Premières leçons de programmation en Turbo Pascal

Laurent, Patrick, Radhia et Thibault Cousot

Disquettes d'accompagnement disponibles

Bon de commande dans le livre, P. 297

A l'usage des écoliers collégiens lycéens et débutants en informatique

COLLECTION BORLAND / McGRAW-HILL

Premières leçons de programmation en Turbo Pascal

Laurent, Patrick, Radhia et Thibault Cousot

McGRAW-HILL

Paris — Auckland — Bogota — Caracas — Hambourg — Jakarta — Lisbonne — Londres — Madrid — Mexico — Milan — Montréal — New Delhi — New York — Panama — San Juan — São Paulo — Singapour — Sydney — Tokyo — Toronto

1991

Maquette de couverture : Françoise Rojare

ISBN : 2-7042-1239-2
ISSN : 0992-5880

McGraw-Hill — 28, rue Beaunier — 75014 Paris

PREMIÈRES LEÇONS DE PROGRAMMATION EN TURBO PASCAL

Laurent COUSOT **Patrick COUSOT**
Radhia COUSOT **Thibault COUSOT**

A l'usage des écoliers,
collégiens,
lycéens,
et débutants en informatique.

Table des matières

Introduction

L'usage de l'ordinateur se généralise dans tous les domaines de l'activité économique. La plupart des utilisateurs emploient des logiciels d'application pour accomplir une ou plusieurs tâches spécifiques : édition de texte, comptabilité, jeux, etc. Par désir de tirer tout le parti de la puissance de la machine, par plaisir ludique ou par simple envie de comprendre le principe de fonctionnement des ordinateurs, de nombreux informaticiens en herbe ont envie d'apprendre, parfois seuls, la programmation. La tâche se révèle souvent longue, difficile voire rébarbative. Nous voudrions en faire une activité créatrice, formatrice pour l'esprit et passionnante.

De nombreuses introductions à la programmation reposent sur la connaissance préalable de l'architecture des ordinateurs ou utilisent des notions mathématiques qui ne sont pas à la portée des enfants ou des adultes n'ayant pas une formation scientifique [9] [1]. (Les petits chiffres surélevés renvoient à une note de bas de page expliquant le mot qui précède. Par exemple, la note qui se trouve en bas de cette page explique l'usage des chiffres entre crochets comme [9].) Nous proposons de procéder autrement.

Nous utilisons le dessin, qui est à la portée de tous. Cette approche est relativement nouvelle dans l'apprentissage de l'informatique, les progrès saisissants de la technologie qui ont permis de mettre à la portée du grand public des micro-ordinateurs dotés d'écrans graphiques très performants ne datant que des années 1980. Il est plaisant de noter que les origines de notre activité graphique remontent à la préhistoire (plus précisément à la période de l'art

1. Les scientifiques ont l'habitude de 'citer leurs sources' en donnant les références aux articles de revues scientifiques et aux livres des auteurs dont ils utilisent les idées. La méthode moderne pour citer les sources est d'utiliser un numéro placé entre crochets (comme par exemple [9]) qui renvoie à une liste de références, placée à la fin du livre, que l'on appelle *bibliographie* (voir page 281).

pariétal[2] entre 22000 et 13000 ans av. J.-C.). L'informatique graphique qui a 10 ans va donc nous permettre à 20000 ans d'intervalle de dessiner avec le même plaisir que nos lointains ancêtres.

Ce livre s'adresse à tous. Les enfants des classes maternelles sont capables de faire des dessins stylisés à la main et avec un peu d'imagination, de les reproduire sur l'écran de l'ordinateur en utilisant un programme de dessin interactif (chapitres 1 et 2). Les enfants des classes primaires peuvent écrire des programmes pour réaliser des dessins figuratifs (chapitres 3 à 5). Très vite ils découvriront la facilité avec laquelle ils peuvent programmer l'ordinateur pour faire des dessins géométriques simples basés sur des motifs répétitifs qu'ils n'auraient pas la patience ni la minutie d'exécuter à la main (chapitres 6 et 7). Les collégiens mettront en œuvre leurs connaissances rudimentaires d'algèbre et de géométrie analytique pour écrire des programmes de dessin utilisant des calculs élémentaires et des repères cartésiens (chapitres 8 à 17). Ils trouveront une base intuitive pour mettre en œuvre des concepts qui peuvent paraître abstraits. Les lycéens sont à même d'assimiler des notions plus informatiques comme le codage, voire des notions moins élémentaires comme la récursivité, qui deviennent évidentes par le dessin (chapitres 17 à 22). Les étudiants trouveront du plaisir à comprendre simplement la programmation qui paraît parfois obscure quand elle est présentée trop rapidement en premier cycle universitaire.

Les langages de programmation n'offrant pas les possibilités graphiques souhaitables, nous avons ajouté une bibliothèque à Pascal[3] qui nous permet d'obtenir de façon simple des effets simples (comme les arcs de cercles) ou spectaculaires (comme la peinture) sans être liés à la technologie des écrans graphiques actuels et sans faire appel à des notions avancées de programmation comme les structures de données. Cette bibliothèque ainsi que tous les exemples et programmes solutions des exercices proposés dans le livre sont fournis sur disquette[4]. Elle offre de nombreuses possibilités qui ne sont pas explorées dans le livre, comme par exemple pour les spécialistes, de créer des

2. Peinture sur les parois d'une grotte.

3. Nous utiliserons les compilateurs Turbo Pascal 5.5 ou 6.0 de Borland sur les ordinateurs compatibles IBM PC et le compilateur Turbo Pascal 1.1 de Borland sur les Macintosh.

4. Un bon de commande de la disquette d'accompagnement se trouve à la fin du livre. Pour apprendre à l'utiliser, l'insérer dans un lecteur puis, sur compatible IBM PC taper `LisezMoi` ou sur Macintosh cliquer deux fois rapidement avec la souris sur l'icône `LisezMoi`.

programmes interactifs d'apprentissage comme ceux qui sont proposés pour l'addition, les coordonnées cartésiennes et la translation.

Ce livre devrait être lu par étapes successives, en prenant le temps et le plaisir de faire de nombreuses expériences personnelles sur ordinateur, en résolvant des exercices, en modifiant les programmes proposés en solution et en recherchant de nouveaux thèmes de dessin.

Addenda

Nous donnons en addenda à la fin de chaque chapitre des notes additionnelles destinées aux lecteurs déjà familiers de l'informatique.

Nous introduisons dans ce livre les activités de base de la programmation comme la saisie du texte, l'interprétation, la compilation, l'exécution et la mise au point de programmes. Les notions de programmation sont présentées en PASCAL : structuration des programmes en procédures, itérations bornées, expressions entières, déclarations de constantes, paramètres (passés par valeur), tests, expressions booléennes, itérations non bornées, codages de l'information, expressions rationnelles (réelles), récursivité, variables de types simples et affectations. Notre approche basée sur le dessin permet de donner une intuition graphique à ces notions et évite les problèmes plus complexes de spécification et de correction des programmes (qui sont des évidences visuelles). Les structures de données (tableaux, enregistrements et pointeurs en PASCAL) sont volontairement laissées pour des apprentissages ultérieurs plus élaborés.

L'apprentissage de la programmation à un niveau très élémentaire comme celui qui est proposé dans ce livre permet d'acquérir le sens du passage du concret à l'abstrait, l'intuition des structures formelles, le raisonnement inductif et la capacité à formaliser qui sont à la base de la programmation mais également le point de départ de toute démarche scientifique.

Remerciements : Nous remercions nos collègues qui ont utilisé le manuscrit de ce livre et les disquettes d'accompagnement pour initier leurs enfants à la programmation et tout particulièrement Guy Cousineau, Michèle Soria, Denis et Marion ; Colette et Jean-Marc Steyaert, Florent et Benoît ; Jean Vuillemin et Thomas.

1

Spécification

Nous allons apprendre à programmer un ordinateur en lui faisant faire des dessins. Pour commencer simplement nous utilisons du papier quadrillé à petits carreaux. La **règle de dessin** à respecter est que *nos dessins ne doivent comporter que des traits droits et des quarts de cercle joignant obligatoirement deux extrémités d'un même côté horizontal, d'un même côté vertical ou d'une même diagonale d'un carreau du quadrillage.* Cette règle de dessin nous offre donc la possibilité de tracer :

- les segments horizontaux ⎯ correspondant aux côtés horizontaux d'un carreau du quadrillage ;
- les segments verticaux | correspondant aux côtés verticaux d'un carreau du quadrillage ;
- les segments obliques ╱ et ╲ correspondant aux diagonales d'un carreau du quadrillage ;
- les quarts de cercles ⌣ et ⌢ joignant les extrémités d'un même côté horizontal d'un carreau du quadrillage ;
- les quarts de cercles (et) joignant les extrémités d'un même côté vertical d'un carreau du quadrillage ;
- les quarts de cercles ◜ , ◞ , ◝ et ◟ joignant les extrémités d'une même diagonale d'un carreau du quadrillage.

Les petits dessins de la figure 1.1 (page 6) sont des exemples simples tracés en respectant notre règle de dessin sur papier quadrillé. Bien que cette règle ne laisse pas une totale liberté à notre imagination, nous pouvons faire des dessins figuratifs comme ceux du stégosaure ou du château fort de la page 6.

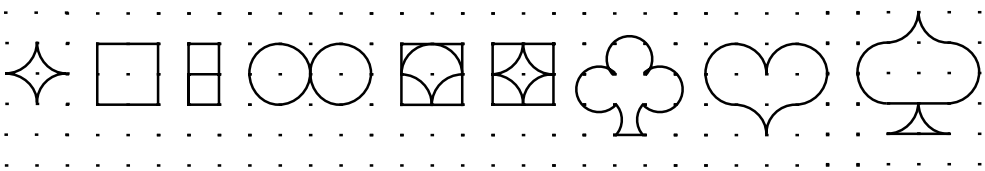


FIG. 1.1 – *Quelques dessins géométriques simples*

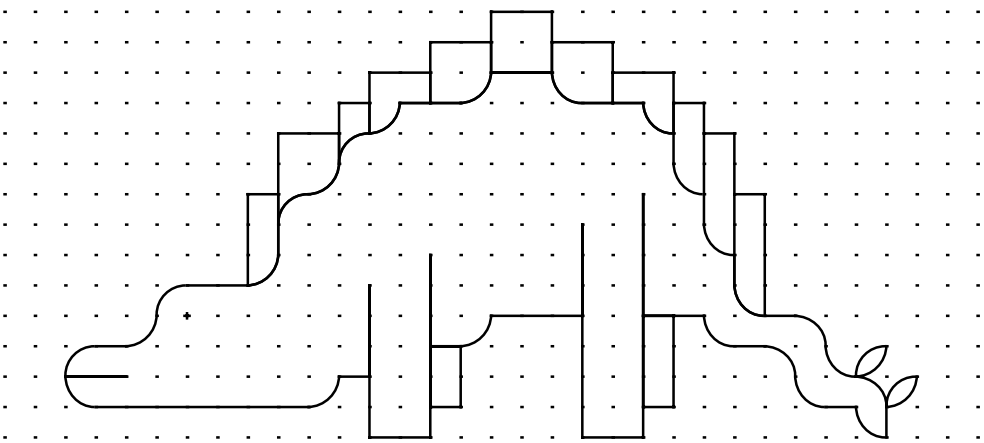


FIG. 1.2 – *Stégosaure*

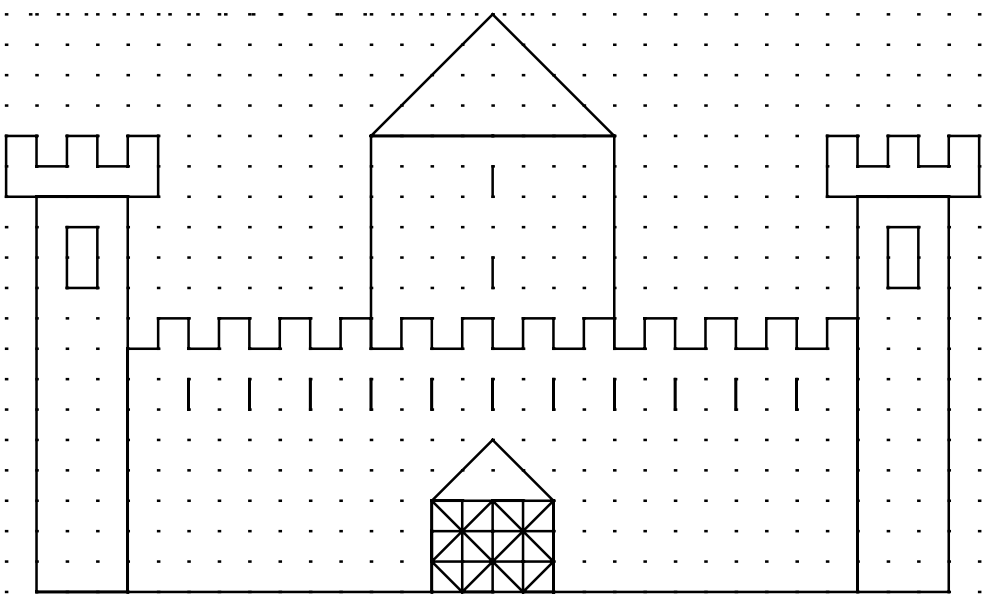


FIG. 1.3 – *Château fort*

Exercice 1

_ *Compléter les dessins de la figure 1.4 ci-dessous, en n'utilisant que des traits horizontaux* __ *, verticaux* | *ou obliques* / *et* \ *joignant obligatoirement deux sommets du quadrillage de numéros successifs, sommets qui sont matérialisés sur cette figure par de petits points.*

_ *Réaliser des dessins géométriques sur papier quadrillé à petits carreaux respectant la règle de dessin. Voici quelques idées (ces dessins seront ensuite réalisés sur ordinateur) : une marelle, un trapèze, un parallélogramme, un rectangle, un carré, une maison, une fleur stylisée, une cocotte en papier, un bonhomme, la lettre H majuscule en relief, les chiffres d'une montre digitale, l'alphabet (en lettres majuscules), un hélicoptère, un camion, un train, une péniche, etc.* □ [1]

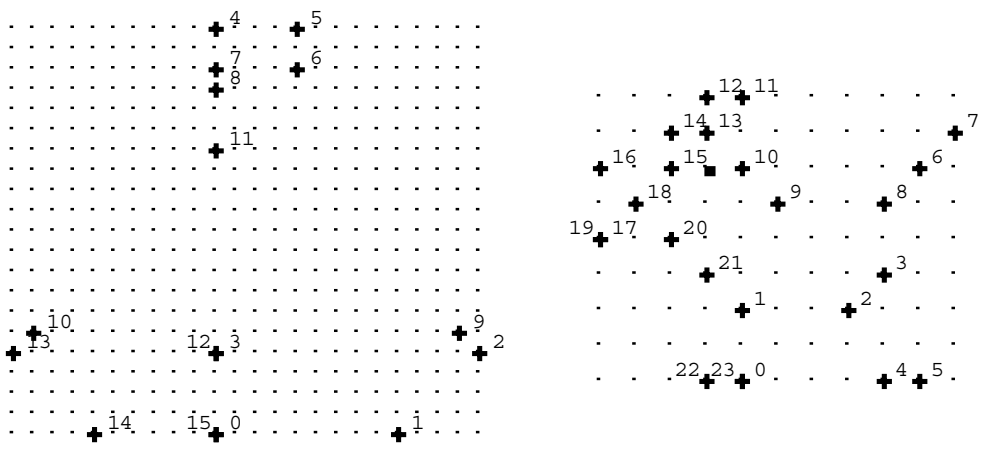


FIG. 1.4 – *Dessins à compléter*

1. La boîte □ marque la fin du texte d'un exemple, d'un exercice, d'un corrigé, etc. Elle permet au lecteur qui ne voudrait pas lire ce texte de trouver rapidement où reprendre pour continuer sa lecture.

Corrigé 1

– *Une fois complétés, les dessins de la figure 1.4 (page 7) représentent un bateau et un chien.*

– *Des exemples de dessins sont donnés aux pages 8, 9 et 10.* □[2]

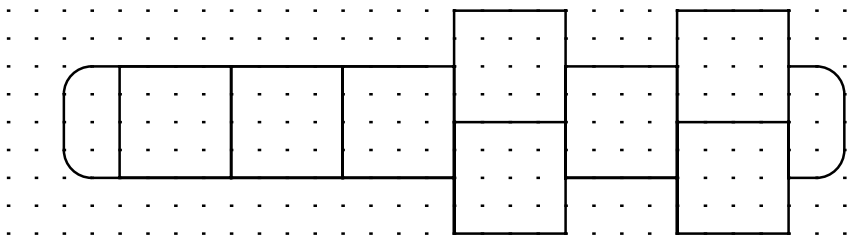


FIG. 1.5 – *Marelle*

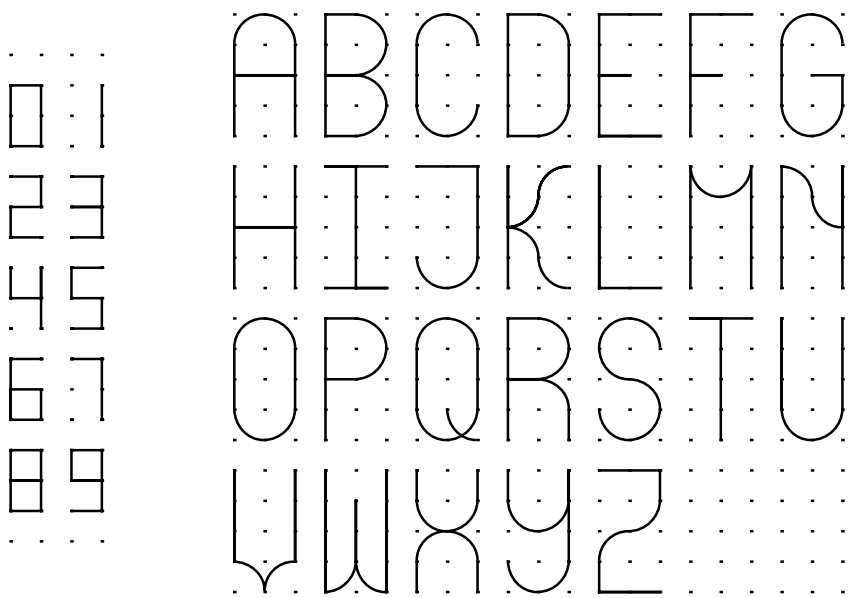


FIG. 1.6 – *Chiffres digitaux et alphabet (en lettres majuscules)*

2. Nous proposons les solutions des exercices quelques pages après en avoir donné les énoncés. Nous conseillons vivement au lecteur de lire ce livre par petites étapes en prenant le soin de faire quelques exercices après chaque chapitre.

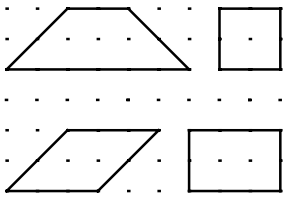


FIG. 1.7 – *Quadrilatères*

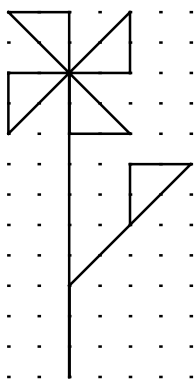


FIG. 1.8 – *Fleur stylisée*

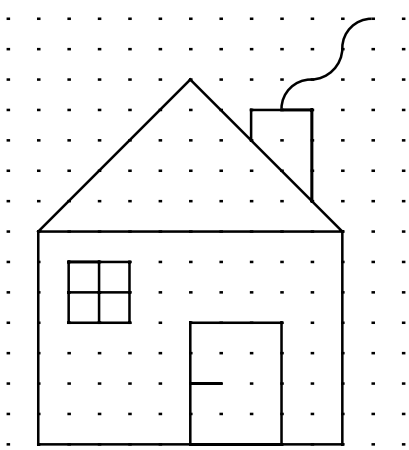


FIG. 1.9 – *Maison*

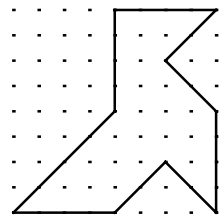


FIG. 1.10 – *Cocotte en papier*

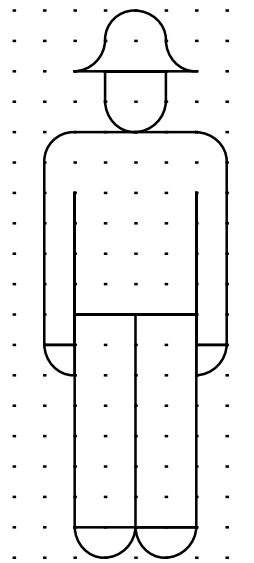


FIG. 1.11 – *Bonhomme*

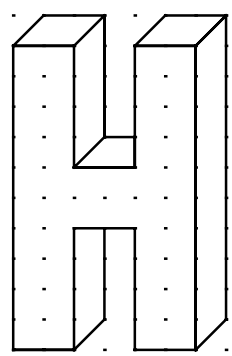


FIG. 1.12 – *Lettre H majuscule en relief*

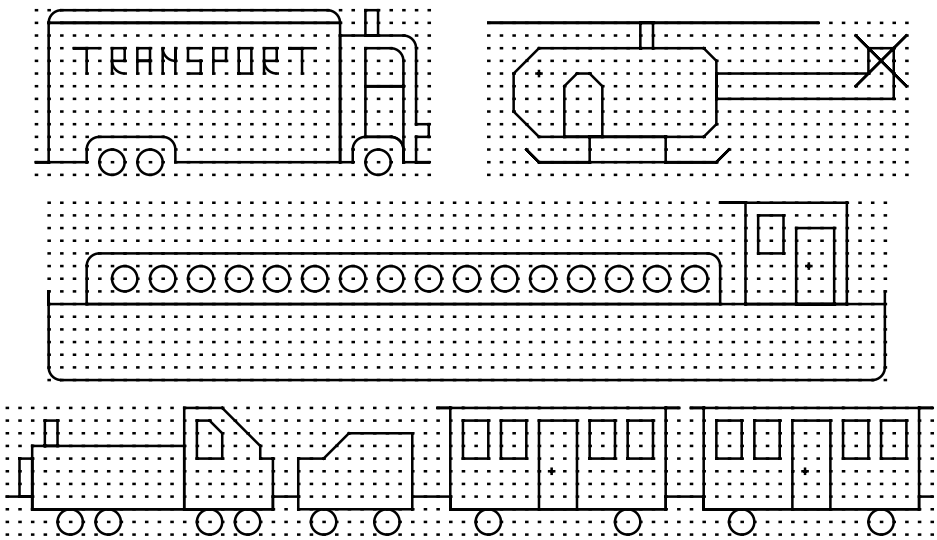


FIG. 1.13 – *Moyens de transport*

Addenda

Avant d'écrire un programme informatique pour résoudre un problème, il faut commencer par spécifier avec précision la tâche que devra accomplir l'ordinateur. Dans notre approche graphique pour expliquer la programmation des ordinateurs, cette *spécification* est un dessin, l'ordinateur devant réaliser ce dessin.

Le dessin à la main en respectant la contrainte de n'utiliser que des traits de la forme ⎯ , | , ⁄ , \ , ◜ , ◞ , ◠ , ◟ , ◡ , ◠ , (ou) est difficile pour les très jeunes enfants. Pour assurer une progression du simple au complexe, on commence par leur faire recopier des dessins uniquement constitués de traits horizontaux et verticaux sur du papier à grands carreaux puis on introduit les segments obliques, les quarts de cercles sur les diagonales et enfin les quarts de cercles sur les côtés d'un carré du quadrillage. Ces dessins géométriques constituent une excellente préparation à l'apprentissage de l'écriture. Ensuite, les enfants ont plaisir à inventer leurs propres dessins. Ils doivent alors apprendre à modéliser leurs idées dans le cadre assez strict qui leur est fixé par les règles de dessin. Cette notion de *modèle*, au sens de représentation simplifiée d'un processus ou d'un système complexe est à la base de la *démarche scientifique*.

Ces dessins géométriques construits en n'utilisant que quelques formes primitives constituent également un premier pas vers la programmation qui consiste à construire un édifice parfois complexe à partir des briques élémentaires que sont les instructions du langage de programmation.

Dans le chapitre suivant, il s'agit non plus de faire des dessins à la main mais de les dessiner sur l'écran de l'ordinateur en pilotant un robot avec le clavier de cet ordinateur. Ceci permet de découvrir le fonctionnement de l'ordinateur en *mode interactif* et d'entrer directement dans le monde de l'*infographie*, c'est-à-dire la création d'images sur ordinateur. Par construction, le robot dessine en respectant les règles de dessins qui ont été fixées. Il permet de corriger ses erreurs facilement, de conserver son dessin sur disque, d'en faire des réductions ou des agrandissements et d'en imprimer autant d'exemplaires que l'on désire.

