LOGORITHMES

PROGRESSER en LOGO

en s'initiant

au traitement de **LISTES**

à la **RECURSIVITE**

aux **ARBORESCENCES**

aux **MOTEURS d'INFERENCE...**

par Bernard GENETAY et Danielle SALLES

Une publication de l'IREM de Basse-Normandie.

1006 IN-Z-GENE

SOMMAIRE

- o -

AVERTISSEMENT

Cette brochure est destinée plus particulièrement au lecteur quelque peu familiarisé avec LOGO: elle sera plus facile à lire si on sait déjà dessiner, définir une procédure, affecter une valeur à une variable...

Pour "rafraîchir" les mémoires et ne pas contraindre l'utilisateur à consulter un autre ouvrage, la première partie est constituée d'un mémento LOGO. Le but des deux parties suivantes est d'amener (tout doucement !) le lecteur à écrire des programmes exploitant une partie des nombreuses possibilités de LOGO, à savoir: la récursivité, le traitement de listes etc...

Comme chacun le sait plus ou moins, LOGO est issu de LISP, langage majeur de l'Intelligence Artificielle. Bien que beaucoup plus lent, LOGO en a pratiquement toute la puissance et, de ce fait, peut servir à l'initiation aux techniques de cette discipline: parcours d'arbres, moteurs d'inférence etc...

Les programmes proposés sont les plus simples possibles et, de ce fait, comportent des défauts: lourdeur d'écriture et lenteur d'exécution, nous espèrons cependant qu'ils vous intéresseront et vous ouvriront de nouvelles perspectives en informatique.

BG. DS.

PREMIERE PARTIE

I Le langage LOGO. Pourquoi l'utiliser ?

C'est un langage interprété, ce qui permet la mise au point interactive et facilite beaucoup l'apprentissage. Le paragraphe II explique la différence entre langages compilés et interprétés.

Il permet aussi un style appelé programmation fonctionnelle où l'affectation disparait au profit de la composition de fonctions.

Il faut bien reconnaitre que LOGO n'est pas très efficace du point de vue informatique et que personne ne s'amuse à écrire de très gros logiciels en LOGO. Voici par exemple les temps d'exécution du même programme en trois langages différents (il s'agit de calculer le 15ème élément de la suite de Fibonacci en utilisant la récursivité).

LOGO	9,2 s	langage interprété
LISP	0,72 s	langage interprété
C	0,05 s	langage compilé

Alors pourquoi LOGO ? il est intéressant surtout par ses qualités pédagogiques. Il permet d'aborder à peu de frais de nombreuses notions utiles en informatique et aussi certaines fécondes en mathématiques.Citons par exemple:

- langage compilé ou interprété
- fonctions et procédures
- variables locales et globales
- paramètres formels et arguments réels d'une fonction ou d'une procédure
- passage de paramètres par valeur et par adresse
- récursivité (étroitement liée à la notion de récurrence en mathématiques et qui peut l'illustrer)

LOGO est bien autre chose que la célèbre tortue. Ce langage permet d'utiliser les 4 opérations sous forme de fonctions en notation préfixée. Il est très utile de faire manipuler les élèves sur des instructions du genre ECRIS SOMME DIFF 3 5 DIV 5 4 pour (3 - 5) + (5 / 4). Elles sont en effet très proches du langage naturel.

Il est aussi facile de définir une fonction (au sens mathématique) et de proposer des activités à base de calculs d'images, d'antécédents, de compositions ...Il est possible d'illustrer la notion d' ensemble de définition d'une fonction prédéfinie ou personnelle :

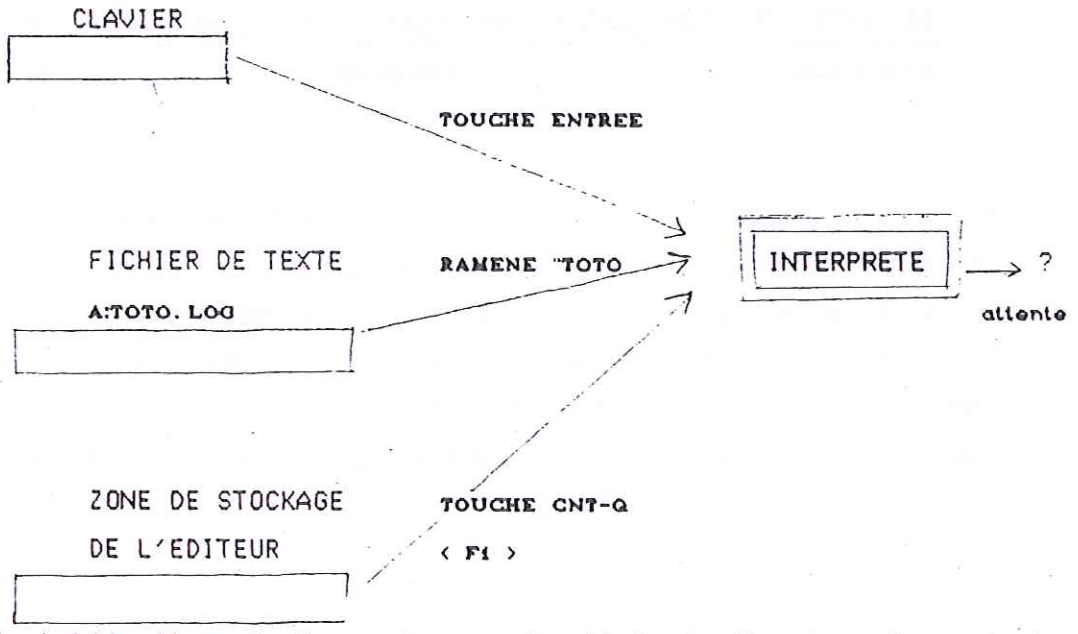
```
? ECRIS RC -25    (racine carrée de -25  ce qui produit une erreur)
ou  POUR F :X
    RENDS 1 / (:X - 5)
    FIN
    ? ECRIS F 5     ( calculer F(5) ce qui produit aussi
                    une     erreur, incitation à la réflexion )
```

LOGO possède, prédéfinies, les primitives de traitement de listes d'objets. Il est assez facile de les utiliser pour créer des structures classiques en informatique comme les piles, les files, les arbres, les graphes.

Enfin LOGO est en français. Ses mots-clés sont en général transparents et ses messages d'erreurs assez "conviviaux". LOGO est agréable pour fabriquer et tester rapidement une maquette de programme qui sera traduite ensuite en un autre langage plus efficace mais plus complexe.

II Langage interprété, Langage compilé.

Lorsque LOGO est actif, il affiche un point d'interrogation. C'est son *signe d'invite* ou *prompt*. Il attend des ordres. Ces ordres peuvent venir de plusieurs sources, voyez l'illustration :



Ainsi, habituellement, les ordres sont soit tapés directement au clavier, soit lus dans un fichier, soit proviennent de l'éditeur.

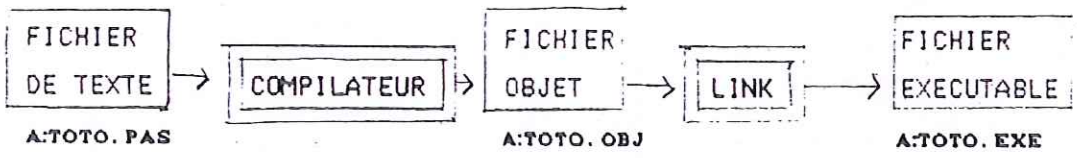
L'INTERPRETE examine un ordre, l'analyse du point de vue syntaxique, puis du sens et déclenche alors les actions convenables. S'il y a une erreur, un message est affiché.
Il passe ensuite à l'examen de l'ordre suivant, s'il y en a un, ou se remet en attente en affichant "?".
Remarquons que l'interprète doit être présent et actif pour exécuter un programme que nous avons écrit. Ce n'est pas le cas pour les langages compilés:

Pour ceux-ci, la démarche est nettement plus complexe:

1) écrire le programme à l'aide d'un EDITEUR DE TEXTES qui fournit un fichier sur disquette (par exemple TOTO.PAS en PASCAL).

2) lancer le COMPILATEUR en lui fournissant comme donnée le fichier précédent. Celui-ci fait une analyse syntaxique puis sémantique et au lieu d'exécuter les instructions comme l'interprète, fabrique un fichier qui contient le "savoir-faire" (TOTO.OBJ). Ce fichier n'est pas encore exécutable.

3) lancer LINK (éditeur de liens) avec en données les différents fichiers objets (il peut y en avoir plusieurs) qui composent le programme.
On obtient enfin un fichier EXECUTABLE (TOTO.EXE ou TOTO.COM) autonome.



Les erreurs éventuelles peuvent se produire à trois stades :la compilation, l'édition de liens, l'exécution. Il faut alors rechercher l'erreur dans le fichier original puis reprendre la démarche au début.
C'est assez fastidieux, mais heureusement, les compilateurs récents de type TURBO... améliorent nettement les choses.
Remarquons en passant que l'interprète LOGO a été développé de cette façon.

III L'éditeur

LOGOPLUS est disponible pour Nanoréseau ou PC et il existe quelques différences (la mémoire du M05 est très réduite) .
Les touches relatives au PC sont entre < >

Pour entrer dans l'éditeur à partir du prompt "?" du LOGO
ED
Pour en sortir avec interprétation CNT-Q <F1>
sans interprétation CNT-C <F10>
Pour se déplacer dans le texte :
← → ↑ ↓ pour de petits déplacements
INS puis ← ou → ↑ ↓ vers les limites du texte
(INS INS en cas de remord)
Sur PC: flèches <PgDn> <PgUp> <CTRL-PgUp> <CTRL-PgDn>
<Home> <End> <SHIFT←> <SHIFT→>

Corrections
on est toujours en mode insertion (ce qui est frappé pousse le reste du texte)
EFF <Del> sous le curseur
CNT-D (←) à gauche du curseur
CNT-S <CTRL→> pour supprimer le reste d'une ligne
CNT-R <CTRL←> pour recopier ce qui vient d'être supprimé par CNT-S
Sur PC on dispose en plus :
<F6> recherche <ENTREE> valide la recherche
<ESC> sort de la recherche
<CTRL-F6> reprend la recherche
<F7> recherche et remplacement
options "T" pour tout le texte
"D" ou "entrée" sur confirmation par "O"
<ENTREE> valide la substitution
<ESC> sort de la substitution
<CTRL-F6> reprend la recherche et substitution.

IV Les objets : les mots et les listes d'objets logo

les mots - les chaines de caractères "TRUC
- les noms de procédures et fonctions TRUC
- la valeur d'une variable :TRUC

Les mots sont séparés par au moins un espace. Pour rendre à ce caractère espace le statut de caractère ordinaire il faut le faire précéder du caractère $.

les listes
exemples [LE [JOLI PETIT] CHAT] est une liste de 3 éléments
[] la liste vide
[[]] une liste à 1 élément

V Les procédures et les fonctions

Les paragraphes qui suivent n'ont pas pour but d'être un manuel de référence mais seulement de présenter en peu d'espace la liste des principaux mots-clés.

Les fonctions rendent un objet (à charge pour l'appelant d'en faire ce qu'il lui plait) .Ceci se fait par l'instruction RENDS. Les procédures sont chargées de faire une action et ne rendent pas de résultat. On parle parfois de "procédures opérations" et de "procédures commandes" au lieu de fonctions et procédures.
Les prédicats sont des fonctions particulières qui rendent VRAI ou FAUX

Les PRIMITIVES sont les procédures et fonctions prédéfinies.
Certaines ont deux formes, complète (TOURNEDROITE) et abrégée (TD) .
En voici quelques-unes:

Sortie
EC (ECRIS) ,TAPE (pas de retour à la ligne)

Présentation de l'écran
FCT 5 (fixe couleur du texte),FCFT 7 (fixe couleur du fond du texte)
FCURS [10 20] met le curseur en col 10 ligne 20
ME 5 écran mixte ,5 lignes en bas de l'écran graphique pour du texte.
VE vide écran graphique
VT vide texte: efface l'écran texte

Entrée
LISCAR (lis un caractère), LISMOT (mot ou nombre), LL (ligne)

Affectation (associer une valeur à une variable)
DONNE "STOCK 500
DONNE "L [IL FAIT TRES BEAU]

Valeur d'une variable (fonction qui rend la valeur de la variable)
CHOSE "STOCK qui s'écrit en abrégé :STOCK

Fonctions mathématiques

SOMME,PROD,DIFF,QUOT,RESTE (division entière), DIV (décimale) à deux arguments

RC (√),COS,SIN,ENT,HASARD à un seul argument

< ATN > < EXP > < LN >

Les opérations peuvent prendre la forme infixée habituelle

(EC 3 + 4 au lieu de EC SOMME 3 4)

+ - * /

Les parenthèses ( ) modifient la priorité .

une facilité sur PC: les fonctions et procédures sont prévues en général pour un nombre précis de paramètres. Le calcul de (((50+10)+15)+5) s'écrit normalement, si ont tient à utiliser la forme infixée :

? EC SOMME SOMME SOMME 50 10 15 5

LOGO permet d'utiliser des paramètres multiples à condition de mettre l'instruction entre parenthèses (pour certaines primitives seulement).

EC (SOMME 50 10 15 5)

Les prédicats prédéfinis (se terminent par convention par ?)

