# Images pour programmer : apprendre les concepts de base. Tome 1

Duchâteau, Charles; Arsac, Jacques

*Publication date:*
1990


# Images pour programmer

1ère partie

Charles Duchâteau


**Département**
**Éducation**
**et Technologie**

- Programmer ?
- Un algorithme ?
- L'exécutant-ordinateur
- Premiers tours de main

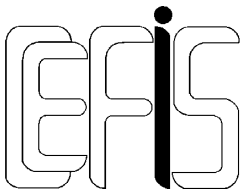
# Images pour programmer

## Apprendre les concepts de base
## 1ère partie (version revue en 2002)

Charles Duchâteau

Préface de Jacques Arsac

Cet ouvrage, publié en 1990 aux éditions De Boeck dans la collection "Accès Sciences" est à présent épuisé. Le présent document en reprend le contenu.




Centre pour la Formation à
l'Informatique dans le Secondaire

# Préface

Encore un livre de programmation, et qui pis est, préfacé par Arsac ! “On pourrait faire un livre avec vos préfaces”, me dit un jour un collègue, et je ne suis pas sûr que ce fût un compliment... Mais comment résister au plaisir de dire le bien que je pense de ce livre, “car le bien pour un livre, c’est d’être lu...” (Umberto Eco).

C’est un livre de programmation, il est vrai, et Charles Duchâteau s’en explique ici. Mais peut-être a-t-il été timide dans son argumentation. Voilà trente ans que j’entends répéter “dans dix ans, plus personne ne programmera”. Or on continue à investir dans de nouveaux langages de programmation, qu’ils s’appellent ADA, PROLOG ou “langages orientés vers les objets”. Le ferait-on si la programmation était condamnée à mort ?

Nous avons maintenant, il est vrai, de grands progiciels, traitements de textes, tableurs ou gérants de bases de données qui permettent à un utilisateur non informaticien de se servir d’un ordinateur sans avoir à connaître la programmation. Mais cela n’a pas fait disparaître un seul programmeur : cela a seulement augmenté le nombre des utilisateurs. Or ceux-ci ont vite fait de s’apercevoir que ces progiciels ne font pas tout. Mon traitement de textes (je ne vous dirai pas lequel : pas de publicité !) ne me permettait pas d’utiliser la petite flèche vers la gauche “←” dans mes papiers. Que faire ? Utiliser le signe “<“ et le tiret : “<--”. Beaucoup moins joli ! Je sais programmer, je me suis arrangé.

Que fait l’utilisateur qui ne sait pas programmer ? Il s'adresse au fabricant du progiciel, qui l’écoutera peut-être, et sortira la version 27.4 de son produit où l’on obtiendra la petite flèche “←” en tapant sur les touches “CTRL ALT Q et R”. Et les progiciels ne cessent de se complexifier, devenant d'apprentissage de plus en plus difficile, nécessitant de consulter souvent la carte aide-mémoire qu’on livre avec le produit, à moins que vous ne fassiez défiler sur l’écran nombre de menus avant d’avoir le renseignement cherché... L’avenir n’est pas aux dinosaures, mais à des systèmes simples d’apprentissage rapide, et tels qu’on puisse retenir aisément les commandes les plus fréquentes.

Comme Charles Duchâteau le montre dans ce livre, programmer c’est commander, ou se faire obéir : faire faire une tâche par un exécutant dont on connaît la compétence. Ce qui implique que l’on connaisse la marche à suivre pour exécuter cette tâche, et qu’on soit capable de la lui communiquer en anticipant correctement les actions qui devront être faites. Est-ce compliqué ? Est-ce un investissement perdu ?

Mais, me dira-t-on (car je fais les questions et les réponses !), si c’était si facile que cela, pourquoi ne programme-t-on pas davantage ? Pourquoi les progiciels coûtent-ils si cher ? Je ne prétends pas qu’ayant lu ce livre, vous serez capable d’écrire un nouveau traitement de textes. Mais vous aurez compris ce qui est en jeu dans la programmation, et vous pourrez rédiger des applications simples, voire utiliser le petit langage de programmation incorporé dans certains progiciels (par exemple, dans des gérants de bases de données).

Continuant ce jeu de questions-réponses, comment se fait-il qu'on ait attendu si longtemps pour proposer ces idées simples ? Parce que la technique ici a précédé la science. Les ordinateurs sont apparus, avec "leur langage de machine", et l'on s'est débrouillé pour s'en servir. Puis un informaticien génial, John Backhus, eut l'idée du premier langage évolué (Fortran). Ces langages se multiplièrent, et on s'occupa beaucoup d'eux, les étudiant, les compilant, les enseignant. Ces activités occultaient complètement les vrais problèmes de la programmation. Il a fallu attendre la fin des années 60 pour que Edsger Dijkstra pose enfin les vrais problèmes de la programmation, déclenchant ainsi un vaste travail de recherche qui aboutit à faire de la programmation une discipline scientifique. Il fallut encore du temps pour que les résultats de la recherche produisent une mutation profonde de la pédagogie de la programmation.

Mais, me direz-vous encore, s'il a fallu si longtemps pour percevoir le problème, et de tels efforts de recherche pour le maîtriser, c'est que la programmation est extrêmement compliquée, nécessite une théorie savante, et n'est accessible qu'aux forts en maths. L'histoire des sciences montre que la solution simple est rarement la première qui s'impose à l'esprit. Le grand mathématicien Lighthill a dédicacé un de ses ouvrages "à Dirac, qui supposa que c'était vrai; à Schwartz qui montra que c'était vrai; à Temple, qui montra que c'était simple". Nous en arrivons, en programmation, à la phase où des pédagogues montrent que c'est simple. Lisez ce livre, et vous verrez qu'il n'attend de vous que l'effort de bien vouloir suivre l'auteur dans la voie qu'il vous trace, et ne suppose aucune compétence en sciences.

Il est vrai, aussi, qu'il n'y a pas de "vérité" en pédagogie, ni établie par la théorie, ni révélée, ni même garantie par une pratique constante. Les professeurs sont divers, divers aussi les élèves. Ma pédagogie n'est pas en tous points celle de Charles Duchâteau. Mais "il y a plusieurs demeures dans la maison du Père". Je crois beaucoup à la voie suivie par l'auteur : je crois que la plupart des gens ont besoin d'images pour s'approprier des concepts. Il est difficile de s'en tenir à de purs concepts non soutenus par des représentations concrètes, même si celles-ci sont imparfaites. On a voulu enseigner des mathématiques formelles, où la droite était un objet mathématique sans rapport avec quoi que ce soit du monde sensible. J'avais, dans ma jeunesse, appris que ce que l'on obtient en traçant un trait avec une règle, ou qu'un fil tendu, sont des images de la droite. Il est vrai que ce sont des images imparfaites. Mais il est plus facile d'abstraire une vraie droite à partir de ces exemples que de travailler sur une droite qui n'est qu'un nom : "nomina nuda tenemus" (nous utilisons des noms vides) terminait le film "le nom de la rose" d'Umberto Eco. Peu d'esprits peuvent se satisfaire de noms vides...

Je suis content que ce livre offre des images à l'apprenti programmeur. Même si elles sont imparfaites, comme l'étaient les images de la droite, du moins permettront-elles de rendre plus significatifs les concepts de la programmation. Le lecteur apprendra plus tard à dépasser ces images. Chaque chose en son temps. Ce livre précède les ouvrages classiques de programmation. Lisez-le, c'est facile, et cela devrait vous rapporter de pouvoir démarrer en programmation, et, éventuellement, de vous lancer dans la lecture d'ouvrages plus techniques pour devenir expert en la matière.

Jacques ARSAC

# AVANT-PROPOS

Voici[1] non seulement encore un nouvel ouvrage qui traite d'algorithmique et de programmation, mais pire, ce doit être la nième fois que l'auteur, dans son avant-propos, tente de justifier "pourquoi" un nouvel ouvrage sur la programmation. Je vais donc être forcé de tenir compte non seulement du fait que je suis l'un des derniers (sur le plan chronologique !) des quelques centaines d'auteurs d'ouvrages consacrés à la programmation, mais encore que quelques-uns de mes devanciers ont consacré une énergie et un talent non négligeables à expliquer dans l'avant-propos de leur ouvrage pourquoi ils avaient cru bon d'enrichir la liste, déjà fort longue, des livres traitant de ce sujet.

Cette tâche est particulièrement périlleuse, étant donné que de multiples facteurs (que je vais évoquer) semblent montrer que le moment est assez mal choisi de sortir un ouvrage (supplémentaire !) consacré à la programmation.

## Constatation n° 1

Pendant les années d'éclosion de la micro-informatique (grosso modo de la fin de la décennie 70 au milieu des années 80), la seule chose "amusante" que l'on pouvait faire avec un ordinateur (et même souvent la seule chose possible), c'était de le "programmer". Entendez de lui fournir - je n'ai pas écrit de concevoir - l'un ou l'autre programme en Basic, puis de s'atteler au jeu des corrections successives qui, peu à peu, allaient (parfois) conduire à la satisfaction de lui voir faire le tracé d'un histogramme ou la conjugaison d'un verbe (régulier !). Et c'est vrai qu'il y avait bien des aspects ludiques dans cette traque incessante des incohérences et des erreurs qui jalonnaient ces productions. Pour peu, on aurait non seulement été fort surpris, mais encore quelque peu déçu, d'avoir (par mégarde, sans doute) écrit un programme qui non seulement marchait du premier coup, mais encore fournissait les résultats escomptés. On était alors privé de ce subtil plaisir de la chasse aux "bugs"[2] et de la mise au point par essais et erreurs. L'utilité effective de ces réalisations était le plus souvent aléatoire. Heureusement, ce n'était pas parce que cela allait être utile qu'on écrivait ces programmes, mais simplement parce que c'était amusant.

L'évolution de la micro-informatique (qui s'est "professionnalisée") fournit aujourd'hui, à travers une multitude de progiciels, de nouvelles manières de "jouer" et de prendre du plaisir à travailler avec les ordinateurs. Il y a dix ans, on achetait un ordinateur tout juste capable de comprendre le Basic. Aujourd'hui, on a un interlocuteur aux multiples facettes, aussi bien capable de se comporter comme une "super machine à écrire" (lorsqu'il est équipé d'un logiciel de traitement de texte), que comme un dessinateur soigneux, rapide et infatigable (quand il est transfiguré par un

[1] A noter que ces lignes ont été écrites en 1989 !

[2] Les "bugs, dans le jargon informatique, sont les erreurs qui subsistent dans les programmes destinés à faire travailler l'ordinateur.

logiciel graphique). De là à jeter aux oubliettes la "programmation" (entendez l'écriture "sur le tas" de programmes), il n'y a qu'un tout petit pas : grâce aux progiciels, on peut non seulement conserver le plaisir de faire travailler l'ordinateur, mais encore on le fait avec bonne conscience puisqu'en plus d'être amusant, c'est même devenu utile !

Ce mouvement me semble particulièrement sensible en ce qui concerne la vision du contenu des cours d'informatique dispensés dans l'enseignement secondaire. Bon nombre des enseignants, pionniers de la micro-informatique et de son enseignement, faisaient partie de ceux-là, pour qui "utilisation de l'ordinateur" rimait avec "programmation" (au sens où j'en parle ci-dessus). Ces "mordus" ont aujourd'hui découvert les multiples progiciels et leurs utilisations.

Dès lors, puisqu'il n'est plus nécessaire de programmer pour utiliser un ordinateur, à quoi bon encore enseigner la programmation ![3]

Et de là, quelle est encore l'utilité d'écrire des ouvrages consacrés à l'apprentissage de la programmation ?

## Réponse n° 1

Ce que je vais tenter de montrer, au long de ces pages, c'est qu'il faut bien savoir de quoi l'on parle lorsqu'on évoque la programmation et son apprentissage. Si, de fait, programmer c'était se centrer sur l'apprentissage de la syntaxe d'un langage de programmation, pour fournir à l'ordinateur, sans réelle réflexion préalable, un programme bâti n'importe comment, alors ce type d'activité n'aurait absolument pas sa place dans une perspective de formation. Heureusement, programmer, c'est bien autre chose que d'être un virtuose du clavier (d'ordinateur !) ou incollable sur les finesses grammaticales de Basic, LSE ou Pascal !

Je crois qu'une découverte de quelques-uns des outils que l'informatique met à notre disposition est souhaitable. Chacun le sait, les qualités de ces "outils" sont essentiellement fonctions de la "matière grise ajoutée" à l'ordinateur sous la forme des programmes qui vont le faire agir : l'outil, c'est toujours le tandem matériel-logiciel, ce dernier étant la concrétisation de la créativité, de la rigueur et de l'habileté du programmeur qui en est l'auteur.

Ne plus jeter sur l'informatique qu'un regard d'utilisateur, uniquement préoccupé de la maîtrise de quelques outils et de l'acquisition des tours de mains présidant à leur emploi, c'est non seulement passer à côté de tout un pan de cet univers, mais encore négliger de nombreuses occasions d'amener les apprenants à certains modes de pensée (comme la pensée algorithmique) ou à des attitudes de travail (comme le souci d'être compris, exhaustif, ...).

Ces pages vont le montrer, l'algorithmique (et la programmation qui en est une incarnation), c'est l'art et la méthode de "déplier" complètement une tâche pour l'expliquer et la faire faire par "un autre". Ce passage d'un "savoir-faire" à un "savoir faire faire", qui est le cœur profond de l'algorithmique, nous place à des lieues des problèmes de matériel, de technique ou d'utilisation d'outils.

Il y a cinq ans encore, il était indispensable de prouver que l'apprentissage de la programmation ne se réduit pas à la connaissance du fonctionnement d'un ordinateur et à l'acquisition des éléments d'un langage de programmation. Je pense que nous n'en sommes plus là aujourd'hui et que chacun sait que, s'il fallait chercher des synonymes à "programmer", on les trouverait plutôt du côté d' "organiser", "décortiquer", "créer", que du côté "pianoter" ou "jargonner"

[3] Je ne parle évidemment pas ici des formations professionnelles destinées aux futurs informaticiens : pendant quelques années encore il faudra bien des programmeurs pour créer les logiciels que d'autres utiliseront et il faudra donc bien leur enseigner la programmation.

et que la programmation est plus proche de la composition française que des règles d'accord du participe ou ... de la dactylographie.

## Constatation n° 2

L'apprentissage de la programmation, même bien conduit (Cf. la réponse n° 1) tel qu'il est conçu dans le cadre de la plupart des initiations à l'informatique comme, par exemple, celles destinées à des élèves de l'enseignement secondaire, ne débouche pas sur une maîtrise suffisante pour qu'à son terme, les nouveaux "initiés" soient réellement devenus des "programmeurs". Il est de loin préférable de leur faire découvrir la manipulation des outils logiciels qui eux seront effectivement utiles et utilisés.

Cette opinion peut, à l'orée de la découverte du monde de la programmation proposée ici, prendre la forme d'une question préalable : "lorsque nous refermerons cet ouvrage (après l'avoir lu !), serons-nous devenus des "programmeurs" (avec, sans doute, les attributs de compétence et d'efficacité attachés à cette profession) ?

## Réponse n° 2

Si le critère d'utilité directe et immédiate des apprentissages est mis en avant, alors il est vrai que l'étude de la programmation doit être supprimée du curriculum de formation des élèves. L'application de ce même critère conduit d'ailleurs, dans la foulée, à supprimer aussi :

- l'analyse littéraire et la composition française;
- l'étude du grec et du latin;
- le cours de mathématique;
- les activités artistiques (dessin, musique, ...).[4]

Je propose que les nombreuses heures ainsi libérées soient consacrées à l'apprentissage des techniques permettant de mener à bien les menus travaux de plomberie et d'électricité domestique et surtout à l'étude des langues étrangères et à la dactylographie[5]. Nous n'aurons plus rien à nous dire, mais nous pourrons l'exprimer en plusieurs langues; nous n'aurons plus rien à écrire, mais nous pourrons le faire rapidement.

Mon objectif s'inscrit dans cette vision non "utilitariste" : je ne veux pas ici former des professionnels de l'informatique (et plus particulièrement de la programmation); ma perspective est plus "culturelle" que "technique", plus "formative" que "pratique". Je souhaite fournir au lecteur des cartes pertinentes du monde de la programmation, lui signaler les collines importantes (qui vont permettre les points de vue intéressants), lui faire voir quelques paysages typiques et non en explorer longuement les moindres recoins.

## Constatation n° 3

Même si l'on se range aux arguments avancés et si l'on accepte, bon gré mal gré, de reconnaître, au-delà d'une perspective directement utilitaire, une valeur formative à l'apprentissage de la programmation, pourquoi faut-il ajouter un ouvrage supplémentaire à la liste (fort longue) de

4 Je vous laisse le soin de compléter la liste.

5 Je ne suis pas le moins du monde opposé à l'étude des langues et je trouve l'apprentissage de la dactylographie fort utile ... ne serait-ce que pour communiquer avec les ordinateurs par l'intermédiaire d'un clavier.

ceux qui traitent déjà de ce même sujet ? Que dire qui n'ait déjà été maintes fois dit ? Qu'apporter de neuf dans un domaine déjà cent fois exploré ?

## Réponse n° 3

Cette objection est probablement la plus malaisée à rencontrer. J'y ferai face en précisant d'où vient cet ouvrage et quels en sont les objectifs.

1. Depuis 1981, j'assure une formation à la programmation pour des enseignants eux-mêmes chargés du cours d'informatique dans le secondaire. La brièveté de cette initiation oblige évidemment à une présentation des concepts essentiels qui les rende digestes et compréhensibles pour des "apprenants" adultes qui n'ont guère de références dans ce domaine. De plus, dans le cas d'un tel public, l'emballage est aussi important que le contenu.
2. Mon environnement de travail est essentiellement constitué de spécialistes de la pédagogie. Les questions de mes collègues pédagogues à propos de la programmation et de son apprentissage sont de vraies questions : il me faut souvent des mois pour en cerner la portée et certaines m'ont demandé des années de réflexion et d'expérimentation pour avancer des réponses pertinentes.
3. A travers mon apprentissage "sur le tas", je me suis peu à peu forgé mes propres représentations (souvent métaphoriques et imagées) des concepts et méthodes de la programmation. J'ai pu constater, à travers mon enseignement, que ces approches étaient utilisables et partageables et qu'elles aidaient les apprenants. Elles se sont d'ailleurs affinées, enrichies et développées au fil des années et ont été passées au crible des réactions des enseignants en formation.
4. Dans la perspective d'enseignement qui reste la mienne, il est excitant de se trouver devant une étendue pratiquement déserte : celle de la didactique de l'informatique et, plus précisément, de la didactique de la programmation et de l'algorithmique. Même si des devanciers illustres[6] ont ouvert des voies, tracé des itinéraires et balisé des parcours, beaucoup reste à concevoir et à expérimenter. C'est passionnant et c'est cette passion que j'aimerais transmettre à mes lecteurs. Au-delà de son contenu même (les méthodes, concepts et outils évoqués), la manière de l'aborder se veut "exemplaire" d'une approche méthodologique (au sens de l'enseignement) possible et exploitable.
5. Enfin, ce livre est complémentaire d'une série d'émissions vidéos qui s'attachent à promouvoir la même approche.

   Le média audio-visuel est irremplaçable, lorsqu'il s'agit d'illustrer des métaphores et d'installer des images chez l'apprenant. Mais il ne permet guère de nuancer, de s'attarder. Son empreinte est fugace et son utilisation est, par nature, séquentielle; il permet de sculpter à grands traits, non de ciseler finement ...

   On trouvera ici tout ce qui n'a pu être exprimé et montré à travers les images de ces réalisations audio-visuelles[7]

---

6 Cf. par exemple [3], [8], [15], [36], [47], [51], ...

7 L'environnement "Image pour programmer" dont cet ouvrage fait partie comporte à la fois des émissions vidéos (6 x 1/2 h), 6 documents de présentation et d'exploitation et un logiciel d'apprentissage (START). Toutes information à ce propos peut être obtenue chez l'auteur à l'adresse suivante : CeFIS, Facultés N-D de la Paix, rue de Bruxelles, 61 B-5000 Namur (Tél. 081/ 72 50 60).

Je ne sais si mes arguments vous ont convaincu de la nécessité absolue de lire la suite. Mais si vous m'avez déjà accompagné jusqu'au bout de cet avant-propos, l'effort n'est pas bien grand de passer aux pages qui suivent !

Alors, on y va ...

# Avertissement en ce qui concerne les révisions

L'ouvrage publié en 1990 aux éditions De Boeck et dont vous venez de parcourir la préface et l'avant-propos est à présent épuisé. Le présent document en reprend, presque sans aucune modification, la teneur.

Dans l'avant propos de la 2ème partie, très récemment publiée au CeFIS[1], j'ai cependant mentionné un certain nombre de "regrets" dans une perspective de "*et si c'était à refaire*". Je vous renvoie à cette seconde partie pour les détails.

J'ai préféré reprendre ici l'intégralité du contenu de l'ouvrage originel, sans modifications autres que de détails ou de corrections. La seule concession consiste à accompagner l'expression des "marches à suivre", primitivement écrites seulement sous forme de graphes de Nassi-Schneidermann, par leur formulation sous forme de "pseudo-code" (ou langage de description d'algorithme). L'expression choisie est d'ailleurs particulièrement "légère" grâce à une utilisation de l'indentation qui évite la prolifération des FinDuSi, FinDuTantQue, etc..

Tous les passages qui se distinguent de l'original apparaissent en bleu (dans la version PDF du présent document); ils risquent bien de passer inaperçus dans la version papier (noir et blanc).

Les textes des programmes Pascal sont restés ceux de l'antique Turbo Pascal, version 3, sous MS-DOS. Les modifications nécessitées par les versions plus récentes (par exemple 1.5 sous Windows) sont en général signalées.

Enfin, les textes des programmes Pascal sont disponibles sur une disquette jointe; ce sont les textes légèrement modifiés pour "tourner" avec la version 1.5 de Turbo Pascal sous Windows. Les noms des programmes tels qu'ils figurent sur cette disquette réfèrent aux numéros de pages où on trouve les textes correspondants dans le présent fascicule.

[1] Voir http://www.det.fundp.ac.be/cefis/index.html et précisément http://www.det.fundp.ac.be/cefis/publications/charles/images2-5-78.pdf.

# Sommaire

# 1

# PROGRAMMER !

Si j'écris le mot "informatique", nul doute que vous pensiez d'abord "ordinateur". Si j'avais écrit "météorologie", les images associées auraient probablement été "temps, anticyclone, dépression, pluie, ..." et pour certains ... "thermomètre". Le mot "astronomie" aurait, lui, appelé "étoiles, planètes, lune, ..." et peut-être "télescope". Mais personne n'est prêt à dire que la météorologie est la science des thermomètres ou l'astronomie celle des télescopes.[1] Et bien, l'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est la science des télescopes ou la météorologie celle des thermomètres ! L'ordinateur est un outil essentiel de l'informatique, comme le télescope est un outil pour l'astronome, mais "faire de l'informatique", c'est bien plus (et bien plus passionnant) que connaître et manipuler un ordinateur.

Cet ouvrage montrera, je l'espère, que les problèmes sont ailleurs que dans la connaissance et la maîtrise d'un ordinateur : j'aborderai fort peu les aspects techniques et dirai bien peu de choses de l'intérieur de l'ordinateur. Pourtant, même s'il sera rejeté à la périphérie de nos préoccupations, il me faut dire un mot de ce qu'il est, ne serait-ce que pour préciser, parmi les multiples facettes qui pourraient être retenues, celle sur laquelle je placerai un éclairage particulier.

## 1. Un ordinateur, c'est quoi ?[2]

### *1.1 Une définition ?*

Je ne tomberai pas dans le piège consistant à proposer une définition précise ou exhaustive de ce qu'est un ordinateur; parmi la multitude des choses qu'on pourrait en dire, je me contente de souligner celles que, pour mon propos, je souhaite placer en avant :

> ***Un ordinateur, c'est une machine à traiter des informations, de manière formelle*, pour autant qu'on lui ait indiqué (dans le détail !) comment mener à bien ce traitement.**

S'il s'agissait là d'une définition, il resterait à définir "machine", "information", "traitement",... Il faudrait encore ajouter que cette machine est électrique, qu'elle est constituée de circuits électroniques, ... ce qui n'éclairerait en rien mon propos.

---

1 Cf. [6]

2 Le lecteur intéressé pourra consulter les chapitres 2 et 3 de "Initiation à l'informatique" à l'adresse http://www.det.fundp.ac.be/cefis/publications/charles/ini-5-51.pdf.

## *1.2 Traitement formel d'informations*

Cette "machine" va donc s'atteler à des tâches de "traitement d'informations". Pour préciser cette assertion, voici quelques exemples de tâches, que nous connaissons bien, pour y être parfois confrontés, et qui mettent en oeuvre un traitement formel d'informations :

- additionner deux nombres;
- calculer une moyenne;
- chercher le plus grand d'une série de nombres;
- ...
- trier un paquet de cartes numérotées et qui se présentent dans le désordre;
- transcrire en toutes lettres un nombre écrit "en chiffres";
- compter et annoncer les points au tennis;
- chercher le numéro de téléphone d'un abonné;
- ...
- conjuguer un verbe régulier du premier groupe à tous les temps de l'indicatif;
- lire un texte et fournir la fréquence des divers mots le composant;
- centrer un titre lors de la dactylographie d'un document;
- ...

Le premier ensemble de tâches concerne des traitements de nombres; et les manipulations de données numériques sont, par essence, formelles. Le processus d'addition de deux nombres n'a que faire du "sens" qui serait attaché aux quantités à additionner : 13 + 12 cela fait toujours 25 quelle que soit la signification associée à ces nombres, qu'il s'agisse de poids, de sommes d'argent ou d'âges. L'ordinateur, manipulateur formel d'informations,[3] va donc exceller lorsqu'il s'agira de traiter des données numériques. Les opérations qu'il effectuera (ou plutôt qu'on lui fera effectuer) seront d'ailleurs similaires à celles que nous réalisons à propos des nombres : les additionner, les soustraire, les comparer, ...

> Je ne dis pas qu'avant et après le traitement formel de nombres nous n'attachons pas un sens aux données traitées mais la manipulation elle-même est indépendante de cette signification ! Il est exclu d'avoir des états d'âme lorsqu'on additionne des nombres !

Le deuxième ensemble de tâches correspond lui aussi, même si les informations traitées ne sont plus numériques, à des traitements formels : trier, chercher un numéro de téléphone, classer, ... sont des actions qui ne font appel qu'à la forme des données manipulées. ANTOINE précèdera toujours BENOIT dans un classement alphabétique et c'est l'apparence externe de ces prénoms qui permet d'en décider et non le fait que BENOIT est notre ami et ANTOINE un parfait inconnu. Évidemment, si dans cette liste de "prénoms" nous trouvons "AAFFRR", nous hésiterons à le classer avant ANTOINE, même si la forme nous y oblige et cela parce que le prénom "AAFFRR" nous paraît insensé; mais si nous excluons ces considérations de "sens", notre classement ne s'appuiera de fait que sur la forme des prénoms proposés.

Le troisième ensemble de tâches nous est nettement moins familier. Il s'agit ici de manipulations formelles de texte. Et la plupart du temps, lorsque nous nous intéressons à du texte,

---

[3] L'ordinateur, même "multimédia", était, est et restera un **calculateur** électronique : il manipule des (représentations physiques de) **nombres**.

ce n'est pas la **forme** qui nous importe mais le **sens** que nous lui attachons. Ainsi, nous ne "lisons" (généralement) pas "Germinal" ou "Notre-Dame de Paris" pour le plaisir de pouvoir déclarer que l'un comporte 1.248.368 mots et l'autre 1.168.413 ! Nous n'apprécions pas la portée d'un essai ou la beauté d'un poème sur base du nombre de fois que le mot "et" s'y trouve présent !

Lorsqu'il s'agira de "texte", l'ordinateur, manipulateur formel, ne pourra se livrer qu'à des opérations portant sur les caractères qui le constituent, sans référence, à la signification véhiculée. Nous parlerons d'ailleurs toujours de "chaînes (ou successions) de caractères" et non de "mots".

On voit que dans les tâches énumérées ci-dessus, j'ai soigneusement évité des travaux comme "résumer un texte", "traduire un texte", "apprécier la poésie se dégageant d'un texte". Ce ne sont plus là ce que j'appelle des traitements "formels" (= ne s'attachant qu'à la forme du texte et pas à ce que nous nommons son "sens").

Il ne viendrait à l'idée de personne, pour résumer un texte, de se contenter de le réécrire en passant un mot sur deux (ce qui constituerait un traitement purement "formel"); pour résumer un texte, nous devons avoir saisi son "sens", avoir identifié les "idées" importantes, les "thèses" avancées, ... Et tous ces concepts renvoient à autre chose qu'à l'aspect "formel" du texte.

De même, pour traduire, il ne suffit pas de consulter un lexique où chaque mot français aurait son équivalent anglais. Une telle attitude "formaliste", qui consiste à remplacer (bêtement) un mot par son correspondant, conduit à des absurdités comme "He the door" pour "Il la porte"!

Une dernière anecdote fera mieux comprendre ce caractère "formaliste" des traitements effectués par l'ordinateur. Il existe aujourd'hui, associés à certains logiciels de traitement de texte, des "correcteurs" orthographiques qui, pour chaque mot du texte, vérifient que ce mot est bien présent dans une (énorme) liste de mots (reprenant la plus grosse partie des mots acceptables). Dans le cas où le mot "fôte" est repéré dans le texte, l'ordinateur détecte bien une erreur... mais propose comme liste de mots possibles "forte", "fonte" ou "foetus"; et face à "saurtir" suggère "sautoir" ou "saurait" ! Par contre, rien dans la phrase "il faut pou voir se rencontrer" n'éveille une "réaction" de sa part, puisque le mot "pou" est dans la liste des mots acceptables, comme aussi le verbe "voir"...

Retenons donc que

**L'ordinateur n'est capable que de traitements *formels* d'informations.**[4]

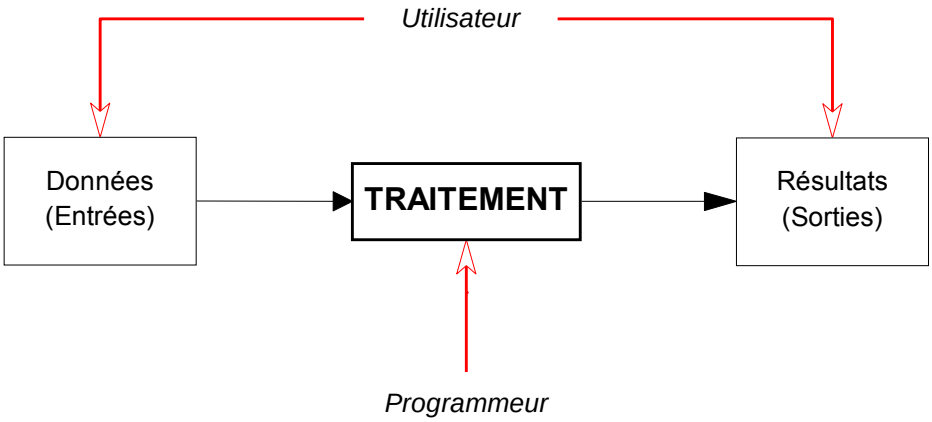
Nous pourrons déléguer à l'ordinateur l'exécution de ces tâches de traitement formel d'informations, à condition qu'il dispose de la "marche à suivre" qui le rende capable d'effectuer ces tâches. Et c'est là une facette essentielle : l'ordinateur sans programme (= marche à suivre explicative) pour le gouverner n'est capable de rien; c'est un principe fondamental :

**TOUT ce que fait un ordinateur, il le fait gouverné par une "marche à suivre", un programme.**

## *1.3 Les deux points de vue possibles*

Schématiquement, on peut représenter les choses de la manière suivante :

4 Les lecteurs intéressés par les débats "philosophiques" sur "forme" et "sens", sur "information" et "connaissance", sur "syntaxe" et "sémantique", consulteront avec profit l'excellent livre de Jacques ARSAC, Les machines à penser (Cf. [7]).




**L'utilisateur** voit dans l'ordinateur, **équipé des programmes convenables** qui le feront agir, un **outil** : outil pour composer, modifier et présenter du texte; outil pour calculer des trajectoires balistiques; outil pour gérer les opérations d'emprunt de livres dans une bibliothèque; outil pour dessiner; ...

L'objectif, c'est alors d'apprendre à se servir au mieux de l'outil adapté à une tâche donnée; la demande exprimée par l'utilisateur potentiel, c'est "apprenez-moi le mode d'emploi de tel outil". Ce qui l'intéresse, c'est essentiellement : "quelles données dois-je fournir ?" et "quels résultats suis-je alors en droit d'attendre après le traitement qui y sera appliqué ?". Pour le reste, l'ordinateur équipé du programme (logiciel) qui le gouverne pendant l'utilisation, c'est une boîte noire.

> Il est important de noter que ce que j'appelle "outil" c'est le tandem ordinateur-programme. Un ordinateur "nu" n'est capable de rien (Cf. le principe énoncé ci-dessus). L'outil, c'est l'ordinateur gouverné par le logiciel de traitement de texte ou l'ordinateur conduit par le logiciel de gestion de bibliothèque, l'ordinateur équipé du programme de calcul, etc..

En résumé,

pour l'utilisateur : ordinateur + programme } = **outil**

Le **programmeur** (ou plutôt l'analyste-programmeur) s'intéresse d'abord au traitement proprement dit. Pour lui, l'ordinateur est d'abord un **exécutant** à qui il va devoir fournir la **marche à suivre** (programme) "expliquant" le traitement à effectuer. L'accent est alors très nettement placé sur la tâche de traitement d'informations qu'on souhaite **faire faire** par l'ordinateur. La programmation, c'est essentiellement la création et la conception des outils (logiciels) qui seront utilisés par d'autres.

L'ordinateur cesse ici d'être une boîte noire ! Ce qui ne signifie nullement qu'on se passionne pour la "quincaillerie" (hardware) électronique qui le constitue : on y voit, non un ensemble de circuits intégrés ou de portes logiques, mais à travers le langage de programmation qui va le transfigurer (Cf. plus loin), un **exécutant** capable d'un certain nombre de traitements élémentaires et formels d'informations.

Ces deux perspectives débouchent évidemment sur des éclairages fort différents de l'ordinateur : outil (lorsqu'il est équipé du programme adéquat) pour l'utilisateur; exécutant, pour qui il faut décortiquer une tâche afin de lui indiquer le traitement à effectuer, pour le programmeur.

Autrement dit, on peut soit s'intéresser à ce dont l'ordinateur paraît capable (quand il est équipé du logiciel ad hoc), soit au problème de le rendre capable de ces activités en concevant pour lui les programmes lui "expliquant" le traitement à exécuter.

De ces deux approches possibles du monde des ordinateurs, c'est la seconde qui sera retenue et développée dans ces pages : qu'est-ce que "programmer", quelles sont les règles qui gouvernent la conception des "marches à suivre" qui nourriront l'ordinateur, comment "décortiquer" une tâche pour la faire exécuter ? ...

## 2. Programmer ?

### *2.1 Tâche ou problème?*

Aux exemples de traitements (formels) d'informations signalés plus haut, on pourrait en ajouter bien d'autres comme :

- être capable de repérer dans une série de nombres quel est le plus petit et le plus grand;
- lire un texte et signaler ensuite combien de fois le mot "et" s'y trouve;
- jeter un dé jusqu'à ce que le 6 apparaisse pour la troisième fois et annoncer alors combien de fois il a fallu jeter le dé pour cela;
- écrire un nombre en chiffres romains;
- décomposer une somme d'argent en coupures (en minimisant le nombre de celles-ci);
- ... ...

Je pourrais, et de beaucoup, allonger cette liste. Mais, à chaque fois, les activités évoquées constituent des **tâches**, souvent fastidieuses, mais en tout cas d'une simplicité remarquable. Je n'ai jamais rencontré personne qui, face au travail consistant à trier 20 cartes fournies dans le désordre, ait demandé un long délai de réflexion ou s'en soit déclaré incapable. Que dire alors de l'activité consistant à additionner deux nombres par calcul écrit ou à compter les mots d'un texte !

Tous ces exemples décrivent des tâches, souvent élémentaires, parfois franchement risibles, tant elles paraissent dénuées de créativité et de la moindre réflexion. En tout cas, personne n'est prêt à qualifier de **problèmes** ces travaux rudimentaires. Le mot "problème" charrie avec lui une aura de difficulté, d'invention, de recherche que ne méritent absolument pas les tâches évoquées ci-dessus.

On ne "résout" pas le "problème" de lancer un dé jusqu'au moment où l'on obtiendra trois 6 de suite; on n' "analyse" pas longuement le "problème" consistant à repérer si un nom est oui ou non présent dans une liste triée alphabétiquement.

> On m'objectera sans doute que s'il s'agissait de 20000 cartes à trier ou du décompte des mots constituant les oeuvres complètes d'Emile Zola, on se trouverait en face de vrais problèmes.
>
> Qu'il soit malaisé de trouver quelqu'un prêt à s'atteler à des tâches d'aussi longue haleine et tellement fastidieuses, c'est vrai ! Mais ce n'est pas la complexité des tâches évoquées qui est en cause. Compter, par exemple, est une activité débile; compter beaucoup reste tout aussi idiot, mais s'avère en plus lassant, éreintant et accablant, ... mais cela ne devient pas pour autant un "problème". Si ces travaux étaient bien payés, nul doute que les candidats s'en déclarant capables ne manqueraient pas. Combien en resterait-il s'il s'agissait de fournir la solution à de petits **problèmes** d'arithmétique ?

**Il est faux de dire qu'en programmation on "résout des *problèmes*" ou même que, pour programmer, il faut d'abord bien "analyser le *problème* posé". On s'intéresse le plus souvent à des *tâches* (souvent assez bêtes et fastidieuses).**

**On ne "résout" pas une tâche, on l'effectue !**

Il y a 40 ans que l'informatique existe et, au risque de paraître provocateur, je dirais volontiers que

**L'informatique n'a (presque) jamais "*résolu*" un seul "*problème*".**

Je pratique l'informatique et la programmation depuis suffisamment longtemps pour mesurer (tout de même) ce que ces propos peuvent avoir d'outrancier et d'exagéré. Je peux évidemment apporter des exemples de tâches tellement longues et fastidieuses qu'il serait bien difficile de trouver un être humain acceptant de s'y atteler (tâches de calcul, de tri de grosses quantités d'informations, ...)(Cf. ci-dessus). Mais l'ordinateur ne **résout** pas ces tâches. Il les **effectue** vite et bien, c'est tout.

Par ailleurs, il est évident que, lorsque la tâche consiste à "gagner aux échecs" ou même simplement "gagner au tic tac toe", nous sommes prêts alors à parler de "problème" : la plupart d'entre nous sont d'ailleurs incapables de "résoudre ces problèmes" (= de gagner à coup sûr). Que dire évidemment dans ce cas du "problème" de "faire gagner l'ordinateur aux échecs" !!!!

Si j'insiste tellement ici sur cette distinction entre tâche et problème, c'est que j'ai encore en mémoire les yeux étonnés et le regard incrédule de mes premiers "élèves" et de mes collègues (non-informaticiens), à qui je parlais du "**problème** (sic) de conjuguer à l'indicatif présent un verbe régulier du premier groupe" ou du "**problème**, plus compliqué, (resic) de compter tous les mots comportant une lettre redoublée dans un texte".

### *2.2 Programmer, c'est faire faire ...*

Et pourtant, chacune des **tâches** évoquées ci-dessus va tout de même donner lieu à un réel **problème** lorsque la programmation va s'en mêler.

En effet, il s'agira à chaque fois pour le "programmeur", non d'exécuter ces tâches lui-même (ce qui le plus souvent n'aurait aucun intérêt et serait affreusement fastidieux), mais de les **faire exécuter**. C'est cela, **programmer** ! Il s'agit, non d'être capable de mener à bien soi-même une tâche donnée, mais **d'expliquer** à "un autre" comment il doit s'y prendre pour exécuter cette tâche à notre place. En quelque sorte,

**PROGRAMMER,**
**c'est**
**FAIRE FAIRE**

Chacune des **tâches** décrites pose un véritable **problème**, dès qu'il s'agit de la **faire faire** par un autre. Évidemment, la difficulté dépend de manière cruciale des possibilités et des caractéristiques de cet "autre".

## Première difficulté : l'exécutant a des capacités limitées;

S'il s'agissait d'un être humain, lui faire exécuter les diverses tâches évoquées, exigerait seulement qu'on lui dise

- repère dans la série de nombres qu'on va te fournir quel est le plus petit et le plus grand;

- lis le texte et signale ensuite combien de fois le mot "et" s'y trouve;
- jette un dé jusqu'à ce que le 6 apparaisse pour la troisième fois et annonce alors combien de fois tu as dû jeter le dé pour cela;
- écris en chiffres romains les nombres que je te fournirai;
- ...

Exactement comme il suffit de préciser "confectionne une blanquette de veau" pour obtenir le résultat souhaité, **lorsqu'on s'adresse à un cuisinier confirmé**.

Si, par contre, on est au prise avec un débutant (ou une débutante), cette petite phrase ne suffit plus : il est alors indispensable de fournir la **recette** correspondante. Et le détail des **explications** qui doivent être données dépend de manière cruciale des possibilités du cuisinier débutant.

L'ordinateur, s'il est bien un traiteur d'informations, peut malheureusement, **quand on le regarde du point de vue de la programmation**, être assimilé à un "débutant". Ses possibilités, on va le voir au chapitre 3, sont extraordinairement limitées. Et il est absolument hors de question qu'il "comprenne" des phrases comme "Trie les nombres qu'on va te fournir" ou "Conjugue au présent de l'indicatif un verbe qui sera précisé"..., même si on les lui dit en anglais ! S'il fallait dès à présent qualifier l'exécutant-ordinateur, les mots qui me viendraient tout naturellement en tête seraient plutôt "borné", "bête", "ne comprenant jamais rien à demi-mot", ...

> Je ne suis évidemment pas dupe de ces anthropomorphismes à propos de l'ordinateur. Si nous le découvrons "bête" ou "borné" quand nous nous intéressons à "lui" à travers la programmation, nous serions par ailleurs tenté de le qualifier d' "efficace" et parfois même d' "intelligent" quand nous posons sur lui un regard d'utilisateur. C'est que, dans ce dernier cas, il est évidemment transfiguré par les indications (le programme) qui lui précisent comment traiter aussi "intelligemment" les tâches considérées.
>
> L'une et l'autre de ces attitudes sont sans fondement réel : il n'y a guère plus de sens de parler d'un ordinateur "borné" qu'à prétendre que ma tondeuse à gazon est "découragée" et il est aussi inadéquat de le qualifier d' "intelligent" que de parler d'une lessiveuse "capricieuse" ou d'une bêche "courageuse"...

**Le problème en programmation ce sera donc, face, d'une part à une tâche (même nous apparaissant comme simple), et face, d'autre part, à un exécutant aux capacités limitées (l'ordinateur), d'expliquer à ce dernier comment effectuer cette tâche à notre place.**

Le problème ne réside pas (en général) dans la tâche qui sert de point de départ, mais dans le fait que nous devrons la faire effectuer par un exécutant qui, initialement, n'en est pas capable. La difficulté est donc de lui fournir les indications nécessaires pour que, s'y conformant, il paraisse capable de mener cette tâche à bien. **Programmer** est donc un verbe à rapprocher plutôt d'**expliquer**, de **décortiquer**, de **déplier** (déplier = ex-pliquer).

## Deuxième difficulté : on "fait faire" en différé

Programmer, c'est "faire faire". Ce "faire faire" n'est cependant pas un "faire faire" en direct. On ne nous permettra malheureusement pas d'être aux côtés du cuisinier débutant, pendant sa prestation, pour lui fournir les indications nécessaires. Nous pourrions dans ce cas réagir à ses actions, ajuster nos explications, rectifier le déroulement de l'exécution,...

Ce qui sera permis, c'est seulement de fournir la **recette** à l'apprenti cuisinier; nous serons ensuite forcés de le laisser seul aux prises avec la confection de la blanquette de veau, sans aucune

possibilité de modifier des instructions inadéquates ou insuffisantes, sans pouvoir intervenir dans l'exécution. **TOUT** doit être prévu dans la recette conçue, rien ne doit être laissé à l'appréciation de l'exécutant. Il n'est capable d'aucune initiative : tout au plus, dans le cas d'une recette incorrecte, son activité s'interrompra ... ou débouchera sur un plat immangeable et qui n'aura plus que de lointains rapports avec la blanquette de veau qu'on souhaitait initialement faire confectionner.

Lorsqu'il s'agit d'indiquer à quelqu'un comment il peut parvenir en voiture jusque chez moi, cela ne pose aucun problème lorsque je suis assis à côté du conducteur pour lui donner "en direct" toutes les instructions nécessaires. Mais, s'il s'agit de lui "expliquer" **à l'avance** comment se rendre **seul** chez moi, ceci nécessite alors beaucoup plus de soin et de rigueur : fournir la "marche à suivre" qui lui permettra d'arriver à coup sûr est bien compliqué, surtout s'il est borné au point de ne pas savoir lire une carte et m'oblige ainsi à transcrire par écrit toutes les indications indispensables...

> **Programmer, c'est donc rédiger une *marche à suivre* pour faire faire une tâche par un exécutant aux capacités limitées**.
>
> > Je préfère le terme "marche à suivre" à celui plus classique d'"algorithme". D'abord, la coloration de ce dernier est fort "mathématique" : la plupart des gens pensent - à tort - que le mot "algorithme" entretient d'étroits rapports avec "logarithme". Ensuite, il laisse davantage dans l'ombre, me semble-t-il, la nécessité d'avoir précisé l'exécutant.
> >
> > On pourrait aussi parler d'ensemble de "consignes", de série d'indications ou d'instructions. De toute manière, ce concept de marche à suivre sera bientôt affiné et plus complètement abordé puisque le chapitre 2 tout entier y est consacré.

Le problème, c'est donc d'être capable de morceler une tâche "complexe" (si on la compare aux possibilités de l'exécutant-ordinateur) en une succession de tâches plus élémentaires (= dont l'exécutant est capable). Il faudra organiser la succession de ces dernières pour que, les effectuant dans l'ordre indiqué, l'exécutant ait finalement accompli la tâche complexe initialement proposée.

Il nous arrive parfois **d'exécuter** des marches à suivre lorsque, par exemple, nous suivons les indications d'une recette de cuisine ou d'un guide de tricot. C'est aussi le cas de l'exécutant-musicien face à une partition-marche à suivre. Chacun mesure cependant, dans ce cas, toute la distance entre ce statut d'exécutant et celui du compositeur-programmeur, dont le rôle est de concevoir et d'écrire les partitions-marches à suivre. Ceci pour insister sur le fait que cette activité (rédiger une marche à suivre) est totalement nouvelle et inconnue pour la plupart d'entre nous (sauf pour les auteurs de guides de tricot ou de livres de recettes !).

## Troisième difficulté : l'exécutant est un "robot";

Tous les exemples mentionnés ci-dessus sont en partie trompeurs : c'est qu'il s'agit à chaque fois de fournir les instructions nécessaires à un **être humain**, qu'il soit apprenti cuisinier, musicien débutant ou tricoteur (tricoteuse ?) novice. Même si la marche à suivre à concevoir doit être complète (en envisageant toutes les situations possibles) et non ambiguë (en ne laissant aucune initiative à l'exécutant), notre interlocuteur est un être humain, comme vous et moi.

Dès lors, nous pourrons, dans certaines limites, nous exprimer à demi-mots, garder certains implicites (partagés par tous les humains), adopter une représentation imagée ou dessinée, ... Tout ceci sera évidemment exclu lorsque l'exécutant des marches à suivre à concevoir sera un "robot". Dans ce cas, les exigences de précision, de chasse aux implicites, d'exhaustivité seront pratiquement "inhumaines".

> Il est un point cependant où le caractère "borné" de l'exécutant-robot est intéressant et même indispensable. On comprend aisément que pour qu'un problème de programmation soit correctement posé, il faut disposer non seulement de la description précise de la tâche à faire effectuer mais aussi de celle de l'exécutant à qui on la destine. C'est dire qu'il faut connaître

complètement et précisément **tout** ce dont ce dernier est capable : c'est possible pour un robot aux actions limitées, mais non pour un être humain ! Ainsi, concevoir une marche à suivre destinée à un humain est un exercice périlleux, puisque, ne disposant pas d'un portrait complet de ses possibilités, nous ne saurons jamais avec certitude, a priori, si nos consignes sont insuffisantes ou inutilement détaillées.

Je voudrais revenir et insister sur le fait que pour qu'un problème soit bien posé en programmation, il faut non seulement que la tâche qu'il va falloir faire faire soit complètement précisée, mais encore que l'exécutant à qui je devrai fournir les consignes explicatives soit parfaitement défini.

On propose pourtant fréquemment aux débutants des énoncés du type

"analyser le problème de la mise en marche d'une voiture"

ou

"écrire l'algorithme pour la préparation du café"

ou

"résoudre le problème consistant à traverser une rue sans se faire écraser par les voitures".

On y désigne comme des "problèmes" des activités bien familières. On me rétorquera qu'il s'agit de décrire la manière dont on s'y prend pour effectuer ces divers travaux. Jusqu'où alors pousser les explications et le décorticage ? En réalité, ce qui est souhaité, c'est qu'on décrive les algorithmes (ou marches à suivre) qui sous-tendent chacune de ces tâches. Mais, dans ce cas, les énoncés proposés sont notoirement incomplets puisqu'on n'a pas précisé, à chaque fois, les capacités de l'exécutant à qui on s'adressera. Que comprend et que peut faire l'apprenti conducteur ou l'exécutant-piéton ? Sans précision supplémentaire, il est impossible de savoir jusqu'où ces diverses tâches doivent être disséquées.

On devine déjà qu'avant d'aborder l'analyse du moindre problème de programmation, il faudra avoir dit quelles sont les caractéristiques de l'exécutant-ordinateur, sinon la démarche toute entière est viciée : **il est impossible de faire faire quelque chose par un autre, sans savoir ce dont il est capable** !

## 3. Les étapes du faire faire

Nous savons à présent que l'essentiel de ces notes a pour but de nous apprendre comment faire faire une tâche (de traitement formel d'informations) par "un autre". La tâche à faire réaliser est généralement anodine. Le problème vient de ce que celui à qui nous souhaitons faire exécuter ce travail n'en est pas directement capable. Il nous faut décortiquer, morceler la besogne en une succession d'actions plus élémentaires correspondant aux capacités de l'exécutant. Ce décorticage, ces explications s'incarnent dans une marche à suivre qui guidera son activité lorsque nous exigerons qu'il effectue cette tâche.

Schématiquement, le cœur du processus se décrit comme :


Sur le chemin qui conduira de la tâche à son exécution par l'ordinateur, plusieurs étapes sont indispensables. Et d'abord, il faudra répondre à la question suivante :

### *3.1 Quoi faire ?*

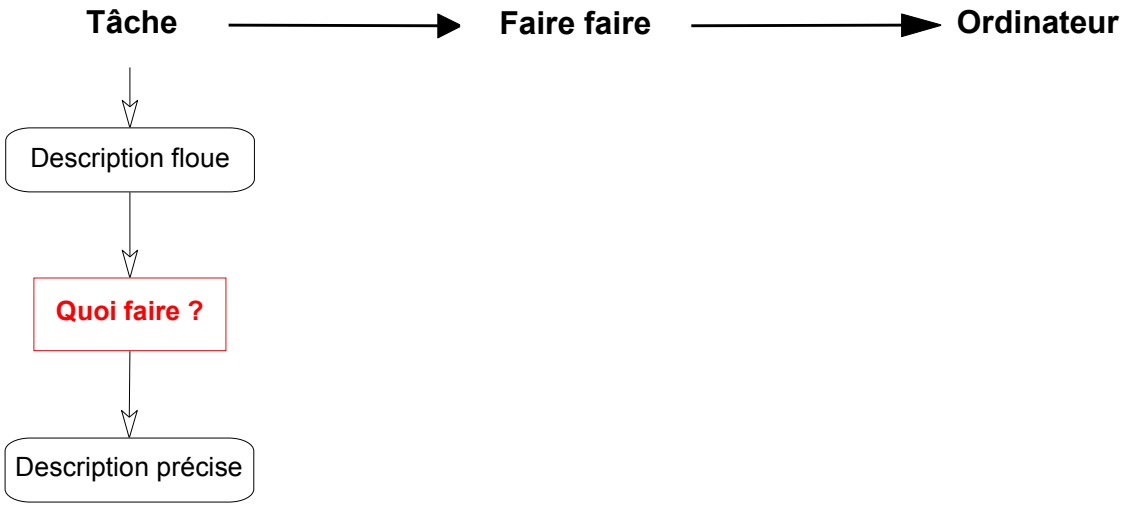
En effet, la tâche se présentera généralement d'abord de manière très floue. Il sera donc primordial, dans un premier temps, d'affiner sa description. Ainsi, face à la besogne consistant à écrire un nombre en toutes lettres, une série de questions viendront préciser ce qui doit être fait :

- De quelle taille seront les nombres à écrire ? Plus petits que cent ? Inférieurs à un million ? A un milliard ?
- Les nombres comporteront-ils une partie décimale ? Dans ce cas, comment exprimer cette dernière ?
- Faut-il les écrire "à la belge" (septante-huit) ou "à la française" (soixante-dix-huit) ?
- Faut-il les écrire en majuscules ? En minuscules ?
- ...

On le voit, même pour une tâche simple, il est indispensable de préciser ce qui est attendu, ce à quoi on s'engage.

**Il est complètement illusoire de croire qu'on va pouvoir faire faire quelque chose par un autre si l'on ne sait pas précisément ce qu'il faut faire (faire) !!!!!**

Cette étape du "Quoi faire ?" est en quelque sorte celle du "cahier des charges", celle de la recherche des "spécifications" (pour parler comme les informaticiens). C'est celle qui nous fera passer de la **description floue** de la tâche à sa **description précise**.




1. On passe très souvent sous silence cette première étape dans les initiations à la programmation. C'est fort dommage et cela donne, en tout cas, une idée complètement tronquée de ce que recouvre la démarche informatique d'analyse (et de programmation). Les tâches qui seront abordées ici resteront évidemment de petite taille et d'une complexité fort raisonnable. Je ne fournirai pas de grands principes méthodologiques permettant de guider la démarche sous-jacente à cette étape : un jeu de questions et réponses nous tiendra lieu de méthode. Comme il s'agira à chaque fois de tâches consistant à traiter des informations, nous orienterons les questions dans trois directions :
   - les données nécessaires au traitement envisagé (entrées) (leur forme, leur nombre, ...);
   - les résultats souhaités (sorties) (forme, présentation, ...);

- le traitement (en quoi les résultats sont liés aux données).

Mais, on le verra, cette étape de précision de ce qui est attendu sera toujours la première et nous n'en ferons jamais l'économie.

2. Lorsque quelque chose "ne tourne pas rond" dans l'utilisation de l'ordinateur, par exemple dans la gestion d'une entreprise, ce n'est (presque) jamais à cause de l'ordinateur lui-même (rien n'est moins "capricieux" qu'un ordinateur). C'est parfois parce que les programmes sont incorrects (ils ne font pas faire par l'ordinateur ce qu'on souhaite ou tentent de lui faire faire des choses impossibles). Mais, le plus souvent, c'est parce qu'on n'a pas consacré le temps nécessaire à préciser ce qui était attendu. On se retrouve en bout de course avec un produit qui ne donne pas satisfaction essentiellement parce que, tout bêtement, **"on" n'a pas dit TOUT ce que l'on voulait**.

3. Dans la programmation telle qu'elle est vécue par les amateurs (ou les apprenants) la tâche abordée et la connaissance des possibilités offertes (par l'exécutant) pour la traiter sont "dans la même tête". Les questions sont posées par celui qui connaît aussi les réponses. Dans la réalité professionnelle, l'informaticien (analyste) débarque généralement dans un milieu qu'il ne connaît pas et qu'il doit "informatiser". Il n'y a pas grand chose de commun entre la tâche consistant à gérer le stock d'un magasin de chaussures et celle de la gestion des emprunts dans une bibliothèque ou de l'organisation d'un cabinet de dentiste ! Et celui qui "sait" ce que recouvre précisément chacune de ces tâches, c'est le client **qui ne sait pas qu'il sait** : il est généralement incapable d'expliciter comment il s'y prend actuellement et encore moins de décrire ce qu'il souhaite (il "fait" mais ne sait pas "comment il fait"). Et en face de lui, il y a l'informaticien qui, lui, connaît les possibilités de l'ordinateur, mais ne sait pas ce qu'il doit en faire dans ce cas précis. En résumé, il y a dans une tête (celle du client) les réponses, mais il ne sait pas qu'elles y sont; et dans une autre tête (celle de l'informaticien) il y a des questions, mais il ne sait pas bien celles qu'il est pertinent de poser.

   **Le bon informaticien n'est pas celui qui commence par apporter des réponses mais celui qui pose d'abord les bonnes questions !**

4. Ce n'est pas un hasard si la formation des informaticiens comporte une grosse partie consacrée à l'acquisition de méthodologies d'exploration du "Quoi faire ?" pour de grandes catégories de tâches : gestion, conception de bases de données, .... Il s'agit là de la première étape, souvent la plus ardue, du travail de l'informaticien, sur le chemin qui conduit de la tâche à son traitement par l'ordinateur.

