CHAPITRE 2

INSTRUCTIONS DE BASE

Tous les langages de programmation impératifs possèdent des instructions pour affecter, sélectionner, débrancher ou itérer, et des procédures prédéfinies d'entrées et de sorties des données.
L'affectation est l'instruction permettant d'attribuer une valeur à une variable.
Par exemple, pour échanger les valeurs de deux variables de noms A et B, il est logique de passer par un intermédiaire (une variable "tampon") X de la façon suivante: X := A ; A := B ; B := X ; A et B ont été réaffectés au moyen de l'affectation temporaire d'une troisième variable.
Sélectionner, c'est décrire les termes d'alternatives "si ... alors ... sinon ... " qui vont s'écrirent par des "if ... then ... else" emboîtés.
Itérer, c'est répéter, par exemple, on veut afficher une table de multiplication ou une table de conversion entre unités physiques etc... Trois instructions font cela couramment .

Exemple de cas emboîtés : maximum de trois nombres

L'exemple de la recherche du maximum de trois nombres donnés montre l'utilisation de l'instuction "if...then...else..." sélectionnant les actions à effectuer suivant la valeur de vérité de la proposition logique qui suit le mot "if" et que l'on appelle un test. Dans le cas où le test est assuré, la ou les instructions intercalées entre "then" et "else" sont exécutées et ce peut d'ailleurs être une autre instruction de sélection comme dans l'exemple présent.

Il est alors essentiel de rédiger de la façon suivante, les termes "then" et "else" correspondant aux deux alternatives d'un même test, étant placés l'un en dessous de l'autre. Il n'y a alors pas de confusion possible pour le lecteur humain même débutant. L'analyseur syntaxique n'a pas ces problèmes; si tout était écrit à la file mais conformément à la syntaxe, celui-ci s'y retrouverait alors que le texte serait illisible pour l'homme.

```
program max ;
        var A, B, C : real;
begin   write ('Donnez trois nombres ') ;
        readln (A , B, C) ;
        if   A < B        then  if B  <  C          then write (C)
                                                    else write (B)
                          else  if  A  <  C         then write (C)
                                                    else write (A)
end.
```

On remarquera plusieurs détails du Pascal : la déclaration obligatoire des noms A, B, C des trois variables utilisées, avec leur type "réel"; la procédure "write" permettant la copie (en général sur l'écran) de ses arguments; et la procédure "read" permettant la lecture d'arguments (en général depuis le clavier).
Notons à ce propos que les trois valeurs doivent être rentrées au clavier séparées par un espace, et suivies, pour la dernière seulement, du caractère "return".

De plus le Pascal utilise toutes sortes de signes séparateurs comme le point-virgule (;) séparant deux instructions ou deux déclarations, le : signalant l'appartenance à un type, le

"begin" et "end" jouant le rôle de parenthèses, et le . (point final du programme), la virgule étant le séparateur de données.

Les accolades { } ont pour mission d'encadrer toutes sortes de commentaires jugés bons par le programmeur, pour la compréhension du programme. Tout ce qui est donc encadré par des accolades sera ignoré lors de la compilation du programme.

Ce programme ne comportait que trois instructions au premier niveau; même si la dernière était composée de deux autres sélections. Cette dernière instruction n'est pas suivie du point-virgule car le "end" qui suit n'est pas une instruction, mais un séparateur signalant à l'analyseur syntaxique la fin d'un bloc ayant dû débuter par un "begin".

Remarque sur la disposition (l'indentation) des textes en Pascal

Tous les exemples présentés dans ce livre suivent la règle consistant à écrire sur une même "verticale" des instructions qui se suivent (à la rigueur sur une même ligne, si elles forment un bloc relativement court en nombre de caractères, nous éviterons l'usage trop répandu consistant à écrire un mot par ligne, ce qui allonge démesurément les programmes).
Lorsqu'une instruction est composée, ses "composantes" sont placées sur une autre marge, décalée vers la droite comme on l'a déjà fait remarquer ci-dessus pour les tests emboîtés.
On verra ainsi par exemple des instructions "for" ... décrivant toute une suite d'actions à effectuer plusieurs fois, ces actions étant écrites sous le "for" mais décalées, afin de respecter une recommandation de lisibilité, et encadrées par "begin" et "end" en Pascal ou par des accolades en C, afin de respecter une obligation liée à la logique du programme.
D'une manière plus générale, si une instruction est composée d'autres instructions, on écrira ces dernières plus à droite que celle qui les "contient". On pratique une "indentation" échelonnée.

Exemple de sélections : le calendrier

Une autre instruction, en attendant une écriture beaucoup plus puissante dans la spécification des cas avec la "cond" du Lisp, permet de sélectionner différentes actions suivant les valeurs d'une variable, c'est le "case ... of ... : ... end;" qu'il faut traduire par "suivant les valeurs de ... parmi ... faire ...".
L'exemple du calendrier met en oeuvre cette instruction, il s'agit de trouver le jour de la semaine pour une date du calendrier grégorien.
La formule donnée dans le programme utilise les fonctions "div" (division entière), ou bien "trunc" (partie entière) et "mod" (reste de la division entière 23 mod 7 = 2).
Cette formule n'est valable qu'à la condition que janvier et février soient comptés comme mois de n° 13 et 14 de l'année précédente. Les termes de la formule faisant intervenir l'année s'expliquent par le fait qu'il y a une année bissextile tous les quatre ans sauf trois siècles exacts sur quatre (1900 ne l'était pas, 2000 le sera). D'autre part, elle n'est valable qu'à partir du vendredi 15 octobre 1582, enfin il faut supprimer les trois derniers termes pour les dates du calendrier julien, toujours en vigueur dans l'église orthodoxe (adopté en 1752 en Gd Bretagne et en 1918 en Russie).

```
program calendrier ;
        var A, M, D, N : integer ; { Représenteront l'année, le mois, le jour, et un numéro de code du jour de la
semaine tel que zéro soit le samedi, ... et six le vendredi. }
begin { Readln, comme Writeln, commandent un passage à la ligne après leur exécution }
write ('Quelle année ? '); readln (A); write ('Numéro du mois ? '); readln (M); write ('Date ? '); readln (D);
if M < 3 then begin  M:= M + 12 ; A := A - 1 end ;
N := (D +2*M + (3*(M+1) div 5)  + (5*A div 4) - (A div 100) + (A div 400) + 2)   mod  7;
        case N of 0     : write ('samedi');
                  1     : write ('dimanche');
                  2     : write ('lundi');
                  3     : write ('mardi');
                  4     : write ('mercredi');
                  5     : write ('jeudi');
                  6     : write ('vendredi') end end.
```

