CHAPITRE 6

CHAINES DE CARACTERES ET AUTRES TYPES DE DONNEES

De même qu'il existe un grand choix d'organisation d'une programmation en tâches et sous-tâches pour un même algorithme, de même il existe une grande variété dans l'organisation des données. Pour les exemples étudiés jusqu'ici la question se posait peu, mais en d'autres domaines la réflexion que l'on peut avoir sur cette structuration des données n'est pas indépendante de l'algorithmique et peut être tout aussi longue. Certains langages évolués comme CAML permettent de travailler avec des types d'objets comme ceux qui sont habituellement définis en mathématiques : applications de E dans F etc. Mais aussi, ils permettent de parler de types abstraits où seules les opérations sur ce type sont connus des autres modules. Nous présentons ici les chaînes de caractères, les ensembles et les enregistrements.

En Pascal, les types simples prédéfinis sont "integer" (entiers codés sur deux octets), "real" (sur 6 octets), entiers et réels ayant donc des sens beaucoup plus restrictifs qu'en mathématiques, "char" (un octet), désignant les caractères qui sont codés par un nombre entre 0 et 255 puisqu'il y a $2^8 = 256$ codes possibles, "boolean" (un bit) représentant soit 0 pour le faux, soit 1 pour le vrai .

Types définis par énumération ou intervalle
Les types peuvent être définis par énumération comme carte = (coeur, carreau, trèfle, pique) par exemple, ou par intervalle , ainsi :

lettre = A..Z; ou encore entier = 0..100;

Pour un type défini par intervalle ou par énumération, "ord" est la fonction qui en donne le numéro d'ordre à partir de 0,. par exemple ord ('A') = 65.
On peut alors commander : for J := lundi to vendredi do ...

Les types structurés les plus usuels sont les tableaux array [... , ... , ...] of ...; (chapitre 5) les chaînes de caractères string[...] et les fichiers file of ... dont il sera question au chapitre 7.

Chaînes de caractères
Les chaînes de caractères (c'est-à-dire tous les mots ou plus généralement les assemblages de lettres et de chiffres) sont déclarées par le type "string" suivi du maximum de leur longueur placé entre crochets, leur nombre de caractères est obtenu par la fonction "length", et le n-ième caractère de C est obtenu par C[n] .
Le rang d'un élément faisant partie d'un type défini par énumération (et en particulier le code A.S.C.I.I.) est obtenu par la fonction "ord" dont la fonction inverse est "chr".
Exemple ord ('B') = 66 (consulter la table des codes ASCII), chr (67) = 'C' , succ ('C') = 'D' est la fonction successeur dans un type donné, et pred ('F') = 'E' est la fonction prédécesseur.
D'autres procédures et fonctions prédéfinies sont encore utiles, notamment pour certains des exercices figurant dans ce chapitre.
"delete" (C , I , J) transforme la chaîne C en lui retirant J caractères à partir du I-ième. Exemple si C est 'alligators', et que la procédure delete(C, 3 , 5) est exécutée, alors C devient 'alors'.
"insert" (C , S , I) est une procédure d'insertion de la chaîne C dans la chaîne S à la I-ième position. Exemple insert ('ligat' , C , 3) aura pour effet de redonner la valeur 'alligators' à la variable C.

"concat "(c1 , c2 , ...) renvoie la chaîne obtenue en concaténant tous ses arguments. Exemple concat ('Buda' , 'Pest') = 'BudaPest' . Concat est abrégé par le signe +
"copy" (C , I , J) renvoie les J caractères de C à partir du I-ième. substr (C, I, J) dans d'autres pascal.
Exemple copy ('loxodromie' , 4 , 2) = 'od'

Exemple de conversion de chaîne en valeur numérique

On met l'accent dans cet exemple sur le fait qu'une valeur numérique comme 123 et la succession des caractères "123" sont, pour la machine, deux objets complètement distincts. Le passage peut cependant se faire de l'un à l'autre.

Considèrons une suite d'entiers dont chaque terme est la somme des carrés des chiffres qui composent le terme précédent dans la suite. On démontre qu'une telle suite est toujours périodique et qu'il n'y a que deux périodes possibles. Essayer avec 7, avec 43 avec n'importe quel entier.
Traîter les termes de la suite comme des chaînes de caractères, est particulièrement adapté à ce problème. Chaque terme N est converti en la chaîne de ses chiffres CH grâce à la procédure prédéfinie "str", puis chacun des chiffres, ayant une valeur X, on en tire le carré qui s'ajoute à M, lequel va constituer le terme suivant; d'où la procédure "suite".
La procédure str (string = chaîne) possède une procédure inverse "val" à trois paramètres, la chaîne, la valeur qui en est déduite, et une valeur servant au contrôle.

```
program suitebizarre ;
        var N : integer;

procedure suite (N : integer);
        var C, M, I, X : integer ; CH : string [6];
{Est une procédure récursive décryptant chaque chiffre X de la chaîne CH correspondant à la valeur N, et
calculant la valeur suivante M }
        begin
        write (N,'/'); { Le nombre est écrit suivi d'un séparateur / }
        if N in [1,4]   { On utilise un ensemble pour tester si N est 1ou 4}
                then writeln (' fini ')
                else    begin M := 0;
                        str (N, CH);
                        for I := 1 to length (CH) do
                                begin val (CH[I], X, C); M := M+X*X end ;
                        suite (M) end
        end;

begin write ('Donnez un entier positif ') ; readln (N) ; suite (N) end.
```

Remarque : on pourra définir la somme des carrés des chiffres récursivement par :
som (n) = si n < 10 alors n^2 sinon som (n div 10) + sqr (som (n mod 10))

Exemple pour un départ à 7 : 7 / 49 / 97 / 130 / 10 / 1 / fini.