5. Les questions posées viendront de deux horizons :
   - Certaines seront motivées par la tâche elle-même dont il faut affiner la description, indépendamment du fait qu'il faille ou non la faire traiter par un ordinateur. Ce sont les questions que nous poserions de toute manière si l'on nous demandait d'effectuer la tâche nous-mêmes.
   - D'autres précisions sont exigées parce qu'il s'agira de faire faire les choses par l'ordinateur. Ainsi, dans l'exemple de l'écriture des nombres en toutes lettres, il est impératif de savoir, dans le cas où les nombres à écrire comportent une partie décimale, si cette dernière sera annoncée par une virgule (comme en français) ou par un point (comme les anglo-saxons). Ce qui n'est qu'un détail futile, auquel nous ne penserions même pas s'il s'agissait d'écrire les nombres nous-mêmes, devient important parce qu'il va s'agir de les faire écrire (l'exécutant-ordinateur préférant souvent alors le point à la virgule). Cette deuxième catégorie de questions est bien entendu la plus difficile à

découvrir pour le débutant puisqu'elle postule qu'on connaisse bien les caractéristiques de l'exécutant-ordinateur qui va les motiver.

A l'issue de cette première étape, le cahier des charges est établi : nous savons alors, avec précision, ce qu'il faut faire (faire). L'étape suivante est évidemment celle du

## *3.2 Comment faire ?*

Il s'agit ici de mettre au jour les stratégies employées : comment, face à la tâche précisée à l'étape précédente, nous y prenons-nous ? Il y a malheureusement un monde entre "être capable de..." et "pouvoir expliquer comment on s'y prend pour ...".

> **"Solving a problem gives you the solution. Knowing how you solved the problem gives you a program."**
>
> M. JAMES dans [43]

Cette petite phrase illustre bien toute la distance existant entre la capacité d'effectuer une tâche et la conscience qu'on a des **stratégies** mises en oeuvre pour y parvenir. Si je vous demande **d'expliquer** comment vous vous y prenez pour trier un (petit) paquet de cartes dans le désordre, il est inutile que vous recommenciez à nouveau l'opération sous prétexte que j'ai mal vu. Je sais fort bien que vous pouvez trier ce paquet de cartes et il ne sert à rien de le faire et le refaire : ce que j'attends c'est que vous expliquiez complètement et dans le détail **comment** vous procédez, que vous mettiez au jour les **stratégies** employées. C'est ici qu'il faut être capable de "faire le petit pas en arrière" qui permet de se "regarder faire"; c'est ici qu'il faut, en quelque sorte, quitter son rôle d'"acteur" pour adopter celui de "metteur en scène".

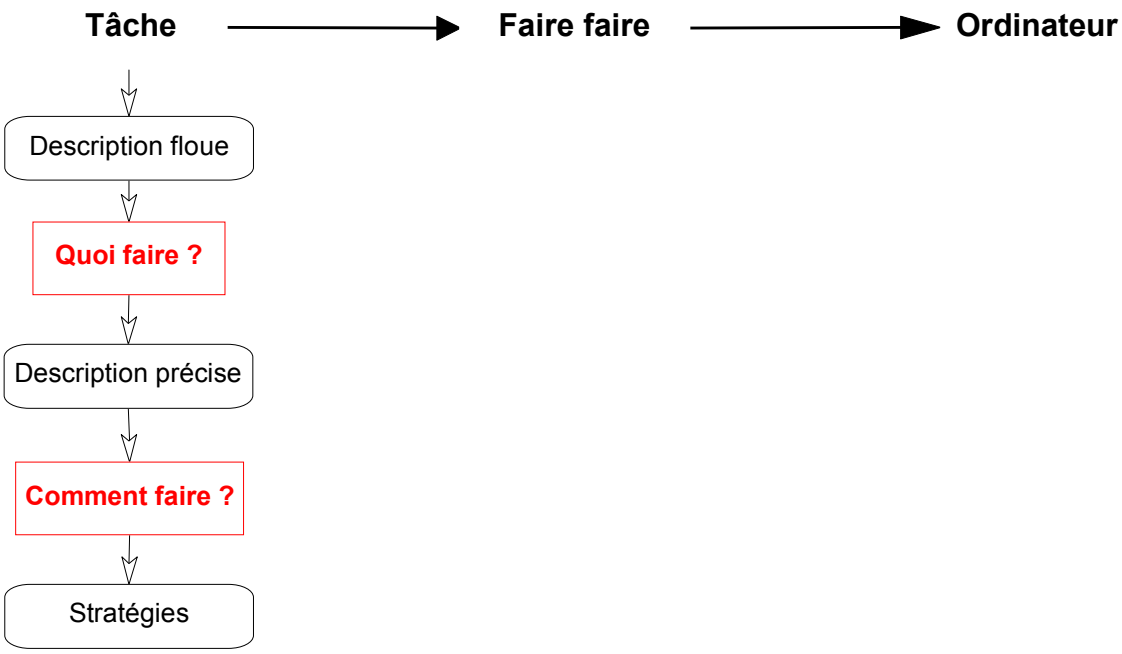
1. Même pour des tâches anodines (comme procéder à l'addition écrite de deux nombres ou compter les points au tennis) il n'est pas simple d'indiquer les stratégies employées. Et les tâches pour lesquelles nous avons le plus de mal à mettre au jour ces stratégies sont probablement celles que nous effectuons le plus automatiquement ou le plus inconsciemment : nous avons à ce point intégré les comportements sous-jacents que la question "comment fais-tu ?" nous prend au dépourvu et nous paraît en tout cas bien incongrue.
2. Il est des tâches un peu plus compliquées, comme écrire un nombre en toutes lettres ou décider si un nombre est premier, qui nous poseront probablement déjà de petits problèmes même s'il s'agit seulement de les effectuer nous-mêmes sans être tenus d'expliquer en détail comment nous nous y prenons. Dans ce cas cependant, les livres que nous consulterons ne sont absolument pas les traités d'informatique. C'est dans une grammaire que nous trouverons les règles permettant d'écrire sans erreur un nombre en toutes lettres et dans un livre d'arithmétique que nous apprendrons (ou réapprendrons) ce qu'est un nombre premier et comment on peut vérifier cette caractéristique. Ainsi, face à une tâche qui nous pose un problème, les stratégies sont à chercher dans le champ (domaine) de cette tâche, pas dans les traités d'informatique !
3. L'attitude clé ici, c'est d'être capable de **passer de l'implicite à l'explicite**.

   C'est un fait que, trop souvent, nous avons appris à effectuer des tâches ou à résoudre des problèmes mécaniquement, sans avoir mis au jour, **explicitement**, comment nous nous y prenons, sans les avoir complètement **dépliés** (déplié = ex-pliqué). La programmation oblige à cette **prise de distance** où l'on se voit aux prises avec la tâche et où l'on prend conscience de sa stratégie. Pour "faire faire", il faut d'abord **mettre à plat** son savoir faire.

C'est un apprentissage extrêmement exigeant et difficile que celui de cette chasse sans pitié aux implicites, aux flous, aux "à peu près"...

Le chemin qui conduit de la tâche à son exécution par l'ordinateur s'enrichit donc d'une étape supplémentaire :



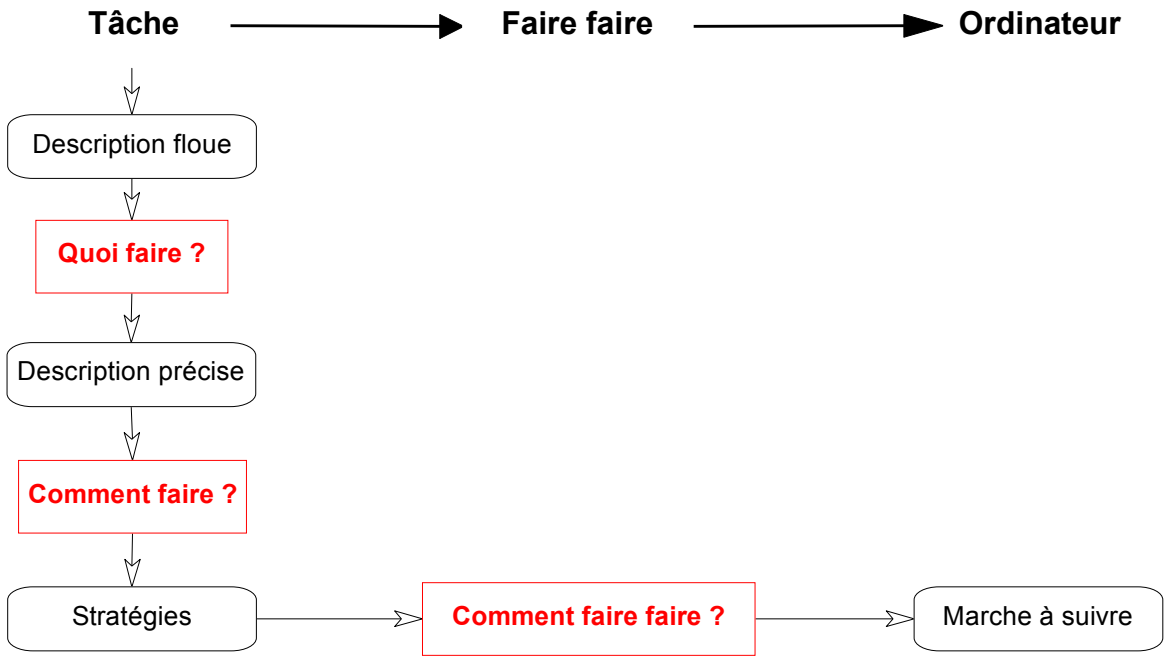

## *3.3 Comment faire faire ?*

Voici l'étape cruciale et absolument neuve : c'est celle où, sur base de la description précise de la tâche et s'appuyant sur les stratégies de traitement mises au jour, nous allons **rédiger la marche à suivre** destinée à l'exécutant ordinateur.

1. C'est bien entendu ici que la connaissance des capacités de cet exécutant devient indispensable : c'est seulement en sachant ce qu'il peut "comprendre" et faire que la rédaction d'une marche à suivre prend un sens. On verra que si le chapitre 2 précise les règles de conception et d'écriture de toute marche à suivre, le chapitre 3 répond à la question "Qui est l'exécutant-ordinateur ?". C'est seulement après avoir tracé son portrait que nous pourrons rédiger les premières marches à suivre qui vont le gouverner.
2. C'est évidemment toujours "en français" qu'à ce stade les marches à suivre seront conçues et exprimées. Bien entendu, on le verra, elles n'utiliseront pas toute la richesse d'expression permise par notre langue : nous y mettrons en évidence certains mots et en éviterons d'autres; nous nous aiderons de graphismes bien choisis pour en montrer la structure;... Mais toujours, nous veillerons à déboucher sur une expression aussi compréhensible et claire que possible pour l'être humain qui serait amené à en prendre connaissance. Ainsi donc, nous serons attentifs, non seulement à nous soumettre aux capacités de l'exécutant à commander, mais aussi à tenir compte de ceux qui voudront relire et comprendre la marche à suivre rédigée.
3. C'est au cours de cette étape que nous nous aiderons d'une méthode particulière : **l'approche descendante et structurée**. Il est trop tôt pour détailler le contenu de cette manière de

procéder[5]; j'ajouterai simplement que l'informatique a redécouvert là une méthode aussi vieille que l'humanité elle-même : face à un problème, on découpe celui-ci en sous-problèmes, plus aisés à traiter, puis ces derniers sont à leur tour morcelés, et ainsi de suite,... Cette attitude qui partira de la tâche dans son intégralité et sa complexité, pour descendre pas à pas, par affinements successifs, vers les actions élémentaires que nous pouvons exiger de l'exécutant, s'incarnera et se formalisera ici tout au long de l'apprentissage proposé.

Le schéma se complète donc comme suit :




## *3.4 Comment dire ?*

Nous disposons à présent, sous l'une ou l'autre forme (Cf. Chapitre 2), de la marche à suivre suffisante pour que, s'y conformant, l'exécutant-ordinateur effectue la tâche décrite.

Malheureusement, l'attention que nous aurons portée à rendre cette marche à suivre aisément lisible par d'autres, aura généralement conduit à l'écrire (ou même à la dessiner) sous une forme qui ne convient pas à l'ordinateur. Il va à présent falloir l'exprimer dans un langage particulier, "compréhensible" par l'ordinateur, dans un **langage de programmation** .

1. Ces langages de programmation sont nombreux : vous connaissez sans doute Basic ou Pascal, Logo, Fortran, Cobol, ... Il en existe des dizaines, sinon des centaines ! Dans chaque cas, il s'agit bien d'un langage, construit de toutes pièces, et non d'une langue, comme celles à travers lesquelles les humains communiquent entre eux. Le vocabulaire de ces langages est (très) pauvre; les règles de grammaire qui les gouvernent sont arbitraires et (heureusement) peu nombreuses. Mais il nous faudra impérativement les respecter si nous souhaitons être "compris" par l'exécutant-ordinateur.
2. Inutile de souligner que ces divers langages ne sont pas les nôtres ! Aucun n'est d'ailleurs non plus vraiment celui de l'ordinateur ! On peut donc parler de **langage compromis**,

[5] On pourra consulter le chapitre 2 ("Diviser pour régner") du second tome de "Images pour programmer".

puisque, ayant choisi un langage de programmation, je ferai l'effort d'y exprimer mes marches à suivre et que, par ailleurs, l'ordinateur acceptera que je m'adresse à lui dans ce langage qui n'est pas vraiment le sien.

C'est dire qu'en tout cas, du point de vue de l'ordinateur, une traduction restera indispensable. Mais, heureusement, c'est lui qui s'en chargera. Ainsi, chacun "y met du sien" : j'accepte d'exprimer mes marches à suivre dans un langage qui n'est pas le mien et l'ordinateur, dont ce n'est pas non plus le langage "naturel", accepte de se charger de la traduction, pour déboucher sur une version qui, enfin, sera "la sienne" et qu'il pourra exécuter.

> Mon objectif n'est évidemment pas ici d'entrer dans des détails "techniques" à propos de ces problèmes. Il faut cependant savoir que le seul langage compréhensible par l'ordinateur (ou plus précisément par le processeur qui en est le "cœur" ou le "cerveau"[6] est le langage machine et non l'un quelconque des langages de programmation cités ci-dessus; c'est pour cela qu'une traduction est indispensable. J'ajouterai simplement que, comme toujours, si l'ordinateur paraît capable de traduire un texte qui lui est fourni dans l'un ou l'autre langage de programmation, c'est qu'il est alors gouverné par un programme qui lui "explique" comment effectuer cette traduction.
>
> Ainsi, dès que je dispose, pour un ordinateur donné, du programme qui va lui faire faire la traduction de tel langage vers le sien, je peux m'exprimer dans ce langage. L'impression que donne parfois l'ordinateur d'être "polyglotte", puisque la même machine va accepter des programmes rédigés en Basic, en Pascal, en Fortran, ... cache en réalité l'existence, pour chacun de ces langages, du programme de traduction correspondant.
>
> Ces programmes particuliers, capables de faire effectuer par l'ordinateur ces traductions s'appellent des **compilateurs** ou des **interpréteurs**.

3. Pourquoi ne pas avoir directement exprimé la marche à suivre dans ce langage compromis ? Les deux étapes du "Comment faire faire ?" et du "Comment dire ?" se trouveraient alors confondues. L'avantage paraît évident : nous nous exprimons d'emblée sous une forme qui peut être comprise par l'exécutant-ordinateur.

   Trois arguments essentiels s'opposent à cette vision simpliste :

   - La rédaction d'une marche à suivre dans l'un ou l'autre langage de programmation, si elle convient parfaitement à l'ordinateur, est assez éloignée de notre manière de nous exprimer. N'oublions pas que ces marches à suivre, si elles sont bien destinées à commander un ordinateur, devront aussi pouvoir être relues par un être humain (y compris par celui qui les a conçues). Il est donc préférable que la forme adoptée permette une lecture et une compréhension les plus aisées possible. Ce sera le cas pour la présentation adoptée à l'issue de l'étape du "Comment faire faire ?"; ce le sera beaucoup moins lorsque la marche à suivre aura pris la forme d'un programme obligé de respecter toutes les règles syntaxiques et orthographiques imposées par le langage de programmation.
   - Il est préférable de mêler le moins possible les difficultés de conception de la marche à suivre (qui sont, on le verra, considérables) avec celles inhérentes au respect de la syntaxe du langage de programmation. Il est, comme toujours, préférable de morceler les difficultés que de les aborder ensemble.
   - Surtout, le mode d'expression retenu pour les marches à suivre, très largement indépendant du langage de programmation choisi, pourra donc être adapté à plusieurs de ces langages. En effet, le portrait que je tracerai de l'exécutant-ordinateur restera (pratiquement) identique quel que soit le langage dans lequel je finirai par devoir

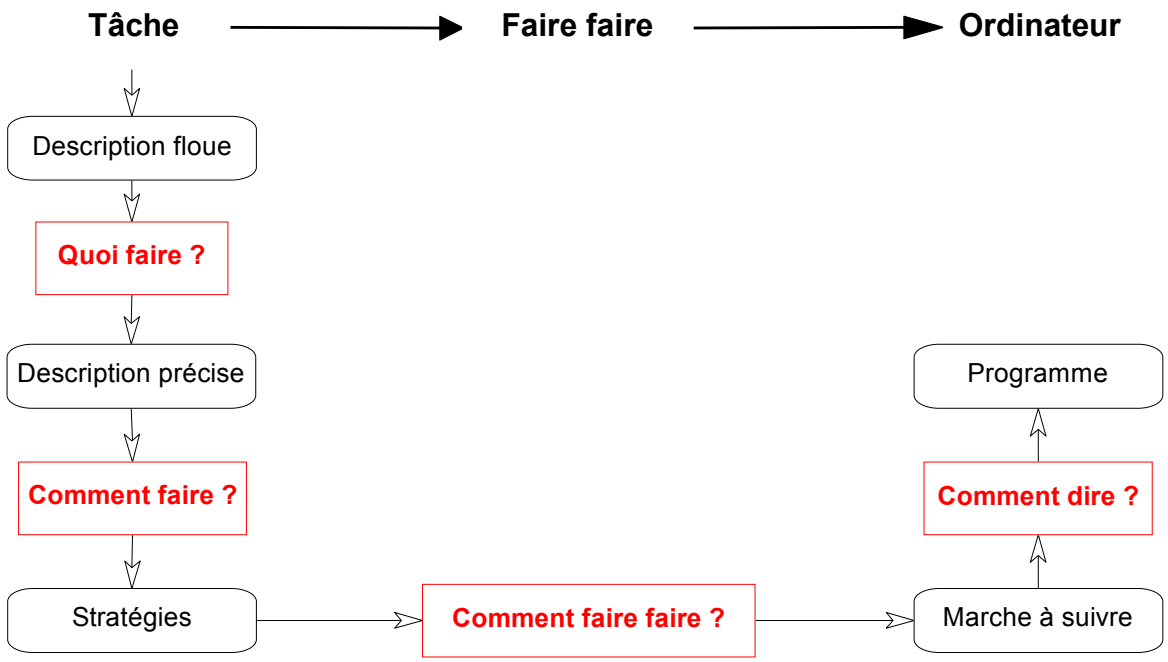
6 Rappelez-vous : la bêche courageuse...

m'adresser à lui. Ainsi, la marche à suivre explicitant telle tâche sera la même que je finisse par m'adresser à l'ordinateur en Pascal ou en Basic, par exemple.

4. Les qualités dont il faudra faire preuve à cette étape sont bien différentes de celles nécessaires aux étapes précédentes. Ici, c'est le règne des conventions, de la grammaire et de l'orthographe (spécifiques au langage de programmation choisi); qualités de rigueur et de docilité, dès lors, et surtout pas d'invention, de créativité ou d'imagination ! Ce qui est indispensable, c'est de savoir quand la virgule ou le point sont requis; comment il faut orthographier les (quelques) mots du langage choisi, quelles sont les "tournures" permises ?

5. A l'issue de cette étape, au cours de laquelle, je le souligne, l'ordinateur n'est toujours pas indispensable, je dispose donc d'un programme. Un **programme**, c'est donc seulement une marche à suivre écrite dans un langage particulier. L'avantage de cette forme d'expression de nos marches à suivre, c'est qu'elle est "compréhensible" par l'ordinateur; son désavantage, c'est d'être nettement moins lisible pour nous !

   La description schématique s'enrichit donc d'une étape supplémentaire :

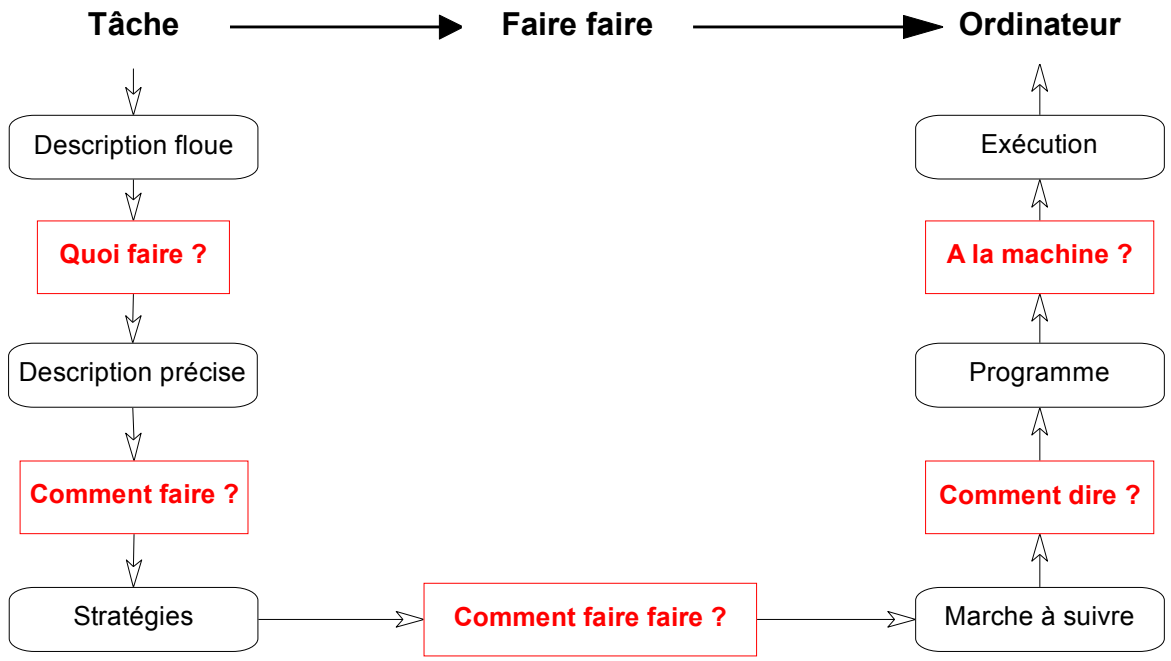



Jusqu'ici le travail d'analyse et de programmation ne nécessitait pas la présence effective de l'ordinateur. Disposant maintenant du programme conçu grâce aux étapes précédentes, je vais enfin me retrouver face à l'ordinateur pour lui fournir le résultat de mes cogitations et lui demander d'exécuter la tâche envisagée.

## *3.5 Et face à la machine ?*

Nous voici donc au terme du parcours, aux prises avec la dernière étape, celle du travail effectif à l'ordinateur.




Ce travail comportera en général plusieurs phases :

1. Je vais d'abord fournir à l'ordinateur le texte de mon programme (exprimé dans le langage de programmation retenu). L'ordinateur se contente d'entasser ce texte dans sa mémoire, tout en m'aidant à le confectionner. Il est alors gouverné par un programme spécifique qu'on appelle un **éditeur de texte**. Ce programme le transforme en quelque sorte en une "super machine à écrire", à ceci près que le texte s'inscrit à l'écran et non sur une feuille de papier et qu'il prend simultanément place dans la mémoire centrale.

2. A l'issue de cette étape d'édition, le texte résidant en mémoire est généralement sauvé sur un support mémoire externe : cassette, disquette ou disque dur. Il y acquiert une existence permanente, contrairement à la version présente en mémoire vive qui peut disparaître, par exemple lors d'une coupure de courant, réduisant alors à néant un travail de dactylographie représentant quelques minutes... ou quelques heures. Le texte ainsi recopié sur support externe pourra bien entendu être ramené (on dit "chargé") dans la mémoire pour être réutilisé.

3. Nous savons que le texte du programme est, tel quel, incompréhensible, donc inexécutable, par l'ordinateur : il doit d'abord être traduit dans le langage de ce dernier (ce qu'on appelle le langage "machine"). Pour effectuer cette traduction, l'ordinateur se laisse gouverner par un programme de traduction qu'on appelle programme **compilateur**; .

   C'est au cours de cette étape de traduction ou de **compilation** que les erreurs de syntaxe présentes dans le texte sont détectées. Cette détection des erreurs est d'ailleurs approximative : l'ordinateur "se plante" pendant la compilation (il ne "comprend" plus un texte où l'on n'a pas scrupuleusement respecté les règles de syntaxe du langage compromis) et il signale l'endroit du texte où il a commencé à ne plus pouvoir traduire. Souvent, il assortit ceci d'une tentative de repérage du type d'erreur possible sous forme d'un message plus ou moins sibyllin ! Il reste alors à l'apprenti-programmeur à réexaminer le texte de son programme, par exemple en recommençant une phase d'édition de ce texte pour le modifier en y corrigeant la ou les erreurs ayant provoqué la fin prématurée de la traduction.

Ce cycle compilation-édition se poursuit tout le temps que des erreurs subsistent. Il peut être fort long si le programmeur est peu soigneux, distrait ou irrespectueux des règles d'écriture prescrites par le langage de programmation adopté. A l'issue de ces péripéties, on finit en général par terminer victorieusement l'étape de traduction et on se trouve alors avec en mémoire la version traduite du texte : on appelle parfois version "objet" ou version "code" le résultat de la compilation réussie pour la distinguer de la version texte ou version "sujet", celle exprimée dans le langage de programmation.

Il reste évidemment, si on le souhaite, à sauver sur support mémoire externe cette version directement exécutable de la marche à suivre.

4. Disposant maintenant de cette version, l'étape suivante consiste évidemment à demander (enfin !) l'exécution de la marche à suivre : l'ordinateur se laisse alors gouverner par la version traduite de cette dernière et effectue les traitements souhaités... du moins quand tout se passe bien ! C'est en effet au cours de cette étape qu'on découvre les erreurs de conception, les incohérences, bref ce qu'on désigne souvent sous le vocable d'erreurs de "logique" (Cf. plus loin).

Cette dernière étape achève la longue marche qui, chaque fois, conduira de la tâche à son exécution par l'ordinateur. Ce n'est ni simple, ni rapide... mais c'est cela programmer !

# 4. Quelques commentaires

## *4.1 Éviter la confusion des genres !*

Je suis de cette génération de ceux qui "font de l'informatique" sans être informaticien (= sans avoir reçu une formation d'informaticien). Mon seul contact avec le monde de l'informatique au cours de mes études (de mathématique) a été une visite du centre de calcul de l'université : j'en ai seulement retenu le bruit et la rapidité des énormes imprimantes. Plus tard, quand j'ai eu besoin d'utiliser l'ordinateur (essentiellement pour calculer), on m'a dit "mais c'est bien simple, il suffit de trois jours de FORTRAN !", ce que j'ai cru ! Et pendant 10 ans, j'ai pensé que programmer, c'était "écrire des choses en FORTRAN" et, simultanément, j'ai constaté que si j'écrivais de tels programmes, c'était essentiellement pour les corriger, puis pour corriger les version corrigées, etc. (et non, comme on pourrait le croire, pour faire faire des choses par un ordinateur).

J'ai donc vécu de l'intérieur cette époque où, face à une tâche décrite de manière très floue, le réflexe était de sortir les feuilles spéciales destinées à recevoir les textes des programmes FORTRAN, puis à courir aux perforatrices pour confectionner le paquet de cartes perforées correspondant, et plein de fébrilité, à soumettre à l'ordinateur le résultat de cette absence de réflexion préalable. Se passait alors ce qui doit immanquablement se passer en pareil cas : après les allers et retours de la perforatrice à l'ordinateur pour la correction des erreurs de syntaxe, le programme finissait par être compilé (= traduit) et, prenant le contrôle de l'ordinateur, par lui faire faire... tout autre chose que ce qui était attendu, cette autre chose étant le plus souvent rien du tout !

D'autres ont dépeint bien mieux que moi les résultats aberrants et les ravages de cette absence de méthode où programmation rimait avec connaissance de la syntaxe d'un langage et où ténacité et endurance tenaient lieu de réflexion et d'intelligence. (Cf. [3], [15], [51], ...).

**Il ne faut pas confondre "dissertation française" et "accord du participe" !**

Dans beaucoup de formations à la programmation, tout se passe souvent comme si l'on annonçait l'organisation de cours sur la dissertation française : le public intéressé s'inscrit en

espérant qu'on explique comment on peut cerner un sujet, comment dresser un plan conducteur, comment argumenter, organiser sa pensée, ... Et il est tout surpris de trouver un cours sur l'accord du participe ou sur le pluriel des noms, quand ce n'est pas simplement une initiation à la dactylographie !

En programmation, il ne faut pas non plus se tromper de cible : programmer, c'est tout autre chose que maîtriser les commandes d'un ordinateur (fut-il micro) ou même qu'être un spécialiste de la syntaxe d'un langage de programmation ! Et la programmation est bien plus cousine de l'organisation des idées sous-jacente à l'art de la dissertation qu'à la connaissance des règles d'accord du participe.

Il ne faudrait pas non plus me faire dire ce que je ne dis pas : la connaissance des règles syntaxiques et orthographiques d'un langage de programmation est indispensable; sinon, jamais on ne pourra exprimer les marches à suivre conçues dans une version assimilable par l'ordinateur et, faute de la maîtrise de cette étape du "Comment dire ?", c'est tout le processus qui est hypothéqué. Mais ces connaissances "grammaticales", indispensables, sont notoirement insuffisantes pour programmer correctement.

## *4.2 Et les erreurs ?*

Dans l'approche proposée ici, qui privilégie les concepts plutôt que les aspects techniques, la programmation peut paraître, à première vue, comme une activité relativement simple. La réalité montre chaque jour qu'il n'en est rien et que, fort souvent, toute cette démarche d'analyse débouche sur des programmes "incorrects" : l'exécutant-ordinateur ne traite pas la tâche de manière satisfaisante ! En vérité, **nous ne la lui faisons pas traiter correctement**.

> Je ne parle pas ici des erreurs de syntaxe qui peuvent subsister dans le texte du programme exprimé dans le langage de programmation. J'ai dit ci-dessus que ces erreurs empêchaient la traduction du programme, et, a fortiori son exécution. Mais ces erreurs (de forme) sont (le plus souvent) aisément corrigées et ne remettent en tout cas pas en cause le contenu de la marche à suivre elle-même : on peut avoir exprimé incorrectement des choses correctes.

Un très court exemple fera sans doute mieux saisir ces incohérences. Supposons un instant que la tâche à faire traiter par l'ordinateur consiste à "conjuguer un verbe régulier de la première conjugaison à l'indicatif présent". Sans préjuger des "capacités" de l'ordinateur et sans entrer dans une formalisation de la marche à suivre, on pourrait penser à des indications du genre de ce qui suit :

- demande quel est le verbe à conjuguer;
- retire les deux derniers caractères de ce qu'on t'a fourni (pour garder le radical);
- écris "je", puis passe un blanc, puis écris le radical auquel tu colles "e";
- en dessous, écris "tu", un blanc, puis le radical en y collant "es";
- puis "il", un blanc, le radical et "e";
- etc..

> Remarquons qu'il s'agit de traitements purement formels et que, même si je n'ai encore rien dit de ce dont l'exécutant-ordinateur est capable, il s'agit là du type de manipulation dont il est friand. Ce n'est qu'une question de détails que de savoir comment lui exprimer ce qui précède à travers l'un ou l'autre langage compromis.

Une première "surprise", c'est qu'avec les explications fournies, il est parfaitement possible de faire "conjuguer" "plombier" ou même "valises". Sans sourciller et conformément à notre marche à suivre, l'ordinateur fournira :

| | | |
|---|---|---|
| je plombie | ou | je valise |
| tu plombies | | tu valises |
| il plombie | | ... |
| ... | | |

On rétorquera qu'il faut être de mauvaise foi pour demander la conjugaison de "verbes" comme ceux-là... et c'est tout à fait vrai, même si l'ordinateur ne sort pas "grandi" de telles facéties.

> Encore une fois, l'ordinateur ne "sait" pas ce qu'est un verbe. Il ne peut s'attacher qu'à la forme de ce qu'il manipule en suivant consciencieusement nos indications.
>
> Nous aurions pu nous prémunir contre un certain nombre de bêtises, en demandant par exemple à l'exécutant-ordinateur de vérifier que le "verbe" fourni se termine bien par "er" et sinon de le refuser (ce qui aurait évité la "conjugaison" de "valises", mais aurait laissé passer "plombier" ou "cerisier"). Mieux, nous aurions pu lui fournir une liste exhaustive de tous les verbes admis en lui demandant de n'accepter que les mots présents dans cette liste.
>
> Remarquons cependant, qu'à chaque fois, c'est à nous, programmeurs, qu'il revient de prévoir les opérations supplémentaires qui vont permettre d'affiner le comportement de l'exécutant.

Revenant à un comportement plus habituel, nous aurons le plaisir de voir l'ordinateur conjuguer correctement les verbes "porter", "parler", "tirer", ... et **nous décréterons donc notre programme correct**, pour autant qu'on se limite réellement à des verbes réguliers du premier groupe.

Et cela jusqu'au jour où quelqu'un demandera la conjugaison du verbe "aimer", ce que l'exécutant-ordinateur, guidé par notre marche à suivre, fournira aussitôt sous la forme

j**e a**ime

tu aimes

il aime

...

ce qui montrera qu'un programme considéré comme valide ne l'est absolument pas !

Bien entendu, il ne s'agit là que d'un tout petit exemple, sans importance; il a cependant le mérite de mettre en évidence quelques vérités premières :

1. Il est vraiment difficile d'être exhaustif et de mettre en évidence **TOUTES** les règles qui régissent notre comportement, face à la réalisation d'une tâche, même très élémentaire. Quand je parlais, plus haut, de déplier complètement une tâche, sans rien laisser dans l'ombre et en faisant une chasse sévère aux implicites, c'est de cela que je parlais !

   C'est cette complexité qui fait de la programmation, quelle que soit la méthode employée, une activité compliquée : concevoir une marche à suivre pour faire effectuer une tâche par l'exécutant-ordinateur ne sera jamais simple et ne se réduit en tout cas pas à la connaissance des règles syntaxiques gouvernant le langage de programmation employé.

2. L'ordinateur est un exécutant rapide mais affreusement docile et obéissant. Il est généralement le dernier à mettre en cause lorsque le traitement d'une tâche ne s'effectue pas comme prévu. Ce qu'il faut revoir, presque toujours, c'est la marche à suivre qui lui est fournie et que **NOUS** avons conçue... en y laissant subsister des erreurs et des incohérences.

Le drame, c'est souvent que **nous croyons lui dire autre chose que ce que nous lui disons vraiment**!

C'est peut-être étonnant, mais il a fallu fort longtemps à l'informatique (ou plutôt aux informaticiens) pour se rendre compte qu'essayer un programme pouvait (avec de la chance) montrer qu'il était incorrect, jamais prouver qu'il était correct.

## 5. Exercices

Il est évidemment fort malaisé de proposer à ce stade des exercices "techniques", puisque ce chapitre introductif a surtout pour objet de baliser notre parcours futur au pays de la programmation.

1. Eclairez quelques facettes importantes des termes suivants :
   - Ordinateur
   - Programmer
2 Quelle est la tâche principale de l'analyste-programmeur ?
3. Donnez quelques exemples de traitements formels d'informations, puis des exemples de traitements non formels.
4. Pensez-vous que les problèmes liés à la commande numérique de certaines machines outils (tours par exemple) soient liés au monde du "faire faire" tel qu'il est présenté dans ce chapitre ?
5. Donnez quelques arguments qui prouvent que l'étape du "Quoi faire ?" est essentielle.

# 2

# UNE MARCHE A SUIVRE, QU'EST-CE QUE C'EST ?

Le premier chapitre nous a montré que "**programmer**", c'est "**faire faire**" et que le coeur de la démarche, l'étape centrale, c'est la **conception et la rédaction d'une marche à suivre**, dans laquelle s'exprime ce "faire faire".

Il nous faut donc, à présent, découvrir quels sont les ingrédients d'une marche à suivre et commencer à apprendre comment elle se construit et s'écrit.

## 1. Découverte des constituants d'une marche à suivre

C'est l'examen de quelques marches à suivre de la vie quotidienne qui nous permettra de cerner quelles en sont les composantes essentielles. Et pour commencer :

### *1.1 Les instructions d'action élémentaire*

*- Effectuer les manœuvres dans l'ordre indiqué sur le schéma -*

❶ *Appuyer sur l'interrupteur POWER pour le régler sur la position ON.*

❷ *Appuyer sur la touche d'éjection (EJECT) pour ouvrir le compartiment à cassette.*

❸ *Mettre une cassette en place, le côté laissant voir la bande dirigé vers le bas et la face à enregistrer dirigée vers l'extérieur. (Vérifier si les languettes de protection de la cassette n'ont pas été enlevées.)*

❹ *Régler le sélecteur sur la position correspondant à la source employée. Se reporter à la page 8 "Sélecteur de signal d'entrée" au chapitre "Nomenclature des organes et de leurs fonctions*

❺ *Régler les commutateurs NR SYSTEM sur la position du que convient.*

❻ *Régler les sélecteurs de bande (TAPE SELECT) en fonction du type de bande utilisée. (Voir la page 13)*

❼ *Enfoncer la touche PAUSE. Lorsque sa diode électroluminescente correspondant s'allume, enfoncer la touche REC. De cette manière, la platine est mise en mode d'attente d enregistrement La diode électroluminescente correspondant aux touches REC et PLAY sont aussi allumées.*

❽ *Ajuster le niveau d'enregistrement. (Voir page 18.)*

❾ *Appuyer de nouveau sur la touche de PAUSE pour libérer le mode de pause et faire démarrer 1 enregistrement.*

*Marche à suivre n° 1*

Ce premier exemple est destiné à un "exécutant“ possesseur d'un enregistreur à cassette et son but, c'est de lui expliquer comment effectuer un enregistrement.

Qu'y trouvons-nous ?

D'abord, et c'est ce qui en constitue l'essentiel, des instructions comme :

*"Appuyer sur l'interrupteur POWER pour le régler sur la position ON"*

*"Appuyer sur la touche d'éjection (EJECT) pour ouvrir le compartiment à cassette"*

*Etc..*

Chacune de ces phases est l'indication d'une action dont le technicien, rédacteur de la marche à suivre, croit capable l'exécutant.

Nous dirons qu'il s'agit là, à chaque fois, d'une

**instruction d'action élémentaire**

1. Il s'agit **d'instructions** d'action, d'ordres et non d'actions. Une marche à suivre ne "fait" rien; elle fera faire des choses par un exécutant au moment où celui-ci commencera à s'atteler à la tâche qui y est décrite. Chaque instruction d'action donnera lieu alors à une action. Il est pourtant classique en informatique de trouver des abus de langage du type "ce programme calcule...." ou "ce programme compte ...". Un programme, qui n'est jamais qu'une marche à suivre exprimée dans un langage de programmation, ne calcule ou ne compte pas plus qu'une recette de cuisine ne fait la cuisine !

2. Le qualificatif **élémentaire** souligne qu'il s'agit là d'une action dont l'exécutant est capable. C'est donc la connaissance des possibilités de ce dernier qui permet de décider que telle action est élémentaire pour lui (et qu'on peut donc faire figurer dans la marche à suivre l'instruction correspondante) et que telle autre action dépasse ses possibilités.

   Dans l'exemple, si le technicien a cru bon de décortiquer de cette manière la tâche, c'est qu'il lui a semblé que l'instruction

   *"Procéder à l'enregistrement",*

   qui désigne l'activité globale explicitée dans la marche à suivre, ne correspondait pas à une action "élémentaire" pour l'exécutant moyen.

   Par contre, si la marche à suivre comporte l'instruction

   *"Appuyer sur l'interrupteur POWER pour le régler sur la position ON",*

   c'est que le rédacteur a estimé qu'il s'agissait là d'une action à la portée de l'exécutant moyen et qu'il pouvait, dès lors, la considérer comme élémentaire.

   > Je l'ai déjà souligné au premier chapitre : dans notre cas, les marches à suivre seront destinées à un "robot" et non à un être humain. Les possibilités de ce "robot" seront définies sans ambiguïté et, dès lors, le caractère élémentaire des instructions d'action sera clairement précisé et non (comme ci-dessus) laissé à l'appréciation du rédacteur de la marche à suivre.

3. La connaissance, face à une tâche, du répertoire des actions élémentaires de l'exécutant à qui l'on veut faire faire cette tâche est indispensable : on ne peut pas donner des ordres à quelqu'un si l'on ne sait pas ce qu'il est capable de comprendre et de faire.

   Cependant, la seule connaissance du lexique

   instructions actions élémentaires;,

   (on lui dit) (il fait)

si elle est indispensable est aussi, on le verra, bien insuffisante pour concevoir et organiser les marches à suivre qui intégreront ces instructions.

## *1.2 Les structures de contrôle*

### 1.2.1 La séquence;

Nous venons de découvrir le premier des ingrédients d'une marche à suivre : ce sont les instructions d'action élémentaire[1].

La donnée dans le désordre de ces instructions d'action élémentaire est évidemment insuffisante.

> Si, disposant de la recette-marche à suivre qui permet de confectionner la blanquette de veau, j'en découpe soigneusement le texte pour isoler chaque instruction sur une petite bandelette de papier, que je mélange soigneusement tous ces bouts de papier et que je passe le tas obtenu à l'exécutant-cuisinier, chacun sait que le résultat risque bien d'être fort éloigné de la vraie blanquette de veau !
>
> Et pourtant, toutes les instructions sont présentes ... mais dans le désordre !

Les diverses instructions doivent être **organisées** pour concourir au résultat souhaité. Et la première manière d'organiser, donc de contrôler le déroulement des actions commandées, est bien traduite dans la recommandation initiale de l'exemple :

*"- Effectuer les manœuvres dans l'ordre indiqué sur le schéma -"*

renforcée par la **numérotation** qui accompagne le texte de la marche à suivre.

Cette première structure organisatrice est tellement naturelle qu'elle passe presque inaperçue; le plus souvent, elle n'est pas explicitement rappelée. Nous l'appellerons

**la séquence**

Simplement, les actions commandées se succéderont dans l'ordre où les instructions correspondantes sont écrites.

1. Cette première manière de marier entre elles les actions élémentaires est aussi, on le verra, la plus simple à maîtriser. Lorsqu'elle est seule présente, le problème c'est seulement de décortiquer la tâche, l'action complexe, en la suite, la **séquence** des actions élémentaires dont la succession équivaudra à l'accomplissement de la tâche tout entière.
2. Il est important de bien percevoir la différence de statut entre l'indication

   *"- Effectuer les manœuvres dans l'ordre indiqué sur le schéma -"*

   et une instruction d'action comme

   *"Enfoncer la touche pause"*

   La première phrase, confirmée par la numérotation, n'est pas directement destinée à faire agir l'exécutant; elle indique plutôt comment les actions doivent être organisées.

Les consignes par lesquelles nous indiquerons comment les actions correspondant aux instructions d'action élémentaire doivent s'organiser, nous les appellerons des

**instructions d'organisation**

---

[1] C'est à dessein que "action élémentaire" est écrit au singulier : à chaque instruction élémentaire correspond une et une seule action de l'exécutant.

ou encore des

**structures de contrôle**

La séquence en est le premier exemple et nous allons en découvrir à présent quelques autres.

### 1.2.2 L'appel de procédure;

Le texte nous montre une seconde manière de structurer une marche à suivre. C'est elle qui est présente dans les phrases :

> *"Régler le sélecteur sur la position correspondant à la source employée.*
> *(Se reporter à la page 8 )"*

ou *"Régler les sélecteurs de bande en fonction du type de bande utilisée.*
*(Voir page 13)"*

ou *"Ajuster le niveau d'enregistrement.*
*(Voir page 18)"*

Il s'agit là, de toute évidence, à chaque fois, d'une **instruction d'action trop compliquée** assortie de la **référence à une annexe**. Les actions commandées sont trop complexes, et si le technicien les a tout de même énoncées telles quelles, il renvoie, en même temps, l'exécutant à des marches à suivre annexes qui explicitent chacune des indications complexes mentionnées.

En programmation, nous retrouverons ce duo "instruction d'action complexe - référence à une annexe explicative" sous le vocable

**appel de procédure**

1. "Procédure" est évidemment le terme désignant chacune des marches à suivre annexes explicitant les instructions d'action complexe figurant dans la marche à suivre principale.

   On parlera aussi de routine, sous-routine, sous-programme, ...

2. On pourrait évidemment se passer de cette structure : plutôt que de laisser subsister l'instruction complexe et, à part, le texte de la procédure explicative, il suffirait d'inclure dans le texte même de la marche à suivre principale, à l'endroit voulu, le texte entier de la procédure annexe.

   Ainsi, au lieu de

   > *"Ajuster le niveau d'enregistrement.*
   > *(Voir page 18)"*

   on trouverait

***Réglage du niveau d'enregistrement***

*Régler le niveau d'enregistrement avec l'aide des indications fournies par les voyants de signalisation du système de détection des niveaux de crête. Le réglage du niveau d'enregistrement est plus facile s'il s'agit d'enregistrer une émission de radio en modulation de fréquence ou bien un disque, car, en ce cas, les crêtes ont déjà été éliminées.*
*Si en revanche, il s'agit d'effectuer un enregistrement sur le vif, ou bien de copier sur le KD-D35 un enregistrement effectué par un magnétophone à bande, le réglage est plus délicat, car de telles sources audio comportent à la fois des signaux très faibles et des signaux très forts.*
*D'une manière générale, les résultats obtenus sont les suivants :*
*(A) Si au cours de l'enregistrement, le voyant indicateur de niveau de saturation +9 s'allume fréquemment, l'enregistrement contiendra des distorsions sonores.*

*(B) Si au cours de l'enregistrement, le voyant indicateur de niveau de saturation 0 ne s'allume pratiquement jamais, l'enregistrement obtenu sera d'un niveau sonore insuffisant et le bruit de souffle de la bande se fera entendre nettement.*

*(C) De manière à obtenir le niveau d'enregistrement optimum, régler la commande de niveau d'entrée, de sorte que l'indicateur à +3dB s'allume lorsque l'entrée maximale est appliquée.*
*(Le voyant de niveau de crête +6 peut s'allumer de temps en temps).*

*Toutefois, le niveau d'enregistrement optimal varie en fonction de la source sonore et du type de bande utilisée. Spécialement avec la bande au métal, du fait que son point de saturation est plus élevé que les bandes ordinaires, ce n'est pas important si le voyant +9 s'allume de temps en temps.*

*Annexe à la marche à suivre n° 1*

Evidemment, ceci allongerait le texte de la marche à suivre principale et la démarche d'analyse de la tâche, telle que l'a menée le technicien, serait moins apparente. Mais nous reviendrons plus loin sur les raisons qui conduisent à utiliser **très intensivement** la structure "appel de procédure".

> On peut, dès à présent, remarquer que l'utilisation d'instructions d'action complexe (= appel de procédure) permet de s'affranchir, du moins au début de l'analyse de la tâche, des caractéristiques de l'exécutant. On ne s'obligera pas à ce que toutes les instructions d'action mentionnées dans la marche à suivre principale soient élémentaires et correspondent à chaque fois aux capacités réduites de ce dernier. On commencera par morceler le travail à faire effectuer en actions dont la portée et le but sont bien compris (du moins par les humains), débouchant ainsi sur un premier niveau d'écriture. Il suffira de reprendre alors chacune des actions trop complexes pour la décortiquer à son tour, et ainsi de suite. Cette division du travail qui, plutôt que de se perdre dans les détails liés aux capacités limitées de l'exécutant, commence par fractionner la tâche en gros constituants (eux-mêmes repris pour être décomposés jusqu'à un émiettement en composants élémentaires), s'appelle, en programmation l'**approche descendante** ou (en anglais) "top-down" programming. Il s'agit là d'une méthode essentielle qui s'incarne naturellement dans la structure d'appel de procédure.

3. C'est à nouveau la connaissance des capacités de l'exécutant qui conduit à considérer comme complexe telle action et comme élémentaire telle autre. Et il est bien évident que, tout compte fait, les instructions données doivent finir par être élémentaires, soit directement, soit au sein des procédures (= marches à suivre annexes) explicatives.

4. Rien n'empêche évidemment qu'au sein d'une procédure, on trouve à nouveau des instructions d'action complexe référant à d'autres procédures et ainsi de suite ... La structure globale ressemble alors beaucoup plus à une cascade, à une arborescence, qu'à une liste (linéaire) d'instructions élémentaires. La marche à suivre principale fait appel à des annexes, certaines de ces dernières renvoyant à leur tour à des annexes ... et ainsi de suite.

### 1.2.3 L'alternative

Le second exemple de marche à suivre va nous en apprendre plus sur les structures de contrôle.

*Procéder de la façon suivante pour lancer le moteur froid et quelle que soit la température extérieure :*

* *Tirer à fond le bouton de commande du dispositif de départ à froid.*
* *Tourner la clé de contact en position (2). Le témoin de préchauffage s'allume au tableau de bord.*
* *Maintenir la clé de contact dans cette position jusqu'à ce que le témoin de préchauffage s'éteigne.*
* *Lancer le moteur en tournant la clé de contact en position (3) aussitôt que le témoin de préchauffage est éteint.*
* *Ne pas accélérer, pendant le lancement, lorsque la température est supérieure à 0°C.*

** Enfoncer complètement l'accélérateur pendant le lancement du moteur, si la température est inférieure à 0°C.*

** Ne pas actionner le démarreur plus de 30 secondes.*

*Si le moteur ne démarre pas :*

** Préchauffer encore une fois après une pause de 30 secondes.*

** Actionner à nouveau le démarreur.*

** Continuer à actionner le démarreur, si des crépitements irréguliers d'allumage se produisent, jusqu'à ce que le moteur tourne.*

** Repousser complètement le bouton-tirette de commande du dispositif de départ à froid après une minute environ lorsque le moteur est lancé.*

*Marche à suivre n° 2.*

Nous y retrouvons bien entendu les instructions d'action élémentaire :

*"Tirer à fond le bouton de commande du dispositif de départ à froid"*

*"Activer à nouveau le démarreur"*

*Etc.*

La structure de contrôle séquentielle, même si elle n'est pas explicitement rappelée, est bien entendu présente. Un exécutant normal commencera par le début et effectuera successivement les actions commandées par la séquence des instructions.
Si nous n'y trouvons pas d'appel de procédure, cette marche à suivre illustre cependant une nouvelle structure de contrôle :

*"Ne pas accélérer, pendant le lancement,* ***lorsque*** *la température est supérieure à 0° C"*

*"Enfoncer complètement l'accélérateur, pendant le lancement du moteur,* ***si*** *la température est inférieure à 0° C"*

*"****Si*** *le moteur ne démarre pas :*

- *préchauffer encore une fois ...*

*...*

Dans chaque cas, l'action (ou le groupe d'actions) commandée est **conditionnée** par la réalisation d'un événement.

Nous retrouvons cette même structure dans le troisième exemple :

*Débrancher le câble négatif (-) de la batterie, vérifier l'état de la céramique des bougies. Vérifier également les câbles haute tension des bougies, de la bobine ainsi que les jonctions à chaque extrémité des câbles haute tension. Vérifier enfin le dessus de la bobine et de l'allumeur. Si nécessaire, nettoyer ces pièces à l'aide d'un chiffon propre. Lorsqu'on intervient sur des bougies, les manipuler avec soin afin d'éviter de détériorer l'isolant céramique. Mesurer l'écartement des électrodes : utiliser une jauge d'épaisseur, éventuellement régler cet écartement en cintrant à la demande l'électrode de masse (voir types de bougies et écartement des électrodes au chapitre CARACTERISTIQUES TECHNIQUES du dépliant). Sur les moteurs 1,1-1,3 et 1,6 litres C.V.H., utiliser uniquement des bougies d'allumage MOTORCRAFT Super avec électrode centrale à âme laiton (Super AGP 22 C ou Super AGPR 12C), ainsi que sur les moteurs V6-2,0/2,3 litres (Super AGR 22 C). Rechercher éventuellement la présence de fêlures très fines ou autres détériorations, le cas échéant, faire remplacer les pièces défectueuses par votre concessionnaire FORD.*

*Si les câbles haute tension sont débranchés de l'allumeur, avant de les rebrancher, noter leur emplacement respectif.*
*Les câbles des bougies sont d'ailleurs numérotés, afin de faciliter leur branchement. Voir ordre d'allumage à la section intitulée CARACTERISTIQUES TECHNIQUES. Après réalisation des contrôles ci-dessus, ne pas omettre de rebrancher le câble de masse de la batterie.*

*Marche à suivre n° 3*

On la voit à l'œuvre à travers les phrases :

*"**Si** nécessaire, nettoyer ces pièces à l'aide d'un chiffon propre"*

*"**Le cas échéant**, faire remplacer les pièces défectueuses par votre concessionnaire FORD"*

*"**Sur** les moteurs 1.1-1,3 et 1,6 litres C.V.H. utiliser uniquement des bougies ..."*

Cette structure, qui se révèle par des mots comme : si,
éventuellement,
lorsque,
le cas échéant,
...,

n'est évidemment pas une instruction d'action. Au contraire, ces phrases énoncent des **conditions** qui vont commander l'effectuation ou la non-effectuation de certaines actions décrites. Nous pourrions en uniformiser la présentation sous la forme :

*__SI__ la température est supérieure à 0° C __ALORS__*
*ne pas accélérer pendant le lancement*
*__SINON__ (si la température est inférieure à 0° C)*
*enfoncer complètement l'accélérateur pendant le lancement du moteur*

*__SI__ le moteur ne démarre pas __ALORS__*
*- préchauffer encore une fois après une pause de 30 secondes*
*- actionner à nouveau le démarreur*
*- ...*

*__SI__ les pièces sont sales __ALORS__*
*nettoyer ces pièces à l'aide d'un chiffon propre*
*__SI__ on constate des détériorations __ALORS__*
*faire remplacer les pièces défectueuses ...*

Il s'agit bien là d'une nouvelle structure de contrôle :

**la structure alternative**

On parle quelquefois aussi à son propos de **structure conditionnelle**.

1) Le qualificatif "**conditionnel**" insiste sur la présence d'une **condition** de laquelle dépendra la suite des actions effectuées. Le mot "**alternative**" met plutôt l'accent sur le **choix** auquel l'exécutant est conduit. De toute manière, ce sont bien ces deux aspects qui sont présents simultanément dans cette nouvelle structure de contrôle.

2) La condition énoncée doit évidemment être décidable sans ambiguïtés par l'exécutant. Ce ne pourra donc jamais être des assertions aussi floues que :

*__SI__ le plat est assez salé __ALORS__ ...*

ou, comme dans la marche à suivre n° 3,

*SI les pièces sont sales ALORS ...*

où l'appréciation est (trop) largement laissée à l'exécutant.

On voit donc apparaître un concept nouveau à l'occasion de la découverte de cette structure de contrôle, celui de **condition**. Il s'agit là d'un énoncé, d'une assertion dont l'exécutant peut, au moment où il la rencontre dans l'exécution de la marche à suivre, dire à coup sûr si elle est vraie ou fausse. La vérité (ou la fausseté) d'une condition dépend évidemment du "contexte" et du moment où elle est évaluée.

Je sais, à présent, que lorsque je tracerai le portrait d'un exécutant (y compris celui de l'exécutant-ordinateur), il me faudra dire :

- quelles sont les instructions d'action élémentaire qui le caractérisent;
- quelles sont les conditions qu'il est capable d'évaluer.

Nous résumerons, en évitant à ce stade une formalisation excessive, la structure alternative (ou conditionnelle) par les mots :

**SI** condition **ALORS**
    Instructions
**SINON**
    Instructions
suite de la marche à suivre

ou

**SI** condition **ALORS**
    Instructions
suite de la marche à suivre

Comme on le constate, elle prend donc deux formes voisines. Celle d'abord où une réelle alternative est présente : lorsque la condition est vraie, certaines actions sont à exécuter, lorsqu'elle est fausse, c'est un autre chemin qui doit être suivi, avant, dans chaque cas, de retrouver la suite "normale" de la marche à suivre. La seconde forme n'offre pas réellement un double chemin : simplement, lorsque la condition énoncée est vraie, les actions qu'elle commande sont effectuées avant de passer à la suite; lorsqu'elle est fausse, on passe directement à cette suite.