Remarques

Signalons tout de suite une erreur classique et pourtant grave, dans une instruction telle que "if M = 1 then A := A-1", les signes = et := n'ont rien de commun. Le premier représente véritablement l'égalité, c'est-à-dire que la proposition M = 1 est vraie ou fausse, M = 1 possède une valeur booléenne que l'on teste, et c'est tout ce qui est fait.
Le signe := est celui de l'affectation, (souvent écrit grâce à ← qui est plus explicite en pseudo-pascal ou en LSE, attention, l'affectation est au contraire = en C, alors que l'égalité dans ce langage est notée ==) A := A-1 est un ordre donné pour modifier la valeur numérique de A, ici une décrémentation.
L'instruction "if M := 1 then A = A-1" n'a donc aucun sens, et ceci pour deux raisons, après le "if", c'est une valeur de vérité qui est évaluée afin de décider où sera continuée l'exécution, après le "then", c'est une instruction qui est commandée.
L'instruction "case" peut s'utiliser sous la forme "case N of -2 .. 5 : " suivi d'une instruction, pour déterminer une action à faire dans le cas où N serait un entier compris dans l'intervalle de -2 à 5. Les deux points alignés .. étant la marque d'un intervalle en Pascal. (chapitre 6)

Les boucles

Trois instructions Pascal permettent de décrire un travail itératif :
"for X := ... to (ou bien "downto" pour une décrémentation de X)... do ..." peut s'utiliser chaque fois que l'on connaît le nombre de "boucles" à effectuer.
La variable dont on choisit le nom (ici X) peut servir à compter les boucles lorsque X va par exemple de 1 à 12 comme ci-dessous. Mais sa valeur initiale peut être toute autre que 1, et peut être donnée par toute expression calculable. Le "to" sous-entend une incrémentation automatique de cette variable à chaque nouveau passage.
Dans le cas où l'arrêt de l'itération n'est déterminé que par une condition sans que l'on puisse savoir le nombre de boucles , on pourra utiliser "while < condition > do < instruction >." ou de manière analogue "repeat < instruction > until < condition>".
Les trois formes suivantes sont donc équivalentes :

a) `for X := 1 to 12 do writeln ( '13 fois ' , X , ' égal ' , 13*X );`
On doit lire: pour X allant de 1 à 12 faire...

b) `X := 1 ; repeat writeln ( '13 fois ' , X , ' égal ' , 13*X ); X := X+1 until X = 13`
(Répéter l'affichage de ... et l'incrémentation de X jusqu'à ce que X soit égal à 13)

c) `X := 1 ; while X < 13 do begin writeln ( '13 fois ' , X , ' égal ' , 13*X ); X := X+1 end ;`
(Tant que X est strictement inférieur à 13 faire ...)
"writeln" signifiant une édition à l'écran suivie d'un passage à la ligne.

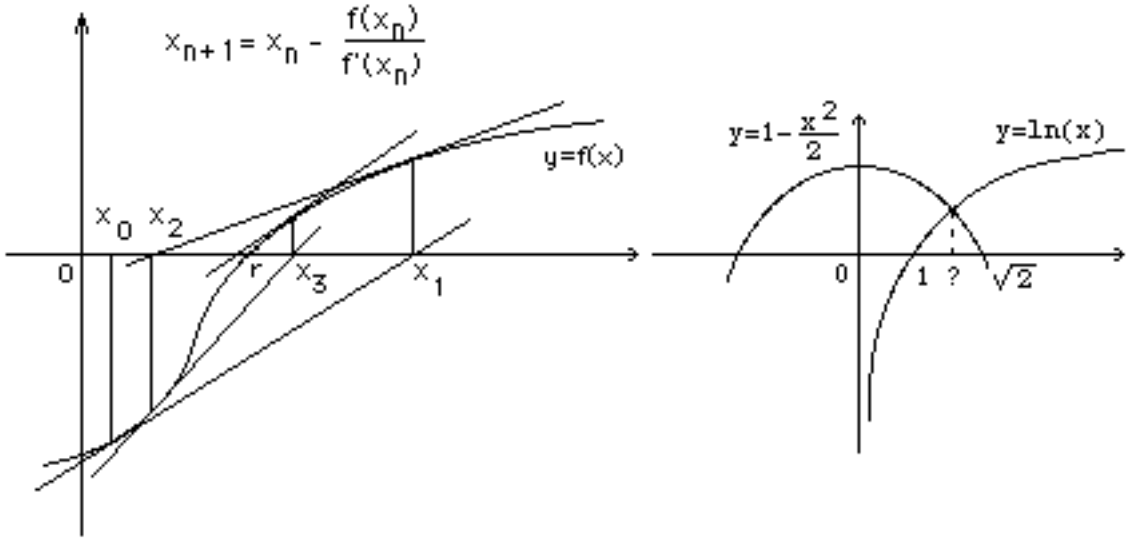
Exemple de successions itérées de réaffectations : suite de Fibonacci

C'est la suite telle que chaque terme est égal à la somme des deux termes précédents, en partant de 1. Les premiers termes sont donc 1 1 2 3 5 8 13 21 ... (Cette suite est déterminée par : $u_0 = u_1 = 1$ puis $u_n = u_{n-1} + u_{n-2}$ elle est presque géométrique pour les termes de rang élevés). Pour ne pas employer trop de variables, (il serait évidemment maladroit de conserver toutes les valeurs successives dans un tableau), on peut se servir simplement de trois variables P Q R représentant toujours trois termes consécutifs.

```
program fibonacci ; { dérouler le programme à la main pour P < 20}
var P, Q, R : integer ;
begin P := 1; Q := 1; { P, Q, R désignent toujours 3 termes consécutifs de la suite }
                  while   P  <  1000        do begin write (P,' / ');
                                              R := P + Q;       { le troisième terme est la somme des 2 précédents }
                                              P := Q;           { on avance d'un cran ; le premier est l'ancien second }
                                              Q := R            { et le nouveau second est l'ancien troisième }
                                              end
end.
```