2

Interaction

2.1 Le robot

Pour réaliser nos dessins géométriques sur ordinateur nous allons piloter un petit robot. Son nom est $\mathcal{R}^{o}_{b}{}^{o}t$[1]. Notre $\mathcal{R}^{o}_{b}{}^{o}t$ porte un crayon qu'il peut lever ou baisser pendant ses déplacements sur l'écran de l'ordinateur. Quand le $\mathcal{R}^{o}_{b}{}^{o}t$ se déplace avec son crayon baissé, il laisse une trace sur l'écran comme le ferait un escargot sur une feuille. Quand il se déplace avec son crayon levé, il ne laisse aucune trace de son passage. Pour dessiner, le $\mathcal{R}^{o}_{b}{}^{o}t$ doit suivre exactement les lignes du dessin avec son crayon baissé. Il peut également aller d'un point à un autre du dessin qui ne sont pas reliés par un trait en levant son crayon. Pour guider le $\mathcal{R}^{o}_{b}{}^{o}t$ nous utilisons le clavier de l'ordinateur qui sert de télécommande. Nous pouvons donc piloter le $\mathcal{R}^{o}_{b}{}^{o}t$ comme s'il s'agissait d'une petite voiture conduite à distance par radiocommande.

Sur l'écran de l'ordinateur notre $\mathcal{R}^{o}_{b}{}^{o}t$ est représenté par une petite flèche ⬆ qui indique dans quelle direction de la rose des vents il est tourné (voir la figure 2.1 , page 14). Il peut s'orienter au nord ⬆, au nord-est ⬈, à l'est ➡, au sud-est ⬊, au sud ⬇, au sud-ouest ⬋, à l'ouest ⬅ ou au nord-ouest ⬉. La flèche est pleine quand le $\mathcal{R}^{o}_{b}{}^{o}t$ a baissé son crayon et évidée (⇧, ⇗, ⇨, ⇘, ⇩, ⇙, ⇦ ou ⇖) quand le crayon est levé.

Comme les trains sur leurs rails, notre $\mathcal{R}^{o}_{b}{}^{o}t$ ne peut pas aller n'importe où sur l'écran. Il doit impérativement se déplacer sur les traits ou arcs joignant deux points successifs d'une grille dont les dimensions sont fixées par la taille

1. La calligraphie de son nom ne laisse aucun doute sur son caractère humanoïde et sympathique.

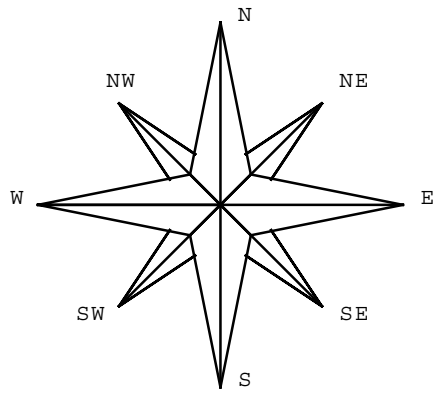


FIG. 2.1 – *Rose des vents à huit directions*

de l'écran de l'ordinateur. Une petite partie de cette grille est représentée ci-dessous à la figure 2.2.

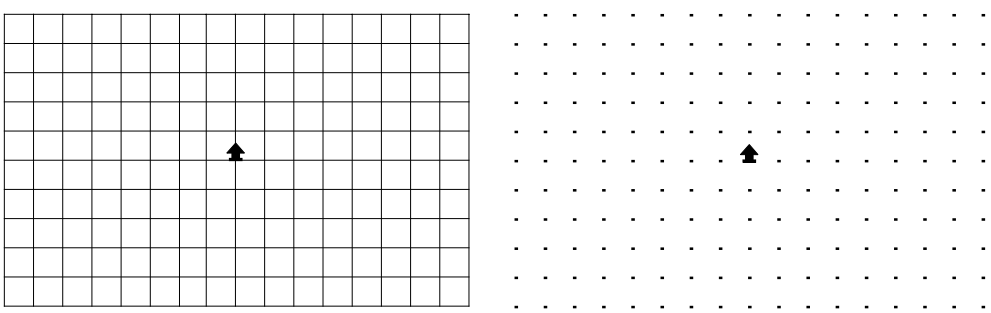


FIG. 2.2 – *Grille et points d'arrêt du Robot*

Cette grille imaginaire correspond au papier quadrillé que nous avons utilisé pour faire nos dessins à la main, sauf que les carreaux ne sont pas visibles sur l'écran de l'ordinateur. Après un déplacement, le *Robot* s'arrête toujours à l'intersection d'une ligne horizontale et d'une ligne verticale de la grille imaginaire. La figure 2.3 (page 15) montre toutes les trajectoires possibles du *Robot* entre deux points d'arrêt quelconques sur la grille. La figure 2.4 (page 15) décrit la position exacte du *Robot* quand il s'arrête sur la grille. Il place la pointe de son crayon située au milieu et à l'arrière de la flèche exactement sur un point d'arrêt de la grille.

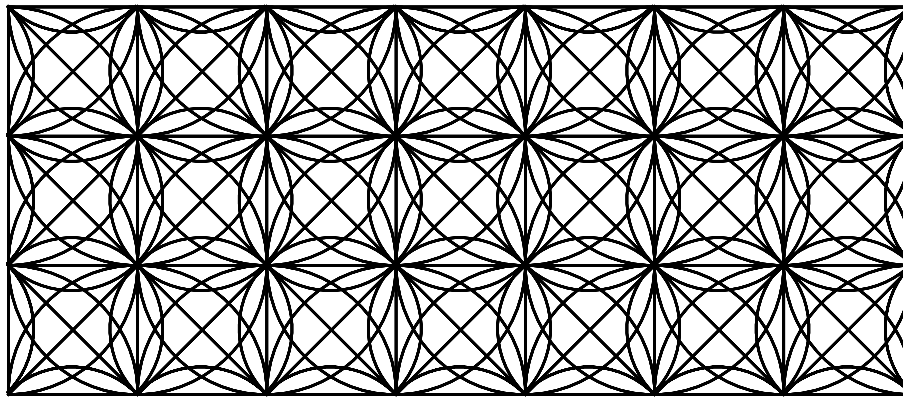


FIG. 2.3 – *Trajectoires du Robot (grossissement* × 4,5*)*

FIG. 2.4 – *Le Robot arrêté (grossissement* × 2,5*)*

2.2 Pilotage du robot

Pour piloter le *Robot* sur l'ordinateur, il faut "*exécuter le programme* `Dessiner`" qui se trouve sur la disquette d'accompagnement de ce livre. La façon d'exécuter ce programme `Dessiner` dépend de l'ordinateur utilisé[2].

2.2.1 Position initiale du robot

Quand l'exécution du programme `Dessiner` commence, l'écran de l'ordinateur se présente comme on le voit en réduction à la figure 2.5 (page 16). Au départ le *Robot* se trouve donc arrêté au milieu de la grille imaginaire, crayon baissé, orienté vers le nord ⬆. Pour piloter le *Robot*, nous utilisons les touches du clavier de l'ordinateur qui sert de télécommande. Nous allons maintenant voir sur quelles touches appuyer pour lever ou baisser le crayon,

2. Introduire dans un lecteur la disquette d'accompagnement de ce livre puis sur compatible IBM PC taper `Dessiner` ou sur Macintosh cliquer deux fois rapidement sur l'icône de l'application `Dessiner`.

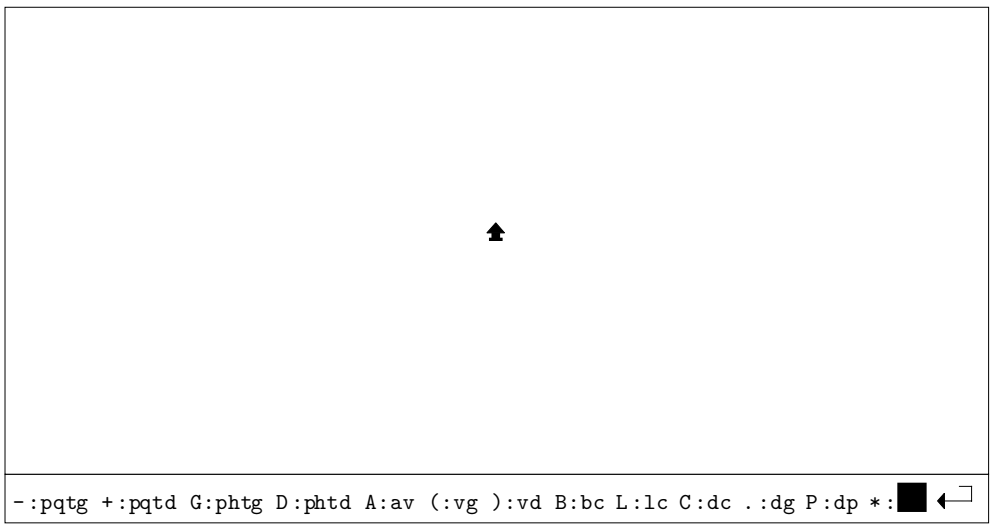


FIG. 2.5 – *Écran de l'ordinateur au début* de l'exécution du programme `Dessiner`

avancer, pivoter sur place, tourner, etc.

2.2.2 Manœuvrer le crayon du robot

En tapant sur la touche marquée [L] du clavier, le $\mathcal{R}obot$ lève son crayon. En tapant sur la touche [B], il le baisse. Faute de disposer de beaucoup de place sur l'écran de l'ordinateur pour permettre de rappeler en détail le mode d'emploi du $\mathcal{R}obot$, nous utilisons des abréviations mnémotechniques[3] placées en bas de l'écran comme on peut le voir à la figure 2.5 ci-dessus. Par exemple :

- **L**:`lc` signifie "la touche [L] permet de lever le crayon" ;
- **B**:`bc` signifie "la touche [B] permet de baisser le crayon".

L'effet des touches [L] et [B] de manœuvre du crayon du $\mathcal{R}obot$ est résumé dans la table 2.1 (page 17).

3. Les abréviations mnémotechniques permettent d'aider la mémoire par association d'idées.

L (lc)	Avant	↑	↗	→	↘	↓	↙	←	↖
	Après	↑	↗	→	↘	↓	↙	←	↖
B (bc)	Avant	↑	↗	→	↘	↓	↙	←	↖
	Après	↑	↗	→	↘	↓	↙	←	↖

TAB. 2.1 – *Manœuvres du crayon du $\mathcal{R}o\underline{b}ot$*

2.2.3 Faire pivoter le robot sur place d'un quart de tour

La touche – fait pivoter le $\mathcal{R}o\underline{b}ot$ sur place d'un quart de tour à gauche (`pqtg` en abrégé). Par exemple si le $\mathcal{R}o\underline{b}ot$ est orienté dans la direction nord ↑, la touche – le fait pivoter sur place dans la direction ouest ←. S'il est orienté à l'est →, il se place dans la direction nord ↑. La touche + fait pivoter le $\mathcal{R}o\underline{b}ot$ sur place d'un quart de tour à droite (`pqtd` en abrégé).

Quand on parle du côté gauche du $\mathcal{R}o\underline{b}ot$, il faut s'imaginer à la place du $\mathcal{R}o\underline{b}ot$, le regard tourné dans la direction de la flèche. Par exemple, la gauche de → est le haut de cette page tandis que la gauche de ↓ est à droite de cette feuille !

L'effet des commandes – (`pqtg`) et + (`pqtd`) est résumé dans la table ci-dessous dans le cas où le crayon est baissé. L'effet est le même quand le crayon est levé.

– (pqtg)	Avant	↑	↗	→	↘	↓	↙	←	↖
	Après	←	↖	↑	↗	→	↘	↓	↙
+ (pqtd)	Avant	↑	↗	→	↘	↓	↙	←	↖
	Après	→	↘	↓	↙	←	↖	↑	↗

TAB. 2.2 – *Pivotements du $\mathcal{R}o\underline{b}ot$ d'un quart de tour sur lui-même*

2.2.4 Faire pivoter le robot sur place d'un huitième de tour

La touche G fait pivoter le $\mathcal{R}o\underline{b}ot$ sur place d'un huitième de tour à gauche (`phtg` en abrégé). Par exemple, s'il est dans la direction nord ↑, le

Robot se place dans la direction nord-ouest ↖. En continuant à taper sur la touche [G], le *Robot* tourne à chaque fois d'un huitième de tour en passant par les positions successives ← puis ↙, ↓, ↘, →, ↗, ...La touche [D] fait pivoter le *Robot* sur place d'un huitième de tour à droite (`phtd` en abrégé). L'effet des touches [G] (`phtg`) et [D] (`phtd`) est résumé ci-dessous :

[G] (`phtg`)	Avant	↑	↖	←	↙	↓	↘	→	↗
	Après	↖	←	↙	↓	↘	→	↗	↑
[D] (`phtd`)	Avant	↑	↗	→	↘	↓	↙	←	↖
	Après	↗	→	↘	↓	↙	←	↖	↑

Tab. 2.3 – *Pivotements du Robot d'un huitième de tour sur lui-même*

2.2.5 Faire avancer le robot

La touche [A] fait avancer le *Robot* tout droit, de la taille d'un côté d'un carreau du quadrillage. L'abréviation mnémotechnique de cette commande du *Robot* est `av`. Si le crayon est baissé, ce déplacement permet de tracer un petit trait. Si le crayon est levé, ce déplacement ne laisse aucune trace sur le dessin. Dans les deux cas le *Robot* avance tout droit dans la direction où il se trouve sans changer d'orientation :

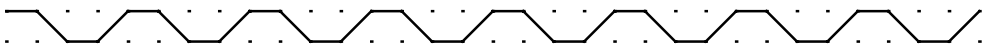
[A] (`av`)	Avant	↑	↗	→	↘	↓	↙	←	↖
	Après	↑	↗	→	↘	↓	↙	←	↖

Tab. 2.4 – *Avancées du Robot dans les huit directions*

Exemple 1 (Frise crantée) En tapant successivement sur les touches [A] (`av`), [+] (`pqtd`), [A] (`av`), [+] (`pqtd`), [A] (`av`), [−] (`pqtg`), [A] (`av`), [−] (`pqtg`), [A] (`av`), [+] (`pqtd`), [A] (`av`), [+] (`pqtd`), [A] (`av`), [−] (`pqtg`), [A] (`av`), [−] (`pqtg`), ..., on obtient la frise suivante :

□

Exemple 2 (Frise crénelée) En tapant successivement sur les touches D (phtd), D (phtd), A (av), D (phtd), A (av), G (phtg), A (av), G (phtg), A (av), D (phtd), A (av), D (phtd), A (av), ..., on obtient la frise suivante :



□

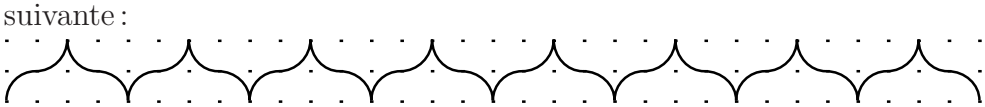
2.2.6 Faire virer le robot

La touche (permet de faire virer le $\mathcal{R}^{o}_{b}{}^{o}t$ à gauche (vg en abrégé) tandis que la touche) permet de le faire virer à droite (vd). Ces touches ont été choisies parce que les parenthèses évoquent l'idée de virage et que la parenthèse gauche (qui se trouve à gauche sur le clavier correspond au virage à gauche tandis que la parenthèse droite) qui correspond au virage à droite se trouve à droite sur le clavier. Si le crayon est baissé, ce déplacement permet de tracer un petit arc de cercle. Si le crayon est levé, ce déplacement ne laisse aucune trace sur le dessin. Dans les deux cas le $\mathcal{R}^{o}_{b}{}^{o}t$ se déplace sur la grille et change d'orientation comme indiqué dans la table ci-dessous :

((vg)	Avant								
	Après								
) (vd)	Avant								
	Après								

TAB. 2.5 – *Virages du $\mathcal{R}^{o}_{b}{}^{o}t$*

Exemple 3 (Frise d'accolades) En tapant successivement sur les touches) (vd), ((vg), + (pqtd), + (pqtd), ((vg),) (vd), + (pqtd), + (pqtd),) (vd), ((vg), + (pqtd), + (pqtd), ..., on obtient la frise suivante :



□

2.2.7 Impossibilité de sortir du cadre

En tapant sur la touche [A] plusieurs fois de suite le $\mathcal{R}o\underset{\frown}{b}ot$ ⬆ avance tout droit en traçant un trait jusqu'à rencontrer le bord du cadre qui délimite son champ d'action. Si l'on continue à taper sur la touche [A] pour essayer de le faire sortir du cadre c'est une *erreur* en ce sens que le $\mathcal{R}o\underset{\frown}{b}ot$ refuse de le faire et émet un « bip-bip » de protestation[4] !

2.2.8 Corriger les erreurs de pilotage

Il arrive souvent que l'on fasse des erreurs de pilotage du $\mathcal{R}o\underset{\frown}{b}ot$. Par exemple, au lieu de le faire tourner à droite avec une parenthèse droite [)] (`vd`), on tape une parenthèse gauche [(] (`vg`) et le $\mathcal{R}o\underset{\frown}{b}ot$ tourne à gauche ! La touche d'*effacement* marquée d'une flèche orientée à gauche [⟵] permet de corriger les erreurs en revenant en arrière et en effaçant les derniers tracés dans l'ordre inverse où ils ont été dessinés par le $\mathcal{R}o\underset{\frown}{b}ot$.

2.2.9 Terminer et reproduire le dessin

Pendant l'exécution du programme `Dessiner`, il est possible, à chaque instant, de terminer le dessin en tapant sur la touche de retour à la ligne [↩], également marquée `Entrée` ou en anglais `Return` ou `Ret` sur certains ordinateurs. Ceci est rappelé en bas à droite de l'écran de l'ordinateur par un symbole ↩ qui clignote.

Le dessin qui vient d'être terminé n'est pas perdu, il est possible de le reproduire et de l'imprimer[5].

2.2.10 Dessiner un point, une croix, la grille des points d'arrêt du robot

La touche [P] (`dp`) permet de dessiner un point, sans déplacer le $\mathcal{R}o\underset{\frown}{b}ot$. On utilise la touche [C] (`dc`) pour dessiner une petite croix `+`. La touche [.] (`dg`) permet de dessiner la grille des points d'arrêt du $\mathcal{R}o\underset{\frown}{b}ot$ comme on la voit par exemple sur la figure 1.13 (page 10).

4. La calligraphie de son nom ne laisse aucun doute sur son mécontentement.

5. La façon de procéder, qui dépend de l'ordinateur utilisé, est expliquée sur la disquette d'accompagnement du livre.

2.2.11 Dessiner en couleur

Quand on dispose d'un ordinateur avec écran couleur, le *Robot* a la possibilité de dessiner avec des crayons de différentes couleurs. Pendant l'exécution du programme `Dessiner`, la couleur du crayon utilisé par le *Robot* figure dans un petit carré placé en bas à droite de l'écran (voir la figure 2.5 page 16). Pour changer la couleur du crayon, il suffit de taper une ou plusieurs fois sur la touche [*] pour faire défiler toutes les couleurs possibles les unes après les autres.

2.2.12 Déplacer rapidement le robot avec la souris ou les flèches de défilement

Pour déplacer le *Robot* sans rien dessiner, on peut lever le crayon [L] (`lc`) puis avancer une ou plusieurs fois [A] (`av`) en changeant éventuellement de direction ([D] (`phtd`) ou [G] (`phtg`)) et enfin baisser le crayon [B] (`bc`). Pour aller plus vite, on peut utiliser la souris pour déplacer le *Robot* sur un point quelconque de la grille des points d'arrêt sans rien tracer. Sur les ordinateurs ne disposant pas de souris, le même effet peut être obtenu avec les flèches de défilement vers le haut [↑], vers le bas [↓], à gauche [←], ou à droite [→].

Résumé des commandes interactives de pilotage du robot

Les commandes de pilotage du *Robot* pour dessiner interactivement avec le programme `Dessiner` sont résumées dans la table 2.6 (page 22).

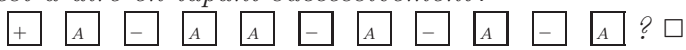
Touche :	*Commande :*	*Effet :*
Grille des points d'arrêt du Robot :		
.	`dg`	dessine la grille des points d'arrêt du *Robot*.
Pivotements du Robot sur place :		
D	`phtd`	fait pivoter le *Robot* sur place d'un huitième de tour à droite.
+	`pqtd`	fait pivoter le *Robot* sur place d'un quart de tour à droite.
G	`phtg`	fait pivoter le *Robot* sur place d'un huitième de tour à gauche.
–	`pqtg`	fait pivoter le *Robot* sur place d'un quart de tour à gauche.
Déplacements du Robot :		
A	`av`	fait avancer le *Robot* tout droit jusqu'au prochain point d'arrêt du quadrillage.
)	`vd`	fait tourner le *Robot* par un virage d'un quart de cercle à droite.
(	`vg`	fait tourner le *Robot* par un virage d'un quart de cercle à gauche.
Manœuvres du crayon du Robot :		
L	`lc`	lève le crayon du *Robot*, pour éviter de dessiner lors de ses déplacements.
B	`bc`	baisse le crayon du *Robot*, pour dessiner lors de ses déplacements.
P	`dp`	dessine un point, sans déplacer le *Robot*.
C	`dc`	dessine une croix, sans déplacer le *Robot*.
Erreurs de pilotage du Robot :		
←		efface, dans l'ordre inverse, les derniers tracés du *Robot*.
Arrêt du Robot :		
↩		termine le dessin (qui peut ensuite être reproduit et imprimé).

TAB. 2.6 – *Résumé des commandes interactives du Robot*

Exercice 2 *Supposons que le* Robot *soit initialement orienté vers le nord avec son crayon baissé* ⬆. *Parmi les chiffres d'une montre digitale représentés à la figure 1.6 (page 8), quel est celui obtenu en donnant au robot la* séquence de commandes[6] *ci-dessous :*

`pqtd; av; pqtg; av; av; pqtg; av; pqtg; av; pqtg; av;`

c'est-à-dire en tapant successivement :

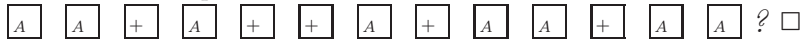


? □

Exercice 3 *Quelle lettre majuscule de l'alphabet de la figure 1.6 (page 8) obtient-on en donnant au* Robot ⬆ *la séquence de commandes suivante :*

`av; av; pqtd; av; pqtd; pqtd; av; pqtd; av; av; pqtd; av; av;`

c'est-à-dire en tapant :



? □

Exercice 4 *Quelle est la suite de commandes à donner au* Robot ⬆ *pour qu'il dessine la lettre* `T` *majuscule représentée à la figure 1.6 (page 8) ?* □

Exercice 5 *Quelle est la suite de commandes à donner au* Robot ⬆ *pour qu'il dessine le chiffre 4 représenté à la figure 1.6 (page 8) ?* □

Exercice 6 *Quelle est la séquence de commandes à donner au* Robot *pour dessiner l'octogone de la figure 2.6 ci-dessous ?* □

Exercice 7 *Quelle est la séquence de commandes à donner au* Robot *pour dessiner les trois cercles tangents de la figure 2.6 ci-dessous ?* □

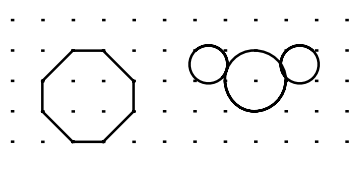


FIG. 2.6 – *Octogone et cercles tangents de tailles différentes*

Corrigé 2 *En essayant sur l'ordinateur on trouve le chiffre neuf.* □

6. Une séquence de commandes est une succession ou suite de commandes données dans un certain ordre.

Corrigé 3 *La lettre F.* □

Corrigé 4 *La lettre T majuscule de la figure 1.6 (page 8) est obtenue en donnant successivement au Robot ⏏ les commandes suivantes :*

```
av; av; av; av; pqtd; av; pqtd; pqtd; av; av;
```

c'est-à-dire en tapant successivement :

A A A A + A + + A A. □

Corrigé 5 *Le chiffre 4 d'une montre digitale est obtenu en donnant au Robot ⏏ la séquence de commandes suivante :*

```
av; av; pqtd; pqtd; av; pqtd; av; pqtd; av;
```

c'est-à-dire en tapant successivement :

A A + + A + A + A. □

Corrigé 6 *L'octogone de la figure 2.6 (page 23), est obtenu en donnant au Robot la séquence de commandes ci-dessous :*

```
av; phtd; av; phtd; av; phtd; av; phtd; av; phtd; av;
phtd; av; phtd; av;
```

c'est-à-dire en tapant successivement :

A D A D A D A D A D A D A D A. □

Corrigé 7 *Les cercles tangents de la figure 2.6 (page 23), sont obtenus en donnant au Robot la séquence de commandes ci-dessous :*

```
phtd;
vd; vd; vd; vd; vd; vd;
phtg;
vg; vg; vg; vg; vg; vg;
phtg;
vd; vd; vd; vd;
```

c'est-à-dire en tapant successivement :

D
))))))
G
((((((
G
)))). □

Addenda

Dans ce chapitre nous avons appris à piloter le $\mathcal{R}^{o}_{b}{}^{o}t$ de manière interactive. Le *mode interactif* est le mode de fonctionnement de nombreux programmes d'ordinateur que l'on appelle des *interpréteurs*. Ces programmes engagent un dialogue avec l'utilisateur en utilisant un langage de communication que l'on appelle le *langage de commande*. L'utilisateur dirige le travail de l'interpréteur en lui donnant des commandes au moyen du clavier de l'ordinateur, de la souris, etc. Les exemples sont très nombreux. Citons les éditeurs de texte, les tableurs, le 'Finder' du Macintosh ou l'interpréteur de commandes des systèmes d'exploitation [a] (comme MS-DOS sur IBM-PC ou Unix sur les stations de travail scientifiques), les systèmes transactionnels d'interrogation de bases de données (comme celles qui sont interrogées par minitel). L'avantage évident de cette interaction avec l'ordinateur est que l'utilisateur peut corriger ses erreurs dès qu'elles se produisent. L'inconvénient majeur est que les tâches répétitives deviennent très vite fastidieuses.