PLP? (<), PLG? (>), EGAL? (=) à deux arguments

<, >, = utilisés sous forme infixée leur sont équivalents

MOT?, NOM?, LISTE?, NOMBRE?, PRIM? ("TRUC est-il une primitive ?), VIDE? sont des prédicats à un argument.

CONTACT? (crayon optique appuyé ?) ,TOUCHE? (clavier appuyé ?) à 0 argument.

ET, OU, NON sont des opérations logiques sur les prédicats

Contrôle des programmes

POUR..... FIN (définir une procédure ou fonction)

RENDS un_objet (spécifique d'une fonction)

REPETE 20 [liste_d_instructions]

SI expression_logique [liste_d_instructions_si_VRAI]
[liste_d_instructions_si_FAUX]

"alors" et "sinon" sont sous-entendus.

STOP (arrêt de la procédure en cours, la main rendue à l'appelante)

LOGO (arrêt et retour au LOGO)

EXEC [liste_d_instructions] < EXECUTE ... sur PC)

(exécuter les instructions contenues dans une liste)

SYSTEME < .AUREVOIR > (pour quitter LOGO et retourner au système d'exploitation)

VI La tortue

l'écran graphique Nanoréseau :

-160 ⟷ 159 ; 100 ↑ ↓ -99

0	8	noir	4	12	bleu
1	9	rouge	5	13	magenta
2	10	vert	6	14	cyan
3	11	jaune	7		blanc
			15		orange

Une instruction graphique met automatiquement en mode graphique

Orientation de la tortue (sans avancer)

de 0 à 360 comme en marine (0=nord 90=est ...)

- *absolue*

FCAP 225 (fixe le cap au sud-ouest)

CAP rend l'orientation de la tortue

- *relative à son orientation actuelle*

TG 90,TD 25 (tourne à gauche ,tourne à dtoite)

Positionner

ORIGINE (tortue au centre cap au 0)

FPOS [num_col num_li] (fixe la position absolue sur l'écran)

Déplacements

AV 20 ,RE 50 (avance ou recule - en points d'écran)

Présentation

VE (vide l'écran), NETTOIE (idem mais la tortue reste à sa place)

CT (cache tortue), MT (montre tortue), LC (leve crayon), BC (baisse crayon)

CLOS (limite le champ de la tortue)

FEN (l'écran est une fenêtre sur un champ immense)

ENROULE (l'écran devient "sphérique")

FCC coul (fixe la couleur du crayon)

FCFG coul (fixe la couleur du fond graphique), FCB coul (fixe couleur bord)

FECH [n1 n2] (fixe l'échelle 0 à 200)

Demander certains renseignements à l'ordinateur

CAP	(rend le cap actuel)	
POS	(	la position de la tortue)
CC	(	la couleur du crayon)
CF	(	la couleur de fond)
ECH	(	la liste definissant l'échelle)
BC?	(	VRAI si le crayon est baissé)
VISIBLE?	(	VRAI si la tortue est visible)

VII Les mots et les listes

a) les primitives communes aux mots et listes

prédicats

VIDE?, EGAL?, MOT?, LISTE?, NOMBRE?

MEMBRE? obj1 obj2 (la lettre obj1 est-elle dans le mot obj2 ?)
(ou obj1 est-il élément de la liste obj2 ?)

accès aux éléments en lecture

PREM obj1 ou PREMIER obj1 (rend la première lettre du mot obj1 ou le premier élément de la liste obj1)

DER obj ou DERNIER obj (rend le dernier élément de l'objet)

SP obj ou SAUFPREMIER obj (rend le mot ou la liste amputé(e) de sa tête)

SD obj ou SAUFDERNIER obj (..... queue)

ITEM n1 obj (rend le n1-ième élément de l'objet)

divers

COMPTE obj (rend le nombre d'éléments)

EC obj et TAPE obj (affiche en ôtant les [] extérieurs)

MO obj ou MONTRE obj (affiche en gardant ces crochets)

b) primitives sur les mots

MOT mot1 mot2 (ex: MOT "BON "JOUR rend le mot BONJOUR)

ASCII mot (rend le code du premier caractère du mot)

CAR n1 (rend le caractère dont le code est n1)

c) primitives sur les listes

LISTE obj1 obj2 (rend la liste à deux éléments [obj1 obj2])

PH liste1 liste2 (ou PHRASE) (rend la liste qui est fusion des 2 listes)

MP obj liste (ou MPREMIER) (rend cette liste ',l'obj inserré en tête)

MD obj liste (ou MDERNIER) (idem en fin de liste)

VIII Musique

DUREE n (de 1 à 96) (fixe la durée des prochaines notes jouées)

96 (ronde) 48 24 12 6 3 (triple croche)

72 36 18 9 pour les notes pointées

64 32 16 8 4 2 pour les triolets

OCTAVE n (de 1 à 5)

TIMBRE n (de 0 à 255)

JOUE melodie (exemple JOUE "DODO<#REMIFASO)
les notes DO RE MI FA SO LA SI PA (pause)
Les altérations <# (diese) et <b (bémol) après la note

IX Les listes de propriétés

En plus d'une éventuelle valeur (attribuée par DONNE ...), un identificateur LOGO peut être complété par une *liste de propriétés* :

```
DPROP "MEDOR "ESPECE "CHIEN
DPROP "MEDOR "COULEUR "NOIR
LPROP "MEDOR  (rend la liste des propriétés associées au nom)
RPROP "MEDOR "COULEUR   (rend la valeur de la propriété)
APROP "MEDOR "COULEUR   (annule et ôte la propriéte de la liste)
EFPROP "MEDOR           (efface toutes les propriétés du nom)
```

X Accès à la mémoire

lecture

```
CATALOGUE (pour examiner la disquette)
CONTENU (rend la liste de tous les mots connus de LOGO)
IMTOUT  (affiche tout l'espace de travail)
IMNS    (affiche les noms créés et leur chose)
IMTS    (affiche les titres des procédures de l'espace de travail)
IM [liste_de_procedures] ( affiche  des définitions de procédures )
PLACE   ( rend un nombre indiquant la place disponible )
.EXA adresse  (rend le contenu de l'octet concerné de 0 à 255)
```

modifications

```
.EFT ( efface tout l'espace de travail , perte du travail fait )
EFN nom_de_variable (efface le nom et sa chose de l'esp. de tr.)
EFP nom_de_procédure (efface le nom de la procédure de l'esp. de tr.)
RECYCLE  ( libère de la place )
.DEP adresse n  (déposer le nombre n à l'adresse précisée. DANGER !!!)
```

XI Accès aux périphériques

```
FLECTEUR n ( 0 pour A:, 1 pour B: ..., fixe le lecteur actif)
CATALOGUE  (pour le lecteur de disque actif)
SAUVED nom_fichier   (sauvegarde du contenu de l'éditeur)
CHARGE nom_fichier   (rappel dans l'éditeur d'un fichier)
SAUVE  nom_fich  [  liste_de_procédures  ]  (sauvegarde travail)
RAMENE nom_fich      (interprétation d'un fichier de procédures)
DETRUIS nom_fich ( efface un fichier de la disquette )
ENTREE n (0 à 5) ( précise d'où vient le flux d'entrée )
```

SORTIE n (précise où va le flux de sortie)
0 (nul) 1 (console) 2 (imprimante) 3 (port série)
4 (éditeur) 5 (réseau)
ED ou ED [liste_de_procédures]
COPIE (recopie de l'écran sur l'imprimante)

La liste des instructions n'est pas complète, il faut se reporter au manuel de référence pour avoir plus de détails.

XII QUE FAIRE POUR PROGRESSER RAPIDEMENT?

- D'abord installez-vous sans complexes à votre clavier et amusez-vous. Acceptez que l'ordinateur vous envoie, au début, de nombreux messages d'erreur. Un ordre erronné n'abîme pas la machine.

- Utilisez exclusivement (ou presque) la *programmation descendante*, c'est à dire partir du problème général et l'affiner en problèmes plus petits jusqu'à ce que la programmation de chacun soit évidente .

- Survolez les manuels d'initiation et de référence sans essayer de tout comprendre, cela viendra plus tard.

- Tapez et surtout modifiez à votre guise des programmes proposés et inventez en d'autres

- Relisez de façon plus approfondie les manuels.

- Et surtout amusez-vous bien ...

DEUXIEME PARTIE

EXEMPLES de programmes LOGO

Remarques préalables importantes:

- Une ligne LOGO peut se prolonger sur plusieurs lignes de l'écran. Ceci se traduit par un signe particulier dans la dernière colonne.
- L'instruction SI [...] [...] doit être écrite sur la même ligne LOGO qui peut donc être très longue.
- Un programme LOGO n'est pas toujours très lisible naturellement. On peut augmenter considérablement la lisibilité en faisant un usage intensif de "l'indentation" (décalages et alignements). Bien sûr, cela augmente la taille du programme en ajoutant des espaces. Il peut être intéressant de disposer de deux versions du même programme: l'une indentée ,l'autre condensée.
- Sur Nanoréseau, les signes d'opérations et les nombres doivent être encadrés par des espaces. Le LOGO du Nanoréseau est bien plus limité que celui du PC.
- Certains programmes ou certaines instructions pourraient être plus simples ou peuvent paraître artificiels, mais le but est de mettre en lumière certains aspects de LOGO

EXEMPLE 1

Bien entendu, directement au clavier, vous vous êtes d'abord amusés avec la tortue, avec les couleurs, vous avez observé ce qui se passe pour un AVANCE avec un grand paramètre en modes CLOS, ENROULE , FEN, vous vous êtes même doucement risqués dans l'éditeur. Passons maintenant aux choses sérieuses.
Un premier type d'activités tourne autour des contructions de polygônes divers, rectangles,parallélogrammes,polygônes réguliers, étoilés ...
par exemple:

```
POUR CARRE :COTE
REPETE 4 [AV :COTE TD 90]
FIN
? CARRE 50
? CARRE 20
```

pour essayer notre procédure. Le point d'interrogation indique le mode direct du LOGO

On distingue le nom d'une variable ("COTE) de sa valeur (:COTE ou CHOSE "COTE). Ceci permet que le nom de la variable soit calculé. Par exemple CHOSE MOT "C "OTE est correct à la place de :COTE.
Pour dessiner le carré où l'on veut, lever le crayon, déplacer la tortue ou fixer la nouvelle position puis baisser le crayon.

```
POUR CARRE2 :X :Y :C
LC
FPOS LISTE :X :Y
BC
CARRE :C
FIN
? CARRE2 -50 20 50
```

Remarquer que FPOS [:X :Y] ne fonctionne pas car l'intérieur de la liste ne serait pas evalué. FPOS a besoin d'une liste formée des valeurs des nouvelles coordonnées.

EXEMPLE 2

Dessiner un triangle de côtes 15,18,20 centimètres sur l'écran.
difficultés 1) retrouver les mesures des angles. Cela peut être résolu par la trigonométrie ou la construction sur une feuille de papier puis mesure au rapporteur selon la classe.

2) traduire les centimètres en "points d'écran". Pour cela dessiner un segment (par exemple AV 200) , mesurer ce segment (par exemple 13,5) puis déterminer le coefficient de proportionnalité (et le garder en conserve si l'on veut).

```
DONNE "K  200 / 13.5     ( K est une variable globale )
```

puis

```
? EC :K * 15
? EC :K * 18
? EC :K * 20
```

Il peut être aussi intéressant d'écrire une procédure qui à partir d'une liste de nombres et d'un coefficient rend la liste des nombres multipliés par ce coefficient.
Une fois connus les angles et les côtés, la construction est facile.

EXEMPLE 3

dessiner un cercle de diverses manières

```
POUR CERCLE1
REPETE 360 [AV 1 TD 1]
FIN
```

On obtient bien un cercle mais ce cercle n'est pas réglable, ni le centre ni le rayon. Ce n'est pas satisfaisant.

```
DONNE "PI 3.14159
POUR CERCLE2  :X :Y :R
LC
FPOS LISTE :X :Y
FCAP 0   AV :R   TD 90
BC
CAUX :R
FIN

POUR CAUX :RAYON
REPETE 360 [AV (:RAYON * :PI) / 180
            TD 1]
FIN
```

On utilise la formule de la longueur d'un arc pour calculer de combien avancer si on tourne de 1 degré.
La procédure CAUX se charge de dessiner le cercle.

Cette procédure CERCLE2 a un défaut: elle ne remet pas les lieux en l'état où ils étaient en entrant. La tortue a changé de sens et de position, c'est un *effet de bord* indésirable. Remède:
au début de CERCLE2 ajouter

```
LOCALE "SENS
LOCALE "COORD
DONNE "SENS CAP
DONNE "COORD POS
```

et juste avant la fin

```
FCAP :SENS
FPOS :COORD
```

Les variables locales "SENS ET "COORD sont chargées de garder les valeurs initiales pour les restituer à la fin. Ces variables sont *locales*, c'est à dire qu'elles n'existent que pendant l'exécution de CERCLE2 et n'interfèrent pas avec d'autres variables éventuelles de même nom.
Le cercle est dessiné par une procédure *itérative* .On peut aussi le faire à l'aide d'une procédure *récursive*.

```
POUR CERCLE3
AV 1 TD 1
CERCLE3
FIN
```

La procédure se rappelle elle-même. Elle ne s'arrête pas et il faut recourir à la touche d'arrêt autoritaire pour la stopper. Ceci nous conduit à remarquer qu'une procédure récursive doit nécessairement contenir une condition d'arrêt.

```
POUR CERCLE4
AV 1 TD 1
SI EGAL? CAP
         0
   [STOP]
   [CERCLE4]
FIN
```

EXEMPLE 4

La suite de FIBONACI et la récursivité.