Exemple de programme itératif, la méthode de Newton

Le programme suivant est celui, très classique, de la résolution d'une équation dont on connaît l'existence d'une solution. Si x_0 est une valeur voisine de la racine r, la tangente en x_0 coupe l'axe des abscisses en x_1, la tangente en x_1 coupe à nouveau l'axe en x_2 , ... en réitérant on obtient une suite x_n généralement convergente vers r. Naturellement cette convergence pose un problème mathématique, sans compter celui de l'existence même de la suite, si pour un x_n, la courbe passe par un sommet, la tangente y est parallèle à l'axe et ne le coupera donc pas. La méthode de dichotomie vue plus loin, est alors plus appropriée.
On calcule aisément que $x_{n+1} = x_n - f(x_n) / f'(x_n)$



L'intersection de la parabole $y = 1 - x^2/2$ avec $y = \ln(x)$ conduit par exemple à une équation : 1 - x*x/2 - ln(x) = 0 dont on montre facilement qu'elle admet une solution unique entre 1 et 1,4 sans pouvoir la calculer exactement.

```
program newton ;

function fonc (X : real ): real ;   { la fonction quel 'on veut annuler c'est à dire la différence des deux }
        begin fonc :=1 - X*X / 2 - ln(X)  end ; { c'est ainsi que l'on déclare une fonction en Pascal }

function der (X : real ) : real ;    { sa dérivée }
        begin der := -X - 1 / X  end ;

function newton (DEP, EPS : real ) : real ;
{ DEP et EPS sont respectivement le point de départ et la précision }
        var F, X : real ;
        begin
        X := DEP;
        repeat F := fonc (X) ; X := X - F / der(X) until abs (F) < EPS;
        newton := X
        end;

begin write (newton (1, 0.000001)) end.
```

Ici les choses sont un peu mieux structurées, le programme proprement dit est réduit à la dernière ligne ce qui est souvent le cas. On remarquera bien sûr que la fonction et sa dérivée ne peuvent constituer des arguments de la "fonctionnelle" newton comme cela serait cohérent, en Pascal, c'est en fait possible en C++ et en Lisp comme nous le verrons.

Exemples de modifications d'affectations : tri de valeurs méthode bulle

Le principe du tri-bulle est de parcourir une liste de nombres en comparant chaque terme avec son suivant, s'ils ne sont pas dans l'ordre croissant, on les intervertit, et si au moins une telle permutation a eu lieu, le parcours étant fini, alors on recommence.
On anticipe légèrement sur les chapitres suivants en créant un nouveau type de données "tab" comme tableau de 10 entiers.

```
program tribul ;
        const N = 10;  { N sera une valeur fixe pour tout le programme.}
        type tab = array [1.. N] of integer ;
        { On définit ainsi un nouveau type de données par un tableau indexé de 1 à N }
        var J : integer ; L : tab ;

procedure echange (var A, B : integer );
        var X : integer ;
        begin X := A ;  A := B ; B :=  X  end;

function balai (var L : tab ; N : integer ) : boolean ; { Est un exemple de fonction manipulant des types de
données tout à fait différents, balai joue le rôle de signal ou "drapeau"., il vaut "vrai" si L est entièrement trié}
        var I : integer, drap : boolean ;
        begin drap := true;
        for I := 1 to N - 1 do if L[I+1] < L[I] then begin echange (L[I], L[I+1]); drap := false end
        balai := drap
        end ;

begin { début des trois instructions du programme }
for J := 1 to N do read ( L[J] ) ;              {lecture des données au clavier }
repeat until balai (L) = true ;     {boucle où rien ne se passe car c'est l'évaluation de "balai" qui modifie L}
for J := 1 to N do write ( L[J] ,' / ' )        {écriture du résultat} end.
```

La solution ci-dessus propose à l'avant-dernière ligne, une boucle qui s'exécute jusqu'à ce qu'une fonction ait une certaine valeur "vrai", rien ne se passe en apparence dans la boucle. En fait, à chaque appel de la fonction, l'argument L est modifié. Ces questions seront vues en détail au chapitre 4. On peut préférer une solution plus classique ne faisant pas appel à cette fonction, mais à deux variables supplémentaires I de type entier et "drap" de type booléen. En ce cas le programme utilisera comme seconde instruction :

```
repeat   drap := true;
        for I  := 1 to N - 1 do if L[I+1] < L[I] then begin echange (L[I], L[I+1]); drap := false end
until drap
```

Remarque : "balai" uniquement peut remplacer "balai = true" puisque cela désigne une variable de type booléenne c'est à dire que sa valeur est nécessairement "true" ou "false".

2-1° Simplifier en détaillant les calculs, l'expression booléenne suivante dans laquelle A et B sont des réels: non [($A \leq 0$ ou non $B < 0$) et ($A < 0 \rightarrow B \leq 0$)]

Avec la définition élémentaire de ($P \rightarrow Q$) = ($\neg P$ ou Q), et grâce aux lois de Morgan (annexe), non (P ou Q) = (non P) et (non Q) ainsi qu'à l'involutivité $\neg\neg P = P$, on obtient rapidement que la condition signifie A, B de même signe.

2-2° Quelle est la différence entre les deux actions de ?

```
        if M < 3 then begin M:= M+12 ; A := A-1 end;
        if M < 3 then  M:= M+12 ; A := A-1;
```

2-3° Quelle erreur y a t-il dans l'écriture suivante ?

```
        if M = 1 then  M:= 13 ; else if M = 2 then M := 14 ;
```

2-4° Quel est le résultat de la séquence d'affectations X := X + Y ; Y := X - Y ; Y := X - Y ?

Un échange des deux (donc sans passer par un tampon).

2-5° Un écran est repéré par un nombre de lignes H+1 et un nombre de colonnes L+1, les lignes sont numérotées de 0 à H de haut en bas, et les colonnes de 0 à L, de gauche à droite. Donner les fonctions affines "col" et "lgn" définies respectivement sur les intervalles [A, B] et [C, D] pour les variables respectives x et y, de façon à obtenir col(A) = lgn(D) = 0, col(B) = L, lgn(C) = H.