Exemple d'utilisation d'ensembles : suite partitionnant les entiers avec ses différences

Le programme qui suit utilise le type ensemble "set of..." pour lequel "in" est l'appartenance, + est l'union et [ ] représente l'ensemble vide. Ce type de représentation est assez limité en Pascal, c'est pourquoi on l'emploiera peu.
On considère la suite croissante d'entiers telle qu'avec la suite formée par toutes les différences de termes consécutifs de cette suite, les deux suites ainsi formées constituent une partition des entiers.

Le début est nécéssairement 1 (+ 2 =) 3 (+ 4 =) 7 (+ 5 =) 13 (+ 6 =) 19 (+ 8 =) etc ...

```
program partition ;
                const M = 100; { une constante dans le programme }
                type entier = 0..255; { On choisit de définir un nouveau type }
                var N, D : integer ; S : set of integer ;

begin N := 1; D := 2; S := [1] ; { initialisation des premiers termes }
write (N) ;
while N < M do  if (N + D)  in  S       then D := D + 1
                                        else    begin N := N + D; S := S + [N];
                                                write ('  (+', D, ')-->', N); D := D + 1
                                                while D in S do D := D + 1
                                                end;
end.
```

Pour analyser le programme, le mieux est de le faire tourner à la main sur le papier pour M inférieur à 60 .
On appelle pour cela N le terme courant de la première suite, D la différence avec le terme suivant, et S l'ensemble des termes de la première suite jusqu'à N.
Si N+D n'est pas déjà dans l'ensemble S, on l'y place, on affiche une indication et on cherche la plus petite différence D suivante qui n'est pas dans S.

Exemple d'utilisation de tableau de booléen : les nombres premiers

Le crible d'Eratosthène est un algorithme de recherche des nombres premiers jusqu'à M, qui n'est pas fondé sur des tests de divisibilité mais sur l'élimination des multiples des premiers éléments restants (d'où le nom de nombre premier). 2 étant le départ, on l'inscrit comme étant premier, puis on raye tous ses multiples, le premier non rayé est alors 3 qui est à son tour déclaré premier, et on raye ses multiples, le premier suivant est 5 etc...

Cet algorithme est très rapide, mais utilise l'ensemble de tous les entiers de 1 à M. On réduit considérablement cet encombrement de la mémoire en utilisant un tableau de booléens où l'élément de rang N ayant la valeur "false" signifiera qu'il a été rayé.

```
program eratosthene;
                const M = 10000;
                var P, N : integer; T : array[2..M] of boolean; { Ce tableau T ne sera rempli que de 0 ou de 1 }
begin
for P := 2 to M do crib[P] := true; {initialisation de tous les entiers entre 2 et M comme non rayés}
P := 2;
repeat write (P : 6); N := P;
                repeat crib [N] := false; N := N + P until N > M;
                repeat P := P + 1 until crib [P]
until P > M
end.
```

Remarque : on peut même rayer P, puis P^2, P^2 + P etc... car les autres ont déjà été rayés.

Les "records" ou "enregistrements"

Quoique d'une utilisation un peu compliquée au départ, le constructeur de type structuré "record", permet de créer des "enregistrements" ou "articles" réunissant des données de types distincts, voire avec "case", en nombre variable, par exemple:

```
etudiant = record       nom , prenom            : string[20];
                              année             : integer;
                              sexe              : boolean;
                        case controle of math   : (alg, ana, géom : integer);
                                         info   : (exam, projet : integer);
                                         chinois: (écrit, oral : integer) end;
```

On descend en cascade dans un tel arbre par le point, ainsi si A est un étudiant, A.info.exam sera sa note d'examen en informatique. Les parenthèses des trois dernières lignes sont indispensables, quant au "end", il termine à la fois le "case" et le "record" ici.