Pour notre $\mathcal{R}^{o}_{b}{}^{o}t$, le langage de commande se réduit à quelques commandes très simples données au clavier (A , B , C , D , G , L , P , + , − , (,) , . , * , ⟵) sans oublier la souris ou, à défaut, les flèches de défilement qui sont des moyens très rapides pour désigner un point précis sur l'écran. Le $\mathcal{R}^{o}_{b}{}^{o}t$ ne communique avec l'utilisateur que par des « bips-bips » lui signalant qu'il ne doit pas le faire sortir du cadre. Les erreurs de dessin peuvent être corrigées par la touche d'effacement ⟵ qui annule l'effet des dernières commandes données au $\mathcal{R}^{o}_{b}{}^{o}t$.

Pendant le dessin interactif, le $\mathcal{R}^{o}_{b}{}^{o}t$ engendre un programme PASCAL qui peut ensuite être exécuté pour reproduire et imprimer le dessin. Ceci peut être comparé aux robots de peinture de l'industrie automobile. Un ouvrier spécialisé peint une voiture au pistolet. Tous ses mouvements sont enregistrés sur ordinateur puis utilisés pour engendrer un programme. En exécutant ce programme, l'ordinateur peut ensuite commander un robot disposant d'un bras manipulateur portant un pistolet pour reproduire fidèlement tous ces mouvements et peindre indéfiniment des voitures. Il s'agit d'un exemple rudimentaire d'*apprentissage*, un thème de recherche actif en *intelligence artificielle* où l'ordinateur engendre son propre programme pour réaliser certaines tâches à partir d'exemples qui lui sont fournis en plus ou moins grand nombre.

---

[a] Les systèmes d'exploitation sont des programmes qui gèrent l'organisation interne et l'affectation des ressources des ordinateurs.

Le dessin interactif sur ordinateur avec le $\mathcal{R}^{o}\underset{\frown}{b}^{o}t$ n'est pas plus difficile et est certainement plus formateur que n'importe quel jeu vidéo. Il apprend aux enfants à utiliser un *formalisme* simple. Il demande un effort d'imagination pour s'orienter dans le plan puisque les commandes ne sont pas relatives à l'observateur mais au $\mathcal{R}^{o}\underset{\frown}{b}^{o}t$. Cependant la tâche devient vite pénible pour les dessins géométriques répétitifs. Seule la programmation, que nous abordons dans le prochain chapitre permet d'obtenir de tels dessins sans peine.

3

Programmation

Jusqu'à présent nous avons utilisé l'ordinateur pour piloter le *Robot* en mode *interactif* ou *conversationnel*. Dans ce mode de fonctionnement nous tapons sur une touche du clavier de l'ordinateur, qui transmet une commande au *Robot* que celui-ci exécute immédiatement à moins que par erreur nous tentions de le faire sortir du cadre auquel cas il répond par un « bip-bip ». Nous recommençons ainsi, en corrigeant les fautes de pilotage au fur et à mesure que le dessin se construit, jusqu'à ce que nous indiquions à l'ordinateur que le dessin est terminé.

Alors que la commande interactive du *Robot* correspond à l'exécution immédiate d'une tâche, la commande programmée du *Robot* consiste à séparer la description de la tâche de ses exécutions ultérieures.

Plus généralement, l'idée de la *programmation* est la suivante : pour faire quelque chose, le *programmeur* écrit un *programme* (dans un certain *langage*) qui décrit avec précision comment effectuer cette chose. Ensuite il demande à un *ordinateur* d'exécuter le programme, en suivant exactement les instructions données par le programme. Autrement dit, au lieu de faire la chose lui-même tout de suite, le programmeur explique à un ordinateur comment le faire à sa place plus tard! On évite ainsi d'avoir à réaliser des tâches longues et répétitives, à condition de savoir décrire ces tâches convenablement.

S'il s'est trompé, le programmeur doit corriger ses fautes en modifiant le programme. C'est ce qu'on appelle *déboguer* le programme. Le *déboguage* du programme consiste à trouver puis à éliminer les *bogues* c'est-à-dire les erreurs se manisfestant à l'exécution par des anomalies de fonctionnement (comme

par exemple `pqtg` au lieu de `pqtd` ou `st` oublié à la fin du programme)[1]. Le principe de fonctionnement des ordinateurs commandé par un programme a des analogies avec de nombreuses activités humaines comme le montre la table ci-dessous :

Programmeur	*Programme*	*Ordinateur*	*Dessin*
Chef-cuisinier	Recette de cuisine	Cuisiniers	Repas
Compositeur	Œuvre musicale	Musiciens	Concert
Auteur	Texte d'une pièce de théâtre	Acteurs	Représentation théâtrale
Chorégraphe	Chorégraphie	Danseurs	Ballet
Réalisateur	Film	Projecteur cinématographique	Séance de cinéma
Tisserand	Cartes perforées	Métier à tisser jacquard	Tissu jacquard
Chef de piste	Plan du parcours	Cavalier et cheval	Parcours de saut d'obstacles

TAB. 3.1 – *Exemples d'activités programmées*

Nous allons maintenant apprendre à programmer les ordinateurs en PASCAL. Le pilotage du $\mathcal{R}^{o}_{b}{}^{o}t$ pour faire des dessins sera notre principale application. Pour faire nos dessins dans ce mode de fonctionnement programmé, nous procédons comme suit :

1. Nous commençons par écrire un programme PASCAL où nous indiquons quelle est la séquence de commandes que le $\mathcal{R}^{o}_{b}{}^{o}t$ devra exécuter pour réaliser le dessin que nous avons en tête ou que nous avons préalablement dessiné sur une feuille de papier quadrillé. Ce programme est écrit avec l'aide de l'ordinateur qui le conserve sur disque ou sur disquette.
2. Ensuite nous demandons à l'ordinateur de *compiler* ce programme PASCAL, c'est-à-dire de le traduire dans son *langage machine*. Le programme ainsi traduit s'appelle le *programme compilé*. Le programme

1. Nous utilisons le vocabulaire recommandé par la commission ministérielle de terminologie [7]. La bogue est la traduction homonymique en français du mot américain *bug* qui désigne les petits insectes nuisibles et plus familièrement les microbes et autres virus aussi bien que les petites choses qui clochent comme les erreurs dans les programmes. La bogue désignant en français l'enveloppe piquante de la châtaigne, l'idée de désagrément est conservée. L'enveloppe fait également penser à la coquille typographique. La traduction ne garde pas l'idée de petit grain de sable aux conséquences dramatiques du mot américain.

compilé et le programme PASCAL original sont équivalents en ce sens qu'ils décrivent exactement la même séquence de commandes pour le *Robot*. Ils sont simplement écrits dans des langages différents.

3. Enfin nous demandons à l'ordinateur d'*exécuter* le programme compilé. Ce faisant, l'ordinateur pilote le *Robot* en lui demandant d'exécuter les commandes dans l'ordre où nous les avons inscrites dans le programme PASCAL original.

Exemple 4 (Carré) Soit à dessiner le carré ci-contre. Le programme PASCAL sera le suivant :

```
program Carre;
  uses  ;
  { Dessiner un carré }
begin
  av; av; pqtd; av; av; pqtd; av; av; pqtd; av; av; pqtd;
  st;
end.
```

Quand il exécute ce programme après l'avoir compilé, l'ordinateur transmet au *Robot* l'ordre d'exécuter la séquence de commandes `av` puis `av` puis `pqtd` puis `av` ... et enfin `pqtd` exactement comme si nous avions tapé au clavier [A] puis successivement [A], [+], [A], [A], [+], [A], [A], [+], [A], [A] et enfin [+]. La dernière commande `st` indique à l'ordinateur qu'il doit attendre que nous tapions sur la touche de retour à la ligne [⟵] pour terminer l'exécution du programme. Ceci nous laisse le temps de contempler à loisir le dessin du *Robot* avant qu'il ne disparaisse. Le dessin n'est pas perdu puisqu'il suffit d'exécuter à nouveau le programme pour le reproduire. □

On constate sur ce premier exemple que le language PASCAL, bien que conçu par le professeur suisse allemand Niklaus Wirth de la Eidgenössische Technische Hochschule à Zürich, utilise des mots anglais comme '**program**' (qui signifie 'programme'), '**uses**' (qui signifie 'utilise' et se prononce "iouseuzzz" avec l'accent anglais évidemment), '**begin**' (qui signifie 'début' et se prononce très approximativement "biguinne") et '**end**' (qui signifie 'fin' et qui, Outre-Manche, se prononce "ainde"). L'informatique conserve les traces de son origine anglo-saxonne !

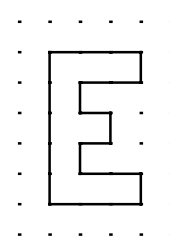
Pour l'instant il nous suffit de savoir que l'ordinateur qui exécute ce programme demande au *Robot* d'exécuter les commandes figurant entre le **begin** et le **end**, les unes après les autres. Les points-virgules qui séparent ces commandes peuvent être compris comme signifiant 'puis'.

Exercice 8 *Exécuter le programme* `Dessiner` *de commande interactive du Robot et lui donner les ordres décrits par le programme* Pascal *suivant (la correspondance entre les touches du clavier et les commandes du Robot est donnée par la table 2.6 page 22) :*

```
program Inconnu;
  uses   ;
begin
  phtd; av; av; av; av;
  phtg; av; av; av; av;
  pqtd; av; av; av; av;
  pqtd; phtd; av; av; pqtg; av; av; phtd; av; av; av; av;
  pqtd; phtd; av; av; pqtg; av; av; phtd; av; av; av; av;
  st;
end.
```

Quel dessin avez-vous demandé au Robot de réaliser ? □

Exercice 9 *Nous voudrions que le programme ci-dessous dessine la lettre* `E` *majuscule ci-contre :*



```
program LettreEavecBogues;
  uses   ;

  { Dessiner une lettre E majuscule épaisse }

begin
  av; av; av; av; av; pqtd;
  av; av; av; pqtd; av; pqtd; av; av; pqtd;
  av; pqtg;
  av; pqtd; av; pqtd; av; pqtg;
  av; pqtg;
  av; av; pqtd; av; pqtd; av; av;
  st;
end.
```

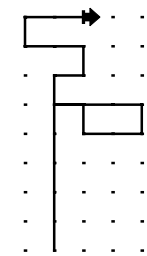
Ce programme erroné contient deux bogues. Pouvez-vous les trouver ? □

Exercice 10 *Exécuter le programme* Pascal `Stegosaure` *figurant sur la disquette d'accompagnement de ce livre.* □

Corrigé 8 *Une cocotte en papier.* □

Corrigé 9 *Le programme* `LettreEavecBogues` *de la page 30 contient deux bogues :*

1. *la troisième commande* `pqtd` *de la deuxième ligne devrait être une commande* `pqtg`,
2. *il manque une commande* `av` *à l'avant-dernière ligne.*



Le résultat de l'exécution de ce programme est reproduit ci-contre (le Robot étant placé dans sa position finale). Le programme débogué qui dessine correctement la lettre `E` *épaisse est le suivant (les corrections sont soulignées) :*

```
program LettreE;
  uses   ;

  { Dessiner une lettre E majuscule épaisse }

begin
  av; av; av; av; av; pqtd;
  av; av; av; pqtd; av; pqtd; av; av; pqtg;
  av; pqtg;
  av; pqtd; av; pqtd; av; pqtg;
  av; pqtg;
  av; av; pqtd; av; pqtd; av; av; av;
  st;
end.
```

□

Corrigé 10 *La façon d'exécuter le programme* Pascal `Stegosaure` *dépend de l'ordinateur utilisé :*

– *Sur le Macintosh, il faut cliquer deux fois rapidement sur l'icône* `Stegosaure.p` *puis choisir l'option '*`Exécuter`*' du menu '*`Compil.`*'.*

– *Sur les compatibles IBM PC, le programme* `Stegosaure` *étant rangé dans le fichier* `Stegosaure.pas`, *taper* `tpc Stegosaure` *pour compiler le programme* Pascal *en un programme compilé* `Stegosaure.exe` *que l'on exécute en tapant* `Stegosaure`.

□

Addenda

Nous devons à Seymour Papert, l'auteur de LOGO [29], la diffusion de l'idée d'utiliser le dessin pour apprendre les concepts de base de la programmation. Cependant l'introduction d'un quadrillage nous évite l'usage des angles et des longueurs comptées en pixels [a]. Le quadrillage permet de tracer très facilement des arcs de cercles, ce qui n'est pas immédiat en LOGO.

L'idée d'utiliser un quadrillage pour faire des dessins géométriques n'est pas nouvelle. On la connaît à l'école depuis longtemps (voir [37] par exemple). On la retrouve, à la base d'exercices de programmation graphique, dans de nombreux ouvrages informatiques de haut niveau comme en TEX [b] par exemple (voir [21] pages 389–390). Le quadrillage élimine les problèmes de représentation discrète sur un écran fini de concepts mathématiques continus. Par conséquent nos segments ressemblent à des segments tels qu'on les dessine sur le papier (et non pas à des traits juxtaposés en escalier), nos cercles ressemblent à des cercles (et non pas à des carrés comme cela arrive sur les écrans graphiques pour des cercles trop petits). Certes l'emploi de grands carreaux limite les possibilités de dessin. La programmation n'en est que plus facile si nous tirons avantage des formes géométriques pour induire la structure des programmes. Une fois que les notions de base seront acquises nous pourrons faire varier la taille de la grille de sorte qu'en passant à la limite fixée par les pixels de l'écran nous retrouverons toutes les possibilités offertes par les écrans graphiques actuels à haute résolution.

Dans le chapitre suivant nous commençons à écrire nos premiers programmes PASCAL. Les algorithmes [c] restent très simples puisqu'ils consistent à codifier le mouvement de la main qui dessine.

---

[a] Le *pixel* est le plus petit élément de l'écran graphique qui peut être allumé (blanc), éteint (noir) ou colorié avec des luminosités et des couleurs plus ou moins nombreuses selon la qualité de l'écran.

[b] TEX ou plutôt sa variante LATEX [22] est le langage de programmation que nous avons utilisé pour composer ce livre.

[c] Un *algorithme* est la description d'un enchaînement d'opérations sur des objets pour accomplir une tâche. Ce mot a pour origine la version latine du nom du mathématicien et astronome persan al-Khwarizmi du IX$^{\text{ème}}$ siècle de notre ère. Son traité *Al-Jabr wa l-muqabala* enseigne comment résoudre algorithmiquement les équations du premier et second degré à cœfficients numériques. Le terme d'*algèbre* dérive du titre même de ce traité.

4

Écriture, compilation et exécution de programmes

4.1 Structure des programmes

De même que pour écrire correctement en français il faut respecter des règles précises d'orthographe et de grammaire, l'écriture de programmes corrects est soumise à un certain nombre de *règles de syntaxe* heureusement plus simples que pour le français. La différence est que des phrases françaises comportant des fautes d'orthographe ou de grammaire peuvent rester compréhensibles par des francophones alors qu'un programme comportant des *erreurs syntaxiques* ne peut pas être compilé, c'est-à-dire traduit dans le langage de l'ordinateur. Cette compilation est effectuée par un programme (appellé *compilateur*) prévu pour indiquer nos erreurs mais pas pour les corriger. Pour corriger nos erreurs de manière satisfaisante, le compilateur devrait pouvoir comprendre nos intentions, un problème bien difficile.

Le programme `Cocotte` ci-dessous est un exemple de programme Pascal correct.

Exemple 5 (Programme syntaxiquement correct)

```
program Cocotte;
  uses  ;
  { Dessiner une cocotte en papier tournée vers l'ouest }
begin
  { Arrière } phtd; av; av; av; av; phtg; av; av; av; av;
  { Dessus }  pqtd; av; av; av; av; pqtd; phtd;
  { Avant }   av; av; pqtg; av; av; phtd; av; av; av; av;
```

```
  { Dessous } pqtd; phtd; av; av; pqtg; av; av; phtd; av; av; av; av;
  { Arrêt du robot } st;
end.
```

□

Tous les programmes PASCAL ont la même forme :

Programme PASCAL *:*

```
program ␣ NomDuProgramme;
 uses   ;
begin
 ... Instructions du programme (séparées par des ';') ...
end.
```

Un programme PASCAL doit commencer par le mot anglais **program**. Ensuite, on indique le `NomDuProgramme` qui est impérativement suivi d'un point-virgule ';'.

Le `NomDuProgramme` est choisi par le programmeur à sa convenance, de façon à évoquer ce à quoi sert son programme. La seule règle à respecter est que le `NomDuProgramme` ne doit comporter que des lettres minuscules ou majuscules non accentuées[1], des chiffres ou des soulignés '_'[2] et doit commencer par une lettre. Par exemple `S`, `s`, `LeStegosaure` ou `Le_Stegosaure` sont des noms corrects de programmes alors que `1S`, `s!`, `Le Stegosaure` ou `Le-Stgosaure` ne respectent pas la règle d'écriture des noms de programmes.

Il y a obligatoirement un blanc entre le mot **program** et le `NomDuProgramme`. Dans un livre il y a des blancs partout et même, en y regardant bien, plus de blanc que toute autre chose. Mettre un blanc dans un livre pour dire qu'il faut taper un blanc est donc idiot parce que la plupart des lecteurs ne s'intéressent pas aux blancs dans les livres mais simplement à ce qui y est écrit. Nous utiliserons donc le symbole '␣' pour écrire nos blancs noir sur blanc ! Autrement dit nous écrivons ␣ pour dire qu'il faut taper sur la grande barre située en bas du clavier de l'ordinateur qui s'appelle la *barre d'espacement* pour obtenir un blanc ' ' dans le programme. Nous ne le faisons pas dans les programmes donnés en exemple qui sont présentés

1. Les accents n'existent pas en anglais (à de très rares exceptions près, comme dans les mots 'naïve', 'protégé' ou 'rôle' empruntés au français).
2. Le souligné '_' ne doit pas être condondu avec le tiret '-'.

comme ils apparaissent à l'écran. De même, à la fin d'une ligne il faut taper sur la touche de retour à la ligne ⏎ pour passer au début de la ligne suivante. Nous ne l'écrivons pas dans les livres car tous les lecteurs savent qu'il faut passer au début de la ligne suivante quand ils sont arrivés en fin de ligne. Certains ordinateurs ne font rien quand on tape une commande tant que l'on n'a pas tapé sur la touche de retour à la ligne. Ne vous êtes-vous jamais laissé surprendre en train d'attendre que l'ordinateur fasse quelque chose alors qu'ayant oublié de taper sur la touche de retour à la ligne, l'ordinateur attendait également que vous réagissiez

Après la première ligne du programme figure une ligne commençant par **uses** (qui signifie 'utilise' en anglais). Cette ligne sert à indiquer que nous utilisons des commandes pour le $\mathcal{R}^{o}\!\!\smile^{o}t$ dans le programme. Elle dépend du type d'ordinateur utilisé. Son contenu exact est fourni sur la disquette d'accompagnement du livre.

Entre le **begin** et le **end**, on trouve les instructions du programme dans l'ordre où elles doivent être exécutées par l'ordinateur. Ces instructions sont séparées par des points-virgules. Les instructions du programme peuvent être des commandes du $\mathcal{R}^{o}\!\!\smile^{o}t$: `av`, `vg`, `vd`, `pqtd`, `pqtg`, `phtd`, `phtg`, `lc`, `bc`, `dc`, `dp`, `dg` ou `st`. En général les programmes de pilotage du $\mathcal{R}^{o}\!\!\smile^{o}t$ se terminent par la commande `st` qui permet d'attendre que nous tapions sur la touche de retour à la ligne avant que le dessin ne disparaisse à la fin de l'exécution du programme.

Finalement, le texte d'un programme Pascal se termine par **end** suivi d'un point final '.'.

Les textes entre accolades `{` et `}` sont des *commentaires* ignorés par l'ordinateur.

Le programme `Cocotte-erron` ci-dessous contient de nombreuses erreurs de syntaxe très courantes chez les débutants.

Exemple 6 (Programme syntaxiquement incorrect)

```
programCocotte-erron;
  { Dessiner une cocotte en papier tournée vers l'ouest
begin
  { Arrière } phtd; av; av; av; av; pgtg; av; av; av; av;
  { Dessus }  pqtd; av; av; av; av; pqtd; phtd;
  { Avant }   av; av; pqtg; av; av; phtd; av; av; av; av;
  { Dessous } pqtd; phtd; av; av; pqtg; av; av; phtd; av; av; av; av
  { Arrêt du robot } st;
end
```

Les erreurs de syntaxe dans ce programme sont les suivantes :

1. *le blanc* ␣ *a été oublié entre* **program** *et le nom du programme* `Cocotte−erron` *;*
2. *le nom du programme* `Cocotte−erron` *contient un signe moins* − *et une lettre accentuée ;*
3. *la ligne '*`uses ;`*' a été oubliée ;*
4. *l'accolade fermante* `}` *a été oubliée à la fin du premier commentaire ;*
5. *l'instruction* `pgtg` *n'est pas une commande du* $\mathcal{R}\breve{o}\underset{\frown}{b}\breve{o}t$ *;*
6. *un point-virgule manque après la dernière commande* `av` *de la ligne commençant par le commentaire* `{ Dessous }` *;*
7. *le point final a été oublié.*

□

Un exemple compliqué de programme Pascal syntaxiquement correct est donné ci-dessous :

Exemple 7 (Le stégosaure) Le programme Pascal de pilotage du $\mathcal{R}\breve{o}\underset{\frown}{b}\breve{o}t$ pour dessiner le stégosaure représenté à la figure 1.2 (page 6) est le suivant :

```
program Stegosaure;
 uses ;
begin
 { Rejoindre l'extrémité de la queue }
  { En bas }
  lc; pqtd; pqtd; av; av; av; av; av; av; av;
  { A droite }
  pqtg; av; av; av; av; av; av; av; av; av; av; av; av; av; pqtg; bc;
 { Dessus de la queue }
  av;
  pqtd; vg; pqtg; vg; pqtd; pqtd; { 1ère épine caudale }
  vg;
  pqtd; pqtd; vg; pqtg; vg; pqtd; { 2ème épine caudale }
  vd; vg; av;
 { Dessiner les 11 plaques osseuses dorsales d'arrière en avant }
  { 1 }  vd; av; av; av; pqtd; av; pqtd; av; av; av; av; pqtd; vd; av;
  { 2 }  pqtg; vd; av; av; av; pqtd; av; pqtd; av; av; pqtd;
            lc; av; bc;
  { 3 }  vd; av; av; pqtd; av; pqtd; av; pqtd; lc; av; bc;
  { 4 }  vd; pqtg; av; pqtd; av; pqtd; av; av; pqtd; av; av; pqtd; vd;
            pqtg; av;
  { 5 }  av; vd; av; pqtd; av; av; pqtd; av; av; pqtd; av; vd; pqtg;
  { 6 }  av; av; pqtd; av; av; pqtd; av; av; pqtd; av; av; pqtd; av;
```

```
            av; pqtg;
    { 7 }  vd; av; pqtd; av; av; pqtd; av; av; pqtd; av; vd; av;
    { 8 }  av; pqtg; vd; pqtd; av; av; pqtd; av; av; pqtd; av; pqtd; av;
            pqtg; vd;
    { 9 }  vg; pqtd; pqtd; av; av; pqtd; av; pqtd; av; pqtd; vg;
    { 10 } vd; vg; pqtd; pqtd; av; av; av; pqtd; av; av; pqtd; av; vd;
            vg;
    { 11 } av; vd; pqtd; av; av; av; pqtd; av; pqtd; av; av; vd;
  { Dessiner la tête }
    { Crane }  av; av;
    { Oeil }   lc; pqtg; av; dc; pqtd; pqtd; av; pqtg; bc;
    { Front }  vg; vd;
    { Dessus du museau } av; vg;
    { Gueule } pqtg; av; av; pqtd; pqtd; av; av; pqtg;
    { Dessous de la tête } vg; av; av; av; av; av; av; av; vg; pqtd; av;
  { Dessiner les pattes antérieures }
    { Patte gauche } pqtg; av; av; av; pqtd; pqtd; av; av; av; av; av;
    pqtg; av; av; pqtg; av; av; av; av; av; av; pqtd; pqtd; av; av; av;
    { Patte droite } av; av; pqtg; av; pqtg; av; av; pqtg; av; pqtd;
    pqtd; av;
  { Dessiner le ventre }
    vg; pqtd; av; av; av;
  { Dessiner les pattes postérieures }
    { Patte gauche } pqtg; av; av; av; pqtd; pqtd; av; av; av; av; av;
    av; av; pqtg; av; av; pqtg; av; av; av; av; av; av; av; av; pqtd;
    pqtd; av; av; av; av; av; av;
    { Patte droite } av; pqtg; av; pqtg; av; av; av; pqtg; av; pqtd;
    pqtd; av;
  { Compléter le dessous de la queue }
  av; pqtd; vg; av; vd; vg; av; pqtd; vg; pqtd;
  st;
end.
```

□

4.2 Commentaires

Un programme Pascal peut comporter des *commentaires* qui sont des textes français quelconques écrits entre accolades `{ … }` ou entre parenthèses-étoiles `(* … *)`. L'ordinateur ne tient pas compte de ces commentaires qui sont uniquement destinés à expliquer l'usage et le principe de fonctionnement du programme aux lecteurs de ce programme. Si l'on enlève tous les

commentaires d'un programme, il devient illisible pour un humain mais rien n'est changé pour l'ordinateur.