Réponses : col(x) = L(x - A)/(B - A) lgn(y) = H(D - y)/(D - C)

2-6° **Equation du second degré.** Ecrire en disposant correctement les emboîtements de conditions, le programme résolvant l'équation du second degré $ax^2+bx+c=0$, dans tous les cas de nullité de a, b, c et du discriminant D

2-7° **Date du lendemain.** Faire un programme capable de donner la date du lendemain lorsqu'on lui fournit le numéro du jour, le mois et l'année. Il est assez difficile de bien disposer tous les cas, même si on ne tient pas compte des années bissextiles (le jour de l'an a été fixé au premier janvier en 1564 par Charles IX).

2-8° **Calcul du poids idéal** (kg) en fonction de la taille (cm) et de l'âge (ans) par les formules: (3*T-250) * (A + 270) / 1200 pour les hommes et (T / 2 - 30) * (180 + A) / 200 (femmes).

```
program poids;
        var A, T, PR : integer; S : char; PI : real;
begin   write ('Donnez votre âge en années '); readln (A);
        write ('Quel est votre taille en cm ? '); readln (T);
        write ('Votre poids en kg ? '); readln (PR);
        write ('Votre sexe (M ou F) ? '); readln (S);
        if   S = 'M'        then PI := (3*T-250) / 1200 * (A + 270)
                            else  PI := (T/2 - 30) / 200 * (180 + A);
        PI := round (10*PI)) / 10 ; {astuce pour ne garder qu'une décimale, "round" étant l'entier le plus proche }
        D := PR - PI; { D est la différence entre le poids réel et le poids idéal }
        if    abs(D) < 3  then write ('Ca va')
                                  else if D < 0      then write ('Vous pouvez prendre ', D, ' kg')
                                                     else write ('Vous devez maigrir de ', D, ' kg')
end.
```

2-9° Créér un tableau de correspondances de poids et tailles, se servir de l'instruction gotoxy (C, L) localisant le curseur dans l'écran à la ligne L (1 < L < 24) et colonne C (1 < C < 80).

2-10° Faire un tableau des valeurs de P, Q, R dans Fibonacci pour les cinq premiers passages.

2-11° Soit le programme dont le corps est :
P := -2; Q := 3; for X := 1 to 5 do begin write (Q,' /'); R := 2*Q + P; P := Q; Q := R end.
Faire un tableau des valeurs successives de P, Q, R.
a) Que voit-on exactement à l'écran ?
b) ..... si les deux dernières instructions d'affectation sont interverties ?

P -2 3 4 5 26
Q 3 4 11 26 63
R 4 11 26 63 152
L'effet est 3/4/11/26/63/ et dans l'autre cas P = Q, c'est 3/4/12/36/108/

2-12° **Ecrire un programme déterminant la date P de Pâques** en fonction de l'année A en suivant la formule de Gauss-Delambre : B = (19 (A mod 19)+M) mod 30 et P = [2 (A mod 4) + 4 (A mod 7) + 6B+ N] mod 7 + B + 22 du mois de mars si P ≤ 31, et d'avril sinon. M et N étant lentement variables dans le calendrier grégorien, actuellement égaux à 24 et 5. On rappelle que X mod Y est le reste de la division entière de X par Y (exemple 47 mod 6 = 5)

```
program paques ;
        var A, B, P : integer;
        begin write ('Donnez l''année '); readln (A);
        B := ((19* (A mod 19)+24) mod 30);
        P := ((2*(A mod 4)+ 4*(A mod 7) + 6*B + 5) mod 7) + B+ 22;
        if  P  <  32        then writeln ('Dimanche ', P,' mars')
                            else writeln ('Dimanche ', P-31,' avril')
end.
```

2-13° **Jour de la semaine.** Reprogrammer le jour de la semaine correspondant à une date, suivant la formule de Zeller. Cette fois on indique que si le mois m est supérieur ou égal à 3 on le change en m-2 et sinon en m+10 ainsi que l'année a en a-1. On pose alors na le numéro de l'année dans le siècle et s le numéro du siècle, et enfin f = j + na - 2*s + na div 4 + s div 4 + (26*m - 2) div 10. Le code est donné par f mod 7 (0 si dimanche).
Grégoire XIII a imposé sa réforme du calendrier le jeudi 4 octobre 1582 qui fut donc suivi du vendredi 15 octobre 1582. Avant ce changement, le programme doit également fonctionner en ajoutant 3 à f.

On utilise le réel R = a + m/100 + j/10000 que l'on compare avec 1582,1004 et 1582,1015
Tester grâce à jeudi 24 octobre 1929, samedi 1-1-2000, mardi 14-7-1789 ...
En pseudo-pascal :

lire (j, m, a)
si m > 3 alors m ← m-2 sinon soient m, a ← m+10, a-1
soient s, na ← a div 100, a mod 100
f ← j + na - 2s + (na div 4) + (s div 4) + (26m - 2) div 10
si a + m /10 + j / 10000 < 1582,1004 alors f ← f + 3
jour ← f mod 7
conditions jour = 0 alors dimanche ... etc.

2-14° Calculer la somme alternée des inverses des entiers impairs en débutant par 1 et en finissant à -1/199, puis multiplier par 4 le résultat et l'afficher.

2-15° Ecrire trois versions différentes d'un programme délivrant la suite des entiers jusqu'à 12, de leurs carrés, cubes, et puissance de 2. On utilisera writeln (X : 3, X*X : 5, X*X*X : 5, exp (X * ln(2)) : 10), "write (X : m : n)" a pour effet d'éditer X sur m caractères en tout avec n chiffres après la virgule s'il s'agit d'un réel.

2-16° **Conversion de degrés Farenheit et Celsius,** sachant que F = 9*C/5 + 32, faire une table de conversion.

2-17° Afficher les valeurs de $n^2 + n + 41$ pour $0 \leq n \leq 39$. Qu'observe-t-on ?

2-18° **Impôt sur le revenu.** Programmer le calcul de l'impôt I sur les revenus, sachant que si on note S le total annuel des salaires, F celui des revenus fonciers, C l'ensemble des charges à déduire alors le revenu imposable R est 80% des 90% de S, auxquels on ajoute F et on retranche C. Si N est le nombre de parts et QF = R / N, alors utiliser l'instruction "case" et les explications fournies sur les déclarations annuelles pour calculer I (les fameuses tranches dont les caractéristiques sont modifiées chaque année). Représenter I en fonction de R.

