CHAPITRE 1

INTRODUCTION

Les grands principes de programmation à résumer pourraient être les suivants. Tout d'abord tout problème consiste à fournir des résultats à partir de données, entre les deux il doit donc y avoir un algorithme c'est à dire des méthodes de calcul (une boîte restant noire pour l'utilisateur), ce qui prime donc c'est :

Dès que le problème est important, il faudra le découper en morceaux, le hiérarchiser, faire en sorte que le programme principal soit un directeur faisant appel à des subordonnés, lesquels feront eux-même appel à des sous-traitants etc... chacun d'entre eux devant spécifier avec exactitude le travail qu'il demande pour en récupérer les fruits, (problème du passage des paramètres) mais n'ayant pas vraiment besoin de savoir comment fait son employé ni avec quels outils. (Pas d'outils servant à tous le monde donc pas de variables globales ou le moins possible, par contre des outils analogues peuvent porter le même nom.) Cette conception du partage ouvre la voie à la compilation séparée d'unités ayant ses algorithmes mais aussi ses propres types d'objets, c'est la :

Enfin une des méthodes essentielles de la programmation évoluée, qui doit à son tour influencer jusqu'à la façon de penser et d'analyser les problèmes, c'est la :

Créé en 1970 par N. Wirth à Zürich, le langage Pascal permet d'écrire des programmes relativement bien structurés, pseudo-modulaires (avec des sous-programmes bien distincts), qui les rendent facilement compréhensibles dans la mesure où on n'est pas avare de commentaires, où on choisit des "identificateurs "(noms des variables) significatifs, et où on fait très attention à la mise en page.

Cette qualité de lisibilité est la chose la plus importante à acquérir pour un débutant, la rédaction rigoureuse de tout travail écrit. Une des qualités du Pascal est, encore une fois, cette lisibilité qui permet de retrouver plus facilement une erreur et qui permet en outre de modifier facilement certaines parties du programme sans toucher aux autres.

Une fois écrit, un programme Pascal doit subir une compilation pour être exécuté. Cette phase consiste en une traduction dans le langage (binaire) de la machine, après une vérification de la morphologie (l'orthographe) des mots employés (mots réservés du Pascal et identificateurs choisis par le programmeur), une vérification syntaxique (grammaticale) sur les enchaînements de mots, et enfin une vérification sémantique sur le nombre et le type des différents objets manipulés par le programme.
Tous ces types d'erreur sont signalés lors de la compilation, par contre les erreurs de conception, à propos de ce que doit réaliser le programme, ne peuvent être décelées que lors de son exécution.

La première tâche, en vue de la résolution automatique d'un problème, c'est la spécification ou description précise du but que l'on se fixe. Il s'agit avant tout d'un travail d'expression écrite, essentiel pour tous les scientifiques : savoir exposer pour d'autres qu'eux-mêmes, et notamment pour une machine, les données techniques d'un problème.

En second lieu, il faut déterminer un algorithme, c'est-à-dire une méthode de résolution.
Conformément à l'esprit cartésien on tente de découper le problème en "sous-problèmes" suffisamment simples pour ce qui concerne leur résolution, et tels, que le problème principal puisse se planifier suivant une hiérarchie de ces petits sous-problèmes. Cette tâche d'analyse n'est jamais facile et peut naturellement recevoir bien des solutions différentes dès que le problème est important.

Comment faire ce découpage ? Peut-on suivre une analyse "descendante" allant de la compréhension générale d'un problème à sa représentation hiérarchisée ?
Ou bien peut-on avoir une stratégie "ascendante" allant des questions simples et réduites, vers celles, complexes, qui en sont des assemblages ? Il est très difficile de répondre à ces questions, l'activité cognitive, même des programmeurs expérimentés, ne suivant pas nécessairement une stratégie bien fixée.
Il est illusoire, par ailleurs, de vouloir donner à tout prix une description d'algorithme indépendante du langage de programmation, car celle-ci n'est que l'aboutissement formalisé d'une suite de descriptions de plus en plus rigoureuses partant du langage naturel souvent flou, incomplet pour certains détails, redondant pour d'autres, et passant par des descriptions écrites souvent données par ce qu'on appelle le "pseudo-Pascal".

