;
    Begin
        assign (f1, nom1);
        assign (f2, nom2);
        reset (f1);
        rewrite (f2);
        CasseFi (f1, f2);
        close (f1);
        close (f2);
    End;
```

4) Dans le cours, les erreurs de fichiers sont à la fin du chapitre sur les fichiers, et le schéma algorithmique de gestion des erreurs est à la fin du chapitre sur les fonctions.

```
    Function CasseNom (nom1, nom2 : ligne_t) : integer;
    Var f1, f2 : text;
        err    : integer;
    Begin
        err := 0;
        assign (f1, nom1);
        assign (f2, nom2);

        {$I-} reset (f1); {$I+}
        if IoResult <> 0
        then err := 1;

        if err = 0 then
        begin
          {$I-} rewrite (f2); {$I+}
          if IoResult <> 0
          then err := 2;
        end;

        if err = 0 then CasseFi (f1, f2);

        if err <> 1 then close (f1);  { ouvert si err = 0 ou 2 }
        if err = 0  then close (f2);

        CasseNom := err;
    End;
```

5) On lit tous les caractères, y compris les retours chariots, et on recopie en minuscule.

```
Procedure CasseFi (var f1, f2 : text);
Var c : char;
Begin
    while not eof(f1) do
    begin
      read (f1, c);
      write (f2, MinusCar(c));
    end;
End;
```

6) Attention il faut employer `readln` et `writeln` sur les strings.

```
Procedure CasseFi (var f1, f2 : text);
Var s : ligne_t;
Begin
    while not eof(f1) do
    begin
      readln (f1, s);
      MinusLig (s);
      writeln (f2, s);
    end;
End;
```

7) On mélange le code de `CasseFi`, `DetecteMots` et `AffiMot`. On aurait pu écrire une procédure `AffiMotFi`.

```
Procedure ColFi (var f1, f2 : text);
Var s : ligne_t;
    i, j, k : integer;
Begin
    while not eof(f1) do
    begin
      readln (f1, s);
      MinusLig (s);

      i := DetecteDebut (s, 1);
      while i <> 0 do
      begin
        j := DetecteFin (s, i);

        for k := i to j do write (f2, s[k]);
        writeln (f2);

        i := DetecteDebut (s, j+1);
      end;
    end;
End;
```

```
8)  PROGRAM MinusCol;

    { ici : Types et constantes, procédures et fonctions }

    VAR n1, n2 : ligne_t;
    BEGIN
        write ('nom1 = '); readln (n1);
        write ('nom2 = '); readln (n2);
        ColNom (n1, n2);
    END.
```

9. Recherche dans un fichier

Dans ces exercices on suppose que l'on a en entrée un fichier texte, résultat du programme `MinusCol` du §8, c'est-à-dire : exactement un mot par ligne, en minuscule et sans espacement (sauf à la fin où il peut y avoir une ligne vide).

Chaque procédure ou fonction fait les opérations d'ouverture et de fermeture des fichiers dont le nom est passé en paramètre. *En TD*, on supposera qu'il n'y a pas d'erreur à l'ouverture des fichiers.

9.1 Recherche de mots

1) Nombre de mots
Écrire la fonction `NombreMots (nomf : ligne_t) : integer;` qui renvoie le nombre de mots présents dans le fichier `nomf`.

2) Recherche d'un mot
Écrire la fonction `RechercheMot (nomf, m : ligne_t) : boolean;` qui renvoie `TRUE` si le mot `m` (reçu en minuscules) est présent dans le fichier `nomf`.

3) Compter un mot
Écrire la fonction `CompterMot (nomf, m : ligne_t) : integer;` qui renvoie le nombre d'occurences du mot `m` (reçu en minuscules) dans le fichier `nomf`.

4) Compter dans un fichier trié
Écrire la fonction `CompterMotTri (nomf, m : ligne_t) : integer;` qui fait la même chose que `CompterMot` mais sur un fichier `nomf` trié sur les mots, en s'arrêtant le plus tôt possible.

5) Programme principal
En TP, faire un programme principal et un fichier texte d'exemple (trié à la main si besoin), de façon à tester les procédures et fonctions de l'exercice.

Correction

1)
```
Function NombreMots (nomf : ligne_t) : integer;
Var f : text; s : ligne_t; n : integer;
Begin
    assign (f, nomf); reset (f); n := 0;

    while not eof (f) do
    begin
      readln (f, s);
      if length (s) > 0 then n := n+1;
    end;

    close (f); NombreMots := n;
End;
```

2)
```
Function RechercheMot (nomf, m : ligne_t) : boolean;
Var f : text; s : ligne_t; trouve : boolean;
Begin
    assign (f, nomf); reset (f); trouve := false;