Exemple 8 (Le stégosaure illisible) En enlevant tous les commentaires du programme `Stegosaure` de la page 36, on obtient un programme Pascal totalement illisible mais qui dessine parfaitement bien le stégosaure représenté à la figure 1.2 (page 6) :

```
program S; uses ;
begin lc;pqtd;pqtd;av;av;av;av;av;av;av;pqtg;av;av;av;av;av;av;av;av;
av;av;av;av;av;pqtg;bc;av;pqtd;vg;pqtg;vg;pqtd;pqtd;vg;pqtd;pqtd;vg;
pqtg;vg;pqtd;vd;vg;av;vd;av;av;av;pqtd;av;pqtd;av;av;av;av;pqtd;vd;
av;pqtg;vd;av;av;av;pqtd;av;pqtd;av;av;pqtd;lc;av;bc;vd;av;av;pqtd;
av;pqtd;av;pqtd;lc;av;bc;vd;pqtg;av;pqtd;av;pqtd;av;av;pqtd;av;av;
pqtd;vd;pqtg;av;av;vd;av;pqtd;av;av;pqtd;av;av;pqtd;av;vd;pqtg;av;av;
pqtd;av;av;pqtd;av;av;pqtd;av;av;pqtd;av;av;pqtg;vd;av;pqtd;av;av;
pqtd;av;av;pqtd;av;vd;av;av;pqtg;vd;pqtd;av;av;pqtd;av;av;pqtd;av;
pqtd;av;pqtg;vd;vg;pqtd;pqtd;av;av;pqtd;av;pqtd;av;pqtd;vg;vd;vg;
pqtd;pqtd;av;av;av;pqtd;av;av;pqtd;av;vd;vg;av;vd;pqtd;av;av;av;pqtd;
av;pqtd;av;av;vd;av;av;lc;pqtg;av;dc;pqtd;pqtd;av;pqtg;bc;vg;vd;av;
vg;pqtg;av;av;pqtd;pqtd;av;av;pqtg;vg;av;av;av;av;av;av;av;vg;pqtd;
av;pqtg;av;av;av;pqtd;pqtd;av;av;av;av;av;pqtg;av;av;pqtg;av;av;av;
av;av;av;pqtd;pqtd;av;av;av;av;av;pqtg;av;pqtg;av;av;pqtg;av;pqtd;
pqtd;av;vg;pqtd;av;av;av;pqtg;av;av;av;pqtd;pqtd;av;av;av;av;av;av;
av;pqtg;av;av;pqtg;av;av;av;av;av;av;av;av;pqtd;pqtd;av;av;av;av;av;
av;av;pqtg;av;pqtg;av;av;av;pqtg;av;pqtd;pqtd;av;av;pqtd;vg;av;vd;vg;
av;pqtd;vg;pqtd;st;end.
```

Cet exemple montre également que la disposition des instructions du programme est importante. L'espacement entre instructions et les alignements convenables facilitent grandement la lecture des programmes. □

4.3 Compilation des programmes

La *compilation* d'un programme Pascal consiste à le traduire en un *programme compilé* écrit dans le langage de l'ordinateur appelé *langage machine*. La compilation est faite par un programme que l'on appelle un *compilateur*[3]. Bien évidemment le programme Pascal et le programme compilé pilotent le *Robot* de la même façon. Cette manière de procéder nous évite d'avoir à

3. Nous utiliserons les compilateurs Turbo Pascal 5.5 ou 6.0 de Borland sur les ordinateurs compatibles IBM PC et le compilateur Turbo Pascal 1.1 de Borland sur les Macintosh. La façon de compiler les programmes dépend de l'ordinateur utilisé. Elle est expliquée sur la disquette d'accompagnement du livre.

apprendre le langage machine. Il est bien difficile de comprendre ce langage machine car tous les programmes s'écrivent uniquement avec des 0 et des 1 !

4.4 Erreurs syntaxiques

Pour pouvoir compiler un programme PASCAL, il ne faut pas qu'il contienne d'erreurs syntaxiques violant les règles d'écriture des programmes. En cas d'erreur le compilateur explique brièvement la cause de l'erreur et désigne l'endroit du programme où l'erreur de syntaxe a été trouvée. Il convient alors de corriger l'erreur et de recommencer la compilation.

Exemple 9 (Programme syntaxiquement incorrect) Le programme `Cocotte−erron` de la page 35 contient à peu près toutes les erreurs syntaxiques imaginables pour un programme aussi simple. Quand on compile ce programme `Cocotte−erron`, le compilateur commence par indiquer '**pro gram** attendu'. Il lit en effet le mot '**program**`Cocotte` alors qu'il s'attend à trouver le mot **program**. Ayant ajouté un ␣ après le mot **program**, une nouvelle tentative de compilation indique '`;` attendu', l'erreur étant signalée sur le signe moins '−'. Diverses causes d'erreurs sont possibles. Nous aurions pu avoir tapé moins '−' à la place du souligné '_' ou à la place de ';' ou bien par erreur, sans intention de le faire. Le compilateur qui ne peut pas connaître nos véritables intentions, indique simplement la cause d'erreur la plus probable. Ayant remplacé le signe moins '−' par un souligné '_', le compilateur signale maintenant une 'erreur de syntaxe' en désignant le 'é'. Supprimant l'accent, nous avons corrigé la première ligne. Essayant à nouveau de compiler le programme PASCAL, le compilateur signale une erreur par le message '**begin** attendu' en désignant la première commande `phtd` du $\mathcal{R}^{o}_{\smile}b^{o}t$. En fait l'erreur se situe plus tôt, une accolade fermante `}` ayant été oubliée à la fin du commentaire expliquant ce que fait le programme. Une nouvelle tentative de compilation montre qu'il y a une erreur sur la première commande `phtd` du $\mathcal{R}^{o}_{\smile}b^{o}t$ qui est signalée par le message 'identificateur inconnu'. Ayant oublié d'indiquer que nous utilisons les commandes du $\mathcal{R}^{\breve{o}}_{\smile}b^{\breve{o}}t$, le compilateur ignore la signification de `phtd`. Après avoir ajouté la ligne "**uses** ;", nous pouvons tenter une nouvelle compilation. Cette fois-ci nous obtenons exactement le même message 'identificateur inconnu' mais le compilateur désigne `pgtd`. C'est une faute de frappe que nous corrigeons en remplaçant `pgtd` par `phtd`. Un nouveau message '`;` attendu' signale qu'un point-virgule manque

après l'avant-dernier `av` du programme. L'ayant rajouté, le compilateur indique la dernière faute 'Fin du texte prématurée'. Ayant rajouté le point final nous obtenons laborieusement le programme `Cocotte` de la page 33 qui est syntaxiquement correct. □

4.5 Exécution des programmes

L'*exécution* d'un programme consiste à faire réaliser par l'ordinateur les actions définies par les instructions du programme[4]. Comme dans notre cas les instructions sont des commandes du $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$, l'exécution du programme permet de piloter le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$.

Quand le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ dessine trop rapidement pour pouvoir observer le déroulement du dessin, on peut l'interrompre pour le ralentir ou l'arrêter à volonté. Si pendant l'exécution d'un programme de pilotage du $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$, nous tapons sur la *touche d'échappement* [Esc] ou sur la touche marquée du caractère [§] ou sur celle marquée [#], nous pouvons stopper le dessin du $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$, exactement comme si nous demandions à l'ordinateur d'exécuter immédiatement une commande `st`. Par exemple si nous interrompons l'exécution du programme `Stegosaure` de la page 36 au moment où le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ termine le dessin de la patte avant gauche, nous obtenons l'écran représenté en réduction à la figure 4.1 de la page 41.

Une fois le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ stoppé, nous pouvons :

- poursuivre l'exécution du programme normalement en tapant sur la touche de retour à la ligne [↩],
- diminuer (en tapant une ou plusieurs fois sur la touche moins [−]) ou augmenter (en tapant une ou plusieurs fois sur la touche plus [+]) la vitesse du $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ (qui peut varier de 0 à 10) puis reprendre l'exécution du programme avec cette nouvelle vitesse du $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ en tapant sur la touche de retour à la ligne [↩],
- arrêter définitivement l'exécution du $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ en tapant une nouvelle fois sur la touche d'échappement [esc] ou sur l'une des touches [§] ou [#] (ce qui donne l'écran de la figure 4.2 page 41) puis en terminant par un retour à la ligne [↩].

4. La façon de demander à l'ordinateur d'exécuter un programme compilé dépend de l'ordinateur utilisé. Lire les explications avec `LisezMoi` sur la disquette d'accompagnement.

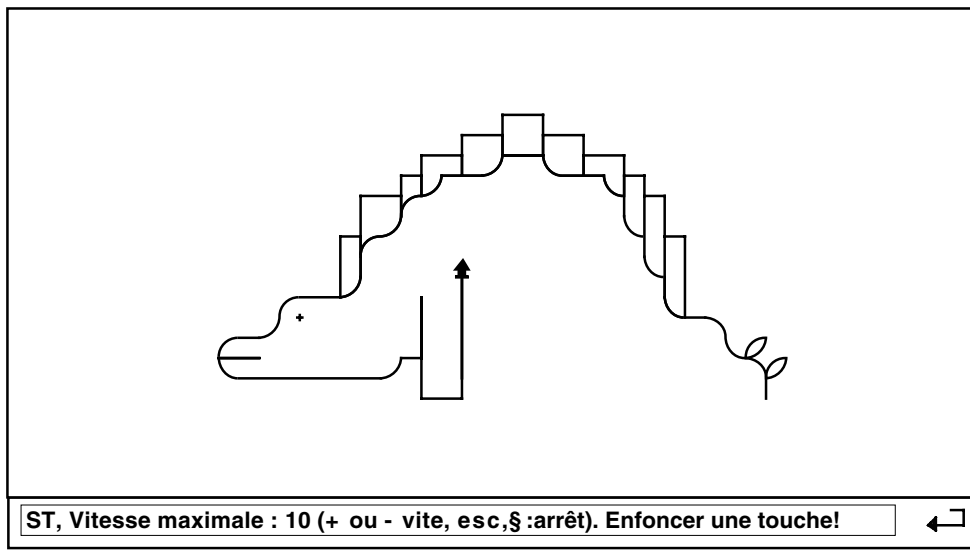


FIG. 4.1 – *Écran de l'ordinateur après un stop*

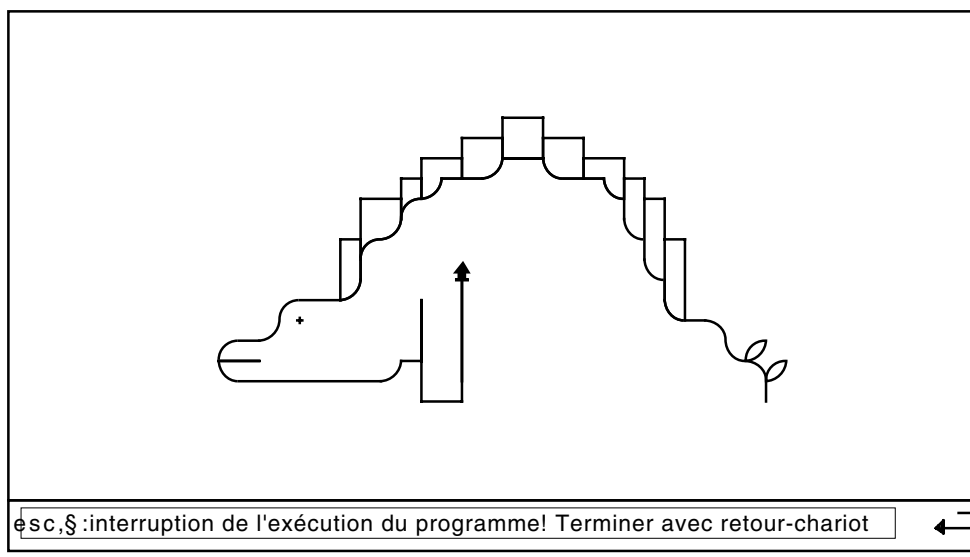


FIG. 4.2 – *Écran de l'ordinateur avant l'arrêt définitif*

4.6 Erreurs logiques

Après avoir éliminé toutes les erreurs syntaxiques c'est-à-dire les erreurs de programmation qui relèvent de la violation des règles de grammaire du langage PASCAL, nous ne sommes pas encore au bout de nos peines avant de pouvoir contempler le dessin ! Il reste encore les *erreurs logiques* que nous découvrirons à l'exécution du programme.

La première erreur possible est que la succession de commandes que notre programme demande à l'ordinateur de donner au *Robot* conduise celui-ci en dehors du cadre. C'est le cas du programme suivant :

```
program SortirDuCadre;
  uses  ;
  { Programme conduisant à une erreur à l'exécution parce qu'il fait }
  { sortir le robot du cadre. }
begin
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
end.
```

Ce programme donne le triste résultat ci-dessous :

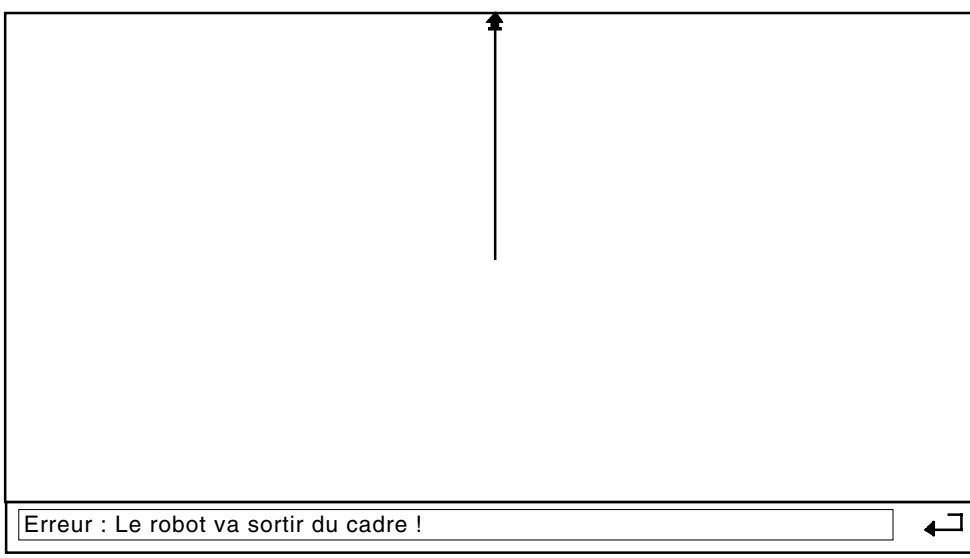


FIG. 4.3 – *Écran de l'ordinateur quand le robot sort du cadre*

La deuxième erreur possible est que nous nous soyons trompés dans le pro-

gramme et que le dessin obtenu ne ressemble pas du tout à ce que nous avions imaginé. Par exemple, le programme `StegosaureErrone` ci-dessous est une version logiquement erronée mais syntaxiquement correcte du programme `Stegosaure` de la page 36.

```
program StegosaureErrone;
  uses   ;
begin

    { 10 } vg; { ERREUR : vd}
               vg; pqtd; pqtd; av; av; av; pqtd; av; av; pqtd; av; vd; vg;

  { Dessiner le ventre }
    vg; pqtg; { ERREUR : pqtd } av; av; av;

end.
```

L'auteur de ce programme `StegosaureErrone` a par deux fois confondu sa gauche et sa droite. Le résultat de l'exécution de ce programme tel que représenté ci-dessous (figure 4.4) étonnerait plus d'un paléontologiste !

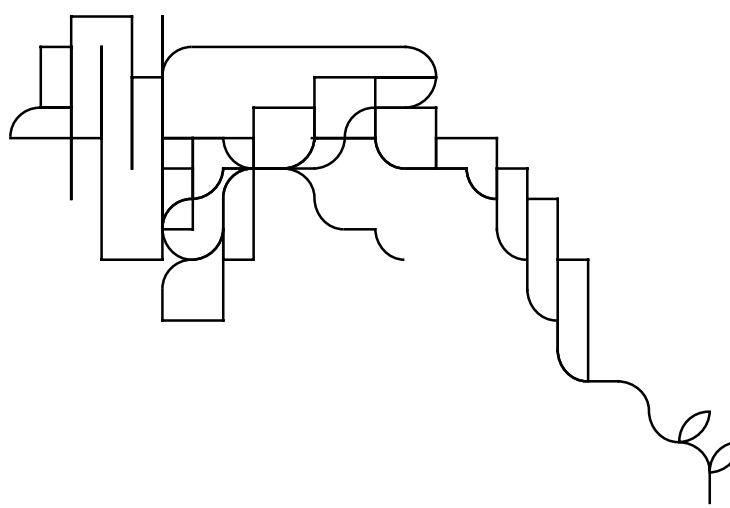


FIG. 4.4 – *Stégosaure plutôt raté*

4.7 Comment localiser les erreurs logiques dans le programme

Pour trouver les erreurs logiques dans le programme il faut s'imaginer que l'on exécute le programme à la place du $\mathcal{R}\breve{o}\breve{b}\breve{o}t$. Quand le programme est très long ceci n'est pas facile. Le mieux est donc d'arrêter l'exécution du programme un peu avant l'erreur, puis de suivre l'exécution du programme en mode *pas à pas*, ce qui permet de suivre les commandes transmises par l'ordinateur au $\mathcal{R}\breve{o}\breve{b}\breve{o}t$ pour localiser l'erreur avec précision. Quand l'erreur est localisée, il faut arrêter l'exécution du programme pour pouvoir la corriger dans le programme. Une fois que c'est fait, on peut compiler et tenter une nouvelle exécution.

- Pour passer en mode pas à pas nous pouvons :
  - taper sur la touche d'échappement [Esc], [§] ou [#] puis réduire la vitesse à zéro en tapant plusieurs fois sur la touche [-] et enfin reprendre l'exécution en mode pas à pas en tapant sur la touche de retour à la ligne [↩];
  - ou insérer une instruction `pp` dans le programme PASCAL, un peu avant l'endroit supposé de l'erreur. Cette instruction `pp` fait passer l'ordinateur en mode d'exécution pas à pas.
- En mode pas à pas, l'ordinateur affiche le nom de la commande du $\mathcal{R}obot$ qu'il va exécuter puis attend que nous lui donnions l'ordre de le faire en tapant sur la touche de retour à la ligne [↩] (ou en cliquant sur le bouton de la souris). On peut donc suivre le déroulement de l'exécution du programme instruction après instruction, ce qui permet en général de trouver assez facilement l'erreur.
- Quand l'erreur est localisée avec précision dans le programme, on peut interactivement arrêter l'exécution du programme en mode pas à pas en tapant sur la touche d'échappement [Esc], [§] ou [#] puis :
  - reprendre l'exécution normale (en tapant sur la touche [+] pour augmenter la vitesse puis sur la touche de retour à la ligne [↩]),
  - ou arrêter l'exécution du programme (en tapant à nouveau sur la touche d'échappement [Esc], [§] ou [#] puis sur la touche de retour à la ligne [↩]).

Il est également possible de programmer le retour à une vitesse d'exécution normale en insérant une instruction `ex` dans le programme ou de prévoir l'arrêt par `st`.

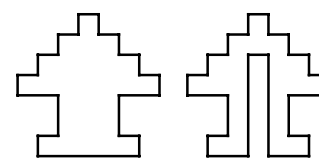
Il faut presque toujours plusieurs essais avant que le programme soit logiquement correct. Quand il l'est, on est souvent tenté de l'améliorer. Le plus difficile est alors de savoir quand s'arrêter !

`st`	stoppe momentanément l'exécution du programme de pilotage du *Robot*. La vitesse peut être diminuée par [-] (minimum 0) ou augmentée par [+] (maximum 10). L'exécution reprend à la vitesse indiquée en tapant sur [⏎] ou en cliquant avec la souris. L'exécution s'arrête définitivement en tapant sur [esc] ou [§] ou [#] puis sur [⏎].
`pp`	exécute la suite du programme en mode pas à pas (vitesse 0). Les commandes du *Robot* s'affichent en bas de l'écran et s'exécutent en tapant sur [⏎].
`ex`	exécute la suite du programme en mode exécution normale (vitesse 10).

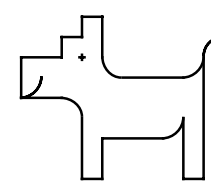
TAB. 4.1 – *Commandes de contrôle du mode de pilotage du Robot*

Exercice 11

– *Écrire, compiler, mettre au point et exécuter des programmes* PASCAL *de dessin du Robot tel qu'il est stylisé ci-contre avec son crayon baissé et son crayon levé.*



– *Écrire, compiler, mettre au point et exécuter un programme* PASCAL *qui pilote le Robot pour lui faire dessiner le chien représenté ci-contre.*



– *Programmer les dessins de la page 9.*

– *Programmer le dessin de vos initiales en lettres majuscules dans le style de la lettre* `H` *en relief représentée à la figure 1.12 (page 9).*

□

Corrigé 11 *Nous donnons un programme, les autres se trouvant sur disquette.*

```
program Chien;
  uses  ;
begin
  { Profil arrière de la patte avant } av; av; pqtd;
  { Ventre } av; av; av; vg;
  { Patte arrière } pqtd; pqtd; av; av; av; pqtg; av; pqtg; av; av; av;
     av;
  { Queue } av; av; vd; pqtd; pqtd; vg; vd;
  { Dos } av; av; av; vd;
  { Tête }
    { Oreille } av; av; pqtg; av; pqtg; av;
    { Oeil } lc; av; dc; pqtd; pqtd; av; bc; pqtg; av; pqtg; av;
    { Museau } pqtd; av; av; pqtg; av; av; pqtg;
    { Gueule } vg; pqtd; pqtd; vd; pqtd; pqtd; av; av; vd;
  { Profil avant de la patte avant } av; av; av; pqtg; av;
  st;
end.
```

□

Addenda

Le cycle écriture, mise au point syntaxique, compilation, mise au point logique et exécution des programmes s'est considérablement raccourci ces dernières années grâce aux progrès fantastiques de l'architecture des ordinateurs qu'il convient de comprendre très schématiquement. Un micro-ordinateur comprend :

- une unité arithmétique et logique pour faire des calculs et manipuler des données,
- une mémoire pour conserver temporairement des données et des programmes (son contenu est perdu quand on coupe l'électricité),
- un ou plusieurs disques durs magnétiques pour conserver en permanence des fichiers contenant des données ou des programmes,
- un ou plusieurs lecteurs de disquettes souples pour conserver en permanence des fichiers contenant des données ou des programmes ou les échanger avec d'autres ordinateurs,
- un clavier pour entrer des données ou des programmes en mémoire,
- un écran graphique pour visualiser des textes et des images,
- une souris pour désigner des points sur l'écran graphique,

- un bus pour permettre des échanges rapides de données et de programmes entre ces différents organes.

Quand on met l'ordinateur en marche, il est cablé pour charger dans sa mémoire et exécuter un programme rangé en permanence sur disque que l'on appelle le *système d'exploitation*. Ce programme a pour tâche essentielle de gérer tous les organes de l'ordinateur et de dialoguer avec l'utilisateur grâce à un *interpréteur de commandes* qui va attendre que l'utilisateur lui donne des ordres par l'intermédiaire du clavier et les exécuter immédiatement les uns après les autres jusqu'à ce que l'utilisateur lui donne l'ordre d'éteindre l'ordinateur. Parmi les commandes de l'interpréteur, citons par exemple la suppression d'un fichier rangé sur disque ou le lancement de l'exécution d'un programme écrit en langage machine (généralement appelé *programme d'application*) comme par exemple un *éditeur de texte* pour créer un texte et le ranger dans un fichier ou un *compilateur* pour traduire un programme PASCAL en un programme machine. Les ordinateurs sont *binaires* en ce sens que toutes les données, en particulier les programmes, sont représentés par des suites de 0 et de 1.

L'ordinateur fonctionne en transférant une instruction du programme depuis une case de sa mémoire dans son unité arithmétique et logique (via le bus) et en l'exécutant (ce qui en général modifie le contenu de la mémoire, du disque, de l'écran, etc.) puis en recommençant ainsi avec l'instruction rangée dans la case suivante de sa mémoire. Certaines instructions dites *instructions de branchement* permettent d'indiquer que l'instruction suivante est rangée dans une case mémoire de numéro donné.