2-19° Calculer la somme des inverses des puissances 4 des entiers impairs de 1 à 199, multiplier par 96, extraire la racine quatrième, et afficher le résultat.

2-20° Pgcd et du ppcm de deux entiers

La méthode la plus simple est de chercher le premier multiple de a qui soit divisible par b, sachant que le produit du pgcd par le ppcm est ab, on a les deux en même temps.

```
program pgcd;
        var a, b, k : integer;
        begin readln (a, b);
        k := 0;
        repeat k := k+1 until a*k modulo b = 0
        writeln ('ppcm de ', a, ' et ', b, ' = ', a*k, ' pgcd = ', b div k)
        end.
```

2-21° **Constante de Vijayaraghavan**, c'est la racine de l'équation $x^3 = x+1$, la déterminer par la méthode de Newton.

2-22° **Loi de Bode** : exprimer (en unités astronomiques) les distances des planètes au soleil, en suivant la loi empirique de Bode $(4 + 3*2^{n-2}) / 10$ pour n = 1 à 10, (les astéroïdes constituant la cinquième planète).

2-23° **Combinaisons C_n^p**, faire le programme en veillant à alterner les opérations de façon à limiter l'erreur de calcul et à éviter les dépassements de capacité. (Le résultat provisoire en cours de calcul doit demeurer entier).

L'astuce consiste à alterner les multiplications et les divisions, en commençant par la multiplication de deux entiers consécutifs. En effet, parmi eux il y en a necéssairement un qui est pair, donc on peut immédiatement après diviser par 2, puis en multipliant par l'entier suivant, des trois facteurs, là aussi, un des trois est multiple de 3, on peut donc aussitôt après diviser par 3. En débutant par un premier facteur égal à n-p+1, et un premier diviseur égal à 1, et en continuant ainsi, le dernier facteur sera n et le dernier diviseur sera p. Cette solution permet de repousser le plus loin possible l'instant où la capacité en nombre entier sera atteinte.

```
program comb;
var n, p, i, c : integer;
begin   write ('Donnez deux entiers n et p ');
        readln(n, p);
        c := 1;
        for i := 1 to p do c := (c*(n-i+p)) div p
        write ('Le résultat est ', c)
end.
```

2-24° Le tri de Shuttle est le suivant :

```
for i := 1 to n-1 do  begin j := i;
                      test := (t[j] > t[j+1]);
                      while test do begin ech(t[j], t[j+1])
                                    j := j-1;
                                    if    j = 0      then test := false
                                                     else test := (t[j] > t[j+1])
                                    end;
                      end;
```

Faire tourner le programme à la main pour t = [4, 7, 22, 3, 9, 11, 0, 32, 5].
Analyser ce que fait le programme.

2-25° Tri par extraction. Le principe du tri par extraction d'un tableau t est d'aller chercher le plus petit élément pour le mettre en premier, puis de repartir du second... En utilisant une procedure "echange", programmer ce tri, puis l'étendre à une extraction dans les deux sens.

Si t est indicé de 1 à n, l'essentiel du programme sera :

```
for k := 1 to n-1 do begin for i := k+1 to n do if t[i] < t[k] then echange (t[i], t[k]); write(t[k]) end;
write (t[n])
```

Si on extraie le plus petit, on peut également mettre le plus grand à la fin :

```
for k := 1 to n div 2  do          begin for i := k+1 to n-k+1 do
                                          begin if t[i] < t[k] then echange (t[i], t[k]);
                                          if t[i] > t[n-k+1] then echange (t[i], t[n-k+1])
                                          end;
                                   write (t[k]) end;
for i := n div 2 + 1 to n do write(t[i])          {Ce tri est assez lent, mais meilleur que le tri-bulle}
```

2-26° Tri bulle boustrophédon : on balaye de gauche à droite suivant le principe du tri bulle, puis on revient de droite à gauche et ainsi de suite.

L'intérêt de la méthode est qu'à chaque parcours, la valeur maximale est repoussée à sa position définitive si c'est un parcours de gauche à droite, et c'est la valeur minimale qui est repoussée au début sinon. Ceci permet de restreindre d'un cran l'étendue de chaque balayage.

```
for i := 1 to n div 2 do  begin drap := false;
                          for k := i to n-i do if t[k] > t[k+1] then   begin drap := true;
                                                                      echange (t[k], t[k+1]) end;
                          if drap then      begin for k := n-i-1 downto i do
                                                if t[k] > t[k+1]  then    begin drap := true;
                                                                          echange (t[k], t[k+1]) end;
                                            end
                                    else affiche(t) { affiche est une procédure que l'on mettra au point }
                          end;
```

Remarque, on peut améliorer encore en signalant les indices des derniers permutés, au chapitre 4, nous donnons une solution mutuellement récursive de ce problème.

2-27° Tri Shell : consiste à trier séparément des sous-suites de la table, formées par les éléments répartis de h en h. Empiriquement, Shell propose la suite d'incréments vérifiant $h_1 = 1$, $h_{n+1} = 3h_n + 1$ en réalisant les tris du plus grand incrément possible vers le plus petit.

```
type tab = array [1..20] of integer; {on demandera shell (t, n) pour un tableau t déjà entré de n éléments}

procedure shell (var t : tab; n : integer);
        var h, i, j, k, x : integer;
        begin h := 1 ; while h < n div 9 do h := 3*h + 1;
        repeat for k := 1 to h do   begin i := h + k;
                                    while i <= n do   begin x := t[i]; j := i-h;
                                                      while (j >= 1) and (t[j] > x) do      begin
                                                                                            t[j+h] := t[j];
                                                                                            j := j-h;
                                                                                            t[j+h] := x end;
                                                      i := i+h end;
                                    end;
                h := h div 3
        until h = 0
        end;
```

2-28° Tri-arbre : on transforme un tableau en un arbre binaire, "tas" de telle sorte que l'élément i ait deux fils de rang 2i et 2i + 1 plus petits que lui. (Un tel arbre est dit "maximier"). Cette construction se faisant à reculons, le but est en même temps de modifier le tableau pour que les deux fils soient inférieurs à leur père. Puis grâce à une transformation également en arrière, on va chercher à modifier le tableau de gauche à droite et renvoyer le plus grand élément entre à la fin. Faire tourner à la main l'algorithme pour une petite table en dessinant l'arbre.