Plus précisément, les principales instructions d'un langage de description d'algorithme sont :

l'affectation (attribution d'une valeur à une variable notée comme $X \leftarrow 3$ ou X := 3), notons encore qu'une variable est dite incrémentée lorsque sa valeur augmente d'une unité, et décrémentée lorsqu'elle diminue de 1.
la sélection (si...alors...sinon...),
l'itération (commander un travail répétitif),
le débranchement (aller à... , retourner vers...), et bien sûr, l'appel à un sous-programme et en particulier :
les entrées et sorties de valeurs au clavier de résultat à l'écran ou vers d'autres périphériques.

Nous allons donner dans cette introduction, tout d'abord un exemple, simple en apparence, qui montrera les difficultés d'énonciation du problème et de l'algorithme. La recherche de la rigueur, conduisant nécessairement au travers de plusieurs étapes, à l'utilisation d'un langage formalisé.
Le second exemple que nous donnons est au contraire assez simple et bien connu pour son énoncé, mais nous voulons montrer qu'il est indispensable de le décomposer en sous-problème, si l'on ne veut pas arriver à des formules inextricables.

1-1° Exemple de problème simple difficile à spécifier : l'escargot

1	2	3	4	5	6	7	8
20	21	→	→	etc...			9
19							10
18	17	16	15	14	13	12	11

a) Présentation "humainement compréhensible"
On peut considérer que le schéma ci-dessus constitue déjà une description de ce que l'on veut faire, mais nous allons préciser le problème lors de différentes phases accompagnées de discussions plus techniques sur la représentation des données et le partage des tâches.

b) Première description informelle
"Remplir un tableau avec les entiers consécutifs, en partant de la case en haut à gauche occupée par 1, et en tournant dans le sens de l'horloge, tout en longeant les bords déjà occupés, et en finissant au centre du tableau sur la dernière case vide."
On voit à la complexité de cette phrase, ne rendant d'ailleurs pas complètement compte du cheminement, toute la difficulté d'exprimer quelque chose d'évident sur un schéma. On est conduit à choisir des noms de variables, et à décrire dans l'ordre et par le menu les opérations qu'il faut effectuer sur ces variables, si l'on veut préciser davantage.

c) Seconde description plus formelle
Soit un tableau A de N lignes et P colonnes et X=1 placé dans la première case. On remplit alors, tant qu'elles sont vides, les cases de la première ligne, de la dernière colonne, de la dernière ligne de droite à gauche, et enfin de la première colonne de bas en haut, avec X incrémenté à chaque fois. Puis on recommence avec le sous-tableau de N-2 lignes et P-2 colonnes encore libre et le X suivant la dernière valeur qui a été placée.
Cette phrase, sans doute plus complète, néglige encore un détail important: celui de l'arrêt du processus, mais, et c'est ce qui est très important, elle sous-entend l'idée de récurrence sur un tour de tableau, ce qui représente déjà une idée pour la programmation ultérieure.
Remarque : l'idée suivant laquelle on peut chercher une formulation du terme d'indice i et j dans A, constitue un autre algorithme qui non seulement risque d'être assez laborieux à analyser, mais encore n'apporterait pas d'amélioration au programme : il y aura toujours N*P "affectations". Il est beaucoup plus élégant de construire quatre tâches se succédant.