On peut imaginer de nombreuses suites récurrentes semblables

F(0)=1 F(1)=1 F(N)=F(N-1)+F(N-2)

L'expression est très simple et il peut sembler extraordinaire que cela fonctionne :

```
POUR FIBO :N
SI EGAL? :N 0 [RENDS 1]
SI EGAL? :N 1 [RENDS 1]
RENDS SOMME FIBO :N - 1 (ou RENDS FIBO :N - 1 + FIBO :N - 2)
            FIBO :N - 2
FIN
? EC FIBO 13
```

Remarquer que FIBO est une fonction qui retourne un nombre et que de ce nombre il faut faire quelque chose, par exemple l'écrire ou l'utiliser comme argument d'une procédure ou fonction.

On peut imaginer des compositions de fonctions:

```
? EC FIBO FIBO 4
```

Comment ça marche ?

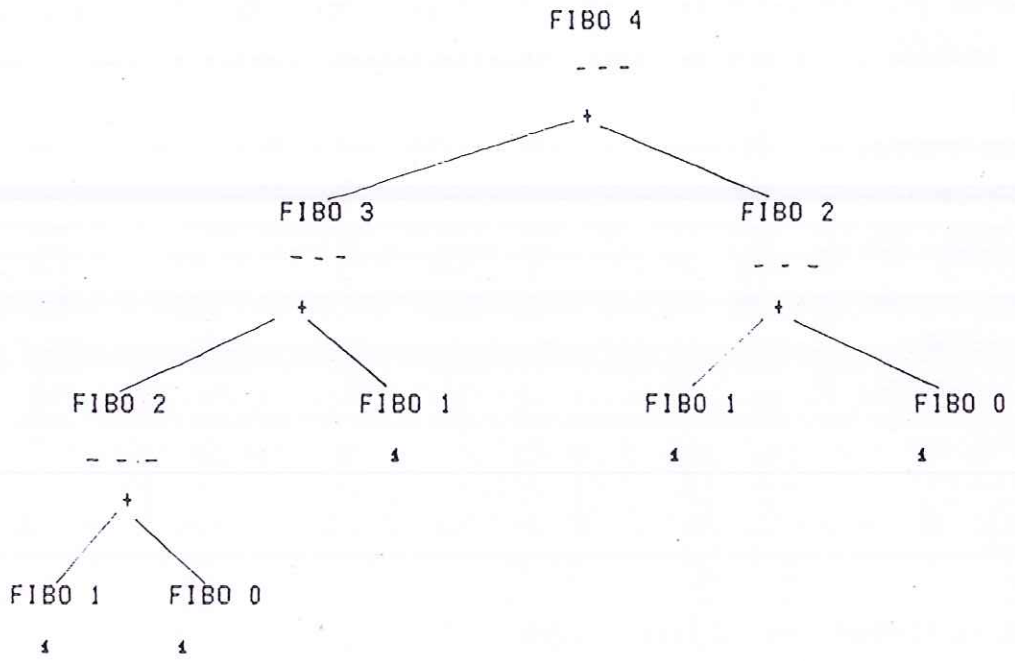
le mécanisme consiste en une *pile d'exécution.*

Chaque appel de fonction au moment de l'exécution crée sur la pile un *bloc d'activation* qui s'empile sur ceux qui existent déjà . Il contient entre autres les emplacements mémoire pour

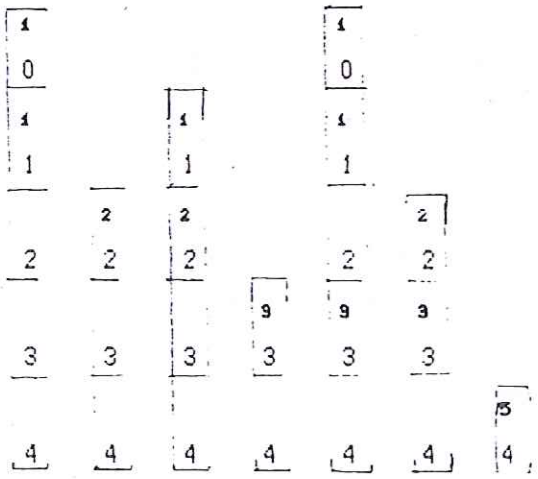
- les arguments de la fonction
- les variables locales s'il y en a
- le résultat

exemple: calculons FIBO 4

Voici *l'arbre des appels*



Voici l'évolution de la pile d'exécution.Sitôt qu'un calcul est possible, les blocs d'activation inutiles sont dépilés.
Les nombres en gros chiffres sont les arguments d'appel successifs de FIBO, ceux en petits chiffres les résultats des appels



On remarque qu'il reste un bloc sur la pile et que ce bloc contient le résultat. Il sera dépilé après utilisation par EC
Le programme FIBO a des avantages: il est simple et proche de la définition de la fonction, mais il n'est pas très efficace: les résultats intermédiaires sont oubliés sitôt utilisés.
Une méthode plus complexe mais infiniment plus rapide fait appel à *des paramètres accumulateurs*.

```
POUR FIBOA :N
RENDS FIBAUX :N 1 1
FIN

POUR FIBAUX :N :RES1 :RES2
SI EGAL? :N 0 [RENDS :RES1]
SI EGAL? :N 1 [RENDS :RES2]
RENDS FIBAUX :N - 1
             :RES2
             :RES1 + :RES2
FIN
? FIBOA 4
```

Pour comprendre comment ça fonctionne, écrire l'arbre des appels.

EXEMPLE 5

Ecrire une liste à l'envers

```
POUR RETOURNE :L
SI EGAL? :L [] [RENDS []]      liste vide
SI EGAL? SP :L                 liste à un seul élément
         []
   [RENDS :L]
RENDS MD PREM :L               autre cas
         RETOURNE SP :L
FIN
? DONNE "L1 [1 2 3 4 5]
? EC RETOURNE :L1          fournit 5 4 3 2 1 (car EC ôte les crochets)
```

Dans le cas où la liste a au moins 2 éléments, on met le premier à la fin du reste de la liste supposé déjà retourné.

EXEMPLE 6

Un dessin récursif : feu d'artifice

Le paramètre :N indique la complexité du dessin et le nombre de pas de la récursivité

```
POUR FEU :N :TAILLE
SI :N = 0 [STOP]
REPETE 8 [AV :TAILLE
          FEU :N - 1
              :TAILLE / 3
          RE :TAILLE
          TD 45 ]
FIN
? FEU 3 50
```

Après avoir avancé, on dessine un feu plus simple et plus petit, on revient au départ puis on tourne vers une autre direction et on recommence.

EXEMPLE 7

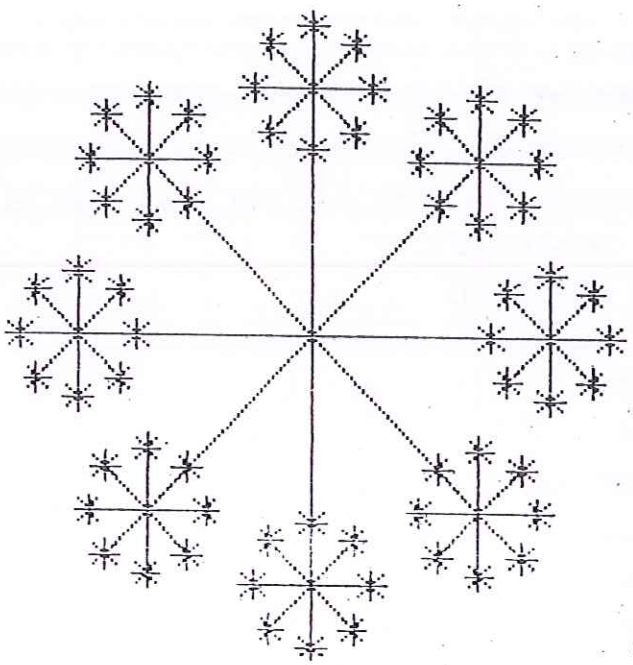
Dessiner un arbre bifide

On suppose la tortue dirigée vers le haut au départ

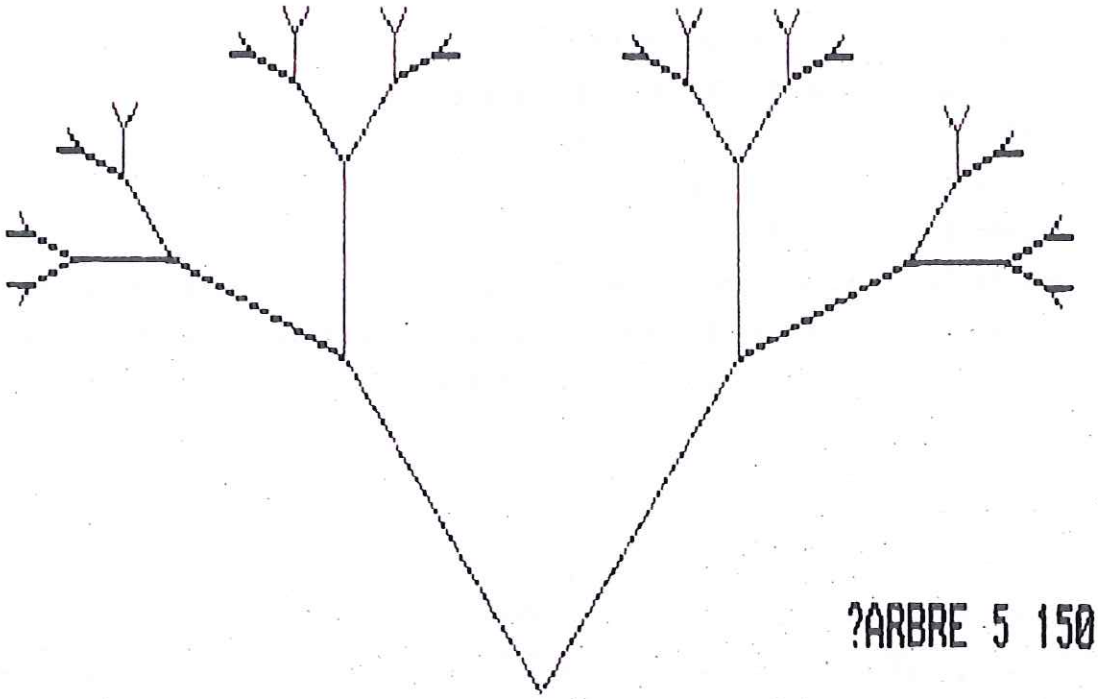
```
POUR ARBRE :N :TAILLE
SI :N = 0 [STOP]
TG 30 AV :TAILLE
ARBRE :N - 1 :TAILLE / 2
RE :TAILLE TD 60
AV :TAILLE
ARBRE :N - 1 :TAILLE / 2
RE :TAILLE TG 30
FIN
? ARBRE 3 50
```

La procédure est comparable à la précédente.

Voir en annexe d'autres exemples de dessins récursifs



?FEU 3 100



?ARBRE 5 150

EXEMPLE 8

Remplacer toutes les occurences d'un élément dans une liste par un autre et réciproquement.

```
POUR ECHANGE :E1 :E2 :L
SI EGAL? :L [] [RENDS []]
SI NON LISTE? :L [
    à force de décortiquer une liste, ce n'est plus une liste
   SI EGAL?  :L :E1 [RENDS :E2 ] []
   SI EGAL?  :L :E2 [RENDS :E1]
                    [RENDS :L ] ]
RENDS MP  ECHANGE :E1
                  :E2
                  PREM :L
          ECHANGE :E1
                  :E2
                  SP :L
FIN
? DONNE "L1 [[A B C] [F E [A B] ] A]
? ECRIS ECHANGE "A "B :L1
```

Si la liste contient des sous-listes, l'échange sera fait à tous les niveaux.

Application:
on peut décrire une figure par une liste, et ainsi cette figure sera en quelque sorte un objet LOGO :

```
DONNE "FIG1 [AV 10 TD 50 AV 20 TD 90 AV 100]
      (on s'autorise AV RE TD TG LC BC REPETE)
? EXEC :FIG1   (? EXECUTE :FIG1 sur PC )
     dessine la figure sur l'écran
? EXEC ECHANGE "TG "TD :FIG1
```

dessine la figure symétrique de la figure par rapport à la direction initiale de la tortue (à condition qu'après le dessin de FIG1 la tortue soit remise à son cap et sa position précédents).