### 1.2.4 La répétition

Si nous continuons à examiner la marche à suivre n° 2 de "lancement du moteur froid", nous y découvrons une structure de contrôle supplémentaire exprimée dans des phrases comme :

*"**Continuer** à actionner le démarreur, si des crépitements irréguliers d'allumage se produisent, **jusqu'à ce que** le moteur tourne."*

*"**Maintenir** la clé de contact dans cette position **jusqu'à ce que** le témoin de préchauffage s'éteigne."*

Le terme important ici c'est "**jusqu'à ce que**" qui commande en quelque sorte la poursuite ou la **répétition** d'une action (ou d'un groupe d'actions). Il s'agit là de la structure que nous appellerons la répétition.

C'est cette même structure répétitive qu'on détecte dans la marche à suivre suivante :

*Serrer à fond la vis de réglage A. Puis la redesserrer d'un tour et demi.*
*Mettre le moteur en marche et le laisser tourner pendant quelques minutes pour le réchauffer.*
*Le moteur tournant à la vitesse normale, serrer avec précaution la vis de réglage A jusqu'à ce que le moteur ralentisse (mélange de carburant pauvre).*
*Desserrer ensuite doucement la vis de réglage au-delà du point où le moteur tourne le plus régulièrement, jusqu'à ce que le moteur commence à tourner irrégulièrement (mélange de carburant riche).*

*Serrer doucement la vis de réglage, jusqu'à ce que le moteur tourne régulièrement. Régler la manette des gaz au ralenti.*
*La marche au ralenti se règle au moyen de la vis B. Le moteur doit avoir une marche au ralenti rapide (env. 1750 tr/min).*
*Tester le réglage du carburateur. Amener la manette des gaz de la position "Ralenti" à la position "Pleins gaz". Le moteur doit accélérer uniformément.*
*Sinon, c'est que le mélange de carburant est un peu trop pauvre. Dans ce cas, desserrer légèrement la vis de réglage A. Ce réglage peut provoquer un ralenti légèrement irrégulier.*

*Marche à suivre n° 4*

avec des phrases comme :

*"Serrer avec précaution la vis de réglage A **jusqu'à ce que** le moteur ralentisse."*

*"Serrer doucement la vis de réglage **jusqu'à ce que** le moteur tourne régulièrement."*

Dans chacun de ces exemples, il s'agit plutôt de **poursuivre** une action **jusqu'à** ce qu'une condition devienne vraie, plutôt que de **répéter** une action **jusqu'à** la réalisation d'une condition; le mot "répéter" n'y est d'ailleurs pas présent.

On verra que les actions élémentaires de l'exécutant-ordinateur ne seront jamais de longue durée et qu'en ce qui le concerne le mot "**jusqu'à ce que**" sera associé au mot "**répéter**" et exprimera la reprise d'une action (ou d'un groupe d'actions), plutôt que la poursuite d'une action pendant un certain temps.

C'est aussi cette structure répétitive qu'on trouve dans les expressions :

*"Feuilleter **jusqu'à** trouver la page portant les coordonnées de l'abonné."*

*"Ajouter le sel, pincée par pincée, et goûter **jusqu'à ce** que le plat soit assez salé."*

que nous pourrions traduire pour mettre clairement en évidence la répétition sous-jacente :

*"**Répéter***
*Tourner une page*
***jusqu'à ce que** la page porte les coordonnées de l'abonné."*

*"**Répéter***
*Ajouter une pincée de sel,*
*goûter*
***jusqu'à ce que** le plat soit assez salé."*

Dans la langue française, la répétition se traduit de bien des manières et, si le mot **jusqu'à ce que** est bien souvent présent pour évoquer la **condition** dont la réalisation arrêtera la répétition, le mot "**répéter**" reste en général implicite ou se cache sous d'autres tournures linguistiques.

Il existe une seconde manière de commander la répétition d'un groupe d'actions, c'est celle qui s'annonce par les mots "**Tant que ...**" , comme dans les instructions :

*"**Tant que** le mélange n'est pas assez mousseux,*
*continuez à battre".*

ou

*"**Tant qu'il** reste des fiches à traiter,*
*appliquez-leur le traitement standard".*

Nous retiendrons donc par la suite, deux formes de structure répétitive :

**REPETER**
Instructions
**JUSQU'A CE QUE** condition
suite de la marche à suivre

et

**TANT QUE** condition
Instructions
suite de la marche à suivre

### 1.2.5 Le branchement

Il existe une dernière manière d'organiser les actions commandées dans une marche à suivre. On la devine dans l'exemple suivant :

1. *Lorsque le message FOUND, le nom du programme et le message LOADING apparaissent à l'écran, mais que ce n'est pas le programme désiré, appuyez sur la touche STOP.*
2. *Assurez-vous que la touche PLAY de l'unité est encore enfoncée.*
3. *Tapez LOAD et appuyez sur RETURN.*
4. *Surveillez le nom du programme sur le point d'être chargé.*
   *Si c'est bon, attendez la fin du chargement et continuez comme dans les exemples précédents.*
5. *Si ce n'est toujours pas votre programme, retournez à la première étape de cette série de directives et répétez-les jusqu'à ce que l'ordinateur trouve le programme voulu.*

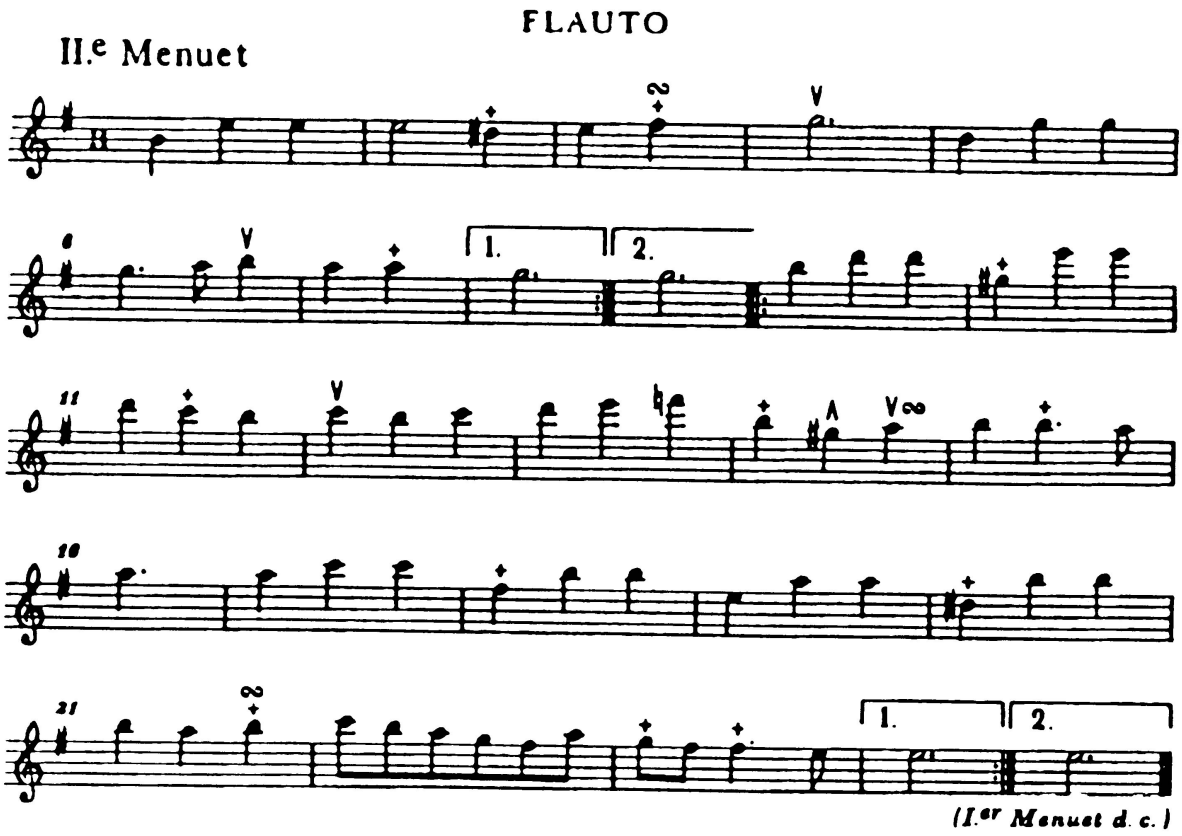
*Marche à suivre n° 5*

Elle se traduit à travers les mots

*"**Retournez** à la première étape ..."*

On la trouve encore, dans ce magnifique exemple de marche à suivre que constitue la partition suivante :




*Marche à suivre n° 6*

avec le symbole

qui renvoie, soit au début de la partition, soit au symbole jumeau

Il s'agit là, à chaque fois, de l'indication explicite de poursuivre les actions commandées, non pas avec celle qui suit dans le texte (ou dans la partition), mais en se rendant **ailleurs** dans la marche à suivre.

C'est en quelque sorte la façon la plus brutale de rompre le pur déroulement séquentiel : on envoie l'exécutant à un endroit de la marche à suivre précisé, où il poursuit les actions commandées. C'est la structure que nous résumerons par :

"**Aller en** ..."

Elle a pour nom, en programmation, le

**branchement**

1. Il est important de ne pas confondre branchement et appel de procédure. Dans le cas du branchement, aucune annexe explicative ne vient expliciter une instruction d'action complexe. L'exécutant est maintenu au sein de la marche à suivre : simplement il est "baladé" d'un endroit à un autre de cette marche à suivre.

2. On conçoit que les marches à suivre où la structure de branchement est utilisée voient certaines de leurs instructions porter un signe distinctif (numéro, étiquette, ...). Lorsqu'on veut envoyer l'exécutant à un endroit particulier de la marche à suivre, cet endroit doit être repérable. Ainsi, dans l'exemple 5 ci-dessus, les instructions sont numérotées et le branchement précise :

   *"Retournez à la **première** étape."*

   Et dans le cas de la partition, c'est le symbole

   qui marque l'endroit où le branchement symbolisé par

   renvoie.

Nous allons arrêter ici notre recherche des structures de contrôle présentes dans les marches à suivre "de tous les jours". Il y a pourtant d'autres structures, comme celle qui s'annonce par les mots "Quand" ou "Lorsque". Mais nous ne retrouverons, dans les marches à suivre destinées à l'exécutant-ordinateur (celles de la programmation) que :

- la séquence;
- l'alternative;
- la répétition (sous les deux formes présentées);
- l'appel de procédure;
- le branchement.

## *1.3 Les commentaires*

Je n'ai pas épuisé la liste des ingrédients des marches à suivre. Un retour à la marche à suivre n°1 va permettre l'évocation d'une troisième et dernière composante.

A côté des **instructions d'action élémentaire** et des **structures organisatrices** qui constituent l'essentiel de la marche à suivre, celle-ci comporte des assertions comme :

> *"De cette manière, la platine est mise en mode d'attente d'enregistrement. Les diodes électroluminescentes correspondant aux touches REC et PLAY sont aussi allumées."*

Il ne s'agit pas là d'une instruction d'action (qu'elle soit élémentaire ou complexe). Ce n'est pas non plus une structure qui va organiser le déroulement des actions prescrites. C'est simplement une remarque, un **commentaire** destiné à rassurer l'être humain exécutant cette marche à suivre. On pourrait d'ailleurs fort bien supprimer cette phrase du texte sans dénaturer la marche à suivre résultante.

Ainsi donc, le dernier constituant est l'ensemble des

**commentaires**

Dans le cas des marches à suivre de la programmation, qui s'adresseront à l'exécutant-ordinateur, il est illusoire et inutile de vouloir l'encourager ou le rassurer. Les commentaires qui seront cependant fort nombreux s'adresseront alors à un autre qu'à l'exécutant : ils seront là pour le **lecteur** de la marche à suivre. Ce dernier est un être humain et chaque commentaire est un "clin

d'oeil" d'un être humain (le programmeur) à un autre être humain (le lecteur). On le verra, la compréhension d'une marche à suivre est une tâche complexe, comme sa correction ou sa modification. Les commentaires, dont l'exécutant n'a que faire, seront là pour faciliter cette compréhension, le premier intéressé étant souvent le programmeur lui-même qui devra, parfois après plusieurs mois, se replonger dans le texte d'une marche à suivre qu'il avait conçue.

## 2. Les structures de contrôle

Nous avons découvert qu'à côté des **instructions d'action élémentaire** et des **commentaires**, les marches à suivre comportaient toujours des instructions d'organisation de ces actions : **les structures de contrôle**. Je vais, à présent, les passer en revue pour compléter les informations à leur propos et décrire des manières de représenter ces structures dans le texte même des marches à suivre.

### *2.1 Les divers modes de représentation des structures de contrôle;*

Il existe diverses possibilités de faire (plus ou moins) clairement apparaître les structures de contrôle présentes dans une marche à suivre et cela, indépendamment des instructions d'action élémentaire qu'elle comporte.

Je présenterai ici cinq types de représentation :

- pseudo-code, encore appelé langage de description d'algorithmes,
- graphes de Nassi-Schneidermann (notés GNS dans la suite),
- Pascal,
- organigrammes,
- Basic.

1. Pascal et Basic sont deux langages de programmation dans lesquels les marches à suivre peuvent être exprimées. Il est normal, dès lors, qu'y figure une manière de traduire les structures de contrôle ou du moins certaines d'entre elles. L'avantage de ce mode d'expression, c'est qu'il est "compréhensible" (après une phase de traduction) par l'exécutant-ordinateur. Le désavantage, c'est qu'il est impératif d'y respecter une syntaxe stricte imposée par chacun de ces langages et que, on le verra, l'expression des structures de contrôle y est moins "visible" que dans les GNS ou les organigrammes.
2. Il est impossible de parler de Basic, tant est grand le nombre de dialectes de ce langage. C'est donc toujours seulement de "tel" Basic qu'il s'agirait. Mon choix s'est porté sur une version particulièrement pauvre, afin de mettre en évidence les problèmes posés lors de l'expression des marches à suivre (et particulièrement des structures de contrôle) dans un langage rudimentaire et étriqué.
3. On verra que s'il fallait classer ces diverses manières d'exprimer les structures de contrôle, on réunirait, d'une part,

   pseudo-code
   GNS
   et Pascal

et, d'autre part,

   organigramme

et Basic.

Il ne reste plus qu'à passer en revue les diverses structures de contrôle découvertes en indiquant comment chacune s'exprime dans ces divers modes de représentation.

## *2.2 La séquence*

C'est le mode d'organisation le plus simple à utiliser. On assiste, lorsqu'elle est seule présente, à une **correspondance parfaite** entre la **suite** des instructions d'action élémentaire constituant la marche à suivre et la **succession** des actions posées lors de l'exécution.

Ainsi, en reprenant les extraits de la marche à suivre n° 1,

| **Texte de la marche à suivre** | **Lors de l'exécution** |
|---|---|
| | L'exécutant |
| ❶ *Appuyer sur l'interrupteur POWER pour le régler sur la position ON.* | appuie sur l'interrupteur POWER pour le régler sur la position ON, **puis** |
| ❷ *Appuyer sur la touche d'éjection (EJECT) pour ouvrir le compartiment à cassette.* | appuie sur la touche d'éjection (EJECT) pour ouvrir le compartiment à cassette, **puis** |
| ❸ *Mettre une cassette en place, le côté laissant voir la bande dirigé vers le bas et la face à enregistrer dirigée vers l'extérieur* | met une cassette en place |

En général, donc :

| **Texte de la marche à suivre** | **Lors de l'exécution** |
|---|---|
| *Instruction d'action 1* → | Action 1, **puis** |
| *Instruction d'action 2* → | Action 2, **puis** |
| *Instruction d'action 3* → | Action 3, **puis** |
| ... | ... |

### 2.2.1 Représentation sous forme de pseudo-code

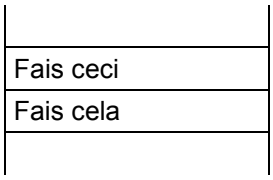
On se contentera de marquer la séquence par un passage à la ligne, pour déboucher sur une écriture du type :

```
...
Fais ceci
Fais cela
...
```

### 2.2.2 Représentation sous forme GNS

La séquence sera marquée par une succession de rectangles accolés dans lesquels s'inscrivent les instructions d'action.



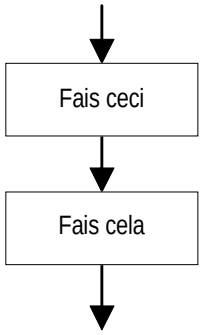

### 2.2.3 Représentation en Pascal

Outre la constatation que les instructions d'action seront codées dans un langage davantage proche de l'anglais, la séquence se marquera par le fait que les instructions seront (généralement) séparées par des " ; " et que, de plus, pour améliorer la lisibilité, nous effectuerons un passage à la ligne :

```
...
Do this;
Do that;
...
```

### 2.2.4 Représentation en organigrammes;

Les instructions seront aussi enfermées dans des rectangles reliés par des flèches qui marqueront la succession :




### 2.2.5 Représentation en Basic

Outre un langage plus proche de l'anglais, la séquence est signalée par la numérotation des instructions :

```
...
70 Do this
80 Do that
...
```

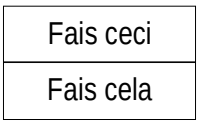
J'espère que personne ne croit réellement que ces "Do this", "Do that" constituent des instructions Pascal ou Basic. Mon but ici n'est évidemment pas de décrire ces langages mais seulement de montrer comment les structures de contrôle s'y incarnent.

En résumé pour la séquence :

**Pseudo-code**

```
...
Fais ceci
Fais cela
...
```

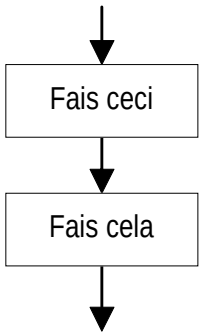
**GNS**




**Pascal**

```
...
Do this;
Do that;
...
```

**Organigramme**




**Basic**

```
...
70 Do this;
80 Do that;
...
```

On le voit, il s'agit bien là de manières équivalentes de traduire et de faire apparaître, quelles que soient les instructions d'action employées, la structure séquentielle.

## *2.3 L'alternative*

A partir de maintenant, la correspondance entre le texte de la marche à suivre et le déroulement de son exécution qui était de règle avec la séquence va disparaître.

Dans le cas de l'alternative, il est impossible de prévoir exactement ce que sera l'exécution de la marche à suivre puisque des conditions, dont nous ne connaissons pas la valeur de vérité au moment de la rédaction, vont être présentes. Par exemple, dans la marche à suivre n° 2,

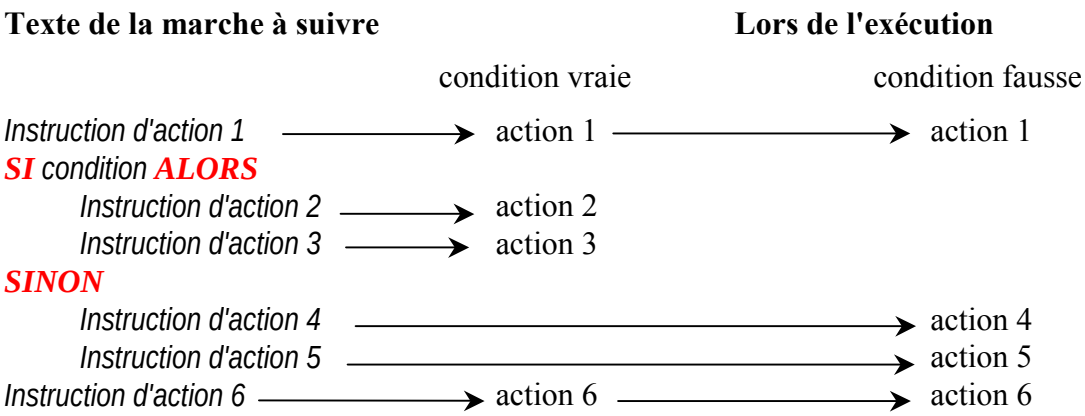
| **Texte de la marche à suivre** | **Lors de l'exécution** |
|---|---|
| ***SI*** *la température est supérieure à 0° C* ***ALORS***<br>*ne pas accélérer pendant le lancement*<br>***SINON***<br>*enfoncer complètement l'accélérateur pendant le lancement du moteur* | impossible à prévoir au moment de l'écriture de la marche à suivre |

Tout va dépendre, lors de l'exécution, de la température. Mais en tout cas, une seule des deux actions décrites sera exécutée, alors que les deux instructions correspondantes figurent pourtant dans le texte de la marche à suivre.

> On peut dire, en quelque sorte que, lorsque l'alternative est présente, il y a moins d'actions exécutées que d'instructions écrites. Si l'évaluation de la condition oblige l'exécutant à emprunter l'un des chemins, c'est bien sûr en négligeant la seconde voie qui était mentionnée dans le texte de la marche à suivre.

En général,




### 2.3.1 Représentation sous forme de pseudo-code

La structure alternative s'écrira :

| | | |
|---|---|---|
| **SI** condition **ALORS** | ou | **SI** condition **ALORS** |
| A | | A |
| **SINON** | | suite |
| B | | |
| suite | | |

A et B représentant des "morceaux de marche à suivre".

Les deux "blocs" représentés par A et B ne sont donc pas forcément des instructions uniques. Chacun de ces blocs peut être constitué de plusieurs instructions liées par une séquence ou même comporter d'autres structures de contrôle (comme ,par exemple, l'alternative).

Ainsi, la marche à suivre suivante :

*Procédez comme suit :*
- *décrochez le combiné ;*
- *à la réception de la tonalité à transmettre, formez : 9080XXXX 0*
  *les X représentent 4 chiffres formant l'heure de réveil, c'est-à-dire*
  *2 chiffres pour l'heure et*
  *2 chiffres pour les minutes*

*Exemples :*
*0720 sept heures vingt le matin,*
*1930, sept heures et demie le soir.*
*Il faut donc envoyer neuf chiffres.*
*Si l'ordre a été composé correctement , le système central de réveil vous informe de la suite réservée à votre demande sous forme d'un message parlé.*
*Trois cas peuvent se présenter :*
*L'heure de réveil est correcte et la période de réveil choisie n'est pas saturée.*
*Vous entendez le message suivant :*
*"Service réveil*
*Votre ordre est accepté"*
*Vous pouvez raccrocher.*
- *L'heure de réveil est incorrecte (par ex. 3810).*
- *L'heure de réveil est incomplète (par ex. 072).*
- *L'ordre ne peut être enregistré pour des raisons techniques.*

*Vous entendez le message suivant :*
*"Votre ordre n'est pas accepté".*
*Raccrochez et recommencez la télécommande.*
*La période de réveil choisie est saturée.*
*Vous entendez le message suivant :*
*"La période est saturée.*
*veuillez choisir une autre période de réveil"*
*Raccrochez et refaites une télécommande.*
*Pour être certain du bon aboutissement de votre demande, vous devez donc attendre la diffusion du message parlé.*
*Remarques :*
- *La communication est interrompue automatiquement, lorsque l'ordre de réveil n'a pas été composé correctement (entre autres quand un nombre supérieur ou inférieur aux 9 chiffres prescrits a été introduit);*
- *Si vous vous heurtez plusieurs fois à un refus, adressez-vous au service de réveil manuel (numéro 999).*

*Marche à suivre n° 6*

peut se réécrire, en faisant apparaître les structures alternatives et en ne retenant que les instructions d'action élémentaire (à l'exclusion des exemples et des commentaires) :

*Décrochez le combiné.*
*Formez 9080XXXXO.*
***SI** l'ordre a été composé correctement **ALORS***
    ***SI** l'heure de réveil est correcte et la période de réveil choisie n'est pas saturée **ALORS***
        *(vous entendez "service réveil, votre ordre est accepté")*
        *vous pouvez raccrocher*
    ***SI** l'heure de réveil est incorrecte OU l'heure de réveil est incomplète OU l'ordre ne peut être enregistré pour des raisons techniques **ALORS***
        *(vous entendez "votre ordre n'est pas accepté")*
        *Raccrochez.*
        *Recommencez la télécommande.*
    ***SI** la période de réveil choisie est saturée **ALORS***
        *(vous entendez "La période est saturée. Veuillez choisir une autre période de réveil")*
        *Raccrochez.*
        *Faites une autre télécommande.*

On le voit, la structure générale de cette marche à suivre est :

Séquence

*Décrochez le combiné*
*Formez 9080XXXX0*
***SI** l'ordre a été composé correctement **ALORS***
    *A*

ou A est un "bloc", un morceau de marche à suivre comportant à son tour plusieurs structures alternatives.

Dans la représentation sous forme de pseudo-code de la structure alternative, telle qu'elle est adoptée ici, il est impératif de mettre en évidence quels sont les blocs présents. C'est l'**indentation** (= le recul de parties du texte par rapport à la marge gauche) qui va permettre d'en décider.

Ainsi, dans la marche à suivre

*SI le demandeur n'est pas en règle ALORS*
*Lui faire remplir le formulaire rose B12*
*SINON*
*Reprendre la teneur de sa demande sur le formulaire jaune C15*
*Compléter le formulaire vert A24.*
*Renvoyer le dossier au bureau central*

Il est clair que le formulaire rose doit être complété lorsque le demandeur n'est pas en règle et que le formulaire jaune doit être complété dans le cas contraire. Mais, faut-il compléter le formulaire vert, quel que soit l'état du demandeur ou seulement lorsqu'il est en règle. En d'autres termes, quelles sont les instructions d'action sur lesquelles porte le SINON et où la suite reprend-elle ?

L'écriture avec indentation va permettre d'en décider :

*SI le demandeur n'est pas en règle ALORS*
⇨ *Lui faire remplir le formulaire rose B12*
*SINON*
⇨ *Reprendre la teneur de sa demande sur le formulaire jaune C15*
*Compléter le formulaire vert A24*
*Renvoyer le dossier au bureau central.*

Ainsi, clairement, le formulaire C15 est à compléter, lorsque le demandeur est en règle et **quel que soit son état (dans tous les cas), le formulaire vert doit être rempli**.

Si le texte avait été

*SI le demandeur n'est pas en règle ALORS*
⇨ *Lui faire remplir le formulaire rose B12*
*SINON*
⇨ *Reprendre la teneur de sa demande sur le formulaire jaune C15*
⇨ *Compléter le formulaire vert A24*
*Renvoyer le dossier au bureau central*

alors, le formulaire vert ne devait être complété **que pour les demandeurs en règle**.

Ainsi, l'indentation de parties du texte, outre qu'elle facilite la lecture, est dans certains cas indispensable pour la compréhension de ce qui est demandé.

Certains lèvent l'ambiguïté de manière différente, en notant explicitement la fin de la structure alternative par une écriture du type :

SI condition ALORS
A
SINON
B
FIN DU SI
suite

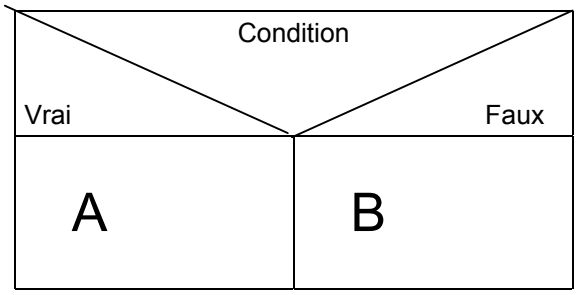
ou

SI condition ALORS
A
FIN DU SI
suite

Dans ce cas, l'indentation de certaines parties de la marche à suivre n'est plus nécessaire pour lever les ambiguïtés et son seul objet est d'en faciliter la lecture.

### 2.3.2 Représentation sous forme de GNS

L'alternative se représente par le schéma :




Le bloc noté B peut être vide (c'est le cas où il n'y a pas de SINON).

Comme plus haut, chacun des blocs notés A ou B est un morceau de marche à suivre (sous forme GNS) qui peut comporter une ou plusieurs instructions d'action élémentaire organisées par n'importe quelle structure de contrôle.

### 2.3.3 Représentation en Pascal

L'alternative se traduira par :

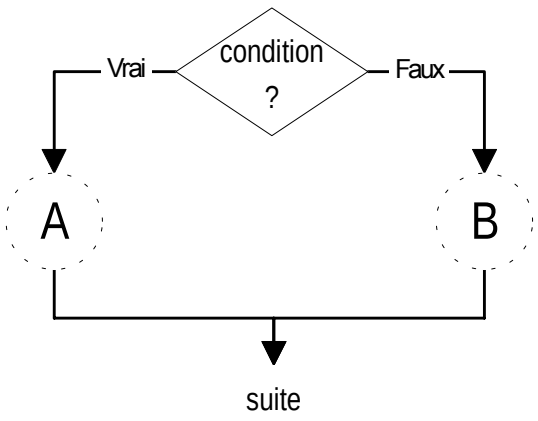
```
IF condition THEN
     A
ELSE
     B
suite
```

ou

```
IF condition THEN
     A
suite
```

On le voit, on est très proche de l'expression sous forme de pseudo-code.

### 2.3.4 Représentation en organigramme

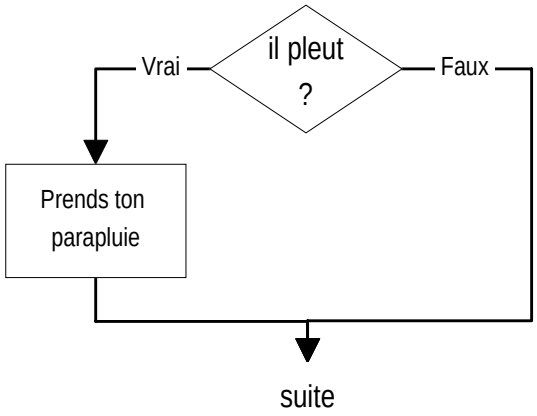



La condition est enfermée dans un losange, deux voies étant ensuite possibles : celle qui mène au morceau A, l'autre qui emprunte le "chemin" B (ce dernier bloc pouvant être inexistant, s'il n'y a pas de SINON).

Ici aussi, les parties notées A et B peuvent en général être constituées de plusieurs instructions élémentaires organisées par n'importe quelle structure de contrôle. Autrement dit, A comme B sont, dans ce cas, des morceaux d'organigramme.

Si A (ou B) était constitué d'une seule instruction d'action élémentaire, je l'aurais évidemment représenté enfermé dans une boîte rectangulaire, comme dans




### 2.3.5 Représentation en Basic

Dans le cas des Basic "pauvres", la seule structure existante est

```
IF condition THEN A
```

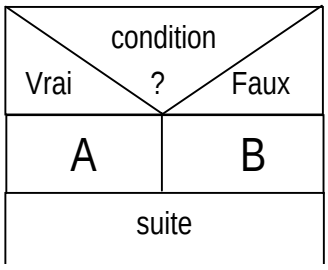
Plusieurs remarques s'imposent :

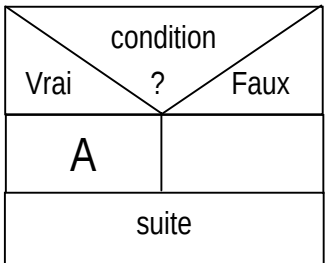
- il n'y a pas de traduction du SINON;
- le bloc noté A doit, en général, être de petite taille;
- dans certains Basic (les très "pauvres"), A ne peut être qu'une structure de branchement (Aller en ...).

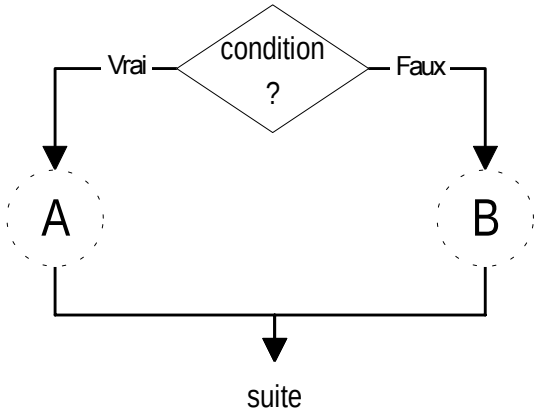
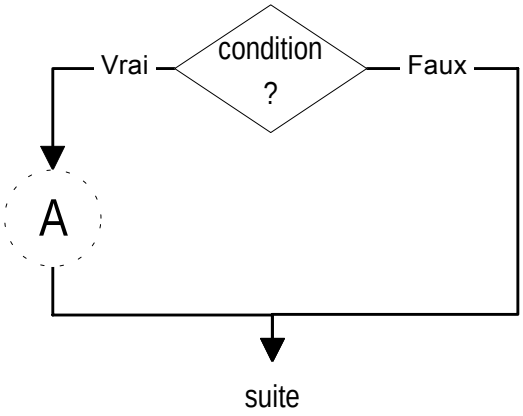
Toutes ces restrictions amèneront des problèmes lors du codage des marches à suivre en Basic (étape du "Comment dire ?").

> Je ne m'étendrai pas sur les problèmes soulevés par les structures permises par Basic; d'une part, l'évolution de ce langage (comme d'autres) l'a conduit à adopter des structures de contrôle plus évoluées (qui l'amènent à ressembler, de ce point de vue, à Pascal); d'autre part, beaucoup de programmeurs ont délaissé ces versions pauvres (et anciennes) de Basic au profit de langages plus riches. De toute manière, le lecteur intéressé par la traduction d'une analyse structurée dans un langage qui ne l'est pas, pourra consulter [28].

En résumé, pour l'alternative :

| Pseudo Code | GNS | Pascal |
|---|---|---|
| **SI** condition **ALORS**<br>A<br>**SINON**<br>B<br>suite<br><br>ou |  <br>ou | **IF** condition **THEN**<br>A<br>**ELSE**<br>B<br>suite<br><br>ou |

| | | |
|---|---|---|
| SI condition ALORS<br>A<br>suite |  | `IF condition THEN`<br>`A`<br>`suite` |

| Organigramme | Basic |
|---|---|
|  <br>ou<br>  | ou<br>`100 IF condition THEN A`<br>`110 suite` |

## *2.4 La répétition*

J'ai mis en évidence précédemment les deux formes (REPETER ... JUSQU'A CE QUE ... et TANT QUE ...) sous lesquelles se traduira la répétition. Ici aussi la correspondance entre le texte de la marche à suivre et son exécution est brisée.

Ainsi,

| Texte de la marche à suivre | Lors de l'exécution |
|---|---|
| | **L'exécutant** |
| *Mettre un décilitre d'eau dans le plat* | • met un décilitre d'eau dans le plat |
| ***REPETER*** | **puis** |
| *Ajouter une cuillerée de farine* | • ajoute une cuillerée de farine |
| | **puis** |
| *Mélanger* | • mélange |
| | **puis** |
| | • ajoute une cuillerée de farine |
| | **puis** |
| | • mélange |
| | **puis** |
| | • ajoute une cuillerée de farine |
| | **puis** |
| | • mélange |
| | ... ... |
| ***JUSQU'A CE QUE*** *la pâte soit onctueuse* | **puis** |
| *Laisser reposer un quart d'heure* | • laisse reposer un quart d'heure |
| | **puis** |
| *Incorporer une pincée de sel* | • incorpore une pincée de sel |
| | **puis** |
| *Goûter* | • goûte |
| ***TANT QUE*** *le mélange n'est pas assez salé* | **puis** |
| *Incorporer une pincée de sel* | • incorpore une pincée de sel |
| | **puis** |
| *Goûter* | • goûte |
| | **puis** |
| | • incorpore une pincée de sel |
| | **puis** |
| | • goûte |
| | **puis** |
| | ... ... |
| | **puis** |
| *Ajouter les oeufs* | • ajoute les oeufs |

On le voit, dans le cas de la structure répétitive, le nombre d'actions exécutées est en général bien plus élevé que le nombre d'instructions écrites, puisque les actions commandées sont de fait **répétées**.

En général,

| Texte de la marche à suivre | Lors de l'exécution |
|---|---|
| | **L'exécutant** |
| *Instruction d'action 1* | • action 1 |
| ***REPETER*** | **puis** |
| *Instruction d'action 2* | • action 2 |
| | **puis** |
| *Instruction d'action 3* | • action 3 |
| | **puis** |
| | • action 2 |
| | **puis** |
| | • action 3 |
| | **puis** |
| | • action 2 |
| | **puis** |
| | • action 3 |
| | ... ... |
| ***JUSQU'A CE QUE*** *condition* | **puis** |
| *Instruction d'action 4* | • action 4 |

Et la même rupture de parallélisme est de mise avec les mots TANT QUE ...

Il faut espérer que lorsque la marche à suivre commande

**REPETER**

...

**JUSQU'A CE QUE** condition,

la condition énoncée finisse par devenir vraie. Sinon, c'est à une répétition proprement interminable qu'on assisterait.

Si je commande

**REPETER**

Fais un pas en avant

Fais un pas en arrière

**JUSQU'A CE QUE** tu touches le mur,

cela risque évidemment de durer (sauf si l'exécutant touchait déjà le mur au départ, auquel cas, le premier pas en avant exigé va le faire s'y écraser !)

La même remarque s'applique à la structure TANT QUE, si j'écris

**TANT QUE** le mélange n'est pas mousseux

Battre

et que je place un bol d'eau sous le fouet de mon exécutant-cuisinier, cela aussi prendra un certain temps ...

Le "désordre" introduit dans les actions commandées par la structure répétitive doit être souligné. Ainsi, la dernière action du groupe dont on demande la répétition précède immédiatement la première de ce groupe (à cause de la répétition).

### 2.4.1 Représentation sous forme de pseudo-code

Les deux formes de répétition s'écrivent :

**REPETER** et **TANT QUE** condition **FAIRE**

A A

**JUSQU'A CE QUE** condition suite

suite

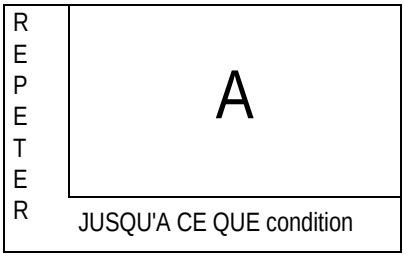
A étant, à nouveau, un "morceau de marche à suivre".

Ces deux expressions de la structure répétitive ne sont évidemment pas équivalentes (Cf. à ce propos l'exercice 3.2. ci-après).

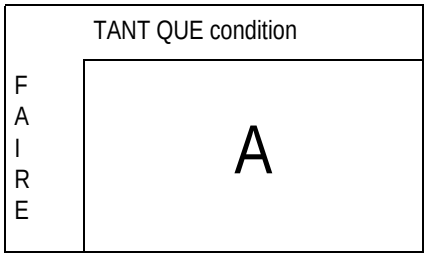
On remarquera à nouveau l'écriture avec indentation qui facilite la lecture dans le cas de la structure REPETER ... JUSQU'A et qui est indispensable pour identifier le bloc d'actions à répéter et la suite dans le cas du TANT QUE.

### 2.4.2 Représentation sous forme GNS

Les deux représentations sont




et




Remarquons que le bloc noté A est, une fois de plus, tout un morceau de marche à suivre, exprimé sous la forme d'un GNS.

### 2.4.3 Représentation en Pascal

On trouve

```
REPEAT
    A
UNTIL condition
suite
```

et

```
WHILE condition DO
    A
Suite
```

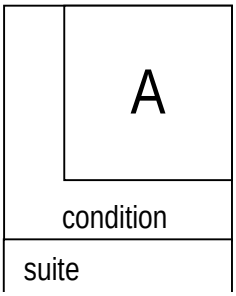
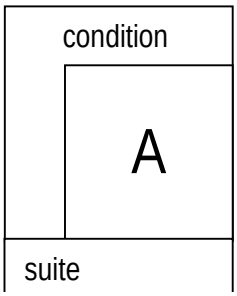
### 2.4.4 Représentation en organigramme

Il n'existe pas de graphisme spécifique traduisant les structures répétitives. (Cf. exercices 3.2. et 3.3. ci-dessous).

### 2.4.5 Représentation en Basic

Les versions pauvres ne comportent pas d'instructions incarnant la répétition.

En résumé,

| Pseudo Code | GNS | Pascal |
|---|---|---|
| **REPETER**<br>A<br>**JUSQU'A CE QUE** condition<br>suite<br><br>ou<br><br>**TANT QUE** condition<br>A<br>suite |  <br>ou<br>  | **REPEAT**<br>A<br>**UNTIL** condition<br>suite<br><br>ou<br><br>**WHILE** condition **DO**<br>A<br>suite |

| Organigramme | Basic |
|---|---|
| Rien | Rien |

Comme pour la structure alternative, on se heurtera donc à quelques problèmes, lors du codage en Basic des structures répétitives, puisque ce langage (dans ses versions pauvres) n'offre pas de traduction immédiate de ces structures.

## *2.5 L'appel de procédure*

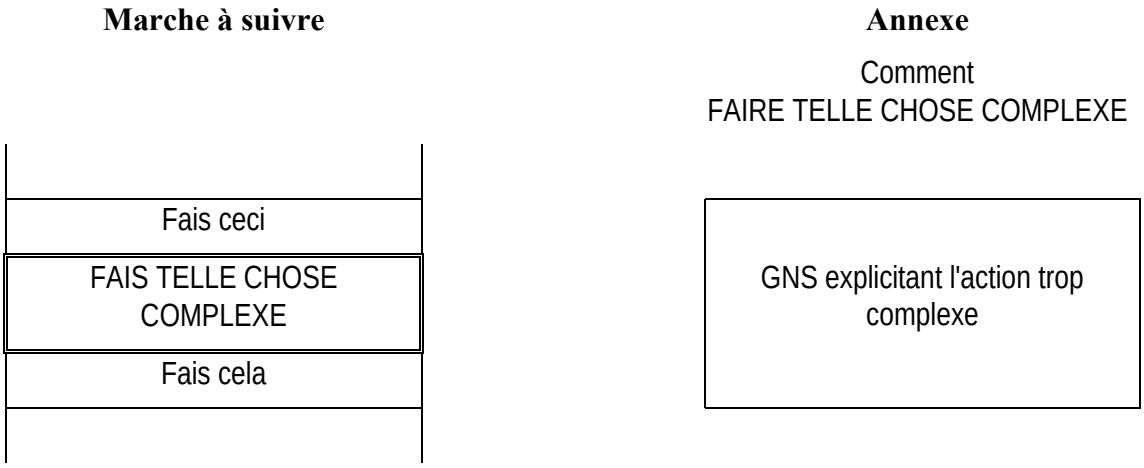
Il s'agissait, je le rappelle, d'une instruction d'action complexe, laissée telle quelle dans le texte de la marche à suivre, mais assortie de l'explicitation indispensable au niveau d'une marche à suivre annexe.

Je ne dirai presque rien pour l'instant des divers modes de représentation de cette structure, sauf en ce qui concerne les GNS.

Simplement, le rectangle comportant une instruction trop complexe sera dédoublé et un GNS annexe reprendra l'explication de cette instruction complexe

Par exemple :




J'ajouterai que cette structure d'appel de procédure est omniprésente en Pascal et ... pratiquement absente dans les deux représentations cousines que sont organigramme et langage Basic.

Les adeptes de Basic rétorqueront, sans doute, que l'appel de procédure existe bel et bien dans ce langage sous la forme de l'instruction GOSUB ... . Sans entrer dans les détails, je dirai simplement qu'en Basic :

- les procédures ne constituent pas véritablement des annexes au programme principal, mais en font partie;
- l'instruction d'appel de procédure GOSUB envoie, comme le branchement (GO TO ...), à un autre endroit du programme, au début de la procédure et, une fois les actions commandées par celle-ci effectuées, l'exécutant revient à l'instruction qui suit immédiatement le GOSUB qui l'avait, un moment, détourné vers la procédure. Pour faire bref, l'appel de procédure en Basic, c'est un branchement avec mémorisation de l'endroit d'où le saut est fait, pour pouvoir y revenir.

L'appel de procédure est l'une des structures les plus importantes en programmation. C'est elle qui permet d'incarner la démarche descendante (dont je reparlerai). C'est aussi l'une de celles pour laquelle les détails techniques (variables globales et locales, paramètres ...) seront les plus difficiles à appréhender. Je n'en dis donc pas plus pour l'instant.

## *2.6 Le branchement*

Il s'agit de ce mode d'organisation des marches à suivre qui brise le déroulement séquentiel en "envoyant" l'exécutant à un autre endroit de la marche à suivre. C'est la structure traduite par l'indication :

**ALLEZ EN ...**

Le branchement est souvent employé combiné à l'alternative pour prendre la forme :

**SI ... ALORS ALLEZ EN ...**

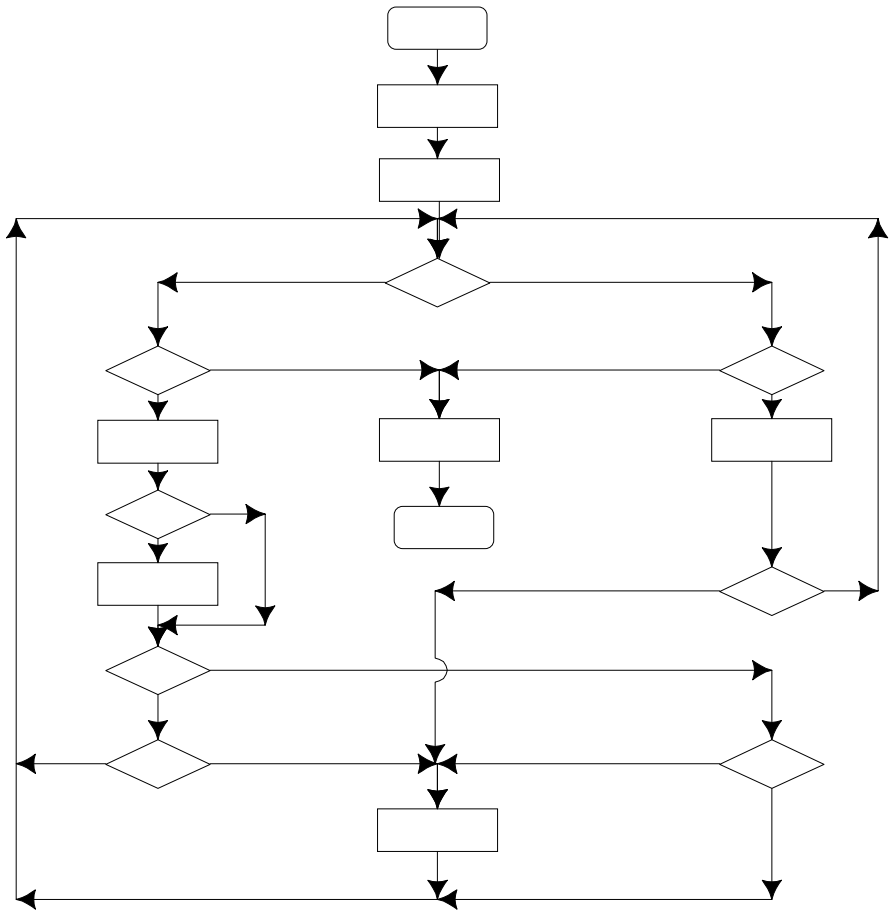
qu'on appelle parfois branchement conditionnel.

Le branchement a fréquemment été dénoncé comme un outil de "désorganisation" des marches à suivre et sa condamnation a coïncidé avec la prise de conscience des nécessités d'une programmation structurée.

Ce débat est, à présent, dépassé. Il a cependant permis de prendre conscience qu'un certain nombre de principes doivent être respectés si l'on souhaite écrire des programmes corrects, clairs, compréhensibles et modifiables.

En ce qui concerne la présentation adoptée ici, le branchement ne nous servira pas en tant que structure de contrôle. Il ne possède d'ailleurs pas de représentations sous forme GNS. Il existe bien en Pascal (traduit par les instructions GO TO ...) mais y est relativement peu employé.

Par ailleurs, il est omniprésent dans les organigrammes et en Basic, qui est le langage dans lequel vont s'incarner les structures présentes dans ces organigrammes. En voici un exemple particulièrement représentatif. Le contenu des rectangles et losanges est sans importance. Ce qu'il faut percevoir, ce sont tous les "parcours fléchés" qui renvoient d'un coin à l'autre de la marche à suivre rendant peu visible ce que sera le déroulement d'une exécution.



J'aurai pu présenter, de la même manière, quelques exemples de programmes Basic, mentionnant l'usage immodéré du branchement et du branchement conditionnel. Je pense que s'il fut un temps où il fallait convaincre de la nocivité du branchement comme outil de pensée et d'organisation, cette époque est, heureusement, révolue. Et puis, il est toujours délicat de présenter des exemples de ce qu'il ne faut pas faire !

## 3. Exercices

1. Pouvez-vous mettre en évidence, dans les exemples de marches à suivre présentées au début de ce chapitre :
   - des instructions d'action élémentaire,

- des structures de contrôle.

  Exprimez, sous forme GNS, les structures de contrôle détectées.

2 Voici, sous forme GNS et pseudo-code, une courte marche à suivre :

| Entrez |
|---|
| Buvez un petit verre |
| JUSQU'A CE QUE vous voyiez double |
| Asseyez-vous |

```
Entrez
REPETER
        Buvez un petit verre
JUSQU'A CE QUE vous voyiez double
Asseyez-vous
```

a) Pouvez-vous la transcrire dans le formalisme des organigrammes ?

b) Réécrivez-la sous forme GNS (ou pseudo-code), en utilisant la structure "TANT QUE ..." et toute autre structure de contrôle à l'exclusion de "REPETER ... JUSQU'A CE QUE..." .

Je ne vous demande, ni de juger du bien fondé ou de la moralité de cette marche à suivre, ni de l'améliorer, mais seulement de la transcrire sous une autre forme. Les exécutions de la marche à suivre proposée et des transcriptions que vous suggérerez doivent être identiques, dans tous les cas.

3. Voici, sous forme GNS et pseudo-code, une marche à suivre :

| Entrez |
|---|
| TANT QUE les élèves veulent travailler |
| Donnez 5 minutes de cours |
| Sortez |

```
Entrez
TANT QUE les élèves veulent travailler
        Donnez 5 minutes de cours
Sortez
```

a) Exprimez-la dans le formalisme des organigrammes.

b) Fournissez, sous forme GNS (ou pseudo-code), une marche à suivre équivalente (= conduisant à des exécutions identiques), sans employer la structure "TANT QUE...".

4. Vous souhaitez faire marcher, jusqu'à ce qu'il touche un mur, un exécutant borné qui est seulement capable de faire un pas, de s'arrêter et de se rendre compte qu'il touche (oui ou non) le mur à atteindre.

a) Ecrivez des marches à suivre pour cet exécutant sous forme

- d'organigramme;
- d'un programme "à la Basic";
- d'un GNS;
- d'un programme "à la Pascal".

b) Que pensez-vous de la solution suivante ?

| FAIS TOUT CE QU'IL FAUT POUR ATTEINDRE LE MUR |
|---|

Comment "FAIRE TOUT CE QU'IL FAUT POUR ATTEINDRE LE MUR"

| tu touches le mur ? V | tu touches le mur ? F |
|---|---|
| Arrête | Fais un pas |
| | FAIS TOUT CE QU'IL FAUT POUR ATTEINDRE LE MUR |

ou, sous forme pseudo-code

| FAIS TOUT CE QU'IL FAUT POUR ATTEINDRE LE MUR |
|---|

Comment "FAIRE TOUT CE QU'IL FAUT POUR ATTEINDRE LE MUR"

SI tu touches le mur ALORS

    Arrête

SINON

    Fais un pas

    FAIS TOUT CE QU'IL FAUT POUR ATTEINDRE LE MUR

## 4. Retour sur les deux formes de structure répétitive

Nous avons retenu deux expressions différentes pour commander la répétition au sein des marches à suivre. Il s'agit de :

**REPETER** et **TANT QUE** ... **FAIRE**

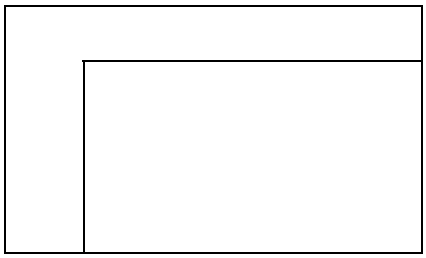
... ...

**JUSQU'A CE QUE** ... suite

suite

représentées graphiquement sous forme GNS par

et



Le graphisme employé permet, à lui seul, de rendre compte des structures évoquées, et, dorénavant, j'omettrai d'y faire figurer explicitement les mots "REPETER ... JUSQU'A CE QUE ..." ou "TANT QUE ..."

Ces deux modes d'expression ont bien entendu des rapports que je vais à présent éclairer. Les exercices 2 et 3 ci-dessus avaient d'ailleurs pour objectif de préparer l'examen de ces liens et, en y apportant une solution, je préciserai ce qu'il faut en retenir.

Une première marche à suivre se présentait comme suit :

| Entrez |
|---|
| Buvez un petit verre |
| JUSQU'A CE QUE vous voyiez double |
| Asseyez-vous |

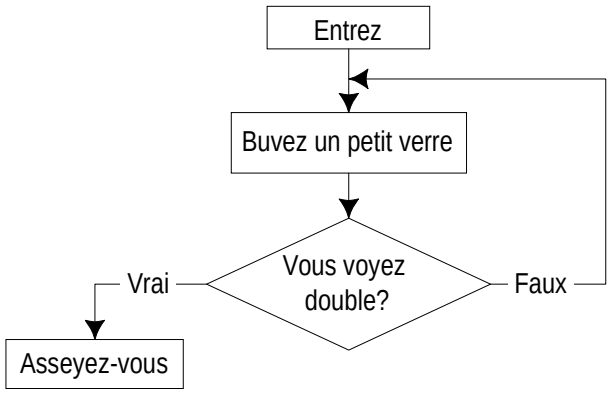
Entrez
REPETER
    Buvez un petit verre
JUSQU'A CE QUE vous voyiez double
Asseyez-vous

La traduction demandée sous forme d'organigramme suppose qu'on ait pris conscience de deux choses :

- la condition énoncée dit quand la répétition s'arrêtera (on arrêtera de boire lorsqu'on verra double);
- l'énoncé de cette condition **suit** la précision des actions à répéter.

Dès lors, dans le formalisme des organigrammes, on pourrait, par exemple, proposer :



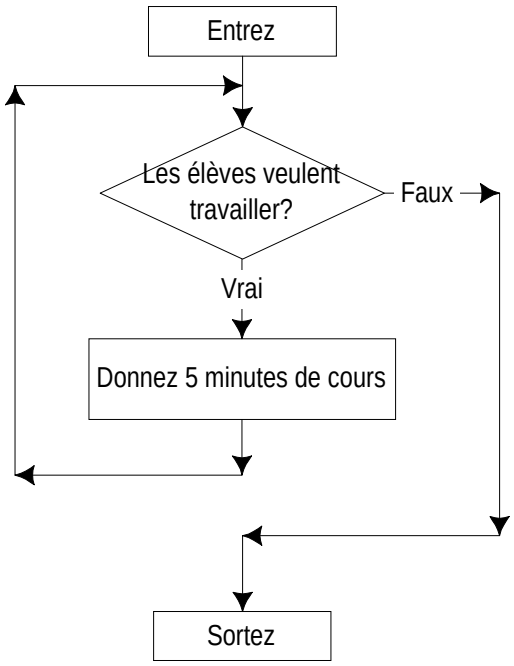

La caractéristique importante de la structure **REPETER ... JUSQU'A CE QUE ...** est bien mise en évidence par ce formalisme : dans tous les cas, l'action (ou plus généralement, le morceau de marche à suivre) à répéter, est effectuée au moins une fois avant que ne soit posée la question de savoir si cela doit être repris. Ainsi, dans l'exemple, l'exécutant aura dans tous les cas droit à un premier verre et c'est seulement ensuite qu'il trouvera la question lui demandant s'il voit double. Dès lors, même dans l'hypothèse où l'exécutant voit double dès son entrée, la marche à suivre lui commande de prendre tout de même un verre avant qu'il ne s'interrompe et s'asseye en rencontrant la condition d'arrêt.

L'exercice 3.3 proposait une marche à suivre qui mettait aussi en jeu une structure répétitive, mais exprimée grâce à un **TANT QUE**.

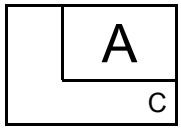
| |
|---|
| Entrez |
| TANT QUE les élèves veulent travailler |
| Donnez 5 minutes de cours |
| Sortez |

```
Entrez
TANT QUE les élèves veulent travailler
        Donnez 5 minutes de cours
Sortez
```

Remarquons, d'abord, qu'ici les actions à répéter sont reprises lorsque la condition énoncée est vraie (c'est l'inverse du REPETER, où l'on trouve derrière les mots JUSQU'A CE QUE la condition qui assure l'arrêt de la répétition). Ensuite, et la traduction sous forme d'organigramme va l'éclairer, on commence par poser la question avant même d'entamer une première fois le groupe d'actions à répéter. Ainsi, l'organigramme correspondant s'écrit :




En résumé, si la structure est



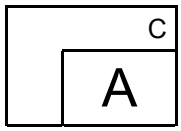

REPETER
A
JUSQU'A CE QUE C

A représentant le morceau de marche à suivre à répéter

et C la condition énoncée[2],

alors **le bloc A est toujours effectué au moins une première fois**, avant que l'évaluation de la condition C fasse reprendre ou oblige à interrompre.