d) Division du problème en sous-tâches et représentation des données
Parmi tous les programmes (écriture formalisée d'un algorithme) possibles, on choisira celui qui découpe au maximum le travail à faire, en sous-tâches .
On écrira pour cela quatre "procédures" indépendantes: ligne-haute, colonne-droite, ligne-basse, colonne-gauche se passant le relais à tour de rôle, avec à chaque fois un paramètre modifié : le numéro de ligne ou de colonne qui vient d'être rempli.
On désignera par X la dernière valeur logée dans une case, et par L0, L1, C0, C1 les numéros de lignes et de colonnes à remplir dans le tableau A. En pseudo-Pascal:

```
LGN.HAUTE (L0, L1, C0, C1)
        Pour C incrémentant de C0 à C1 faire X ← X + 1 puis A (L0, C) ← X
        Si L0 < L1 alors COL.DROITE (L0 + 1, L1, C0, C1)

COL.DROITE (L0, L1, C0, C1)
        Pour L incrémentant de L0 à L1 faire X ← X + 1 puis A (L, C1) ← X
        Si C0 < C1 alors LGN.BASSE (L0, L1, C0, C1-1)

LGN.BASSE (L0, L1, C0, C1)
        Pour C décrémentant de C1 à C0 faire X ← X + 1 puis A (L1, C) ← X
        Si L0 < L1 alors COL.GAUCHE (L0, L1-1, C0, C1)
```

COL.GAUCHE (L0, L1, C0, C1)
Pour L décrémentant de L1 à L0 faire X ← X + 1 puis A (L, C0) ← X
Si C0 < C1 alors LGN.HAUTE (L0, L1, C0 + 1, C1)

L'arrêt pouvant se situer à la fin de l'une quelconque des quatre procédures.

e) Programme en langage évolué

Le programme proprement dit nécessite encore la demande des valeurs de N et P , puis l'exécution de la machinerie sera lancée par LGN.HAUTE pour les paramètres (1, N, 1, P), X ayant été préalablement initialisée à 0. Enfin il faut éditer à l'écran le tableau A, on peut bien sûr concevoir directement cette édition sans utiliser A.

```
program escargot;
        var n, p, x : integer ; {déclaration des 3 variables du programme}
procedure ecrire (var x : integer; L, C : integer); {modifie et écrit X}
        begin x := x + 1; gotoxy (5*C - 4, L); write (x) end;
procedure lignehaute (L0, L1, C0, C1 : integer); forward;
{Ce mot "forward" indique au compilateur que la définition de "lignh" se trouve plus loin}

procedure colonnegauche (L0, L1, C0, C1 : integer);
        var L : integer;
        begin for L := L1 downto L0 do ecrire (x, L, C0);
        if c0 < c1 then lignehaute (L0, L1, C0 + 1, C1) end;

procedure lignebasse (L0, L1, C0, C1 : integer);
        var C : integer;
        begin for C := C1 downto C0 do ecrire (x, L1, C);
        if L0 < L1 then colonnegauche (L0, L1 - 1, C0, C1) end;

procedure colonnedroite (L0, L1, C0, C1 : integer);
        var L : integer;
        begin for L := l0 to L1 do ecrire (x, L, c1);
        if C0 < C1 then lignebasse (L0, L1, C0, C1 - 1) end;

procedure lignehaute; { (L0, L1, C0, C1 : integer) ne se répète pas }
        var C : integer;
        begin for C := C0 to C1 do ecrire (x, L0, C);
        if L0 < L1 then colonnedroite (L0 + 1, L1, C0, C1) end;

procedure escargot (n, p : integer);
        begin x := 0; lignehaute (1, n, 1, p) end;

{Début du programme réalisé pour 24 lignes et 16 colonnes au maximum}
begin write ('Combien de lignes ? '); readln (n); write ('Combien de colonnes ? '); readln (p);
escargot (n, p) end.
```

Exécution pour 8 lignes et 12 colonnes.

```
 1   2   3   4   5   6   7   8   9  10  11  12
36  37  38  39  40  41  42  43  44  45  46  13
35  64  65  66  67  68  69  70  71  72  47  14
34  63  84  85  86  87  88  89  90  73  48  15
33  62  83  96  95  94  93  92  91  74  49  16
32  61  82  81  80  79  78  77  76  75  50  17
31  60  59  58  57  56  55  54  53  52  51  18
30  29  28  27  26  25  24  23  22  21  20  19
```

Dans la suite, on écrira directement en Pascal car il n'y a que les détails de syntaxe vus au cours des prochains chapitres, qui ne doivent pas arrêter un débutant, celui-ci les apprenant petit à petit en travaux pratiques, et aussi éventuellement une gêne due à la traduction des mots anglais. La conception générale des programmes doit, par contre, être étudiée avec soin sur le papier.