    while not eof (f) and not trouve do
    begin
      readln (f, s);
      if s = m then trouve := true;
    end;

    close (f); RechercheMot := trouve;
End;
```

3)
```
Function CompterMot (nomf, m : ligne_t) : integer;
Var f : text; s : ligne_t; n : integer;
Begin
    assign (f, nomf); reset (f); n := 0;

    while not eof (f) do
    begin
      readln (f, s);
      if s = m then n := n+1;
    end;

    close (f); CompterMot := n;
End;
```

4)
```
Function CompterMotTri (nomf, m : ligne_t) : integer;
Var f : text; s : ligne_t; n : integer; continuer : boolean;
Begin
    assign (f, nomf); reset (f); n := 0; continuer := true;

    while not eof (f) and continuer do
    begin
      readln (f, s);
      if s = m then n := n+1
      else if s > m                  {  .. and length (s) > 0     }
           then continuer := false;  { inutile de tester car s ne }
    end;                             { peut être vide qu'à la fin }

    close (f); CompterMotTri := n;
End;
```

9.2 Le programme `NuMots`

1) Compter tous les mots

Écrire la procédure `CompterChaqueMot (nomf1, nomf2 : ligne_t);`. Le fichier d'entrée `nomf1` est trié sur les mots ; dans le fichier `nomf2` on va écrire chaque mot du fichier d'entrée de façon unique, suivi sur sa ligne d'un espace et de son nombre d'occurences.

La procédure fait un seul passage sur le fichier d'entrée. Celui-ci étant trié, on utilise le fait que les occurences d'un même mot sont forcément consécutives.

2) Programme principal

En TP, faire le programme principal `NuMots` qui lit au clavier le nom de deux fichiers et appelle `CompterChaqueMot`.

Correction

```
1)  Procedure CompterChaqueMot (nomf1, nomf2 : ligne_t);
    Var f1, f2 : text;
        n : integer;
        lu, prec : ligne_t;
    Begin
        assign (f1, nomf1); reset (f1);
        assign (f2, nomf2); rewrite (f2);

        n := 0;
        while not eof (f1) do
        begin
          readln (f1, lu);

          if n = 0 then n := 1  { init de lu puis prec }
          else if lu = prec then n := n+1
          else begin
                 { n est le nb d'occur. de prec, on l'écrit }
                 writeln (f2, prec, ' ', n);
                 n := 1;
               end;

          prec := lu;
        end;

        { On écrit le dernier mot, sauf s'il est vide }
        if (n > 0) and (length (prec) > 0)
        then writeln (f2, prec, ' ', n);

        close (f1); close (f2);
    End;
```

10. Tris

On déclare les types suivants :

```
CONST MaxMot = 63;
      MaxVec = 1000;
TYPE  TMot = string[MaxMot];
      TVecteur = array [1..MaxVec] of TMot;
```

Soit `v` un `TVecteur` de `nv` éléments ($1 \leq$ `nv` $\leq$ `MaxVec`).

10.1 Tri par permutation

Principe On est à l'étape `i`.

- ▷ Supposons déjà trié `v[1..i-1]`, et non traité `v[i..nv]`.
- ▷ On cherche le min dans `v[i..nv]`
- ▷ On le permute avec `v[i]`.
- ▷ Maintenant `v[1..i]` est trié.

1) Indice du minimum

Écrire la fonction `ind_min (v : TVecteur; nv, i : integer) : integer;` qui cherche le minimum dans `v` *à partir* de la position `i`, puis renvoie son indice.

2) Tri par permutation

Écrire la procédure `tri_permutation (var v : TVecteur; nv : integer);` qui effectue le tri par permutation du vecteur `v`.

Correction

La méthode a été vue en cours avec un exemple.

1) On écrit une variante de la fonction vue en cours pour le tri par remplacement.

```
Function ind_min (v : TVecteur; nv, i : integer) : integer;
Var j, k : integer;
Begin
    j := i;
    for k := i+1 to nv do
      if v[k] < v[j] then j := k:
    ind_min := j;
End;
```

2) On fait les étapes de `1` à `nv-1` ; en effet à la fin de l'étape `nv-1`, il ne reste plus qu'un seul élément dans la partie non traitée du vecteur, donc cet élément est à sa place.