Pour exécuter un programme écrit en langage machine rangé sur disque, le système d'exploitation demande à l'unité arithmétique et logique de charger ce programme en mémoire c'est-à-dire de recopier le programme dans un endroit de la mémoire qui était inoccupé puis d'exécuter une instruction de branchement à la première instruction de ce programme. Le programme doit se terminer par une instruction de branchement en retour vers le système d'exploitation. Les ordinateurs modernes possèdent également la faculté d'interrompre l'exécution du programme en cours puis de la reprendre ultérieurement comme si rien ne s'était passé. Entre temps, c'est le système d'exploitation qui s'exécute pour réagir en fonction de la cause de l'interruption. Par exemple si l'utilisateur a bougé la souris, ses nouvelles coordonnées sont rangées en mémoire. Si l'horloge sonne, ce qui arrive tous les soixantièmes de seconde par exemple, l'heure est mise à jour, etc. En particulier l'utilisateur dispose de boutons ou de touches (Esc, §, # pour notre $\mathcal{R}\mathcal{O}\mathcal{B}\mathcal{O}\mathcal{T}$) permettant d'interrompre l'exécution de son programme s'il constate une erreur.

Parmi les applications que peut exécuter le système d'exploitation, il y en a une qui nous intéresse particulièrement, c'est le compilateur.

Les compilateurs modernes se chargent de l'édition du texte du programme, c'est-à-dire de sa création et de ses modifications par interaction avec l'utilisateur. Le programme PASCAL est affiché sur l'écran et rangé en mémoire au fur et à mesure que l'utilisateur le tape au clavier. Il est prévu qu'il puisse le modifier en utilisant diverses touches de fonctions et la souris. La possibilité lui est également offerte de sauvegarder son texte sur le disque dur dans un fichier dont il doit donner le nom.

Quand l'édition de texte est terminée, le compilateur peut compiler le programme c'est-à-dire le traduire en un programme équivalent écrit en langage machine. Le programme compilé peut être rangé sur disque auquel cas l'utilisateur met fin à l'exécution du compilateur et retourne au niveau du système d'exploitation pour charger en mémoire et exécuter son programme compilé comme expliqué précédemment. Pendant la phase de débogage du programme, il est plus rapide de demander au compilateur de ranger le programme compilé en mémoire, le compilateur se chargeant directement de son exécution puis de reprendre la main quand cette exécution est terminée ou interrompue.

On comprend les progrès considérables qui ont été faits depuis les premiers ordinateurs pour lesquels il fallait modifier le cablage pour changer le programme [23] :

- le programme étant rangé en mémoire (selon l'idée de Von Neumann [a]) et la mémoire reliée à des disques rapides, il est possible de changer le programme de la machine presque instantanément ;
- l'usage de langages de haut niveau comme PASCAL et de compilateurs performants, permet d'éviter d'avoir à programmer en un langage machine ingrat ;
- l'écran alphanumérique permet à l'utilisateur d'éditer son programme de manière interactive au clavier, ce qui fait gagner un temps considérable par rapport aux manipulations de cartes perforées en carton encore utilisées dans les années 1970 ;
- la puissance de calcul, liée à la densité des transitors sur les circuits (qui double tous les 18 mois ce qui laisse prévoir un milliard de composants par puce en l'an 2000), permet de gérer des écrans graphiques directement reliés à la mémoire, ce qui permet des modifications très rapides et donc l'usage de fenêtres, souris, etc ;

[a] Johann von Neumann, 1903–1957, mathématicien américain.

- la taille de la mémoire (quelques millions de caractères) et des disques (quelques milliards de caractères) permet d'envisager des programmes et des données de plus en plus grands et donc des applications de plus en plus sophistiquées.

La taille des plus grands programmes que l'on peut actuellement écrire se chiffre maintenant en millions de lignes, ce qui est considérable et ne va pas sans difficultés car on atteint les limites de la compréhension humaine. Une formation de l'esprit, une nouvelle rigueur, plus précise encore que la logique des mathématiciens, devient la qualité première qui, en plus de l'imagination et du sens de l'esthétique, fait les bons informaticiens. Un peu de méthode aide également beaucoup. Toute l'histoire des langages de programmations est remplie d'inventions permettant de formuler plus clairement la façon d'organiser les données en mémoire ou les calculs de l'ordinateur. A chaque fois les capacités humaines, donc la taille limite des plus grands programmes, ont été repoussées. Une idée très ancienne de structuration des calculs, due à Von Neumann et Turing [a] est celle de *procédure*, on disait autrefois *routine* ou *sous-programme*. Nous commençons à l'introduire dans le chapitre suivant.

[a] Alan Turing, 1912–1954, mathématicien anglais.

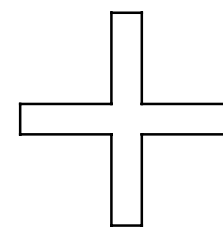
5

Procédures

5.1 Recopie de texte

Dans le programme `CroixGrecque_1` ci-dessous on retrouve quatre fois la même séquence de commandes pour dessiner les quatre branches identiques de la croix et tourner. Chacune se décompose en deux cotés égaux de longueur 3 et une extrémité de longueur 1 (l'unité étant la longueur d'un côté d'un carreau du quadrillage).

```
program CroixGrecque_1;
  uses  ;
begin
  av; av; av; pqtg; av; pqtg; av; av; av; pqtd;
  av; av; av; pqtg; av; pqtg; av; av; av; pqtd;
  av; av; av; pqtg; av; pqtg; av; av; av; pqtd;
  av; av; av; pqtg; av; pqtg; av; av; av; pqtd;
  st;
end.
```



Cette situation où l'on trouve la même séquence de commandes du *Robot* plusieurs fois de suite dans un programme est très fréquente car les dessins géométriques sont souvent composés de morceaux identiques. Il est alors très fastidieux de taper plusieurs fois le même texte. On peut l'éviter en utilisant les possibilités offertes par l'ordinateur de copier un texte déjà enregistré dans sa mémoire et de le recopier ailleurs dans le programme. La manière de procéder est expliquée pour les compatibles IBM PC à la figure 5.1 (page 52) et pour le Macintosh à la figure 5.2 (page 53).

Sélectionner et recopier un texte sur IBM PC *et compatibles*

1. Placer le curseur d'édition au début du texte à recopier en le déplaçant à l'aide des flèches de défilement ([↑], [↓], [←], [→]).
2. Maintenir la touche de contrôle [Ctrl] enfoncée le temps de taper [K] suivi de [B].
3. Placer le curseur à la fin du texte à recopier en le déplaçant à l'aide des flèches de défilement ([↑], [↓], [←], [→]), de sorte que le texte à recopier apparaisse en *vidéo-inverse* c'est-à-dire en blanc sur fond noir.
4. Maintenir la touche de contrôle [Ctrl] enfoncée le temps de taper sur la touche [K] deux fois de suite.
5. Déplacer le curseur à l'aide des flèches de défilement ([↑], [↓], [←], [→]) à l'endroit où le texte doit être recopié.
6. Maintenir la touche de contrôle [Ctrl] enfoncée, puis taper [K] suivi de [C].

L'opération de recopie ([Ctrl] [K] [C]) en un endroit quelconque du programme peut être répétée autant de fois que nécessaire (sauf à l'intérieur de la zone recopiée).

FIG. 5.1 – *Marquage et copie de bloc sur les compatibles IBM PC*

Sélectionner et recopier un texte sur Macintosh

1. Placer le curseur d'édition clignotant au début du texte à recopier en le déplaçant à l'aide des flèches de défilement (↑, ↓, ←, →) ou bien en cliquant avec la souris à l'endroit désiré.
2. Maintenir la touche majuscules ⇧ enfoncée, sans la relâcher.
3. Placer le curseur d'édition clignotant à la fin du texte à recopier en le déplaçant à l'aide des flèches de défilement (↑, ↓, ←, →) ou bien en cliquant avec la souris à l'endroit désiré, de sorte que le texte à recopier passe en *vidéo-inverse* c'est-à-dire en blanc sur fond noir (si par erreur vous tapez du texte au clavier, il remplace le texte sélectionné en vidéo-inverse ; la touche d'effacement ← supprime le texte sélectionné en vidéo-inverse ; dans les deux cas, choisir l'option `Annuler` du menu `Edition` pour rétablir l'original).
4. Relâcher la touche majuscules ⇧.
5. Sélectionner l'option `Copier` du menu `Edition` (ou de manière équivalente maintenir la touche commande ⌘ enfoncée et taper sur la touche C).
6. Placer le curseur d'édition clignotant à l'endroit où le texte doit être recopié en le déplaçant à l'aide des flèches de défilement (↑, ↓, ←, →) ou bien en cliquant à l'endroit désiré avec la souris.
7. Sélectionner l'option `Coller` du menu `Edition` (ou de manière équivalente maintenir la touche commande ⌘ enfoncée et taper sur la touche V).

L'opération de recopie (`Coller`) en un endroit quelconque du programme peut être répétée autant de fois que nécessaire.

FIG. 5.2 – *Copier-coller sur le Macintosh*

Faire beaucoup de recopies à la main est long. De plus on imagine bien que dans un programme de plusieurs pages, le lecteur a du mal à retrouver les parties de texte identiques. Ce phénomème est illustré dans le programme `Nombre888_1` ci-dessous. La difficulté serait évidemment encore plus grande si les parties de texte identiques se trouvaient sur des pages différentes.

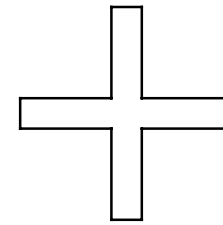
```
program Nombre888_1;
  uses  ;

{ Dessiner le nombre 888 dans le }
{ style des montres digitales.   }

begin
  { Dessiner le 8 des centaines }
  av; pqtd; av; pqtg; av; pqtg; av; pqtg; av; pqtg; av; pqtd; av; pqtd;
  av;
  { Prévoir un espace et réorienter le robot vers le nord }
  lc; pqtd; pqtd; av; av; pqtg; bc;
  { Dessiner le 8 des dizaines }
  av; pqtd; av; pqtg; av; pqtg; av; pqtg; av; pqtg; av; pqtd; av; pqtd;
  av;
  { Prévoir un espace et réorienter le robot vers le nord }
  lc; pqtd; pqtd; av; av; pqtg; bc;
  { Dessiner le 8 des unités }
  av; pqtd; av; pqtg; av; pqtg; av; pqtg; av; pqtg; av; pqtd; av; pqtd;
  av;
  st;
end.
```

5.2 Exemples de procédures

Plutôt que de faire des recopies de textes à la main soi-même, on peut demander au compilateur Pascal de le faire à notre place en utilisant des *procédures*. Commençons par réécrire le programme `CroixGrecque_1` (page 51) en utilisant des procédures pour mettre en évidence la décomposition de la croix grecque en quatre branches identiques, chacune d'elles se décomposant en deux cotés égaux de longueur 3 et une extrémité de longueur 1.



```
program CroixGrecque_2;
 uses  ;

 procedure Cote;
 begin
  av; av; av;
 end;

 procedure Extremite;
 begin
  pqtg; av; pqtg;
 end;

 procedure Branche;
  { Dessiner le côté gauche, l'extrémité puis le côté droit d'une }
  { branche, et orienter le robot en direction de la branche sui- }
  { vante.                                                        }
 begin
  Cote; Extremite; Cote; pqtd;
 end;

 procedure Croix;
 begin
  Branche; Branche; Branche; Branche;
 end;

begin
 Croix; st;
end.
```

L'emploi d'une procédure comprend :

- la *déclaration de procédure* qui définit une séquence d'instructions (appelée *corps de la procédure*) et lui donne un nom (appelé *nom de la procédure*) ;
- l'*appel de procédure* qui indique que la séquence d'instructions constituant le corps de la procédure doit être recopiée à la place du nom de la procédure.

Pour le lecteur du programme, l'utilisation de procédures est préférable aux recopies de textes. A chaque fois qu'il voit le nom d'une procédure, il comprend qu'il s'agit de la copie d'un texte qu'il peut consulter dans la déclaration de procédure. S'il rencontre plusieurs appels d'une même procédure, il n'a pas à faire d'effort pour comprendre immédiatement qu'il s'agit du même texte qui est recopié. Le programme est donc plus clair. N'est-ce pas le cas

du programme `Nombre888_2` ci-dessous qui est plus facilement compréhensible que le programme `Nombre888_1` ?

```
program Nombre888_2;
 uses  ;

{ Dessiner le nombre 888 comme }
{ sur les montres digitales.   }

 procedure Huit;
  { Dessiner  le  chiffre  8, en partant en bas à gauche du 8 avec le }
  { robot orienté au nord et en arrivant en bas à gauche du 8 avec le }
  { robot orienté à l'ouest. }
 begin
  av; pqtd; av; pqtg; av; pqtg; av; pqtg; av; pqtg; av; pqtd; av; pqtd;
  av;
 end;

 procedure Espace;
  { Partant  avec  le  robot  orienté à l'ouest en bas à gauche du 8, }
  { reculer en bas à droite du 8 puis d'un espace  et  réorienter  le }
  { robot au nord. }
 begin
  lc; pqtd; pqtd; av; av; pqtg; bc;
 end;

begin
 { Dessiner le 8 des centaines } Huit; Espace;
 { Dessiner le 8 des dizaines  } Huit; Espace;
 { Dessiner le 8 des unités    } Huit; st;
end.
```

5.3 Déclaration de procédures

Les déclarations des procédures d'un programme se placent entre **uses** et **begin**. La structure d'un programme PASCAL est donc la suivante :

Programme PASCAL *:*

program ␣ `NomDuProgramme`;
uses ;
... Déclarations des procédures du programme ...
begin
... Instructions du programme[a] (séparées par des ';') ...
end.

[a] Y compris les appels des procédures du programme.

En PASCAL, une déclaration de procédure a la forme suivante (nous utiliserons des formes plus élaborées dans la suite) :

Déclaration de procédure :

procedure ␣ `NomDeLaProcedure`;
begin
... Instructions du corps de la procédure[a] (séparées par des ';') ...
end;

[a] Y compris les appels de procédures précédemment déclarées.

Le `NomDeLaProcedure` est choisi par le programmeur à sa convenance, de façon à évoquer ce à quoi sert cette procédure. On notera qu'il faut un point-virgule ';' après le `NomDeLaProcedure`, qu'il n'en faut pas après le **be gin**, mais qu'il en faut un après le **end** qui termine le corps de la procédure.

5.4 Identificateurs

Le nom d'un programme ou d'une procédure s'appelle un *identificateur* en PASCAL. En voici une définition précise à laquelle il faudra se référer à

chaque fois que nous parlerons d'identificateurs :

Identificateur :

Un *identificateur* est une séquence de lettres minuscules ou majuscules non accentuées, de chiffres ou de soulignés '_' qui commence obligatoirement par une lettre et qui est différente des mots anglais **program**, **uses**, **procedure**, **begin**, **end**, etc.

5.5 Appel de procédures

Pour appeler une procédure, il suffit d'écrire son nom. Le `NomDeLaProcedure` est alors remplacé par le corps de la procédure c'est-à-dire par la liste d'instructions comprise entre le **begin** et le **end** dans sa déclaration.

On notera que si l'on déclare une procédure sans jamais l'appeler, elle ne sert absolument à rien.

On peut appeler une procédure dans le programme ou dans une autre procédure. Par exemple dans le programme `CroixGrecque_2`, on appelle la procédure `Croix` qui appelle la procédure `Branche` qui elle-même appelle les procédures `Cote` et `Extremite`.

Il faut impérativement respecter la règle qu'une procédure doit être déclarée avant d'être appelée. Supposons par exemple que l'on ait déclaré une procédure sous le nom `NomProc`. Supposons également que par suite d'une faute de frappe, on ait appelé cette procédure sous le nom `NomProg`. Le compilateur nous indique alors une erreur 'identificateur inconnu'. Ne sachant pas que nous avons écrit par erreur `NomProg` au lieu de `NomProc`, le compilateur suppose que la procédure `NomProc` a été déclarée mais n'a jamais été utilisée. C'est inutile mais autorisé. Il suppose également que la procédure `NomProg` a été appelée mais n'a pas été déclarée. C'est interdit. Retenant la faute la plus grave, le compilateur indique de manière lapidaire que le nom `NomProg` aurait dû figurer dans une déclaration de procédure.[1]

1. Nous parlons souvent des programmes de manière anthropomorphique, c'est-à-dire en leur attribuant des caractères propres à l'homme. En fait nous pensons aux programmeurs qui les ont écrits. Par exemple, l'ordinateur ne se trompe pas ; c'est son constructeur qui s'est trompé ou le programmeur qui a fait une erreur logique dans son programme.

5.6 Commandes du robot définies par des procédures

Certaines commandes du *Robot* sont équivalentes à une séquence de commandes élémentaires et peuvent donc s'exprimer avec des procédures. Par exemple, pour pivoter d'un quart de tour, il suffit de pivoter deux fois de suite d'un huitième de tour. On a donc :

```
procedure pqtd;
 { Fait pivoter le robot sur place d'un quart de tour à droite }
begin
 phtd; phtd;
end;

procedure pqtg;
 { Fait pivoter le robot sur place d'un quart de tour à gauche }
begin
 phtg; phtg;
end;
```

Voici de nouvelles commandes du *Robot* qui peuvent s'exprimer à l'aide de procédures en utilisant des commandes que nous connaissons déjà :

```
procedure p3htd;
 { Fait pivoter le robot sur place de 3 huitièmes de tour à droite }
begin
 pqtd; phtd;
end;

procedure p3htg;
 { Fait pivoter le robot sur place de 3 huitièmes de tour à gauche }
begin
 pqtg; phtg;
end;

procedure pdt;
 { Fait pivoter le robot sur place d'un demi tour }
begin
 pqtd; pqtd;
end;
```

Exercice 12 *Écrire des programmes* Pascal *comportant des procédures pour réaliser les croix, l'étoile astroïdale, la loupe ou le double carré cranté ci-dessous.* □

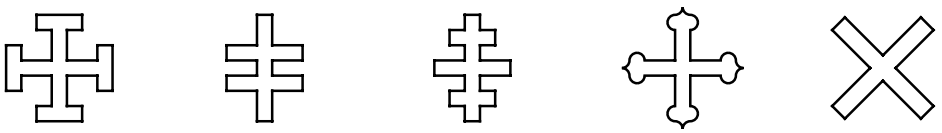


FIG. 5.3 – *Croix potencée, double, papale, tréflée et de Saint André*

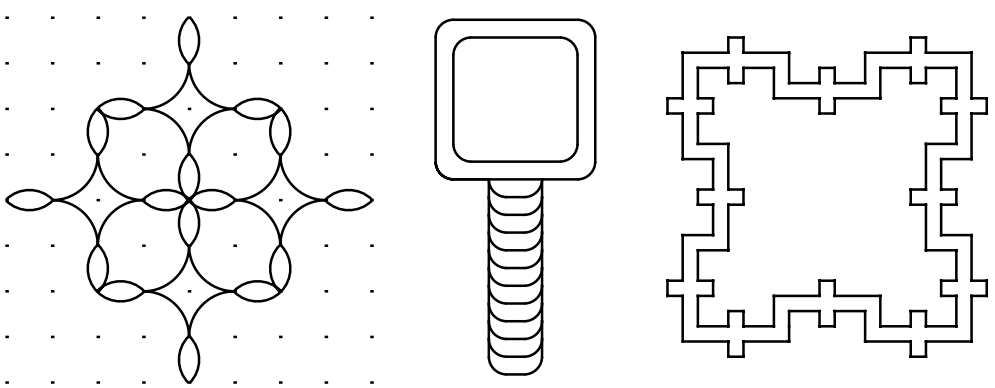


FIG. 5.4 – *Étoile astroïdale, loupe et double carré cranté*

Corrigé 12 *Nous donnons un exemple de programme, les autres se trouvant corrigés sur la disquette d'accompagnement.*

```
program EtoileAstroidale;
 uses  ;
 { Dessiner une étoile formée de quatre astroïdes à boucles }

 procedure Arc;
 begin
  phtg; vd; phtg; vd; phtg; vd;
 end;

 procedure Astroide;
 begin
  Arc; p3htd; Arc; p3htd; Arc; p3htd; Arc;
 end;

 procedure Etoile;
 begin
  Astroide; phtd; Astroide; phtd; Astroide; phtd; Astroide;
 end;

begin
 Etoile; st;
end.
```

□

Addenda

La notion de procédure introduite par Von Neumann et Turing fût utilisée en 1949 sous le nom de *routine* sur l'EDSAC (*Electronic Delay Storage Automatic Computer*) généralement considéré comme le premier ordinateur à programme enregistré ayant fonctionné. Cette machine fût construite dès 1947 à l'université de Cambridge en Angleterre par l'équipe de Wilkes[a] et exécuta ses premiers programmes en Mai 1949 [23].

Les procédures illustrent le concept d'*abstraction* : les propriétés d'un objet sont isolées afin de les utiliser ou de les étudier à part. Plus précisément, une procédure donne un nom à un algorithme de construction d'une partie d'un dessin. Dans le corps de la procédure, on s'intéresse à la façon de construire le dessin. Dans les appels on s'intéresse uniquement au dessin terminé, tout en désirant complètement ignorer comment le dessin est obtenu. On sépare donc l'objet et la façon de le construire, le quoi du comment.

[a] Maurice Wilkes, l'un des inventeurs du radar.

Nous trouvons utile de décrire le dessin réalisé par une procédure dans un commentaire placé au début de la procédure que l'on appelle la *spécification* de la procédure. On remarquera que pour qu'une procédure puisse être utilisée sans jamais consulter son corps, il faut que la spécification de la procédure décrive quel dessin elle fait mais également quelles sont les positions et orientations du $\mathcal{R}\mathcal{o}\mathcal{b}\mathcal{o}t$ avant et après la construction du dessin. Il se dégage ainsi une notion d'*invariant* sur laquelle nous reviendrons dans les chapitres suivants sur l'itération.

Les procédures sont à la base des méthodes de structuration des très grands programmes : pour être compréhensibles, les programmes de plusieurs milliers de lignes doivent être découpés en procédures ne dépassant pas quelques pages. Les procédures constituent donc un moyen élégant pour segmenter logiquement un programme. Cependant le concept de procédure est insuffisant pour structurer des programmes de plusieurs millions de lignes. On utilise alors la notion de *module* (on dit également *unité*) et de *bibliothèque* pour regrouper des ensembles de procédures.

Des procédures regroupées dans une *bibliothèque* (en Turbo Pascal on dirait *unité*) peuvent être fournies aux programmeurs pour les aider dans leurs tâches. Pour utiliser une procédure, le programmeur doit connaître son nom et sa spécification sans avoir besoin de connaître son fonctionnement interne. La clause **uses** au début du programme a pour but d'indiquer au compilateur quelles sont les bibliothèques utilisées dans ce programme. Les commmandes du $\mathcal{R}\mathcal{o}\mathcal{b}\mathcal{o}t$ sont des procédures regroupées dans des bibliothèques.

L'analyse de dessins géométriques pour en dégager une structure que l'on peut décrire algorithmiquement à l'aide de procédures est très formatrice pour l'esprit. En ce sens l'apprentissage de la programmation peut jouer pour les enfants un rôle formateur analogue à celui que jouait autrefois la géométrie euclidienne.

6

Sauts du robot au bord et au centre du cadre

Notre *Robot* peut sauter au bord ou au centre du cadre (qui délimite la zone de dessin sur l'écran de l'ordinateur), sans changer son orientation, ni rien dessiner. C'est exactement comme si nous pouvions le porter à la main pour le poser au bord ou au centre du cadre comme on peut le faire avec une voiture télécommandée. L'ordinateur le fait à notre place grâce aux commandes données dans la table ci-dessous et illustrées à la figure 6.1 (page 64) :

`ag`	place le *Robot* à gauche de l'écran, sans changer sa position horizontale, ni son orientation, et sans rien dessiner.
`ad`	place le *Robot* à droite de l'écran, sans changer sa position horizontale, ni son orientation, et sans rien dessiner.
`eh`	place le *Robot* en haut de l'écran, sans changer sa position verticale, ni son orientation, et sans rien dessiner.
`eb`	place le *Robot* en bas de l'écran, sans changer sa position verticale, ni son orientation, et sans rien dessiner.
`ce`	place le *Robot* au centre de l'écran, sans changer son orientation, et sans rien dessiner.