Solution : {Ce tri est aussi appelé tri par tas}

```
type tab = array [1..20] of integer; {On demandera tri (t, n) pour un tableau entré comportant n éléments.}

procedure echange (var x, y : integer);
        var z : integer; begin z := x; x := y; y := z end;
procedure vue (var t : tab; n : integer);
        var i : integer; begin for i := 1 to n do write (t[i] : 3); writeln end;

procedure maximier (var t : tab; i, j : integer);{les sous-arbres fils de t[i] à t[j] seront maximiers si leurs sous
arbres le sont déjà, ce qui signifie simplement que t[s] est supérieurs à ses deux fils t[2s] et t[2s+1]}
        var s, f : integer; {s représente un sommet, f son fils}
        begin s := i;
        while 2*s <= j do
                begin f := 2*s;{on échange le sommet avec le plus grand des deux fils}
                if (f + 1 <= j) and (t[f+1] > t[f]) then f := f+1;
                if t[f] > t[s] then echange (t[s], t[f]); s := f
                end
        end;
procedure tri (var t : tab; n : integer); {t étant maximier, on ordonne à reculons }
        var i : integer;{on rend d'abord maximier à reculons}
        begin for i := n div 2 downto 1 do maximier (t, i, n);
        for i := n downto 2 do      begin  maximier (t, 1, i) ; echange (t[1], t[i]) end
                                    {t est maximier de 1 à i et ordonné croissant de i+1 à n}
       end;
```

2-29° Etudier la complexité du tri par arbre en supposant que le nombre n d'éléments est de la forme 2^m, on montrera que la construction de l'arbre est en O(n), puis que le tri proprement dit est en O(n.log(n)).

2-30° Recherche des solutions entières de l'équation $x^2-y^2 = a$, en cherchant les facteurs (x+y) et (x-y) de même parité, a étant une constante fixée. Exemple pour $x^2-y^2 = 45$; si x-y vaut successivement 1, 3, 5, et x+y, respectivement 45, 15, 9, alors x admet pour valeurs 23, 9, 7, avec respectivement y, 22, 6, 2.

Si x - y = p , x + y = q, pour que x, y ∈ N, il faut absolument que p et q aient la même parité, c'est à dire que p + q soit pair, ce qui est équivalent. D'autre part, pour éviter les répétitions on se limite aux couples (p, q) de diviseurs de a tels que $1 \le p \le \sqrt{a} \le q \le a$

```
program eqentiere;
        var a, p, q : integer;
procedure reso (p, q : integer) ; {Résoud x - y = p , x + y = q }
        var x, y : integer;
        begin x := (p+q) div 2 ; y := (q-p) div 2 ; writeln ('Le couple ',x,' , ',y,' est solution.') end;
function divise (m, n:integer) : boolean; {Indique si m divise n ou non}
        begin divise := ((n mod m) = 0) end;
        {c'est l'expression booléenne elle-même qui est la valeur de "divise"}

begin write ('Solutions entières de x2-y2 = a, donnez la valeur de a '); readln (a);
for p := 1 to round ( sqrt (a) ) do {notons que "round" est l'entier le plus proche, et que "odd" est "impair"}
                if divise (p, a) then  begin q := a div p; if odd (p) = odd (q) then reso (p, q) end
end.
```

2-31° Fonction logarithme néperien, on veut la réaliser sans avoir recours à celle qui est déjà implantée. Pour cela, montrer que si x > 0 alors x s'écrit de façon unique sous la forme $2^k y\sqrt{2}$ avec k ∈ Z et y ∈ [1/√2, √2[, pour cet y, montrer en posant u = (y-1)/(y+1) que l'on a $\ln(y) = \Sigma\ 2u^{2i+1}/(2i+1)$, série pour i=0 jusqu'à l'infini. Ecrire la procédure "decompose" qui va valculer ces k et y, pour un x > 0, puis la function "serie" qui va donner la somme partielle pour i < 7 et y ∈ [1/√2, √2[, et enfin la function "lnapprox" qui pour x > 0, donne une valeur approchée de ln(x).

Pour tout x positif, $x= 2^k z$ avec k unique dans Z, on pose $z = y\sqrt{2}$ d'où y ∈ [1/√2, √2[
u = (y-1) / (y+1) entraîne y = (1+u) / (1-u) et on vérifie que -1 < u < 1 donc ln(y) = ln(1+u) - ln(1-u) = $\sum 2u^{2i+1}/(2i+1)$ pour i entier.

```
procedure decompose (x : real ; var k : integer; var y : real);
        var p : real; {p représente 2 puissance k}
        begin k := 0; p := 1; {voir les cas de x = 1 ou 2}
        if x >= 2       then repeat p := 2*p; k := k+1 until (x < 2*p)
                        else if x < 1 then repeat p := p/2; k := k-1 until (p <= x);
        y := x / (p*sqrt(2)) end;

function serie (y : real) : real;
        var u, v, s : real; i : integer; {s est la somme partielle, v est u^2i}
        begin u := (y-1) / (y+1); s := 1; v := u*u;
        for i := 1 to 6 do begin s := s + v / (2*i+1); v := v*v end;
        serie := 2*u*s end;

function lnapprox (x : real) : real;
        var k : integer; y : real;
        begin decompose ( x, k, y); lnapprox := (k + 1/2)*0.69 + serie (y) end;
```

2-32° Nombres premiers : calculer tous les nombres premiers jusqu'à une certaine constante m = 400 par exemple, en testant l'éventuelle divisibilité. On impose un exercice de style consistant à ne pas utiliser "for", "repeat" et "while", mais "goto". L'utilisation se fait grâce à "goto étiquette" où les "étiquettes" sont des identificateurs choisis par le programmeur et déclarés par "label" e1, e2 ..., elles sont alors placées aux endroits voulus et suivies de deux points.