Exemple de types donnés par énumération : les annonces du bridge

Dans une main de 13 cartes, on totalise les points d'honneur 4 pour as, 3 pour le roi, jusqu'à 1, et les points dedistribution : 3, 2 ou 1 suivant que l'on a respectivement 0, 1 ou 2 cartes seulement d'une couleur. On suppose l'existence d'une fonction "ord" qui donne le rang à partir de 0 d'un objet dans un type énuméré. Pour créér cette fonction, on peut définir :

```
type        couleur = (trefle, carreau, coeur, pique);
            valeur = (as, roi, dame, valet, dix, .... , deux);
            carte = record coul : couleur; val : valeur end;
            main = array [1..13] of carte;

function distribution (m : main) : integer;
            var compte : array [trefle..pique] of integer;
                  d, i : integer; c : couleur;
            begin
            for c := trefle to pique do compte [c] := 0;
            d := 0;
            for i := 1 to 13 do     begin compte[m[i].coul] := 1 + compte  [m[i].coul];
                                    if main [i].val < dix then d := d + 4 - ord (main[i].val)
            for c := trefle to pique  do  case  compte[c]   of  0 : d := d + 3;
                                                                1 : d := d + 2;
                                                                2 : d := d + 1 end;
            distribution := d
            end;
```

Remarque : si on déclare le type "valeur" = (sansvaleur, valet, dame, roi, as), alors on a une simplification d := d + ord (main[i].val)

6-1° La machine à écrire endiablée: faire en sorte que la pression de chaque lettre au clavier provoque la sortie sur l'écran, non pas de la lettre mais d'un mot, par exemple DIABLE au lieu de D etc... (utiliser c := getchar;).

6-2° Epeler un mot en séparant les lettres par un tiret.

6-3° Faire apparaître la liste des caractères et de leurs codes ASCII

```
program ascii;      var i: integer;
                    begin for i := 33 to 255 do write (i : 4,'-->', chr (i),'  ') end.
```

6-4° Ecrire un mot à l'envers. Essayer sur "Laval" ou "Esope reste ici et se repose".

6-5° Produire à l'écran une pyramide avec les parties centrales d'un mot.

6-6° Doubler les lettres d'un mot.

6-7° Effectuer les permutations circulaires d'un mot. (LUC UCL CLU)

6-8° **Cryptogramme** : Un prénom est codé par translation de longueur aléatoire, (exemple LUC par la translation -3 devient IRZ). La même méthode appliquée aux mots d'une phrase, en les comparant à ceux d'un petit lexique, donne des résultats extraordinaires.

6-9° On veut épeler un nom de trente lettres maximum à la façon de N comme NOEMIE etc... On définit pour cela un type que l'on appelle "nom", et qui représente des objets différents de ceux typés par "prenom".

```
program telephone ;
        type    prenom  = string [15]; nom = string [30];
        var P : array[1..26] of prenom ; N : nom;
{Le fait de ranger les prénoms par ordre alphabétique dans un tableau de vingt-six éléments permet de calculer très facilement la place de chacun d'entre eux suivant sa première lettre. Etant donné un caractère qui est noté X (pas nécessairement la lettre X !), la fonction "ord" renvoie son code, il suffit alors de savoir que le code (en décimal) de la lettre A est 65, pour obtenir l'indice où se trouve le prénom commençant par ce caractère dans le tableau. C'est ce que réalise la fonction "comme".}

function comme (X : char) : prenom;
        begin   comme := P [ ord (X) - 64 ]     end;
procedure epelle (Y : nom);
        var I : integer ;
        begin for I := 1 to length (Y) do writeln (Y[I],' comme ', comme (Y[I]) ) end;
begin { Début du programme }
        P[1] := 'ANNA'; P[2] := 'BASILE'; P[3] := 'CESAR'; P[4] := 'DANIEL';
        P[5] := 'ERNEST'; P[6] := 'FABRICE'; P[7] := 'GASTON';
        { - Ici on continuera avec les prénoms de son choix - }
        write ('Donnez un nom ') ; readln (N) ; epelle (N)
end.
```

6-10° **Nombre de Champernowne** 0,123456789101112...construit avec les entiers successifs, l'écrire en se limitant à 20 lignes d'écran.

6-11° Chercher les nombres entiers positifs entre 1 et 500, qui sont égaux à la somme des cubes de leurs chiffres (on doit en trouver cinq).

6-12° Trouver le plus petit entier N dont le chiffre des unités est 6, et tel que quand on efface 6, et que celui-ci est placé à gauche du nombre sans modifier les autres chiffres, on obtienne le quadruple de N.

6-13° Réaliser une fonction binaire donnant la chaîne des chiffres binaires correspondant à un entier.

```
function binaire (n : integer) : string;
        begin   case  n  of     0 : binaire := '0';
                                1 : binaire := '1';
                                otherwise binaire := concat (binaire(n div 2), binaire (n mod 2)) end end;
```

6-14° **Distance de Levenstein entre deux chaînes :** c'est le coût minimal calculé comme somme des coûts des opérations élémentaires de substitution $f(a, b)$, d'insertion $f(\emptyset, a)$ et de destruction $f(a, \emptyset)$ de caractères. Programmer l'algorithme de Wagner-Fisher qui détermine ce coût ainsi que la suite des opérations pour 2 chaînes $x = a_1a_2 \ldots a_n$ et $y = b_1b_2 \ldots b_m$, on note x_i le préfixe $a_1a_2 \ldots a_i$ et $\partial(i, j)$ la distance entre x_i et y_j.