Si la structure est :




TANT QUE C
A

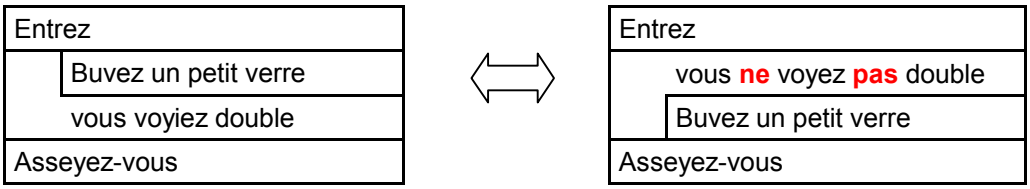
alors la condition est d'abord évaluée. Si elle est fausse, **le bloc A n'est pas effectué une seule fois** : on passe directement à ce qui suit; si elle est vraie, on effectue le bloc A et on revient à l'évaluation de la condition pour savoir si la répétition doit être poursuivie.

Ces remarques étant faites, nous sommes en mesure de transformer la structure REPETER ... JUSQU'A CE QUE ... en TANT QUE ... et réciproquement.

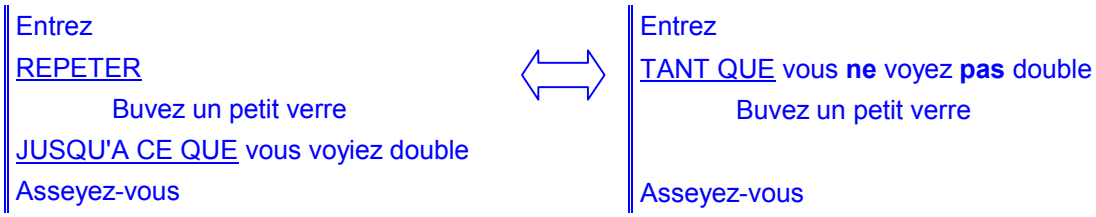
Revenant à l'exercice 3.2, on pourrait proposer l'équivalence :

[2] Replacez bien à chaque fois, mentalement, les mots REPETER ..., JUSQU'A CE QUE ..., TANT QUE ... suggérés par le graphisme des GNS, là où ils interviennent.



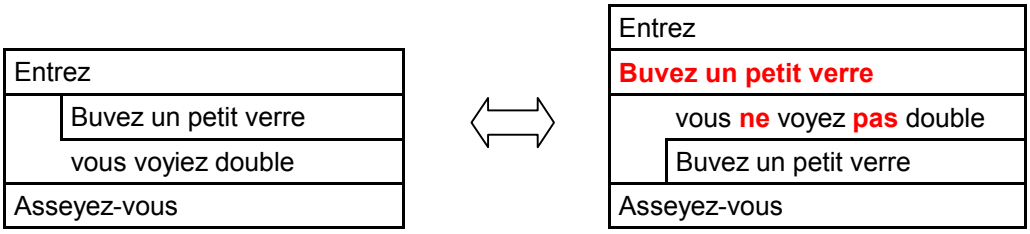

ou



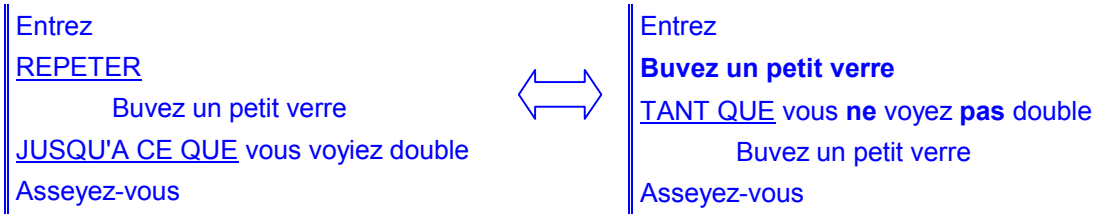

Le seul changement perceptible est la négation de la condition (puisque derrière JUSQU'A CE QUE, on énonce la condition qui provoque l'arrêt et, derrière TANT QUE, celle qui assure la reprise de la répétition). Cette solution n'est malheureusement pas correcte : les deux marches à suivre ci-dessus ne sont pas équivalentes, car elles ne conduisent pas à des exécutions identiques dans tous les cas.

En effet, dans le cas où, dès son entrée, l'exécutant voit déjà double, la première (qui met en jeu la structure REPETER ...) l'oblige à tout de même prendre un verre; la seconde (exprimée grâce à un TANT QUE) va le conduire à s'apercevoir d'abord qu'il voit déjà double et donc ne lui permet pas de boire le moindre verre.

Si l'on veut une expression grâce à la structure TANT QUE qui soit, dans tous les cas, identique à la première, il faut écrire :

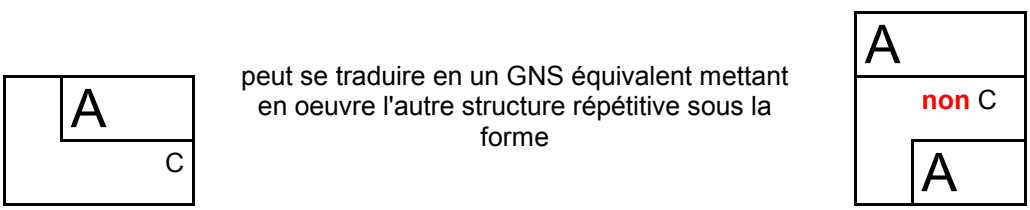



ou




On peut vérifier sans peine que ces deux marches à suivre conduisent à des exécutions identiques, quel que soit l'état initial de l'exécutant-buveur.

Ainsi donc, en général, le GNS




non C désignant la négation (l'assertion contraire) de l'assertion C.

Et, sous forme pseudo-code

```
REPETER
    A
JUSQU'A CE QUE C
```

peut se traduire en un équivalent mettant en oeuvre l'autre structure répétitive sous la forme

```
A
TANT QUE non C
    A
```

non C désignant la négation (l'assertion contraire) de l'assertion C.

J'utilise indifféremment les mots "condition" et "assertion". On pourrait peut-être aussi parler de "question".

Il s'agit là de facettes différentes d'une même réalité :

- une assertion dont l'exécutant peut évaluer la vérité;
- une condition qu'il peut tester;
- une question à laquelle il répond oui ou non.

Il faut signaler un corollaire important de cette possibilité de remplacer la structure REPETER .. JUSQU'A CE QUE par la structure TANT QUE ... Cette dernière suffira pour exprimer la répétition, puisque la constatation ci-dessus permet de substituer au REPETER ... JUSQU'A CE QUE, un GNS n'utilisant que la structure TANT QUE.

Intéressons-nous à présent au problème réciproque : comment substituer la structure REPETER ... JUSQU'A CE QUE ... à la structure TANT QUE ?

L'exercice 3.3 posait cette question à propos de la marche à suivre :

```
Entrez
TANT QUE les élèves veulent travailler
    Donnez 5 minutes de cours
Sortez
```

```
Entrez
TANT QUE les élèves veulent travailler
        Donnez 5 minutes de cours
Sortez
```

La proposition d'une solution sous la forme

```
Entrez
    Donnez 5 minutes de cours
  les élèves ne veuillent plus travailler
Sortez
```

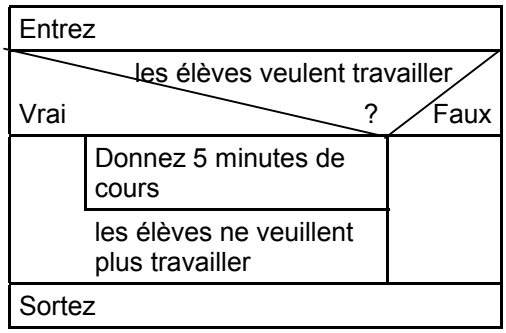
```
Entrez
REPETER
        Donnez 5 minutes de cours
JUSQU'A CE QUE les élèves ne veuillent plus
        travailler
Sortez
```

se heurte aux mêmes critiques que celles formulées précédemment : dans le cas où l'exécutant-professeur débarque dans une classe où les élèves en ont déjà assez avant qu'il ne commence son cours, la marche à suivre ci-dessus, utilisant le TANT QUE, l'empêche de débuter le cours, alors que celle faisant usage du REPETER l'oblige, même dans ce cas, à donner les 5 premières minutes avant de sortir.

On sent bien qu'une traduction convenable doit énoncer la condition avant que les actions à répéter ne soient proposées.

On peut, par exemple, écrire

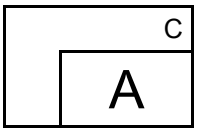
 



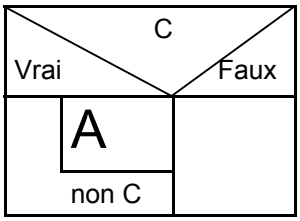

```
Entrez
SI les élèves veulent travailler ALORS
    REPETER
        Donnez 5 minutes de cours
    JUSQU'A CE QUE les élèves ne veuillent
                   plus travailler
Sortez
```

qui conduira, dans tous les cas à une exécution équivalente à celle proposée avec la structure TANT QUE.

Ainsi donc, le GNS




peut toujours s'exprimer de manière équivalente sous la forme :




Et, sous forme pseudo-code

```
TANT QUE C
    A
```

peut toujours s'exprimer de manière équivalente sous la forme :

```
SI C ALORS
    REPETER
        A
    JUSQU'A CE QUE non C
```

## 5. Exercices

1. Ecrire sous forme GNS chacune des marches à suivre ci-dessous. Dans le cas où des structures répétitives sont présentes, réécrire le GNS, en faisant usage de l'autre structure répétitive.

a)

```
Asseyez-vous au bureau
TANT QUE le tas de fiches à traiter n'est pas vide
        Prenez la fiche au-dessus du tas
        SI elle est jaune ALORS
                Placez-la à gauche du bureau
        SINON
                Placez-la à droite
Allez prendre une tasse de café
```

b)

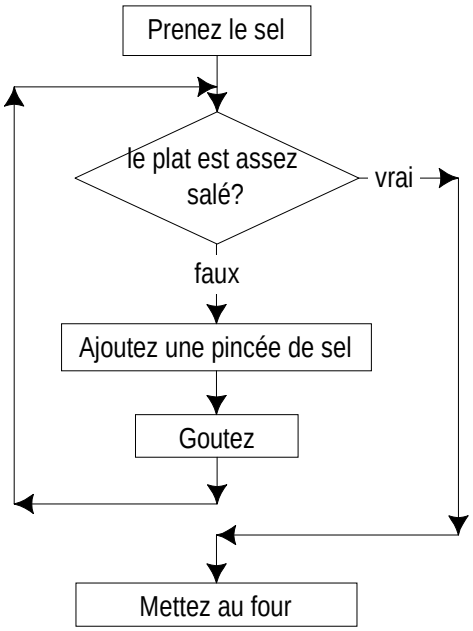
```
Lisez le premier nom de la liste
TANT QUE le nom n'est pas "DUPONT" et qu'il reste des noms à lire
        Lisez le nom
        SI le nom est "DUPONT" ALORS
                prenez note du n° de téléphone
        SINON
                passez au nom suivant sur la liste
SI vous avez trouvé "DUPONT" ALORS
        Téléphonez-lui
```

Traduire chacun des GNS proposé en organigramme.

*Notez bien qu'il n'est pas demandé de juger de la cohérence ou de l'opportunité des exemples proposés : il faut seulement les représenter sous forme GNS et non en "améliorer" la teneur !*

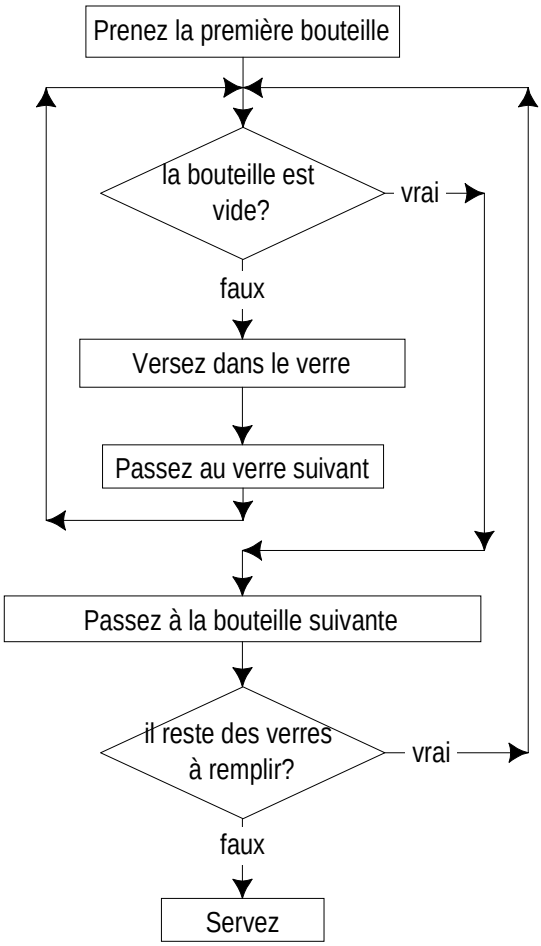
2. Pouvez-vous traduire sous forme GNS (ou en pseudo-code) les organigrammes suivants :

a)



b)

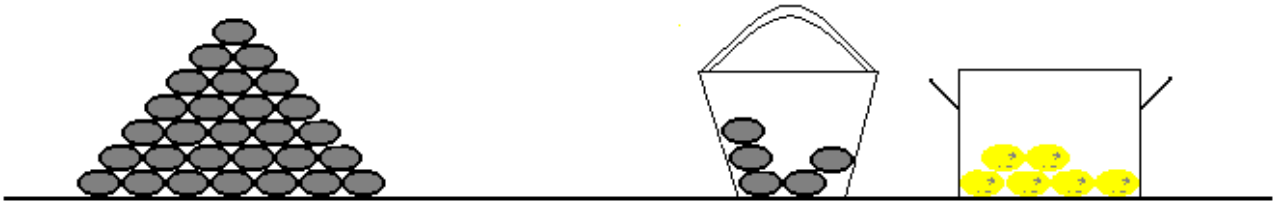



## 6. Et si l'on rédigeait des marches à suivre !

Nous avons, à présent, à notre disposition les outils nécessaires à la conception et à l'écriture de marches à suivre. Bien entendu, je n'ai pas encore brossé le portrait de l'exécutant-ordinateur (cela sera fait au chapitre suivant). Cependant, nous pouvons déjà "jouer" à commander d'autres exécutants, en commençant, par exemple, par :

### *6.1 Le robot peleur de pommes de terre*

Cet exécutant évolue dans un environnement schématisé comme suit :



Il dispose d'un panier qu'il peut aller remplir à un tas de pommes de terre. Ce tas est de taille suffisante pour satisfaire n'importe quelle commande : nous dirons qu'il est inépuisable. Le robot est aussi capable de prendre une pomme de terre dans le panier, de la peler et de la jeter dans la marmite.

Cet exemple n'est pas nouveau ! Il fait même en quelque sorte partie du folklore informatique et se trouve mentionné, dans des formes voisines de celle présentée ici, dans [23] ou [55].

Les conditions de travail de notre robot étant précisées, il reste à le caractériser en tant qu'exécutant : il est indispensable pour cela, d'indiquer quelles instructions d'action élémentaire vont le faire agir et quelles sont les conditions qu'il est capable de tester.

### 6.1.1 Les instructions d'action élémentaire

On peut en dresser la liste sous la forme d'un lexique :

| **Vous lui dites** | **Il fait** |
|---|---|
| Remplis | Il va remplir le panier et le ramène près de son lieu de travail. |
| Pèle | Il prend une pomme de terre dans le panier, la pèle et la place dans la marmite. |

Il est important de noter qu'il s'agit là des deux seules instructions qu'il "comprend" et est capable d'exécuter. Il est donc interdit de faire figurer dans la marche à suivre d'autres instructions que celles-là. Pas question donc de commander "vide le panier", "va chercher une pomme de terre au tas", ni même "recommence" ou "arrête de travailler."[3] On peut bien entendu employer la possibilité offerte par la structure d'appel de procédure, en laissant figurer dans la marche à suivre une instruction complexe, à condition qu'on puisse la décortiquer en termes d'instructions élémentaires, dans une annexe explicative.

### 6.1.2 Les conditions que l'exécutant peut tester

On le sait, ces conditions figurent au sein des structures alternatives et répétitives (derrière les mots SI, JUSQU'A CE QUE ou TANT QUE). Elles seront ici au nombre de deux :

la marmite est remplie;

le panier est vide.

On pourra donc faire figurer dans la marche à suivre les indications correspondant, par exemple, à

**SI** le panier est vide **ALORS** ...

**TANT QUE** la marmite est remplie ...

Il est de plus permis de former des conditions plus complexes en :

- énonçant le contraire des deux conditions ci-dessus ;
- en les liant par les mots "ET" et "OU".

On se permettra donc d'écrire :

**SI** la marmite n'est pas remplie **ET** le panier est vide **ALORS** ...

**TANT QUE** le panier **n**'est **pas** vide ...

Le but est évidemment de fournir au robot la marche à suivre, sous forme GNS (ou pseudo-code) , qui va lui permettre de peler la quantité de pommes de terre suffisante pour remplir la marmite.

J'ajoute que :

---

3 Il ne s'arrêtera qu'arrivé au terme du texte de la marche à suivre : tout en bas du GNS.

- vous ne pouvez rien supposer quant à l'état initial du panier lorsque le robot commencera l'exécution de votre marche à suivre : il est peut-être vide, peut-être plein, peut-être à moitié vide ...

  De plus, vous ne pouvez rien conjecturer non plus à propos de sa taille : il peut être tout petit (ne pouvant contenir qu'une pomme de terre) ou très grand !

- vous ne pouvez non plus présumer de la taille de la marmite à remplir, ni de la quantité de pommes de terre déjà pelées qui y est contenue. La marche à suivre doit être valable, qu'il s'agisse d'une énorme marmite, d'une marmite ne pouvant contenir qu'une seule pomme de terre ou même d'une marmite qu'on amène déjà à moitié ou complètement remplie (dans ce dernier cas, le robot ne doit évidemment rien peler !).

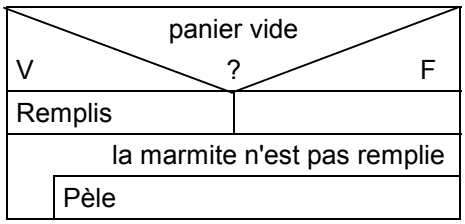
> Ceci constitue pour moi plus qu'un exemple folklorique ou amusant. Ce problème (car c'en est un !) nous plonge au coeur de la programmation ! Bien sûr le robot à commander n'est pas encore l'exécutant-ordinateur, manipulateur d'informations; même si la liste des actions élémentaires et des conditions va changer lorsque notre interlocuteur sera l'ordinateur, le problème de faire faire en fournissant une marche à suivre est, au fond, identique.
>
> Je suggère à tous les lecteurs qui seraient tentés de trouver peu sérieux des problèmes de ce type de s'atteler à la solution de celui proposé ci-dessus. Mon expérience m'a montré que, même parfois chez les gens qui ont une certaine pratique de la programmation (à travers, par exemple, la connaissance du langage Basic), sa résolution n'est pas toujours immédiate !

## *6.2 Examen de quelques marches à suivre pour le robot peleur de pommes de terre*

Nous allons, si vous le voulez bien, avant d'apporter une solution, au problème proposé, nous livrer à l'examen de quelques propositions de "solutions" en tentant d'imaginer les exécutions auxquelles conduisent chacune des marches à suivre ainsi présentées.

Et d'abord,

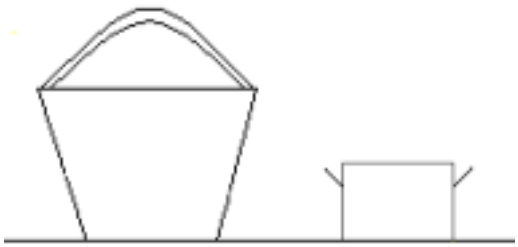
### 6.2.1 Proposition n°1




```
SI panier vide ALORS
        Remplis
TANT QUE la marmite n'est pas remplie
        Pèle
```

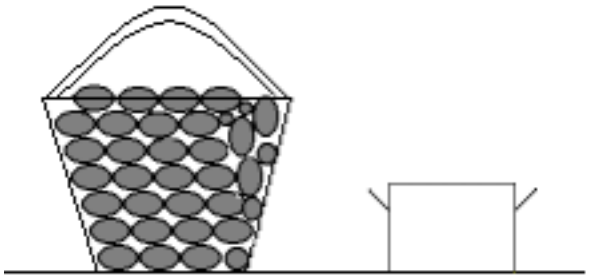
Voici une proposition qui va paraître satisfaisante ... dans le cas d'un gros panier vide et d'une petite marmite !

Imaginons, en effet, avec un panier et une marmite transparents pour les besoins de l'illustration, une situation telle que

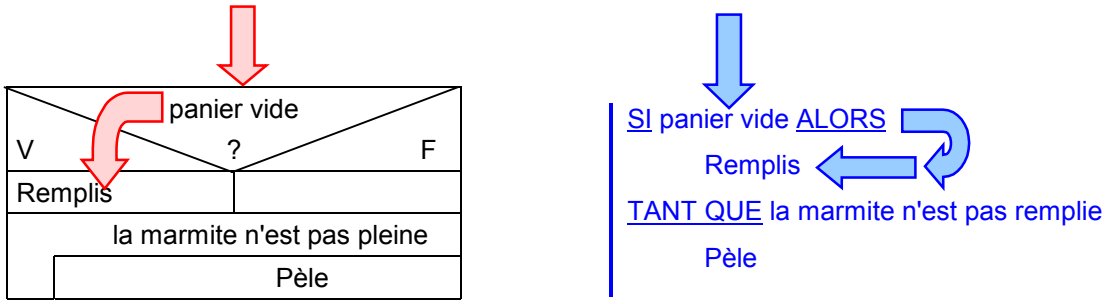


Le robot commence l'exécution de la marche à suivre. [4]

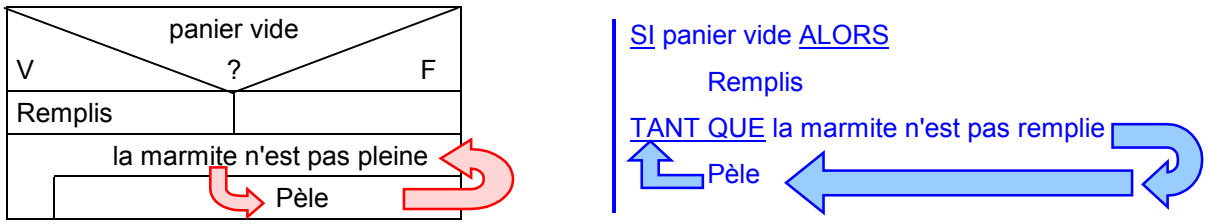
La condition "panier vide ?" étant vraie, on lui commande dans ce cas de remplir le panier.



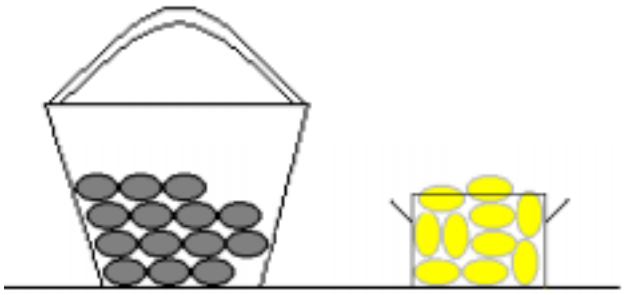
La traversée du GNS s'est donc, jusque là, opérée de la façon suivante :




La suite lui dit que "TANT QUE la marmite n'est pas pleine, il doit peler". C'est ce qu'il fait. Au bout de quelques passages dans cette boucle TANT QUE illustrés par le schéma




la situation devient :

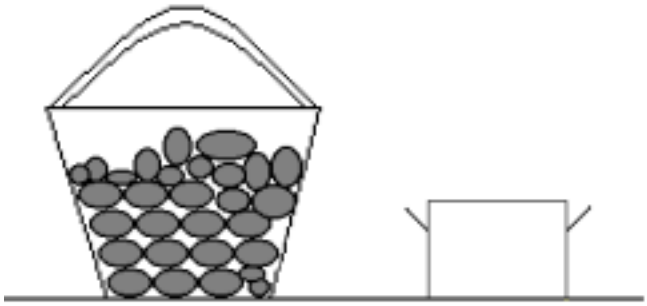


La marmite étant alors remplie, l'évaluation de la condition "la marmite n'est pas pleine" donne une réponse fausse et le robot s'arrête. Tout va bien : la quantité de pommes de terre souhaitée est atteinte ! On notera d'ailleurs que même si le panier n'était pas vide au début, mais

[4] Il est important de garder sous les yeux le GNS ou le pseudo-code correspondant.

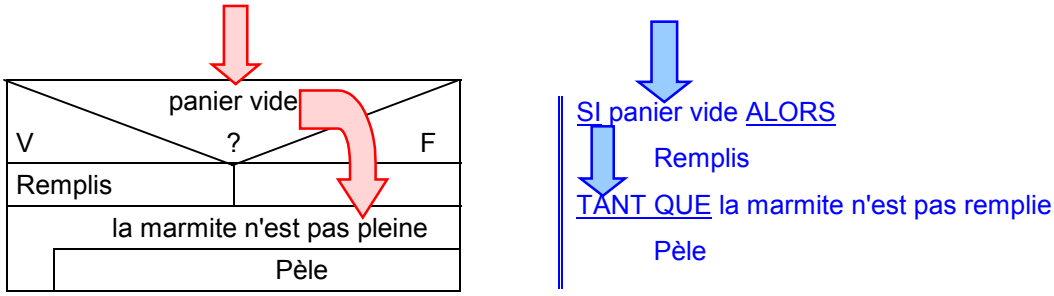
 

contenait assez de pommes de terre, tout continuerait à fort bien marcher. Si, par exemple, la situation initiale est



L'exécution se déroule comme suit :

Le panier n'étant pas vide, il ne doit pas être rempli, ce qui correspond à la traversée suivante du début du GNS



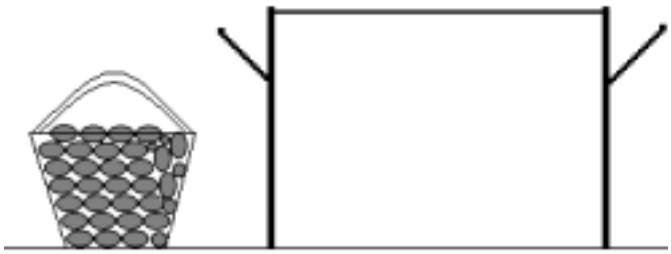

La boucle TANT QUE débute alors. La marmite n'étant pas pleine, l'exécutant pèle une pomme de terre, constate à nouveau qu'elle n'est pas pleine, pèle de nouveau et ainsi de suite, jusqu'à l'avoir rempli. Comme le panier contenait assez de pommes de terre pour ce remplissage, aucun problème ne se pose ! Il ne faut malheureusement pas conclure hâtivement, après ces deux "essais", que la marche à suivre proposée convient.

C'est le drame de la programmation :

**Essayer un programme peut éventuellement montrer qu'il ne marche pas, jamais prouver qu'il est correct !**

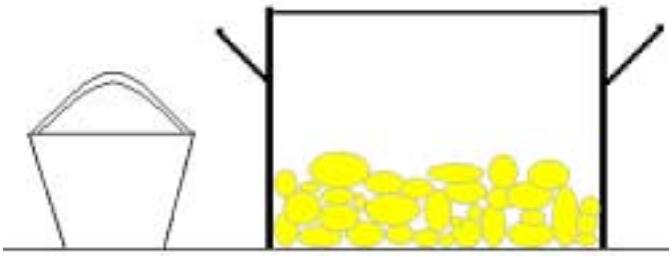
J'espère que vous avez déjà pressenti que, dans le cas d'un petit panier et d'une grosse marmite, les choses vont se passer nettement moins bien !

Si la situation initiale est :

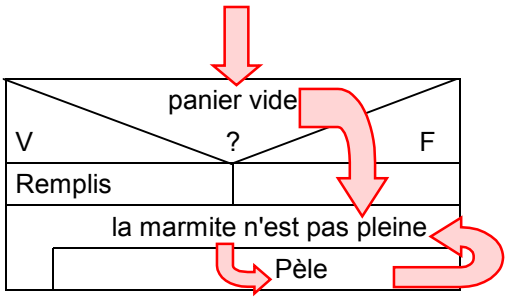


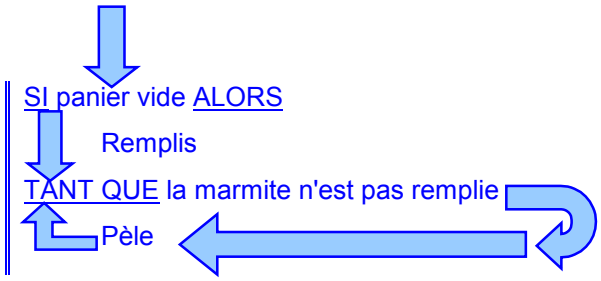
Le panier n'étant pas vide (il est même plein), l'alternative débutant le GNS le laisse inchangé. On entame donc la boucle TANT QUE. Comme la marmite n'est pas remplie, l'exécutant

pèle, constate qu'elle n'est pas toujours pleine, pèle à nouveau ... et cela, jusqu'à la situation où le panier finit par être vide...



correspondant à un parcours :




La marmite n'est toujours pas remplie. Or TANT QU'elle n'est pas pleine, la marche à suivre oblige l'exécutant à peler. Mais le panier est, à présent, vide et **rien** ne commande de le remplir ... et l'exécutant reste planté là, ne sachant que faire, puisqu'on continue à lui commander de saisir une pomme de terre dans un panier vide !

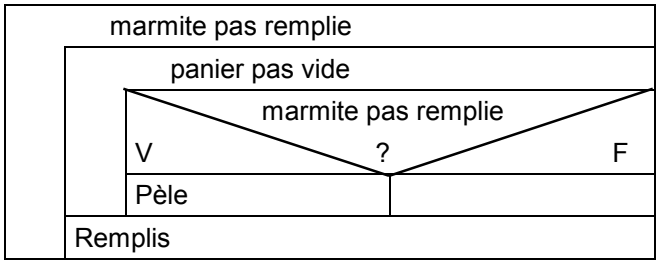
Voici donc un cas où la marche à suivre proposée ne conduit pas à une exécution réussie. Il suffit évidemment d'un seul cas malheureux pour que cette marche à suivre soit décrétée incorrecte !

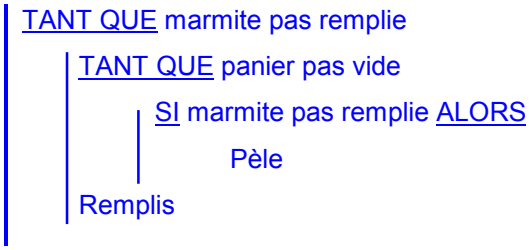
En effet, pour prouver qu'une marche à suivre est erronée, il suffit de mettre en évidence un seul exemple d'exécution insatisfaisante et c'est ce que je viens de faire.

Le problème crucial en programmation, ce sera de s'assurer que les marches à suivre conçues sont bien correctes, ce qui signifie qu'elles conduisent à des exécutions satisfaisantes **dans tous les cas imaginables** !

### 6.2.2 Proposition n°2

Voici un autre GNS :



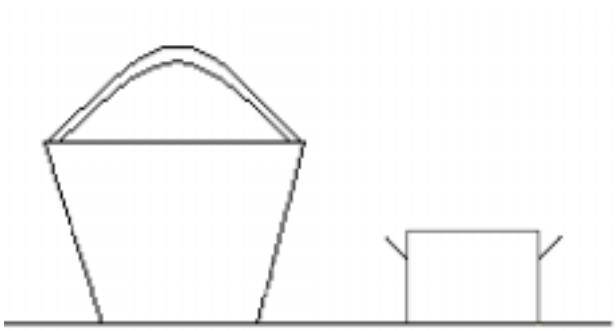

Je n'invente évidemment rien ! Tous ces exemples de marches à suivre sont des productions d'étudiants confrontés à ce problème. J'en tiens plusieurs dizaines d'autres à la disposition des incrédules !

Avec sous les yeux la marche à suivre ci-dessus, nous pouvons entamer la discussion de sa correction.

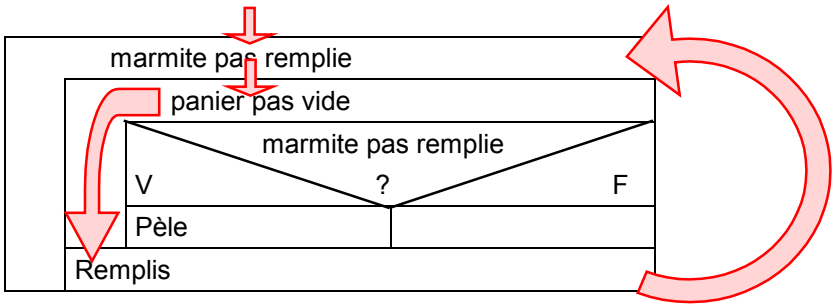
L'idéal serait de disposer vraiment de l'exécutant et de le regarder faire ..., tout en gardant un oeil (si je puis dire) sur le GNS qui est en train de le faire agir. Faute de ces images qui montreraient en parallèle le parcours progressif au sein du GNS et les actions effectuées par notre robot, il faut bien se contenter du "reportage" qui suit. Attention, il n'est pas toujours simple de percevoir, avec les seules explications fournies, le déroulement de l'exécution et de débusquer les erreurs commises dans la conception de la marche à suivre.

Si la situation initiale est la suivante

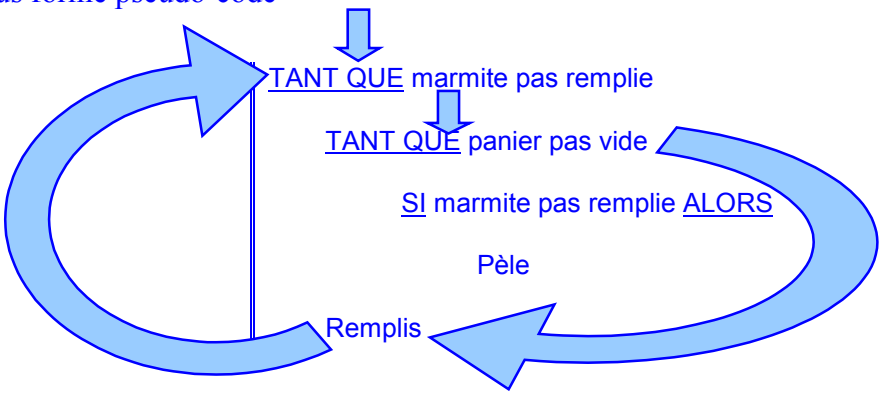


- On trouve un premier TANT QUE obligeant à travailler tant que la marmite n'est pas remplie.
- Passant au détail des actions qu'il faut alors répéter, on trouve un second TANT QUE, qui oblige à agir tant que le panier n'est pas vide. Il est vide, on passe directement outre de ce TANT QUE, pour trouver l'instruction "Remplis".

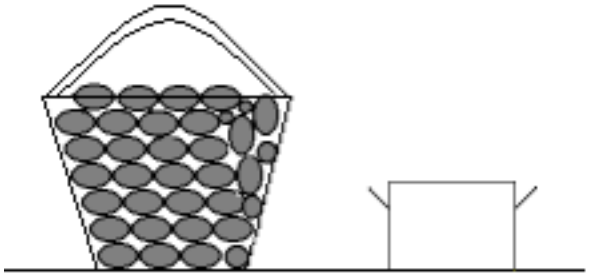
Le parcours, jusque là est donc



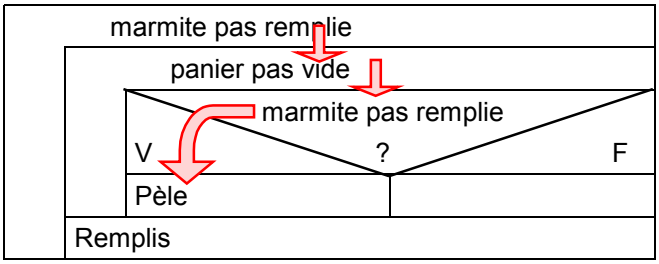

ou sous forme pseudo-code

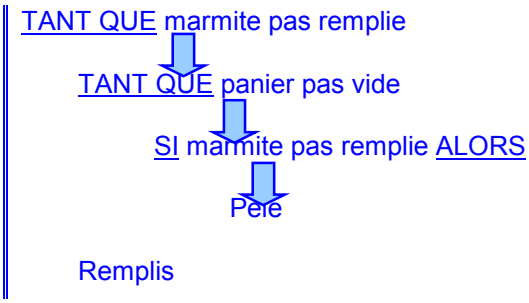



Cela fait, on revient au premier TANT QUE, la situation étant à présent :

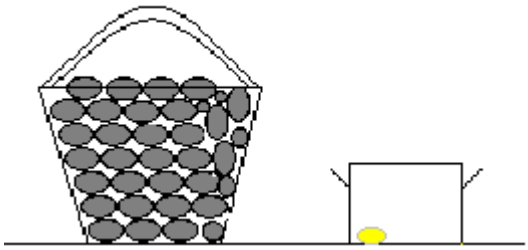


- La marmite n'est toujours pas remplie (forcément, il n'y a même pas une pomme de terre supplémentaire dedans !). On entame donc à nouveau les actions commandées par ce premier TANT QUE.
- On tombe sur le second TANT QUE qui oblige à travailler tant que le panier n'est pas vide.
- Ce travail commence par l'évaluation de la condition "la marmite n'est pas remplie". Cette condition est évidemment vraie et, dès lors, la suite du parcours est :

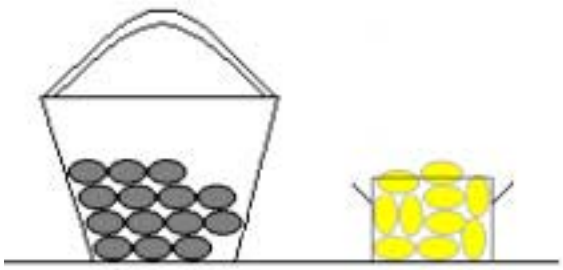



conduisant à la situation :



- On se trouve alors, à nouveau, à la condition "TANT QUE panier pas vide ...". Il n'est pas encore vide et on recommence donc.
- La marmite n'étant pas remplie, on pèle une seconde pomme de terre.
- Ce cycle se poursuit jusqu'à la situation :

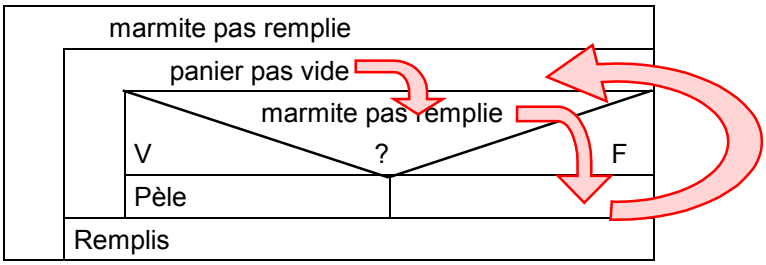


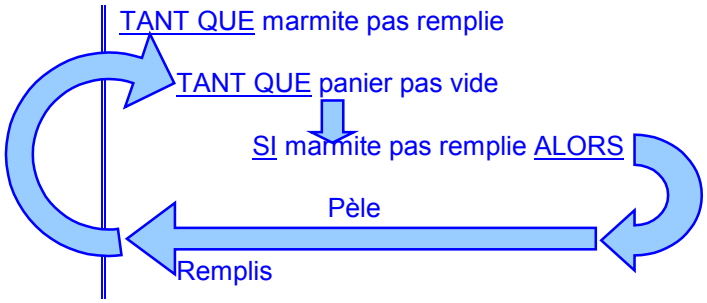
La marmite est alors remplie et le panier n'est pas encore vide ! Et l'on va alors rester prisonnier sans fin dans la boucle "TANT QUE le panier n'est pas vide ...".

En effet, les actions commandées dans cette boucle consistent d'abord à évaluer la condition "marmite pas remplie". Cette condition étant fausse, on ne pèle rien de plus et on revient à "TANT QUE le panier n'est pas vide" ..., ce qui reste bien sûr vrai, puisqu'on n'y a puisé aucune pomme de terre en plus.

Le flot des "activités" continue sans fin de la manière suivante :




ou en pseudo-code




Nous voici, dès lors, aux prises avec le cauchemar des programmeurs : conduit par la marche à suivre proposée, l'exécutant "travaille" sans jamais s'arrêter ! Cette situation, assez fréquente, est désignée par des expressions comme "cyclage" ou "bouclage" : à chaque fois, l'exécutant y est prisonnier d'une boucle de répétition qu'il reprend sans cesse, la condition qui permettrait de la quitter ne se réalisant jamais. C'est évidemment une erreur que nous rencontrerons aussi dans le cadre de la "vraie" programmation, lorsque notre interlocuteur sera l'exécutant-ordinateur.

Après ces tentatives infructueuses, le moment est venu de proposer une solution correcte au problème du peleur de pommes de terre.

## *6.3 Construction d'une marche à suivre*

Il n'existe malheureusement aucune règle qui conduise à coup sûr à l'écriture d'une marche à suivre satisfaisante; autrement dit, il n'existe aucun algorithme pour la conception des algorithmes. En ce sens, la programmation reste et restera un art qui ne peut s'apprendre que lentement : c'est en s'exerçant à la conception des marches à suivre que, peu à peu, on maîtrise les difficultés qui y sont attachées.

Je vais cependant indiquer, dans le cas du robot peleur de pommes de terre, une manière de procéder qui illustre ce que peut être la démarche descendante qui va s'incarner dans l'utilisation de l'appel de procédure et que nous retrouverons plus tard dans la "vraie" programmation.

L'objectif ici est donc de faire en sorte que le robot pèle la quantité de pommes de terre nécessaire pour remplir la marmite. On "sent" bien que ce processus est essentiellement répétitif. Dès lors, on serait tenté de proposer :

```
FAIS TOUT CE QU'IL FAUT POUR METTRE UNE POMME DE TERRE EN PLUS DANS LA MARMITE
JUSQU'A CE QUE la marmite soit remplie
```

```
REPETER
    FAIS TOUT CE QU'IL FAUT POUR METTRE UNE POMME DE TERRE EN PLUS DANS LA MARMITE
JUSQU'A CE QUE la marmite soit remplie
```

en sachant fort bien que l'action dont on commande la répétition ne figure pas dans le répertoire des actions élémentaires de notre exécutant. Ce n'est pas grave; nous pouvons laisser cette instruction trop complexe au sein de la marche à suivre, quitte à l'expliciter dans une annexe.

Ce qui est important, c'est de sentir que, par rapport à l'énoncé original qu'on pouvait synthétiser par

**"Remplis la marmite"**,

nous avons déjà fait un petit pas dans le décorticage rendu obligatoire par les caractéristiques de l'exécutant. Le problème n'est plus d'expliquer "comment faire pour remplir la marmite", mais seulement "comment faire pour ajouter une pomme de terre supplémentaire dans cette marmite".

Ainsi donc nous pouvons, à ce niveau de l'analyse, nous poser la question suivante : si l'exécutant était capable de comprendre :

```
FAIS TOUT CE QU'IL FAUT POUR METTRE UNE POMME DE TERRE EN PLUS DANS LA MARMITE
```

la marche à suivre proposée ci-dessus serait-elle correcte ?

Et là, on constate que tout ira bien, sauf dans le cas (limite !) où aucune pomme de terre ne doit être pelée, parce que la marmite fournie est déjà remplie avant même que le robot ne commence son travail. Je rappelle que j'avais signalé cette possibilité lors de l'énoncé du problème.

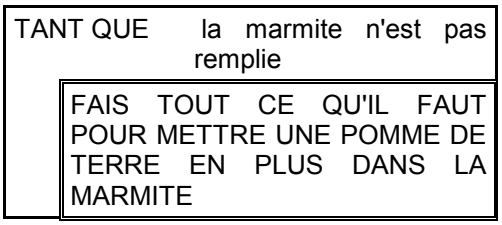
On va objecter que vraiment c'est "chercher la petite bête" que d'évoquer le cas extrême où la marmite est déjà remplie avant que le travail ne commence. Je répondrai seulement que le folklore informatique foisonne de ces exemples de cas "extrêmes" et non envisagés par les programmes de traitement.

> Combien de fois les consommateurs forcés d'informatique se voient-ils opposer, au cours de leurs démarches administratives, la petite phrase "l'ordinateur n'a pas prévu votre cas", comme si l'ordinateur pouvait "prévoir" quoi que ce soit ! La justification correcte est évidemment "l'analyste-programmeur a omis d'envisager les cas comme le vôtre et n'a donc pas indiqué à l'ordinateur d'y apporter un traitement convenable" ou même "je suis incompétent et incapable d'utiliser correctement l'ordinateur équipé du logiciel adéquat pour traiter votre problème".
>
> Nous touchons là deux traits fondamentaux de l'informatisation :
>
> - L'ordinateur est devenu un "parapluie" providentiel derrière lequel abriter tout un tas d'incompétences, soit de ses utilisateurs, soit, le plus souvent, des concepteurs des programmes qui le contrôlent et le font agir.
> - Le passage du "faire" au "faire faire" requis par l'informatisation gomme volontiers les cas limites ou exceptionnnels, soit parce que ceux-ci n'ont pas été débusqués lors de l'analyse, soit parce que les efforts de programmation nécessaires à leur mise en oeuvre sont trop importants.

La souplesse apportée par les réactions "sur le tas" d'un être humain face aux situations nouvelles et difficilement prévisibles est évidemment absente du comportement de l'ordinateur : la seule souplesse alors est celle que le programmeur a bien voulu inclure dans les traitements commandés par ses programmes. Et c'est vrai que, dès lors, l'informatique a une tendance "naturelle" à standardiser, supprimer les nuances, raboter les cas extrêmes ...

Dans la solution proposée, en effet, à cause du choix de la structure REPETER ..., la question de savoir si la marmite est remplie est posée trop tard : quand le robot aura déjà fait tout ce qu'il faut pour y ajouter une pomme de terre en plus (et en trop !). Il faudrait donc commencer, avant tout travail, par faire évaluer cette condition. C'est possible, par exemple, si j'exprime la répétition grâce à la structure TANT QUE ...




TANT QUE la marmite n'est pas remplie
    FAIS TOUT CE QU'IL FAUT POUR METTRE UNE POMME DE TERRE EN PLUS DANS LA MARMITE

Cette fois et, à condition que nous arrivions à expliquer au robot (dans une annexe) "COMMENT FAIRE TOUT CE QU'IL FAUT POUR METTRE UNE POMME DE TERRE EN PLUS DANS LA MARMITE", nous disposons d'une marche à suivre correcte.

Le problème global du remplissage de la marmite a maintenant disparu. Il a donné naissance à une question plus simple, puisqu'il reste seulement à expliquer comment peler une pomme de terre supplémentaire. On sent à nouveau que le seul paramètre gênant, à ce propos, c'est l'état du panier au moment où l'on va commander au robot de peler cette pomme de terre en plus. Il suffit donc de faire d'abord remplir le panier, si c'est nécessaire, et seulement après d'exiger de peler.

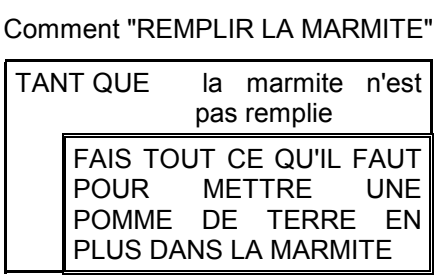
Cela conduit à :

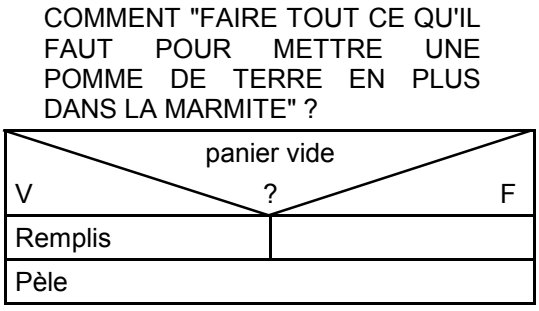

COMMENT "FAIRE TOUT CE QU'IL FAUT POUR METTRE UNE POMME DE TERRE EN PLUS DANS LA MARMITE" ?

SI panier vide ALORS
    Remplis
Pèle

Nous avons donc, avec la marche à suivre principale et l'annexe ci-dessus, une solution au problème posé !

Nous soumettrons donc au robot l'ensemble :




ou sous forme pseudo-code :

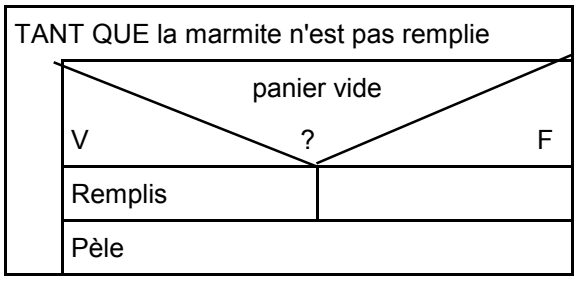
```
Comment "REMPLIR LA MARMITE"
TANT QUE la marmite n'est pas remplie
    FAIS TOUT CE QU'IL FAUT
    POUR METTRE UNE POMME
    DE TERRE EN PLUS DANS LA
    MARMITE
```

et

```
COMMENT "FAIRE TOUT CE QU'IL FAUT
POUR METTRE UNE POMME DE TERRE
EN PLUS DANS LA MARMITE" ?
SI panier vide ALORS
    Remplis
Pèle
```

A chaque fois qu'au cours d'exécution de la marche à suivre principale, l'exécutant trouvera l'instruction trop complexe "FAIS TOUT CE QU'IL FAUT POUR ...", son "regard" va se détourner vers l'annexe explicative : il exécutera ce qui y est commandé, puis reviendra à la marche à suivre principale pour en poursuivre l'exécution.

Bien évidemment, je peux en quelque sorte intégrer les explications annexes dans le corps de la marche à suivre principale en proposant au robot :




```
TANT QUE la marmite n'est pas remplie
    SI panier vide ALORS
        Remplis
    Pèle
```

Cette dernière proposition est peut-être plus "simple", mais elle a deux défauts importants :

- Elle cache la démarche que nous avons menée et qui consistait à réduire petit à petit le problème initial en passant à des problèmes plus simples. Il est sans doute préférable de choisir la présentation qui fait usage de l'appel de procédure. Ainsi, le lecteur; de la marche à suivre retrouvera facilement le cheminement de l'analyse menée et pourra la comprendre et la partager.
- Dans le cas où les tâches deviendront plus complexes, cette démarche d'**affinements successifs** et de simplifications par petits pas va s'avérer de plus en plus payante.

Elle s'incarnera, comme ci-dessus, dans la structure d'appel de procédure et nous essayerons de la rendre aussi visible que possible, dans un souci constant de faciliter la tâche de compréhension du lecteur.

## *6.4 Quelques remarques*

1. Je n'ai en aucune manière "démontré" que la marche à suivre proposée ci-dessus était correcte, c'est-à-dire que, dans tous les cas possibles, elle conduisait l'exécutant à mener à bien le travail proposé (quels que soient les tailles et les états initiaux du panier et de la marmite). "Essayer" cette marche à suivre avec tel ou tel panier, telle ou telle marmite pourrait seulement montrer que, dans chacun des cas testés, ça marche. On n'aurait pas pour autant prouvé que la solution proposée est satisfaisante dans tous les cas imaginables.

   Le problème de la preuve de la correction d'un algorithme est bien difficile. Elle s'apparente de très près aux mathématiques et dépasse, de loin, le cadre de mon propos.[5]

[5] Le lecteur intéressé peut, par exemple, consulter [1] ou [5].

Je ne **démontrerai** jamais que les marches à suivre développées dans cet ouvrage sont correctes. Simplement, j'essayerai de vous en convaincre et de vous faire partager mes certitudes.

2. On pourrait épiloguer longuement sur ce qu'on appelle une marche à suivre "correcte". Je dirai seulement que ceci n'a de sens que rapporté à ce qui était attendu et précisé dans le "Quoi faire ?". Le cahier des charges étant précisé, de deux choses l'une :
   - Le programme élaboré fait faire exactement ce qui est attendu, **dans tous les cas prévus**, et nous pouvons alors décréter qu'il est correct, ou, plus justement, conforme aux spécifications élaborées dans l'étape du "Quoi faire ?" (Cf. Chapitre 1).
   - Certaines situations, pourtant mises en évidence lors de la précision de ce qui est attendu, ne sont pas correctement traitées : l'exécutant, guidé par le programme, ne fait pas ce qu'il faudrait, parce que le programme comporte des lacunes ou des erreurs. Nous dirons bien entendu dans ce cas qu'il est incorrect.

   Cette vision reporte évidemment à l'étape du "Quoi faire ?" une bonne partie des problèmes. C'est alors en effet qu'il faudra avoir précisé parfaitement la tâche qu'il faut (faire) accomplir : dans quelles limites pourront être comprises les données à traiter, comment les résultats doivent-ils être présentés, ...

   En résumé, ou bien le programme fait faire ce qui était prévu et, dans ce cas, il est correct, ou bien il ne fournit pas, dans les situations envisagées, les résultats escomptés et il est alors erroné.

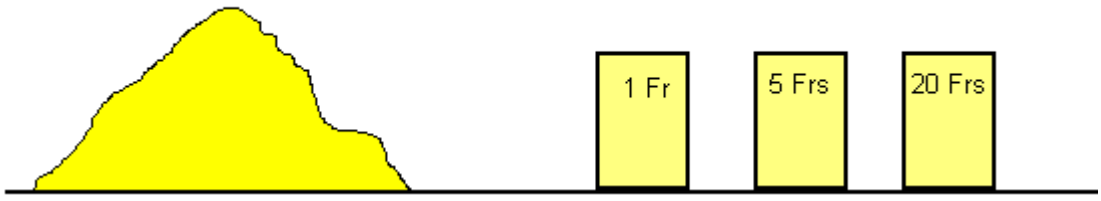
# 7. Exercices

## *7.1 Commandes de robots "virtuels"*

Dans chacune des situations suivantes, un "robot" est décrit : les instructions d'action élémentaire qu'il "comprend" et les conditions qu'il est capable de tester sont précisées. Il s'agit à chaque fois de proposer, sous forme GNS ou sous forme pseudo-code, la marche à suivre qui le conduise à effectuer la tâche envisagée.

Je rappelle que les conditions énoncées peuvent être niées et que plusieurs de ces conditions peuvent être liées par les mots "ET" et "OU" pour composer des conditions plus complexes.

### 7.1.1 Le robot "trieur de pièces de monnaie"




Tas de pièces de 1, 5 et 20 Frs

Ce robot va devoir répartir un tas constitué de pièces de 1, 5 et 20 Frs dans trois récipients destinés à recevoir les pièces des diverses valeurs.

**Instructions d'action élémentaire**

| | | |
|---|---|---|
| Prends | : | le robot saisit une pièce du tas. |
| Dépose 1 | : | le robot dépose la pièce qu'il tient dans le récipient destiné à recevoir les pièces de 1 Fr. |
| Dépose 5 | : | le robot dépose la pièce dans le récipient destiné à recevoir les pièces de 5 Frs. |
| Dépose 20 | : | le robot dépose la pièce qu'il tient dans le récipient destiné à recevoir les pièces de 20 Frs. |

**Conditions**

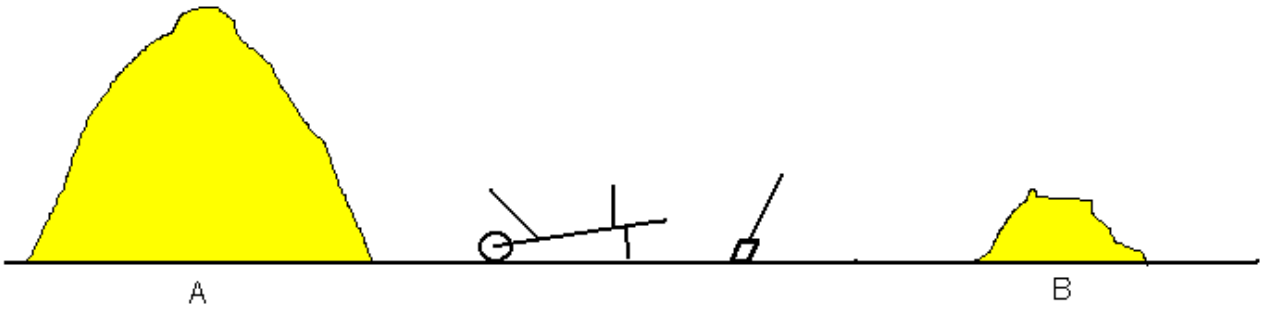
- tas vide ? : le tas de pièces est vide;
- pièce de 5 Frs : la pièce tenue en main est une pièce de 5 francs;
- pièce de 20 Frs : la pièce tenue en main est une pièce de 20 francs;

et les contraires de ces conditions, qu'on peut aussi lier par ET ou OU.

**Remarques**

- les récipients où ranger les pièces sont de tailles suffisantes pour recevoir n'importe quel nombre de pièces;
- il peut, certains jours, n'y avoir aucune pièce à trier;
- le robot ne peut tenir plusieurs pièces à la fois.