Solution, pour chaque entier impair n à partir de 3 ou de 5, on teste sa divisibilité par p allant de 2 en 2 depuis 3 jusqu'à la racine carrée de n :

```
program premiers ;
const m = 400 ;
label e1, e2, e3 ;                  { sont les trois labels choisis }
var n, p : integer ;                { n est l'entier courant que l'on teste, p, son diviseur éventuel }
begin write (2 : 4, 3 : 4) ;        {on traite à part les cas initiaux }
        n := 5 ;                    { on commence à reconnaître si 5 est premier }
        e1 : p := 3 ;
        e2 : if n mod p = 0 then goto e3 ;          { n est divisible, on passe au suivant }
        p := p + 2 ;
        if p > sqrt (n)     then write (n : 4)      { n vient d'être reconnu premier }
                            else goto e2 ;
        e3 : n := n + 2 ;           { à partir de 3, tous les nobres premiers sont impairs }
        if n <= m then goto e1
end.
```

```
  2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71
 73  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397
```

2-33° **Le nombre caché** : l'ordinateur choisit un nombre au hasard entre 1 et 1000 , il faut le deviner en dix questions au plus, l'ordinateur ne répondant que par "trop petit" ou "trop grand" ou "gagné". On se sert de X := random (1000) et on peut bien sur écrire une condition logique composée avec "and" ou "or".

Cet exemple montre le cas très fréquent d'une boucle dont on peut sortir de deux façons différentes. Le test de réitération comporte donc deux cas dont un est nécessairement répété ensuite.

```
program jeu;
        var X, R, Q : integer;
begin
X := random (1000); { en turbo-pascal sur PC, sur Mac, "random" seul, renvoie un réel entre 0 et 1 }
Q := 1; { numéro de la question }
repeat  write ('Question numéro ', Q, ' quel est ce nombre ? ');
        readln(R); {R est la réponse du joueur }
        if R < X        then writeln (' trop petit !')
                        else if R > X   then writeln (' trop grand !')
                                        else writeln (' Gagné !');
        Q := Q + 1
until (Q > 10) or (R = X);
if R = X then writeln (' Perdu c"était ', X)
end.
```

2-34° Refaire le même programme que précédemment, mais en utilisant l'instruction de débranchement "goto" endroit; dans laquelle "endroit" est une position ultérieure du programme qui aura été déclarée comme un "label" au début du programme.

2-35° **Deux personnes jouent avec un tas d'allumettes** en prenant à tour de rôle de 1 à 3 allumettes sans les remettre. Le perdant est celui qui tire la dernière allumette. Simuler ce jeu.

2-36° **Etudier la convergence** de la suite $x_0 = 11/2$, x1 = 61/11, et $x_{n+2} = 111 - 1130/x_{n+1} + 3000/(x_n x_{n+1})$, les limites possibles sont 5, 6 et 100.

2-37° **Calculer le produit a*b de la manière suivante** : on divise a par 2 tant que c'est possible, en doublant b, sinon on décrémente a et on ajoute b au résultat.

Il s'agit en fait de toujours prendre le quotient entier de a par 2 dans une colonne en doublant b dans une seconde colonne, puis le résultat sera la somme des nombres de la seconde colonne situés en regard d'un nombre impair dans la première. Exemple (en ligne) :

26	13	6	3	1	0	
54	108	216	432	864		sont additionnés 108 + 432 + 864 = 1404

```
program mult ;
var a, b, r : integer;
begin
write ('Entrez deux entiers '); readln (a, b);
r := 0; {a*b + r est dit un invariant de boucle }
while a <> 0 do if odd(a)       then begin a := a-1; r := r + b end
                                else begin a := a div 2; b := 2*v end;
write ('Le résultat est ', b)
end.
```

2-38° Anniversaire Quelle est la probabilité p(n) qu'au moins deux personnes dans un groupe de n personnes, aient leur anniversaire le même jour ? Ne pas tenir compte des années bissextiles, et ne pas utiliser la formule de Strirling $n! \sim n^n e^{-n}\sqrt{(2\pi n)}$

En considérant l'évenement contraire "chaque personne du groupe a une date d'anniversaire qui lui est spécifique", la probabilité de cet évenement est le nombre d'injections de n personnes vers 365 jours, alors que le nombre de cas possibles est le nombre d'applications quelconques de n personnes vers 365 jours.

$$p_n = 1 - \frac{A_{365}^n}{365^n} = 1 - \frac{365*364* \ldots (365 - n + 1)}{365^n}$$

```
program anniversaire;
function proba (n : integer) : real;
        var i : integer; r : real;
        begin r := 1; for i := 0 to n - 1 do r := r * (365 - i) / 365;
        proba := 1 - r end;

var m : integer;
begin write ('Donnez le nombre de personnes dans le groupe '); readln (m);
write ('La proba que deux d''entre eux au moins soient nés le même jour est ', proba (m) : 5 : 3);
end.
```

2-39° Fonction Gamma, faire un programme de calcul de la fonction Γ par la formule :

$\Gamma(x) = \lim n!\, n^x / [\, x(x+1)(x+2) \ldots (x+n)\,]$ pour x positif et $n \to +\infty$

2-40° Calcul d'une intégrale par une méthode de Monte-Carlo: si sur [a,b] on a toujours $0 \le f(x) \le M$, on tire un grand nombre de fois N, un nombre x au hasard dans l'intervalle [a,b], et un nombre y dans l'intervalle [0,M]. Si $y \le f(x)$ a été obtenu n fois, alors l'intégrale est approchée par nM(b-a) / N.

2-41° Intégrale par Simpson. Ecrire une procédure de calcul d'intégrale pour une fonction f préalablement déclarée, entre deux valeurs A et B, en utilisant la formule d'approximation de Simpson où le pas est h = (b - a) / 2n :

$$I = \frac{h}{3}\left[f(a) + f(b) + 2*\sum_{p=1}^{n-1} f(x_{2p}) + 4*\sum_{p=0}^{n-1} f(x_{2p+1})\right]$$

Solution, on a choisit la fonction dérivée de Arctan, de façon à retrouver π/4 en l'intégrant entre 0 et 1 :

```
program simpson;
function f (x : real) : real;
        begin f := 4 / (1 + x*x) end;
function simpson (a, b : real;  n : integer) : real;
        var p : integer; s, h : real;
        begin h := (b - a) / 2 / n; s := f(a) + f(b) + 4*f(a + h);
        for p := 1 to n - 1 do s := s + 2*f(a + 2*p*h) + 4*f(a + (2*p + 1)*h);
        simpson := h*s / 3 end;
begin write ('vérification ', simpson (0, 1, 500) :  5 : 3) end.
```

2-42° Recalculer Γ(x) comme l'intégrale impropre pour t de 0 à $+\infty$ de la fonction $e^{-t}\, t^{x-1}$

2-43° Méthode de la plus grande pente de Cauchy. On cherche le minimum d'une fonction réelle de plusieurs variables à partir d'un point M_0 arbitraire. Si la fonction admet des dérivées partielles, son gradient (qui mesure la plus grande pente en ce point) est le vecteur ligne $\Delta f = (\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)$ dans le cas de trois variables. La méthode consiste alors à passer du point M_k au point M_{k+1} en cherchant le $t \leq 0$ qui permet de minimiser la valeur de $f(M_k - t\,\Delta f_k)$ en posant pour cette valeur t, $M_{k+1} = M_k - t\,\Delta f_k$.