$\partial(0, 0) \leftarrow 0$

Pour i de 1 à n faire : $\partial(i, 0) \leftarrow \partial(i-1, 0) + f(a_i, \emptyset)$

Pour j de 1 à m faire : $\partial(0, j) \leftarrow \partial(0, j-1) + f(\emptyset, b_j)$

Pour i de 1 à n faire / Pour j de 1 à m :

$m1 \leftarrow \partial(i-1, j-1) + f(a_i, b_j)$

$m2 \leftarrow \partial(i-1, j) + f(a_i, \emptyset)$

$m3 \leftarrow \partial(i, j-1) + f(\emptyset, b_j)$

$\partial(i, j) \leftarrow \min (m1, m2, m3)$

La distance entre les chaîne x et y est alors $\partial(n, m)$.

Essayer à la main pour x = "abc" et y = "dacb".

6-15° Permutations ?
Voici une procédure où A contient 7 lettres, U et V étant deux tableaux d'entiers indexés de 1 à 8000, "ech" est la procedure d'échange de deux variables entières :

```
procedure remplissage (n: integer) ; {fonctionne pour 0 ≤ n ≤ 7}
                var p, k, m, s : integer;
begin v[1] := 1; v[2] := 1; m := 2;
for p := 3  to n do         begin k := 0;
                            repeat  for  s := 1  to  p - 1  do    begin u[k*p+s] := s; u[k*p+p+s] :=p-s end
                                    u[k*p+p] := v[k+1] ;
                                    u[k*p+p+p] := v[k+2] + 1 ;
                                    k := k+2
                            until k = m-2;
                            m := m*p;
                            for k := 1 to m do v[k] := u[k]
                            {marque 1} end; {marque 2}
for p := 1 to m do          begin ech ( A[v[p]], A[v[p] + 1]);
                             write (p, ' : ');
                            for k := 1 to n do write (A[k]) ; writeln
                            end;
end;
```

Quelle est la valeur de m au début de la seconde grande boucle (marque 2) ?
Dérouler à la main le programme pour n = 2 puis pour n = 3.
Que contient le tableau V à la marque 1 ?
Que va-t-il se produire en définitive ?

6-16° Suite de Kaprekar : si u_0 est un nombre de 4 chiffres, on calcule u1 comme la différence entre le nombre obtenu en ordonnant les chiffres de u_0 dans l'ordre décroissant et celui obtenu avec l'ordre croissant. Recommencer jusqu'à obtenir 6174.

6-17° Un montage électrique formé de résistance va être codé par une chaîne de "s" pour série et "p" pour parallèle, suivis chaque fois de deux sous-circuits, les résistances étant des chiffres de 1 à 9.
a) Programmer la fonction circuit lisant les caractères un à un au clavier et donnant la résistance du circuit en appelant deux autres fonctions.
b) Programmer une fonction résitance faisant la même chose toute seule.

```
program resis;
                var a  : integer;

function chiffre (c : char) : boolean;
                begin chiffre := (47 < ord(c)) and (ord(c) < 58) end;

function parallele : real; forward;
function serie : real; forward;
function circuit : real;
                var c : char;
                begin c := readchar; write(c);
                if chiffre (c)     then circuit := ord (c) - 48
                                   else if c = 'p'      then circuit := parallele
                                                        else circuit := serie end;

function parallele ;
                var a, b : real;
                begin a := circuit; b := circuit ; parallele := a*b / (a+b) end;

function serie ;
                var a, b : real;
                begin a := circuit; b := circuit ; serie := a+b end;
```

Une autre solution avec un point de vue moins "effet de bord" est :

```
function resistance (s : string; var i: integer): real; {autre possibilité}
        var a, b : real; {i est la position courante dans la lecture de s}
        begin if chiffre(s[i])     then resistance := ord (s[i]) - 48
                                   else if s[i] = 's'   then     begin i := i + 1; a:= resistance (s, i);
                                                                 b:= resistance (s, i);
                                                                 resistance := a + b end
                                                        else if s[i] = 'p' then      begin i := i + 1;
                                                                                     a := resistance (s, i);
                                                                                     b := resistance (s, i);
                                                                                     resistance := a*b / (a+b) end
        end;
begin a := 1; {pour la première solution :} {writeln ('résistance ', circuit : 8 : 3);}
{ pour la seconde } writeln (resistance ('ss5ps23s4p236', a) : 8 : 3) end. {donnera 15}
```