### 7.1.2 Le robot "transporteur de sable"




Ce robot va devoir transporter un tas de sable situé en A pour le déposer en B. Il dispose pour cela d'une brouette qu'il peut charger à l'aide d'une pelle.

**Instructions d'action élémentaire**

| | | |
|---|---|---|
| Mets une pelletée dans la brouette | : | le robot prend une pelletée de sable dans le tas A et la dépose dans la brouette; |
| transporte | : | le robot transporte la brouette de A en B; |
| vide | : | le robot déverse la brouette où elle se trouve; |
| ramène | : | le robot ramène la brouette de B en A. |

**Conditions**

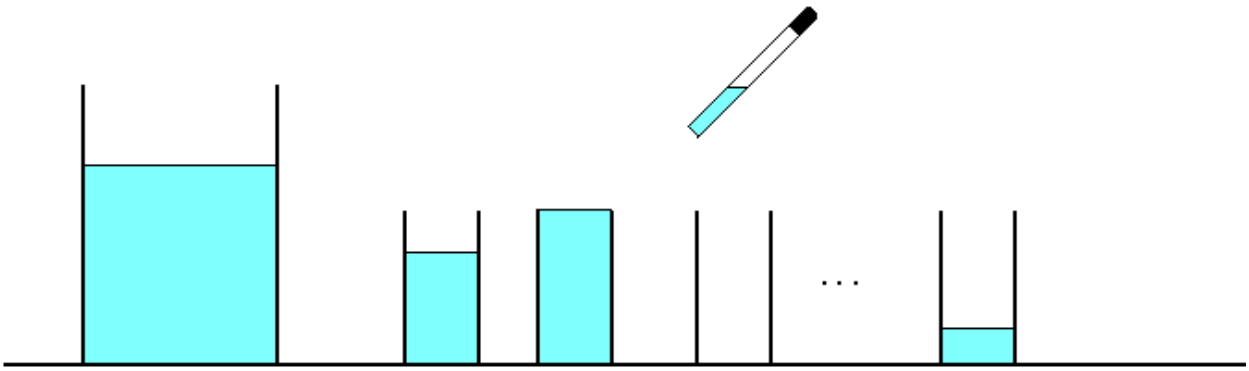
- il y a encore du sable à transporter en A;
- la brouette est pleine;

et leurs contraires.

**Remarques**

- il y a toujours du sable à transporter quand le robot commence;
- la brouette est toujours vide à côté du tas en A quand le robot commence;
- le contenu de la brouette correspond à plusieurs pelletées de sable;
- le travail terminé, la brouette peut rester n'importe où.

### 7.1.3 Le robot "compteur de gouttes"



Il dispose d'un récipient contenant une quantité inépuisable de liquide et d'un compte-gouttes. L'objectif est de lui faire remplir une série de verres. Comme d'habitude, il "comprend" certaines

**Instructions d'action élémentaire**

| | | |
|---|---|---|
| Choisis le premier verre | : | il vient se placer devant le premier verre de la série et est prêt à s'en occuper; |
| Presse une goutte dans le verre choisi | : | il presse une goutte dans le verre dont il est en train de s'occuper; |
| Passe au verre suivant | : | il passe au verre suivant; |
| Remplis le compte-gouttes | : | il remplit le compte-gouttes et revient s'occuper du verre où il travaillait au moment où on lui a commandé ce remplissage. |

**Conditions**

Il est également capable de tester quelques conditions :

- le verre choisi est plein;
- le compte-gouttes est vide;
- tous les verres sont remplis.

Comme d'habitude, on peut faire figurer dans la marche à suivre :

- ces conditions;
- leurs contraires (le verre choisi N'est PAS plein, ...);
- des conditions plus complexes utilisant les mots ET ou OU
  (le verre choisi n'est pas plein ET le compte-gouttes est vide, ...).

**Remarques**

- les verres sont déjà plus ou moins remplis quand l'exécutant commence;
- l'état initial du compte-gouttes n'est pas précisé : il peut être vide, rempli, à moitié rempli, etc...;
- le nombre de verres à remplir est fini.

## *7.2 Vérification de marches à suivre*

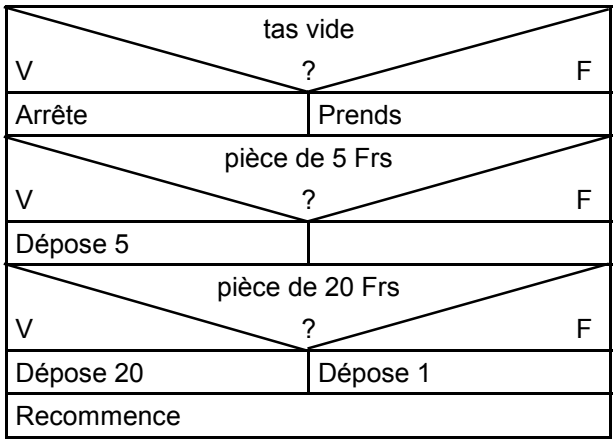
Pour chacun des robots décrits à l'exercice précédent, plusieurs marches à suivre sont proposées ci-dessous.[6]

Pouvez-vous apprécier chacune d'elles au point de vue de :

- sa correction syntaxique (= les GNS sont correctement tracés, les pseudo-codes, correctement écrits);
- sa correction "logique" (= la marche à suivre conduit bien le robot à faire ce qui était attendu);
- son efficacité.

### 7.2.1 Le robot "trieur de pièces de monnaie"

a)

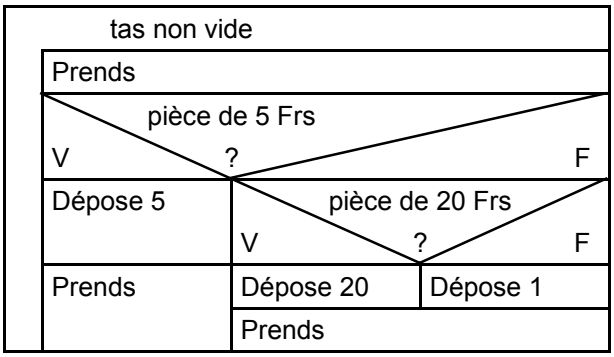



```
SI tas vide ALORS
        Arrête
SINON
        Prends
SI pièce de 5 Frs ALORS
        Dépose 5
SI pièce de 20 Frs ALORS
        Dépose 20
SINON
        Dépose 1
Recommence
```

6 Je me permettrai, sans changer le sens, d'écrire sous une forme plus concise les conditions permises.

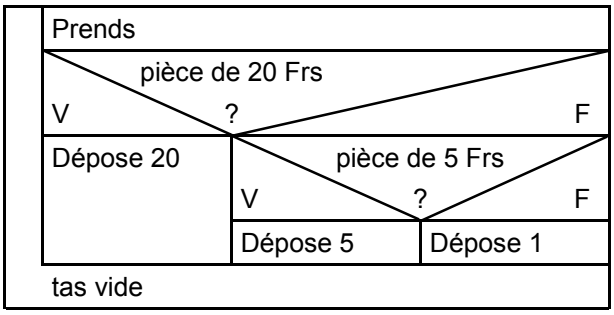
 

b)




```
TANT QUE tas non vide
    Prends
    SI pièce de 5 Frs ALORS
        Dépose 5
        Prends
    SINON
        SI pièce de 20 Frs ALORS
            Dépose 20
        SINON
            Dépose 1
        Prends
```

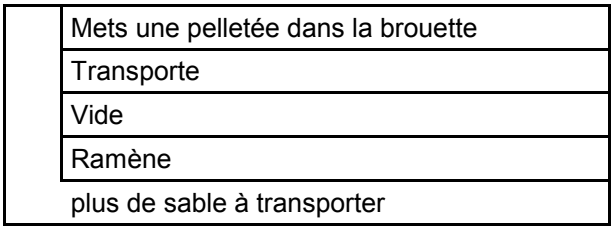
c)




```
REPETER
    Prends
    SI pièce de 20 Frs ALORS
        Dépose 20
    SINON
        SI pièce de 5 Frs ALORS
            Dépose 5
        SINON
            Dépose 1
JUSQU'A CE QUE tas vide
```

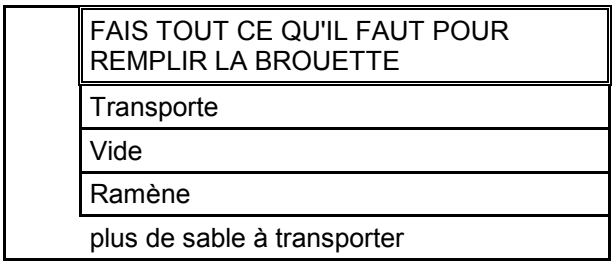
### 7.2.2 Le robot "transporteur de sable"

a)



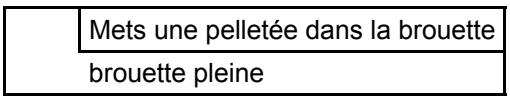

```
REPETER
    Mets une pelletée dans la brouette
    Transporte
    Vide
    Ramène
JUSQU'A CE QUE plus de sable à
    transporter
```

b)




et

Comment "FAIRE TOUT CE QU'IL FAUT POUR REMPLIR LA BROUETTE"




```
REPETER
    FAIS TOUT CE QU'IL FAUT POUR
    REMPLIR LA BROUETTE
    Transporte
    Vide
    Ramène
JUSQU'A CE QUE plus de sable à transporter
```

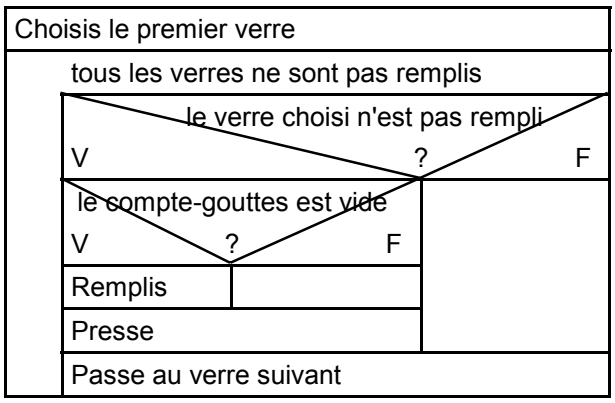
et

```
Comment "FAIRE TOUT CE QU'IL
FAUT POUR REMPLIR LA BROUETTE"
```

```
REPETER
    Mets une pelletée dans la brouette
JUSQU'A CE QUE  brouette pleine
```

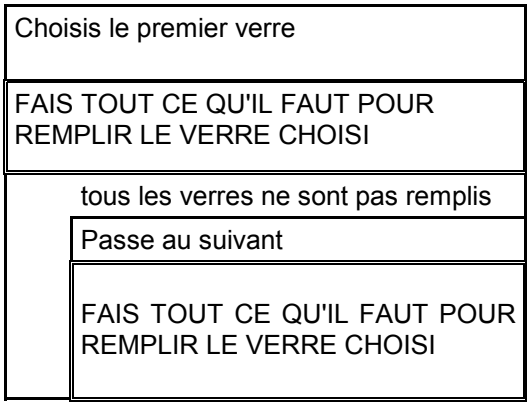
### 7.2.3 Le robot compteur de gouttes

a)

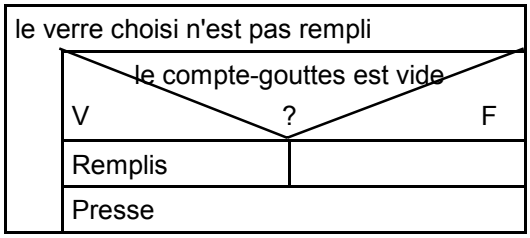



```
Choisis le premier verre
TANT QUE tous les verres ne sont pas remplis
    SI le verre choisi n'est pas rempli ALORS
        SI le compte-gouttes est vide ALORS
            Remplis
        Presse
    Passe au verre suivant
```

b)




Comment "FAIRE TOUT CE QU'IL FAUT POUR REMPLIR LE VERRE CHOISI"



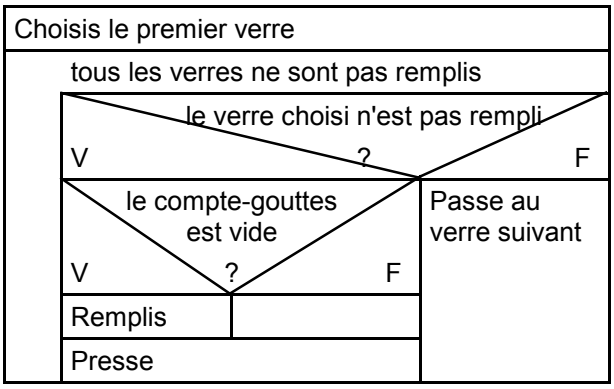

```
Choisis le premier verre
FAIS TOUT CE QU'IL FAUT POUR REMPLIR LE VERRE CHOISI
TANT QUE tous les verres ne sont pas remplis
    Passe au suivant
    FAIS TOUT CE QU'IL FAUT POUR REMPLIR LE VERRE CHOISI
```

```
Comment "FAIRE TOUT CE QU'IL FAUT POUR REMPLIR LE VERRE CHOISI"

TANT QUE le verre choisi n'est pas rempli
    SI le compte-gouttes est vide ALORS
        Remplis
    Presse
```

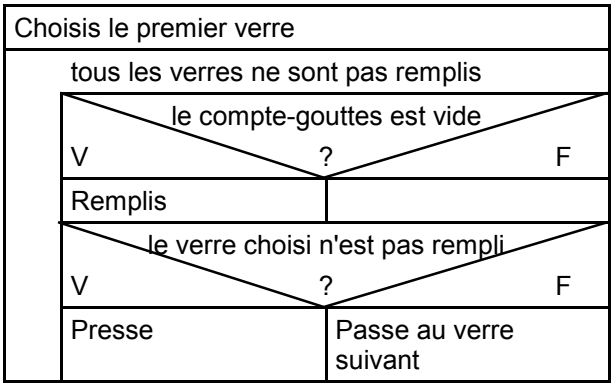
 

c)




```
Choisis le premier verre
TANT QUE tous les verres ne sont pas remplis
    SI le verre choisi n'est pas rempli ALORS
        SI compte-gouttes est vide ALORS
            Remplis
        Presse
    SINON
        Passe au verre suivant
```

d)

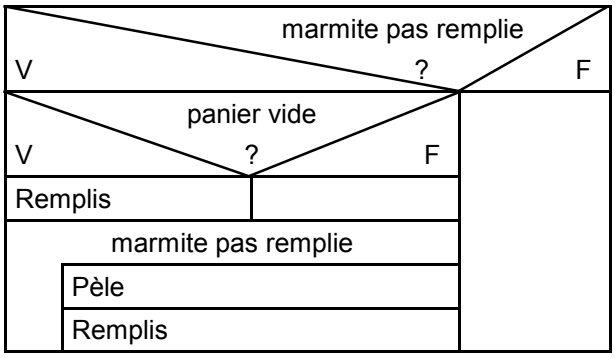



```
Choisis le premier verre
TANT QUE tous les verres ne sont pas remplis
    SI compte-gouttes est vide ALORS
        Remplis
    SI le verre choisi n'est pas rempli ALORS
        Presse
    SINON
        Passe au verre suivant
```

### 7.2.4 Le robot peleur de pommes de terre

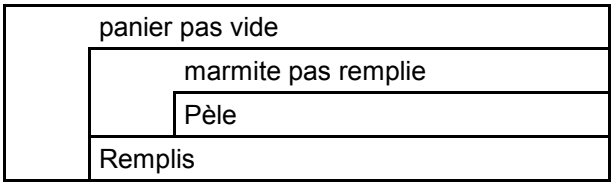
Que pensez-vous des GNS ou pseudo-codes suivants destinés au robot "peleur de pommes de terre" ?

a)



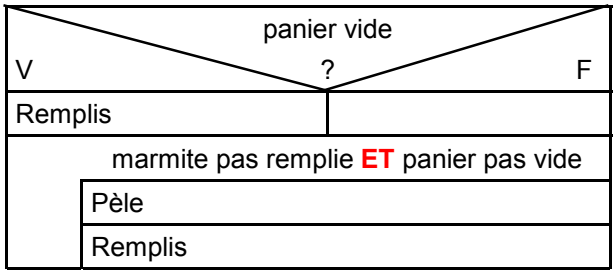

```
SI marmite pas remplie ALORS
    Si panier vide ALORS
        Remplis
    TANT QUE marmite pas remplie
        Pèle
        Remplis
```

b)




```
TANT QUE panier pas vide
    TANT QUE marmite pas remplie
        Pèle
    Remplis
```

c)




```
SI panier vide ALORS
    | Remplis
TANT QUE marmite pas remplis ET
         panier pas vide
    | Pèle
    | Remplis
```

# 3

# MAIS QUI EST L'EXECUTANT-ORDINATEUR ?

Nous savons (Chapitre 1) que le cœur de l'activité de programmation est la rédaction de "marches à suivre"; nous avons vu de quoi elles étaient constituées et avons joué à en rédiger pour des exécutants "bidons" (Chapitre 2). Il reste donc seulement à décrire les caractéristiques du véritable exécutant, pour lequel nous allons dorénavant concevoir ces marches à suivre, les vraies, celles de la programmation.

1. Rappelez-vous que cet exécutant, celui à qui nous nous adresserons, n'est pas seulement l'ordinateur-machine avec ses mémoires, son processeur. Ce n'est pas toute la "quincaillerie" électronique qui nous intéresse ici. C'est plutôt l'ordinateur transfiguré par la possibilité qu'il semble montrer de "comprendre" ce que je lui dis en Basic ou en Pascal (ou en tout autre langage de programmation). Quelles sont alors les actions élémentaires dont il semble capable ? Quelles conditions peut-il évaluer ? A quelles structures de contrôle est-il sensible ?

2. Si notre exécutant, c'est donc l'ordinateur vu à travers le langage de programmation qui l'équipera, il est évident que ses caractéristiques dépendent, en partie du moins, de ce langage. Le portrait que je pourrai en tracer variera (un peu) selon que je m'adresserai à un exécutant "comprenant" Basic, LSE ou Pascal. Mais, on va le voir, c'est sur des points de détail que se marqueront les différences : les grands traits seront communs quel que soit le langage de programmation employé. C'est ce qui me permet de brosser un "portrait-robot" de cet exécutant-ordinateur, avant même d'indiquer dans le détail les caractéristiques du langage de programmation à travers lequel je finirai par devoir m'adresser à lui.

   Notons donc que

   **L'exécutant-ordinateur = l'ordinateur-machine + le langage de programmation**

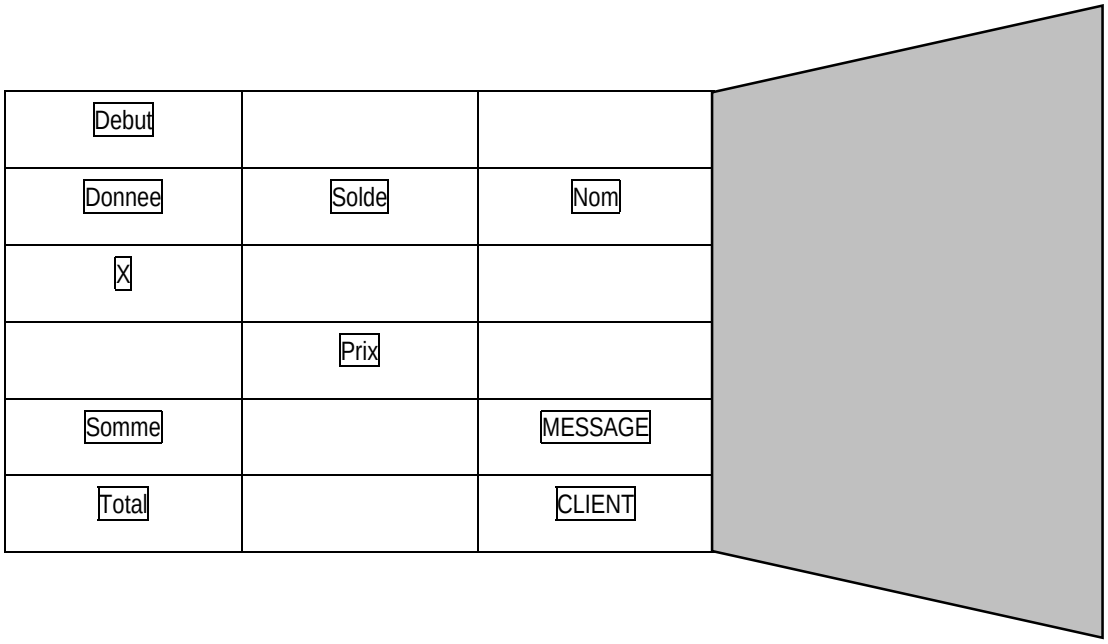
J'ai dit que, quel que soit le point de vue envisagé, l'ordinateur est une machine à traiter l'information. Si notre regard est celui du "programmeur", nous allons retrouver, évidemment, cette caractéristique comme trait essentiel de l'exécutant-ordinateur : c'est un manipulateur, un traiteur d'informations.

**L'exécutant-ordinateur est un manipulateur d'informations.**

Et d'abord,

## 1. C'est un gestionnaire de casiers

Nous pouvons l'imaginer comme enfermé dans une pièce tapissée de rayonnages, de casiers.




Inutile de préciser qu'il s'agit évidemment d'une métaphore ! Elle est pertinente si notre objectif est de découvrir les premières caractéristiques de notre futur exécutant. Tout se passe **comme si** c'était un gestionnaire de casiers. Mais comme toute image, toute comparaison, cette métaphore "ment" ou travestit en partie la "réalité" (par simplification abusive ou omission). Elle véhicule cependant un fond de vérité : ainsi les "casiers" sont évidemment les cellules (ou groupes de cellules) de la mémoire centrale.

Il va passer le plus clair de son temps à déposer "des choses" dans ces casiers et à prendre copie de ce qui s'y trouve.

## *1.1 Les caractéristiques d'un casier*

Si l'on y regarde de plus près, on constate qu'un tel casier est caractérisé par trois choses :

- le **nom** qu'il porte, autrement dit, l'étiquette du casier;
- son **type**, déterminant ce qu'il pourra accueillir;
- ce qu'il y a réellement dedans, son **contenu**.

| Donnee | Solde | Nom |
|---|---|---|
| 14 | -112.5 | DUPONT |
| X | | |
| 27 | 47.66 | Rsl!tZ |
| | Prix | |
| -212 | 13.75 | 22A1 |
| Somme | | MESSAGE |
| 69 | 14.0 | Bonjour! |

1. En présentant des casiers de "tailles" différentes, le schéma illustre à sa manière que ceux-ci ne sont pas tous du même type. Un casier ne pourra pas servir au stockage de n'importe quoi : c'est son type qui déterminera justement le genre de "choses" qu'il pourra contenir.

2. On le voit aussi, certains casiers portent une "étiquette;" qui va permettre de les identifier. Ces étiquettes, c'est nous (programmeurs) qui choisiront ce qu'elles sont (avec certaines contraintes de forme imposées par le langage compromis). Nous nous contenterons d'exiger tel et tel casier en en précisant le type et l'étiquette : à l'exécutant de se débrouiller pour savoir où trouver les emplacements nécessaires dans les rayonnages disponibles. Mais toute marche à suivre commencera toujours par l'indication des casiers indispensables.

   Lors de la phase d'analyse et de rédaction de la marche à suivre, je ne serai pas encore contraint par les règles syntaxiques du langage de programmation comme je le serai ensuite dans la phase de codage ("Comment dire ?").

   En ce qui concerne les noms des casiers, je vais cependant "préparer le terrain", en respectant dès à présent quelques conventions :

   - les étiquettes ne contiendront que :
     - des lettres majuscules ou minuscules (non accentuées),
     - des chiffres,
     - le symbole de soulignement "_ ";
   - elles commenceront toujours par une lettre.

   Ainsi

   *Somme, X, AGE_DU_CAPITAINE, A12, NomDuClient*

   seront des étiquettes permises, mais pas,

   *1A, Total Des Cotes, NOM-DU-CLIENT.*

   Une fois une étiquette posée sur un casier, elle y restera collée tout le temps du déroulement de l'exécution de la marche à suivre, sans jamais pouvoir être manipulée par l'exécutant. C'est grâce à ces étiquettes que je pourrai lui désigner l'un ou l'autre casier.

3. Reste à dire, à présent, quelles sont ces "choses" qui transiteront à l'intérieur des casiers, ce que l'exécutant va y déposer, en un mot ce qu'il va manipuler. Mais cela, nous le savons déjà, puisque nous avons annoncé que nous aurions à commander un "manipulateur d'informations" : ce qui sera contenu dans ces casiers, c'est donc de l'information.

## *1.2 Les informations manipulées*

D'abord, il faut bien s'entendre sur le mot "information". Dans son acception habituelle, ce terme comporte généralement deux facettes :

- la forme sous laquelle se présente l'information : le ou les signes, caractères, dessins qui codent l'information et en constituent l'aspect "formel" (Cf. Chapitre1);
- le sens dont est porteur, pour l'être humain, cet assemblage de signes.

Lorsque je parlerai ici d'information, il s'agira seulement des aspects formels, sans référence à la signification qui y est attachée. On appelle parfois **donnée** ce support formel de l'information. Ainsi, 112 est une donnée (un nombre entier) qui peut subir des manipulations formelles : on peut doubler ce nombre, le diviser par deux, lui en ajouter un second ... Ce nombre ne devient une information que si l'être humain qui en prend connaissance sait que c'est un solde débiteur, le poids d'un patient ou le nombre de pages d'un syllabus.

L'ordinateur manipule des données, c'est l'homme qui en fait des informations.

Plutôt que d'essayer d'apporter une définition supplémentaire du concept "information"[1], je préfère indiquer quelles seront celles manipulées par l'exécutant-ordinateur.

### 1.2.1 Les nombres

Il traitera évidemment des nombres. C'est bien le moins pour une machine qu'on appelait, il y a quelques années encore, un "calculateur électronique".

Comme par exemple :

```
15      139     - 14                    (nombres entiers)
12.5    - 6.9   132.46  13.0            (nombres réels)
```

On aura compris que les nombres entiers sont ceux (positifs ou négatifs) sans partie décimale tandis que les nombres réels comportent (en général) une partie décimale : il y a des chiffres "après la virgule", qui devient d'ailleurs, pour sacrifier à la notation anglo-saxone, un point. Certains casiers pourront donc être prévus pour accueillir ces informations numériques entières ou réelles. Nous dirons qu'ils sont de type entier ou de type réel.

Remarquons que si les nombres négatifs sont (évidemment) précédés du signe -, on ne prendra pas la peine d'accoler le signe + aux nombres positifs.

### 1.2.2 Les chaînes de caractères

L'exécutant-ordinateur manipulera aussi des informations que nous appellerions volontiers des "mots" ou du "texte". En réalité, nous parlerons plutôt de chaînes ou de successions de caractères, car si nous sommes prêts à nommer "mot" des informations comme :

```
            'BONJOUR'
ou          'Au revoir !'
par contre, 'ABCD'
            '123'
            '?!ER,.'
```

nous apparaissent bien comme de simples successions de caractères et ne méritent certes pas le statut de "mots".

Ceci n'est pas étonnant, puisque, comme je l'ai dit plus haut, l'ordinateur se contente de manipuler formellement des informations, indépendamment du sens, de la connaissance qu'elles portent. C'est l'homme qui fait d'une chaîne de caractères un mot, qui déchiffre l'information sous la donnée, le sens sous la forme.

## *1.3 Remarques*

1. Si, dès à présent, on devine qu'en ce qui concerne les nombres, l'exécutant-ordinateur pourra, outre les placer dans l'un ou l'autre casier, les additionner, les soustraire, les multiplier, ... (ce que nous faisons aussi avec des nombres), il est plus difficile de percevoir les manipulations qui s'appliqueront aux informations "textuelles" : nous traitons rarement des chaînes de caractères sans nous référer au sens dont les mots qu'elles représentent sont porteurs.

   Disons sans entrer maintenant dans les détails que l'exécutant-ordinateur pourra

   - les "recoller" pour faire de

[1] Il existe pour cela d'excellents dictionnaires.

'TURLU' et 'TUTU'

la chaîne

'TURLUTUTU';

- ôter des morceaux d'une chaîne
- comparer deux chaînes, etc.

2. On remarquera les symboles apostrophes employés lorsque nous souhaitons parler d'une chaîne de caractères : la succession des caractères qui la composent est enclose à l'intérieur de ' '. On comprendra dans la suite la nécessité de cette contrainte.

3. Je ne m'intéresse pas du tout ici au fait de savoir comment l'ordinateur-machine "écrira" ou "codera" ces informations (qu'elles soient numériques ou textuelles). Ce qui est important c'est que je puisse, dans le texte de mes marches à suivre, écrire 15 lorsque je souhaite faire manipuler l'information que je note habituellement 15. Qu'il "l'écrive" XV, 1111, ou IIIIIIIIIIIIIII, peu importe. Ce qui est essentiel, c'est que je lui dise 15 (= à un moment ou un autre, je frapperai au clavier les touches 1 et 5) et que lorsqu'il voudra me parler de 15, il m'écrive 15 (= il affichera à l'écran les symboles 1 et 5).

   L'ordinateur ne "voudra" jamais me parler, pas plus que ma lessiveuse ne "veut" me faire plaisir en lavant mon ligne, pas plus que mon automobile ne "souhaite" faciliter mes déplacements. Rappelez-vous : je ne dis généralement pas que ma bêche est courageuse ...

4. J'ai parlé de chaînes de caractères, sans même dresser la liste de ce que sont les divers caractères qui pourront y figurer. Ce n'est pas trop gênant pour le moment puisque je ne veux pas être exhaustif ou complet.

   Sachons simplement que parmi ces caractères, il y aura :

   - les lettres (majuscules, minuscules et accentuées),
   - les chiffres,
   - les symboles de ponctuation : . , ; ? ! ... en y incluant l'espace (= le blanc),
   - etc..

5. On le sait déjà, notre exécutant-ordinateur, lorsqu'il manipulera des chaînes de caractères, le fera sans **aucune référence au sens** que nous donnons ("c'est un mot") ou que nous refusons ("ce n'est pas un mot") à ces chaînes.

   Le traitement sera purement formel (= ne s'attachant qu'à la forme de ces chaînes).

   Ainsi, il pourra répondre que 'BONJOUR' et 'BON JOUR' ne sont pas deux chaînes identiques, mais il est incapable de décider "de lui-même" que

   'BONJOUR' est un "mot",

   'RUOJNOB' n'en est pas un,

   ou que 'PETIT' est un adjectif qualificatif épithète ou attribut.

   Rappelons bien que j'en suis à décrire les capacités de base de l'exécutant-ordinateur; je serais tenté d'écrire ses "aptitudes innées". Mais, chaque fois que je lui aurai expliqué (sous la forme d'une marche à suivre) comment mener à bien une tâche, j'aurai en quelque sorte enrichi son stock de possibilités, je lui aurai appris (si l'on peut dire) comment mener à bien un travail dont il était incapable. Mais, toute la "matière grise", toute "l'intelligence", c'est évidemment à l'être humain qui a décortiqué la tâche qu'on les doit.

6. Il est important de ne pas confondre nombre et chaînes de caractères : 126 (sans apostrophe) est une information de type nombre entier et il acceptera volontiers d'y additionner 3 (en "trouvant" d'ailleurs - heureusement - 129).

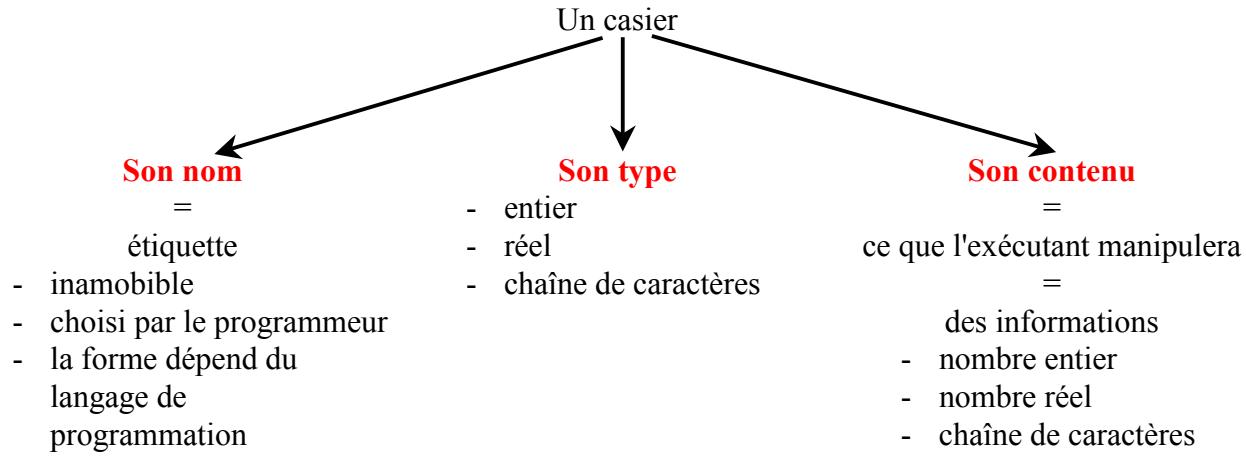
   Mais '126' est la succession des trois caractères '1' '2' et '6' et il "ne comprendra rien" si nous lui demandons d'y additionner le caractère '3' (et encore moins le nombre 3). Ou bien, comme certains enfants, il répondra que '126' + '3', c'est '1263'.

7. Les étiquettes des casiers ne sont pas des informations manipulées par l'exécutant. Au moment où nous commencerons à le faire agir (par le biais du texte de la marche à suivre proprement dit), ces étiquettes (qui ne sont pas des chaînes de caractères) seront déjà installées. Jamais il n'aura à les manipuler.

   La métaphore employée induit à penser que seules les informations manipulées résideront dans les casiers de la mémoire. Apparemment quand l'exécutant commence à agir, sous l'emprise de la marche à suivre, les étiquettes nécessaires sont déjà collées sur les casiers qui seront utilisés et il tient "en main" cette marche à suivre. Si cette image "convient" quand il s'agit de dépeindre les caractéristiques de l'exécutant-ordinateur, l'objectif étant d'apprendre à programmer, elle est évidemment complètement fausse si l'on s'intéresse à ce qui se passe "vraiment" au niveau de l'ordinateur- machine. On a vu que, non seulement les informations manipulées sont stockées en mémoire centrale, mais encore la marche à suivre elle-même, et aussi les étiquettes de ce que nous appelons les casiers. En réalité, tout ce qui est nécessaire à l'ordinateur pour mener à bien la tâche souhaitée réside dans les cellules de la mémoire centrale.

   Rappelons aussi que la phase pendant laquelle l'ordinateur exécute réellement le programme que nous avons conçu a été préparée par l'étape d'édition où l'ordinateur, gouverné par le programme éditeur, se transforme en "super-machine à écrire". Puis, est venue l'étape de traduction : l'ordinateur, sous l'emprise du programme compilateur, traduit le texte (exprimé dans le langage compromis) dans le langage de la machine (le seul que comprenne le processeur). A l'issue de chacune de ces étapes, le résultat du travail (texte du programme pour l'étape d'édition, version traduite après la phase de compilation) est généralement "sauvé" sur un support mémoire externe (disque ou disquette, le plus souvent). Tout cela, la métaphore du gestionnaire de casier le cache ou le travestit. ça n'a guère d'importance : l'objectif ici c'est de découvrir ce qu'est programmer et en particulier, puisque programmer c'est faire faire des choses par un exécutant, d'apprendre quels en sont les grands traits.

Si je fais le point sur ce que nous venons de découvrir à propos des casiers que l'exécutant-ordinateur devra gérer, je peux proposer le schéma suivant :




Il me reste à reparler des étiquettes des casiers à présent que nous savons que ce qui y sera déposé, c'est de l'information. J'ai dit que nous (programmeurs) avions le libre choix des noms que nous ferons porter aux casiers utilisés (sauf quelques contraintes syntaxiques (orthographiques)

imposées par le langage compromis). Ce choix, il faudra le faire à bon escient, en restant attentif à faciliter la lecture (ou la relecture) de la marche à suivre par les êtres humains qui voudraient en prendre connaissance.

Dès lors, si je pense faire déposer dans un casier le nom d'un client, je choisirai comme étiquette pour ce casier *NOM_DU_CLIENT* et pas *X*. Si un casier est destiné à contenir le solde du compte d'un client, je l'appellerai *SOLDE* ou *SoldeDuCompte* et pas *AGE* ou *Nombre* ou *Y*. En un mot, j'essayerai de choisir une étiquette évocatrice des informations qui transiteront dans le casier.

J'ai déjà insisté sur la nécessité de rédiger des marches à suivre qui soient non seulement correctes, mais encore facilement lues et comprises par un être humain. Nous savons à présent que l'essentiel de ces marches à suivre consistera à faire gérer des casiers. Il est donc impératif que dans, le choix des étiquettes des casiers utilisés, je reste fidèle à cet objectif de lisibilité, de compréhension aisée, ...

## *1.4 L'instruction d'affectation*

Nous savons maintenant que l'exécutant-ordinateur, à qui nous destinons nos marches à suivre, sera constamment en train de manipuler des informations et même qu'il passera la plus grande partie de son temps à déposer (à notre demande) ces informations dans l'un ou l'autre casier.

J'en arrive, dès lors, tout naturellement à la **première instruction d'action élémentaire** destinée à cet exécutant. Peu importe pour l'instant la manière dont cette instruction se traduira dans le langage compromis. Je la laisserai, dans le texte de la marche à suivre, sous la forme

**Place *telle information* dans *tel casier***

C'est en quelque sorte l'instruction de "remplissage" d'un casier; on l'appelle, en programmation, l'instruction **d'affectation**. On dira donc qu'on affecte telle information à tel casier.

Nous l'abrégerons parfois sous la forme :

***Nom du casier* ← information à y placer**

Il serait, sans doute, plus naturel d'écrire :

Information → *Nom du casier*

Il faut savoir que dans la plupart des langages de programmation l'instruction d'affectation prend des formes proches de :

*Nom du casier* ← information

Ainsi, en Pascal, on écrira :

```
Nom du casier := information
```

et en Basic :

```
Nom du casier = information
```

C'est pour préparer le respect de cette contrainte que le nom du casier sera placé à gauche du symbole d'affectation ← et l'information qui y prendra place à droite.

Pour préciser le casier à remplir, il suffira de citer son nom, son étiquette. Par contre, la précision de l'information à y déposer prendra des formes diverses. Nous pourrons la décrire de trois manières, et d'abord comme :

1. **Une constante**

   Par exemple, nous écrirons :

   Place 15 dans le casier *Nombre*

abrégé en

*Nombre* ← 15

ou

Place 'DUPONT' dans le casier *CLIENT*

abrégé en

*CLIENT* ← 'DUPONT'

Nous pouvons imaginer, dès lors, que l'exécutant-ordinateur dispose d'une énorme "malle à informations" d'où, à notre demande, il sort l'information souhaitée pour la déposer dans le casier indiqué. Dans le premier cas, il en sort le nombre entier 15 et le place dans le casier *Nombre*; dans le second, c'est la chaîne de caractères 'DUPONT' qui en est sortie pour être placée dans le casier *CLIENT*.

En tout cas, à chaque fois, **l'information qui résidait jusque là dans le casier qu'il remplit est perdue**. La nouvelle information que nous demandons d'y placer remplace la précédente, un casier ne contenant jamais qu'une information à la fois.

> Je suppose que personne ne croit réellement que "quelque part" sont stockées toutes les constantes possibles. Ça ferait un fameux tas ! En réalité, les constantes citées dans la marche à suivre (et celles-là seulement) sont aussi (comme tout le reste) installées dans les cellules de la mémoire-centrale.
>
> Mais, dans la métaphore mise en place, tout se passe comme si les constantes étaient tirées d'une grande "malle à informations".

Enfin, il faut évidemment que le type du casier que nous voulons faire remplir soit compatible avec l'information que nous demandons d'y placer. Les exemples précédents postulent donc que Nombre soit un casier prévu pour recevoir des nombres entiers (comme 15) et que CLIENT puisse recueillir des chaînes de caractères.

Si nous avions écrit :

Place 15 dans *CLIENT*

ou

Place 'DUPONT' dans *Nombre*

cela n'aurait évidemment eu aucun sens (et l'exécutant se serait chargé de nous le faire savoir !!!).

2. **Un nom de casier**

   Nous écrirons, par exemple :

   Place *SOMME* dans *TOTAL*

ou

Place *NOM* dans *FOURNISSEUR*,

*SOMME* et *TOTAL* étant deux casiers de même type, *NOM* et *FOURNISSEUR* également. Ce que nous voulons dire ici, c'est que l'information à placer dans le casier *TOTAL* est à chercher dans le casier *SOMME*.

Une information est en réalité toujours une constante. Si l'exécutant "tient en main", à un moment donné, l'information 15, cette information restera toujours 15. Elle ne va pas se transformer en 16 ou en 14. Dès lors, l'exécutant ne manipule que des constantes. C'est la manière dont je désigne dans le texte de la marche à suivre ces constantes manipulées qui peut varier. Je peux parler de l'information 16 en écrivant simplement 16, mais aussi en la désignant comme le casier *NUMERO* (si *NUMERO* contient justement 16). Dans ce cas, ce que je veux que l'exécutant manipule ce n'est évidemment pas le casier *NUMERO* lui-même, mais l'information 16 qui y est contenue.

Ainsi donc, en réponse à l'instruction

Place *SOMME* dans *TOTAL*

l'exécutant se rendra face au casier *SOMME*, **prendra copie de l'information qui y réside** (sans vider ce casier) pour aller la placer dans le casier *TOTAL*, où **elle remplacera l'information qui s'y trouvait auparavant**.

Si *SOMME* contenait l'information 16, après l'exécution de l'instruction ("Place *SOMME* dans *TOTAL*"), *SOMME* contiendra encore 16, mais *TOTAL* contiendra évidemment la même information 16.

Lorsque j'écris

*TOTAL* ← *SOMME*

il faut bien percevoir que, même s'il s'agit là de deux noms de casier, ils n'ont pas le même statut. *TOTAL* (qui est à gauche du symbole ←) est l'étiquette du casier à remplir. *SOMME* désigne en réalité, non le casier proprement dit, mais l'information (forcément constante) qui est contenue à l'intérieur du casier d'étiquette *SOMME*.

On comprend mieux pourquoi, dans le texte de la marche à suivre, la mention d'une constante de type chaîne de caractères s'accompagne d'apostrophes. C'est essentiellement pour distinguer ces constantes des noms de casier.

En effet, si j'écris

Place *NOM* dans *CLIENT*

l'exécutant recopie le contenu du casier *NOM* dans le casier *CLIENT*; tandis que, face à l'instruction

Place 'NOM' dans *CLIENT*

il se contente de sortir la chaîne 'NOM' de sa malle à informations pour la placer dans le casier *CLIENT* (qui doit bien entendu être du type adéquat). Ce sont les apostrophes qui lui précisent qu'on lui parle d'une constante chaîne de caractères et non d'un casier.

### 3. Une expression

Je vais devoir ici quelque peu anticiper sur la suite, écrivant par exemple :

Place 12 + *X* dans *DEBUT*.

L'information à placer dans le casier *DEBUT* prend ici la forme de ce que nous appellerons une **expression**.

Face à cette instruction et pour autant que *X* et *DEBUT* soient des casiers de type adéquat (par exemple de type entier), l'exécutant va :

- prendre 12 dans la malle à informations;

- aller chercher copie du contenu du casier *X*;
- additionner (nous savons qu'il en est capable) les deux informations recueillies;
- placer l'information résultante dans le casier *DEBUT* où elle prend la place de celle qui s'y trouve.

Si donc X contenait l'information 10, l'information à placer dans *DEBUT* et que nous désignons ici par 12 + *X*, est en réalité le nombre 22. Après les quelques manipulations provoquées par l'écriture 12 + *X*, l'exécutant se retrouve avec "en main" l'information 22 et c'est elle qu'il place dans *DEBUT*.

Voici quelques autres expressions :

*C* + 1

(2 * *SALAIRE* ) + 120

Nous en découvrirons bien davantage dans la suite.

En résumé,

**La première instruction d'action élémentaire est l'instruction d'affectation (= remplissage d'un casier)**.

Elle s'énonce :


Un peu de vocabulaire avant de poursuivre. Même si le terme casier semble bien adéquat pour désigner ce que j'ai justement appelé jusqu'à présent "casier", ce n'est pas ce mot qui a été retenu dans le vocabulaire informatique.

On parlera plutôt de **variable**. Ainsi, partout où se trouve écrit le mot casier, il est préférable de parler de variable, du moins, si on veut se conformer aux habitudes et au jargon de la programmation.

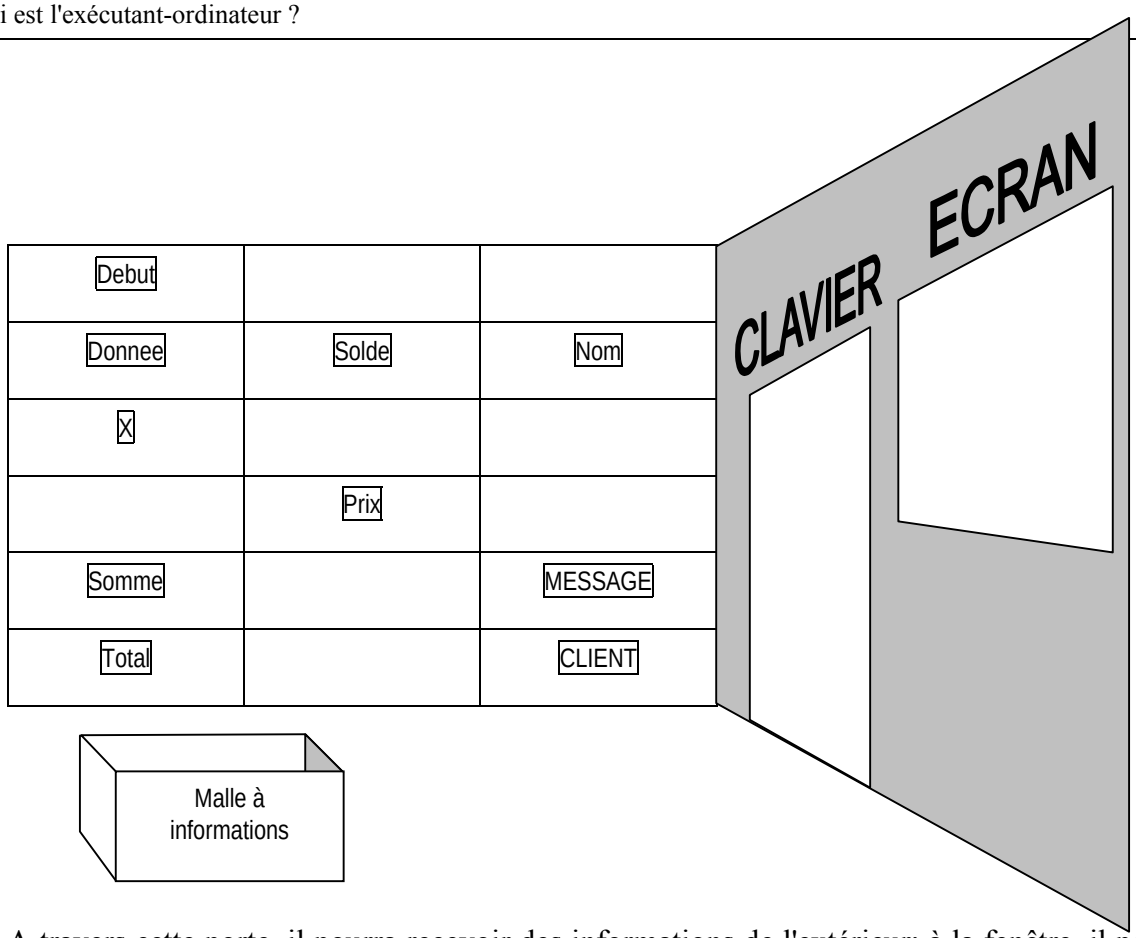
> Le vocable "variable" me semble très mal choisi. Ce qu'il y a de variable dans un casier ce n'est certes ni son nom (son étiquette) ni son type. Quant à son contenu, il reste aussi constant et inchangé, tant qu'une instruction (d'affectation par exemple) ne demande pas qu'il soit modifié. Si l'information 'Anne' occupait le casier (= la variable) *NOM*, il y a quelques instants, c'est toujours elle que j'y trouverai à présent et c'est elle qui subsistera jusqu'à ce que je commande (à travers la marche à suivre) de placer autre chose dans *NOM*.

Nous savons, à présent, que l'environnement de notre exécutant, ce sont ces casiers dans lesquels il va déposer des informations ou en reprendre (des copies). Son portrait, cependant, n'est pas encore complet : il reste à ajouter quelques caractéristiques essentielles.

## 2. Il peut communiquer avec l'extérieur

Le dessin va s'enrichir d'une porte et d'une fenêtre,




A travers cette porte, il pourra recevoir des informations de l'extérieur; à la fenêtre, il pourra afficher des informations destinées à être lues à l'extérieur. L'autre nom de la porte, c'est évidemment "clavier" et celui de la fenêtre, c'est bien sûr "écran". C'est à travers la porte-clavier que l'exécutant saisira les données attendues, c'est sur la fenêtre-écran qu'il affichera les résultats obtenus.

Il reste seulement à préciser les instructions par lesquelles je commanderai à l'exécutant de se rendre à la porte pour y prendre les informations fournies ou d'afficher à la fenêtre d'autres informations.

## *2.1 L'instruction d'entrée*

Il s'agit là de la **deuxième instruction d'action élémentaire**.

C'est l'instruction d'entrée ou de lecture, celle qui commande à l'exécutant de placer dans un casier une information recueillie à la porte et venant de l'extérieur :

**Va à la porte, attend l'information qu'on va te fournir et place-la dans tel casier.**

que nous abrégerons par :

**Lis et place dans tel casier.**

C'est à nouveau pour sacrifier à une habitude bien enracinée en programmation que je parle d'instruction de lecture et que j'écris "Lis et place dans ..." En réalité, l'exécutant ne lit pas plus l'information reçue à travers la porte qu'il ne la lit quand il la sort de la malle à informations. Cette instruction lui commande simplement de remplir un casier avec une information qu'il reçoit à travers le clavier.

1. Remarquons qu'en même temps que nous commandons à l'exécutant de "prendre" une information à travers la porte-clavier, nous sommes tenus de lui indiquer dans quel casier il va immédiatement déposer l'information reçue. Il ne peut faire subir aucun traitement à

l'information qu'il a "en main", en revenant de la porte : il peut seulement la déposer dans le casier indiqué.

2. Ce qu' "on" passera à l'exécutant à travers cette porte, c'est bien évidemment la seule chose qu'il soit capable de manipuler : une information.

   Jamais donc, au cours de l'exécution de la marche à suivre, on ne pourra lui fournir une instruction, un ordre. Pour les instructions, c'est trop tard : les seules dont il dispose sont celles enfermées dans la marche à suivre et celles-là il n'est pas question de les compléter ou de les corriger alors, qu'il est justement en train d'exécuter cette marche à suivre.

   > Il y a eu une phase préalable au cours de laquelle j'ai fourni à l'ordinateur les instructions constituant la marche à suivre : c'est lors de l'édition. Une fois, cette phase (et celle de la compilation) terminée, l'ordinateur dispose sous une forme adéquate du texte de la marche à suivre et, dès que je lui commande d'exécuter celle-ci, il devient le gestionnaire de casiers, manipulateur d'informations, qui n'attend plus (éventuellement) de l'extérieur que des informations à placer dans l'un ou l'autre casier.

3. Si, obéissant à l'instruction d'entrée qui lui est donnée (dans la marche à suivre dont il dispose), l'exécutant se rend à la porte pour y recueillir une information, il faut bien qu'il y ait "quelqu'un" de l'autre côté de cette porte pour fournir cette information. Ce "quelqu'un", nous l'appellerons l'**utilisateur**. De tout le processus en cours, ce dernier ne voit que ce qui est affiché à la fenêtre. Ce qui est attendu de lui, c'est seulement qu'il fournisse, chaque fois que l'exécutant vient entrouvrir la porte, l'information attendue.

   > Il est important de bien percevoir, au cours des multiples étapes du travail qui conduisent de la définition d'une tâche à son exécution par l'ordinateur, les divers intervenants. Il y a bien sûr, le programmeur (le mot étant employé au sens large, on dit aussi analyste) : le concepteur de la marche à suivre, celui qui a répondu successivement aux questions "QUOI FAIRE ?", "COMMENT FAIRE ?", "COMMENT FAIRE FAIRE ?", "COMMENT DIRE ?", débouchant sur l'obtention d'un programme. Jusque là, l'ordinateur-machine n'est pas nécessaire. Le programmeur se mue alors en opérateur : il se trouve face à un écran et un clavier, à travers lesquels il fournit à l'ordinateur le programme obtenu. Cela fait, lui ou quelqu'un d'autre va utiliser ce programme ou, pour être plus précis, utiliser l'ordinateur gouverné par ce programme. Il se transforme alors en utilisateur. Restent face à face, dès ce moment, cet utilisateur (celui qui est derrière la porte et la fenêtre) et l'exécutant-ordinateur ("celui" qui "voit" cette porte et cette fenêtre de l'intérieur de son local rempli de rayonnages).

4. Les informations que manipulera l'exécutant-ordinateur proviennent donc de deux sources :

   - de sa malle à informations : il les en sort lorsque la marche à suivre lui en parle. Le programmeur a parfaitement prise sur elles : c'est lui qui les a explicitement mentionnées;
   - de l'extérieur, à travers la porte. C'est l'utilisateur qui les détermine; le programmeur, au moment où il conçoit la marche à suivre, ne sait pas ce qu'elles seront précisément.

## *2.2 L'instruction d'affichage*

L'exécutant peut se rendre à la porte pour y prendre de l'information. Il reste à préciser comment l'envoyer à la fenêtre afficher d'autres informations. Il s'agit là de la **troisième instruction d'action élémentaire**.

C'est l'instruction de **sortie** ou d'**affichage** :

**Affiche (à la fenêtre) telles informations.**

Ainsi, on commandera, par exemple :

Affiche 'Donnez-moi votre nom'

Affiche 112.5

l'information affichée prenant ici la forme d'une constante

ou

Affiche *Total*

Affiche *NOM*

l'information étant ici celle contenue dans les casiers indiqués (*Total* et *NOM*)

ou encore

Affiche 2 * *Total*

Affiche 12 + (4 * *Somme*)

où l'information affichée se présente comme une expression

ou enfin, par un mélange de toutes ces modalités.

Affiche 'Total ', *Total*

Affiche 'Voici les résultats: ', *SOMME*, *PRODUIT*

1. On le voit à nouveau, l'exécutant ne peut afficher que ce qu'il manipule : des informations. Mais je peux lui en parler comme de constantes, de noms de casiers (ce qu'il affiche alors, c'est l'information qui y est contenue), ou encore comme des expressions (qu'il évalue pour en déduire les informations qu'elles représentent). C'est une règle générale qu'une information se désigne toujours sous l'une des trois formes :

   - constante;
   - nom de casier;
   - expression.

2. Il faut bien saisir que lorsque j'écris "Affiche 'Donnez-moi votre nom' " je demande simplement à l'exécutant d'afficher une chaîne de caractères constitué de la succession D, o, n, n, etc. Même si elle a un sens pour l'utilisateur qui est de l'autre côté de la fenêtre, du point de vue de l'exécutant (qui ne sait pas ce qu'est un mot), ça n'a pas plus de sens que d'exécuter :

   Affiche 'Turlututu'

   ou

   Affiche 'A???R!EZ!'.

3. Les informations dont je commande l'affichage sont les seules choses qui apparaissent à l'utilisateur. Rien de ce qui se passe à l'intérieur du local où l'exécutant est en train de s'agiter au milieu des casiers n'est visible "de l'extérieur" ni a fortiori le texte de la marche à suivre qui provoque cette agitation. L'ignorance (pour le programmeur) de cette réalité conduit à des situations loufoques du genre de la suivante :

   Pour l'une ou l'autre raison, je souhaite que l'exécutant recueille une information qui est le nom de l'utilisateur et je voudrais qu'il la dépose dans le casier Client. Je commande donc dans la marche à suivre :

   Lis et place dans Client.

L'exécutant va donc entrouvrir la porte et attend, attend, attend ... que l'utilisateur, qui lui ne sait rien de tout cela puisque je n'ai pas commandé d'affichage, lui fournisse son nom. Les deux protagonistes attendent donc, chacun d'un côté de la porte, l'un (l'utilisateur) ne sachant pas pourquoi la porte s'entrouvre à ce moment-là, l'autre (l'exécutant) qu'on veuille bien fournir l'information souhaitée.

J'aurais donc dû écrire dans la marche à suivre, par exemple :

Affiche 'Donnez-moi votre nom'

Lis et place dans Client.

A l'exécution, l'utilisateur verra d'abord s'afficher à la fenêtre le message :

Donnez-moi votre nom

et la porte s'entrouvrant immédiatement après, il sait qu'il doit y glisser DUPONT ou DURAND ...