TAB. 6.1 – *Sauts du Robot au bord et au centre du cadre*

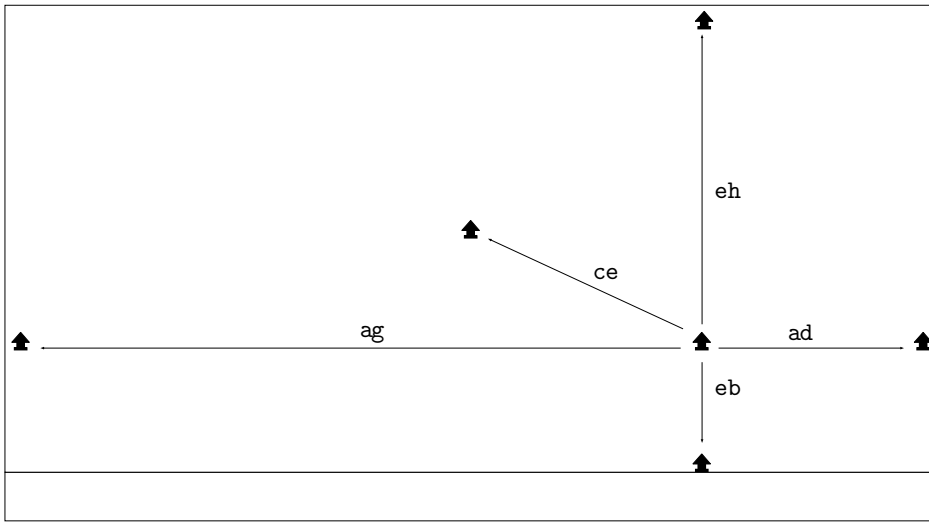


FIG. 6.1 – *Sauts du Robot au bord et au centre du cadre*

Exercice 13 *Écrire un programme* PASCAL *qui dessine des croix au bord et au centre de l'écran comme le montre la figure ci-dessous.* □

Corrigé 13

```
program CroixDeCoins;
  uses  ;
begin
  ag; dc; { Croix au milieu du bord gauche du cadre }
  ad; dc; { Croix au milieu du bord droit du cadre }
  ce; eh; dc; { Croix au milieu du bord supérieur du cadre }
  eb; dc; { Croix au milieu du bord inférieur du cadre }
  eh; ag; dc; { Croix au coin supérieur gauche du cadre }
  ad; dc; { Croix au coin supérieur droit du cadre }
  eb; ag; dc; { Croix au coin inférieur gauche du cadre }
  ad; dc; { Croix au coin inférieur droit du cadre }
  ce; dc; { Croix au centre du cadre }
  st;
end.
```

□

Addenda

Les commandes de déplacement du $\mathcal{R}o\underset{\sim}{b}ot$ au bord du cadre seront définies en PASCAL ultérieurement après avoir expliqué les notions de test et d'itération. Nous les introduisons dès maintenant pour pouvoir dessiner aisément des frises. Dans le chapitre suivant ces frises à motifs répétitifs vont nous permettre d'introduire simplement la notion d'itération bornée.

La commande ce de recentrage du $\mathcal{R}o\underset{\sim}{b}ot$ sera essentielle dès que nous utiliserons un repère cartésien pour désigner les points de la grille.

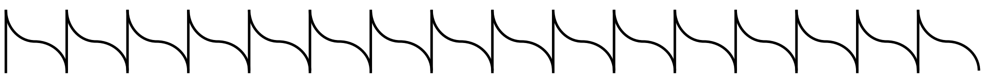
FIG. 6.2 – *Croix au bord et au centre du cadre*

7

Boucles “for”

7.1 Répétitions

Nous avons déjà vu quelques exemples de programmes Pascal dans lesquels une séquence de commandes du *Robot* ou d’appels de procédures se trouve répétée de nombreuses fois de suite. C’est le cas par exemple dans le programme `FriseDeN_1` dont l’exécution produit la frise ci-dessous :



```
program FriseDeN_1;
  uses   ;
begin
  ag;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  av; av; pdt; vg; vd; pdt; av; av; pdt; vg; vd; pdt;
  st;
end.
```

Pour éviter les répétitions successives des mêmes séquences d’instructions, nous allons étudier une nouvelle instruction du langage Pascal qui s’appelle la *boucle “for”* (le mot anglais “for”, qui signifie “pour”, se prononce comme

il s'écrit[1]). Voici le programme `FriseDeN_1` de dessin de la frise ci-dessous réécrit en utilisant une boucle "for" :

```
program FriseDeN_2;
  uses   ;
  var J : integer;
begin
  ag;
  for J := 1 to 16 do
    begin
      av; av; pdt; vg; vd; pdt;
    end;
  st;
end.
```

Juste après la ligne '**uses** ;' nous trouvons une *déclaration de variable* '**var** `J : integer;`'. Cette déclaration sert à indiquer qu'il va falloir que l'ordinateur utilise une case de sa mémoire comme *compteur*. Le mot anglais `integer` (prononcer 'inetédjeur') signifie 'entier'. Il indique que l'on compte avec des entiers 1, 2, ... Nous devons donner un nom à ce compteur. C'est un identificateur que nous pouvons choisir comme il nous plaît (ici par exemple `J`). Ce compteur va servir à l'ordinateur pour compter le nombre de fois qu'il aura à répéter la séquence d'instructions dans la boucle "for", exactement comme nous pourrions le faire en comptant sur nos doigts !

En français, la boucle '**for** `J := 1` **to** `16` **do begin end**;' peut se traduire par 'pour J de 1 à 16 répéter début ... fin;'. Le mot anglais **to** se prononce 'tou'. Le mot anglais **do** se prononce 'dou'.

La séquence d'instructions à répéter s'écrit entre le **begin** (début) et le **end** (fin). Elle s'appelle le *corps de la boucle*. Le *nombre d'itérations* c'est-à-dire le nombre de fois qu'il faut répéter le corps de la boucle est indiqué après le mot **to**. Dans notre exemple, il faut répéter 16 fois de suite le corps de la boucle '`av; av; pdt; vg; vd; pdt;`'.

1. Nous parlons de boucle "for" et non pas de boucle "pour" pour rester dans le contexte du vocabulaire utilisé dans le programme.

7.2 Déclarations de variables entières

Il faut indiquer dans un programme PASCAL que l'on utilise un ou plusieurs compteurs de boucles "for" par des *déclarations de variables* qui se placent dans le programme juste après les déclarations de procédures :

Programme PASCAL *:*

```
program ␣ NomDuProgramme;
 uses   ;
 ... Déclarations des procédures du programme ...
 var
  Identificateur1 : integer;
  Identificateur2 : integer;
  ...
begin
 ... Instructions du programme (séparées par des ';') ...
end.
```

Le mot anglais **var** (abréviation de 'variable') doit obligatoirement être séparé de l'identificateur qui le suit par un blanc ␣ ou un retour à la ligne.

7.3 La boucle "for"

Une *boucle "for"* a la forme suivante :

Boucle "for":

```
for ␣ Identificateur := 1 ␣ to ␣ Nombre d'itérations ␣ do
 begin
  ... Instructions du corps de la boucle (séparées par des ';') ...
 end;
```

L'identificateur utilisé comme nom du compteur d'itérations doit être déclaré comme désignant une variable entière. La séquence d'instructions séparées par des ';' qui est répétée s'appelle le *corps de la boucle*.

Il faut prendre garde à ne pas mettre de blanc ␣ entre le deux-points : et le égal = du symbole :=. Par contre il ne faut pas oublier ce blanc ␣ entre '**to**' et le 'Nombre d'itérations'.

Si le 'Nombre d'itérations' est plus petit que 1 (0 par exemple) ces instructions ne sont pas exécutées du tout. Une erreur grave est de mettre un point-virgule ';' après le **do**. Dans ce cas les instructions figurant entre le **begin** et le **end** sont exécutées une seule fois.

Exemple 10 (Frise de pointes) La frise suivante :

est produite par le programme ci-dessous :

```
program FriseDePointes;
  uses  ;
  var J : integer;
begin
  ag; pqtd;
  for J := 1 to 32 do
    begin
      lc; av; bc; pdt; vd; pdt; av; av; pdt; vd;
    end;
  st;
end.
```

□

7.4 Utilisation de la boucle "for" dans une procédure

Les compteurs de boucles "for" utilisés dans une procédure doivent être déclarés dans la procédure, comme suit :

Déclaration de procédure :

```
procedure␣NomDeLaProcedure;
 var
  Identificateur1 : integer;
  Identificateur2 : integer;
  ...
begin
 ... Instructions du corps de la procédure[a] (séparées par des ';') ...
end;
```

[a] Y compris les boucles "for" utilisant les compteurs Identificateur1,...

Un même identificateur peut apparaître plusieurs fois dans des déclarations de variables '**var** Identificateur : `integer`;' de procédures, à la condition expresse que ce soit dans des procédures différentes. Ceci est illustré dans le programme `Fleche` ci-dessous où des compteurs de boucles différents sont utilisés dans des procédures différentes (`Empennage` et `Tige`) portent le même nom `I`.

Exemple 11 (Flèche)

```
program Fleche;
 uses  ;

 procedure Plume;
 begin
  av; p3htg; av; lc; p3htg; av; av; p3htg; bc; av; phtd;
 end;

 procedure Empennage;
  var I : integer;
 begin
  for I := 1 to 5 do
   begin
    Plume;
   end;
 end;

 procedure Tige;
  var I : integer;
 begin
```

```
    for I := 1 to 10 do
      begin
        av;
      end;
  end;

  procedure Pointe;
  begin
    pqtg; av; p3htd; av; pqtd; av; p3htd; av;
  end;

begin
  EB; Empennage; Tige; Pointe; st;
end.
```

□

7.5 Boucles “for” imbriquées (indépendantes)

Le corps d’une boucle “for” :

for ␣ Identificateur1 := 1 ␣ **to** ␣ Nombre d’itérations1 ␣ **do**
begin
…
end;

peut à nouveau contenir des boucles “for” :

for ␣ Identificateur1 := 1 ␣ **to** ␣ Nombre d’itérations1 ␣ **do**
begin
…
for ␣ Identificateur2 := 1 ␣ **to** ␣ Nombre d’itérations2 ␣ **do**
begin
…
end;
…
end;

On dit alors que les boucles “for” sont *imbriquées*. La boucle utilisant le compteur Identificateur2 pour faire Nombre d’itérations2 itérations s’appelle la *boucle intérieure*. La boucle utilisant le compteur Identificateur1 pour faire Nombre d’itérations1 itérations s’appelle la *boucle extérieure*. Considérons par exemple le dessin d’un carré :

```
program Carre_1;
 uses  ;
 procedure Cote;
  { Dessiner un côté du carré et pivoter à droite }
  var J : integer;
 begin
  for J := 1 to 4 do
   begin
    av;
   end;
  pqtd;
 end;
 procedure Carre;
  { Dessiner un carré }
  var I : integer;
 begin
  for I := 1 to 4 do
   begin
    Cote;
   end;
 end;
begin
 Carre; st;
end.
```

En remplaçant les appels des procédures par leurs corps, on obtient deux boucles imbriquées :

```
program Carre_2;
 uses  ;
 var
  I : integer;
  J : integer;
begin
 { Dessiner un carré }
 for I := 1 to 4 do
  begin
   { Dessiner un côté du carré et pivoter à droite }
   for J := 1 to 4 do
    begin
     av;
    end;
   pqtd;
```

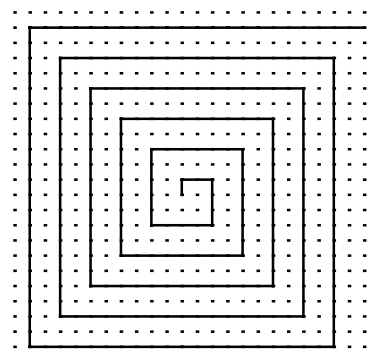
```
    end;
  st;
end.
```

La boucle intérieure sert à construire un côté tandis que la boucle extérieure dessine les quatre côtés.

7.6 Boucles “for” imbriquées (dépendantes)

Dans des boucles “for” imbriquées, le nombre d’itérations dans la boucle intérieure peut être différent à chaque fois que le corps de la boucle extérieure est exécutée. Commençons par considérer l’exemple simple de segments de droite de longueurs 1, 2, 3, ...disposés en spirale (le dessin est représenté ci-dessous à l’échelle $\frac{1}{2}$ avec la grille) :

```
program SpiraleCarree_1;
  uses  ;
  var
    I : integer;
    J : integer;
begin
  for I := 1 to 22 do
    begin
      for J := 1 to I do
        begin
          av;
        end;
      pqtd;
    end;
  st;
end.
```



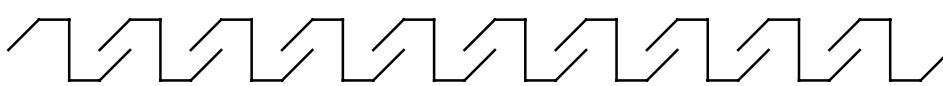
Le nombre d’itérations dans la boucle “for” intérieure est égal à la valeur du compteur `I` de la boucle extérieure, c’est-à-dire successivement à 1, 2, 3, ..., 22. Le programme `SpiraleCarree_1` est donc équivalent au programme `SpiraleCarree_2` ci-dessous où les boucles “for” ont été éliminées pour montrer exactement quelle est la séquence de commandes exécutée par le $\mathcal{R}^{o}\!\!\mathcal{b}^{o}t$.

```
program SpiraleCarree_2;
  uses  ;
begin
  av; pqtd;
```

```
  av; av; pqtd;
  av; av; av; pqtd;
  av; av; av; av; pqtd;
  av; av; av; av; av; pqtd;
  av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
  pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
  av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
  av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
  av; av; av; pqtd;
  av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av; av;
  av; av; av; av; pqtd;
  st;
end.
```

7.7 Invariants de boucles "for"

Quand on dessine une frise comme celle-ci :



avec une boucle "for", il faut que le $\mathcal{R}^{o}_{\smile}{}^{o}t$ se trouve toujours dans la même direction et dans la même position par rapport au prochain motif qui va être dessiné en exécutant le corps de la boucle. Autrement dit, à la fin du corps de la boucle il faut remettre le $\mathcal{R}^{o}_{\smile}{}^{o}t$ dans la position de départ :

```
program FriseDeTildes;
 uses   ;
```

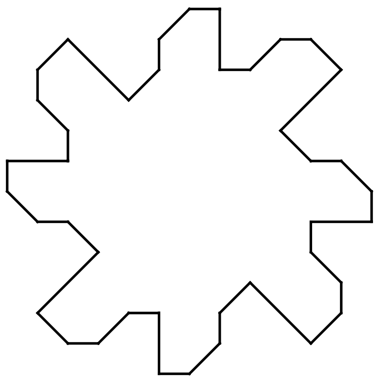
```
procedure Tilde;
  { Dessiner un motif en forme de tilde ~ }
begin
  phtd; av; phtd; av; pqtd; av; av; pqtg; av; phtg; av;
end;

var I : integer;

begin
 ag;
 for I := 1 to 10 do
  begin
   Tilde;
   { Reculer au début du tilde suivant }
   lc; p3htg; av; bc;
   { Orienter le robot au nord }
   pqtd;
  end;
 st;
end.
```

On dit que la position du *Robot* est un *invariant* de la boucle en ce sens qu'elle doit être la même à chaque itération.

Si dans notre exemple on dessine un motif puis le suivant sans remettre entre-temps le *Robot* dans la position de départ, on obtient le résultat suivant, dont il faut avouer qu'il n'est pas très ressemblant à l'original :



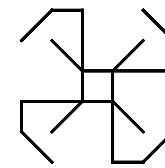
```
program FriseDeTildesErronee1;
 uses   ;

 procedure Tilde;
  { Dessiner un motif en forme de tilde ~ }
 begin
  phtd; av; phtd; av; pqtd;
  av; av; pqtg; av; phtg; av;
 end;

 var I : integer;

begin
 for I := 1 to 10 do
  begin
   Tilde;
  end;
 st;
end.
```

Si l'on pense à faire reculer le *Robot* mais que l'on oublie de le remettre dans la bonne direction, le résultat est encore plus étrange :



```
program FriseDeTildesErronee2;
  uses   ;
  procedure Tilde;
    { Dessiner un motif en forme de tilde ~ }
  begin
    phtd; av; phtd; av; pqtd; av; av; pqtg; av; phtg; av;
  end;
  var I : integer;
begin
  for I := 1 to 13 do
    begin
      Tilde; { Reculer au début du tilde suivant } lc; p3htg; av; bc;
    end;
  st;
end.
```

Une bonne façon de trouver les erreurs dans une boucle "for" est donc de chercher l'invariant, c'est-à-dire la position et l'orientation dans lesquelles doit se trouver le *Robot* avant de dessiner le motif en exécutant le corps de la boucle. Il faut alors vérifier que le *Robot* est convenablement placé quand on arrive pour la première fois dans la boucle et qu'il l'est à nouveau après avoir exécuté le corps de la boucle.

La même notion d'invariant s'applique pour les procédures. Avant chaque appel d'une procédure, il faut que le robot soit dans un ou plusieurs états possibles qui doivent être parfaitement définis par la position levée ou baissée du crayon du *Robot* et par la position et l'orientation du *Robot* par rapport à la figure dessinée par la procédure.

Exercice 14

– *Écrire des programmes* Pascal *utilisant des boucles "for" pour dessiner quelques-unes des frises de la figure 7.3 page 79.*

– *Écrire des programmes* Pascal *utilisant des boucles "for" dans des procédures pour dessiner quelques-unes des frises de la figure 7.4 page 80.*

– *Écrire des programmes* Pascal *pour dessiner quelques-unes des rosaces, frises et pavage de la figure 7.5 page 81, en utilisant des boucles "for" imbriquées indépendantes.*

– *Écrire des programmes* Pascal *pour dessiner quelques-unes des spirales, quadrillages et autres tuyaux d'orgues de la figure 7.1 ci-dessous, en utilisant des boucles "for" imbriquées dépendantes.*

– *Écrire un programme* Pascal *pour dessiner le pavage de la figure 7.2 ci-dessous en indiquant avec précision les invariants des procédures et boucles "for".* □