6-18° **Alphabet Morse** faire un programme de traduction de message en morse écrit avec des points et des tirets, et inversement.

```
program tradmorse ;
type mot = string [9];
var PH : string; M : array ['A'..'Z'] of mot; {On ne considère que les majuscules }
procedure init ;
        begin M['A'] := '._'; M['B'] := '_...'; M['C'] := '_._.'; M['D'] := '_..'; M['E'] := '.'; M['F'] := '.._.';
        M['G'] := '_ _.'; M['H'] := '....'; M['I'] := '..'; M['J'] := '._ _ _'; M['K'] := '_._';
        M['L'] := '._..'; M['M'] := '_ _'; M['N'] := '_.'; M['O'] := '_ _ _'; M['P'] := '._ _.';
        M['Q'] := '_ _._'; M['R'] := '._.'; M['S'] := '...'; M['T'] := '_'; M['U'] := '.._';
        M['V'] := '..._'; M['W'] := '._ _'; M['X'] := '_.._'; M['Y'] := '_._ _'; M['Z'] := '_ _..';
        end;
procedure son (C : mot); {il faut un pascal ayant une procédure capable de générer un son }
        var i : integer ;
        begin for i := 1 to length (C) do
                if C[i] = '-' then sound (130, 5) else sound (440, 2) end;
procedure traducmorse (C : string); {écrit et sonorise la suite C de caractères}
        var i : integer; X : char;
        begin for i := 1 to length (C) do
                begin X := C[i] ;
                if X in ['A'..'Z']          then begin write (M[X], '  '); son (M[X]) end
                                            else write (' / ')
        end end ;
procedure alphabet ;
        var Z : char ;
        begin for Z := 'A' to 'Z' do begin write (Z : 5, ' : ', M[Z] : 5); son (Z) end; writeln end;

begin init; alphabet; write ( 'Ecrivez une phrase en majuscules :'); readln (PH); traducmorse (PH) end.
```

6-19° Faire fonctionner à la main le programme du crible d'Eratosthène, jusqu'à une valeur raisonnable de N.

6-20° Ecrire l'algorithme d'Eratosthène en utilisant deux ensembles PREM, CRIBLE, plutôt qu'un tableau.

6-21° Ecrire un programme réalisant un ensemble formé par les lettres données parmi tous les caractères entrés au clavier sur une ligne (on peut utiliser le prédicat "eoln" indiquant la fin d'une ligne).

```
program mots;
        type lettre = 'A'..'Z'; mot = set of lettre; {deux définitions de nouveaux types}
        var ch : lettre; m : mot;
begin m := []; {initialisation d'un ensemble vide} while not eoln do begin read (ch); m := m + [ch] end end.
```

6-22° Programmer une fonction récursive lisant une suite de caractères sans parenthèses suposée représenter une molécule comme OHH ou CHOHHCHOCHHHHH et devant donner le résultat (OH)H pour la première et (CH(OH)H)(CH(O(CHHH))H) pour la seconde. Les radicaux sont de la forme H ou bien ($XR_1R_2...R_n$) si les R_i sont eux-même des radicaux et X un corps de valence n. Les radicaux ont une connexion libre, une molécule est formée par deux radicaux.

La solution consiste à avancer (pos) dans la lecture de la chaîne en sachant que l'on va devoir écrire deux radicaux.

```
function valence (x : char) : integer ;
                begin case x of 'C' : 4; 'O' : 2 ; 'H' : 1 end end; { On peut mettre davantage de cas }

function radical (n, l : integer; ch : string; var pos : integer) : string;
{n est le nombre de connexions libres, l la longueur de la chaîne ch}
                var x : char;
begin           if (n = 0) or (pos > l)    then radical := ''
                else     begin pos := pos + 1; x := ch [pos];
                         if x = 'H'         then radical := x + radical (n-1,  l, ch, pos)
                                            else radical := '(' + x + radical (valence (x), l, ch, pos) + ')' +
                                                                     radical (n-1, l, ch, pos)
end;
```

On appelera radical (2, ch[0], ch, pos) avec pos := 1

6-23° **Les différentes couches électronique** d'un atome correspondent à des niveaux d'énergie, les couches sont numérotées 1, 2, 3, 4, ... (n) et contiennent chacune 4 sous couches (k) nommées s, p, d, f. L'ordre de remplissage des orbitales se fait suivant n + k croissant, si deux électrons doivent avoir le même n + k, c'est celui du plus petit n qui est rempli en premier. Ainsi pour le manganèse de numéro atomique Z = 25, on aura la disposition électronique 1s2-2s2-2p6-3s2-3p6-4s2-3d5. La sous-couche k possède un maximum de 4k-2 électrons (donc la couche n qui possède n sous-couches en a $2n^2$).
En se servant d'un tableau tab[1..9, 0..3] d'entiers qui sera construit au fur et à mesure, programmer la fonction structure (Z : integer) : string; donnant la disposition électronique des corps purs.

6-24° **Code de Vigenere.** Pour crypter un message, le code de César consiste à décaler l'alphabet en comptant l'espace pour 0, A, pour 1, ... si par exemple on décale de 3, A sera codé D, et Y sera A. L'analyse statistique vient rapidement à bout de ces codes, aussi le code de Vigenere reprend ce principe mais en codant grâce à une clé, un mot tel que "TIGRE JAUNE", qui reproduit indéfiniement permet de donner les différents décalages le long du message. Ainsi pour coder "RENDEZ VOUS DEUX HEURES TROCADERO", on placera la clé donnant le nombre de lettres à ajouter :

```
R  E  N  D E Z     V  O  U  S     D   EUX HEURES TROCADERO
T  I  G  R E     J   A  U  N  E     T   IGRE JAUNE TIGRE JAUNE TIGRE JAUNE
20 9  7  18 5  0 10  1  21 14  5 0  20
K  N U  V J  Z  J   W  I   H  X     X
```