EXEMPLE 9

LOGO peut concurrencer BASIC pour bricoler rapidement un petit programme simple. Essayons un traceur de courbes

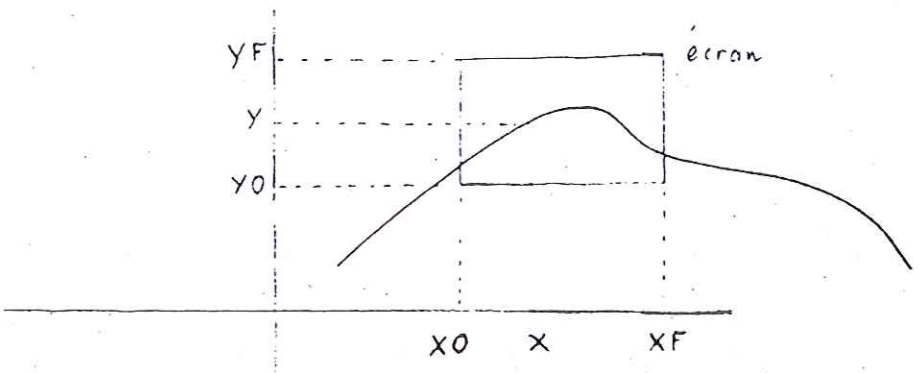
```
POUR F :T        définition de la fonction à représenter
RENDS :T * :T - 5
FIN

DONNE "ECHELLEX 1        pour ajuster la courbe sur l'écran
DONNE "ECHELLEY 0.01

POUR GRAPH
VE
DONNE "X -100
REPETE 200 [POINT LISTE :ECHELLE * :X
                        :ECHELLE * F :X
            DONNE  "X :X + 1]
FIN
? GRAPH
```

Cette procédure est trop fruste pour être vraiment utile. On peut faire mieux :

EXEMPLE 10

Traceur de courbes

- le nom de la fonction à représenter sera passé en paramètre
- l'écran sera une fenêtre sur un plan virtuel où la courbe est représentée.
- pas d'effacement préalable pour permettre plusieurs courbes



```
POUR F1 :X        les fonctions à représenter
RENDS 2 * :X + 3
FIN

POUR F2 :X
RENDS :X * :X - 1
FIN


POUR GRAPH :FONC :X0 :XF :Y0 :YF
DONNE "XE0 -159          en points écran
DONNE "XEF  159           "   "      "
DONNE "YE0 -99            "   "      "
DONNE "YEF  99            "   "      "
DONNE "A (:XEF - :XE0) / (:XF - :X0)
                              pour changement de repère horizontal
DONNE "B :XE0 - (:A * :X0)
DONNE "PAS 1 / :A          écart entre deux X du plan virtuel
DONNE "A1 (:YEF - :YE0) / (:YF - :Y0)
                              pour changement de repère vertical
DONNE "B1 :YE0 - (:A1 * :Y0)
DONNE "X :X0
REPETE (:XEF  - :XE0)                    un point par colonne de l'écran
          [DONNE "XE :A * :X + :B
  (*)     DONNE "YE :A1 * ( EXEC LISTE :FONC :X) + :B1
           SI  NON  (OU  (:YE  <  :YE0)  (:YE  >  :YEF))
                          afficher le point seulement s'il est sur l'écran
               [POINT LISTE :XE :YE ] []      (*)
           DONNE "X :X + :PAS ]
FIN
? GRAPH "F1 -10 10 -20 20
```

(*) voir paragraphe suivant

Les coordonnées sur l'écran sont des fonctions affines des coordonnées dans le plan virtuel où la courbe est dessinée
XE=A*X + B et YE=A1*Y + B1 il faut ajuster ces fonctions dans le programme pour que la courbe soit sur l'écran.

L'expression (EXEC LISTE :FONC :X) peut paraître mystérieuse:

:FONC ---) F1

:X ---) -10

LISTE :FONC :X ---) [F1 -10]

EXEC [F1 -10] équivaut à ? F1 -10 ,c'est à dire calcule et rend la valeur de F1 pour la valeur -10 du paramètre.

On peut éventuellement représenter des composées en définissant d'abord

```
POUR F1OF2 :X
F1 F2 :X
FIN
```

Il est facile si on le souhaite de rajouter des axes et des vecteurs unitaires.

Cette procédure a encore un défaut: si la fonction n'est pas partout définie sur l'intervalle d'étude il peut y avoir arrêt de l'exécution avec un message d'erreur . Un mécanisme est prévu pour y remédier sur PC:

Encadrer par

ATTRAPE "ERREUR [(*) ... (*)] les instructions repérées par une étoile où l'erreur est susceptible de se produire. Le message d'erreur est bloqué et le programme continue à l'instruction suivante.

On peut alors représenter des fonctions primitives ou personnelles :

? GRAPH "RC -1 5 -1 2 (représenter racine carrée entre -1 et 5)

EXEMPLE 11

Calculer la valeur à l'ordre n de la série (par une méthode itérative)

$1 + \frac{x}{1} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}$ pour une valeur de x choisie

```
POUR EX :X :N     exponentielle
LOCALE "TGEN      ( (LOCALE "TGEN "RES "N1) sur PC )
LOCALE "RES
LOCALE "N1
DONNE "TGEN  1
DONNE "N1 0
REPETE :N [
          DONNE "N1 :N1 + 1
          DONNE "TGEN (:TGEN * :X) / :N1
          DONNE "RES :RES + :TGEN ]
RENDS :RES
FIN
? EC EX 1 10   (on obtient une valeur approchée de e )
```

Il ne serait pas adroit de recalculer complètement le terme général à chaque tour de boucle. Il vaut mieux profiter de la valeur précédente.

exercice : programmer le calcul de la série

$$\sum_{k=1}^{k=n} \frac{x^k}{k}$$

qui diverge lorsque x=1 et n -> ∞ et admirer la lenteur de la croissance.

EXERCICE 12

Plus difficile: le TRI RAPIDE qui est l'un des meilleurs algorithmes de tri connus.

Pour trier une liste comme [4 7 5 8 10 5 1 4 2], en utilisant 4, séparons en deux le reste de la liste :[1 4 2] elts inférieurs ou égaux à 4

[7 5 8 10 5] elts supérieurs à 4

Si par un heureux hasard, ces deux listes étaient triées, il suffirait de les recoller en intercalant 4 entre les deux. Ce n'est pas le cas, eh bien, appliquons la même procédure (récursivement) à ces deux listes. Et ça marche!

```
POUR TRIRAP :L
SI EGAL? :L [] [RENDS  [] ]
RENDS PH PH TRIRAP PREM SEPARE PREM :L
                                SP :L
                                []
                                []
          PREM :L
     TRIRAP DER SEPARE PREM :L
                           SP :L
                           []
                           []
FIN
```

noter les 2 PH (PHRASE), en effet PH réclame seulement 2 paramètres

```
POUR SEPARE :E  :L  :LI  :LS
   sépare la liste de nombres L en deux à l'aide de l'élément E.
   LI et LS, vides au départ, contiendront après le balayage de L
   respectivement les éléments de L inférieurs ou égaux à E et les
   éléments de L supérieurs à E.
   rend la liste formée de LI et LS.
SI EGAL? :L [] [RENDS LISTE :LI :LS]
SI PREM :L > :E
   [RENDS SEPARE :E
                 SP :L
                 :LI
                 MP PREM :L
                    :LS]
   [RENDS SEPARE :E
                 SP :L
                 MP PREM :L
                    :LI
                 :LS ]
FIN
? DONNE "L2 [ 5 2 4 6 20 66 25 10 12 1 4 5 5 8 14 20 ]
? EC TRIRAP :L2
```

En mettant un compteur à l'entrée de SEPARE, on peut constater que celle-ci est appelée 126 fois pour trier L2.
Si on observe TRIRAP, on voit que SEPARE PREM :L SP :L [] [] est calculé deux fois, on peut "factoriser" ce calcul:

```
POUR TRIRAP2 :L
LOCALE "AUX
SI EGAL? :L [] [RENDS [] ]
DONNE "AUX SEPARE PREM :L
                  SP :L
                  []
                  []
RENDS PH PH TRIRAP2 PREM :AUX
            PREM :L
         TRIRAP2 DER :AUX
FIN
```

EXERCICE 13

Le PARCOURS D'UN ARBRE sera le prétexte pour aborder *la programmation descendante*. Voici le problème: étant donné un arbre, visiter sa racine puis toute la descendance de son premier fils (qui est elle même un arbre qui sera parcouru selon la même règle) avant de visiter toute la descendance des autres fils dans l'ordre naturel. C'est le classique parcours en *profondeur d'abord* (il en existe d'autres). Pour corser un peu le travail, nous mémoriserons les noeuds visités.

Remarquons que dans les illustrations, les arbres informatiques poussent à l'envers.

Raisonnons de façon générale, il nous suffit de connaître la racine, si elle a des fils, l'arbre de filiation de son premier fils, et les arbres de filiation des autres fils. "Diviser pour régner" est une maxime féconde aussi en informatique.

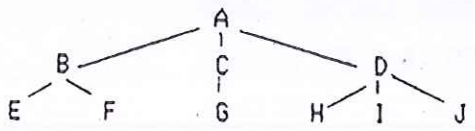
Comment l'arbre est vraiment représenté, comment sont écrites les fonctions RACINE, RAC_A_FILS?, PREMFILIAT, RESTEFILIAT qui dépendent de cette représentation, nous indiffère à ce stade de l'analyse. Ce sont là tâches vulgaires et subalternes qui pourraient être déléguées. D'ailleurs n'est-il pas mieux vu de dire ce qu'il faut faire, que de le faire soi-même ...

Voici les fonctions de haut niveau:

```
POUR EXPLORE :ARBRE
          la fonction principale. rend la liste des noeuds de
          l'arbre rencontrés
SI NON RAC_A_FILS? :ARBRE
           [RENDS LISTE RACINE :ARBRE]
RENDS (PH MP RACINE :ARBRE
             EXPLORE PREMFILIAT :ARBRE
          EXPLISTE RESTEFILIAT :ARBRE)
FIN

POUR EXPLISTE :L
RENDS EXPLISTEAUX :L []
FIN

POUR EXPLISTEAUX :L :CUMUL
SI VIDE? :L [RENDS :CUMUL]
RENDS EXPLISTEAUX  SP :L
                   PH :CUMUL
                      EXPLORE PREM :L
FIN
```

Maintenant, voici l'implantation et les fonctions de bas niveau:
Une première façon de faire est d'utiliser une liste de propriétés associée à chaque noeud de l'arbre. Avec ce système, chaque noeud possède les informations suffisantes pour progresser: l'arbre sera défini par sa racine.

```
DPROP "A "FILS [B C D]      A a pour fils  B, C, D
DPROP "B "FILS [E F ]
DPROP "C "FILS [G]
DPROP "D "FILS [H I J]
DPROP "E "FILS []
DPROP "F "FILS []
DPROP "G "FILS []
DPROP "H "FILS []
DPROP "I "FILS []
DPROP "J "FILS []
```




```
POUR RAC_A_FILS? :ARBRE     teste si la racine a des fils
SI VIDE? RPROP :ARBRE "FILS
   [RENDS "FAUX]
   [RENDS "VRAI]
FIN

POUR PREMFILIAT :ARBRE          rend le premier fils et sa filiation
RENDS PREM RPROP :ARBRE "FILS
FIN

POUR RESTEFILIAT :ARBRE         rend la liste des arbres de filiation
                                des autres fils
RENDS SP RPROP :ARBRE "FILS
FIN

POUR RACINE :ARBRE
RENDS :ARBRE
FIN
? EXPLORE "A
```

ou bien

```
? DONNE "A1 "A    A est la racine de l'arbre A1
? EXPLORE :A1
```

ceci si on tient à distinguer l'arbre de sa racine, et pour respecter l'unité avec ce qui suit.

Deuxième représentation classique: par une liste, chaque noeud étant suivi des listes représentant les arbres des descendants:

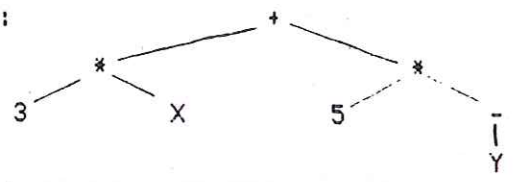
```
DONNE "A1 [A [B [E] [F] ]
             [C [G ] ]
             [D [H] [I]] [J] ]
```

L'indentation, nullement nécessaire, n'est là que pour faciliter la lecture.

```
POUR RACINE :ARBRE
RENDS PREM :ARBRE
FIN

POUR RAC_A_FILS? :ARBRE
SI VIDE? SP :ARBRE
   [RENDS "FAUX]
   [RENDS "VRAI]
FIN

POUR PREMFILIAT :ARBRE
RENDS PREM SP :ARBRE
FIN

POUR RESTEFILIAT :ARBRE
RENDS SP SP :ARBRE
FIN

? EXPLORE :A1
```

Exercice: critiquer et comparer ces deux choix de représentation.

A quoi celà peut-il bien servir? d'abord, on peut s'intéresser aux arbres pour eux-mêmes et les prendre pour objet d'étude. Mais il existe aussi des application "pratiques". Si nous voulons faire faire du calcul formel à l'ordinateur, il est d'usage de représenter une expression algébrique comme 3X + 5(-Y) par un arbre:



(la multiplication est notée * pour éviter la confusion avec X)

On peut souhaiter mettre cette expression en notation préfixée: [+ * 3 X * 5 - Y] . Celle-ci est très efficace sous l'angle de l'occupation de la mémoire puisque les parenthèses sont inutiles. Le programme que nous venons d'écrire peut se charger de la transformation.

Cette façon de procéder, la démarche descendante, a beaucoup amélioré la productivité et aussi la fiabilité des programmes. La recherche des inévitables erreurs se trouve aussi grandement facilitée.

Un mot sur la recherche des erreurs. Il est souvent utile de faire une *trace* des appels de fonctions et procédures. Pour cela faire imprimer, dès l'entrée ou à des endroits bien choisis, le nom de la procédure et les valeurs de certains ou tous les paramètres. On peut ainsi suivre leur évolution, ce qui fournit souvent la clé du problème.