Avec l'affichage, je viens de terminer la liste des instructions d'action élémentaire; elles sont donc seulement au nombre de trois :

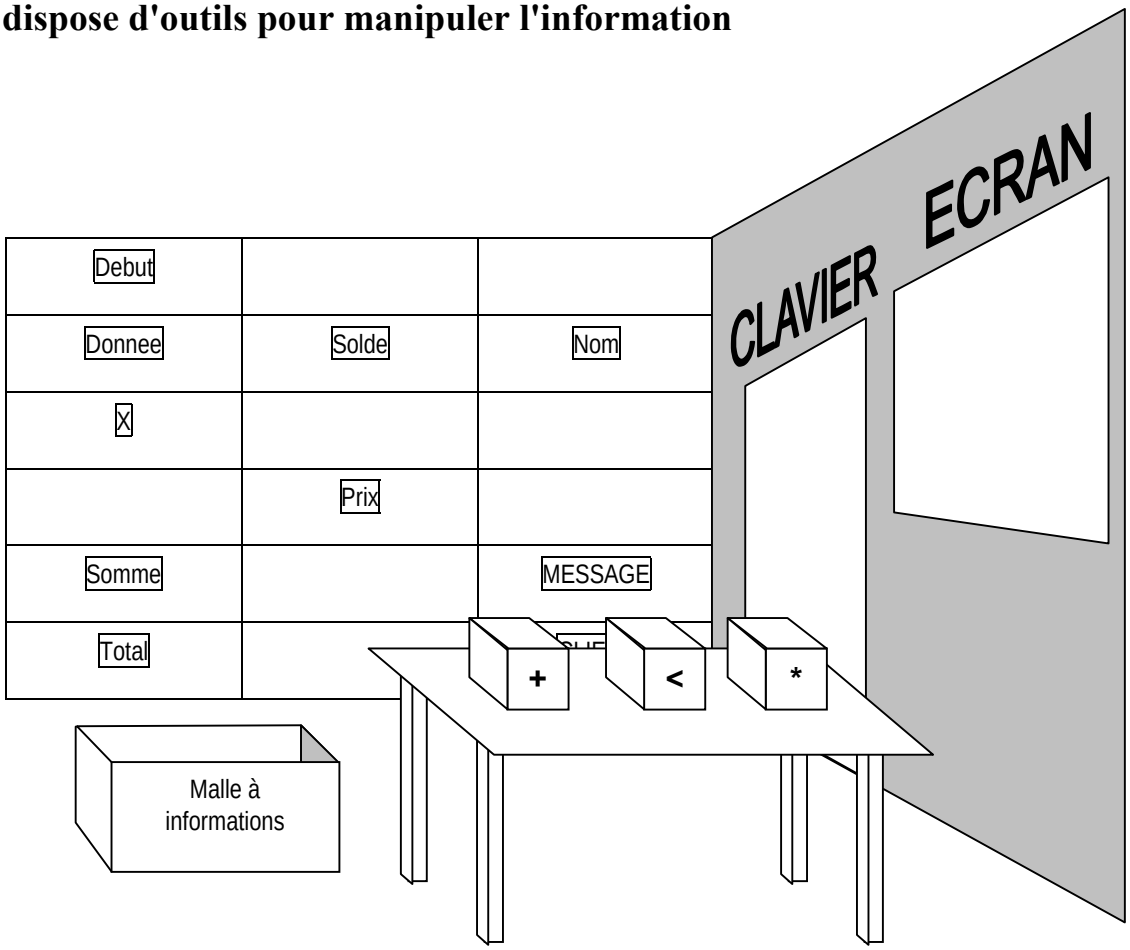
**L'exécutant-ordinateur est capable des trois actions commandées par les instructions suivantes :**

1. **Place telle information dans *tel casier***
2. **Lis et place dans *tel casier***
3. **Affiche telles informations**

Son portrait n'est pas pour autant complet. J'ai, à quelques reprises, signalé qu'une information pouvait être décrite comme une expression : mélange de constantes, de noms de casiers et de quelques symboles : +, *, -, ... ; ce sont ces derniers dont il faut à présent parler.

## 3. Il dispose d'outils pour manipuler l'information




Les rayonnages, la porte et la fenêtre se complètent d'une table de travail sur laquelle sont disposés quelques outils au nombre desquels une "additionneuse", une "multiplieuse", une "recolleuse" de chaînes de caractères, etc.. Ce sont ces outils qui interviendront dans l'évaluation des expressions.

> Cette "table de travail" couverte d'outils a bien entendu certains rapports avec le processeur équipant l'ordinateur-machine. Mais les manipulations d'informations dont le processeur est capable sont bien moins nombreuses et moins riches que celles permises par les langages de programmation (et donc celles de l'exécutant-ordinateur dont je parle ici). Rappelons qu'entre les deux, il y a une traduction et, dès lors, une manipulation qui se décrit fort simplement en Pascal peut donner lieu à tout un ensemble d'opérations successives au niveau du processeur. Mais cela, ce n'est absolument pas mon problème, à moi, programmeur.

1. Plus que pour les autres caractéristiques de l'exécutant vues jusqu'à présent, la présence (ou l'absence) et la dénomination de ces outils vont dépendre du langage de programmation dans lequel j'exprimerai les marches à suivre. Ainsi, si je lui parle en Pascal (version UCSD ou TURBO), l'outil à recoller ensemble deux chaînes de caractères s'appelle CONCAT; si je lui parle en Basic, il est le plus souvent simplement désigné par + (comme l'addition des nombres).

2. De toute manière, mon propos, à ce stade, n'est absolument pas de dresser une liste exhaustive des divers outils à manipuler les informations dont dispose l'exécutant : ce serait indigeste et inutile. Nous les découvrirons au fur et à mesure des besoins, motivés par l'exécution de certaines tâches.

   > Connaître la liste des outils qui vont permettre les manipulations de l'information, c'est (finalement) indispensable, mais ce n'est pas cela savoir programmer. Il est évidemment impératif de connaître les outils auxquels l'exécutant pourra avoir recours (et auxquels nous

aurons recours, à travers lui). Si j'ignore qu'il est capable d'additionner des nombres, ou d'évaluer le nombre de caractères composant une chaîne, je le croirai incapable de certains traitements. Mais cette connaissance, indispensable, ne suffit pas : programmer c'est davantage décortiquer une action complexe en une succession d'actions élémentaires (où interviennent ces outils), c'est décider quand et comment ces outils seront utilisés.

3. Ces outils sont évidemment adaptés aux informations à manipuler. Cela n'étonnera personne de savoir, qu'en ce qui concerne les nombres, ils ont pour nom :

   \+ (addition);

   * (multiplication);

   / (division);

   \- (soustraction);

   ...

   Ceux servant à manipuler les chaînes de caractères sont plus difficilement prévisibles (et beaucoup plus dépendant du langage compromis choisi).

4. Parmi ces outils, il en est dont je vais reparler dans un instant (à propos des structures de contrôle) et qui sont fort importants; ce sont ceux qui servent aux diverses comparaisons :

   = égal;

   < strictement plus petit;

   > strictement plus grand;

   ≤ plus petit ou égal;

   ≥ plus grand ou égal;

   ≠ différent.

Voici donc notre exécutant au milieu des rayonnages avec les casiers étiquetés, prêt à se rendre (si la marche à suivre le lui commande) à la porte ou à la fenêtre, disposant sur sa table de travail d'un certain nombre d'outils. Il ne reste qu'une chose à ajouter, pour être certain qu'il sera capable d'exécuter nos marches à suivre :

## 4. Il "comprend" les structures de contrôle

1. Les marches à suivre voyaient les diverses instructions d'action élémentaire mariées entre elles, organisées, par les structures de contrôle (telles que nous les exprimions dans les GNS, par exemple).

   L'exécutant est bien sûr capable d'organiser les actions élémentaires commandées, en suivant les indications de ces structures :

   - séquence;
   - appel de procédure;
   - alternative;
   - répétition;
   - branchement.

Ici aussi, je l'ai déjà dit, les structures de contrôle permises lors de la phase de rédaction du programme proprement dit (lors de l'étape "Comment dire ?", lorsque la marche à suivre sera codée dans le langage) dépendent du langage de programmation choisi. Mais celles mises en avant dans la phase d'analyse et de rédaction de la marche à suivre (avant le codage) sont celles exprimées dans les GNS, donc essentiellement celles énumérées ci-dessus, à l'exception du branchement.

2. Corollaire de cette possibilité de "comprendre" les structures de contrôle, l'exécutant est donc capable d'évaluer les conditions qui interviennent dans les structures alternative et répétitive. Ces conditions, je le rappelle, sont des assertions dont l'exécutant peut décider si elles sont vraies ou fausses. Elles n'auront pas, pour l'exécutant-ordinateur, la diversité qu'on trouvait dans les marches à suivre rédigées jusqu'à présent (Cf. Chapitre 2). C'est un traiteur d'informations : les conditions énoncées porteront uniquement sur des comparaisons d'informations; ainsi par exemple, nous écrirons

**SI** telle information **=** telle autre information **ALORS**

...

ou encore

**REPETER**

...

**JUSQU'A CE QUE** telle information **<** telle autre information

Les outils de comparaison d'informations sont, comme annoncé plus haut :

**= < > ≤ ≥ ≠**

Ils s'appliquent aussi bien aux informations numériques qu'à celles de type chaînes de caractères. Les informations comparées seront, comme toujours, décrites soit comme des constantes, soit comme des noms de casiers, soit comme des expressions.

3. Ces comparaisons peuvent être liées par des mots comme **ET**, **OU** ou précédées de **NON**. Nous postulons donc que l'exécutant-ordinateur maîtrise l'emploi de ces termes et est capable d'évaluer des conditions comme :

... *Nom* = 'DUPONT' **ET** *Salaire* ≤ 5000 ...

... *X* ≠ *TOTAL* * 2 **OU** *Y* ≤ *NUMERO* ...

Il ne sert à rien de vouloir trop en dire à ce propos pour le moment. Les choses s'éclairciront avec la résolution des premiers problèmes.

**Les conditions évaluées par l'exécutant sont essentiellement des comparaisons d'informations à l'aide des symboles =, <, >, ≤, ≥, ≠.**

**Plusieurs comparaisons peuvent être connectées par les mots ET ou OU. L'emploi de NON, pour nier une condition, est également permis.**

Avec cette description des conditions qu'il peut tester, le portrait robot de l'exécutant-ordinateur est à présent complet. Notre tâche de programmeur sera d'abord de décider quels casiers seront nécessaires pour l'exécution d'une tâche (qui sera toujours un traitement d'informations). Disposant des trois instructions d'action élémentaire vues, elle consistera ensuite à décortiquer cette

tâche en ses constituants élémentaires et à l'organiser à l'aide des structures de contrôle (sous forme GNS).

J'ajouterai cependant encore un trait important de l'exécutant-ordinateur.

**L'exécutant-ordinateur est un amnésique.**

Je ne pourrai donc, à travers la marche à suivre, lui parler qu'à l'impératif présent. Je devrai proscrire des mots comme "tantôt", "auparavant", "successivement".

Sa seule "mémoire" c'est, à tout moment, le contenu actuel des casiers. Pas question de lui demander "Est-ce que tantôt le casier Machin ne contenait pas telle information ?". Il ne voit que le présent et n'est "conscient" que de ce qui est enfermé dans les casiers (que je lui aurai fait gérer).

## 5. Exercices

1. Voici à un moment donné la configuration de casiers, tous de type entier :

| A | B | C |
|---|---|---|
| 12 | 19 | 14 |
| D | | |
| 19 | -12 | 5 |

Quelle sera la situation de ces casiers après exécution de

a) A ← B

B ← A

b) B ← A

A ← B

c) C ← A + B

B ← A + B

A ← B

d) B ← A

C ← B

D ← C

A ← D

2. Je souhaite que l'exécutant échange les contenus des deux casiers A et B. Quelles instructions comportera la (petite) marche à suivre que je dois dès lors lui soumettre.
3. Quel est l'effet de la marche à suivre suivante :

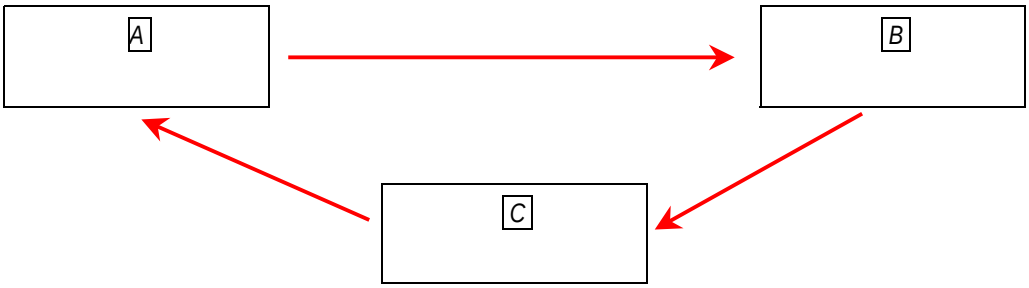
A ← A + B

B ← A - B

A ← A - B

4. Je souhaite une permutation circulaire des contenus des casiers A, B et C. Donc




Quelle marche à suivre dois-je lui fournir ?

5. Critiquez et commentez les instructions suivantes :

   Lis et place dans 'NOM'

   *SOMME* - 5 ← *TOTAL*

   *RESULTAT* ← 'SOMME' + 15

6. Ecrire la (petite) marche à suivre qui permettra à l'utilisateur d'obtenir la somme et la différence de deux nombres réels qu'il fournira.

7. Pour chacun des petits programmes suivants, et étant donné les informations fournies par l'utilisateur, quelles sont les valeurs qu'il verra affichées ?

   a) Programme

   X et Y sont des casiers de type nombre entier

   Lis et place dans *X*

   Lis et place dans *Y*

   Lis et place dans *X*

   Place X + Y dans *X*

   Affiche X, Y

   L'utilisateur fournit 2, puis 3, puis 4

   b) Programme

   X et Y sont des casiers de type entier

   Lis et place dans *X*

   Place 0 dans *Y*

   Place *X* * *Y* dans *X*

   Affiche *X*, *Y*

   L'utilisateur fournit 2

8. Expliquez les termes suivants :

   - type d'un casier
   - utilisateur

9. Que manipule l'exécutant-ordinateur ?

# 5

# Tours de main

Nous savons à présent que "programmer" ce sera "rédiger une marche à suivre (comportant instructions d'action et structures de contrôle) pour un exécutant gestionnaire de casiers, manipulateur d'informations communiquant avec l'extérieur et disposant de quelques outils".

Il reste maintenant à présenter un certain nombre d'exemples illustrant la démarche de traitement informatique d'une tâche. Le chemin, nous le savons, sera à chaque fois balisé par les étapes mises en évidence : "QUOI FAIRE ?", "COMMENT FAIRE ?", "COMMENT FAIRE FAIRE ?" et "COMMENT DIRE ?". Cette dernière étape, traduction en Pascal de la marche à suivre obtenue sous forme GNS, ne sera cependant décrite qu'au chapitre suivant.

Les premières tâches abordées seront particulièrement simples (pour ne pas dire ridicules). Elles seront peu motivantes, conduisant en général à des programmes qui ne seront pas utilisés. Connaissant les outils de conception d'une marche à suivre et les instructions d'action élémentaire permises pour l'exécutant-ordinateur, il s'agit à présent, non de traiter des tâches utiles ou passionnantes, mais d'**acquérir un certain nombre de tours de main de la programmation**.

Il faut donc "jouer le jeu" : lorsque l'apprenti-plafonneur débute dans le métier, on ne lui confie pas d'emblée le traitement d'un bâtiment dans son ensemble; on lui fait acquérir les "trucs du métier". Il travaille "pour rire" (même si ce n'est pas très amusant de passer un long moment à travailler "pour rien"). C'est pareil pour l'apprenti-programmeur : il ne s'agit pas (encore) de programmer des tâches intéressantes, utiles ou motivantes ! Il faut (bêtement) acquérir, à l'occasion de tâches élémentaires, les premiers tours de main de la programmation.

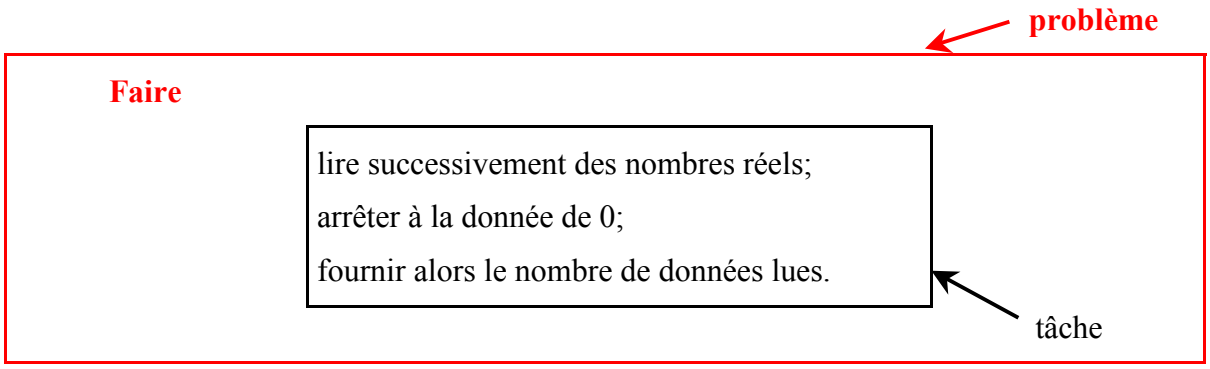
A côté des exercices proposés (et résolus), on trouvera, comme d'habitude, également une série d'exercices qui permettront à l'apprenti-programmeur de se faire la main. C'est uniquement par un travail personnel de résolution de ces exercices que vous pourrez progresser. La simple "compréhension" des solutions apportées aux exercices traités ici est illusoire : c'est seulement en agissant que l'apprentissage sera effectif.

## 1. Traitement des exemples

### *1.1 Le tour de main du "compteur"*

La première étape consistera à préciser quelle est la tâche proposée : que faut-il faire (ou plutôt faire faire). Les tâches abordées ici sont tellement simples, que cette étape de spécification sera immédiate. Il ne faudrait pas en déduire qu'il en sera toujours ainsi. Dans le cas de vrais problèmes, c'est en général l'étape la plus longue et la plus difficile.

### 1.1.1 Enoncé (Quoi faire ?)




Ce qui est attendu est suffisamment clair pour ne pas nécessiter de développements plus longs. On peut imaginer que quelqu'un ("l'utilisateur") va nous dicter des nombres; lorsque c'est 0 qui sera fourni, on attend simplement que nous répondions en précisant combien de nombres auront été fournis au total. Cette **tâche** est débile; elle ne devient un **problème** que parce que son énoncé commence par "Faire..." Le problème, c'est de **rédiger la marche à suivre** qui expliquera à l'exécutant-ordinateur comment accepter des nombres (fournis à travers la porte-clavier par l'utilisateur), les compter et donner à l'apparition de 0 (qui doit être compté aussi) le résultat sur la fenêtre écran.

Il reste dès lors à passer à l'étape suivante : comment réagirions-nous, face à cette tâche ? Quelle serait notre méthode ?

### 1.1.2 Comment faire ?

Cette étape est celle où, nous imaginant aux prises avec la tâche, nous sommes capables d'exprimer les stratégies que nous employons. Le travail ici est tellement élémentaire, que nous sommes bien en peine d'indiquer comment nous nous y prenons. Je vais, dès lors, préciser un peu le contenu de cette étape, en donnant essentiellement la consigne suivante : que retenez-vous lorsque vous êtes en train d'effectuer la tâche décrite. Autrement dit, quelles sont les informations que vous gardez dans un coin de votre mémoire, pendant que vous écoutez la série de nombres qu'on vous dicte et pendant que vous les comptez ?

> Cette réflexion débouchera, dans l'étape suivante, sur l'établissement de la liste des casiers (variables) nécessaires à l'exécutant-ordinateur. Nous savons en effet qu'il est amnésique et, qu'à tout instant, sa seule "mémoire", c'est le contenu des casiers que nous aurons choisi de lui faire utiliser.

Ainsi, si l'on me dicte des nombres, avec comme consigne préalable de dire, à l'apparition du nombre 0, combien j'en ai reçu au total, il est inutile que j'essaye de retenir tous ces nombres. Simplement, je retiendrai chacun d'eux un court instant, le temps de le compter et de m'apercevoir que c'est ou ce n'est pas 0.

Aux prises avec l'exécution de cette tâche, je garderais donc en mémoire :

- chaque donnée **pendant un court instant**,
- le décompte des données, qui évoluera au fur et à mesure qu'elles sont fournies (je les compte !).

On pourrait aussi ajouter à cette liste le fait suivant : c'est le nombre 0 qui nous fera arrêter les lectures; ceci reste également présent dans notre mémoire, tout au long du travail. Cette connaissance n'est cependant pas susceptible d'être modifiée en cours de route, et cela à la différence des diverses données et de leur décompte. Une telle information "constante" si elle figure bien dans

la liste de ce que nous retenons ne donnera pas naissance à un casier (à contenu variable). Les autres informations retenues (et qui se modifieront au fur et à mesure du travail) se traduiront, elles, dans l'étape suivante, par la définition des variables correspondantes.

> Cette étape de recherche de stratégies, où nous devons être capables, face à une tâche, d'expliciter complètement comment nous nous y prenons, sera en général moins anodine. Pensez à la difficulté d'expliquer, complètement et de manière méthodique, la manière dont vous triez des nombres, dont vous écrivez en toutes lettres un nombre, ou dont ... vous comptez les points au tennis !
>
> Pour l'instant, la seule utilité de cette étape, c'est se regardant faire aux prises avec les tâches décrites, d'être capable de préciser les informations que nous gardons à l'esprit pour en venir à bout. Nous saurons ainsi, à l'étape suivante, quels seront les casiers nécessaires.

Nous pouvons à présent passer à la suite : nous allons tenir vraiment compte des caractéristiques de l'exécutant, pour rédiger la marche à suivre qui lui expliquera comment traiter cette tâche à notre place.

### 1.1.3 Comment faire faire ?

L'exécutant est un gestionnaire de casiers. Il est impératif, dès lors, de commencer par préciser quels casiers lui seront indispensables pour effectuer la tâche désirée. Rappelons que c'est un amnésique. Nous ne pourrons lui parler qu'à l'impératif présent et sa seule "mémoire", à tout instant, c'est le contenu des casiers que nous aurons demandé de réserver !

**Liste des variables (casiers)**

Un casier sera évidemment nécessaire pour accueillir les données successivement fournies par l'utilisateur. Comme chacune de ces données peut être oubliée à l'apparition de la suivante, cet unique casier suffira. La donnée lue (= reçue à travers la porte-clavier) y résidera le temps nécessaire, avant d'être remplacée par la suivante. Ces données seront des nombres réels : le casier destiné à les recevoir, l'une après l'autre, sera donc de type réel. Pour rappeler les informations qui y seront contenues, je l'étiquetterai *NombreFourni*.

Ainsi, les caractéristiques de ce casier sont les suivantes :

étiquette (nom) : *NombreFourni*,

type : réel,

contenu : l'une après l'autre, chaque donnée lue.

De plus, je dois faire retenir le décompte des nombres fournis. Pour cela, je prévois un casier qui s'augmentera de 1, à chaque nouvelle donnée; il sera de type entier et je le baptise *Compteur*;, puisqu'il servira à compter !

Ses caractéristiques sont donc les suivantes :

étiquette : *Compteur*,

type : entier,

contenu : le nombre de données déjà reçues.

> A ce stade, nous gardons toute liberté quant au choix du nom des casiers utilisés. C'est seulement lors du codage de la marche à suivre en Pascal, que les contraintes syntaxiques feront leur apparition. Nous sacrifierons cependant, dès à présent, aux prescriptions grammaticales de Pascal concernant les noms des variables : ils ne comporteront que des lettres (majuscules ou minuscules mais non accentuées) et des chiffres, à l'exclusion de tout autre symbole (comme l'espace, le tiret, ...) et débuteront par une lettre. C'est pour cette raison que je parle ci-dessus de *NombreFourni* et pas de Nombre Fourni ou *Nombre-Fourni*

> Au-delà de ces (petites) contraintes orthographiques, le plus important, quant au choix des noms de variables, c'est bien sûr d'opter pour des termes liés aux informations qui y seront contenues. Mieux vaut donc *NombreFourni* que *X*, *N* ou *Salaire* et *Compteur* que *U*, *I* ou *Age* !

On le voit, ces casiers correspondent bien aux informations que je devais à tout moment garder en tête pour effectuer le travail :

| **je retenais** | **casiers** |
|---|---|
| chaque donnée (un instant) | *NombreFourni* |
| le nombre de données lues | *Compteur* |

Il reste seulement à préciser comment ces casiers seront utilisés, en un mot quelle est la

**Marche à suivre**

Nous allons demander à l'exécutant de répéter essentiellement les mêmes actions : d'afficher à la fenêtre-écran un court "message"[1] invitant l'utilisateur à donner un nombre, d'aller lire ce nombre à partir de la "porte-clavier" (pour le déposer dans le casier *NombreFourni*) et de le compter (en augmentant le contenu du casier *Compteur*). Et cela, il devra le répéter jusqu'au moment où le nombre lu[2] (et déposé dans *NombreFourni*) sera 0. Il suffira alors de lui demander d'afficher, sur la vitrine-écran, à l'intention de l'utilisateur, le contenu du casier *Compteur* (qui contiendra alors le nombre de données lues).

Ainsi la marche à suivre pourrait être :

| |
|---|
| Affiche 'Donnez un nombre' |
| Lis et place dans *NombreFourni* |
| Place *Compteur* + 1 dans le casier *Compteur* |
| *NombreFourni* = 0 |
| Affiche *Compteur* |

```
REPETER
  Affiche 'Donnez un nombre'
  Lis et place dans NombreFourni
  Place Compteur + 1 dans le casier Compteur
JUSQU'A CE QUE NombreFourni = 0
Affiche Compteur
```

Il subsiste cependant un petit problème : lorsque l'exécutant commence le travail décrit par cette marche à suivre, les casiers contiennent n'importe quoi (n'importe quelle information du type correspondant au type du casier envisagé). Il est facile de voir, dans ces conditions, que si le casier *Compteur* contient (primitivement et par hasard) l'entier 12 et si l'utilisateur fournit les nombres 3.5, -4.66 et 0, il lui sera répondu qu'il a fourni 15 nombres au total (puisqu'on commence à compter avec un *Compteur* qui contient déjà 12 !).

Il est donc impératif que l'exécutant commence bien son travail avec un *Compteur* contenant 0 et que, dès lors, je lui demande, avant de commencer à répéter les lectures et comptages successifs, d'y placer 0. Cette étape de réflexion à propos du contenu préalable des casiers est indispensable : c'est l'**initialisation** des casiers.

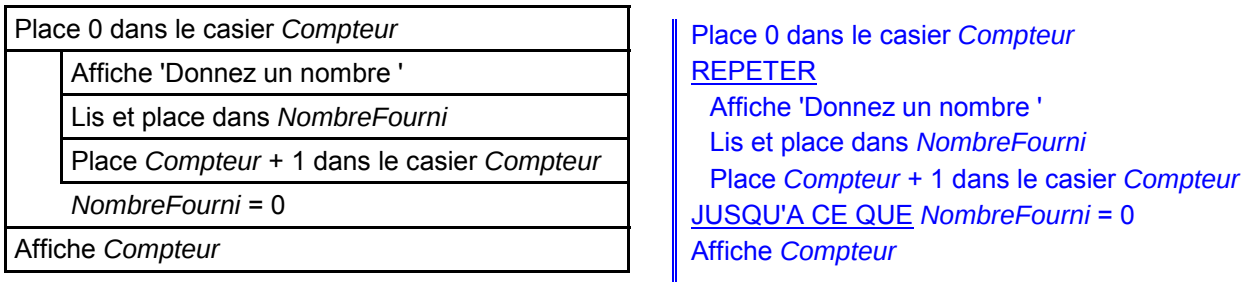
Ici, le contenu primitif de *Compteur* doit être 0; celui de *NombreFourni* est sans importance, puisque la première chose que je commande de faire à propos de ce casier, c'est d'y placer une information (lue au clavier). Dès lors, la marche à suivre complète et conforme aux spécifications est

---

1 Du point de vue de l'exécutant, il s'agit bien sûr d'une information constante de type chaîne de caractères.

2 Je rappelle (une dernière fois) que l'action désignée sous le nom de "lecture" est celle où l'exécutant vient chercher une information à la porte-clavier.




**Remarques**

Les outils (de la table de travail) mis en oeuvre dans ce problème sont :

\+ qui fournit la somme de deux nombres;

= qui permet la comparaison de deux informations.

Disposant à présent de la liste des casiers nécessaires et de la marche à suivre correspondante, il resterait à traduire tout ceci dans le langage compromis, "compréhensible" par l'ordinateur. Nous réserverons au chapitre suivant la réalisation de cette étape.

On le verra, l'essentiel est à présent terminé. Nous disposons de la marche à suivre qui va permettre à l'exécutant d'effectuer la tâche décrite, à notre place. Et il ne nous reste plus qu'à ... passer à la tâche suivante.

## *1.2 Le tour de main de la variable signal*

### 1.2.1 Enoncé (Quoi faire)

Faire :

- lire successivement des mots;
- arrêter à la lecture de "STOP", à condition que "ATTENTION" ait été fourni (n'importe quand) auparavant;
- donner alors le nombre de mots lus au total.

On le voit, à nouveau, cette tâche est particulièrement simple et peu intéressante. Le problème de programmation auquel elle conduira sera ... moins simple et ... (je l'espère) plus intéressant.

Cet énoncé comporte en tout cas un terme inadéquat : on y parle de lire des "mots". Ce que l'utilisateur fournira à chaque fois, c'est bien entendu une "chaîne de caractères", qu'il s'agisse vraiment d'un mot ou d'une succession insensée de caractères. Ce qui devra provoquer l'arrêt, c'est la chaîne "STOP" (écrite en majuscules et sans espace supplémentaire), pour autant que, de manière tout aussi formelle, "ATTENTION" ait été fourni auparavant. Ainsi donc,

| | | |
|---|---|---|
| Attention | et | STOP, c'est fini |
| ou | | ou |
| ATTENTION! | | stop |

ne conviennent pas. Ce qu'il faut tester, c'est la réception de "ATTENTION", tel quel, à un moment, puis celle de "STOP", exactement écrit de cette manière.

L'énoncé, strictement équivalent à celui-ci, où l'on aurait commandé la lecture de nombres avec arrêt à 0, à condition que 1 ait été fourni auparavant, aurait sans doute été plus clair, mais il n'aurait pas permis ces premières manipulations élémentaires de chaînes de caractères.

### 1.2.2 Comment faire ?

Comme pour la tâche précédente, ce qui importe c'est, se regardant faire, aux prises avec ce travail, de déterminer quelles sont les informations que nous sommes forcés de retenir et qui sont susceptibles de se modifier.

Deux approches (au moins) sont sans doute possibles :

1. On ne retient qu'un court instant le mot lu. Dans un premier stade, on est seulement attentif à l'apparition de "ATTENTION"; dès réception de ce mot, on passe alors à un second stade où l'on est attentif à l'arrivée de "STOP", qui arrête tout le processus. On compte les mots au cours de la première étape et on continue ce décompte au cours de la seconde.

Ou encore :

2. On peut aussi retenir un instant chaque mot, mais retenir en plus, à tout moment, si "ATTENTION" a ou n'a pas encore été fourni. On s'arrête lorsque "STOP", apparaît, à condition qu'on se souvienne que "ATTENTION" est passé. De toute manière, on compte les mots à chaque fois.

Quelle que soit l'approche envisagée, on retient évidemment les mots "ATTENTION" et "STOP" (auxquels on compare les mots lus) mais ces deux informations ne se modifieront pas plus au cours de la tâche que le 0 provoquant l'arrêt du problème précédent.

### 1.2.3 Comment faire faire ?

**Première solution**

Comme précédemment, je dois d'abord, sur base des informations que j'étais forcé de retenir à l'étape du "Comment faire?", décider quels seront les casiers indispensables.

Il en faudra seulement deux :

- celui qui contiendra successivement chacun des mots fournis à l'exécutant à travers la porte clavier et que je baptiserai pour cette raison *MotFourni*. Il doit être du type chaîne de caractères;

  A nouveau, j'ai décidé de respecter dès à présent, certaines contraintes orthographiques qui seront imposées lors de la traduction en Pascal, en écrivant par exemple *MotFourni* et pas *Mot Fourni* ou *Mot-Fourni*. Le plus important, c'est de choisir un nom de variable qui informe sur son contenu : j'aurais pu choisir, *Mot*, *Donnee*, ...

- celui qui servira à compter : *Compteur*, de type entier.

  Notons une dernière fois, que tout au long du traitement de ce problème, chaque fois que je parlerai de "mot", il s'agira bien entendu, du point de vue de l'exécutant, d'une chaîne de caractères.

  De plus, lors de la définition d'un casier de ce type, je préciserai la longueur maximale des chaînes qui pourront y prendre place. Il s'agit là d'une contrainte qui n'apparaîtra vraiment que lors de la traduction en Pascal[3]. J'en tiens cependant compte dès à présent pour faciliter ce travail d'expression (Comment dire ?). On verra dans la suite que cette longueur maximale à préciser pour les variables de type chaîne de caractères doit être comprise entre 1 et 255.

Ainsi donc, je peux à présent dresser la

*Liste des variables*

*MotFourni*, de type chaîne d'au plus 20 caractères;

---

[3] Lorsque l'implémentation choisie est, par exemple, Turbo Pascal Version 3. Cette contrainte disparaît d'ailleurs dans la version 4.

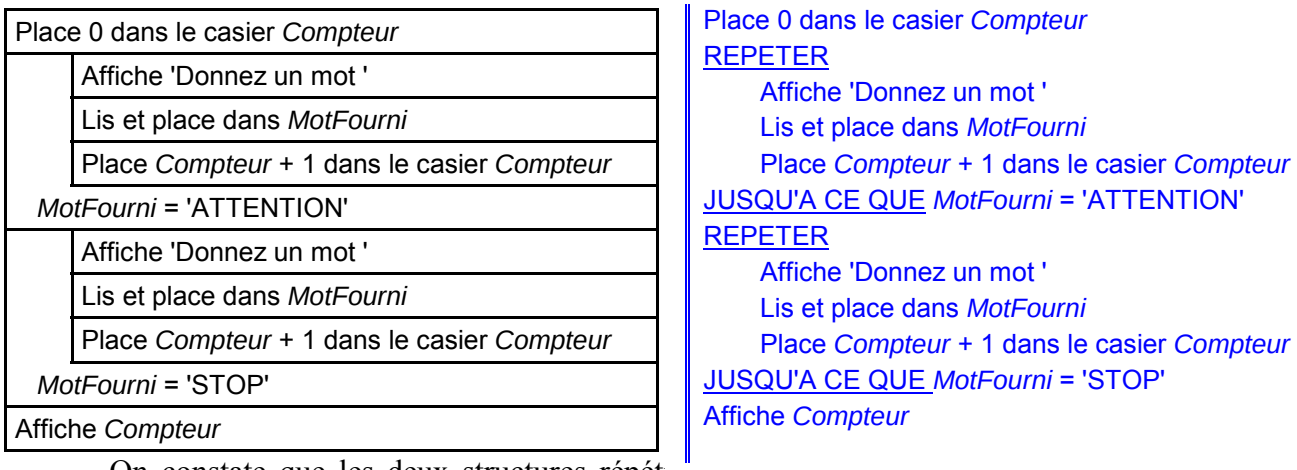
*Compteur*, de type entier.

Il reste à indiquer comment ces deux casiers seront utilisés en fournissant la

*Marche à suivre*

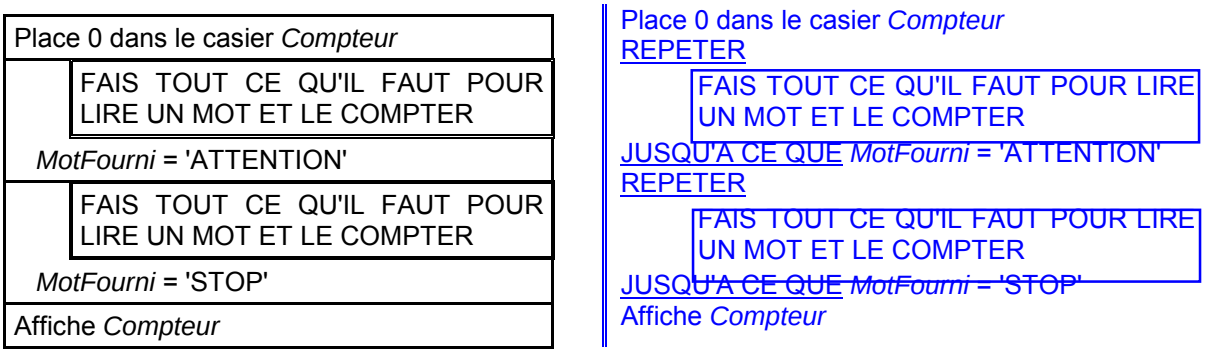
Nous allons commander deux répétitions successives des mêmes actions : lire une donnée (pour la déposer dans MotFourni) et la compter (grâce au Compteur). La première répétition s'interrompra lorsque le mot lu sera 'ATTENTION', faisant alors place à la répétition suivante, qui s'arrêtera à 'STOP'. Il ne restera plus alors qu'à demander l'affichage du contenu du Compteur ... qu'il ne faudra pas oublier, dès le début, d'initialiser.

Ceci conduit au GNS suivant :




On constate que les deux structures répétitives demandent la reprise du même groupe de trois actions. Nous pourrions donc, pour sacrifier à une certaine paresse, désigner par une seule instruction ce groupe de trois instructions élémentaires. Notre marche à suivre comporterait alors une instruction d'action complexe (appel de procédure), explicitée dans une marche à suivre annexe (procédure).

Ainsi, en désignant par "FAIS TOUT CE QU'IL FAUT POUR LIRE UN MOT ET LE COMPTER" l'instruction complexe (qui se scindera en trois instructions élémentaires), la marche à suivre devient :




avec

Procédure annexe : COMMENT "FAIRE TOUT CE QU'IL FAUT POUR LIRE UN MOT ET LE COMPTER "

| |
|---|
| Affiche 'Donnez un mot ' |
| Lis et place dans *MotFourni* |
| Place *Compteur* + 1 dans le casier *Compteur* |

Procédure annexe : COMMENT "FAIRE TOUT CE QU'IL FAUT POUR LIRE UN MOT ET LE COMPTER "

Affiche 'Donnez un mot '
Lis et place dans *MotFourni*
Place *Compteur* + 1 dans le casier *Compteur*

Il ne faudrait pas croire que l'utilisation des procédures ne se fera que lorsqu'une portion de marche à suivre apparaîtra à divers endroits, pour sacrifier (comme ci-dessus) à une certaine paresse. On le verra, ce concept permettra d'incarner dans nos analyse la démarche descendante.

Rappelons que, face à cette marche à suivre, l'exécutant s'intéressera d'abord au texte principal, en détournant son regard vers les explications fournies dans la procédure annexe, chaque fois qu'il rencontrera l'instruction inconnue "FAIS TOUT CE QU'IL FAUT POUR LIRE UN MOT ET LE COMPTER". Après avoir exécuté les actions prescrites dans l'annexe explicative, il reviendra à la suite de la marche à suivre principale.

**Deuxième solution**

Dans cette seconde approche, une seule répétition sera commandée. Elle s'arrêtera lorsque le mot fourni sera 'STOP' et que l'exécutant "gardera le souvenir" du passage de 'ATTENTION'. Le seul problème, c'est donc qu'il "garde en mémoire" le fait que 'ATTENTION' a ou n'a pas encore été lu. Mais, pour cela, il suffit d'un casier dont le contenu signalera que ce mot a (ou n'a pas) été fourni. Ce casier, je le baptiserai *Signal*. J'ai le choix en ce qui concerne son type. L'important, c'est qu'il contienne une certaine information tant que 'ATTENTION' n'a pas été reçu et qu'à la réception de ce mot le contenu de ce *Signal* se modifie. Ainsi, il pourrait être de type entier, contenant primitivement 0 (ou toute autre valeur entière) et basculant à 1 (ou à toute autre valeur différente de celle s'y trouvant), lors du passage de 'ATTENTION'. Il pourrait aussi être de type réel, mais j'ai choisi ici le type chaîne (d'au plus 5 caractères). *Signal* contiendra la chaîne 'VERT' tant que le mot 'ATTENTION' n'aura pas été reçu et ce contenu deviendra 'ROUGE' (pour le rester) à la réception du mot 'ATTENTION'.

Ainsi, on vérifiera, à la lecture de chaque mot, qu'il s'agit oui ou non de 'ATTENTION' pour faire éventuellement basculer le contenu de *Signal* de 'VERT' à 'ROUGE'. Les lectures (accompagnées de ces vérifications et du comptage) seront répétées jusqu'à ce que le mot fourni soit 'STOP' et que le casier *Signal* "soit" 'ROUGE'.

Cette utilisation d'une variable "signal" (on dit aussi "drapeau" ("baissé" ou "levé") ou encore "sentinelle" ("endormie" ou "éveillée")) est fréquente en programmation. Elle est indispensable, chaque fois qu'un "événement" doit "rester en mémoire" chez l'exécutant amnésique.

Nous avons choisi ici un *Signal* de type chaîne de caractères, plutôt qu'entier ou réel. Nous verrons dans la suite un type de casier se prêtant particulièrement bien à cet usage : le type booléen.

A côté du casier *Signal*, un casier *MotFourni* (de type chaîne d'au plus 20 caractères) est évidemment indispensable (pour accueillir les mots fournis par l'utilisateur), ainsi que le traditionnel *Compteur* (de type entier).

Nous pouvons donc dresser la

*Liste des variables*

- *MotFourni*, de type chaîne d'au plus 20 caractères,
- *Compteur*, de type entier,

- *Signal*, de type quelconque, pour autant qu'il puisse contenir une information au début et basculer en accueillant une autre information dans la suite. Ici, je l'ai choisi de type chaîne d'au plus 5 caractères, passant de 'VERT' à 'ROUGE'.

Ayant "réservé" les casiers indispensables, il reste, comme d'habitude, à indiquer comment ils seront utilisés.

*Marche à suivre*

Il suffit de commander une répétition au cours de laquelle :

- un mot nouveau sera lu et placé dans *MotFourni*;
- *Compteur* verra son contenu augmenter de 1;
- le *Signal* passera à 'ROUGE' si *MotFourni* contient 'ATTENTION'.

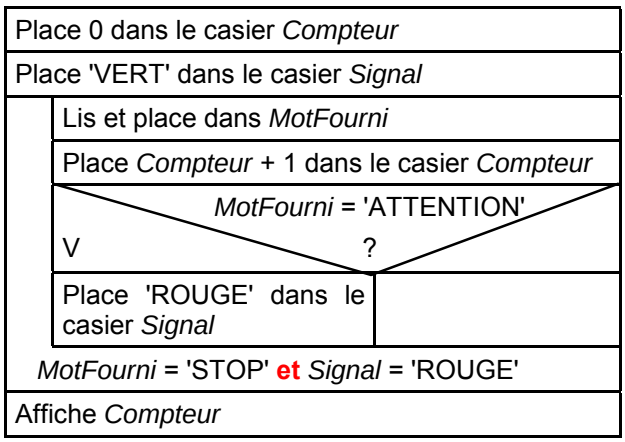
> Lorsqu'on décrit la liste des instructions qui devront être répétées (on dit aussi "le corps de la boucle de répétition"), il ne faut surtout pas la dresser en s'inspirant de la toute première fois que les actions correspondantes seront effectuées. Il faut en quelque sorte "prendre le train en marche". On suppose, pour écrire le corps de la boucle, que les actions sur lesquelles porte la répétition ont déjà été effectuées quelques fois et l'on écrit ce que sera ce corps lors d'une quelconque des répétitions. C'est ainsi que, dans le texte ci-dessus, on parle de "mot nouveau", supposant dès lors qu'il s'en est déjà présenté et donc que les actions à répéter l'ont déjà été quelques fois. On s'intéresse ensuite à la condition d'arrêt de la boucle et aux actions qui doivent préparer la répétition (initialisations).
>
> En d'autres termes, on écrira les actions à répéter au temps présent, tout en se disant que ces actions qu'on décrit ont déjà été réalisées dans le passé et qu'elles se poursuivront dans le futur (jusqu'à ce que la condition d'arrêt en stoppe la reprise).

Cette répétition s'interrompra, lorsque le dernier mot lu (contenu dans *MotFourni*) sera 'STOP' et que *Signal* sera 'ROUGE'.

Comme toujours, il faudra commencer par les initialisations indispensables :

- *Compteur* doit primitivement receler 0 et
- *Signal* contenir 'VERT'.

Nous sommes en mesure, dès lors, de rédiger la marche à suivre correspondante :




```
Place 0 dans le casier Compteur
Place 'VERT' dans le casier Signal
REPETER
    Lis et place dans MotFourni
    Place Compteur + 1 dans le casier Compteur
    SI MotFourni = 'ATTENTION' ALORS
        Place 'ROUGE' dans le casier Signal
JUSQU'A CE QUE MotFourni = 'STOP' et Signal = 'ROUGE'
Affiche Compteur
```

*Remarques*

1. On a tout naturellement utilisé ici, dans l'énoncé de la condition de fin de répétition, le mot "et". La condition énoncée ne sera vraie que si les deux comparaisons la constituant le sont. Cet outil (comme aussi "ou"), permettant de lier des comparaisons est connu de l'exécutant

et autorise à énoncer des conditions plus ou moins élaborées. L'outil "non" (qui permet de nier une condition) est un peu du même genre.

2. Retenons ici le "tour de main" de la variable-signal, fréquent en programmation. Il intervient chaque fois qu'il est nécessaire de "faire retenir" qu'un événement s'est produit.

## *1.3 Le dernier et le précédent*

La tâche est, on va le voir, bien proche de la précédente :

### 1.3.1 Enoncé (Quoi faire ?)

Faire :

- lire successivement des mots (d'au plus 20 caractères),
- arrêter à la donnée de "STOP", à condition que "ATTENTION" ait été fourni **juste avant**,
- dire alors combien de mots ont été lus.

Les étapes essentielles de la démarche ont été longuement illustrées et commentées sur les deux exemples précédents. Je me permettrai à présent d'être un peu plus laconique.

### 1.3.2 Comment faire ?

1. On retient un instant chaque mot lu, en étant particulièrement attentif au mot suivant lorsque c'est "ATTENTION" qui vient d'être lu. Si le mot suivant n'est pas "STOP", alors on "oublie" que "ATTENTION" venait d'être lu. De plus, on compte tous les mots.

Ou encore

2. On retient un instant le dernier mot lu et l'avant-dernier. On compte au fur et à mesure.

### 1.3.3 Comment faire faire ?

**Première solution**

Je vais faire usage, à nouveau, d'un signal qui détectera le passage de 'ATTENTION'. *Signal* passera à 'ROUGE' à la lecture de 'ATTENTION'. Mais ce 'ROUGE' ne comptera que si, à la lecture suivante, le mot 'STOP' est fourni. Sinon, *Signal* doit repasser au 'VERT'. Autrement dit, lorsque le mot lu n'est pas 'ATTENTION' et que le *signal* est 'ROUGE' (donc 'ATTENTION' vient d'être lu), il est impératif qu'il repasse à 'VERT', si le mot lu n'est pas 'STOP'. (Ouf ... !)

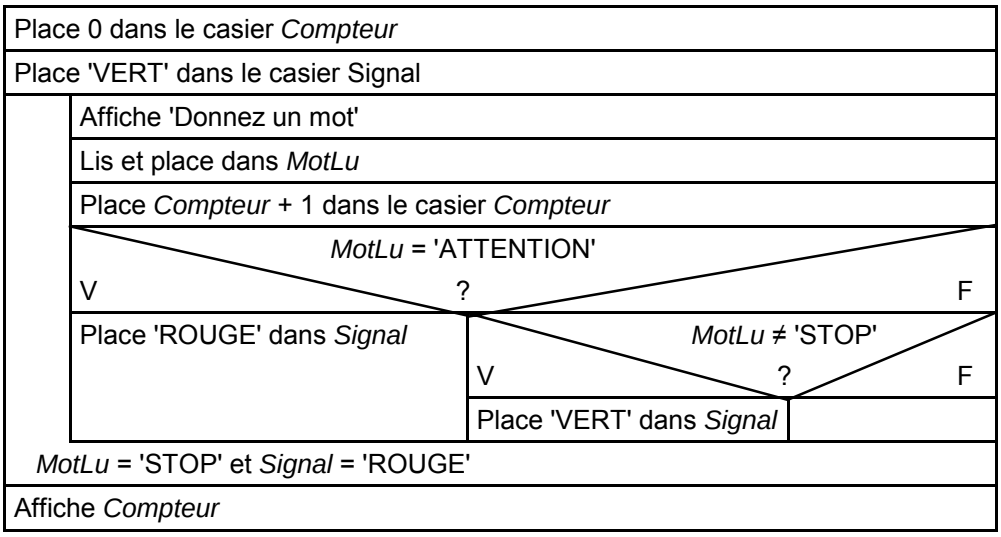
> Cette approche, qui privilégie une nouvelle utilisation de variable signal est, on en conviendra, assez complexe, pour ne pas dire obscure. On le verra, la deuxième solution semble nettement plus simple et "naturelle". Rappelons encore que "lu" signifie ici que l'exécutant-ordinateur saisit l'information correspondante à travers la porte-clavier.

*Liste des variables*

- *MotLu*, de type chaîne d'au plus 20 caractères,
- *Compteur*, de type entier,
- *Signal*, de type chaîne d'au plus 5 caractères. Il passera au 'ROUGE' si 'ATTENTION' est lu, mais en restant 'ROUGE', seulement si c'est 'STOP' qui suit; sinon, il repasse au 'VERT'.

*Marche à suivre*




ou sous forme pseudo-code :

```
Place 0 dans le casier Compteur
Place 'VERT' dans le casier Signal
REPETER
        Affiche 'Donnez un mot'
        Lis et place dans MotLu
        Place Compteur + 1 dans le casier Compteur
        SI MotLu = 'ATTENTION' ALORS
                Place 'ROUGE' dans Signal
        SINON
                SI MotLu ≠ 'STOP' ALORS
                        Place 'VERT' dans Signal
JUSQU'A CE QUE  MotLu = 'STOP' et Signal = 'ROUGE'
Affiche Compteur
```

**Deuxième Solution**

Elle correspond à l'attitude consistant à retenir chaque mot, ainsi que le précédent, tout en les comptant. Trois variables seront dès lors nécessaires : celle contenant le dernier mot lu, celle contenant le précédent et celle permettant de compter.

*Liste des variables*

- *Dernier*, de type chaîne d'au plus 20 caractères, qui contiendra le dernier mot lu,
- *AvantDernier*, de type chaîne d'au plus 20 caractères, qui contiendra l'avant-dernier mot,
- *Compteur*, de type entier.

*Marche à suivre*

La seule difficulté, c'est de faire en sorte, qu'au "bon moment", le contenu de *Dernier* soit transféré dans *AvantDernier*.

Après les initialisations d'usage, je commanderai de répéter un même groupe d'actions. On va voir que l'ordre dans lequel ces actions sont commandées est, ici, fort important. Si je commande de répéter :

- lis un mot et place-le dans *Dernier*,

- recopie *Dernier* dans *AvantDernier*,
- compte.

et cela, jusqu'à, ce que *Dernier* contienne 'STOP' et *AvantDernier* contienne 'ATTENTION', j'aboutis à une absurdité, puisque *Dernier* et *AvantDernier* contiendront à coup sûr le même mot.

En réalité, il faut d'abord faire transférer le contenu de *Dernier* dans *AvantDernier* **avant** de faire lire un nouveau mot pour le placer dans *Dernier* (et non après, comme ci-dessus). Je commanderai donc la répétition des actions suivantes (cette fois dans le bon ordre) :

- recopie *Dernier* dans *AvantDernier*,
- lis un mot et place le dans *Dernier*,
- compte.

Avec l'habituelle initialisation du *Compteur*, ceci conduit au GNS :

| Place 0 dans le casier *Compteur* |
|---|
| Place *Dernier* dans le casier *AvantDernier* |
| Affiche 'Donnez un mot' |
| Lis et place dans *Dernier* |
| Place *Compteur* + 1 dans le casier *Compteur* |
| *Dernier* = 'STOP' et *AvantDernier* = 'ATTENTION' |
| Affiche *Compteur* |

```
Place 0 dans le casier Compteur
REPETER
    Place Dernier dans le casier AvantDernier
    Affiche 'Donnez un mot'
    Lis et place dans Dernier
    Place Compteur + 1 dans le casier Compteur
JUSQU'A CE QUE Dernier = 'STOP' et
    AvantDernier = 'ATTENTION'
Affiche Compteur
```

Malheureusement, un détail rend ce programme incorrect. Imaginons un instant que par un improbable et malheureux hasard, le casier *Dernier* contienne, avant le début des opérations, le mot 'ATTENTION' et que le tout premier mot fourni par l'utilisateur soit 'STOP'. L'exécutant va successivement poser les actions suivantes (commandées par la marche à suivre) :

- mettre 0 dans *Compteur*,
- recopier le contenu de *Dernier* (il y "traîne" 'ATTENTION') dans *AvantDernier*, qui contiendra alors, lui aussi, 'ATTENTION',
- lire le premier mot ('STOP') et le placer dans *Dernier*,
- augmenter le contenu de *Compteur*, qui passe donc à 1,
- voir si *Dernier* contient bien 'STOP' (c'est le cas) et *AvantDernier* contient 'ATTENTION' (c'est aussi le cas) et donc interrompre immédiatement la répétition,
- afficher 1, contenu de *Compteur* et arrêter.

Il est facile de voir qu'il faut à tout prix éviter que *Dernier* contienne primitivement 'ATTENTION' et prendre la peine de le faire initialiser en y plaçant par exemple 'BONJOUR'.

> Ceci nous enseigne qu'il faut obligatoirement être fort attentif à ce problème d'initialisation des casiers. Lorsque la première manipulation commandée à propos d'un casier consiste à en prendre le contenu (comme ce serait le cas ici, si je n'avais pas fait les initialisations, pour *Dernier* et *Compteur*), il faut impérativement réfléchir à l'importance de son contenu primitif. Par contre, lorsque la première manipulation faisant intervenir un casier commande d'y placer une information, son initialisation est, généralement, inutile.

Cette précaution conduit au GNS :

| |
|---|
| Place 0 dans le casier *Compteur* |
| **Place 'BONJOUR' dans le casier *Dernier*** |
| Place *Dernier* dans le casier *AvantDernier* |
| Affiche 'Donnez un mot' |
| Lis et place dans *Dernier* |
| Place *Compteur* + 1 dans le casier *Compteur* |
| *Dernier* = 'STOP' et *AvantDernier* = 'ATTENTION' |
| Affiche *Compteur* |

```
Place 0 dans le casier Compteur
Place 'BONJOUR' dans le casier Dernier
REPETER
    Place Dernier dans le casier AvantDernier
    Affiche 'Donnez un mot'
    Lis et place dans Dernier
    Place Compteur + 1 dans le casier Compteur
JUSQU'A CE QUE Dernier = 'STOP' et
    AvantDernier = 'ATTENTION'
Affiche Compteur
```

## *1.4 Compteur et compteur*

### 1.4.1 Enoncé (Quoi faire ?)

**Faire**

simuler des jets successifs d'un dé en montrant les résultats et en arrêtant au troisième six obtenu; indiquer alors combien de jets ont été effectués.

Simuler signifie ici "faire comme si". Le programme à rédiger doit faire en sorte qu'apparaisse à l'utilisateur une succession de nombres, compris entre 1 et 6 (les résultats des "lancers"). Cette liste doit s'interrompre à l'apparition du troisième 6 et être alors suivie du nombre total de chiffres apparus.

Cet énoncé n'a évidemment de sens que si l'exécutant dispose d'un outil permettant de tirer des nombres au hasard.

Disons simplement à ce propos qu'on pourra se permettre d'écrire l'instruction d'affectation :

**Place un nombre au hasard entre 1 et 6 dans ...**

Je me permets donc de désigner une information entière à manipuler par :

un nombre (entier) au hasard entre 1 et 6,

en réservant au chapitre suivant le problème d'indiquer comment ceci sera traduit, lors du codage en Pascal.

### 1.4.2 Comment faire ?

Ce qui importe, une fois de plus, c'est de percevoir ce que je retiendrais comme informations (susceptibles de se modifier), si c'était moi qui lançais le dé.

Je garderais à l'esprit :

- un instant, chaque résultat, pour l'annoncer (le faire connaître à l'utilisateur);
- le décompte de tous les lancers, puisqu'il faudra, à la fin, pouvoir indiquer combien il y en a eu au total;
- le nombre de six déjà obtenu, pour être en mesure d'arrêter au troisième.

Ces informations retenues vont à nouveau donner naissance aux casiers nécessaires lorsqu'il s'agira de s'intéresser à l'exécutant "gestionnaire de casiers".