Programmer les fonctions d'encodage et de décodage.

```
function encode (texte, cle : string) : string;
                var st : string; i, j : integer;
begin j := 0; st := '';
for i := 1 to length (texte) do     begin if j < length (cle) then j := j + 1 else j := 1;
                                    st := st + chr (((asc(texte[i]) + asc(cle[j]) - 128) mod 27) + 64) end;
encode := st
end;
```

Pour "decode", il suffit d'écrire ((asc(texte[i]) - asc (cle[j])) mod 27)

6-25° L'horloge, la transcription littérale de l'heure en adoptant le parler usuel (cinq heures moins vingt ; midi et quart ; etc ...) fournit un bel exemple de programme se découpant en petites fonctions conduisant à des emboîtements de conditions qui demandent une certaine attention.

Rappel : round est la fonction Pascal renvoyant l'entier le plus proche, ainsi, round (4.63) = 5, alors que trunc (4.23) = 4 .

```
program horloge ;
        type mot = string [40];
        var H, M, S, X : integer;
            TH : array [1..13] of mot ; TM : array[1..6] of mot;
        { TH est le tableau contenant les données une, deux,trois, etc...  alors que TM contient les noms
des minutes cinq, dix, quart, etc... }

 function minute (M : integer) : mot;
{produit les chaînes "moins dix" , "et quart", etc ... suivant la valeur M des minutes, en arrondissant de cinq en
cinq minutes. Lorsque la demie est passée, on fait bien sûr la soustraction avec 60 minutes.}
        var A : mot ;
        begin  case  M  of  3..28          : A := TM[M div 5)];
                            29..33         : A := TM[6];
                            34..57         : A := TM[(60 - M) div 5];
                           otherwise       A := ' '   end;
        if A=TM[3] then if M < 30 then A := ' et ' + A  else A := 'le ' + A;
                                  { cas de "et quart" ou "moins le quart"}
        case  M   of       34..57          : minute := ' moins ' + A;
                           otherwise       minute := ' ' + A end
        end;

function heure (H : integer) : mot;
{produit les chaînes "midi",  "dix heures", etc ... en prenant garde au pluriel, et au fait que "midi" et "minuit" ne
sont pas suivis de "heures"}
        var A : mot;
        begin if (H=0) or (H=24)        then A := 'MINUIT'
                                        else IF H > 12  then A := TH[H-12]
                                                        else A := TH[H];
        if (H mod 12) <> 0      then    begin A := A + ' heure';
                                        if (H <> 1) and (H <> 13) then A := A + 's';
                                        end;
        heure := A
        end;

function ampm (H : integer) : mot; { produit le complément "du matin", "du soir",  etc}
        begin case H of 1..11    : AMPM := ' du matin';
                        13..17   : AMPM := ' de l''après-midi';
                        18..23   : AMPM := ' du soir';
                       otherwise AMPM := "  end
        end;

function litteral (H, M : integer) : mot;
        begin
        if M > 33       hen litteral := heure (H+1) + minute (M) + ampm (H+1) + '                    '
                        else litteral := heure (H) + minute (M) + ampm (H) + '                    '
        { Les blancs sont destinés à couvrir l'inscription antérieure } end;

procedure remisalheure (var H : integer ; var  M : integer );
                begin clrscr; { Instruction pour effacer l'écran }
                write ('Quelle heure ? '); readln (H);
                write ('Combien de minutes ? '); readln (M);
                clrscr { sur PC, "clearscreen" sur MAC } end;
```

```
begin { début du programme }
                        TM[1] := 'CINQ'; TM[2] := 'DIX';TM[3] := 'QUART';
                        TM[4] :='VINGT'; TM[5] := 'VINGT-CINQ'; TM[6] :='ET DEMIE';
                        TH[1] := 'UNE'; TH[2] := 'DEUX';TH[3] := 'TROIS';
                        TH[4] := 'QUATRE'; TH[5] := 'CINQ'; TH[6] := 'SIX';
                        TH[7] := 'SEPT'; TH[8] := 'HUIT'; TH[9] := 'NEUF';
                        TH[10] := 'DIX'; TH[11] := 'ONZE'; TH[12] := 'MIDI';
remisalheure (H, M);  { Initialisation de H, M et S }
S := 0;
repeat          gotoxy (5, 12); {permet de localiser le curseur à la colonne 5 et la ligne 12}
                write (H, ' h ', M,' mn ', S, ' s ', litteral (H, M));
                if S = 60 then begin S := 0 ; M := M + 1 end;
                if M = 60 then begin M := 0 ; H := H + 1 end;
                {Ici il suffit d'introduire une pause pour que l'horloge soit réellement à l'heure.}
                if H = 24 then H := 0;
                S := S+1
until keypressed  {Boucle interrompue par la pression d'une touche au clavier.}
end.
```