EXERCICE 14

DOMINOS

Maintenant, laissons l'ordinateur jouer seul aux dominos. Donnons lui une liste de dominos (RESERVE) et une liste de un ou plusieurs dominos déjà posés (POSE) selon la règle habituelle. Et laissons le trouver une solution pour les jouer.

C'est un problème du type à essais successifs (ou retour arrière ou BACKTRACK). Si l'idée est simple , la réalisation est assez complexe à suivre. C'est cependant un type extrêmement important, de très nonbreux problèmes relevant de cette stratégie.

Utilisons une liste L qui contient les candidats potentiels à être posés et essayons logiquement le premier de la liste, 3 possibilités:

ou il est posable et permet de poursuivre jusqu'au bout, tout va bien: on tient une solution

ou il est posable, mais ce choix conduit plus tard à une impasse (il reste des éléments dans RESERVE impossibles à placer)

ou il ne convient pas.

Il est évident que dans les deux derniers cas, on repart avec le candidat suivant. Mais la grosse difficultés est de tester le résultat d'un calcul qui n'a pas encore eu lieu, ce qui parait étrange. Une solution est de recevoir dans une variable (SOL) la conséquence du choix et de ne prendre en compte SOL que si le processus a pu se poursuivre jusqu'au bout, sinon on revient en arrière (backtrack) et on repart en essayant la valeur suivante.

Le processus étant hautement récursif, il est assez difficile à suivre. Il importe de ne pas perdre de vue

- que l'ordre d'exécution des instructions n'est pas l'ordre d'écriture
- que des instructions sont laissées en attente pour être exécutées plus tard.
- que RENDS renvoie quelque chose à la procédure qui a appelé mais n'interrompt pas le déroulement.

Le mieux est de suivre patiemment, jusqu'au bout, l'arbre des appels successifs sur un exemple simple, par exemple:

```
? DOMINO [[5 1]]                    1er paramètre: les dominos déjà posés
         [[6 1] [4 6] [1 1] ]       2eme paramètre: la reserve
(DOMINO) appelée 11 fois pour trouver une solution
[[5 1] [1 1] [1 6] [6 6] ]
```

```
POUR DOMINO :JOUE :RESERVE                amorce le processus
DONNE "RES   DOMINO1 :JOUE :RESERVE :RESERVE
SI EGAL? :RES []
  [EC [PAS DE SOLUTION ] ]
  [TAPE [SOLUTION  :] EC :RES]
FIN

POUR DOMINO1 :JOUE :RESERVE :L
LOCALE "SOL
LOCALE "J
EC [DOMINO1]                              |       instructions inutiles :
EC :JOUE                                  |       trace
EC :RESERVE                               |
EC :L                                     |
SI VIDE? :RESERVE                 a-t-on posé tous les dominos ?
   [ RENDS :JOUE ]                ici sortie au dernier appel
   [SI VIDE? :L                   a-t-on tout essayé ?
       [RENDS []]                 sorties intermédiaires en cas d'échecs
       [SI CONDITION?
           [RENDS :SOL     ]            sorties intermédiaires
           [RENDS DOMINO1 :JOUE              mauvais choix :
                  :RESERVE                   on essaie le domino suivant
                  SP :L      ] ]]
FIN

POUR  CONDITION?        le choix du premier candidat est-il pertinent ?
DONNE "J JOUABLE? :JOUE :L
SI :J
   [DONNE "SOL DOMINO1 POSE  PREM :L :JOUE        on joue un domino
                       SUPPRIME PREM :L :RESERVE
                       SUPPRIME PREM :L :RESERVE]
SI NON :J
   [RENDS "FAUX]
   [SI VIDE? :SOL
       [RENDS "FAUX]
       [RENDS "VRAI]]
FIN
```

```
POUR JOUABLE? :J :L              le premier domino de L peut-il prolonger
RENDS (OU                        un des deux boute de J ?
          EGAL? PREM PREM :J   PREM PREM :L
          EGAL? PREM PREM :J   DER  PREM :L
          EGAL? DER DER :J     PREM PREM :L
          EGAL? DER DER :J     DER  PREM :L)
FIN

POUR SUPPRIME :EL :L          supprime l'élément EL de la liste L
SI VIDE? :L
   [RENDS []]
SI EGAL? :EL PREM :L
   [RENDS SUPPRIME :EL SP :L]
   [RENDS MP PREM :L
             SUPPRIME :EL SP :L]
FIN

POUR POSE :DOM :J                ajoute un domino jouable à la liste J
SI EGAL? PREM PREM :J    PREM :DOM
   [RENDS MP RETOURNE :DOM   :J]
SI EGAL? PREM PREM :J    DER :DOM
   [RENDS MP  :DOM   :J]
SI EGAL? DER DER :J      PREM :DOM
   [RENDS MD  :DOM   :J]
SI EGAL? DER DER :J      DER  :DOM
   [RENDS MD RETOURNE :DOM  :J]
FIN


POUR RETOURNE :DOM          retourne un domino
RENDS LISTE DER :DOM
            PREM :DOM
FIN

DONNE "J1 [[1 3]]
DONNE "L1 [[5 2][6 1][2 2][1 2][4 3]]      (pas de solution)
DONNE "L2 [[2 4][6 3][1 6][2 2][4 1][1 5]]
? DOMINO :J1 :L2
```

Il est possible facilement d'obliger le programme à écrire toutes les solutions. Il suffit d'ôter une paire de crochets ! Comprendre pourquoi demande du temps et un bon cachet d'aspirine.
Le comportement du programme a changé, celui-ci ne s'interrompt plus dès qu'il a trouvé une solution, il faut la faire imprimer au passage.
Cette nouvelle fonction retourne toujours la liste vide.

```
POUR DOMINO :JOUE :RESERVE          donne toutes les solutions
EC DOMINO1 :JOUE :RESERVE :RESERVE
FIN

POUR DOMINO1 :JOUE :RESERVE :L
LOCALE "SOL
LOCALE "J
SI VIDE? :RESERVE
   [ EC :JOUE RENDS :JOUE ]        impression d'une solution éventuelle
   [SI VIDE? :L
       [RENDS []]
       [SI CONDITION?
           [RENDS :SOL     ]
        RENDS DOMINO1 :JOUE
                          le déroulement continue, qu'une solution
                     :RESERVE                ait été trouvée ou non
                     SP :L       ]]
FIN
? DOMINO :J1 :L2  (beaucoup de solutions)
```

TROISIEME PARTIE

Flocons et dragons

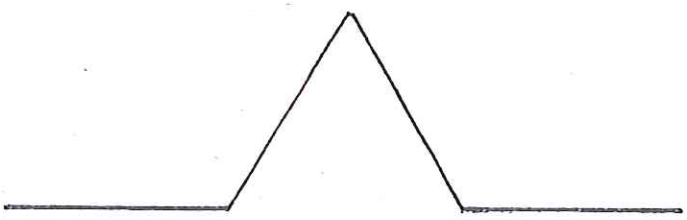
ou

de l'art de la récursivité en LOGO

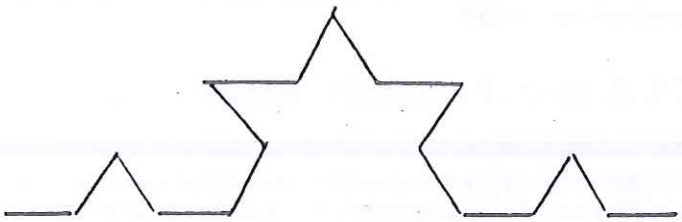
Chacun sait que l'atout maître de LOGO c'est le DESSIN: la petite tortue avance, tourne et recule au gré de l'inspiration du programmeur, petit ou grand (plus souvent petit, d'ailleurs); ce que l'on sait peut-être moins bien, c'est que LOGO, comme tout langage récursif, se prête particulièrement bien (c'est à dire en quelques lignes de programme) à de nombreux dessins curieux et souvent élégants. Je veux parler plus précisément des dessins " en arborescence " et des courbes fractales. C'est Benoit Mendelbrot qui a - dans les années 77 - lancé l'étude des courbes fractales.

Une courbe fractale est une courbe dont la forme générale est la même quel que soit le grossissement sous lequel on l'observe. Par exemple: vous observez une chaîne de montagnes à l'horizon, elle est constituée d'une suite de PICS. Si vous vous rapprochez de cette chaîne, vous constatez que chaque pic est constitué d'une série de PICS PLUS PETITS. De tout près, le sol inégal comporte, lui aussi de nombreux POINTES et CREUX. Un autre exemple souvent cité est celui de la côte Bretonne dont personne ne peut calculer la longueur, tant elle est déchiquetée, chaque BAIE étant formée de MULTIPLES PETITES BAIES et ainsi de suite.

Comment dessiner une courbe fractale pour, ensuite, la programmer en LOGO? Une méthode parmi d'autres est la suivante: on trace un dessin de base, par exemple une ligne brisée:



Puis sur chaque trait de la ligne brisée on redessine le dessin de base (en plus petit naturellement):



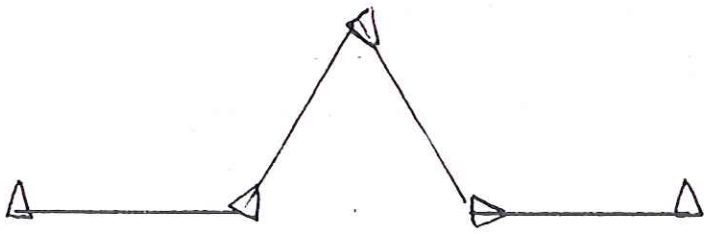
et ainsi de suite...

La courbe fractale ainsi obtenue s'appelle courbe de VON-KOCH sa programmation en est très élégante mais un peu subtile aussi nous allons la détailler.

L'écriture du dessin de base est la suivante (en LOGOPLUS):

```
Pour von-koch :nombre :longueur
tournedroite 90 avance :longueur
tournegauche 60 avance :longueur
tournedroite 120 avance :longueur
tournegauche 60 avance :longueur
tournegauche 90
fin
```

(rappelons que ces termes LOGO se notent en abrégé respectivement TD, AV TG). Nombre est un compteur du nombre de dessins de base il vaut ici 1 et longueur est la longueur du segment élémentaire. Le cap de la tortue à l'origine étant NORD il faut tourner à droite de 90°, en fin de dessin elle est EST il faut donc tourner à gauche de 90° pour remettre la tortue en position d'origine.



On veut ensuite décomposer chaque segment en un motif de base mais le segment élémentaire sera, en gros, 3 fois PLUS PETIT; on va donc remplacer AV :longueur par VON-KOCH :nombre-1 :longueur/3 la valeur du paramètre nombre va servir à arrêter le processus à la demande. Que devient l'instruction AV :longueur ? Supposons que nous voulions exécuter VON-KOCH :1 :30 (c'est notre dessin de base) on va avoir la suite d'instructions suivante:

```
td 90 von-koch 0 10
tg 60 von-koch 0 10
td 120 von-koch 0 10
tg 60 von-koch 0 10
tg 90
fin
```

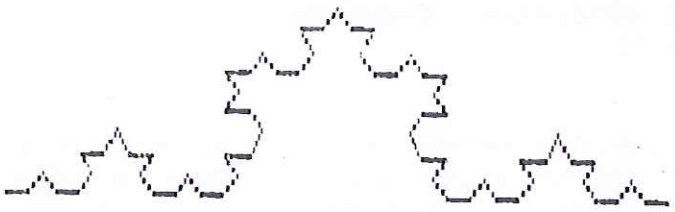
(remarquons que ces différents programmes, écrits sous LOGOPLUS, apparaitrons AUTOMATIQUEMENT en MAJUSCULES)

Pour retrouver le dessin de base il suffit donc que Von-koch 0 10 fasse simplement avancer la tortue de 10 et ARRETE la RECURSIVITE. On résout ce problème en ajoutant à la procédure la ligne suivante:
Si :nombre=0 [AV :longueur STOP]

La procédure complète devient alors:

```
Pour von-koch :nombre :longueur
si :nombre=0 [av :longueur stop]
td 90 von-kock :nombre-1 :longueur/3
tg 60 von-koch :nombre-1 :longueur/3
td 120 von-koch :nombre-1 :longueur/3
tg 60 von-koch :nombre-1 :longueur/3
tg 90
fin
```

Ce petit programme ne dessine pas le fort joli et connu FLOCON mais une ligne de plus en plus brisée lorsque la valeur entrée pour "nombre" augmente:



Von-koch 3 250

Pour obtenir un flocon on effectue 3 fois von-koch avec, à chaque fois, une rotation de 120°

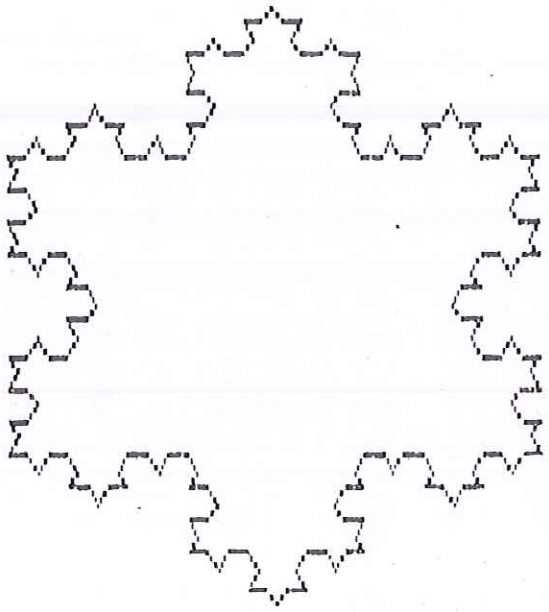
```
Pour flocon :nombre :longueur
origine              ( pour nettoyer l'écran et mettre la tortue )  *)
nettoie                        ( en position d'origine )
répète 3 [von-koch :nombre :longueur td 120]
fin
```

Remarque:
A chaque appel récursif de von-koch le paramètre longueur est divisé par 3, donc PLUS le paramètre nombre (qui compte les appels récursifs) est élevé, MOINS le dessin de base est grand et PLUS la figure est compliquée - et jolie ! -

(*) ou vidécran

Voir le dessin page suivante:

FLOCON 3 200



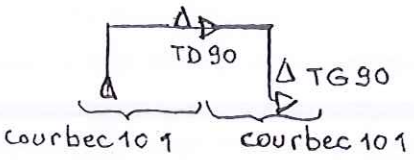
Une autre fractale bien connue et de programmation particulièrement élégante est la "courbe en C", son programme est le suivant:

```
Pour courbec :longueur :nombre
Si :nombre=0 [av :longueur stop]
courbec :longueur :nombre-1
td 90
courbec :longueur :nombre-1
tg 90
fin
```

On voit que l'idée de base est - à peu près - la même que celle du flocon: indexation sur un compteur, dessin seulement quand le compteur est à 0, par contre la longueur est TOUJOURS la MEME donc la dimension du dessin augmente avec la valeur du paramètre "nombre". Faisons tourner le programme "à la main" (attention à la position de la tortue!)