### 1.4.3 Comment faire faire ?

Nous pouvons donc immédiatement dresser la

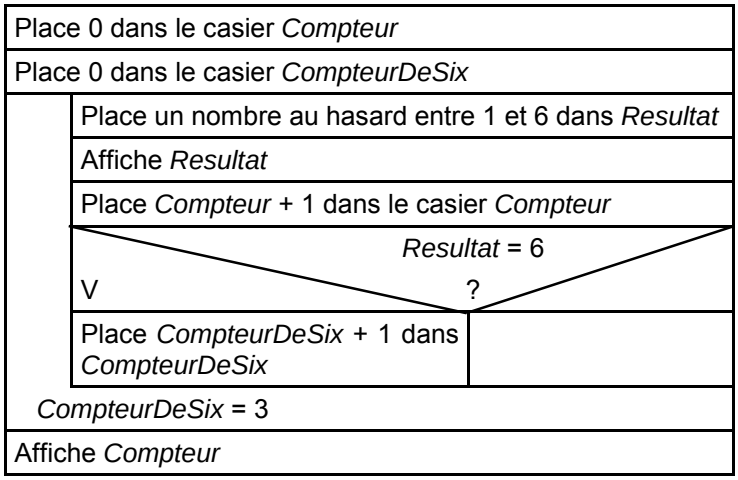
*Liste des variables*

- *Resultat*, de type entier, qui contiendra un instant chaque résultat du tirage au hasard;
- *Compteur*, de type entier, pour compter les "lancers"; il s'augmentera de 1 à chaque nouveau tirage au hasard;
- *CompteurDeSix*, de type entier, pour compter les six qui se présenteront.

> On remarquera que, pour préparer la traduction en Pascal, j'ai, dès à présent, choisi de noter "*Resultat*" (sans accent), la variable comportant les résultats des tirages. En effet, les noms de variables ne pourront, en Pascal, comporter de lettres accentuées.

*Marche à suivre*

Son écriture est immédiate. Je vais à nouveau commander une répétition au cours de laquelle un "tirage" sera effectué, affiché, compté et comptabilisé dans *CompteurDeSix*, s'il s'agit d'un 6. Cette répétition s'arrêtera lorsque le *CompteurDeSix* contiendra 3.




```
Place 0 dans le casier Compteur
Place 0 dans le casier CompteurDeSix
REPETER
    Place un nombre au hasard entre 1 et 6 dans Resultat
    Affiche Resultat
    Place Compteur + 1 dans le casier Compteur
    SI Resultat = 6 ALORS
        Place CompteurDeSix + 1 dans CompteurDeSix
JUSQU'A CE QUE CompteurDeSix = 3
Affiche Compteur
```

## *1.5 Enfin une tâche "utile"*

### 1.5.1 Enoncé (Quoi faire ?)

**Faire :**

- lire 100 données réelles,
- fournir ensuite leur moyenne, la plus petite et la plus grande donnée lue.

> Voici enfin une tâche un peu moins gratuite et un peu plus utile. Cette utilité est cependant gravement réduite par le fait que le programme réalisé ne pourra traiter qu'une succession d'exactement 100 données, ni plus ni moins. Pas question, dès lors de s'en servir pour calculer la moyenne d'une série quelconque de données.
>
> Un exercice (Cf. ci-après) tentera d'étendre ces spécifications, pour obtenir un produit réellement utilisable.

### 1.5.2 Comment faire ?

Que retiendrons-nous, face à ce travail et question cruciale, sommes-nous forcés de garder en tête, simultanément, les 100 données à traiter ?

En réalité, chacune des données ne sera retenue qu'un court instant, le temps de la traiter. Ce traitement de la donnée reçue consistera d'abord à la compter (puisqu'il faut arrêter à la centième qui sera fournie). Ensuite, pour pouvoir à la fin évaluer la moyenne, il suffira qu'au fur et à mesure je retienne la somme des données déjà acquises. Je mettrai donc cette somme à jour en y ajoutant à chaque fois la dernière donnée reçue. Enfin, à tout moment, je garderai aussi en mémoire quelle a

été jusqu'alors la plus petite et la plus grande donnée : chaque donnée nouvelle sera comparée à ces deux quantités et elles s'en trouveront éventuellement modifiées. A la fin, il suffira de diviser la somme (obtenue au fur et à mesure) par 100 pour obtenir la moyenne;.

Ainsi donc, je retiendrais :

- chaque donnée, le temps de la traiter,
- le décompte des données reçues,
- la somme des données obtenues,
- la plus petite des données,
- la plus grande.

### 1.5.3 Comment faire faire ?

Nous pouvons immédiatement dresser la

*Liste des variables*

- *Donnee*, de type réel, qui contiendra chaque donnée lue,
- *Compteur*, de type entier, qui permettra de les compter,
- *Somme*, de type réel, qui accueillera la somme des données,
- *PlusPetite*, de type réel, qui contiendra la plus petite donnée,
- *PlusGrande*, de type réel, qui contiendra la plus grande.

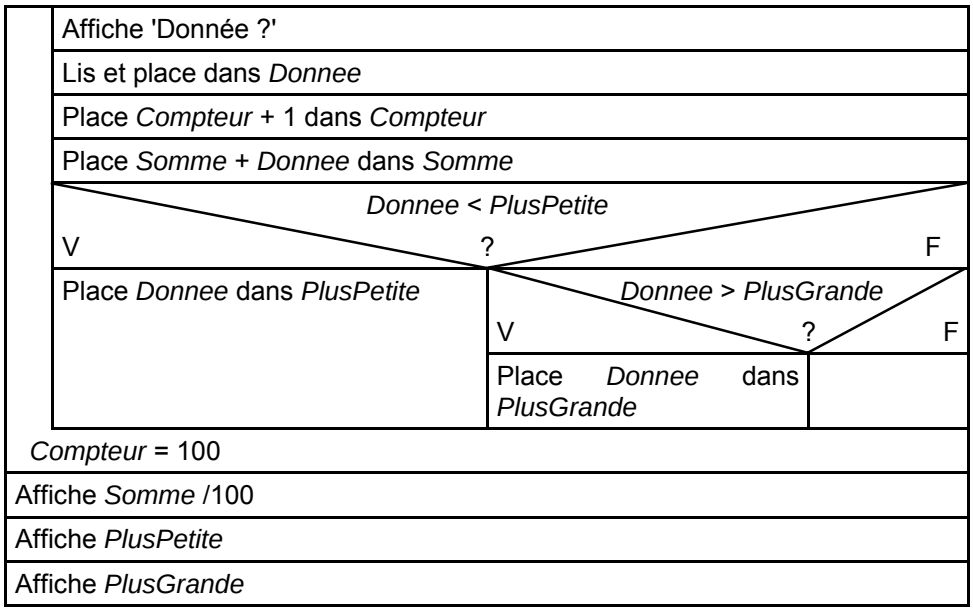
*Marche à suivre*

Le cœur de celle-ci sera à nouveau constitué d'une répétition qui comportera :

- la lecture d'une donnée (qui sera déposée) dans *Donnee*,
- l'augmentation du casier *Compteur* permettant de les compter,
- l'ajout de la *Donnee* à la *Somme*,
- la comparaison de *Donnee* à *PlusPetite* et, éventuellement, à *PlusGrande* et la mise à jour éventuelle d'un de ces casiers.

Cette répétition s'interrompra lorsque le Compteur signalera que la 100ème donnée a été acquise et traitée.

Ceci conduit au GNS :



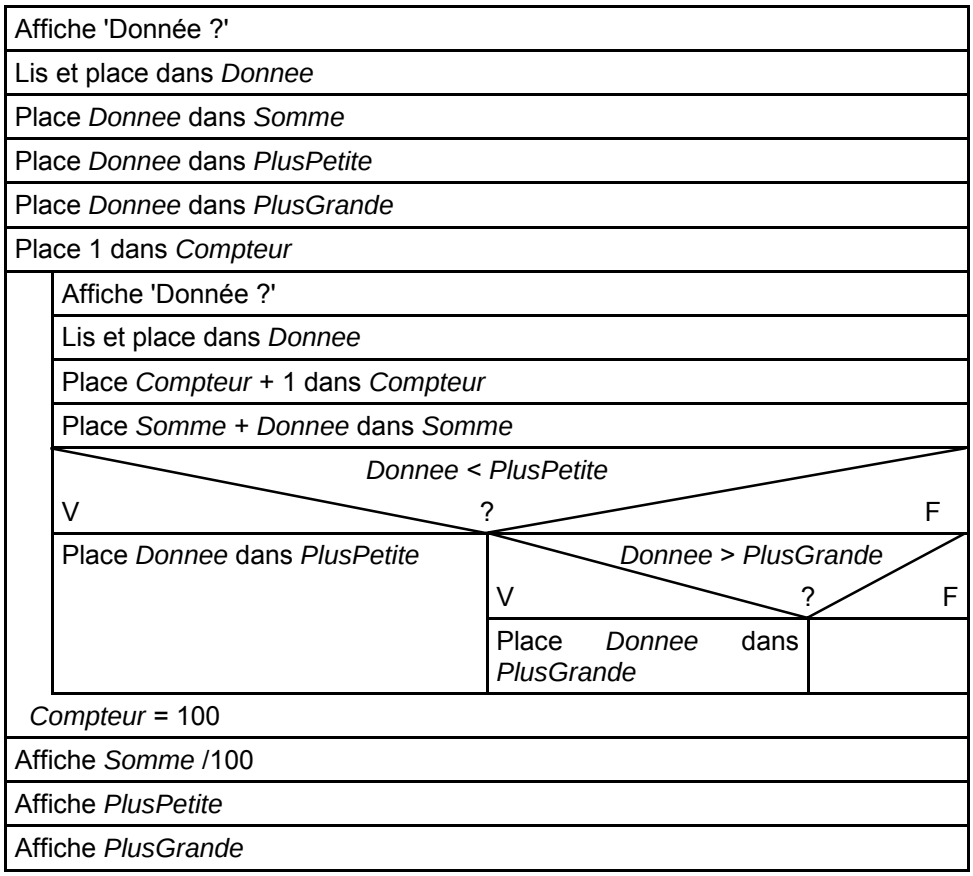

et sous forme pseudo-code :

```
REPETER
        Affiche 'Donnée ?'
        Lis et place dans Donnee
        Place Compteur + 1 dans Compteur
        Place Somme + Donnee dans Somme
        SI Donnee < PlusPetite ALORS
        Place Donnee dans PlusPetite
        SINON
                SI Donnee > PlusGrande ALORS
                        Place Donnee dans PlusGrande
JUSQU'A CE QUE Compteur = 100
Affiche Somme /100
Affiche PlusPetite
Affiche PlusGrande
```

Il reste évidemment à traiter le problème des initialisations. On voit aisément que *Compteur* et *Somme* doivent être placés à 0, avant le début des répétitions. Le cas des casiers *PlusPetite* et *PlusGrande* est plus délicat. Ainsi, si je fais placer primitivement 0 dans *PlusPetite* et si la plus petite des données fournies est 15, il sera répondu, à la fin que la plus petite est 0 (valeur primitivement contenue dans *PlusPetite* et inférieure à toutes les données reçues). En effet, à la question "*Donnee* < *PlusPetite*", il sera toujours répondu "non" et le contenu primitif de *PlusPetite* restera inchangé. On voit donc, qu'en réalité, il faudrait faire placer dans *PlusPetite* une quantité à coup sûr supérieure aux données qui seront fournies. On pourrait y placer une valeur très grande (1000000 ou plus) mais ceci conduirait à un programme erroné dans le cas (bien improbable) où toutes les données reçues seraient plus élevées que cette quantité. Le même problème se pose évidemment pour la détection de la plus grande des données.

En réalité, une approche un peu différente permet de venir à bout de cette difficulté. Il suffit de traiter un peu différemment la toute première donnée qui peut servir à initialiser *Somme*, *PlusPetite* et *PlusGrande*. On n'oubliera pas de placer 1 dans le *Compteur*. En effet, la première donnée fait l'objet d'un traitement spécifique, hors de la boucle de répétition; on commencera donc celle-ci avec une donnée déjà lue et traitée.




et sous forme pseudo-code :

```
Affiche 'Donnée ?'
Lis et place dans Donnee
Place Donnee dans Somme
Place Donnee dans PlusPetite
Place Donnee dans PlusGrande
Place 1 dans Compteur
REPETER
        Affiche 'Donnée ?'
        Lis et place dans Donnee
        Place Compteur + 1 dans Compteur
        Place Somme + Donnee dans Somme
        SI Donnee < PlusPetite ALORS
                Place Donnee dans PlusPetite
        SINON
                SI Donnee > PlusGrande ALORS
                        Place Donnee dans PlusGrande
JUSQU'A CE QUE Compteur = 100
Affiche Somme /100
Affiche PlusPetite
Affiche PlusGrande
```

Nous voici au terme de cette découverte des premiers tours de main de programmation. Il en reste bien entendu beaucoup à découvrir et à mettre en oeuvre. A présent, il faudrait surtout aborder des tâches un peu plus "utiles" ou motivantes, en y intégrant les tours de main acquis ... et à acquérir.

# 2. Exercices

## *2.1 Conception d'algorithmes*

Pour chacune des tâches suivantes, répondre aux questions :

- Comment vous y prendriez-vous, aux prises avec ce travail, en étant essentiellement attentif aux informations que vous retiendriez ?
- Quelles sont les variables indispensables, quel en est le type et quel en sera le rôle ?
- Quelle est la marche à suivre, exprimée sous forme GNS (ou pseudo-code) ?

**FAIRE**

1. Lire successivement 10 nombres réels et signaler, ensuite, combien il y en avait de négatifs.
2. Demander d'abord à l'utilisateur combien de données réelles devront ensuite être lues; lire successivement ces données et fournir ensuite le nombre de données strictement négatives, le nombre de données strictement positives et le nombre de données nulles.
3. Simuler des lancers successifs d'une pièce de monnaie jusqu'à ce que le nombre de "pile" dépasse de 3 le nombre de "face" et fournir alors le nombre de lancers qui auront été nécessaires pour cela.

   On peut utiliser l'affectation :

   Place au hasard, soit 1, soit 2 dans ....
4. Simuler 1000 lancers successifs d'une pièce de monnaie, en fournissant ensuite le nombre de "pile" et le nombre de "face" obtenus.
5. Simuler des lancers successifs d'un dé en montrant chacun des résultats et en s'arrêtant au troisième "six" obtenu. Signaler ensuite en quelle position se trouvait le premier six obtenu, en quelle position se trouvait le second, et en quelle position se trouvait le troisième.
6. Lire une valeur entière et positive N; calculer et afficher alors N! (factorielle de N).

   Rappel : N! = 1 * 2 * 3 * 4 * ... * N
7. Simuler des jets successifs d'un dé (en montrant les résultats) en arrêtant lorsqu'un "cinq" est immédiatement suivi d'un "six". Fournir alors le nombre de "un" obtenus.
8. Lire successivement des nombres réels en s'arrêtant lorsque 0 est fourni. Donner alors la somme des nombres positifs et celle des nombres négatifs.
9. Demander à l'utilisateur combien de données entières seront à lire; procéder à la lecture de ces données en précisant ensuite combien de fois il s'en est présenté qui soient identiques à la première.
10. Lire des nombres entiers, en arrêtant lorsque 0 est fourni; signaler alors combien de fois il s'est présenté deux 1 successifs. Par exemple, si la succession des nombres est :

    1 4 1 1 2 1 1 1 0

    on détectera trois paires de 1 successifs.
11. Lire des nombres entiers en arrêtant à la troisième fois que le nombre 1 est fourni et indiquer alors combien de fois le nombre 0 était présent dans la succession.

12. Vérifier la connaissance des tables de multiplication de l'(élève) utilisateur. Pour cela, proposer successivement 20 énoncés du type :

    ... x ... = ?

    les nombres y intervenant étant choisis au hasard entre 2 et 10. A chaque énoncé, lire la réponse fournie par l'utilisateur. Si cette dernière est correcte, le féliciter; si elle est fausse, le signaler et laisser une seconde chance; si elle est fausse à nouveau, donner la réponse correcte. A la fin, fournir le nombre de bonnes réponses obtenues directement (sans second essai).

13. Lire successivement des mots (d'au plus 20 caractères) en arrêtant à la donnée de "STOP" à condition que "ATTENTION" ait été fourni n'importe quand auparavant. Donner alors la position du premier et du dernier "ATTENTION" apparu.

14. Demander à l'utilisateur par quel nombre il terminera la liste des données réelles qu'il fournira. Lire alors successivement ces données (jusqu'à l'apparition de la donnée marquant la fin). Préciser ensuite combien de données n'étaient pas comprises entre -10 et 10. Attention, la dernière donnée provoquant l'arrêt ne doit en aucun cas être comptabilisée.

    Remarque : le programme ne doit pas "se planter", même si la seule donnée fournie par l'utilisateur (idiot) est justement celle qu'il avait choisie pour provoquer l'arrêt.

## *2.2 Le problème du calcul de moyenne*

J'ai signalé, en ce qui concerne le problème 5 (calcul de la moyenne), que la restriction à 100 données rendait le programme résultant peu utilisable.

Comment pourrait-on changer les spécifications pour s'affranchir de cette limitation en obtenant un programme qui puisse traiter un nombre de données quelconque ? Peut-on faire en sorte que l'utilisateur ne soit pas tenu de compter les données avant de commencer à les fournir ?

Pour chacun des énoncés auxquels conduiront ces extensions, exposer la démarche de programmation suivie en allant jusqu'à l'écriture du GNS.

## *2.3 Autres expressions de structures répétitives*

Les marches à suivre correspondant aux problèmes abordés ont essentiellement comporté des répétitions sous la forme "RÉPETER ... JUSQU'À CE QUE ...". Pouvez-vous réécrire les GNS correspondants en utilisant la structure "TANT QUE ..." ?

# 5

# TRADUCTION EN PASCAL

Avant de se retrouver (enfin) assis face à l'ordinateur, l'avant-dernière étape du traitement informatique d'une tâche est celle au cours de laquelle la marche à suivre obtenue sous forme GNS sera traduite dans un langage de programmation.

Cette étape du "Comment dire ?" ne réclame plus guère d'imagination et de recherche. Simplement, il s'agit à présent de respecter scrupuleusement les règles de grammaire et d'orthographe imposées par le langage compromis. Rien d'essentiel ne sera ajouté au contenu des GNS : nous les exprimerons différemment et avec moins de liberté quant à la syntaxe et à l'orthographe. Le seul avantage de ce type d'expression de la marche à suivre, c'est qu'il sera "compris par l'ordinateur". Cette étape est donc à la fois indispensable, pour finir par essayer vraiment les marches à suivre conçues, et fastidieuse, puisqu'il ne s'agit plus du tout de faire preuve d'invention ou de recherche, mais seulement de docilité et de rigueur formelle.

Chacun des exemples abordés et traités dans le chapitre précédent va maintenant être repris et la traduction en Pascal sera indiquée et commentée. Le traitement de ces exemples permettra évidemment de découvrir les premiers rudiments de ce langage de programmation.

Enfin, une synthèse reprendra, en fin de chapitre, de manière structurée, les éléments du langage qui, peu à peu, auront été découverts.

## 1. Traduction des marches à suivre

### *1.1 Le problème du compteur*

Il s'agissait, je le rappelle, de faire lire une série de nombres réels, jusqu'à ce que 0 apparaisse et de fournir alors le décompte de ces données. Notre analyse avait débouché sur le résultat suivant :

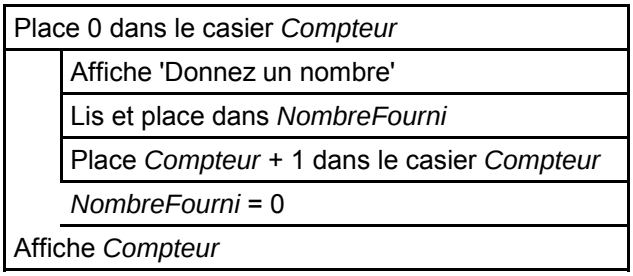
#### 1.1.1 Traduction

**Liste des variables (casiers)**

*NombreFourni*, de type réel, qui contiendra chaque donnée lue;

*Compteur*, de type entier, pour les compter.

### Marche à suivre



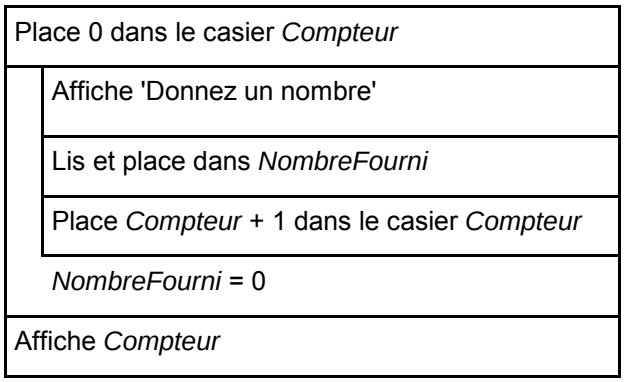

Place 0 dans le casier *Compteur*
REPETER
  Affiche 'Donnez un nombre'
  Lis et place dans *NombreFourni*
  Place *Compteur* + 1 dans le casier *Compteur*
JUSQU'A CE QUE *NombreFourni* = 0
Affiche *Compteur*

Dans un premier temps, et afin de montrer clairement qu'il s'agira (bêtement) d'exprimer différemment les structures de contrôle et les actions élémentaires présentes dans le GNS ou le pseudo-code, j'ai fait figurer côte à côte ces derniers et leur expression en Pascal.

**Liste des variables (casiers)**

*NombreFourni*, de type réel

*Compteur*, de type entier




```
program ARRETA0;
var
NombreFourni : real
Compteur : integer;
begin
Compteur := 0;
repeat
     writeln('Donnez un nombre');
     readln(NombreFourni);
     Compteur := Compteur + 1;
until NombreFourni = 0;
writeln(Compteur);
end.
```

et avec le pseudo-code :

Place 0 dans le casier *Compteur*
REPETER
  Affiche 'Donnez un nombre'
  Lis et place dans *NombreFourni*
  Place *Compteur* + 1 dans le casier *Compteur*
JUSQU'A CE QUE *NombreFourni* = 0
Affiche *Compteur*

```
program ARRETA0;
var
NombreFourni : real
Compteur : integer;
begin
Compteur := 0;
repeat
     writeln('Donnez un nombre');
     readln(NombreFourni);
     Compteur := Compteur + 1;
until NombreFourni = 0;
writeln(Compteur);
end.
```

Revoici sous une forme plus lisible le texte du programme Pascal mis en évidence. Il sera suivi d'un certain nombre de remarques explicatives.

```
program ARRETA0; [1]                                    (1)
var   NombreFourni : real ;                             (2)
      Compteur : integer ;                              (3)
begin                                                   (4)
Compteur := 0;                                          (5)
repeat                                                  (6)
      writeln('Donnez un nombre');                      (8)
      readln(NombreFourni);                             (7)
      Compteur := Compteur +1;
until NombreFourni = 0;                                 (6)
writeln (Compteur);                                     (8)
end .                                                   (4)
```

### 1.1.2 Remarques

(1) `program ARRETA0;`

Tout programme Pascal commence par un **en-tête**. Le mot réservé program est suivi de l'identification de celui-ci (ici, ARRETA0). Cet **identificateur** du programme doit commencer par une lettre, ne comporter que des lettres (majuscules ou minuscules non accentuées), le symbole de soulignement (_) et des chiffres. La même règle s'appliquera à tous les identificateurs rencontrés dans la suite (par exemple ceux désignant les variables ou les procédures).

> J'avais déjà respecté ces règles orthographiques lors de l'analyse, dans le choix des noms de variables. Je pourrai donc, sans effort, transcrire en Pascal les noms ainsi choisis.

(2)
```
var  NombreFourni : real;
     Compteur : integer;
```

La deuxième zone d'un programme Pascal est la partie **déclaration**. Pour l'instant, cette zone comporte seulement la liste des variables (casiers) utilisées. Cette liste s'annonce par le mot réservé var (qui n'est écrit qu'une seule fois). Vient ensuite la liste des divers types : integer, real, ... avec, on s'en doute, integer pour le type entier et real pour le type réel.

(3) On notera que les diverses listes de variables sont terminées par le ";". Les identificateurs des variables suivent les règles énoncées ci-dessus : ce sont celles que nous avions déjà respectées lors du choix des noms de casiers à l'étape précédente.

> Le lecteur jugera peut-être que je suis incomplet et que je n'en dis pas assez pour l'instant à propos des règles de grammaire de Pascal.
>
> A cela, je répondrai que
>
> - les règles découvertes lors du traitement des exemples seront mises en ordre à la fin de ce chapitre;
> - cet ouvrage ne constitue pas un manuel de Pascal et si j'y fais une place à l'apprentissage de ce langage, ce n'est certes pas là mon objectif principal;

[1] Les mots réservés du Pascal seront toujours soulignés et apparaîtront en gras à leur première rencontre. Il n'en sera pas de même dans les textes de programmes que vous aurez l'occasion de fournir à l'ordinateur : aucun mot n'y sera particulièrement mis en évidence. Les numéros, entre parenthèses, à droite ne font pas partie du programme.

- je ne suis pas un fanatique de grammaire !

**(4)** Vient ensuite le corps du programme, écrit entre les mots begin et end, ce dernier étant suivi du **point final**. Ce corps traduit la marche à suivre telle qu'elle est présentée dans le GNS.

**(5)**
```
Compteur := 0;
```

L'instruction d'affectation est traduite par le symbole **:=**; le nom de la variable à remplir est écrit à gauche, l'information à y placer figurant à droite.

**(6)**
```
repeat ... until NombreFourni = 0;
```

La structure de répétition REPETER ... JUSQU'A CE QUE ... est traduite par repeat liste d'instructions until ...

**(7)**
```
readln (NombreFourni);
```

L'instruction de lecture "Lis et place dans ..." est traduite par readln suivi par le nom de la variable où déposer l'information lue. Ce nom de variable doit être écrit entre parenthèses.

**(8)**
```
writeln ('Donnez un nombre :'); writeln (Compteur);
```

L'instruction d'affichage "Affiche ..." est traduite par writeln suivi par l'information (ou, on le verra, la liste des informations) à afficher, placée entre parenthèses.

**(9)** J'y reviendrai en détail dans la suite, mais dès à présent, il faut remarquer que le symbole qui permet de séparer les diverses instructions Pascal est le point-virgule. Les passages à la ligne sont syntaxiquement sans importance (en général) de même que les retraits (indentations) de certaines lignes.

> Au-delà de la syntaxe à respecter, faute de quoi des erreurs seront détectées lors de la compilation (= traduction), il faut aussi être attentif à rédiger un programme clair, bien présenté et (donc) facilement lisible. J'y reviendrai ci-dessous en parlant de l'attention à porter au lecteur du programme.

Nous voici presqu'au terme de notre démarche. Il nous reste seulement à "**habiller**" le programme obtenu.

### 1.1.3 Habillage du programme

"Habiller" un programme, ce sera d'abord être attentif à **l'utilisateur**. Ce dernier, je le rappelle, dans ma terminologie, c'est l'être humain qui se trouve de l'autre côté de l'écran et du clavier pendant que l'exécutant effectue les actions commandées par la marche à suivre. C'est lui qui lira à l'écran les informations dont je demanderai l'affichage; c'est lui qui passera à travers le clavier les informations que l'exécutant attend lors des instructions de lecture.

Dans l'état actuel du programme, l'utilisateur n'est absolument pas averti des objectifs, des données qui sont attendues, ni des résultats qui seront fournis. Tout au plus voit-il apparaître l'invitation "Donnez un nombre", correspondant à l'affichage que je commande.

En tant que programmeur, je ne suis évidemment pas en contact direct avec l'utilisateur; j'ai, en quelque sorte, délégué à l'exécutant mes prérogatives en lui fournissant la marche à suivre. C'est donc seulement à travers les instructions fournies à cet exécutant que je peux me préoccuper, en différé, de l'utilisateur.

La seule manière d'être attentif à ce dernier consiste à **commander l'affichage de messages explicatifs** qui lui indiqueront le traitement que permet le programme, les données qui sont attendues et les résultats qui seront fournis. Je vais donc ajouter au programme existant ces instructions d'affichage indispensables.

Ceci conduit à la version suivante, où les instructions supplémentaires sont en gras (et en rouge) :

```
program ARRETA0;
var   NombreFourni : real ;
      Compteur : integer ;
begin
writeln('Vous allez me fournir des nombres réels en faisant');          (1)
writeln('suivre chacun d''eux de l''appui sur la touche d''entrée.');   (2)
writeln('Vous terminerez la série par 0 et je vous dirai alors');
writeln('combien de nombres vous m''avez fournis au total.');
Compteur := 0;
repeat
      write('Donnez un nombre');                                        (3)
      readln(NombreFourni);
      Compteur := Compteur +1;
until NombreFourni = 0;
writeln ('Nombre de données fournies : ',Compteur);                     (4)
end.
```

**Remarques**

(1) Les "messages" dont nous commandons l'affichage ne sont bien entendu rien d'autre pour l'exécutant que des constantes de type chaîne de caractères. Il faut savoir qu'une telle constante peut comporter n'importe quel caractère sauf celui indiquant un passage à la ligne. Nous ne pouvons donc pas écrire :

```
writeln ('Vous allez me fournir des nombres réels en faisant
suivre chacun ...
```

C'est pourquoi chaque nouvelle ligne de message demande une nouvelle instruction d'affichage, d'où la multiplication des mots writeln.

(2) Lorsqu'une constante chaîne de caractères (qui est toujours enclose entre des apostrophes) comporte le caractère apostrophe, on est tenu de redoubler celui-ci. Lors de l'affichage, il ne sera écrit qu'une seule fois. Attention, il ne faut pas confondre l'apostrophe redoublé (' ') et les guillemets (").

(3) J'avais déjà, dans la version "nue" du programme, fait figurer un affichage sommaire invitant l'utilisateur à fournir un nombre. J'ai quelque peu modifié et enjolivé cette instruction, essentiellement en remplaçant writeln par write. L'instruction writeln provoque un passage à la ligne à la suite des informations affichées. L'instruction write ne commande pas ce passage à la ligne. Ainsi, pendant l'exécution du programme "habillé" l'écran se présentera de la façon suivante :

```
Vous allez me fournir des nombres réels en faisant
suivre chacun d'eux de l'appui sur la touche
d'entrée.
Vous terminerez la série par 0 et je vous dirai alors
combien de nombres vous m'avez fournis au total.
```

```
Donnez un nombre : 12.5
Donnez un nombre : -2
Donnez un nombre : 0
Nombre de données fournies : 3
```

les informations soulignées étant celles fournies au clavier par l'utilisateur.

(4) Nous commandons l'affichage de deux informations qui se suivront, collées l'une à l'autre, sur la même ligne. La première est la constante chaîne de caractères 'Nombre de données fournies : ' (et l'on remarquera l'espace qui la termine); la seconde est le contenu du casier *Compteur*.

Ce premier type d'habillage étant terminé, il reste à indiquer ce que sera le second. Il ne s'adressera plus à l'utilisateur mais au **lecteur** du texte même du programme. Il s'agit là de l'être humain, connaissant généralement la programmation, et à qui je transmets pour information tout le dossier de programmation d'une application (du "Quoi faire ?" au "Comment dire ?"). Le texte du programme est la dernière pièce de ce dossier et j'y ferai donc figurer des **commentaires** (remarques) pour aider ce lecteur à comprendre l'analyse que j'ai menée et le programme qui en a résulté.

Ainsi, je rappellerai les objectifs du programme construit; pour chacune des variables, j'indiquerai à quoi elle servira (les informations qui y seront contenues), ...

Cette préoccupation conduit, par exemple, au programme suivant :

```
program ARRETA0;
(*    Ce programme fait lire des nombres réels jusqu'à ce que 0 soit fourni et
      fait alors afficher combien de nombres ont été lus. *)
var   NombreFourni
      (* qui contiendra les données lues *)
      : real ;
      Compteur
      (* pour compter les données *)
      : integer ;
begin
(* Avertissement de l'utilisateur *)
writeln('Vous allez me fournir des nombres réels en faisant');
writeln('suivre chacun d''eux de l''appui sur la touche d''entrée.');
writeln('Vous terminerez la série par 0 et je vous dirai alors combien');
writeln('de nombres vous m''avez fournis au total.');
(* Initialisation du compteur *)
Compteur := 0;
repeat
      write('Donnez un nombre : ');
      readln(NombreFourni);
      Compteur := Compteur +1;
until NombreFourni = 0;
writeln ('Nombre de données fournies : ',Compteur);
end .
```

Les commentaires sont ici notés en caractères gras et condensés. On voit qu'ils sont enclos dans les symboles (* *). On pourrait aussi employer { }. Ces commentaires sont bien entendu

ignorés de l'exécutant et ne seront pas visibles pour l'utilisateur. Ils peuvent être introduits n'importe où dans le texte du programme (sauf évidemment en plein milieu d'un mot réservé ou d'un identificateur); ce ne sont pas des instructions et, dès lors, il est inutile de les faire suivre par le point-virgule.

A côté des commentaires qui explicitement sont là pour faciliter la lecture et la compréhension du programme, il faut noter la présentation utilisée : passages à la ligne, indentation (retrait) de certaines parties du programme (par exemple le corps de la boucle de répétition), passage de lignes blanches, ... Pas de règle syntaxique très stricte à ce niveau : du bon sens en tant que rédacteur et la volonté de faciliter la tâche de celui qui aura à relire et à comprendre le programme conçu.

Dans le même ordre d'idée, nous avons pris la bonne habitude de choisir des noms de variables "parlants". Rien n'est plus pénible que ces variables *X*, *Y*, *T*, *C* ... dont l'étiquette n'annonce en rien le contenu !

> Cette préoccupation de lisibilité est sans doute un peu inutile pour des programmes qui font seulement quelques lignes. Elle deviendra impérative dès que les exemples traités deviendront plus complexes et déboucheront sur des programmes plus longs. Faute de cette volonté d'être clair et lisible, les textes des programmes deviennent rapidement une jungle dans lequel le lecteur se perd ... y compris d'ailleurs lorsque le lecteur est le concepteur-même du programme qui essaye tout bêtement de se relire.
>
> C'est pour cette raison que, dès à présent, je vous recommande de prendre ces bonnes habitudes de rédaction et de présentation. Il vaut mieux avoir adopté cette attitude lorsqu'elle est encore accessoire et "facultative" que de l'ignorer lorsqu'elle devient indispensable. Nul ne saura jamais les milliers de programmes qui ont été mis à la poubelle faute qu'on puisse simplement les relire pour les modifier et les amender.
>
> **La programmation, c'est essentiellement de la "matière grise ajoutée" : il faut veiller à ce qu'elle soit facilement perçue et non la dissimuler dans des textes de programmes obscurs et illisibles.**

Nous voilà au terme du traitement de ce premier exemple. Avec l'analyse présentée au chapitre précédent, nous venons d'achever l'illustration de l'ensemble de la démarche, de la définition précise de la tâche à l'écriture et l'habillage du programme Pascal. Il reste à indiquer comment les autres problèmes analysés seront traduits en Pascal et comment les programmes résultants seront à leur tour "habillés".

## *1.2 Le problème du "signal"*

Il s'agissait, cette fois de (faire) lire et compter des mots avec arrêt à "STOP" à condition que "ATTENTION" soit survenu auparavant.

### 1.2.1 Première solution

C'était celle où deux répétitions successives étaient commandées, la première s'interrompant à la lecture de "ATTENTION", la seconde à celle de "STOP".

#### *1.2.1.1 Traduction*

Voici à nouveau, côte à côte, le GNS (tel que nous l'avions conçu dans l'étape précédente) et son expression en Pascal :

**Liste des variables**

*MotFourni*, de type chaîne d'au plus 20 caractères,

*Compteur*, de type entier.

**Marche à suivre**

| |
|---|
| Place 0 dans le casier *Compteur* |
| Affiche 'Donnez un mot ' |
| Lis et place dans *MotFourni* |
| Place *Compteur* + 1 dans le casier *Compteur* |
| *MotFourni* = 'ATTENTION' |
| Affiche 'Donnez un mot ' |
| Lis et place dans *MotFourni* |
| Place *Compteur* + 1 dans le casier *Compteur* |
| *MotFourni* = 'STOP' |
| Affiche *Compteur* |

```
program ATTENTION1;
var
    MotFourni : string[20] ;
    Compteur : integer ;

begin
Compteur := 0;
repeat
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'ATTENTION';
repeat
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'STOP';
writeln(Compteur);
end.
```

et avec le pseudo-code :

```
Place 0 dans le casier Compteur
REPETER
     Affiche 'Donnez un mot '
     Lis et place dans MotFourni
     Place Compteur + 1 dans le casier Compteur
JUSQU'A CE QUE MotFourni = 'ATTENTION'
REPETER
     Affiche 'Donnez un mot '
     Lis et place dans MotFourni
     Place Compteur + 1 dans le casier Compteur
JUSQU'A CE QUE MotFourni = 'STOP'
Affiche Compteur
```

```
program ATTENTION1;
var
    MotFourni : string[20] ;
    Compteur : integer ;
begin
Compteur := 0;
repeat
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'ATTENTION';
repeat
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'STOP';
writeln(Compteur);
end.
```

Revoici d'ailleurs le texte de ce programme assorti des habituelles remarques :

```
program ATTENTION1;
var   MotFourni : string[20] ;                                    (1)
      Compteur : integer ;
begin
Compteur := 0;
repeat
```

```
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'ATTENTION';                                    (2)(3)
repeat
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'STOP';                                         (2)
writeln(Compteur);
end.
```

#### 1.2.1.2 Remarques

(1) `var MotFourni : string[20] ;`

Le type "chaîne d'au plus 20 caractères" se traduit en Pascal par string[20]. Bien évidemment, il est possible de définir des variables chaînes de caractères en comportant un nombre maximal différent : il suffit de faire figurer ce nombre entre crochets à la suite du mot string; il s'agit toujours du nombre maximal de caractères que pourra comporter une chaîne accueillie dans cette variable. La limite de ce nombre maximal de caractères est cependant fixée à 255.

(2) `until MotFourni = 'ATTENTION';`

Nous avions, jusqu'à présent, fait comparer des nombres à l'aide du symbole =. On peut aussi faire comparer des informations de type chaîne de caractères.

(3) Les constantes de type chaîne de caractères doivent, rappelons-le une fois de plus, être écrites entre apostrophes.

#### 1.2.1.3 Habillage

Il reste à présent à habiller ce programme en tenant compte de l'utilisateur :

```
program ATTENTION1;
var  MotFourni : string[20] ;
     Compteur : integer ;
begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l''appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
Compteur := 0;
repeat
     write ('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur +1;
until MotFourni = 'ATTENTION';
repeat
     write ('Donnez un mot ');
```

```
      readln(MotFourni);
      Compteur := Compteur +1;
until MotFourni = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

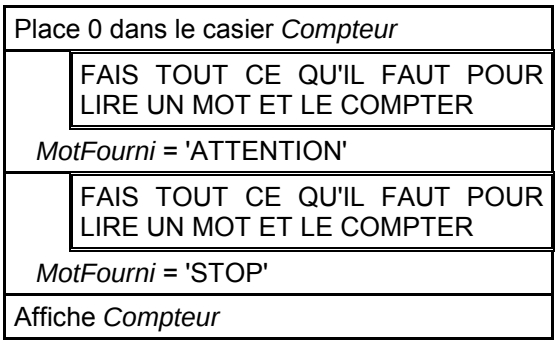
Cette dernière version reste à habiller en étant attentif, par des commentaires, à son lecteur potentiel :

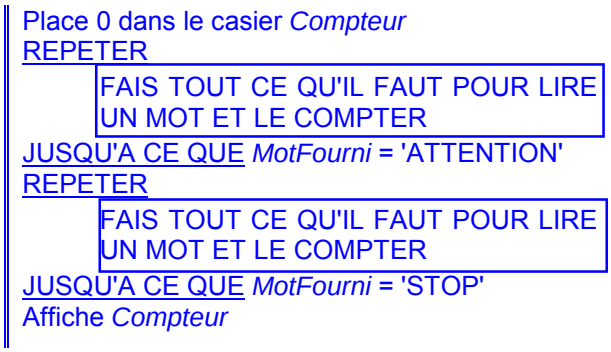
```
program ATTENTION1;
(*    Il fait lire des mots jusqu'à ce que STOP soit fourni à condition que
      ATTENTION ait été fourni auparavant et fait compter les mots lus. On y
      utilise une double répétition.*)
var   MotFourni : string[20] ;
      (* qui contiendra chacun des mots lus *)
      Compteur : integer ;
      (* pour les compter *)
begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l'appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
Compteur := 0;
repeat
      write ('Donnez un mot');
      readln(MotFourni);
      Compteur := Compteur +1;
until MotFourni = 'ATTENTION';
repeat
      write ('Donnez un mot');
      readln(MotFourni);
      Compteur := Compteur +1;
until MotFourni = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

Toujours avec la même structure de solution, j'avais également développé un GNS illustrant la structure d'appel de procédure. La marche à suivre "principale" comportait alors une instruction d'action complexe ("FAIS TOUT CE QU'IL FAUT POUR LIRE UN MOT ET LE COMPTER") qui se trouvait explicitée dans une marche à suivre annexe (procédure).

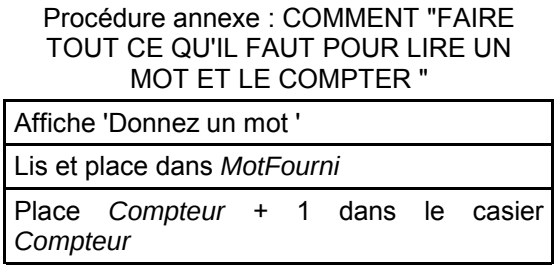
En désignant sous la forme condensée "TRAVAILLE" l'instruction complexe "FAIS TOUT CE QU'IL FAUT POUR LIRE UN MOT ET LE COMPTER", les marches à suivre correspondantes étaient :

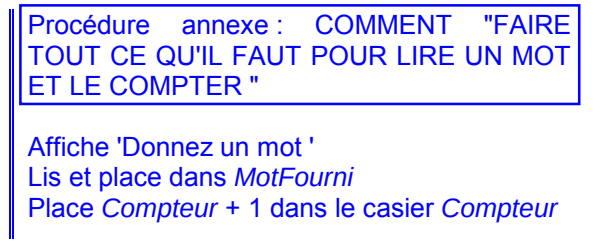
 




avec




### *1.2.1.4 Traduction*

En voici la traduction en Pascal, illustrant la structure d'appel de procédure :

```
program ATTENTION2;
var   MotFourni : string[20] ;
      Compteur : integer ;

      procedure TRAVAILLE;                                    (1)
      begin                                                   (2)
      write('Donnez un mot ');
      readln(MotFourni);                                      (4)
      Compteur := Compteur + 1;                               (4)
      end;                                                    (2)

begin
repeat
      TRAVAILLE;                                              (3)
until MotFourni = 'ATTENTION';
repeat
      TRAVAILLE;                                              (3)
until Motfourni = 'STOP';
writeln(Compteur);
end.
```

**Remarques**

(1) `procedure TRAVAILLE;`

Le texte de la procédure (annexe) est annoncé par le mot réservé procedure. Il se situe entre la partie déclaration et le corps du programme principal. La procédure est désignée par un identificateur (ici TRAVAILLE) qui suit les règles habituelles.

**(2)** On verra que le texte d'une procédure peut comporter, après son en-tête, une partie déclaration; elle comporte en tout cas un corps (reprenant la traduction de la marche à suivre correspondant à l'annexe) situé (comme pour le corps du programme principal) entre les mots begin et end. Cependant, **le texte de la procédure se termine par un point-virgule et non par un point final**.

**(3)** `TRAVAILLE;`

L'appel de la procédure (instruction d'action trop complexe, référant à la marche à suivre annexe) se fait simplement en citant son nom (ici TRAVAILLE). Lors de la rencontre de cette instruction, l'exécutant se détourne un moment du programme principal pour suivre les instructions commandées par la procédure. Après l'exécution des actions commandées par celle-ci, il revient au programme principal, là où il l'avait laissé.

Il est trop tôt pour approfondir les détails entourant les possibilités liées à l'appel de procédure.

Cependant, si l'on veut pouvoir, dans la suite, rendre compte des concepts de variables locales et globales et de paramètres, il est utile de s'imaginer, dès à présent, les choses comme suit.

L'exécutant "principal" est toujours accompagné de son "installateur d'étiquettes". Cet installateur dispose de la partie "déclaration" du programme principal, essentiellement constituée pour l'instant par la liste des variables. Il installe donc sur des casiers du type adéquat les étiquettes réclamées. Ensuite, l'exécutant principal, disposant du "corps" du programme (= la partie exécutable) se met au travail. Dès qu'il rencontre, dans la marche à suivre, une instruction complexe (appel de procédure), il s'interrompt et "passe la main" à un exécutant "auxiliaire" qui possède, lui, le texte explicatif de la procédure. Dans le cas de l'exemple ci-dessus, cet auxiliaire dispose du texte décrivant la procédure TRAVAILLE.[2]

On verra (dans un second volume de cet ouvrage !) que cet exécutant adjoint peut, lui aussi être accompagné de son installateur d'étiquettes (si la procédure comporte une partie déclaration de variables).

Une fois le travail commandé dans le texte de la procédure terminé, le couple se retire (en emportant d'ailleurs ses étiquettes). L'exécutant principal continue alors l'exécution du programme principal, là où il l'avait laissé

(4) Les actions commandées dans le texte de la procédure peuvent faire manipuler des variables définies dans le programme principal. Ainsi, le texte de la procédure TRAVAILLE fait mention des variables *MotFourni* et *Compteur*, définies dans le programme principal. Nous reviendrons dans la suite beaucoup plus longuement sur cette possibilité.

#### *1.2.1.5 Habillage*

Le programme doit, comme sa version précédente, être habillé en tenant compte de son utilisateur :

```
program ATTENTION2;
var   MotFourni : string[20] ;
      Compteur : integer ;

      procedure TRAVAILLE;
      begin
      write('Donnez un mot ');
      readln(MotFourni);
```

[2] Cet éclairage imagé de la structure d'appel de procédure est décrit dans un fascicule "Une approche métaphorique des concepts de procédure, variables globales ou locales et paramètres" que le lecteur intéressé peut se procurer en m'écrivant. On le trouvera détaillé dans le second volume.

```
     Compteur := Compteur + 1;
     end;

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l''appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
repeat
     TRAVAILLE;
until MotFourni = 'ATTENTION';
repeat
     TRAVAILLE;
until Motfourni = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

Et en tenant compte du lecteur de cette version :

```
program ATTENTION2;
(*   Il fait lire des mots jusqu'à ce que STOP soit fourni à condition que
     ATTENTION ait été fourni auparavant et fait compter les mots lus. On y
     utilise un appel de procédure.*)
var  MotFourni : string[20] ;
     (* qui contiendra chacun des mots lus *)
     Compteur : integer ;
     (* pour les compter *)

     procedure TRAVAILLE;
     (* Elle fait lire un mot pour le placer dans MotFourni et fait incrémenter
     le Compteur. *)
     begin
     write('Donnez un mot ');
     readln(MotFourni);
     Compteur := Compteur + 1;
     end;

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l''appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
repeat
     TRAVAILLE;
until MotFourni = 'ATTENTION';
repeat
```

```
      TRAVAILLE;
until Motfourni = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

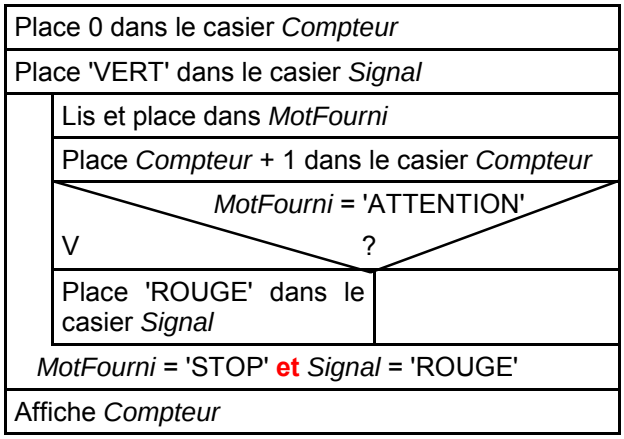
### 1.2.2 Deuxième solution

La seconde solution apportée au problème utilisait essentiellement une variable signal qui passait du "vert" au "rouge" à la réception du mot ATTENTION. Cette approche avait conduit à :

**Liste des variables**

- *MotFourni*, de type chaîne d'au plus 20 caractères;
- *Compteur*, de type entier;
- *Signal*, de type chaîne d'au plus 5 caractères, passant de 'VERT' à 'ROUGE', lorsque ATTENTION est fourni.

**Marche à suivre**




```
Place 0 dans le casier Compteur
Place 'VERT' dans le casier Signal
REPETER
    Lis et place dans MotFourni
    Place Compteur + 1 dans le casier Compteur
    SI MotFourni = 'ATTENTION' ALORS
        Place 'ROUGE' dans le casier Signal
JUSQU'A CE QUE MotFourni = 'STOP' et Signal = 'ROUGE'
Affiche Compteur
```

#### *1.2.2.1 Traduction*

En voici la traduction en Pascal :

```
program ATTENTION3;
var   MotFourni : string[20] ;
      Compteur : integer ;
      Signal : string[5];                                    (1)
begin
Compteur := 0;
Signal := 'VERT';
repeat
      readln(MotFourni);
      Compteur := succ(Compteur);                            (2)
      if Motfourni = 'ATTENTION' then                        (3)
          Signal := 'ROUGE';
until (MotFourni = 'STOP') and (Signal = 'ROUGE');           (4)
writeln(Compteur );
end.
```

**Remarques**

(1) `Signal : string[5];`

La variable *Signal* n'aura à contenir que 'ROUGE' ou 'VERT', chaînes d'au plus 5 caractères; *Signal* sera donc de type string[5].

(2) `Compteur := succ(Compteur);`

Plutôt que de désigner par "*Compteur* + 1" la valeur suivante prise par le casier Compteur, nous avons utilisé un outil différent de l'habituelle addition. C'est l'outil succ (successeur de) qui permet de passer d'un entier à son successeur (ce qui revient au même que de lui ajouter 1). Notons cependant que l'emploi de cet outil n'aurait aucun sens pour des informations de type real ou string. Que serait en effet le successeur de 2.23 ou de 'Bonjour' ?

> Il s'agit donc d'un outil supplémentaire, disponible sur la table de travail de l'exécutant ... à côté de beaucoup d'autres qui n'auront pas été présentés dans ces pages.
>
> On devine aisément que tout à côté de l'outil "succ", on trouvera un outil "pred".

(3)
```
if MotFourni = 'ATTENTION' then
    Signal := 'ROUGE';
```

La structure alternative "SI ... ALORS ..." se traduit en Pascal par "if ... then ...". La syntaxe en est la suivante :

```
if condition then instruction unique
```

ce qui nécessitera des explications (voir plus loin) lorsque le then portera sur plusieurs instructions.

(4) `(MotFourni = 'STOP') and (Signal = 'ROUGE')`

Le connecteur logique "ET" se traduit par "and". Il permet de former une condition rassemblant plusieurs comparaisons. Pour que la condition complète soit vraie, il faut que chacune des comparaisons la composant le soit. De plus, les diverses comparaisons liées par and **doivent être encloses dans des parenthèses**.

Il resterait comme pour les deux versions précédentes à habiller ce programme, d'abord en pensant à son utilisateur, puis au lecteur éventuel. Je vous en laisse le soin !

## *1.3 Le problème du dernier et de l'avant-dernier*

Il s'agissait cette fois de faire lire des mots avec arrêt à "STOP" à condition que "ATTENTION" soit survenu **juste avant**.

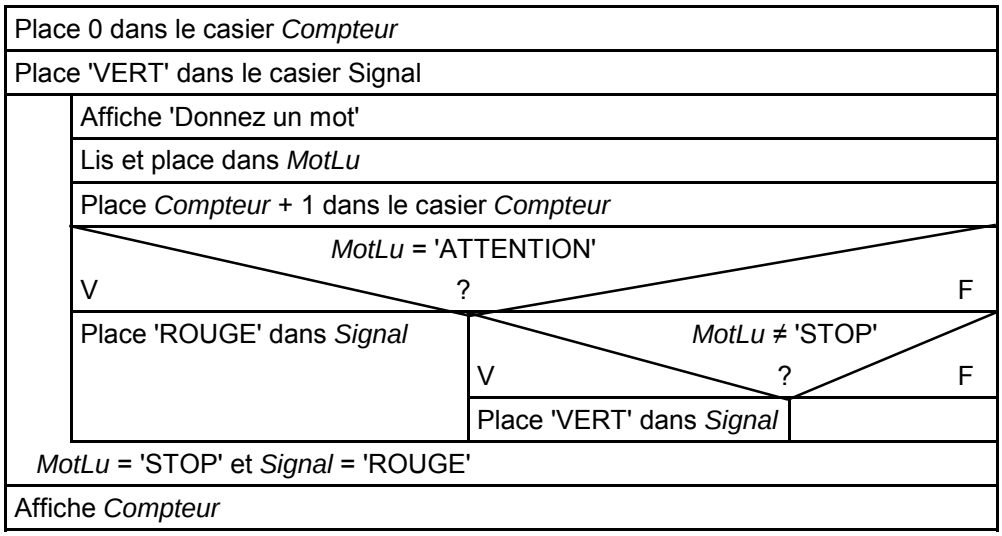
### 1.3.1 Première solution

Elle faisait à nouveau usage d'une variable *Signal* qui passait du 'VERT' au 'ROUGE' lorsque survenait 'ATTENTION', mais qui ne pouvait rester 'ROUGE' que si ensuite c'était 'STOP' qui était reçu.

**Liste des variables**

- *MotLu*, de type chaîne d'au plus 20 caractères;
- *Compteur*, de type entier;
- *Signal*, de type chaîne d'au plus 5 caractères.

**Marche à suivre**




ou sous forme pseudo-code :

```
Place 0 dans le casier Compteur
Place 'VERT' dans le casier Signal
REPETER
        Affiche 'Donnez un mot'
        Lis et place dans MotLu
        Place Compteur + 1 dans le casier Compteur
        SI MotLu = 'ATTENTION' ALORS
                Place 'ROUGE' dans Signal
        SINON
                SI MotLu ≠ 'STOP' ALORS
                        Place 'VERT' dans Signal
JUSQU'A CE QUE  MotLu = 'STOP' et Signal = 'ROUGE'
Affiche Compteur
```

#### *1.3.1.1 Traduction*

Et voici la traduction Pascal de cette marche à suivre :

```
program ATTENTIONSTOP;
var   MotLu : string[20] ;
      Compteur : integer ;
      Signal : string[5];
begin
Compteur := 0;
Signal := 'VERT';
repeat
      write('Donnez un mot');
      readln(MotLu);
      Compteur := Compteur + 1;
      if MotLu = 'ATTENTION' then
          Signal := 'ROUGE'
      else                                                    (1)
```

```
          if MotLu <> 'STOP' then                                    (2)
               Signal := 'VERT';
until (MotLu = 'STOP') and (Signal = 'ROUGE');
writeln(Compteur );
end.
```

**Remarques**

(1) ***else***

La structure alternative complète "SI condition ALORS ... SINON ..." se traduit par

```
if condition then ...  else ...
```

La syntaxe de cette instruction Pascal est la suivante :

```
if condition then
     instruction unique
else
     instruction unique
```

On remarquera l'**absence de point-virgule avant le mot else**.

(2) On peut parfaitement avoir des structures alternatives imbriquées comme ici ou l'instruction unique attendue après le else est à nouveau une alternative if ... then ... On remarquera les diverses indentations qui mettent en évidence ces structures imbriquées.

### *1.3.1.2 Habillage*

Et voici la version complètement habillée de ce programme :

```
program ATTENTIONSTOP;
(*    Il fait lire des mots jusqu'à ce que STOP soit fourni à condition que
      ATTENTION ait été fourni juste avant et compte les mots lus. On y utilise
      une variable signal. *)
var   MotLu : string[20] ;
      (* qui contiendra chacun des mots lus *)
      Compteur : integer ;
      (* pour les compter *)
      Signal : string[5];
      (* qui passera de 'VERT' à 'ROUGE' lorsque 'ATTENTION' sera lu *)
begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l''appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni juste avant ATTENTION.');
Compteur := 0;
Signal := 'VERT';
repeat
      write('Donnez un mot : ');
      readln(MotLu);
      Compteur := Compteur + 1;
      if MotLu = 'ATTENTION' then
           Signal := 'ROUGE'
```

```
     else
          if MotLu <> 'STOP' then
               Signal := 'VERT';
until (MotLu = 'STOP') and (Signal = 'ROUGE');
writeln('Vous m''avez donné en tout ',Compteur, ' mots.');
end.
```

### 1.3.2 Deuxième Solution

Cette seconde approche faisait retenir par l'exécutant le dernier et l'avant-dernier mots lus.

**Liste des variables**

- *Dernier*, de type chaîne d'au plus 20 caractères,
- *AvantDernier*, de type chaîne d'au plus 20 caractères,
- *Compteur*, de type entier.