6-26° Transcription littérale des nombres Transcrire littéralement en français un entier entre 1 et 1000 (exemple 31 = trente et un ; 279 = deux cent soixante-dix-neuf) puis prévoir les versions belges et suisses.
Reprendre ce problème avec une autre langue, notamment une numération vigésimale (breton ou basque) ou encore le cambodgien dont la numération quinaire est assez régulière : muy pir bei buon pram sont les 5 premières unités et 10 = dap, 100 = roy, 1000 = pan, ce qui fait que:
2369 = pir-pan-bei-roy-pram-muy-dap-pram-buon

L'analyse est un travail d'expression française, ce qui n'a rien à voir avec un commentaire de programme fait après coup. Il convient de commencer par les grandes lignes du problème, on peut pour cela distinguer :
_ les numérations régulières (chinois, japonais, corréen, finnois, espéranto...) où seuls des mots de un à dix suffisent avec cent, mille, ... ainsi 67 se dira six-dix sept.
243781 = ducent kvardek tri mil sepcent okdek unu
= (2*100 + 4*10 + 3)*1000 + 7*100 + 8*10 + 1 en esperanto.
_ semi-régulières comme la plupart des langues européennes où il faut un mot pour chaque nombre de 1 à 20 et pour 30, 40, ...
_ les numérations vigésimales (langues celtiques, mexicaines ...) nécessitant des mots pour les nombres de 1 à 20 puis pour 100... ainsi 56 serait deux-vingt seize.
_ les numérations quinaires (cambodgien, wolof du Sénégal ...)
3267 = niente djouni niarre temer djiourome bene fonk djiourome niarre
= 3*100 + 2*100 + (5 + 1)*10 + (5 + 2) en wolof.
_ le français, qui est de loin le plus compliqué avec ses survivances vigésimales (base 20), telles que quatre-vingt et l'hôpital des quinze-vingts (300 lits). Si L $\in$ {F, B, S} est une langue telle qur français, belge ou suisse, et si C, D, U sont les chiffres de centaine, dizaine et unité, on doit au minimum donner des conditions logiques telles que :
Si L = F et (D = 7 ou D = 9) alors la transcription est celle correspondant à D-1 et U+10
Si U = 1 et (L = F $\Rightarrow$ (D $\neq$ 8 et D $\neq$ 9)) et (L = B $\Rightarrow$ (D $\neq$ 8)) alors on intercale une conjonction "et". Mais est-ce l'écriture logique la plus simple ? (noter que "huitante" se dit en Suisse et "octante" subsiste au Canada)
Les pluriels (mille invariable, vingt et cent au pluriel s'ils ne sont pas suivis, le cas de million etc ...) ne sont pas considérés ici.

Parallèlement à l'exposé de l'algorithme, on doit s'interroger sur la représentation des données, le problème est intéressant car il n' a pas de solution unique. En effet, on peut tout dire au sein même des procédures, cela n'est guère recommandé en regard de la souplesse du programme. Cependant il n'est pas possible de séparer véritablement les données des algorithmes. Car même si, comme cela paraît naturel, on déclare un tableau à double entrée renfermant les mots nécessaires aux différentes langues, et même si on prévoit autant de procédures que les types de numérations énumérés ci-dessus, alors on est en butte à une foule de cas particulier.

Ainsi le japonais, le grec ou l'éthiopien prévoient-ils un mot spécial pour 10000, le breton, vigésimal, dit cependant "hanter-kant" (demi-cent) pour 50, l'allemand et l'arabe inversent-ils les dizaine et unité ("ein und zwanzig" pour 21), certaines langues disent-elles "cent", et d'autres "un-cent" etc ....
Du point de vue de la programmation il est évident qu'il faut arriver à définir deux ou trois fonctions telles que :