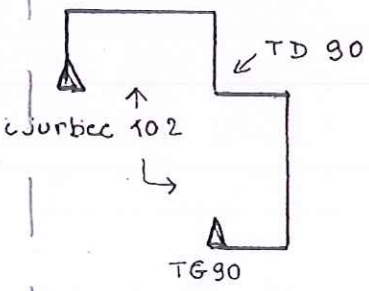
courbec 10 1

courbec 10 2

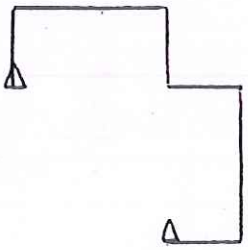


qui donne:

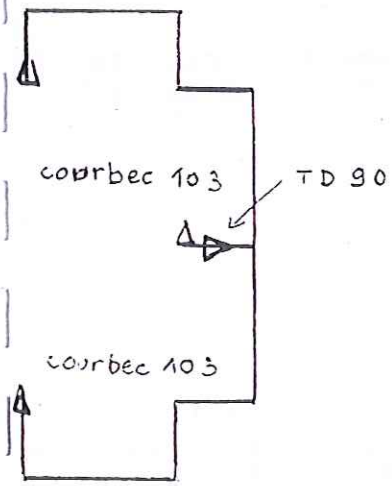
courbec 10 3



qui donne:



courbec 10 4



on voit que le dessin est parfois inattendu!
(voir courbec 5 10 annexe 1)

Enfin, il est impossible de ne pas citer le fameux DRAGON qui fait appel à deux procédures récursives imbriquées, en effet le programme comporte deux procédures dragondroit et dragongauche qui s'appellent mutuellement, il y a deux dessins de base:

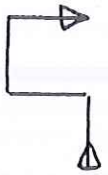
dragongauche 10 1

dragondroit 10 1

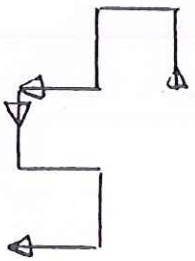
qui donnent, par exemple:

dragongauche 10 2

dragondroit 10 2

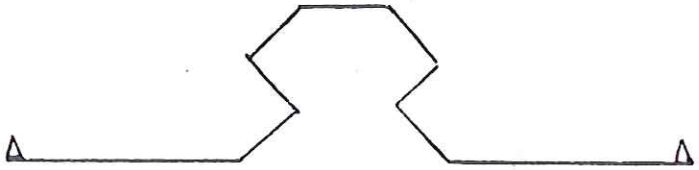


et dragongauche 10 3



On voit déjà poindre (vaguement!) l'ombre inquiétante du dragon qui se dessine plus joliment dans dragongauche 6 11.(annexe 2)

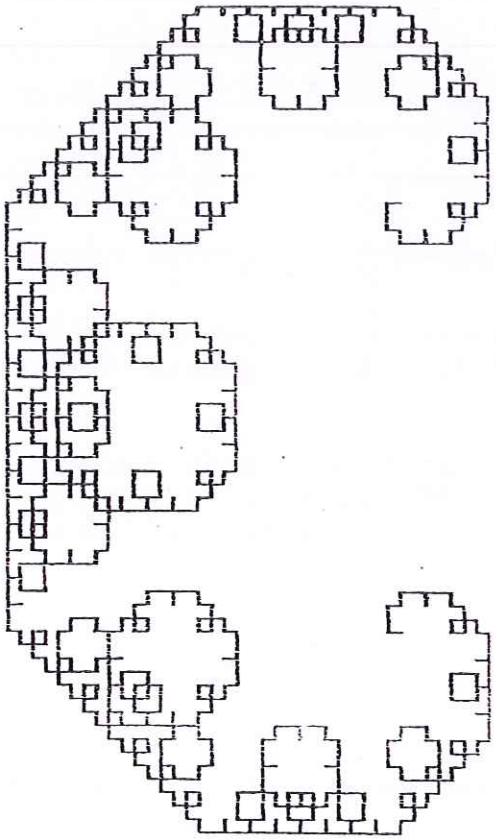
J'espère que ces quelques lignes vont vous donner l'envie de vous mettre "à vos pinceaux" ou plus efficacement à vos ordinateurs. A titre d'exemple voici ce que peut donner un dessin de base relativement simple:



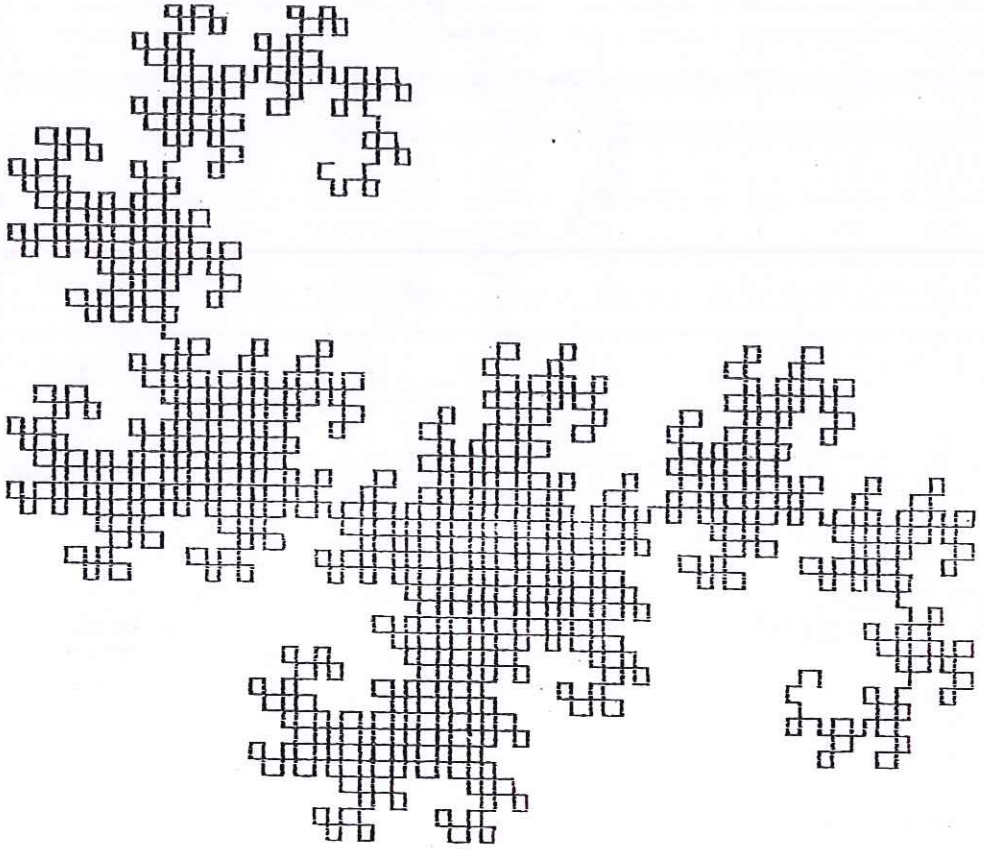
cote 9 1

Exécuté sur deux niveaux, puis inséré dans deux procédures, l'une destinée à le dessiner successivement dans deux sens différents, l'autre à effectuer des rotations comme dans le cas du dragon.(annexe 3)

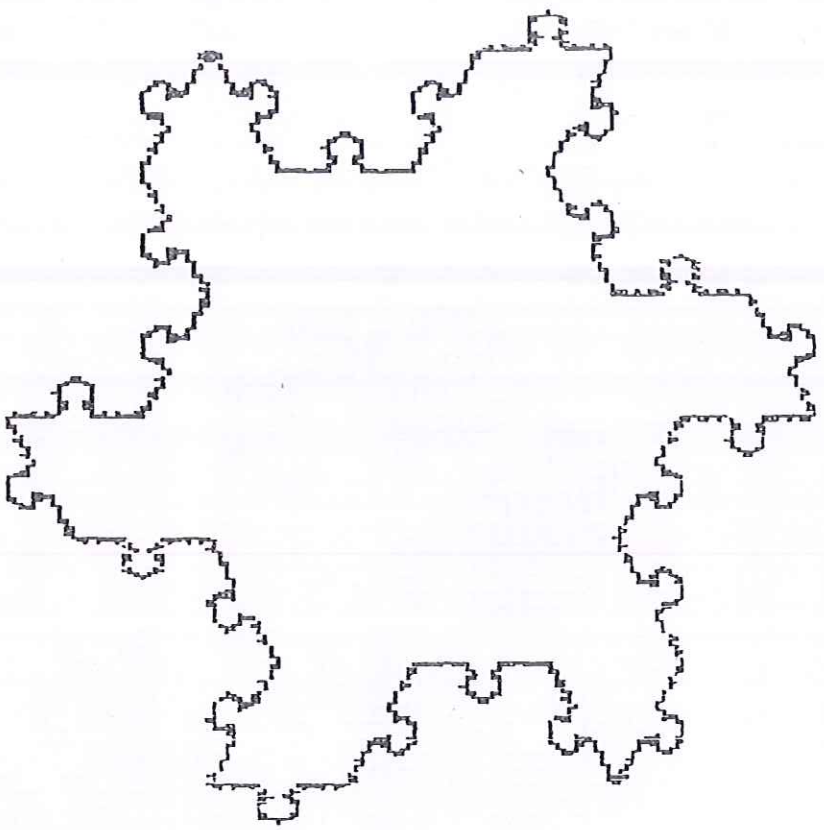
A vous d'inventer une petite fractale, si possible jolie, le concours est ouvert.



COURBEC 5 10



DRAGON à 11



motif

```
POUR COTE :DIM :NIV
SI :NIV = 0 [AV :DIM STOP]
TD 90
COTE :DIM / 9 :NIV - 1
COTE :DIM / 9 :NIV - 1
COTE :DIM / 9 :NIV - 1
TG 45
COTE :DIM / 9 :NIV - 1
TG 90
COTE :DIM / 9 :NIV - 1
TD 90
COTE :DIM / 9 :NIV - 1
TD 45
COTE :DIM / 9 :NIV - 1
TD 45
COTE :DIM / 9 :NIV - 1
TD 90
COTE :DIM / 9 :NIV - 1
TG 90
COTE :DIM / 9 :NIV - 1
TG 45
COTE :DIM / 9 :NIV - 1
COTE :DIM / 9 :NIV - 1

COTE :DIM / 9 :NIV - 1
TG 90
FIN
POUR ELEMENT :DIM :NIV
REPETE 2 [COTE :DIM :NIV TG 60]
REPETE 2 [COTE :DIM :NIV TD 60]
FIN
POUR MOTIF
ORIGINE NETTOIE
LC AV 100 BC
REPETE 6 [ELEMENT 60 2 TD 60]
FIN
```

JOUER SUR LES MOTS AVEC LOGO

Vous connaissez sans doute le principe du VERLAN: pour écrire un mot en VERLAN, il faut découper le mot en syllabes et ré-écrire celles-ci dans l'ordre inverse, ainsi "CHARLEMAGNE" devra être découpé en "CHAR LE MA GNE" et écrit "GNEMALECHAR"; pour écrire une phrase complète en VERLAN, on écrit successivement tous les mots ainsi transformés en respectant leur ordre, ainsi "TU ECRIS VERLAN" deviendra: "TU CRISE LANVER". Ecrire en VERLAN nécessite donc une bonne connaissance du concept de SYLLABE.

Définition du Grand Larousse: Syllabe: Voyelle ou réunion de lettres qui se prononcent d'une seule émission de voix: le mot "Paris a deux syllabes. La syllabe est donc un concept PHONETIQUE. Ecrire en VERLAN peut donc conduire à une activité intéressante puisque il faut "faire le tour" des différentes syllabes du Français. Le programme LOGO qui vous est proposé ici écrit en verlan une phrase entrée au clavier.