Solution : trouver le minimum de la fonction $\phi(t) = f(M_k - t\,\Delta f_k)$ n'est pas facile, aussi on peut simplement procéder avec un pas dt assez petit, et augmenter t de ce pas tant que ϕ dimimue. Quant à l'arrêt de la procédure, il est également déterminé par l'arrêt de diminution de ϕ. En fait dans la solution ci-dessous nous avons testé l'arrêt sur un point singulier pour lequel le gradient possède une norme de Hamming inférieure à un seuil.
Cette méthode donne bien le minimum sur tout domaine où la fonction est convexe, mais ne donne qu'un minimum local sinon.
On pourra tester sur bol $(x, y) = \sqrt{(1 - x^2 - y^2)}$ dont le minimum -1 est réalisé en (0, 0), mais la fonction de Rosenbrock $f(x, y) = (x - 1)^2 + 10\,(x^2 - y)^2$ est plus intéressante à cause de sa vallée en forme de banane, qui conduit au minimum 0 pour (1, 1).

```
program gradient;
        var u, v, mini : real;
function f (x, y : real) : real;          {fonction de Rosenbrock}
        begin f := sqr(x-1) + 10* sqr (sqr(x) - y) end;
function df1 (x, y : real) : real;        {dérivée partielle par rapport à x}
        begin df1 := 2*(20*x*(sqr(x) - y) + x - 1) end;
function df2 (x, y : real) : real;        {dérivée partielle par rapport à y}
        begin df2 := -20*(sqr(x) - y) end;

procedure descente (var x, y, m0 : real; eps : real);
        var t, dx, dy, m1 : real; n, p : integer;
        begin n:= 0; p:= 1;
        repeat writeln ('nouveau point ',p); m1 := f(x, y); dx := df1 (x, y); dy := df2 (x, y); t := 0;
                repeat m0 := m1; t:= t + eps; x := x -t*dx; y := y - t*dy; m1 := f(x, y) ; n := n+1
                until (m1 >= m0); {on a fini une descente suivant un gradient}
                p := p + 1; x := x + t* dx; y := y + t*dy  {on revient au point précédent }
        until abs(dx) + abs(dy) < eps ;
        end;
begin u := -1; v := 1; descente (u, v, mini, 0.001);
write ('Minimum ', mini:6:3, ' pour x = ', u:6:3, ' et y = ', v:6:3) end.
```

On part ici du point (-1, 1) pour s'apercevoir qu'on longe une vallée où x augmente et y diminue presque à 0 avant de remonter vers 1. Pour $\varepsilon = 0{,}001$ il faut 4551 itérations et 619 mesures de dérivées partielles avant d'obtenir le minimum $7{,}3.10^{-7}$.

Une variante plus simple à réaliser, est de faire descendre le long du gradient d'un pas constant, soit en appliquant toujours le même dt, soit en refaisant une mesure du gradient de telle sorte que M_kM_{k+1} soit une distance constante. Une autre variante consiste à faire descendre le long de toutes les coordonnées l'une après l'autre (méthode de relaxation), mais l'ordre n'est pas indifférent et la méthode est lente et inefficace.

2-44° Méthode du gradient conjugué. Pour trouver le minimum d'une fonction réelle de plusieurs variables, on part également d'un point arbitraire M_0, puis en posant $u_0 = \Delta_0$, et Δ_k le gradient en M_k, on passe de M_k au point M_{k+1}, si A matrice symétrique définie positive (on peut prendre l'identité ou le Hessien de f) en posant :

$$\overrightarrow{M_kM_{k+1}} = -(u_k^t.\Delta_k)\frac{\overrightarrow{u_k}}{|u_k^2|} \text{ et } \overrightarrow{u_{k+1}} = \Delta_{k+1} -(u_k^t.A.\Delta_{k+1})\frac{\overrightarrow{u_k}}{|u_k^2|}$$

2-45° Plateau d'Arsac. Un tableau t indicé de 1 à n est trié par ordre croissant, on cherche la longueur du plus grand plateau (suite d'éléments consécutifs égaux) dans t.
Programmer en une seule boucle et avec seulement deux variables (faire un exemple).

Solution

```
p := 1; i := 2;
while   i < n  do       begin if t[i] = t[i-p] then p := p+1; { l'astuce consiste à regarder p crans derrière }
                        i := i+1
                        end;
```

A la fin de la boucle, p est le résultat cherché. Il est également possible de regarder en avant en faisant p ← 1; i ← 1;
tant que i + p ≤ n faire si t[i] = t[i + p] alors p ← p + 1 sinon i ← i + 1

2-46° Problème de Dijkstra. Un tableau contient 3 couleurs bleu, blanc, rouge en nombre quelconque et dans le désordre. On souhaite ranger le tableau dans l'ordre bleu, blanc, rouge en une seule boucle.

Supposons le processus en cours d'exécution, soit p l'indice du dernier bleu, puis des éléments encore non examinés dont le dernier possède l'indice q, et r l'indice du dernier blanc, les éléments de r+1 à n étant rouges. On aura :

```
p := 0; q := n; r := n;
repeat if t[p+1] = 'bleu' then  p := p+1
                        else   if t[p+1]  = 'blanc' then begin echange (t[p+1], t[q]); q := q-1 end
                                                    else      begin echange (t[r], t[q]);
                                                              echange (t[r], t[p+1]);
                                                              r := r-1;
                                                              q := q-1
                                                              end;
until  p = q
```