1-2° Equation complexe du second degré, construire un programme calculant les deux solutions complexes d'une équation du type $az^2 + bz + c = 0$ où a, b, c sont trois complexes donnés.

Ce problème est intéressant, sans même qu'il soit question pour l'instant de définir un type de donnée complexe, car il doit recevoir en entrée trois complexes c'est à dire six réels, les parties réelles et imaginaires de a, b, c, et fournir deux sorties complexes, c'est à dire les quatre réels parties réelles et imaginaires des deux solutions de l'équation. Nous rappelons que les formules $z = (-b \pm d) / (2a)$ où +d et -d sont les racines carrées de $\Delta = b^2 - 4ac$, sont toujours valables.
Ce problème des racines carrées d'un complexe est à traiter indépendamment de celui de l'équation et il pourra d'ailleurs être repris pour un autre programme. Considérons formellement un complexe a + ib, ses racines carrées se trouvent en posant $(x + iy)^2 = a + ib$, ce qui en développant conduit au système :

$$x^2 - y^2 = a$$
$$2xy = b$$

Le fait d'égaler les modules donne une troisième équation (redondante) mais qui permet de résoudre simplement le système : $x^2 + y^2 = m$ où m est le module a + ib.
On a donc $x^2 = (m + a) / 2$ et $y^2 = (m - a) / 2$, le signe de b permet d'avoir le signe de y correspondant à un x positif D'autre part on sait que tout complexe admet deux racines carrées opposées, ce qui achève d'expliquer le sous-programme ci dessous.
Un autre sous-problème mérite d'être traité à part, c'est celui de la division de deux complexes, car il intervient deux fois ici.

```
program complexe;
function module (a, b : real) : real; ( fonction donnant le module d'un complexe a + ib }
        begin module := sqrt (sqr (a) + sqr (b)) end;
procedure racinecarree (a, b : real; var x, y : real);
        { calcule une racine carrée x + iy de a + ib, l'autre étant -x - iy}
        var m : real;
        begin m := module (a, b);
        x := sqrt ((m + a) / 2); y := sqrt ((m - a) / 2);
        if b < 0 then y := -y end;
procedure division (a, b, c, d : real; var e, f : real); { calcule e + if = (a + ib) / (c + id) }
        var r : real;
        begin r := sqr (c) + sqr (d);
        e := (a*c + b*d) / r; f := (b*c - a*d) / r end;
procedure affiche (a, b : real); {affiche a + ib avec 2 décimales, chaque valeur sur 5 colonnes}
        begin writeln (a : 5 : 2, ' + i ', b : 5 : 2) end;

var ar, ai, br, bi, cr, ci, dr, di, zr, zi : real; { sont les variables du programme proprement dit }
begin
write ('Donnez les 6 réels correspondant aux parties réelles et imaginaires des coeff a, b, c de ax2 + bx + c = 0 ');
readln (ar, ai, br, bi, cr, ci);
racinecarree (sqr (br) - sqr (bi) - 4*ar*cr + 4*ai*ci, 2*br*bi - 4*ar*ci - 4*ai*cr, dr, di);
division (- br - dr, - bi - di, 2*ar, 2*ai, zr, zi); affiche (zr, zi);
division (- br + dr, - bi + di, 2*ar, 2*ai, zr, zi); affiche (zr, zi);
end.
```

Execution, pour $(4 - i) z^2 + (-29 + 3i)z + 75 + 28i = 0$, on obtient les solutions 2 + 3i et 5 - 2i.