Après consultation du "BECHERELLE: l'orthographe pour tous", très orienté sur la phonétique, donc assez bien adapté à notre problème, on peut choisir pour définir une syllabe l'"axiome" suivant:
Une syllabe est un groupe de lettres comportant:
une voyelle OU deux voyelles adjacentes (remarquons que 3 voyelles adjacentes sont généralement prononcées en 2 fois: oui en ou-i, jouer en jou-er. Ceci ne suffit évidemment pas à définir la syllabe mais permet d'en limiter au mieux la longueur; ainsi "achat" comporte deux syllabes puisque ce mot a deux voyelles non adjacentes.

La programmation est entièrement récursive ainsi, pour une phrase donnée, on traite le premier mot de la phrase, puis le reste de la phrase et on recommence. On peut donc traiter tout d'abord le cas ou la phrase est réduite à un mot.

La technique est la suivante: on va COMPTER les lettres de la lière syllabe. Pour cela on avance dans le mot jusqu'à ce que l'on trouve une voyelle, on est alors sûr d'avoir le DEBUT de la SYLLABE (ex: dans chat, on avance jusqu'à a). En même temps on met dans un COMPTEUR le nombre de lettres que l'on a fait défiler.

On étudie alors le mot restant, il commence donc nécessairement par une voyelle suivie d'une consonne (si la voyelle est suivie d'un blanc on arrête le travail) ou par une voyelle suivie de "y" (lorsque le "y" est suivi d'une voyelle il joue, phonétiquement, le rôle d'une consonne) ou par 2 voyelles.

On traite alors les CAS PARTICULIERS:

On est en FIN de MOT et:

- le mot comporte une lettre (la voyelle) on ajoute 1 au compteur.
- le mot comporte 2 lettres, on ajoute 2 au compteur. (ex :on)
- le mot comporte 3 lettres: la voyelle et 2 consonnes, on ajoute 3 au compteur. (ex :ent dans le mot dent)
- le mot comporte 4 lettres: la voyelle et 3 consonnes, on ajoute 4 au compteur. (ex :ents dans le mot agents)

On est en DEBUT ou en MILIEU de MOT et:

- la voyelle est suivie de trois consonnes dont aucune n'est doublée (c'est le cas de plusieurs mots d'origine étrangère: cockpit, angström et de certains mots français: vingtaine, symptôme, constat...), on ajoute 3 au compteur pour couper après la 2ième consonne (ex: symp-tôme).
- la voyelle est suivie de "y" puis d'une autre voyelle (ex: oyat) on ajoute 1 au compteur (ce qui donne o-yat).
- la voyelle est suivie de 2 voyelles (ex: oui) on ajoute 2 au compteur (ce qui donne ou-i)
- les consonnes qui suivent la voyelle sont gn, on ajoute 1 au compteur. (ex: ègn dans règne)
- la 2ième consonne qui suit la voyelle est "l","r", "u" ou "h" et n'est pas doublée, on ajoute 1 au compteur. (ex :abl dans table,igr dans aigre,aqu dans laque, ach dans achat)

On traite ensuite les deux cas les PLUS GENERAUX:

- la voyelle est suivie de 2 consonnes mais on n'est pas dans les cas précédents, on ajoute 2 au compteur. (ex : ass dans assure, osc dans tocsin)
- la voyelle est suivie d'une consonne puis d'une voyelle, on ajoute 1 au compteur. (ex :ite dans suite)

C'est la fonction NBRELET :mot qui effectue ce comptage des lettres de la 1ière syllabe du mot. Puisque c'est une fonction (elle "rends" un nombre), on en sort dès que l'on a trouvé le cas concerné, elle doit donc aller du PARTICULIER au GENERAL (ou du cas le moins probable au cas le plus probable) . La fonction CONSTRUIS :mot :n construit la première syllabe de longueur n dans le mot. La fonction RESTEMO :mot :n enlève au mot la première syllabe de longueur n. On obtient alors, en utilisant récursivement ce procédé dans la fonction LISTESYL :mot la liste des syllabes du mot séparées par un blanc.

La fonction ENVERS :liste écrit la liste des syllabes du mot à l'envers, la fonction VERLAN :phrase effectue ce travail récursivement sur la totalité de la phrase.

Détaillons un peu le programme:

Il est présenté en PROGRAMMATION DESCENDANTE: pour écrire en VERLAN on écrit à l'envers la liste des syllabes du premier mot de la phrase puis on recommence sur la phrase diminuée de ce premier mot (fonction VERLAN :phrase).

Pour écrire une liste de syllabes à l'envers, on écrit la dernière syllabe de la liste puis on recommence sur le reste de la liste (fonction ENVERS :liste).

Pour écrire la liste des syllabes d'un mot on écrit une phrase constituée de la première syllabe du mot suivie d'un blanc et du reste du mot, puis on recommence sur le reste du mot (fonction LISTESYL :mot).

Pour trouver la première syllabe d'un mot on calcule le nombre de ses lettres (grâce à la fonction NBRELET :mot) on reconstitue la syllabe grâce à la fonction CONSTRUIS :mot :n (qui construit la première syllabe de longueur n dans le mot). La fonction RESTEMO :mot :n délivre le mot privé de sa première syllabe.

Enfin la fonction VOYELLE? :lettre rend VRAI si la lettre est une voyelle, FAUX sinon. Elle est utilisée dans NBRELET.

Vous trouverez dans ce qui suit des exemples d'exécution pour chacun des cas étudiés et le listing du programme. Vous remarquerez que la fonction NBRELET :mot, qui ne fait, en réalité qu'exprimer un suite de REGLES, est écrite de façon assez lourde afin de bien expliciter chacun des cas, vous pourrez, bien entendu, l'alléger.

Enfin, malgré de nombreuses corrections et en l'absence de documents "officiels" sur les syllabes, vous trouverez sûrement dans ce programme des oublis ou des erreurs, merci de les corriger.

Si ce genre d'activité vous intéresse, vous pouvez, en aménageant un peu le programme en écrire un autre qui écrit en JAVANAIS. Le principe est le suivant: pour chaque syllabe du mot, on stocke dans une variable tampon la syllabe privée de la ou des consonnes du début, on ajoute à la syllabe "d" suivi du contenu du tampon puis "gu" suivi du contenu du tampon. Ainsi "le chat" devient "ledegue chatdagua".

A vous de jouer!

QUELQUES EXEMPLES D'EXECUTION

```
?
?
?EC VERLAN [LES AGENTS SONT DE BRAVES GENS]
LES GENTSA SONT DE VESBRA GENS
?
?
?EC VERLAN [LE COCKPIT MESURE UNE VINGTAINE D'ANGSTROM]
LE PITCOCK RESUME NEU NETAIVING STROMD'ANG
?
?
?EC VERLAN [LE PRATICIEN CONSTATE LES SYMPTOMES]
LE CIENTIPRA TETACONS LES MESTOSYMP
?
?
?EC VERLAN [OUI IL AIME LES OYATS]
IOU IL MEAI LES YATSO
?
?
?EC VERLAN [LE CANARD LAQUE EST POSE SUR LA TABLE]
LE NARDCA QUELA EST SEPO SUR LA BLETA
?
?
?EC VERLAN [FAIT SONNER LE TOCSIN TOUT DE SUITE]
FAIT NERSON LE SINTOC TOUT DE TESUI
```

```
PROGRAMME POUR PARLER "VERLAN" PAR D. SALLES  MARS 90




POUR VERLAN :PHRASE
SI :PHRASE = [] [RENDS " ]
SI SP :PHRASE = [] [RENDS ENVERS LISTESYL PREM :PHRASE]
RENDS PHRASE (ENVERS LISTESYL PREM :PHRASE) (VERLAN SP :PHRASE)
FIN



POUR ENVERS :LISTE
SI :LISTE = [] [RENDS "]
SI SP :LISTE = " [RENDS :LISTE]
RENDS MOT DER :LISTE ENVERS SD :LISTE
FIN



POUR LISTESYL :MOT
SI :MOT = " [RENDS "]
DONNE "M NBRELET :MOT
RENDS PHRASE CONSTRUIS :MOT :M LISTESYL RESTEMO :MOT :M
FIN



POUR RESTEMO :MOT :N
SI :N = COMPTE :MOT [RENDS "]
SI :N =  0 [RENDS :MOT]
RENDS RESTEMO SP :MOT :N-1
FIN



POUR CONSTRUIS :MOT :N
SI :N =0 [RENDS "]
SI :MOT = " [RENDS "]
RENDS MOT PREM :MOT CONSTRUIS SP :MOT :N-1
FIN
```

```
POUR NBRELET :MOT


SI COMPTE :MOT = 0 [RENDS 0]
SI COMPTE :MOT = 1 [RENDS 1]
SI COMPTE :MOT = 2 [RENDS 2]

SI COMPTE :MOT = 3 [SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
                          (NON VOYELLE? ITEM 3 :MOT) [RENDS 3]]
SI COMPTE :MOT = 4 [SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
                          (ET NON VOYELLE? ITEM 3 :MOT
                              NON VOYELLE? ITEM 4 :MOT) [RENDS 4]]


SI COMPTE :MOT > 3
 [SI ET(ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
           (ET NON VOYELLE? ITEM 3 :MOT NON VOYELLE? ITEM 4 :MOT))
       (ET NON EGAL? ITEM 2 :MOT ITEM 3 :MOT NON EGAL? ITEM 3 :MOT ITEM 4 :MOT)
       [RENDS 3]]




SI ET (ET VOYELLE? PREM :MOT EGAL? ITEM 2 :MOT "Y)
      (VOYELLE? ITEM 3 :MOT) [RENDS 1]
SI ET (ET VOYELLE? PREM :MOT VOYELLE? ITEM 2 :MOT)
      (VOYELLE? ITEM 3 :MOT) [RENDS 2]
SI ET (ET VOYELLE? PREM :MOT EGAL? ITEM 2 :MOT "G)
      (EGAL? ITEM 3 :MOT "N) [RENDS 1]
SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
      (ET NON EGAL? ITEM 2 :MOT "L EGAL? ITEM 3 :MOT "L) [RENDS 1]
SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
      (ET NON EGAL? ITEM 2 :MOT "R EGAL? ITEM 3 :MOT "R) [RENDS 1]
SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
      (OU EGAL? ITEM 3 :MOT "U EGAL? ITEM 3 :MOT "H) [RENDS 1]


SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
      (NON VOYELLE? ITEM 3 :MOT) [RENDS 2]
SI ET (ET VOYELLE? PREM :MOT NON VOYELLE? ITEM 2 :MOT)
      (VOYELLE? ITEM 3 :MOT) [RENDS 1]

RENDS (NBRELET SP :MOT)+1
FIN



POUR VOYELLE? :LETTRE
SI :LETTRE = " [RENDS "FAUX]
SI MEMBRE? :LETTRE "AEIOUY [RENDS "VRAI][RENDS "FAUX]
FIN
```

DEVENEZ "EXPERT" AVEC LOGO

Mettre à la disposition du grand public les connaissances des "experts" est un rêve accessible mais difficile. Accessible car les spécialistes en Intelligence Artificielle (I.A pour les initiés) ont écrit depuis déjà longtemps d'excellents programmes capables d'exploiter les connaissances d'experts, celles-ci étant écrites sous formes de REGLES. Difficile car écrire ces programmes de telle sorte qu'ils soient utilisables par un non-informaticien et modifiable facilement par l'utilisateur compétent est tout un art! Malgré la technicité de l'écriture d'un générateur de système expert il est peut-être encore plus délicat de faire "mettre en forme" ses connaissances à l'expert lui-même.

Ecrire un générateur de système expert n'est pas trop difficile si on s'en tient à la technique la plus simple, c'est, de plus, un travail de programmation intéressant. Ecrire un système expert -dans un domaine qui vous intéresse et/ou vous êtes compétent- est une expérience à faire!

Voici le système le plus simple d'exploitation d'un ensemble de règles (on appelle cela un moteur d'inférence): le moteur zéro en chaînage avant. La technique est la suivante: l'expert écrit une suite de règles du genre: "si l'animal porte des piquants et est un mammifère alors c'est un hérisson", cette suite de règles, pour plus de simplicité, peut être écrite directement dans le programme. L'utilisateur entre au clavier un ensemble de FAITS concernant son problème, par exemple "l'animal est un mammifère", "l'animal porte des cornes" etc...Le programme parcourt alors l'ensemble des règles pour voir si une des règles s'applique (c'est à dire si les hypothèses de la règle sont vérifiées par les faits: dans notre exemple ce n'est pas le cas puisque l'animal n'a pas de piquants). Si une des règles s'applique le programme affiche la conclusion de la règle et AJOUTE cette conclusion à l'ensemble des faits puis il passe à la règle suivante. A la fin de l'exécution toutes les règles ont été parcourues et toutes les conclusions affichées. Afin de conserver une certaine souplesse dans l'écriture des règles, on parcourt celles-ci plusieurs fois afin de ne pas être obligé de les écrire dans un ordre logique. Le critère est le suivant: on parcourt les règles jusqu'à ce qu'on AJOUTE PLUS de fait NOUVEAU.