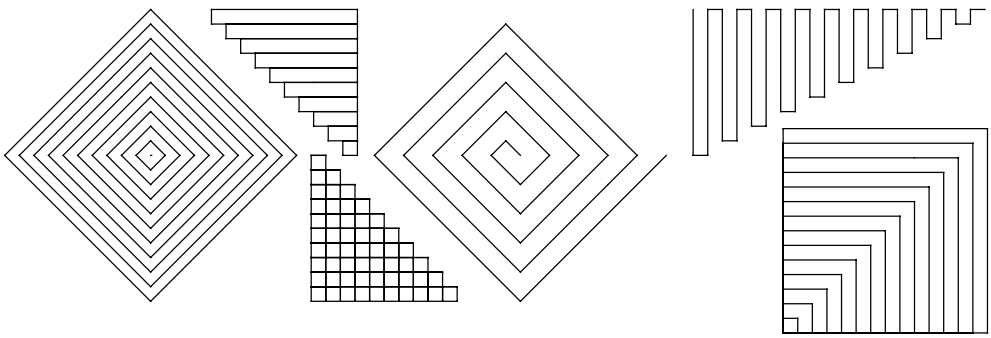


Fig. 7.1 – *Quadrillages, carrés, spirale et tuyaux d'orgues*

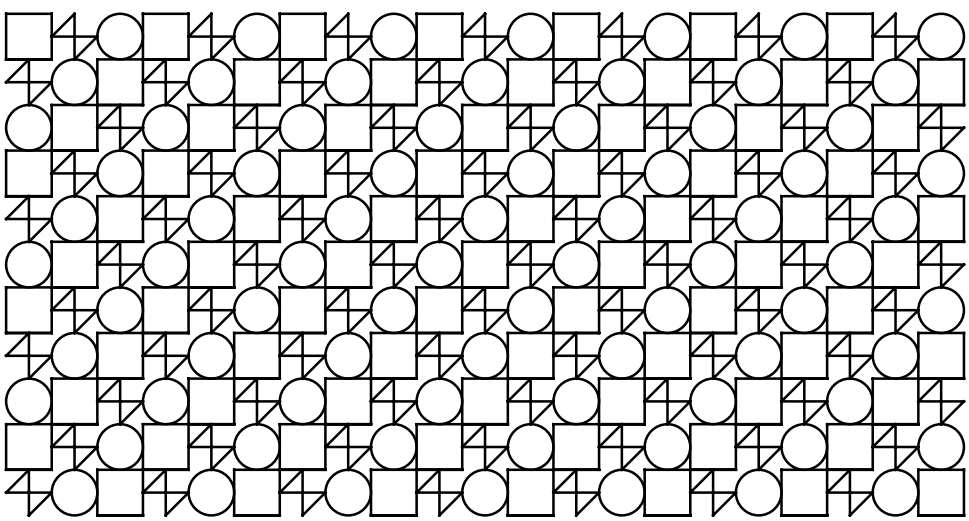


Fig. 7.2 – *Pavage de carrés, triangles et cercles*

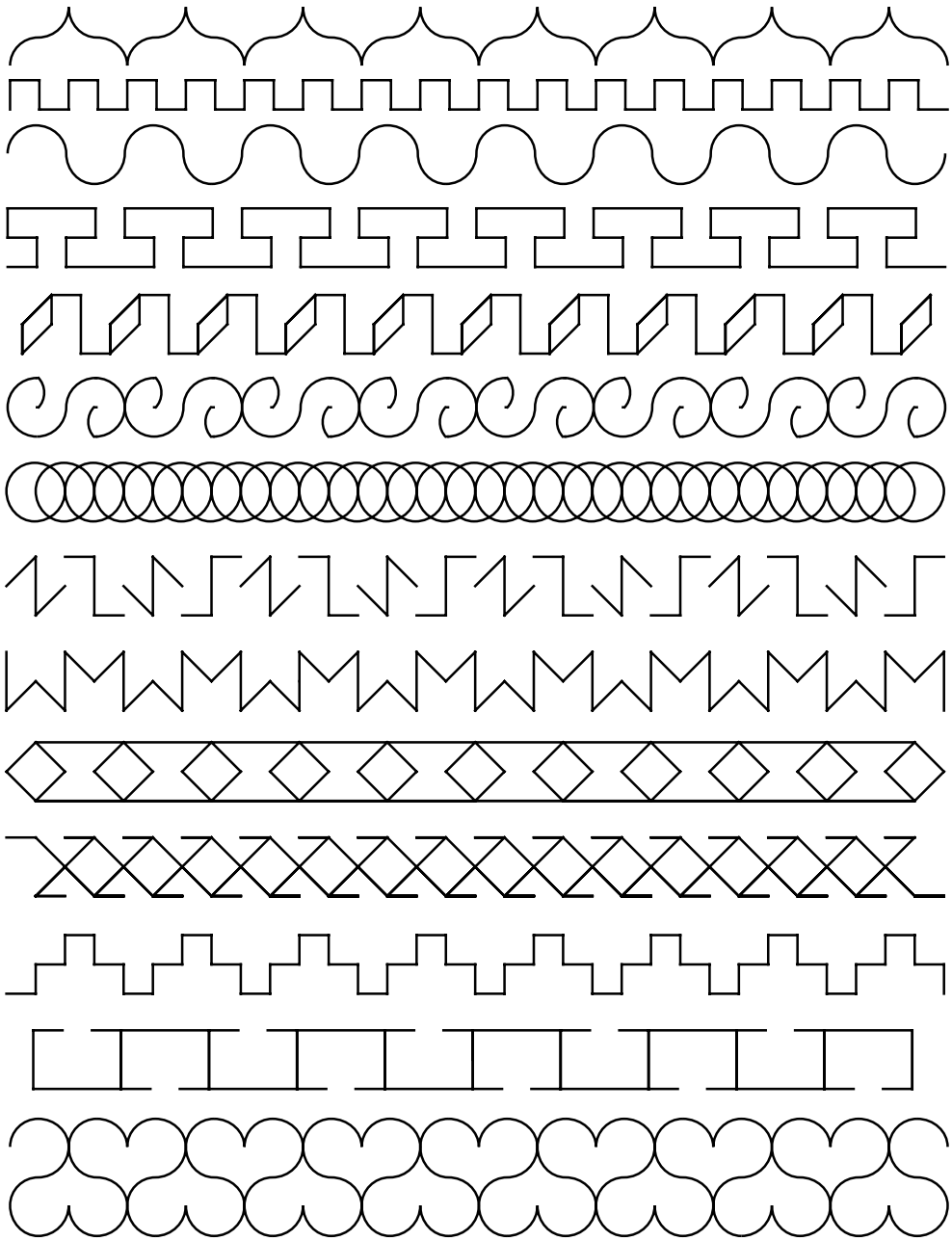


FIG. 7.3 – *Frises (1)*

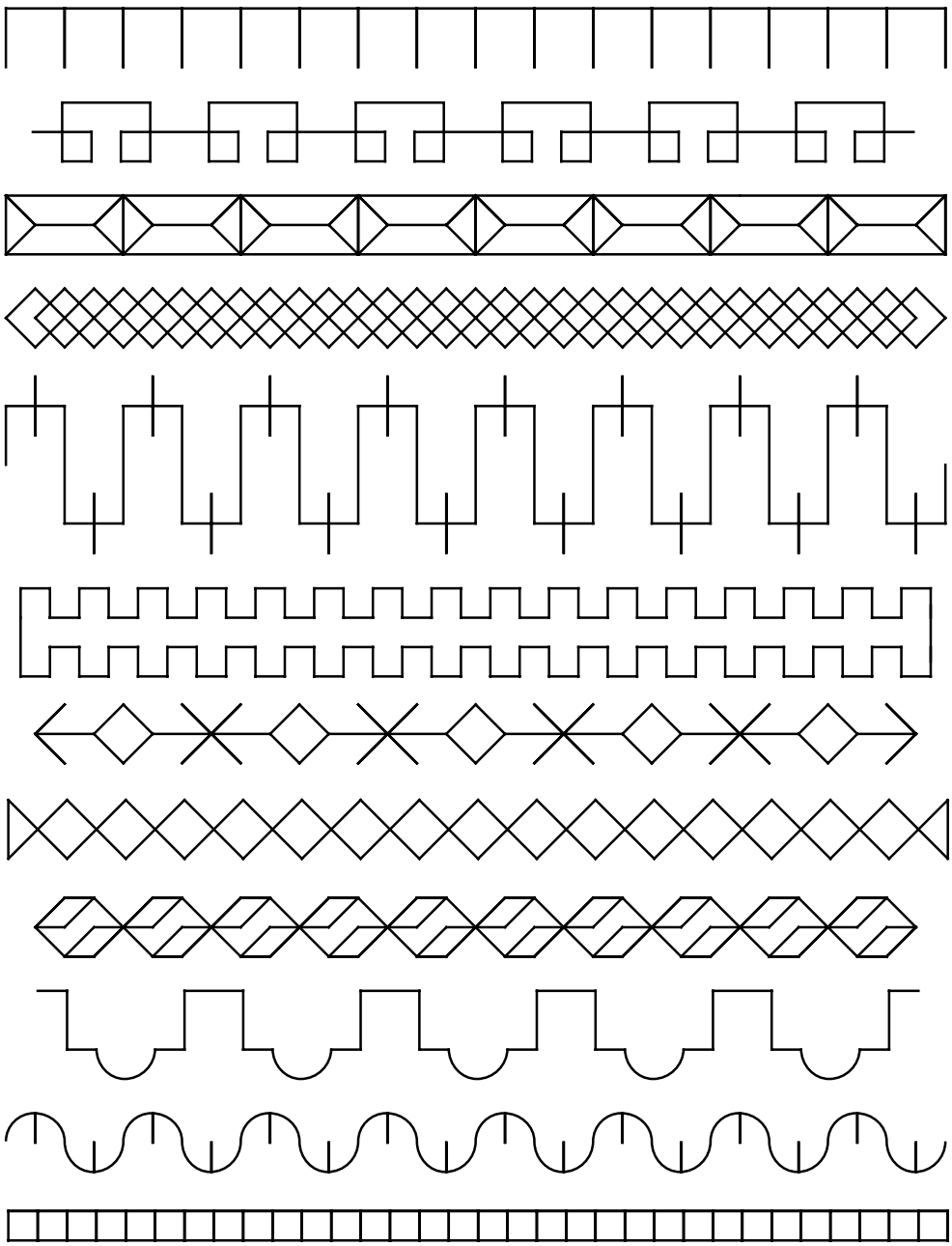


FIG. 7.4 – *Frises (2)*

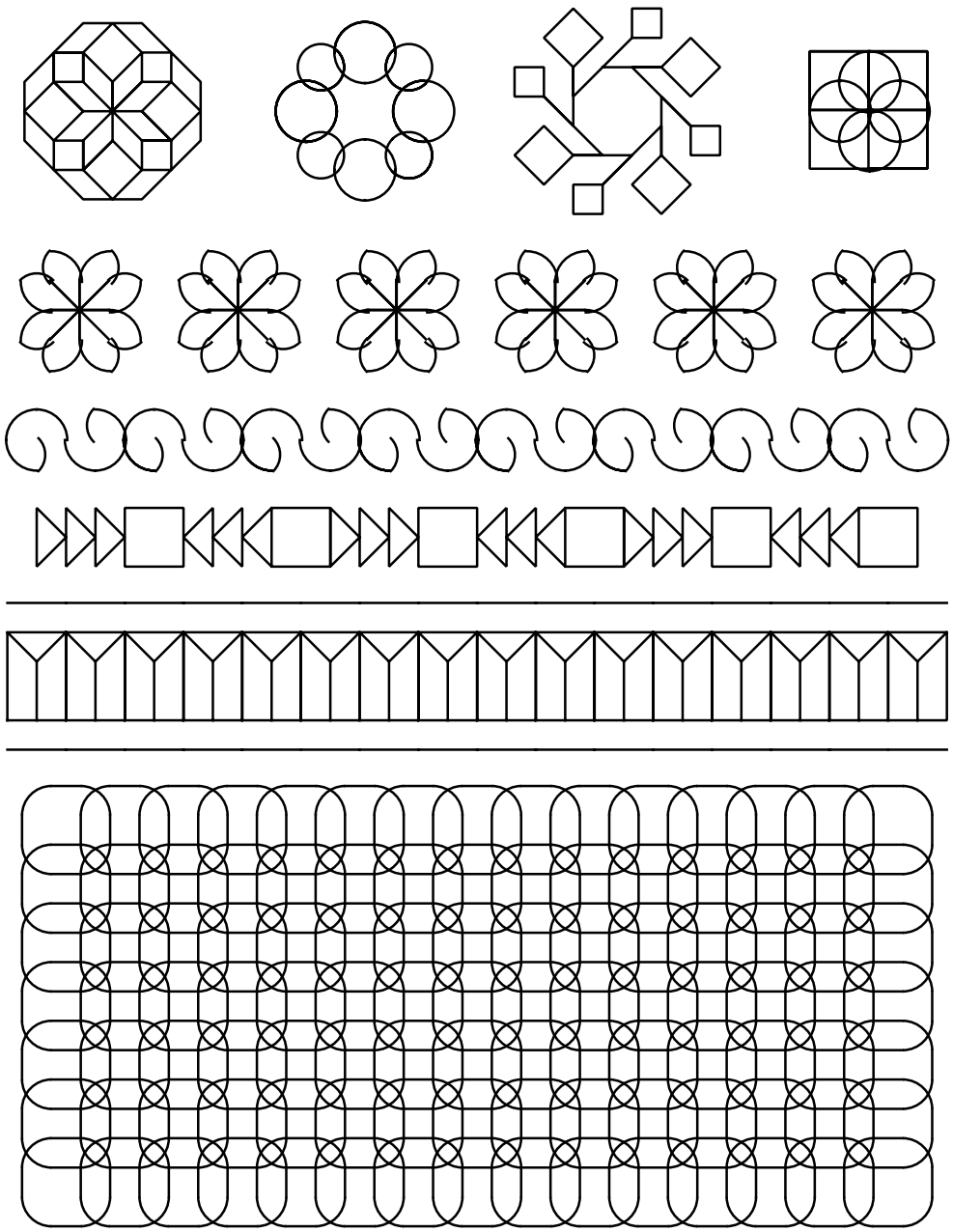


FIG. 7.5 – *Rosaces, frises et pavage*

Corrigé 14 *Nous donnons quelques programmes, les autres se trouvant sur la disquette d'accompagnement du livre :*

-
```
program FriseCrantee;
  uses  ;

  procedure Cran;
    { Dessiner le motif 'n_' }
  begin
     av; pqtd; av; pqtd; av; pqtg; av; pqtg;
  end;

  var I : integer;

begin
  ag;
  for I:= 1 to 21 do
    begin
      Cran;
    end;
  st;
end.
```

-
```
program QuadrillageTriangulaire;
  uses  ;

  var
    I : integer;
    J : integer;
    K : integer;
  begin
    { Dessiner 10 lignes de carrés juxtaposés horizontalement }
    pqtd;
    for I := 1 to 10 do
      begin
        { Dessiner une ligne de  I  carrés juxtaposés horizontalement, }
        { en partant en haut à gauche et en arrivant en haut à droite, }
        { le robot étant dans les deux cas orienté vers l'est.         }
        for J := 1 to I do
          begin
            { Dessiner un carré, en partant du coin supérieur gauche et en }
            { arrivant  au  coin  supérieur droit, le robot étant dans les }
            { deux cas orienté vers l'est.                                  }
```

```
      for K := 1 to 4 do
       begin
         av; pqtd;
       end;
      av;
     end;
   { Revenir, sans dessiner, au début de la ligne suivante, le robot }
   { étant orienté vers l'est au départ et à l'arrivée.              }
   lc; pdt;
   for J := 1 to I do
    begin
      av;
    end;
   pqtg; av; pqtg; bc;
  end;
 st;
end.
```

■
```
program PavageCarresTrianglesCercles;
 uses  ;

 procedure Carre;
  { Dessiner un carré de côté de taille 2, le robot partant du coin }
  { inférieur  gauche tourné vers l'est, et arrivant crayon baissé  }
  { au coin inférieur droit également orienté à l'est.              }
 begin
  { Dessiner le carré.                                              }
  av; av; pqtg; av; av; pqtg; av; av; pqtg; av; av; pqtg;
  { Avancer au coin inférieur droit.                                }
  av; av;
 end;

 procedure Triangles;
  { Dessiner  deux triangles opposés par un sommet inscrits dans un }
  { carré invisible du quadrillage de côté 2, en  partant  du  coin }
  { inférieur  gauche du carré tourné vers l'est, et en arrivant au }
  { coin inférieur droit, crayon baissé et tourné vers l'est.       }
 begin
  lc; av; bc;
  pqtg; av; av; p3htg; av; p3htg; av; av; p3htd; av; p3htg;
  lc; av; bc;
 end;
```

```
procedure Cercle;
  { Dessiner  un  cercle de rayon 1 inscrit dans un carré invisible }
  { du quadrillage de côté 2, en partant du coin  inférieur  gauche }
  { du  carré  tourné  vers l'est, et en arrivant au coin inférieur }
  { droit du carré, crayon baissé et tourné vers l'est.             }
begin
  lc; av; bc;
  vg; vg; vg; vg;
  lc; av; bc;
end;

procedure LigneCarre;
{ Dessiner une ligne constituée d'un motif répété 7 fois, le motif }
{ étant  formé d'un carré, de triangles et d'un cercle juxtaposés, }
{ le robot partant du coin inférieur gauche de  la  ligne  orienté }
{ vers l'est, et arrivant au coin inférieur droit également orien- }
{ té à l'est.                                                      }
  var I : integer;
begin
  for I := 1 to 7 do
    begin
      { Le robot est crayon baissé, à l'est, en bas à gauche du motif }
      Carre; Triangles; Cercle;
    end;
end;

procedure LigneTriangles;
{ Dessiner une ligne constituée d'un motif répété 7 fois, le motif }
{ étant  formé de triangles, d'un cercle et d'un carré juxtaposés, }
{ le robot partant du coin inférieur gauche de  la  ligne  orienté }
{ vers l'est, et arrivant au coin inférieur droit également orien- }
{ té à l'est.                                                      }
  var I : integer;
begin
  for I := 1 to 7 do
    begin
      { Le robot est crayon baissé, à l'est, en bas à gauche du motif }
      Triangles; Cercle; Carre;
    end;
end;

procedure LigneCercle;
{ Dessiner une ligne constituée d'un motif répété 7 fois, le motif }
{ étant  formé d'un cercle, de triangles et d'un carré juxtaposés, }
{ le robot partant du coin inférieur gauche de  la  ligne  orienté }
{ vers l'est, et arrivant au coin inférieur droit également orien- }
{ té à l'est.                                                      }
```

```
  var I : integer;
begin
  for I := 1 to 7 do
    begin
    { Le robot est crayon baissé, à l'est, en bas à gauche du motif }
      Cercle; Carre; Triangles;
    end;
end;

procedure AlaLigne;
  { Ramener le robot du coin inférieur droit d'une ligne au coin }
  { inférieur gauche de la ligne suivante sans changer l'orienta- }
  { tion du robot à l'est, sans dessiner et en baissant le crayon }
  { au début de la ligne suivante.                                }
begin
  lc; ag; pqtd; av; av; pqtg; bc;
end;

procedure Pavage;
  { Paver l'écran de 11 lignes commençant par un carré, puis par }
  { des triangles puis par un cercle constituées de motifs répéti- }
  { tifs formés d'un carré suivi d'un cercle et de triangles juxta- }
  { posés en partant du coin supérieur droit et en arrivant au coin }
  { inférieur droit de l'écran, avec le robot tourné vers l'est.    }
  var J : integer;
begin
  { Dessiner les lignes 1 à 9 de haut en bas.                      }
  for J := 1 to 3 do
    begin
      AlaLigne; LigneCarre;
      AlaLigne; LigneTriangles;
      AlaLigne; LigneCercle;
    end;
  { Dessiner les lignes 10 puis 11.                                }
  AlaLigne; LigneCarre;
  AlaLigne; LigneTriangles;
end;

begin
  { Placer le robot au coin supérieur droit orienté vers l'est.      }
  eh; ad; pqtd;
  { Paver l'écran, terminer au coin inférieur droit orienté à l'est. }
  Pavage; st;
end.
```

□

Addenda

La boucle "for" est une des premières notions linguistiques qui soit apparue en informatique puisqu'elle fût inventée à l'ETH de Zürich en 1952 par Heinz Rutishauser [31].

La comparaison des dessins que l'on peut obtenir en utilisant la boucle "for" avec ceux que l'on peut faire en mode interactif donne la mesure de la puissance des ordinateurs quand il s'agit d'effectuer rapidement des tâches répétitives. En cela l'ordinateur se distingue très nettement des calculettes non programmables. On comprend également que cette puissance de calcul n'a rien à voir avec une quelconque forme de pensée ou d'intelligence si ce n'est celle du programmeur de l'ordinateur.

En PASCAL, on utilise une boucle **for** `I` $:= n_1$ **to** n_2 **do begin** ...**end**; pour répéter $(n_2 - n_1) + 1$ fois le corps de la boucle compris entre **begin** et **end** sauf si $n_2 < n_1$ auquel cas le corps de la boucle n'est pas exécuté. A chaque itération, le compteur de boucle `I` prend les valeurs successives n_1, $n_1 + 1$, $n_1 + 2$, ..., n_2. Dans la forme **for** `I` $:= n_1$ **downto** n_2 **do begin** ...**end**;, `I` prend les valeurs successives n_1, $n_1 - 1$, $n_1 - 2$, ..., n_2 sauf si $n_2 > n_1$ auquel cas la boucle n'est pas exécutée.

L'utilisation de boucles "for" exerce la capacité de percevoir quand un dessin peut se décomposer en un certain nombre de sous-dessins identiques (dans le cas d'une boucle simple) ou similaires (dans le cas de boucles imbriquées). C'est un exercice quotidien pour l'informaticien qui doit savoir décomposer une tâche complexe en sous-tâches simples, ce qui demande parfois énormément d'imagination. Il s'agit d'un premier pas vers l'acquisition du raisonnement inductif des scientifiques qui, à partir d'observations spécifiques, créent une hypothèse générale rendant compte de ces observations, hypothèse qu'il reste à valider dans tous les cas possibles, ici de manière tout à fait expérimentale, en observant le dessin à l'écran. Les mathématiciens appellent *conjecture* une telle hypothèse induite de l'expérience et non encore démontrée.

Ce raisonnement inductif a longtemps été confondu avec le raisonnement par récurrence qui est essentiel en mathématiques. Il a fallu plus d'un millénaire pour que les mathématiciens (Fermat [a], Pascal [b] notamment) comprennent pleinement ce raisonnement par récurrence ([20] page 17).

[a] Pierre de Fermat, 1601–1665, mathématicien français.
[b] Blaise Pascal, 1623–1662, mathématicien français.

Nous en sommes pour l'instant aux prémisses, l'absence d'induction correcte étant encore admise dans le raisonnement par récurrence : si le dessin est bien fait pour le motif de base et pour la première répétition du motif alors il est réputé correct, seul le problème du cadrage dans la fenêtre restant à résoudre par simple expérimentation.

La notion d' *invariant* (c'est-à-dire pour notre *Robot* la description de ses positions et orientations possibles au début du corps des boucles) est due à Naur [a] en 1966 et Floyd [b] en 1967. Elle repose sur un raisonnement par récurrence : si l'invariant est vrai quand on arrive pour la première fois dans la boucle (c'est-à-dire que la position et l'orientation du *Robot* est conforme à la description qui en est faite dans l'invariant) et si, en supposant que l'invariant soit vrai au début de l'exécution du corps de la boucle, il reste encore vrai après l'exécution de ce corps de boucle alors l'invariant est toujours vrai au début de chaque itération dans la boucle et l'est encore quand on sort de la boucle. La notion s'applique aussi bien aux procédures : on peut utiliser un *invariant d'entrée* décrivant les positions et orientations du *Robot* avant l'appel et un *invariant de sortie* décrivant ces positions et orientations après l'appel, de sorte que le corps de la procédure soit compréhensible sans avoir à consulter tous les points d'appel pour savoir dans quelles conditions la procédure est appelée.

L'ordinateur peut être utilisé pour manipuler des programmes, en particulier pour découvrir automatiquement des invariants de forme simple en chaque point d'un programme. C'est l'objet de l' *interprétation abstraite* des programmes [8]. Pour le *Robot*, par l'exemple, une forme simple d'invariant est constituée par l'ensemble de ses orientations possibles, en ignorant la position de son crayon, ses coordonnées sur l'écran, etc. Les orientations possibles du *Robot* concret peuvent être calculées en exécutant le programme avec un *Robot* abstrait qui exécute les commandes du *Robot* concret sans bouger mais en changeant simplement d'orientation. Pour ce *Robot* abstrait la commande `av` ne fait rien car elle ne change pas l'orientation du *Robot* concret ; la commande `vd` a le même effet que `pqtd` ; etc. Pendant l'exécution abstraite les orientations possibles du *Robot* abstrait sont notées comme commentaire en chaque point du programme. L'exécution abstraite du programme s'arrête après stabilisation des résultats. Par exemple, pour le programme `FriseDeN_2` (page 68), on trouve :

[a] Peter Naur, informaticien danois.

[b] Robert Floyd, informaticien américain.

```
program FriseDeN_2;
  uses  ;
  var J : integer;
begin
  { Nord }
  ag;
  { Nord }
  for J := 1 to 16 do
    begin
      { Nord } av; { Nord } av; { Nord } pdt; { Sud }
      vg; { Est } vd; { Sud } pdt; { Nord }
    end;
  { Nord }
  st;
  { Nord }
end.
```

La règle des signes est un exemple d'interprétation abstraite utilisée en mathématiques. Le signe d'une expression peut être calculé en utilisant le signe des variables, qui remplace leurs valeurs, grâce aux opérations abstraites comme $\oplus + \oplus = \oplus$, $\oplus - \ominus = \oplus$, $\oplus \times \ominus = \ominus$, etc. Le prix à payer pour cette simplification est évidemment l'imprécision des résultats, comme pour $\oplus + \ominus$.

8

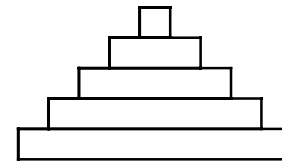
Expressions entières

Un ordinateur est capable de faire des *additions* $(15+3)=18$, des *soustractions* $(15-3)=12$, des *multiplications* $(15*3)=45$ et des *divisions* d'entiers (15 **div** 3) $=5$. Si la division ne tombe pas juste, le résultat est le quotient entier. Le reste est donné par l'opération *modulo* notée **mod**. Par exemple (17 **div** 3) $=5$ et (17 **mod** 3) $=2$ car la division du dividende 17 par le diviseur 3 donne le quotient 5 et le reste 2. En mathématiques on utilise une croix '$\times$' pour noter la multiplication tandis qu'en Pascal on utilise l'astérisque $*$. On évite ainsi de confondre le signe multiplié '$\times$' avec la lettre `x` minuscule.

Pour faire un calcul comportant plusieurs opérations, on utilise des parenthèses. Par exemple $((2*3)-1)$ est égal à 5. Le calcul commence par l'évaluation de l'expression entre parenthèses la plus intérieure. Par exemple $((((5+1)*3)-4)$ **div** $2) = (((6*3)-4)$ **div** $2) = ((18-4)$ **div** $2) = (14$ **div** $2) = 7$.

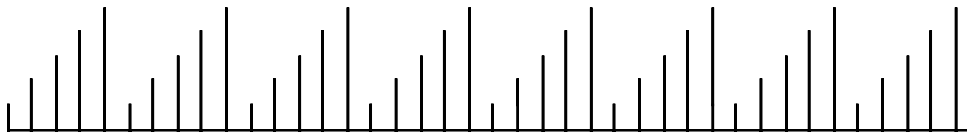
Dans ces *expressions entières* nous pouvons également utiliser la valeur des compteurs de boucles "for". Par exemple lors des itérations successives dans le corps de la boucle "**for** `I` := `1` **to** `5` **do begin end**;", le compteur `I` vaut 1, puis 2, puis 3, puis 4 et enfin 5. Dans ce corps de boucle l'expression $((2*$`I`$)-1)$ vaut donc successivement $((2*1)-1)=1$, $((2*2-1)=3$, $((2*3)-1)=5$, $((2*4)-1)=7$ et enfin $((2*5)-1)=9$.

Exemple 12 (Pyramide) *Pour dessiner la pyramide ci-après, nous pouvons commencer par le sommet et dessiner de haut en bas des rectangles successifs de longueurs 1, 3, 5, 7, 9. La longueur du côté du* `I`$^{\text{ème}}$ *rectangle à partir du sommet est donc* $((2*$`I`$)-1)$.



```
program Pyramide;
  uses  ;
  { Dessiner une pyramide de 5 étages. }
var
  I : integer;
  J : integer;
begin
  { Orienter le robot à l'ouest. }
  pqtg;
  for I := 1 to 5 do
    begin
      { Partant du coin inférieur gauche d'un étage avec le robot }
      { orienté à l'ouest,  dessiner  le coin inférieur gauche de }
      { l'étage inférieur et pivoter à l'est.                     }
         av; pqtg; av; pqtg;
      { Dessiner le Ième étage rectangulaire. }
         { Côté inférieur   }
            for J := 1 to ((2 * I) - 1) do
              begin
                av;
              end;
         { Extrémité droite }
            pqtg; av; pqtg;
         { Côté supérieur   }
            for J := 1 to ((2 * I) - 1) do
              begin
                av;
              end;
      { Redescendre au coin inférieur gauche et orienter le robot }
      { à l'ouest.                                                }
         pqtg; av; pqtd;
    end;
  st;
end.
```

Exercice 15 *Compléter le programme* `PiquetsEnDentsDeScie` *pour obtenir l'alignement de piquets représenté ci-après :*

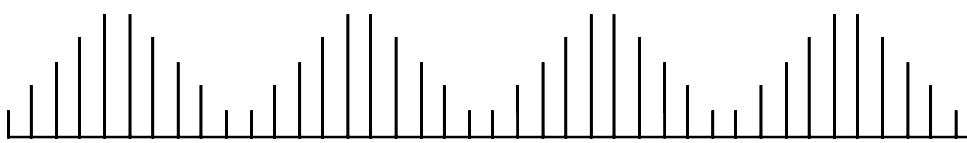


```
program PiquetsEnDentsDeScie;
  uses   ;
  var I : integer; J : integer; K : integer;
begin
  ag;
  for I := 1 to 40 do
    begin
      for J := 1 to 2 do
        begin
          for K := 1 to   do
            begin
              av;
            end;
          pdt;
        end;
      pqtd; av; pqtg;
    end;
  st;
end.
```

□

Exercice 16 *Écrire des programmes* Pascal *de commande du* Robot *pour réaliser les dessins géométriques de la figure 8.1 (page 92).* □

Exercice 17 (Difficile) *Compléter le programme* `VaguesDePiquets` *pour dessiner l'alignement de piquets représenté ci-dessous :*



```
program VaguesDePiquets;
  uses   ;
  var I : integer; J : integer; K : integer; L : integer;
begin
  ag;
```

```
  for I := 1 to 8 do
    begin
      for J := 1 to 5 do
        begin
          for K := 1 to 2 do
            begin
              for L := 1 to   do
                begin
                  av;
                end;
              pdt;
            end;
          pqtd; av; pqtg;
        end;
    end;
  st;
end.
```

□

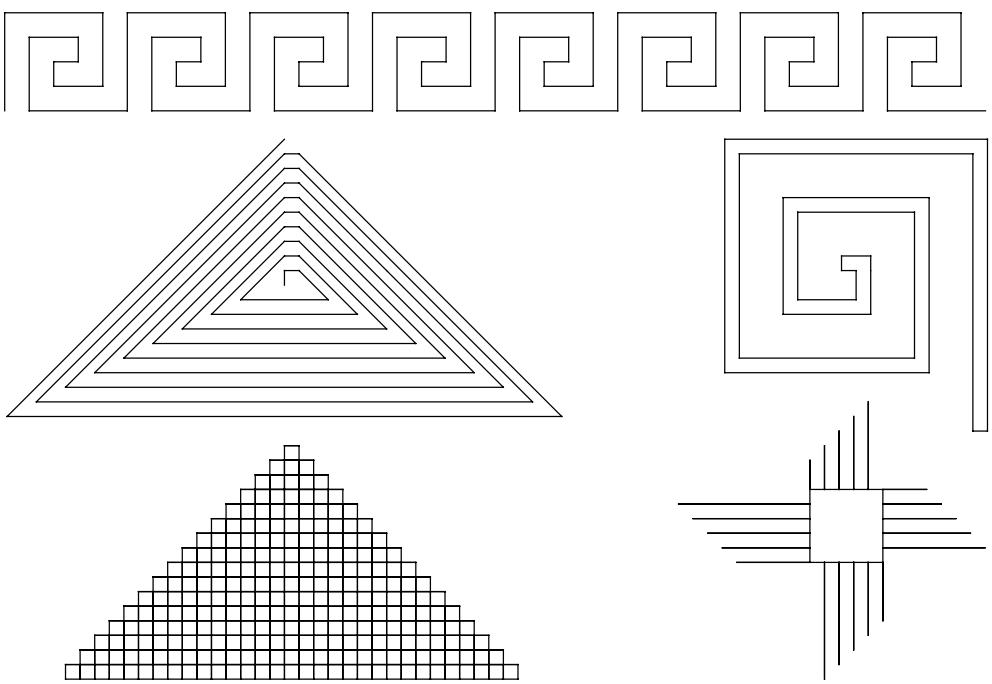


FIG. 8.1 – *Grecque, spirales, pavage triangulaire et carré à franges*

Exercice 18 (Difficile)