**Marche à suivre**

| |
|---|
| Place 0 dans le casier *Compteur* |
| Place 'BONJOUR' dans le casier *Dernier* |
| Place *Dernier* dans le casier *AvantDernier* |
| Affiche 'Donnez un mot' |
| Lis et place dans *Dernier* |
| Place *Compteur* + 1 dans le casier *Compteur* |
| *Dernier* = 'STOP' et *AvantDernier* = 'ATTENTION' |
| Affiche *Compteur* |

Place 0 dans le casier *Compteur*
Place 'BONJOUR' dans le casier *Dernier*
REPETER
    Place *Dernier* dans le casier *AvantDernier*
    Affiche 'Donnez un mot'
    Lis et place dans *Dernier*
    Place *Compteur* + 1 dans le casier *Compteur*
JUSQU'A CE QUE *Dernier* = 'STOP' et
    *AvantDernier* = 'ATTENTION'
Affiche *Compteur*

Voici la traduction en Pascal de cette marche à suivre :

```
program ATTENTIONSTOP2;
var  Dernier, AvantDernier : string[20] ;                    (1)
     Compteur : integer;
begin
Compteur := 0;
Dernier := 'BONJOUR';
repeat
     AvantDernier := Dernier;
     write('Donnez un mot : ');
     readln(Dernier);
     Compteur:=Compteur+1;
until (AvantDernier='ATTENTION') and (Dernier='STOP');
writeln(Compteur);
end.
```

**Remarques**

(1) `var Dernier, AvantDernier: string[20];`

Lorsque plusieurs variables du même type doivent être définies, il est loisible d'en dresser la liste en une seule fois à condition de séparer leurs noms par la virgule. Comme toujours, cette liste est suivie de deux points et du type des variables de la liste.

On aurait cependant pu écrire également

```
var Dernier: string[20];
    AvantDernier: string[20];
    Compteur : integer;
```

Il reste une fois de plus à habiller le programme. Cet habillage sera en grande partie identique à ce qu'il est pour le précédent ... et je vous en laisse donc le soin.

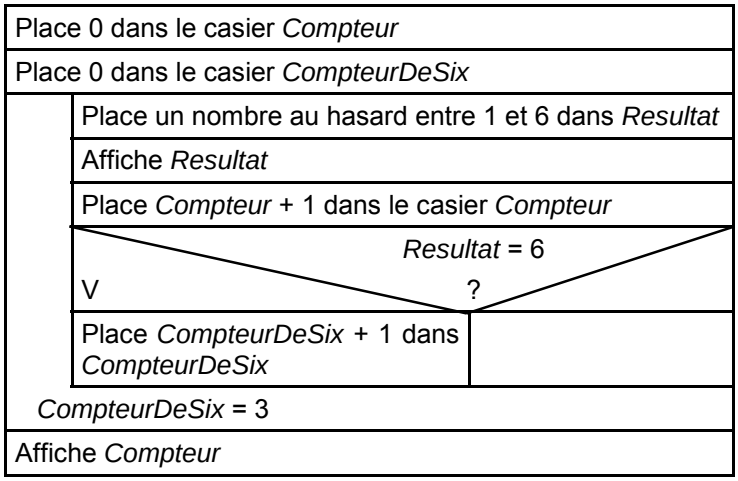
## *1.4 Le problème du lancer de dé*

Nous avions rédigé un GNS qui faisait simuler des jets successifs d'un dé jusqu'à l'obtention de trois six.

**Liste des variables**

- *Resultat*, de type entier, qui contiendra un instant chaque résultat du tirage au hasard,
- *Compteur*, de type entier, pour compter les lancers; il s'augmentera de 1 à chaque lancer;
- *CompteurDeSix*, de type entier, pour compter les six qui se présenteront.

**Marche à suivre**




Place 0 dans le casier *Compteur*
Place 0 dans le casier *CompteurDeSix*
REPETER
    Place un nombre au hasard entre 1 et 6 dans *Resultat*
    Affiche *Resultat*
    Place *Compteur* + 1 dans le casier *Compteur*
    SI *Resultat* = 6 ALORS
        Place *CompteurDeSix* + 1 dans *CompteurDeSix*
JUSQU'A CE QUE *CompteurDeSix* = 3
Affiche *Compteur*

La traduction en est à nouveau immédiate : le seul petit problème est d'indiquer comment, en Pascal (dans l'implémentation Turbo), on peut parler d'un nombre (entier) au hasard entre 1 et 6.

```
program JETDEDES;
var Resultat, Compteur, CompteurDeSix : integer;                    (1)
begin
Compteur := 0;
CompteurDeSix:=0;
repeat
     Resultat := random(6)+1;                                       (2)
     write(Resultat);
     Compteur:=succ(Compteur);
```

```
     if Resultat=6 then
          CompteurDeSix:=succ(CompteurDeSix);
until CompteurDeSix=3;
writeln(Compteur);
end.
```

(1) `var Resultat, Compteur, CompteurDeSix : integer;`

Nous avons à nouveau dressé une seule liste de toutes les variables entières.

(2) `Resultat := random(6)+1;`

L'outil qui permet à l'exécutant de disposer d'un nombre entier tiré au hasard s'appelle random. L'exécutant y place un nombre (ici 6) à l'entrée et il y reprend un nombre entier compris entre 0 (inclus) et le nombre entré (exclu) (ici donc entre 0 et 5). Il ne reste plus alors qu'à ajouter 1 à l'information ainsi obtenue pour disposer à coup sûr d'un nombre entre 1 et 6.

L'habillage de ce programme va (pour une fois) nous amener à découvrir quelques possibilités supplémentaires du langage Pascal (dans son implémentation Turbo).

```
program JETDEDES;
(*    Il fait simuler des jets successifs d'un dé jusqu'à obtention d'un
      troisième 6. *)
var   Resultat,
      (* qui contiendra les résultats du "lancer" *)
      Compteur,
      (* pour compter le nombre de lancers *)
      CompteurDeSix
      (* pour compter les 6 obtenus *)
      : integer;
begin
(* effacement de l'écran *)
clrscr;                                                          (1)
writeln('Je vais lancer pour vous un dé en arrêtant');
writeln('au troisième 6 obtenu.');
(* demande d'une pause *)
delay(5000);                                                     (2)
clrscr;                                                          (1)
(* placement du curseur à la 2ème ligne, 36ème colonne *)
gotoxy(36,2);                                                    (3)
write('Résultats');                                              (4)
(* placement du curseur à la 3ème ligne, 36ème colonne *)
gotoxy(36,3);                                                    (5)
writeln('==========');
(* passage d'une ligne blanche *)
writeln;                                                         (6)
Compteur := 0;
CompteurDeSix:=0;
repeat
      Resultat := random(6)+1;
```

```
      write(Resultat);
      Compteur:=succ(Compteur);
      if Resultat=6 then
            CompteurDeSix:=succ(CompteurDeSix);
until CompteurDeSix=3;
writeln;
writeln('Nombre total de jets : ',Compteur);
end.
```

(1) **clrscr**

Cette instruction exige que l'exécutant efface complètement l'écran. Il s'agit d'un raccourci pour les mots "clear screen" qui signifient de fait "efface l'écran". C'est une sage précaution, en début de programme, de faire procéder à cet effacement.

On pourrait, sans doute, voir dans l'instruction clrscr une instruction d'action élémentaire ne correspondant à aucune des trois actions élémentaires dont est capable l'exécutant. On pourrait aussi y voir un raccourci pour "Affiche ... un écran vide" et l'assimiler à un cas (très) particulier de l'instruction d'affichage.

> clrscr, comme les instructions suivantes, gotoxy, delay, ... constituent autant de détails supplémentaires concernant les possibilités de l'exécutant. Il ne s'agit pas là de possibilités du Pascal "standard", mais bien d'ajouts présents dans l'implémentation "Turbo Pascal, Version 3". Il n'est pas certain du tout que ces possibilités se retrouvent dans d'autres versions de Pascal. Il ne faut donc pas faire de la maîtrise de tels détails l'objectif d'un apprentissage de la programmation.
>
> Ainsi, dans l'implémentation 1.5 de Turbo Pascal sous Windows, on peut utiliser clrscr ou gotoxy, mais à condition de faire suivre l'entête du programme de la mention uses wincrt;. Dans cette implémentation, cette mention est pratiquement obligatoire.

(2) **delay(5000);**[3]

L'instruction delay( ...) fait temporiser l'exécutant pendant un moment proportionnel à la taille du nombre entier inscrit entre les parenthèses. (Ce temps dépend de l'ordinateur utilisé, mais il doit en général être de quelques milliers pour que la temporisation soit "visible"). Pendant cet "arrêt", l'exécutant ne poursuit pas l'accomplissement de la marche à suivre et tout reste donc inchangé.

Pourquoi donc "stopper" ainsi l'exécutant pendant un temps plus ou moins long ? Il faut remarquer qu'ici, sans cette instruction, le morceau de programme concerné serait

```
clrscr;
writeln('Je vais lancer pour vous un dé en arrêtant');
writeln('au troisième 6 obtenu.');
clrscr;
```

Dès lors, après avoir effacé l'écran (clrscr), l'exécutant y afficherait les deux lignes de message exigées, puis immédiatement après effacerait de nouveau, faisant aussitôt disparaître les deux lignes qu'il venait d'afficher. Et comme il est (heureusement en général, malheureusement ici) fort rapide, l'utilisateur n'aurait absolument pas eu le temps de lire mon message ! Je commande donc (grâce à delay) à l'exécutant de rester un moment "à ne rien faire", avant d'effacer, moment pendant lequel l'utilisateur aura le loisir de lire les deux lignes de présentation de l'objectif du programme ... puis les choses reprendront leur cours.

---

[3] Cette instruction n'existe plus dans les implémentations plus récentes de Pascal (sous Windows).

Il s'agit réellement cette fois d'une instruction d'action élémentaire; nouvelle. Elle ne figurait pas au répertoire de l'exécutant tel qu'il avait été décrit dans le chapitre trois. J'ai donc menti (par omission !). Deux remarques à ce propos : d'abord cette action nouvelle ne change pas énormément le portrait de l'exécutant gestionnaire de casiers, avec sa porte, sa fenêtre et les trois actions dont il était capable; ensuite, cette omission (volontaire) est la première d'une longue série que nous digérerons petit à petit.

Il s'agit là de détails : ce n'est pas la connaissance de ceux-ci qui fait la différence entre les "bons programmeurs" et les "autres". Mais autant savoir ...

**(3)** (et **(5)**) `gotoxy(36,2);`

Et voici à nouveau une instruction de présentation des informations affichées à l'écran. Il s'agit ici de positionner le curseur à un endroit déterminé de ce dernier. Le curseur, c'est ce petit rectangle (ou carré, ou tiret ...) que vous voyez à tout moment à l'écran. Le texte qui sera affiché à l'écran, d'où qu'il vienne (affiché par l'exécutant ou fourni par l'utilisateur à travers son clavier), vient toujours prendre place à partir de la position du curseur. Le fait d'envoyer le curseur à un endroit précis, préalablement à un affichage de texte, permet donc une présentation plus attrayante des messages ou des résultats.

L'écran est (en général) constitué de 24 lignes de 80 caractères. On peut à l'aide de gotoxy(...,...) envoyer le curseur à n'importe lequel de ces emplacements de l'écran. Les arguments de cette instruction (ce sont les deux nombres écrits dans les parenthèses) doivent être des informations entières (écrites comme des constantes, des variables ou des expressions). Le premier argument indique la colonne où se déplacera le curseur (elle détermine donc son déplacement horizontal); le second précise la ligne où il sera envoyé (déplacement vertical).

Ainsi,

```
gotoxy(36,2)
```

enverra le curseur à la 36ème colonne, 2ème ligne (en haut et à peu près au milieu de l'écran).

**(4)** `write('Résultats');`

Le titre dont write commande l'affichage viendra donc s'inscrire (à cause de la position prise par le curseur après l'instruction gotoxy qui précédait) sur la 2ème ligne (en haut de l'écran), à partir de la 36ème position. Ce titre sera ainsi (à peu près) centré.

**(6)** `writeln;`

L'instruction writeln employée seule (sans argument) commande un saut à la ligne à l'écran.

## *1.5 Le problème du calcul de moyenne*

Il s'agissait cette fois de faire lire une série de 100 nombres réels et de fournir ensuite la plus grande donnée, la plus petite et leur moyenne. Notre analyse avait finalement conduit au GNS suivant :

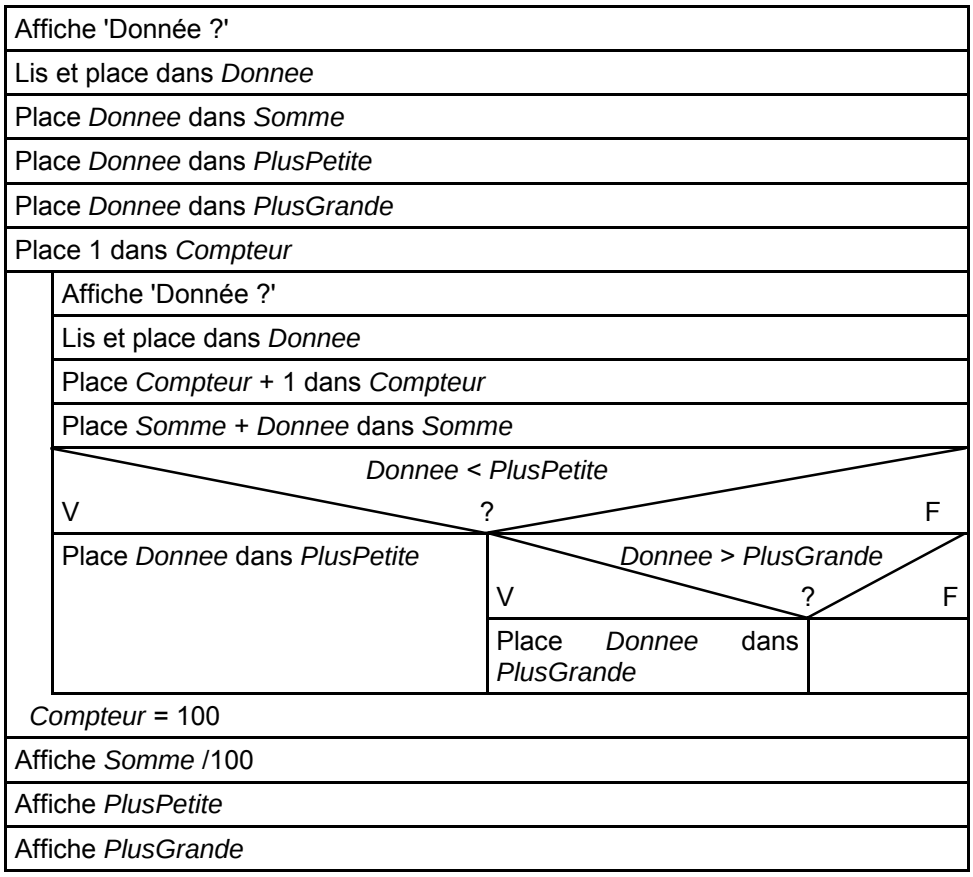
**Liste des variables**

- *Donnee*, de type réel, qui contiendra chaque donnée lue,
- *Somme*, de type réel, pour accueillir leur somme,
- *PlusPetite*, de type réel, qui contiendra la plus petite,

- *PlusGrande*, de type réel, pour la plus grande,
- *Compteur*, de type entier, pour les compter.

**Marche à suivre**




et sous forme pseudo-code :

```
Affiche 'Donnée ?'
Lis et place dans Donnee
Place Donnee dans Somme
Place Donnee dans PlusPetite
Place Donnee dans PlusGrande
Place 1 dans Compteur
REPETER
        Affiche 'Donnée ?'
        Lis et place dans Donnee
        Place Compteur + 1 dans Compteur
        Place Somme + Donnee dans Somme
        SI Donnee < PlusPetite ALORS
        Place Donnee dans PlusPetite
        SINON
                SI Donnee > PlusGrande ALORS
                        Place Donnee dans PlusGrande
JUSQU'A CE QUE Compteur = 100
Affiche Somme /100
Affiche PlusPetite
Affiche PlusGrande
```

Voici le programme Pascal correspondant :

```
program MOYENNE;
var  Donnee, Somme, PlusPetite, PlusGrande : real;                    (1)
     Compteur : integer;
begin
write('Donnée : ');
readln(Donnee);
Somme:=Donnee;
PlusPetite:=Donnee;
PlusGrande:=Donnee;
Compteur:=1;
repeat
     write('Donnée : ');
     readln(Donnee);
     Compteur:=succ(Compteur);
     Somme:=Somme+Donnee;
     if Donnee < PlusPetite then
          PlusPetite:=Donnee
     else
          if Donnee > PlusGrande then
               PlusGrande:=Donnee;
until Compteur=100;
writeln(Somme/100);                                                   (2)
writeln(PlusPetite);
writeln(PlusGrande);
end.
```

**(1)** `var Donnee, Somme, PlusPetite, PlusGrande : real;`

Nous le savions déjà, le type réel se traduit real. Une variable de ce type pourra donc contenir un nombre réel. Le langage Pascal permet cependant, outre l'affectation d'une information réelle à une variable réelle, l'affectation à celle-ci d'une information entière. Ainsi, on pourrait écrire

```
Donnee := 13.567
```

mais aussi

```
Donnee := 13
```

ou

```
Donnee :=Compteur
```

*Donnee* étant une variable réelle et *Compteur* une variable entière.

L'inverse (affectation d'une information de type réel à une variable entière) n'est évidemment pas permis.

**(2)** `Somme/100`

Nous trouvons ici, pour la première fois, l'opération de division. L'outil correspondant figure bien entendu sur la table de travail de l'exécutant. Les deux informations à fournir doivent être de type entier ou réel; le résultat lui est toujours de type réel.

Ainsi

4/2 est de type réel

comme

*Somme*/100

même si, dans le cas de 4/2, la division "tombe juste".

C'est une règle (assez) générale qu'on puisse ainsi lorsqu'on commande une opération ( +, -, *, / ) mélanger des arguments entiers et réels. Si les deux nombres sont de type entier, le résultat l'est lui aussi (sauf pour /), sinon le résultat est de type réel.

Comme d'habitude, il reste à habiller le programme ainsi obtenu :

```
program MOYENNE;
(*    Il fait lire une série de 100 données réelles et fait afficher la plus
      petite, la plus grande et leur moyenne. *)
var   Donnee,
      (* qui contiendra successivement les diverses données fournies *)
      Somme,
      (* qui contiendra la somme des données *)
      PlusPetite,
      (* qui finira par contenir la plus petite donnée *)
      PlusGrande
      (* qui contiendra la plus grande donnée *)
      : real;
      Compteur
      (* pour compter les 100 données lues *)
      : integer;
begin
writeln('Vous allez me fournir 100 données réelles et');
writeln('je vous donnerai ensuite la plus petite, la plus grande');
writeln('et leur moyenne.');
writeln;
write('Votre première donnée : ');
readln(Donnee);
Somme:=Donnee;
PlusPetite:=Donnee;
PlusGrande:=Donnee;
Compteur:=1;
repeat
      write('Donnée numéro ',Compteur+1,' : ');                          (1)
      readln(Donnee);
      Compteur:=succ(Compteur);
      Somme:=Somme+Donnee;
      if Donnee<PlusPetite then
```

```
            PlusPetite:=Donnee
      else
            if Donnee>PlusGrande then
                  PlusGrande:=Donnee;
until Compteur=100;
clrscr; (* effacement de l'écran *)
writeln('La moyenne des données est : ',Somme/100);
writeln('La plus petite est : ',PlusPetite);
writeln('et la plus grande : ',PlusGrande);
end.
```

(1) `write('Donnée numéro ',Compteur+1,' : ');`

Au moment d'inviter l'utilisateur à fournir une donnée, je lui rappelle (ou, plutôt fais rappeler) la position de celle-ci. L'affichage résultant sera du type

```
Votre première donnée : 23.45
Donnée numéro 2 : -34.67
Donnée numéro 3 : 23
...          ...
```

les quantités soulignées étant les nombres fournis par l'utilisateur.

Il est important de bien voir pourquoi je commande

```
write('Donnée numéro ',Compteur+1,' : ')
```

et pas

```
write('Donnée numéro ',Compteur,' : ')
```

## 2. Exercices

1. Traduisez en Pascal et "habillez" les programmes obtenus pour chacune des marches à suivre développées dans les exercices proposés au chapitre précédent.
2. A et B désignant deux quantités entières (avec A<B), par quelle expression désigneriez-vous, en Pascal, un nombre entier aléatoire compris entre A et B (A et B inclus) ?
3. Pourquoi ne redouble-t-on pas les apostrophes qui figurent dans les commentaires ?

## 3. Synthèse sur le langage

### *3.1 Structure générale d'un programme Pascal*

#### 3.1.1 En-tête du programme

La première ligne comporte nécessairement l'indication

```
program NomChoisi ;
```

##### *3.1.1.1 Les mots réservés*

Le mot program est ce qu'on appelle un mot réservé du langage Pascal. Nous en connaissons à présent beaucoup d'autres : var, repeat, readln, until, then, ... J'ai pris l'habitude de les souligner

dans les exemples précédents. Ces mots réservés ne peuvent évidemment être utilisés librement pour désigner le programme, les variables, procédures ...

Ils doivent être écrits tels quels, sans y faire figurer d'espace, de tirets, de passage à la ligne, ... On peut indifféremment les écrire en majuscules ou minuscules. Ils doivent être suivis d'au moins un caractère permettant de voir qu'ils sont bien terminés. Ces caractères "terminateurs" peuvent être l'espace, la virgule, le point-virgule, le passage à la ligne, la parenthèse, les symboles, +, /, -, ... ou tout autre symbole de "ponctuation" admis par le langage et **syntaxiquement permis après le mot réservé qu'ils clôturent** et permettent de reconnaître.

Ainsi,

```
programATTENTION
varCompteur
```

sont incorrects puisqu'ils ne permettent pas d'identifier les mots réservés.

De même

```
program, ATTENTION
var:Compteur
```

ne conviennent pas puisque la virgule et les doubles points ne peuvent figurer à ces endroits du programme.

Lorsque nous en serons à la toute dernière étape de la démarche, face à l'ordinateur et en train de lui fournir les textes des programmes que nous aurons conçus, ces textes devront être dépouillés de tous les ornements (soulignés, caractères gras, italiques, ...) qui figurent dans les textes repris ici. Ainsi, les mots réservés ne devront pas être soulignés. (Cela ne sera d'ailleurs généralement pas possible).[4]

#### *3.1.1.2 Les identificateurs*

NomChoisi est le nom que l'on désire donner au programme. C'est ce que j'ai appelé un **identificateur** : c'est un identificateur qui désignera le programme, mais aussi les variables (casiers) et les procédures utilisées.

Un identificateur est une succession de lettres et de chiffres à l'exclusion de tout autre symbole sauf le caractère de soulignement (_ ), commençant obligatoirement par une lettre. Les lettres peuvent être écrites en majuscules ou en minuscules mais Pascal ne fait pas de différence : pour lui A et a, c'est la même lettre (sauf lorsqu'on lui parlera de constantes chaîne de caractères, ce qui n'a rien à voir avec les identificateurs). Notez cependant que les lettres accentuées (é, è, ê, à, ...) ne sont pas acceptées.

Il va sans dire que l'identificateur du programme ne peut être réemployé tel quel pour identifier dans la suite des variables, des procédures, etc. De plus, je l'ai signalé ci-dessus, les mots réservés du langage (program, var, ...) ne peuvent servir comme identificateurs. Ces mots réservés peuvent cependant être inclus dans des identificateurs valables : par exemple, programme, variable, différence, ... sont permis.

Ainsi

```
factorielle
```

[4] Dans les implémentations récentes de Turbo Pascal, l'éditeur de texte permettant d'écrire les programmes Pascal est généralement capable de mettre en évidence (par la couleur, l'italique, le gras,...) les mots réservés, les commentaires, etc..

```
EQUATIONDUSECONDDEGRE
EQUATION_DU_PREMIER_DEGRE
EquationDuPremierDegre
A123
```

sont des identificateurs valables.

Par contre, ne conviennent pas :

| | |
|---|---|
| `1A3` | (Commence par un chiffre) |
| `Program` | (Mot réservé) |
| `T**` | (Symboles * inacceptables) |
| `Début` | (Comprend une lettre accentuée) |
| `Un Grand` | (Comporte un espace blanc) |
| `Age`<br>`duCapitaine` | (Comporte un passage à la ligne) |

Quelques remarques pour terminer :

- C'est une excellente habitude de se servir d'identificateurs qui "disent quelque chose". Cela permet souvent une relecture plus aisée des programmes.
- Par ailleurs, puisque le compilateur ne fait aucune différence entre lettres majuscules et minuscules, nous emploierons les unes et les autres, au mieux, de façon à rendre les programmes le plus lisible possible.
- Enfin, comme pour les mots réservés, un identificateur doit être suivi d'au moins un symbole permettant de voir qu'il est terminé : espace, passage à la ligne, point-virgule, virgule, :=, +, (, ....

### 3.1.2 Partie déclarative

Après l'en-tête vient la partie déclarative qui reprend, entre autres, la liste des variables qui vont être employées dans le programme. Nous verrons par la suite (dans le Tome II) que c'est ici aussi que seront déclarés les constantes, types, etc.

Cette déclaration prend, par exemple, la forme:

```
var Nombre,X1,X2 : integer;
    Toto, Tutu, Turlututu : real;
    DisMoiOuiOuNon, Signe : string[80];
```

le mot var, annonçant la liste des variables de divers types n'étant écrit qu'une fois.

J'ai retenu, pour l'instant, trois types de variables :

- entières ( integer ),
- réelles ( real ),
- chaînes de caractères ( string[ ] ),

correspondant aux trois types d'informations que nous avons vu manipuler jusqu'à présent par l'exécutant-ordinateur.

Je rappelle qu'il est indispensable, en ce qui concerne les variables string, d'indiquer, entre crochets, la taille maximale des chaînes de caractères qui pourront y résider. Cette taille maximale doit être comprise entre 1 et 255.[5]

Nous découvrirons dans la suite d'autres types d'informations manipulées par l'exécutant-ordinateur, donc d'autres types de variables susceptibles de les accueillir. Les trois types répertoriés jusqu'ici sont amplement suffisants pour débuter ...

### 3.1.3 Le corps du programme

Enfin, le corps du programme comporte les instructions qui seront exécutées. Cette partie prend la forme :

```
begin
Une série d'instructions séparées par des ;
end.
```

Le dernier symbole du programme étant toujours un point.

#### *3.1.3.1 Présentation*

Les blancs, les indentations (quand une partie du texte est décalée), les passages à la ligne n'ont (en général) aucune importance au point de vue syntaxique (compte tenu du fait qu'ils ne peuvent figurer dans les mots réservés et les identificateurs). Vous les utiliserez cependant pour faire clairement apparaître la structure du programme.

Ainsi, le programme

```
program ARRET_A_0;
var  NombreFourni : real ;
     Compteur : integer ;
Compteur := 0;
repeat
     readln(NombreFourni);
     Compteur := Compteur +1;
until NombreFourni = 0;
writeln (Compteur);
end .
```

pourrait aussi s'écrire

```
program ARRET_A_0;var NombreFourni:real ;Compteur : integer ;begin
Compteur := 0;repeat readln(NombreFourni);Compteur := Compteur +1;until
NombreFourni = 0;writeln (Compteur);end .
```

ou encore

```
program
ARRET_A_0
var
NombreFourni
:
```

[5] Dans les implémentations plus récentes, une variable peut être simplement de type string (sans crochet); elle permet alors d'accueillir des chaînes de taille maximale, soit 255 caractères.

```
 real
;
Compteur
 :
integer ;
Compteur := 0;
etc...
```

Inutile de souligner que la première version est plus lisible que les deux suivantes.

#### *3.1.3.2 Les instructions en Pascal*

Ainsi que nous le découvrirons ci-dessous, il ne faut pas confondre ce que j'appellerai "instruction" en Pascal et le concept d'instruction d'action élémentaire que nous connaissons.

Bien entendu, les trois instructions d'action élémentaire donneront naissance aux trois instructions Pascal d'affectation (:=), de lecture (readln) et d'affichage (write ou writeln). Mais la traduction en Pascal des structures de contrôle donnera lieu à ce que nous appellerons **aussi** instruction en Pascal (repeat ... until ..., while ... do ..., if ... then ... else ..., etc.).

Nous parlerons donc de l'instruction repeat ... until ..., de l'instruction if ... then ..., etc.

#### *3.1.3.3 Les "terminateurs" d'instruction Pascal*

C'est, on l'a vu, le symbole point-virgule qui permet d'indiquer qu'une instruction est terminée (et non le passage à la ligne).

Il faut noter cependant que begin n'est pas une instruction, repeat non plus. C'est en réalité begin ... end qui constitue une instruction, comme aussi repeat ... until ... dans son ensemble.

On peut donner l'impression à l'exécutant que des instructions successives constituent en fait une instruction unique. Il suffit de faire précéder la première instruction de la série par begin et de faire suivre la dernière de end. Cette globalisation est d'ailleurs parfois obligatoire (comme après le then ou le else); j'y reviendrai.

## *3.2 Quelques instructions Pascal fondamentales*

Je passerai d'abord en revue les instructions Pascal correspondant aux trois instructions d'action élémentaire "comprises" par l'exécutant.

### 3.2.1 L'instruction d'affectation :=

Elle correspond à l'instruction, notée dans les GNS, par

Place telle information dans *tel casier*

La traduction va malheureusement inverser l'ordre dans lequel se présentaient l'information et le casier à remplir puisque l'instruction d'affectation; va se traduire :

```
nom du casier := information à y placer
```

Elle permet de placer dans la variable (casier) située à gauche du signe := l'information décrite à droite. Rappelons que cette information s'écrira comme une constante, une variable (dont on désigne alors le contenu), ou encore une expression (résultat d'une manipulation mettant en jeu l'un ou l'autre outil de traitement des informations).

Voici quelques exemples :

```
Debut := 15
Rho := Epsilon
Indice := Indice + 1
VALEUR := ACC * (TYU-TRE)/BIBI
```

Il faut bien sûr veiller, lors de l'emploi de l'affectation, à la compatibilité du type de la variable à remplir (tel qu'il a été précisé dans la partie déclaration du programme) et de l'information qu'on demande d'y placer. Par exemple, il est impossible d'affecter à une variable entière une information réelle ou chaîne de caractères (qu'elle prenne la forme d'une constante, d'une variable ou d'une expression). La seule exception permise est l'affectation d'une information entière à une variable réelle.

Il faut être attentif à ce que :=, bien que constitué de 2 caractères, est considéré comme un symbole insécable. Les : ne peuvent donc être séparés de = par un espace.

### 3.2.2 Les instructions de lecture READ et READLN

Elles permettent de faire lire, à partir du clavier, les informations (données) fournies par l'utilisateur. Ces données sont placées dans les casiers indiqués. Read et readln traduisent donc l'instruction notée dans les GNS par

Lis et place dans *tel casier*

la traduction en Pascal devenant

```
read(nom du casier à remplir)
```

ou encore

```
readln(nom du casier à remplir)
```

la différence entre ces deux formes étant (pour l'instant et dans l'implémentation Turbo) sans importance.

Il faut cependant souligner une possibilité (qui sera rarement employée) et que je n'avais pas mentionnée lors du portrait de l'exécutant : il est permis de faire lire par une seule instruction de lecture plusieurs informations qui devront bien entendu prendre place dans des variables différentes. Il suffit de faire suivre l'instruction read ou readln de la liste des variables à remplir par les diverses informations qui seront fournies par l'utilisateur. Les diverses variables citées seront séparées par des virgules.

Par exemple

```
read(Nom, Age, Profession)
readln(X, Somme)
```

Il est, en général, préférable de réserver ces lectures multiples aux cas d'informations numériques.

L'instruction de lecture provoque un arrêt du déroulement des actions de l'exécutant qui attend que la (les) donnée(s) adéquate(s) soi(en)t tapée(s) au clavier. L'utilisateur signalera la fin de l'information fournie par l'appui sur la touche d'entrée aussi appelée <RETURN> ou <ENTER>.

### 3.2.3 Les instructions d'affichage WRITE et WRITELN

Ce sont les instructions Pascal correspondant à l'instruction d'action élémentaire

Affiche telle(s) information(s)

Elles font apparaître à l'écran les informations indiquées. Elles se traduisent par

```
write(informations à afficher)
```

ou encore

```
writeln(informations à afficher)
```

Par exemple :

```
write('Resultat : ',Nombre)
writeln(X,Y,Somme)
writeln(2*X,' : ',15)
```

On le voit, les informations à afficher prennent (comme toujours) la forme soit de constantes ('Resultat : ',15), soit de variables (*Nombre*, *X*, *Y*, *Somme*), soit encore d'expressions (2**X*).

Les diverses informations dont on demande l'affichage doivent être séparées par des virgules. Elles seront affichées les unes à la suite des autres, collées l'une à l'autre.

L'instruction writeln provoque, **après** l'impression des informations concernées un saut à la ligne. Les informations qui apparaîtront donc ensuite à l'écran, qu'elles y soient à cause d'un nouvel affichage ou à cause d'une lecture (les informations frappées par l'utilisateur apparaissant (évidemment !) à l'écran au fur et à mesure qu'il les frappe), seront situées à la ligne suivante.

L'instruction writeln peut aussi être utilisée seule (sans les parenthèses) pour effectuer un passage à la ligne.

Après write, il n'y a pas de saut à la ligne.

Il est donc possible par write et writeln de faire apparaître à l'écran un "message". Il s'agit simplement de commander l'affichage d'une constante de type chaîne de caractères : il faut, rappelons-le, que celle-ci apparaisse entre deux symboles apostrophe. Par exemple, si la variable *X* contient la valeur 17, l'instruction

```
writeln('La valeur de X est ',X,' unités')
```

affichera à l'écran

```
La valeur de X est 17 unités
```

suivi d'un saut à la ligne.

Il importe d'être attentif, lors de l'affichage de plusieurs informations au fait qu'elles apparaîtront collées l'une à l'autre. Ainsi, en reprenant l'exemple ci-dessus, si l'on avait exigé

```
writeln('La valeur de X est',X,'unités')
```

l'affichage aurait été

```
La valeur de X est17unités
```

Il me faut à présent passer en revue les instructions Pascal correspondant aux structures de contrôle.

### 3.2.4 L'instruction alternative; IF...THEN ... ELSE ...

C'est celle qui traduira les mots

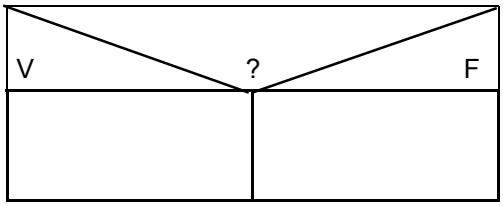
 

SI ... ALORS ... SINON ...

ou encore (en l'absence de SINON)

SI ... ALORS ...

représenté dans les GNS par le graphisme




Elle prend généralement la forme :

```
if condition then
   instruction unique
else
   instruction unique
```

et, en l'absence de SINON,

```
if condition then
   instruction unique
```

Nous trouvons ici l'un des cas où la syntaxe de Pascal commande impérieusement de pouvoir amalgamer plusieurs instructions en une instruction unique à l'aide de begin et end.

Par exemple :

```
if A = B-C then
   begin
   C := B+C;
   A := 2*C;
   end
else
   A := 3*A;
```

Tous ces exemples, destinés seulement à illustrer les aspects grammaticaux de Pascal, constituent autant d'exemples de **ce qu'il ne faut pas faire en ce qui concerne le choix des noms de variables**.

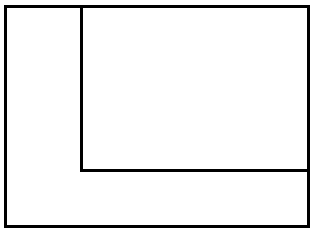
Ajoutons encore que la partie else ... peut être absente. L'ensemble if ... then ... else ... forme une instruction unique pour Pascal; on ne pourra donc jamais trouver de ";" juste avant le mot else (c'est pourtant une erreur très classique).

### 3.2.5 L'instruction de répétition REPEAT... UNTIL ...

C'est elle qui traduira la structure

REPETER ... JUSQU'A CE QUE ...

représentée par le graphisme



Elle prend la forme :

```
repeat
   série d'instructions
until condition
```

Les instructions situées entre le repeat et le until seront répétées jusqu'à ce que la condition énoncée devienne vraie. Dans tous les cas, on le sait, cette série d'instructions sera exécutée au moins une fois.

Par exemple :

```
repeat
   I:=I+1;
   Somme := Somme + Ajout;
   Reste := Reste - 1;
until I>Max
```

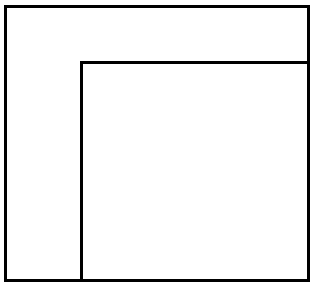
Vous prendrez l'habitude d'indenter le texte de vos programmes comme dans l'exemple ci-dessus. On peut ainsi voir d'un coup d'œil où commence et où se termine la série des instructions qui seront répétées.

### 3.2.6 L'instruction de répétition WHILE ... DO...

Nous n'avons pas encore eu l'occasion de l'utiliser. Elle traduit la structure

TANT QUE ... FAIRE ...

représentée par le graphisme



Elle prend la forme :

```
while condition do
   instruction unique
```

Voilà encore un cas où l'on est forcé de rassembler plusieurs instructions, que l'on veut voir exécuter à la suite de do, en une seule à l'aide de begin et end.

Par exemple :

```
while (Total <> TT) or (S=U) do
   begin
   TT := TT+1;
   Produit := Produit * (Rest + 4.1);
```

```
    end;
```

Contrairement à l'instruction repeat, l'instruction while peut voir les instructions qu'elle contrôle ne pas être exécutées une seule fois si la condition sur laquelle porte le while a la valeur FAUX dès le début.

### 3.2.7 L'appel de procédure

Pascal permet de faire figurer dans le texte du programme des actions complexes qui sont explicitées dans des procédures annexes. Il s'agit là de la possibilité d'appel de procédure.

Il faut bien distinguer à ce propos deux choses :

- l'action complexe commandée dans le texte du programme (appel de la procédure);
- l'explicitation de cette action (texte de la procédure).

Je n'essayerai pas pour l'instant d'être complet à ce propos. Beaucoup de concepts importants viendront dans la suite enrichir cette structure (variables globales ou locales, paramètres, arborescence des divers niveaux de procédures, ...).

#### *3.2.7.1 Appel d'une procédure*

L'appel d'une procédure consiste à simplement citer l'identificateur par laquelle on la désigne. Cet identificateur suit les règles syntaxiques habituelles.

#### *3.2.7.2 Texte d'une procédure*

Ce texte doit figurer au sein du programme lui-même, entre la partie déclarative et le corps du programme. L'identificateur de la procédure (ce qui dans le texte du programme sert à l'appeler) doit être précédé du mot réservé procedure et suivi d'un point-virgule : le tout constitue l'en-tête de la procédure. Cet en-tête peut être suivi d'une partie déclarative. Vient enfin le corps de la procédure, enclos entre les mots begin et end, **suivi d'un point-virgule**.

Le texte d'une procédure est donc rigoureusement identique à celui d'un programme sauf au niveau de l'en-tête où le mot procedure remplace le mot program et tout à la fin où le point-virgule remplace le point final du programme.

#### *3.2.7.3 Variables globales et variables locales*

Un mot encore, **très incomplet**, sur le fait qu'on peut donc définir des variables à l'intérieur du texte d'une procédure : ces variables, locales à cette procédure, peuvent évidemment être utilisées dans le texte de celle-ci, mais ne peuvent l'être dans le corps du programme principal ou d'autres procédures.

Les variables définies au niveau du programme principal, elles, peuvent être utilisées partout, tant dans le corps du programme que dans celui des procédures qui y sont nichées.

J'ai mis en évidence à côté des instructions Pascal correspondant aux trois instructions d'action élémentaire et aux structures de contrôle quelques instructions auxiliaires.

### 3.2.8 Instructions diverses

1) L'instruction delay( ) permet d'interrompre pendant un moment l'exécution de la suite du programme. L'information à faire figurer entre les parenthèses doit être de type entier.
2) L'instruction gotoxy( , ) permet de placer le curseur en un endroit précis de l'écran. Les deux quantités indiquées entre parenthèses doivent être de type entier : elles précisent respectivement la position horizontale (entre 1 et 80) et la position verticale (entre 1 et 25).

Ainsi gotoxy(1,1) envoie le curseur dans le coin supérieur gauche et gotoxy(80,25) dans le coin inférieur droit de l'écran.

3) L'instruction clrscr fait effacer l'écran et amène le curseur dans le coin supérieur gauche (en position (1, 1)).

J'ai signalé qu'à côté des instructions d'action élémentaire et des structures de contrôle, les marches à suivre comporteraient également des commentaires. Il reste à dire comment ceux-ci peuvent s'insérer dans le texte d'un programme Pascal.

### 3.2.9 Les commentaires;

Les programmes, surtout s'ils sont longs, ne sont pas toujours aisés à relire. Le langage Pascal nous autorise (et le bon sens nous le conseille vivement) à insérer dans le texte des commentaires qui ne seront pas pris en compte par le compilateur. Ces commentaires doivent être écrit entre les symboles (* et *) ou bien { et }.

Le texte écrit entre les deux symboles (* et *) n'est soumis à aucune contrainte syntaxique.

Exemple :

```
(* Ceci est un commentaire. J'y écris ce que
             je veux, comme je veux.  *)
```

## *3.3 Quelques outils disponibles*

Nous avons déjà rencontré un certain nombre d'outils de traitement des informations à l'occasion de la solution des problèmes.

### 3.3.1 Les opérations arithmétiques;

Certains outils s'appliquent à des nombres et fournissent comme résultat un nombre. Ce sont les opérations habituelles :

+ pour l'addition

- pour la soustraction

* pour la multiplication

/ pour la division

Lorsque les opérandes (les nombres sur lesquels porte l'opération) sont de type entier, le résultat est aussi de type entier, sauf pour la division où le résultat est toujours de type réel. Lorsqu'une des opérandes ou les deux sont de type réel, le résultat est de type réel.

A ces opérations, il faut en ajouter deux qui concernent spécifiquement le type entier et que nous n'avons pas eu l'occasion d'employer jusqu'à présent :

div pour la division entière,

mod pour le reste d'une division entière.

div désigne le **quotient** de la division entière. Ainsi

7 div 3 vaut 2 (reste 1)

15 div 4 vaut 3 (reste 3)

mod désigne le **reste** de la division entière :

7 mod 3 vaut 1

15 mod 4 vaut 3

Les opérandes doivent être de type entier et le résultat l'est également.

La priorité des opérateurs est celle habituelle en mathématique :

*, /, div et mod ont la priorité 1

+ et - ont la priorité 2

On pourrait ajouter que les parenthèses ( ) et certaines fonctions que nous verrons par la suite ont la plus grande priorité et qu'il vaut mieux quelques parenthèses superflues que des expressions illisibles.

Pour une même priorité, les opérations sont effectuées de gauche à droite. Par exemple :

A+B*C est équivalent à A + (B * C)

A*B/C*D est équivalent à (A * B * D) / C

C-D+E/K est équivalent à (C - D) + (E / K)

Signalons enfin qu'il n'existe pas d'opération d'élévation à la puissance (exponentiation). Nous devrons donc, dans un premier temps, évaluer les puissances entières en les transformant en multiplications.

Par exemple :

$A^4$ se calcule par A*A*A*A

### 3.3.2 Les fonctions arithmétiques;

Nous n'en avons découvert que deux jusqu'à présent :

- La fonction succ( ) (successeur de) qui s'applique à une information entière et fournit un résultat entier. Elle a le même effet que d'ajouter 1. Je peux cependant déjà signaler qu'il existe aussi une fonction pred( ) qui, appliquée à un nombre entier, fournit le prédécesseur de cet entier. Ces fonctions n'ont pas de sens dans le cas d'un argument réel ou chaîne de caractères.
- La fonction random( ) qui désigne un entier aléatoire compris entre 0 (inclu) et l'argument (exclu).

  Ainsi

  random(5) désigne, au hasard, l'un des entiers 0, 1, 2, 3 ou 4.

### 3.3.3 C. Les expressions "booléennes"

Ce sont celles que nous avons désignées précédemment par le vocable de "condition". Elles s'emploient fréquemment avec if, while, repeat until, ... . Elles fournissent comme résultat VRAI ou FAUX.

Elles font usage :

des symboles <, >, <=, >=, =, <>,

des connecteurs and et or,

de la négation not,

de parenthèses.

Je n'entrerai pas, pour l'instant, dans les détails syntaxiques de l'emploi de ces symboles. Je me contente de dire que, comme l'emploi des parenthèses est parfois **obligatoire**, il est conseillé d'en placer autant qu'il faut pour que les expressions soient claires et non ambiguës.

En tout cas, lorsque plusieurs comparaisons sont connectées par les mots and ou or, chacune d'elles doit figurer entre parenthèses.

Par exemple :

```
if (A<>0) and ((Compteur=-1) or (FF <= NombMax)) then
   ...
```

```
while (I <= N) and not (I > MM) do
   ...
```

Les divers symboles de comparaison pourront s'écrire aussi bien entre des informations numériques qu'entre celles de type chaîne de caractères. L'ordre sous-jacent dans ce dernier cas est celui du dictionnaire (ordre lexicographique), les majuscules précédant les minuscules.

Ainsi :

| | |
|---|---|
| 'chat' < 'chien' | est vrai |
| 'Chien' < 'chat' | est vrai |

# CONCLUSION (PROVISOIRE)

Que voici un moment mal choisi pour se quitter : la description de tous les concepts importants nous place en un point de vue d'où nous apercevons à la fois le chemin suivi et la longue route qu'il reste à parcourir.

Jetant un regard en arrière, nous discernons des repères importants : "faire faire", "marche à suivre", "structures de contrôle", "exécutant-ordinateur", etc. Ce sont ces jalons qui ont scandé notre exploration, ont orienté notre progression et ont préparé la suite du périple.

Les quelques deux cents pages de cette introduction se résument en peu de questions :

- Qu'appelle-t-on problème en programmation ?
- Quels sont les outils de conception et d'expression d'une marche à suivre ?
- Quelles sont les caractéristiques de l'ordinateur, considéré du point de vue de la programmation ?

Je vous suggère un dernier tout petit exercice : c'est d'apporter vos réponses personnelles à ces trois questions.

Si les concepts abordés sont simples, nous avons cependant mesuré que leur mise en oeuvre, lorsqu'il s'agit de faire faire une tâche par l'exécutant-ordinateur est souvent complexe et difficile. Il s'agit bien d'un mode de pensée nouveau, où "savoir faire" ne suffit plus, mais où il faut devenir capable de **dire** ce "savoir faire".

La complexité n'est pas dans la tâche à accomplir mais dans la prise de conscience de l'organisation des constituants élémentaires dont l'enchaînement conduira, sans coup férir, à l'achèvement du travail.

C'est dire que si les concepts qui sont à la base de cette pensée algorithmique sont maintenant disponibles il reste à les exercer longuement. Il faut continuer à les mettre en oeuvre en poursuivant l'analyse de tâches plus complexes, débouchant sur des programmes plus importants. La poursuite de cette découverte du pays du "faire faire" nous amènera d'abord à enrichir l'ensemble des outils dont dispose l'exécutant-ordinateur. Ceci ira de pair, bien entendu, avec l'approfondissement de la connaissance du langage Pascal. Mais au-delà de cet achèvement indispensable du portrait de l'exécutant, ce que la suite de l'apprentissage placera en avant, c'est surtout la pratique des saines attitudes de programmation : approche descendante et modulaire (et son incarnation dans l'appel de procédure), souci d'écrire des programmes robustes et aisément utilisables, préoccupation de faire clairement apparaître l'analyse menée et de rendre visible pour le lecteur la "matière grise ajoutée", etc.

Quelques concepts restent à découvrir : tableau, paramètre de procédure, variable booléenne, etc. La connaissance de ces derniers ne modifiera en rien les difficultés liées au "faire faire";

toutefois en complétant le portrait de l'exécutant-ordinateur, ces notions importantes permettront d'élargir le répertoire des tâches auxquelles il peut s'atteler.

Nous venons de faire ensemble nos premiers pas : la direction choisie oriente, conditionne et prépare la suite du périple. J'ai tenté de baliser le début de la progression de quelques idées-phares et de choisir le cap qui me paraît le plus pertinent. A vous, à présent, de poursuivre ... et rendez-vous, sans doute, dans le second volume d'"Images pour programmer".

# BIBLIOGRAPHIE


[1] ALAGIC S., ARBIB M.A., *The design of well - structured and correct programs*, Springer Verlag, New-York, 1978.

[2] ARSAC J., *Et si on essayait d'écrire de bons programmes ...*, Education et Informatique, Avril 1980, pp. 35-41.

[3] ARSAC J., *Premières leçons de programmation*, Cedic/Nathan, Paris, 1980.

[4] ARSAC J., *Jeux et casse-tête à programmer*, Dunod, Paris, 1985.

[5] ARSAC J., *La construction de programmes structurés*, Dunod, Paris, 1977.

[6] ARSAC J., *Les bases de la programmation*, Dunod, Paris, 1983.

[7] ARSAC J., *Les machines à penser. Des ordinateurs et des hommes.*, Editions du Seuil, Paris, 1987.

[8] ARSAC MONDOU O., BOURGEOIS CAMESCASSE C., GOURTAY M., *Premier livre de programmation*, Cedic/Nathan, Paris, 1982.

[9] ARSAC MONDOU O., BOURGEOIS CAMESCASSE C,. GOURTAY M., *Deuxième livre de programmation*, Cedic/Nathan, Paris, 1983.

[10] ARSAC MONDOU O., BOURGEOIS CAMESCASSE C., GOURTAY M., *Pour aller plus loin en programmation*, Cedic/Nathan, Paris, 1983.

[11] ARSAC MONDOU O., BOURGEOIS CAMESCASSE C., GOURTAY M., *Option Informatique. Classe de seconde.*, Editions Fernand Nathan, 1987.

[12] ARSAC MONDOU O., BOURGEOIS CAMESCASSE C., *Option Informatique. Classe de Terminale.*, Editions Fernand Nathan, 1988.

[13] BALACHEFF N., KUNTZMANN J., LABORDE C., *Formation mathématique des instituteurs avec ouverture sur l'informatique*, Cedic/Nathan, Paris, 1981.

[14] BAUER F.L., *Top down teaching of informatics in secondary school*, in O. LECARME and R. LEWIS (Eds), *Computers in Education* North Ho Publ. Co, 1975, pp. 53-61.

[15] BIONDI J., CLAVEL G., *Introduction à la programmation 1. Algorithmique et langage.*, Masson, Paris, 1984.

[16] CLAVEL G., BIONDI J., *Introduction à la programmation 2. Structures de données.*, Masson, Paris, 1984.

[17] CLAVEL G. , JORGENSEN F-B., *Introduction à la programmation. Exercices corrigés.*, Masson, Paris, 1985.

[18] CROZET J-M., SERAIN D., *Le Langage Pascal*, Masson, Paris, 1980.

[19] DEFAYS D., *L'esprit en friche*, Pierre Mardaga, 1988.

[20] DELACHARLERIE A., *Informatique et Programmation Structurée*, Wesmael-Charlier, Leuze-Longchamps, 1984.

[21] DELANNOY C., *Initiation à la programmation*, Eyrolles, Paris, 1986.

[22] DIEUDONNE D., BERTHON F., NEVIANS A., *Apprendre à Programmer. Niveau 1.*, Cedic/Nathan, Paris, 1985.

[23] DIJKSTRA E., *A short introduction to the art of programming*, International Summer School on Structured Programming,, Lecture Notes, Eindhoven, 1971.

[24] DIJKSTRA E.W., FEIJEN W.H.J., *A Method of Programming*, Addison-Wesley, 1988.

[25] DUCHATEAU C., *Informatique et Enseignement Secondaire., Une proposition de bibliographie*, CeFIS, Facultés N-D de la Paix, Namur, 1982.

[26] DUCHATEAU C., *Programmer ?*, CeFIS, Facultés N-D de la Paix, Namur, 1982.

[27] DUCHATEAU C., *L'informatique, technique nouvelle ou nouvel humanisme ?*, CeFIS, Facultés N-D de la Paix, Namur, 1982.

[28] DUCHATEAU C., *Programmer ! Pour une découverte des méthodes de la programmation.*, Wesmael-Charlier, Leuze-Longchamps, 1983.

[29] DUCHATEAU C., *Incursion au pays du faire faire*, CeFIS, Facultés N-D de la Paix, Namur, 1983.

[30] DUCHATEAU C., *Du savoir faire au savoir faire faire*, CeFIS, Facultés N-D de la Paix, Namur, 1984.

[31] DUCHATEAU C., *Additionner deux entiers ... ou faire additionner deux entiers.*, *Du problème à l'algorithme.*, CeFIS, Facultés N-D de la Paix, Namur, 1985.

[32] DUCHATEAU C., *Compléments sur le Langage Pascal et le système U.C.S.D.*, CeFIS, Facultés N-D de la Paix, Namur, 1985.

[33] DUCRIN A., *Programmation. 1. Du problème à l'algorithme.*, Dunod, Paris, 1984.

[34] DUCRIN A., *Programmation. 2. De l'algorithme au programme.*, Dunod, Paris, 1984.

[35] ENGEL A., *Mathématique élémentaire d'un point de vue algorithmique*, Cedic/Nathan, Paris, 1979.

[36] GERBIER A., *Mes premières constructions de programmes*, Lecture Notes in Computer Science, Springer Verlag, New-York, 1977.

[37] GOLDSCHLAGER L., LISTER A., *Informatique et algorithmique*, InterEditions, Paris, 1986.

[38] GRAM A., *Raisonner pour programmer*, Dunod, Paris, 1986.

[39] GREGOIRE, *Cours d'Informatique. Programmation. Tome 1.*, E.S.I., Paris, 1983.

[40] GREGOIRE, *Cours d'Informatique. Programmation. Tome 2.*, E.S.I., Paris, 1984.

[41] GRIBAUMONT A., LATOUCHE G., ROGGEMAN V., *Algorithmes, Programmes et Langage Pascal*, A. de Boeck, Bruxelles, 1984.

[42] JAMES M., *Mieux programmer*, Belin, Paris, 1985.

[43] JAMES M., *Elegant Programming*, Computing Today, Août 1982, pp. 24-37.

[44] KNUTH D.E., *The art of computer programming. Vol. 1 : Fundamental algorithms.*, Addison-Wesley, 1973.

[45] KNUTH D.E., *The art of computer programming. Vol. 2 : seminumerical algorithms.*, Addison-Wesley, 1981.

[46] KNUTH D.E., *The art of computer programming. Vol. 3 : sorting and searching.*, Addition-Wesley, 1973.

[47] LAURENT J-P., *Intiation à l'analyse et à la programmation*, Dunod, Paris, 1982.

[48] LAURENT J-P., AYEZ J., *Exercices commentés d'Analyse et de Programmation*, Dunod, Paris, 1985.

[49] LE BEUX P., *Introduction au Pascal*, Sybex, Paris, 1980.

[50] LECOMTE P., MEZOTTE D., *Accès à l'informatique et à la programmation*, Hermann, Paris 1988.

[51] LEDGARD H.F., *Proverbes de Programmation*, Dunod, Paris, 1975.

[52] LEDGARD H., NAGIN P., HUERAS J., *Pascal with style; programming proverbs.*, Hayden Book Cy, 1979.

[53] LESUISSE R., BORSU A., *Initiation aux raisonnements de la programmation*, Presses Universitaires de Namur, Namur, 1987.

[54] LIGNELET P., *Algorithmique, Méthodes et Modèles*, Masson, Paris, 1985.

[55] LUCAS M., PEYRIN J-P., SCHOLL P-C., *Algorithme et représentation des données., 1. Files, automates d'états finis*, Masson, Paris, 1983.

[56] LUCAS M., *Algorithme et représentations des données., 2. Evaluations, arbres, graphes, analyse de textes.*, Masson, Paris, 1983.

[57] MEYER B., BAUDOIN C., *Méthodes de programmation*, Eyrolles, Paris, 1980.

[58] NIVAT M., *Savoir et savoir-faire en informatique*, La Documentation Française, Paris, 1983.

[59] PAIR C., MOHR R., SCHOTT R., *Construire les algorithmes, les améliorer, les connaître, les évaluer.*, Dunod, Paris, 1988.

[60] RICHARD C., RICHARD P., *Initiation à l'algorithmique*, Belin, Paris, 1981.

[61] RICHARD C., RICHARD P., *Programmatique. Initiation à la programmation méthodique.*, Belin, Paris, 1984.

[62] THURNER R., *Structured programming. Self instruction course.*, Pitman, 1983.

[63] TURSKY W. H., *What is Computer Science or Informatics*, in W.M. TURSKY (Edr), *Programming Teaching Techniques*, North Holland Publ. Co, 1973, p. 208.

[64] WIRTH N., *Introduction à la programmation systématique*, Masson, Paris, 1981.

[65] WIRTH N., *Systematic programming. An introduction.*, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[66] WIRTH N., *Algorithms + Data structures = programs*, Prentice-hall, Englewood Cliffs, New jersey, 1976.

# Index alphabétique

## U



## V



## W