Ecrire un système expert est donc une activité qui a deux intérêts d'origine très différente: celui de la programmation en LOGO et celui de l'écriture des règles. L'intérêt informatique est un peu limité: ce programme n'est pas interactif et il faut écrire les faits rigoureusement sous la même forme que les hypothèses des règles; l'intérêt pédagogique du à l'écriture des règles me semble par contre beaucoup plus grand. Prenons un exemple: un professeur de dessin veut donner à ses élèves des connaissances en peinture, il peut leur présenter un certain nombre de reproductions d'oeuvres de peintres célèbres puis leur demander d'ECRIRE DES REGLES permettant de reconnaître ces oeuvres. Autrement dit ce sont LES ELEVES eux-mêmes qui deviennent les experts chargés d'apprendre la peinture à l'ordinateur.(à la limite on peut se passer de l'ordinateur mais cela diminue évidemment le charme de l'activité). Il n'est donc pas indispensable que les élèves sachent programmer, il suffit de savoir entrer dans le programme pour écrire les règles.

Détaillons maintenant le fonctionnement du programme:

- La procédure EXPERT sert à écrire la liste des règles, c'est donc cette procédure que vous aurez à adapter si vous voulez écrire votre PROPRE SYSTEME. LISTEREGLES est une variable globale qui est la liste de toutes les règles. Chacune des règles est une liste comprenant le mot "règle", le numéro de la règle puis son énoncé, attention, s'il y a plusieurs conditions à la règle, elles ne sont pas séparées par un "et", celui-ci est sous-entendu; chaque règle n'a qu'une conclusion. L'ordre des règles n'a pas d'importance même si la règle numéro n utilise la conclusion de la règle n+1 puisque l'ensemble des règles est parcouru plusieurs fois.

- Les procédures HYPOTHESE et CONCLUSION extraient du texte de la règle ses hypothèses et sa conclusion.

- La fonction à valeur booléenne TESTHYP teste, pour un ensemble de faits rentrés au clavier (voir plus loin la procédure SYSTEXP), si les hypothèses de la règle font partie de ces faits (autrement dit PEUT-ON APPLIQUER LA REGLE ?).

- La fonction APPLIQUER rend VRAI si la règle s'applique ET la conclusion est un fait NOUVEAU, dans ce cas elle affiche à l'écran ce nouveau fait.

- La fonction AJOUTFAIT ajoute le nouveau fait à la liste des faits.

- Les procédures INFERENCE et MOTEUR parcourent la liste des règles jusqu'à ce qu'il n'y ait PLUS de FAITS NOUVEAUX.

- La procédure SYSTEXP demande l'entrée au clavier de la liste des faits de départ puis lance MOTEUR.

Comment utiliser le programme qui vous est proposé ici? Il faut écrire -ou faire écrire si vous n'êtes pas habitué à LOGO- le programme avec un système de règles très simples, par exemple des règles abstraites du type " si A et B alors C ". L'execution a lieu lorsqu'on tape SYSTEXP après le prompt: ? . Lorsque le programme sera techniquement au point, vous remplacerez ce système de règles par VOTRE PROPRE SYSTEME en entrant dans l'éditeur LOGO.

Vous trouverez ci-dessous un exemple -très incomplet!- de règles concernant la peinture contemporaine ainsi que des exemples d'exécution et, pour le folklore, une liste de règles de météorologie pratique pour la Basse-Normandie écrite par Y.SALLES pour le générateur de système expert XPERT de PH.BRUTUS de l'Université de Caen. Bon courage!

Note: Le programme qui vous est proposé ici est très largement inspiré d'un programme écrit en LE-LISP 80 et proposé en projet par F.FAGES à ses étudiants de l'Université PARIS-DAUPHINE. Merci à lui.

```
POUR EXPERT
DONNE "LISTEREGLES
[[REGLE 1 [SI [LES COULEURS SONT EXALTEES] ] [ALORS [C'EST UN FAUVE]]]
 [REGLE 2 [SI [LES TRAITS SONT GEOMETRIQUES] ] [ALORS [C'EST UN CUBISTE]]]
 [REGLE 3 [SI [C'EST UN FAUVE] [LE SUJET EST UN CHAMP DE COURSES] ]
          [ALORS [C'EST UN DUFY]]]
 [REGLE 4 [SI [C'EST UN FAUVE] [LE SUJET EST UN NU] ]
          [ALORS [C'EST UN MATISSE]]]
 [REGLE 5 [SI [C'EST UN FAUVE] [LE SUJET EST UN INTERIEUR] ]
          [ALORS [C'EST UN MATISSE]]]
 [REGLE 6 [SI [LE DESSIN EST EXPLICITE] ] [ALORS [C'EST UN FIGURATIF]]]
 [REGLE 7 [SI [C'EST UN FIGURATIF] [LE SUJET EST UNE TAHITIENNE] ]
          [ALORS [C'EST UN GAUGUIN]]]
 [REGLE 8 [SI [C'EST UN CUBISTE] [LE SUJET EST VU SUR PLUSIEURS FACES] ]
          [ALORS [C'EST UN PICASSO]]] ]
FIN
```

EXEMPLES D'EXECUTION

```
VOUS VENEZ DE DEFINIR EXPERT
VOUS VENEZ DE DEFINIR HYPOTHESE
VOUS VENEZ DE DEFINIR CONCLUSION
VOUS VENEZ DE DEFINIR TESTHYP
VOUS VENEZ DE DEFINIR APPLIQUER
VOUS VENEZ DE DEFINIR AJOUTFAIT
VOUS VENEZ DE DEFINIR INFERENCE
VOUS VENEZ DE DEFINIR MOTEUR
VOUS VENEZ DE DEFINIR SYSTEXP
?
?
?SYSTEXP
ENTREZ LA LISTE DE FAITS SOUS LA FORME [FAIT1] [FAIT2] ... [FAITN]
[LES COULEURS SONT EXALTEES][LE SUJET EST UN INTERIEUR]
DE LA REGLE 1 JE DEDUIS QUE C'EST UN FAUVE
DE LA REGLE 5 JE DEDUIS QUE C'EST UN MATISSE
?
?
?SYSTEXP
ENTREZ LA LISTE DE FAITS SOUS LA FORME [FAIT1] [FAIT2] ... [FAITN]
[LE DESSIN EST EXPLICITE][LE SUJET EST UNE TAHITIENNE]
DE LA REGLE 6 JE DEDUIS QUE C'EST UN FIGURATIF
DE LA REGLE 7 JE DEDUIS QUE C'EST UN GAUGUIN
?
```

```
POUR EXPERT

DONNE "LISTEREGLES

[[REGLE 1 [SI [LE VENT VIENT DU NORD][LE VENT NE TOURNE PAS][LA TEMPERATURE EST
          FRAICHE]] [ALORS [LE TEMPS DOIT RESTER BEAU]]]

 [REGLE 2 [SI [LE VENT VIENT DU NORD][LE VENT TOURNE A L'EST][LA TEMPERATURE
          EST FRAICHE]] [ALORS [LE TEMPS DOIT RESTER BEAU DURABLEMENT]]]

 [REGLE 3 [SI [LE VENT VIENT DE L'OUEST][LE VENT NE TOURNE PAS]] [ALORS [LE
          TEMPS SERA VARIABLE AVEC AVERSES]]]

 [REGLE 4 [SI [LE VENT VIENT DE L'OUEST][LE VENT TOURNE AU SUD][LA TEMPERATURE
          NE CHANGE PAS]] [ALORS [LE TEMPS VA SE COUVRIR ET IL VA PLEUVOIR]]]

 [REGLE 5 [SI [LE VENT VIENT DE L'OUEST][LE VENT TOURNE AU SUD][LA TEMPERATURE
          MONTE]] [ALORS [IL Y A RISQUE DE COUP DE VENT AVEC FORTES PLUIES]]]


 [REGLE 6 [SI [LE VENT VIENT DU SUD][LE VENT NE TOURNE PAS][LA TEMPERATURE EST
          CHAUDE]] [ALORS [LE TEMPS SERA STATIONNAIRE, PLUVIEUX ET COUVERT]]]


 [REGLE 7 [SI [LE VENT VIENT DE L'EST][LE VENT NE TOURNE PAS]] [ALORS [LE BEAU
          TEMPS VA SE MAINTENIR]]]

 [REGLE 8 [SI [LE VENT VIENT DE L'EST][LE VENT TOURNE AU SUD-EST][LA
          TEMPERATURE MONTE]] [ALORS [IL Y A RISQUE D'ORAGE]]]

 [REGLE 9 [SI [LE VENT VIENT DU NORD][LE VENT TOURNE AU SUD][LA TEMPERATURE
          MONTE]] [ALORS [LE VENT VA FORCIR AVEC RISQUE DE PLUIE]]]

 [REGLE 10 [SI [LE VENT VIENT DU SUD][LE VENT NE TOURNE PAS][LE CIEL EST GRIS
          AVEC DES ZONES NOIRES]] [ALORS [RISQUE DE GRAIN AVEC FORTE PLUIE]]]

 [REGLE 11 [SI [LE VENT VIENT DU NORD][LE VENT TOURNE A L'OUEST]]
          [ALORS [RISQUE DE TEMPS VARIABLE AVEC AVERSES]]]

 [REGLE 12 [SI [LE VENT VIENT DU NORD][LE VENT TOURNE AU NORD-OUEST]] [ALORS[LE
           TEMPS PEUT RESTER BEAU MAIS VA SE COUVRIR LEGEREMENT]]]



 [REGLE 13 [SI [LE VENT VIENT DE L'OUEST][LE VENT TOURNE AU NORD][LA
           TEMPERATURE DESCEND]] [ALORS [LE TEMPS VA S'ECLAIRCIR, IL DEVRAIT
           FAIRE BEAU]]]

 [REGLE 14 [SI [LE VENT VIENT DU SUD][LE VENT TOURNE AU SUD-OUEST][LA
           TEMPERATURE DESCEND]] [ALORS [LE TEMPS VA S'ECLAIRCIR AVEC DES
           AVERSES TEMPORAIRES]]]

 [REGLE 15 [SI [LE VENT VIENT DU SUD][LE VENT TOURNE A L'OUEST][LA TEMPERATURE
           DESCEND]] [ALORS [LE TEMPS VA DEVENIR VARIABLE AVEC AVERSES]]] ]

FIN
```

```
MOTEUR ZERO EN CHAINAGE AVANT PAR D. SALLES AVRIL 90



POUR EXPERT
DONNE "LISTEREGLES
[[REGLE 1 [SI [LES COULEURS SONT EXALTEES] ] [ALORS [C'EST UN FAUVE]]]
 [REGLE 2 [SI [LES TRAITS SONT GEOMETRIQUES] ] [ALORS [C'EST UN CUBISTE]]]
 [REGLE 3 [SI [C'EST UN FAUVE] [LE SUJET EST UN CHAMP DE COURSES] ]
             [ALORS [C'EST UN DUFY]]]
 [REGLE 4 [SI [C'EST UN FAUVE] [LE SUJET EST UN NU] ]
             [ALORS [C'EST UN MATISSE]]]
 [REGLE 5 [SI [C'EST UN FAUVE] [LE SUJET EST UN INTERIEUR] ]
             [ALORS [C'EST UN MATISSE]]]
 [REGLE 6 [SI [LE DESSIN EST EXPLICITE] ] [ALORS [C'EST UN FIGURATIF]]]
 [REGLE 7 [SI [C'EST UN FIGURATIF] [LE SUJET EST UNE TAHITIENNE] ]
             [ALORS [C'EST UN GAUGUIN]]]
 [REGLE 8 [SI [C'EST UN CUBISTE] [LE SUJET EST VU SUR PLUSIEURS FACES] ]
             [ALORS [C'EST UN PICASSO]]] ]
FIN


POUR HYPOTHESE :REGLE
RENDS   SP PREM SP SP SD :REGLE
FIN



POUR CONCLUSION :REGLE
RENDS  SP DER :REGLE
FIN



POUR TESTHYP :REGLE
DONNE "HYP HYPOTHESE :REGLE
DONNE "TEST "VRAI
REPETE COMPTE HYPOTHESE :REGLE
 [SI NON MEMBRE? PREM :HYP :LISTEFAITS [DONNE "TEST "FAUX] [DONNE "HYP SP :HYP]

RENDS :TEST
FIN




POUR APPLIQUER :REGLE
DONNE "UTILE "FAUX
DONNE "CONCL CONCLUSION :REGLE
SI (ET TESTHYP :REGLE NON MEMBRE? PREM :CONCL :LISTEFAITS)
         [(EC [DE LA REGLE] PREM SP :REGLE [JE DEDUIS QUE] PREM :CONCL)
             DONNE "UTILE "VRAI]

RENDS :UTILE
FIN
```

```
POUR AJOUTFAIT :REGLE
SI  APPLIQUER :REGLE
    [DONNE "LISTEFAITS PHRASE :LISTEFAITS CONCLUSION :REGLE RENDS "VRAI]
RENDS "FAUX
FIN


POUR INFERENCE
DONNE "NOUVEAUFAIT "FAUX
DONNE "CONNAISSANCE :LISTEREGLES
REPETE COMPTE :CONNAISSANCE
      [SI AJOUTFAIT PREM :CONNAISSANCE [DONNE "NOUVEAUFAIT "VRAI][]
         DONNE "CONNAISSANCE SP :CONNAISSANCE]
RENDS :NOUVEAUFAIT
FIN



POUR MOTEUR
SI NON INFERENCE [STOP][MOTEUR]
FIN


POUR SYSTEXP
EXPERT
EC [ENTREZ LA LISTE DE FAITS SOUS LA FORME [FAIT1][FAIT2]...[FAITN]]
DONNE "LISTEFAITS LISLISTE
MOTEUR
FIN
```

- o -

Dépôt légal. Juin 1990