C → mot de la centaine
D → mot de la dizaine
U → mot pour l'unité

Mais encore faut-il que ce soient de véritables fonctions, et non pas des morceaux de programme agissant sur des variables globales. Des fonctions sans paramètre n'ont pas d'intérêt, il faut prévoir que pour un nombre comme 164 238 475 102, elles seront appelées quatre fois chacune. On peut également prévoir la forme archaïque 1789 = dix-sept cent quatre-vingt neuf, pour laquelle la même fonction serait appelée pour les paramètres 1 et 7, puis pour 8 et 9.

```
program nombres;
        type mot = string[40];
        var  TU : array [0..19] of string[10]; TD : array[2..9] of string[10];
        TE : array [1..8] of string[10];

procedure initialise;
begin   TU[0]:='';TU[1]:='un '; TU[2] := 'deux '; TU[3]:='trois '; TU[4] := 'quatre ';
        TU[5] :='cinq '; TU[6] := 'six '; TU[7] := 'sept '; TU[8]:='huit '; TU[9] := 'neuf ';
        TU[10] :='dix '; TU[11]:='onze '; TU[12] := 'douze '; TU[13] := 'treize ';
        TU[14] :='quatorze '; TU[15] :=' quinze '; TU[16] := 'seize '; TU[17] := 'dix-sept ';
        TU[18] :='dix-huit '; TU[19] := 'dix-neuf ';
        TD[2] :='vingt '; TD[3] := 'trente ';TD[4]:='quarante '; TD[5] := 'cinquante ';
        TD[6] :='soixante '; TD[7] := 'septante ';TD[8] := 'huitante '; TD[9] := 'nonante ';
        TE[1] := 'mille '; TE[2] := 'million'; TE[3] := 'milliard'; TE[4] := 'trillion '  end;

procedure decomp (N : integer; var C, D, U : integer); {calcule les 3 derniers chiffres C D U d'un nombre N}
        function unite (X : integer) : integer; {fonction locale à "decomp"}
                begin unite := X mod 10 end;
begin U := unite (N); D := unite (N div 10) ; C := unite (N div 100) end;

function diz (D, U : integer; L : char): mot; {traduit les nombres < 99}
begin if  D  <  2  then diz := TU [10*D+U]
        else if ((D = 7) or (D = 9)) and (L='F') then diz: = diz (D-1, U+10, L)
        else if (D=8) and (L <>'S') then diz := 'quatre-vingt '+ TU[U]
        else if U mod 10 = 1 then diz := TD[D] + 'et ' + TU[U]
        else diz:=TD[D] + TU[U]
end;

function cent (N : integer; L : char)  mot;
{traduit les nombres < 999, les pluriels de cent et quatre-vingt ont été négligés ici }
        var C, D, U:integer;
begin decomp (N, C, D, U);
case C of       0 : cent := diz(D, U, L);
        1 : cent := 'cent ' + diz (D, U, L);
        otherwise cent := TU[C] + 'cent ' + diz (D, U, L) end
end;

begin {début du programme}  initialise;
writeln ('278 = ', cent(278, 'F'), ' en français');
writeln ('394 = ', cent(394, 'B'), ' en belge');
writeln ('485 = ', cent(485, 'S'), ' en suisse');
end.
```

Voici maintenant une extension pour des nombres plus grands N, considérés comme chaînes de caractères.

```
function echelle (N : mot; K : integer; L : char) : mot; {transcrit les trois derniers chiffres de N en fonction de
l'échelle K, K est l'exposant de 10 3. Val est une fonction donnant la valeur d'une chaîne composée de chiffres,
elle est laissée en exercice.}
var M : integer;
begin  M := val (N);
case M of       0 : echele := '';
                1 : if K = 1 then echelle := TE[1] else echelle := TU[1] + TE[K] + ' ' ;
                otherwise if K=1        then echelle := cent (M, L) + TE[1]
                                        else echelle := cent (M, L) + TE[K] + 's ' end end;

function trans (N : mot; L : char) : mot; {traduit les nombres < 10^12 écrits comme mots, trans est à utiliser en
place de "cent" dans toute utilisation.}
var K : integer;
begin K := trunc ((length (N) - 1) /3);
if K = 0        then trans := cent (val (N), L)
                else trans := echelle (copy (N, 1, length (N) - 3*K), K, L)
                                + trans (copy (N, length (N) - 3*K+1, 3*K), L)
end;
```

6-27° Logiciel de solfège 𝄞 ♪♫ Il s'agit de créer un logiciel ayant le menu minimum suivant :
a) Exercice de lecture sur la portée (une suite de notes sera reconnue par leur noms entrés au clavier par l'utilisateur)
b) Dictée musicale (même chose mais reconnaissance auditive en supposant une procédure "sound")
c) Jeu libre au clavier (un octave et demi avec les touches s=si, d=do, éventuellement un octave de plus avec les majuscules) chaque note devra s'inscrire sur la portée tandis qu'elle sera jouée (éventuellement avec la durée de pression de la touche). L'utilisateur dira auparavant s'il veut créér un fichier enrégistrant son morceau.
d) Enregistrement d'un air dans un fichier, les notes étant écrites par leurs noms au clavier.
e) Jeu d'une mélodie déjà entrée dans un fichier de notes (prévoir au moins deux ou trois petits airs de démonstration, la partition se déroulant à l'écran au cours de l'exécution du morceau)

On ne considèrera que la clé de sol. Deux ou trois lignes de portée sont suffisantes à l'écran. Les notes repésentées d'une manière adéquate iront seulement du LA_2 au MI_4 en notant LA_{-1} la note la plus basse du piano, LA_6 la plus haute et DO_3 celle du milieu. La fréquence du LA_3 est 435 Hz. On rappelle qu'un octave est en fait une succession de 12 demi-tons dont les fréquences sont disposées en suite géométrique dont la raison est la racine douzième de 2 (DO_{n+1} a une fréquence double de celle de DO_n). Les temps et les figures correspondantes (rondes, blanches, noires, croches ...) peuvent être ignorées et n'être prises en compte par la suite.

6-28° Crible d'Eratosthène avec ensembles

```
program crible;
        const n = 250; type entier = 0..n; var p, m : entier; C set of entier;
begin C := [0..n]; p := 1;
repeat repeat p := p+1; until p in C; write(p, ' '); while m <= n do begin C := C - [m]; m := m+p end
until C = [] end.
```