```
Procedure tri_permutation (var v : TVecteur; nv : integer);
Var i, j : integer;
    t : TMot;
Begin
    for i := 1 to nv-1 do
    begin
      j := ind_min (v, nv, i);

      if j <> i
      then begin
             { permutation }
             t := v[i]; v[i] := v[j]; v[j] := t;
           end;
      { sinon i=j, permutation inutile ferait perdre du temps }
    end;
End;
```

Le coût *a été vu en cours*

Les performances sont meilleures que le tri par remplacement : il y a environ $\texttt{nv}^2 / 2$ comparaisons, et $\texttt{nv} / 2$ permutations en moyenne.

Par exemple si $\texttt{nv} = 1000$, on aura 500 000 comparaisons et 1500 affectations.

Il y a aussi un gain de place (1 seul vecteur en mémoire).

10.2 Tri à bulles optimisé

Rappel du tri à bulles On est à l'étape `i`.

- ▷ Supposons déjà trié `v[1..i-1]`, et non traité `v[i..nv]`.
- ▷ On parcourt `v[i..nv]` en descendant et, chaque fois que deux éléments consécutifs ne sont pas dans l'ordre, on les permute.
- ▷ En fin de parcours le min de `v[i..nv]` se retrouve dans `v[i]`.
- ▷ Maintenant `v[1..i]` est trié.

Tri à bulles optimisé

Si lors d'une étape `i`, aucune permutation n'a lieu, c'est que `[i..nv]` est déjà dans l'ordre, et le tri est fini.

1) Tri à bulle

Écrire la procédure `tri_bulle (var v : TVecteur; nv : integer);` qui effectue le tri à bulles du vecteur `v`.

2) Tri à bulle optimisé

Écrire la procédure `tri_bulle_opt (var v : TVecteur; nv : integer);` qui effectue la version optimisée du tri à bulles.

Correction

La méthode a été vue en cours.

1) On fait les étapes de `1` à `nv-1` ; en effet à la fin de l'étape `nv-1`, il ne reste plus qu'un seul élément dans la partie non traitée du vecteur, donc cet élément est à sa place.

```
Procedure tri_bulle (var v : TVecteur; nv : integer);
Var i, j : integer;
    t : TMot;
Begin
    for i := 1 to nv-1 do
      for j := nv downto i+1 do
        if v[j-1] > v[j] then { permutation }
        begin t := v[j-1]; v[j-1] := v[j]; v[j] := t; end;
End;
```

2) Version optimisée : on ne peut plus utiliser un `for i` puisqu'on veut pouvoir s'arrêter lorsqu'il n'y a pas eu de permutation.

On utilise un booléen `permu` qui est vrai si on a permuté.

```
Procedure tri_bulle_opt (var v : TVecteur; nv : integer);
Var i, j  : integer;
    t     : TMot;
    permu : boolean;
Begin
    { init }
    i := 1; permu := true;

    while permu do
    begin
      permu := false;

      for j := nv downto i+1 do
        if v[j-1] > v[j] then
        begin
          t := v[j-1]; v[j-1] := v[j]; v[j] := t;
          permu := true;
        end;

      i := i+1;
    end;
End;
```

Si il y a eu des permutations jusqu'au bout, le programme s'arrête bien. En effet lorsque `i` vaut `nv`, le `for j` n'est pas exécuté, donc `permu` reste faux, et donc le `while` s'arrête.

Bilan *vu en cours*

Le tri à bulles est une variante du tri par permutation, un peu moins efficace ; la version optimisée est quant à elle un peu meilleure que le tri par permutation.

Le nom du « tri à bulles » vient de ce que à chaque étape `i`, les éléments les plus « légers » remontent vers la surface, sont transportés à gauche.

On constate aussi que à chaque étape, l'ordre général est accru.

10.3 Suppléments pour le TP

Dans ces exercices on suppose que l'on a en entrée un fichier résultat du programme `MinusCol` du §8, c'est-à-dire : exactement un mot par ligne, en minuscule et sans espacement (sauf à la fin où il peut y avoir des lignes vides, qui donc ne contiennent pas de mots).

1) Chargement en mémoire

Faire une procédure `chargerMots` qui lit un fichier de mots et le stocke dans un vecteur `v : TVecteur`.

2) Sauvegarde sur disque

Faire une procédure `sauverMots` qui fait l'opération inverse, c'est-à-dire qui écrit dans un fichier texte les mots contenus dans le vecteur `v : TVecteur`, les uns en dessous des autres.

3) Programme `TriMots`

Faire un programme principal `TriMots` qui charge un fichier de mots en mémoire, tri ces mots par la méthode de votre choix, puis sauvegarde les mots triés dans un nouveau fichier.

4) Dictionnaire

Sur un fichier texte quelconque, appliquer dans l'ordre les programmes `MinusCol`, `TriMots` et `NuMots`, de façon à constituer un dictionnaire des mots du texte.

Correction

Le 1) et le 2) ont été vus en cours.