1. *Écrire un programme* Pascal *pour dessiner le pavage de Nicomaque*[1] *représenté à la figure 8.2 page 94 en utilisant des boucles "for" imbriquées dépendantes. Décrire les invariants de boucles par des commentaires dans le programme. Le pavage de Nicomaque est obtenu par 10 couronnes constituées de carrés dont les longueurs des côtés sont respectivement 1, 2, 3, 4, ..., 10.*
2. *Calculer la surface du carré de deux façons différentes suggérées par la figure 8.3 page 98 :*
   (a) *en calculant la longueur du demi-côté du carré obtenue en additionnant celles des carrés du pavage,*
   (b) *en calculant la surface du quart du grand carré délimité par les diagonales de ce grand carré (voir cette partie en grisé à la figure 8.3, page 98), la surface de ce quart de grand carré étant obtenue par addition des surfaces des petits carrés le constituant.*
3. *En déduire l'égalité :*

$$1^3 + 2^3 + 3^3 + 4^3 + \ldots + 10^3 = (1 + 2 + 3 + 4 + \ldots + 10)^2$$

□

Corrigé 15 *L'expression entière manquante dans le programme* `PiquetsEnDentsDeScie` *est :*

$$(1 + ((\texttt{I} - 1) \textbf{ mod } 5))$$

qui prend successivement les valeurs 1, 2, 3, 4, 5, 1, 2, 3, *...quand* `I` *prend les valeurs* 1, 2, 3, 4, 5, 6, 7, 8, ...□

Corrigé 16 *Nous donnons un programme, les autres se trouvant sur la disquette d'accompagnement du livre :*

```
program SpiraleTrapezoidale;
 uses   ;
 { Dessiner une spirale trapézoïdale }
 var
   I : integer;
   J : integer;
begin
 av; pqtd;
```

1. Nicomaque de Gérase, mathématicien grecque, env. I^{er} siècle après J.-C.

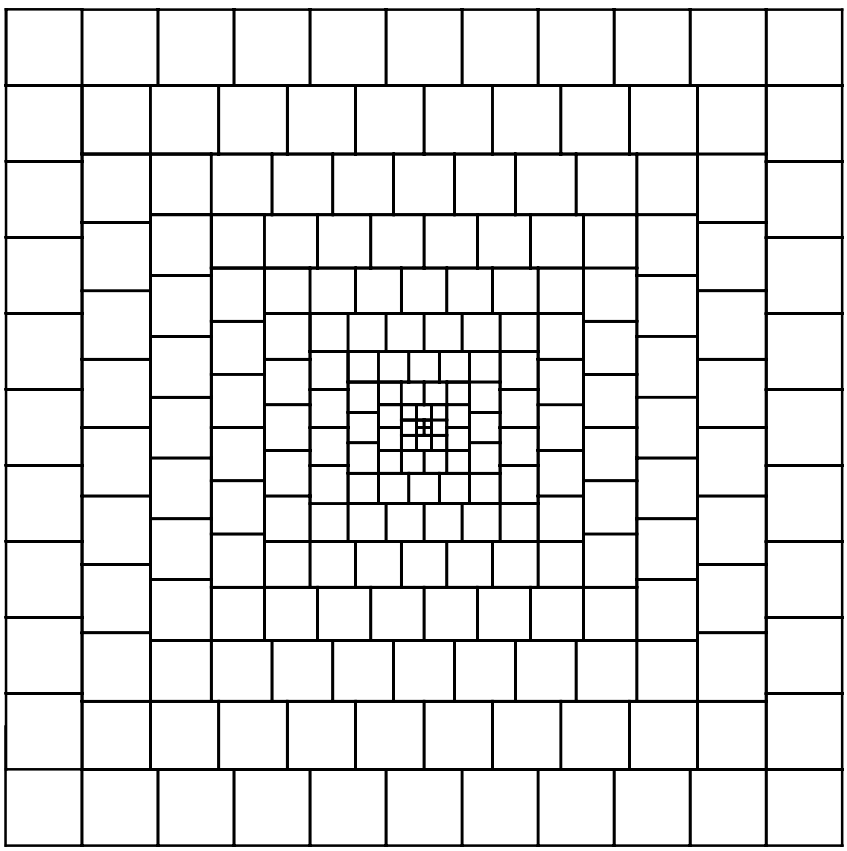


FIG. 8.2 – *Pavage de Nicomaque (Échelle* $\frac{1}{4}$)

```
    for I := 1 to 9 do
      begin
        { Dessiner le côté supérieur }
        av;
        { Dessiner un côté droit }
        phtd;
        for J := 1 to (2 * I) do
          begin
            av;
          end;
        { Dessiner le côté inférieur }
        p3htd;
        for J := 1 to ((4 * I) + 2) do
          begin
            av;
          end;
        { Dessiner un côté gauche }
        p3htd;
        for J := 1 to ((2 * I) + 1) do
          begin
            av;
          end;
        phtd;
      end;
    st;
  end.
```

□

Corrigé 17 *L'expression entière manquante dans le programme* `VaguesDePiquets` *est la suivante :*

$$(((6-\texttt{J}) * ((\texttt{I}+1) \textbf{ mod } 2)) + (\texttt{J} * (\texttt{I} \textbf{ mod } 2)))$$

Quand `I` *est impair,* $((\texttt{I}+1) \textbf{ mod } 2) = 0$ *et* $(\texttt{I} \textbf{ mod } 2) = 1$ *donc l'expression vaut* `J`*. Quand* `I` *est pair,* $((\texttt{I}+1) \textbf{ mod } 2) = 1$ *et* $(\texttt{I} \textbf{ mod } 2) = 0$ *donc l'expression vaut* $(6-\texttt{J})$*.* □

Corrigé 18

```
program PavageDeNicomaque;
  uses  ;

  { Paver un carré à l'aide de couronnes successives constituées }
  { de carrés de tailles 1, 2, 3, ... en partant du centre.      }
```

```
var
  I : integer;
  J : integer;
  K : integer;
  L : integer;
  M : integer;

begin
  for I := 1 to 4 do
    begin
      { Dessiner les 4 côtés de la Ième couronne en les pavant avec }
      { des carrés de taille I.                                     }
      for J := 1 to 4 do
        begin
          { Dessiner un côté de la Ième couronne, pavé de I carrés de }
          { longueur de côté I, le  robot partant du premier carré et }
          { arrivant  au  dernier  carré dans le même coin et dans la }
          { même direction qu'au départ.                              }
           { Dessiner I - 1 carrés de la Ième couronne                 }
            for K := 1 to I - 1 do
              begin
                { Dessiner un carré de longueur I,  le robot partant et }
                { arrivant au même coin et dans la même direction.      }
                for L := 1 to 4 do
                  begin
                    for M := 1 to I do
                      begin
                        av;
                      end;
                    pqtd;
                  end;
                { Avancer jusqu'au carré suivant de la Ième couronne.   }
                pqtd;
                for L := 1 to I do
                  begin
                    av;
                  end;
                pqtg;
              end;
            { Dessiner le Ième carré de longueur I de la Ième couronne. }
            for K := 1 to 4 do
              begin
```

```
            for L := 1 to I do
              begin
                av;
              end;
            pqtd;
          end;
        { Pivoter d'un quart de tour pour placer le robot au début }
        { du côté suivant de la Ième couronne.                     }
        pqtd;
      end;
    { Rejoindre le point de départ de la (I + 1)ème couronne.  }
    pqtg;
    for J := 1 to I do
      begin
        av;
      end;
    pqtd;
    for J := 1 to I do
      begin
        av;
      end;
  end;
st;
end.
```

En partageant un côté du grand carré en deux, la longueur de ce demi-côté est la somme $(1+2+\ldots+10)$ *des longueurs des carrés de tailles 1, 2, 3, ..., 10 du pavage. Ceci apparaît clairement sur la figure 8.3 page 98 grâce à la réglette que nous avons placée en dessous à droite du grand carré. La surface du grand carré est donc le carré de la longueur du côté soit :*

$$(2 \times (1+2+\ldots+10))^2 = 4 \times (1+2+\ldots+10)^2$$

Si l'on partage le grand carré selon les diagonales, on obtient un triangle comme on le voit en grisé à la figure 8.3 (page 98). Ce triangle contient 1 carré de longueur de côté 1, 2 carrés de longueur de côté 2, ..., 10 carrés de longueur de côté 10. Sa surface est donc $1 \times 1^2 + 2 \times 2^2 + \ldots + 10 \times 10^2$ *soit* $1^3 + 2^3 + \ldots + 10^3$*. En faisant pivoter 4 fois ce triangle, on recouvre entièrement le grand carré. La surface du grand carré est donc :*

$$4 \times (1^3 + 2^3 + \ldots + 10^3)$$

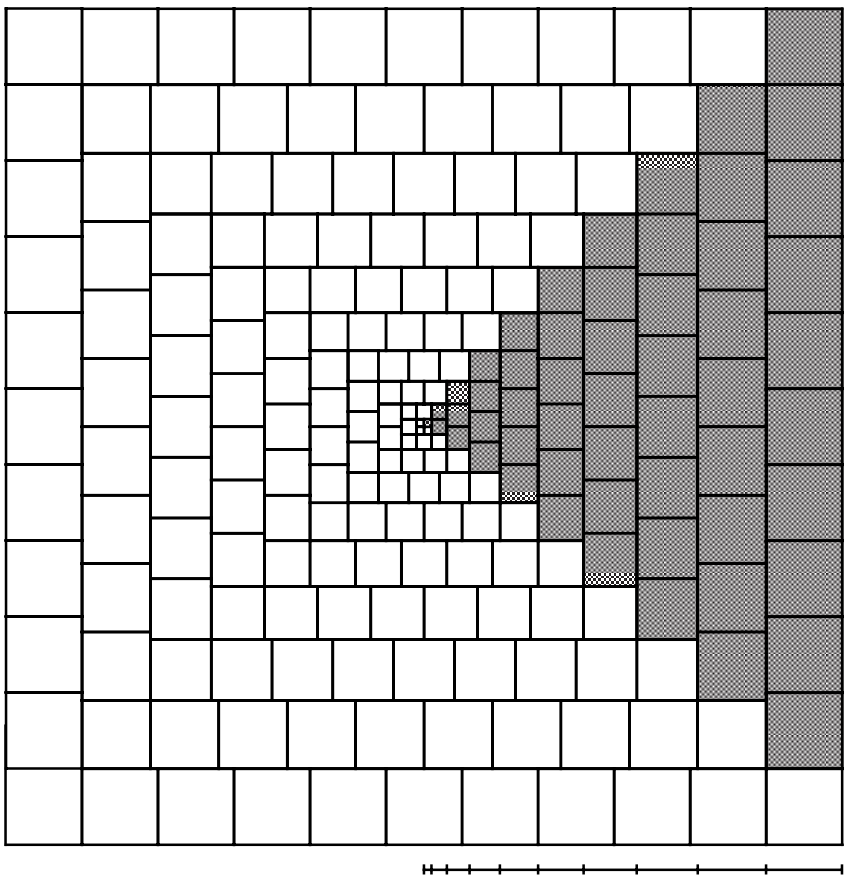


FIG. 8.3 – *Côté et surface du carré de Nicomaque (Échelle* $\frac{1}{4}$*)*

Le résultat est le même quelle que soit la façon de calculer la surface du carré de Nicomaque. Par conséquent :

$$4 \times (1^3 + 2^3 + \ldots + 10^3) = 4 \times (1 + 2 + \ldots + 10)^2$$

soit :

$$1^3 + 2^3 + \ldots + 10^3 = (1 + 2 + \ldots + 10)^2$$

Une autre démonstration de cette identité a été proposée par Al-Karagi[2] *et repose sur le pavage du carré en équerres comme illustré ci-dessous :*

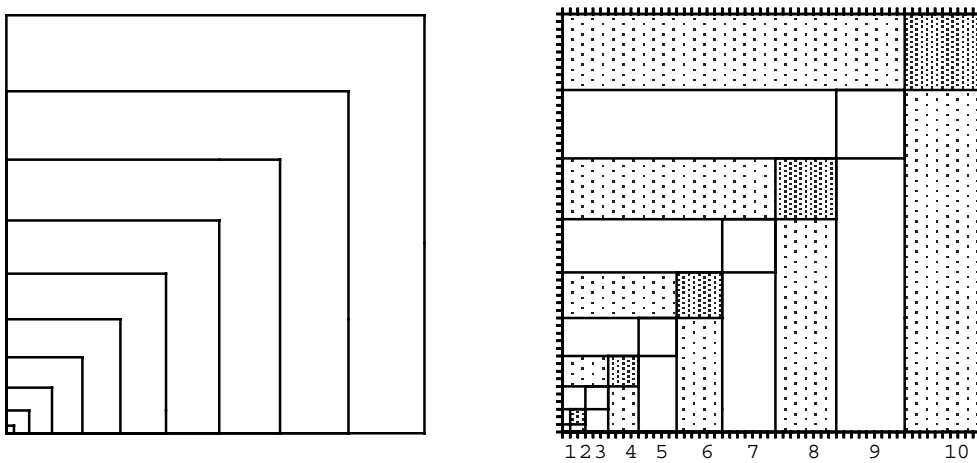


FIG. 8.4 – *Pavage de Al-Karagi*

L'aire de l'équerre extérieure de petit côté 10 vaut :

$$\begin{aligned}(2 \times 10 \times (1 + 2 + \ldots + 10)) - 10^2 &= 2 \times 10 \times \tfrac{10\times(10+1)}{2} - 10^2 \\ &= -10^3 + 10^2 - 10^2 = 10^3\end{aligned}$$

L'aire de l'équerre de petit côté 9 est 9^3*, celle de l'équerre de petit côté 8 est* 8^3*, ..., l'aire du petit carré est* $1 = 1^3$*. Par conséquent, l'aire du grand carré qui est la somme des aires du petit carré et des équerres est* $1^3+2^3+\ldots+10^3$*. D'autre part, la longueur du côté du grand carré est* $(1 + 2 + \ldots + 10)$ *donc sa surface est* $(1 + 2 + \ldots + 10)^2$*. Comme l'aire du grand carré est la même quelque soit la façon de la calculer, on obtient :*

$$1^3 + 2^3 + \ldots + 10^3 = (1 + 2 + \ldots + 10)^2$$

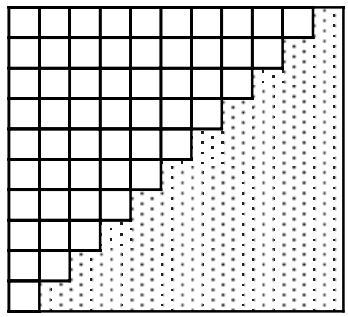


FIG. 8.5 – *Cacul de* $1+2+3+\ldots+10$

Pour justifier que $(1+2+\ldots+10)=\frac{10\times(10+1)}{2}$*, on utilise un rectangle de largeur 10 et de longueur (10 +1) dont la moitié est pavée comme suit :*

Une autre justification géométrique de l'égalité $1^3+2^3+\ldots+10^3=(1+2+\ldots+10)^2$ *proposée par S. Golomb [13] est illustrée ci-dessous :*

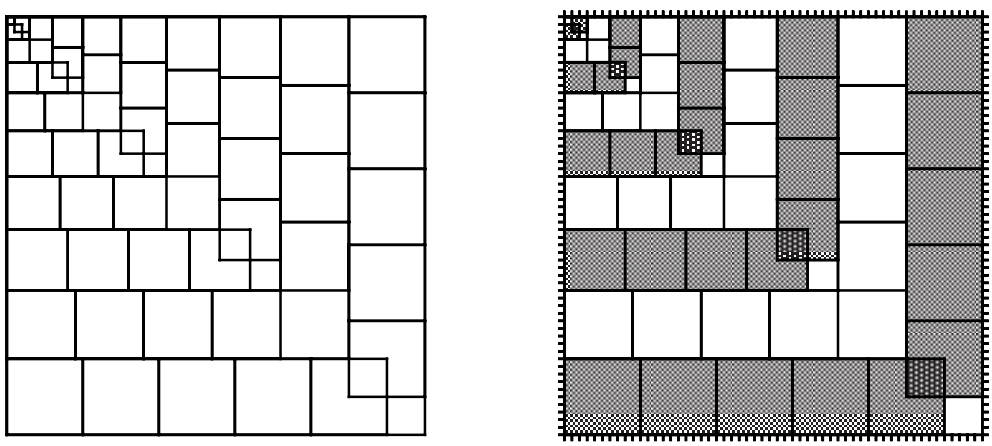


FIG. 8.6 – *Pavage de Golomb*

La longueur du côté du grand carré est bien $(1+2+\ldots+10)$ *donc sa surface est* $(1+2+\ldots+10)^2$*. Cette surface est égale à la somme des surfaces des carrés constituant le pavage soit 1 carré de côté 1, 2 carrés de côté 2, ..., 10 carrés de côté 10. On remarque que pour les carrés de longueur de côté paire montrés en gris clair sur la figure, il y a un recouvrement de deux petits carrés figuré en gris foncé. Mais chacun de ces petits carrés est adjacent à une*

2. Al-Karagi, fin X$^{\text{ème}}$ – début XI$^{\text{ème}}$ siècle, mathématicien persan.

région non couverte de même taille montrée en blanc. On peut donc enlever un des deux petits carrés qui se recouvrent pour boucher le trou et obtenir un pavage parfait. La surface est donc égale à $1^3 + 2^3 + \ldots + 10^3$. □

Addenda

Pour simplifier l'apprentissage de la syntaxe de PASCAL nous écrivons les expressions entières avec des parenthèses. Les expressions entières *unaires* (avec un seul *opérande* E) sont de la forme $(-E)$ où $-$ est l'*opérateur* de signe moins. Les expressions entières *binaires* (avec un *opérande* gauche E_1 et un opérande droit E_2) sont de la forme $(E_1\ \mathbf{op}\ E_2)$ où **op** est l'un des *opérateurs* d'addition $+$, de soustraction $-$, de multiplication $*$, de division entière **div** ou de reste de la division entière **mod** (modulo). Les opérandes E, E_1 et E_2 sont une constante entière (positive comme 1, 17345 (sans blancs), ...ou négative comme -10, -5247, ...), une variable (déclarée sous la forme **var** `I : integer;` et utilisée pour l'instant comme compteur de boucle "for") ou une *sous-expression* entière de la forme que nous venons de décrire.

Le calcul de la valeur d'une expression s'appelle l'*évaluation* de l'expression. Les règles d'évaluation des expressions complètement parenthésées sont très simples. La valeur d'une constante est l'entier qui est dénoté par cette constante. La valeur d'une variable `I` utilisée comme compteur dans le corps C d'une boucle **for** `I` $:=$ `1` **to** n **do begin** C **end**; (où $n \geq 1$ est une constante entière) est successivement 1 puis 2 puis ...puis n lors des itérations successives dans la boucle. La valeur d'une expression $(-E)$ s'obtient en calculant la valeur v de l'expression E puis en lui appliquant le signe moins. La valeur d'une expression $(E_1\ \mathbf{op}\ E_2)$ s'obtient en calculant d'abord la valeur v_1 de l'expression E_1 puis en calculant la valeur v_2 de l'expression E_2 puis en calculant la valeur $v_1\ \mathbf{op}\ v_2$ du résultat. Les mêmes règles d'évaluation s'appliquent aux sous-expressions E, E_1 et E_2.

Pour éviter les lourdeurs d'écriture, on peut suivre en PASCAL la pratique courante en mathématiques qui consiste à utiliser des *règles de priorité* (on dit aussi *règles de précédence*) permettant d'éliminer certaines parenthèses redondantes. Les règles d'évaluation des expressions entières sont définies et illustrées ci-après :

- En PASCAL comme en mathématiques, l'expression entière $10 - 10 - 10$ vaut $((10 - 10) - 10)$ soit -10. De même $2 * 7\,\mathbf{div}\ 3\,\mathbf{mod}\,2$ vaut $(((2 * 7)\,\mathbf{div}\,3)\,\mathbf{mod}\,2)$ soit 0. Dans les deux cas la règle est que des opérateurs de même *priorité* (à savoir $+$ et $-$ d'une part ; $*$, **div** et **mod** d'autre part) s'évaluent de gauche à droite.

- L'expression $8 + -2 * 3$ vaut $(8 + ((-2) * 3))$ soit 9. De même $1 - 5\,\mathbf{div}\,2$ vaut $(1 - (5\,\mathbf{div}\,2))$ soit -1. La règle est qu'en présence d'opérateurs de priorités différentes, on commence par évaluer les opérateurs de plus forte priorité. En PASCAL les priorités sont les suivantes :

Priorité des opérateurs entiers		
$-$ (unaire)	opérateurs unaires	priorité forte
$*$ **div mod**	opérateurs multiplicatifs	priorité moyenne
$+$, $-$ (binaires)	opérateurs additifs	priorité faible

- L'expression $(8+-2)*3$ vaut $((8+(-2))*3)$ soit 18. De même $(1 - 5)\,\mathbf{div}\,2$ vaut $((1-5)\,\mathbf{div}\,2)$ soit -2. Dans les deux cas on commence par l'évaluation des sous-expressions entre parenthèses.

De manière équivalente, l'ordre d'évaluation des expressions peut être défini en indiquant à quels opérateurs sont liés les opérandes d'une expression :

- Un opérande figurant entre deux opérateurs de priorités différentes est lié à l'opérateur de plus forte priorité (donc $2 + 3 * 5 = (2 + (3 * 5))$).
- Un opérande figurant entre deux opérateurs de même priorité est lié à l'opérateur de gauche (donc $2 - 3 - 5 = ((2 - 3) - 5)$).
- Les expressions entre parenthèses sont évaluées avant d'être utilisées comme opérandes.

Les règles d'évaluation des expressions changent d'un langage de programmation à l'autre. Par exemple en APL [17], il n'y a pas de priorités d'opérateurs (car il y en a trop dans le langage pour qu'on puisse espérer s'en souvenir) et les expressions sont associatives à droite de sorte que $10 - 10 - 10 = (10 - (10 - 10)) = (10 - 0) = 10$. Cette règle contraire à la tradition mathématique a l'avantage que l'analyse grammaticale des expressions est grandement simplifiée pour l'ordinateur. Les langages de programmation plus classiques qu' APL suivent la convention mathématique d'évaluation de gauche à droite avec priorités. Cependant certains langages possèdent beaucoup d'opérateurs. Par exemple le langage C [19] comprend 43 opérateurs différents répartis en 15 niveaux de priorité. Les priorités des opérateurs ne sont donc pas faciles à retenir. De plus elles changent d'un langage à l'autre ce qui rend la vie difficile aux informaticiens pratiquant journellement plusieurs langages. Pour ne pas se tromper ils utilisent des parenthèses redondantes, ce que nous conseillons aux débutants de faire.

Signalons enfin qu'en PASCAL la valeur des expressions entières doit être comprise entre deux valeurs minimales et maximales qui sont généralement -32768 et 32767 ou -2147483648 et 2147483647 voire plus selon le modèle d'ordinateur utilisé.

9

Appel de procédures avec paramètres

9.1 Faire avancer et reculer le robot

Nous avons vu au chapitre 5 qu'une déclaration de procédure permet de donner un nom à une séquence de commandes que le *Robot* peut ensuite exécuter une ou plusieurs fois quand on appelle la procédure par son nom. Une procédure peut avoir un ou plusieurs *paramètres*. Dans ce cas la procédure peut réaliser des actions semblables mais différentes quand on l'appelle avec des paramètres différents.

Par exemple la procédure `avf` commande au *Robot* d'avancer tout droit une ou plusieurs fois de la taille d'un côté d'un carreau du quadrillage. Elle a un paramètre qui indique le nombre de fois qu'il faut avancer. '`avf(1)`' est équivalent à '`av`' tandis que '`avf(4)`' est équivalent à '`av; av; av; av;`'. Quand le paramètre est nul, la commande '`avf(0)`' ne fait pas changer le *Robot* de position. Enfin quand le paramètre est un nombre négatif, c'est-à-dire un nombre précédé du signe moins $-$, le *Robot* recule du nombre de fois indiqué sans changer d'orientation. Par exemple '`avf(−4)`' est équivalent à '`pdt; av; av; av; av; pdt;`'.

Exemple 13 Le programme `StriesOctogonales` ci-dessous dessine les stries triangulaires réparties sur les côtés d'un octogone invisible représentées à la figure 9.1 (page 104) :

```
program StriesOctogonales;
 uses   ;
```

```
  var
    I : integer;
    J : integer;
begin
  lg(4);
  for I := 1 to 8 do
    begin
      { Dessiner 10 stries de longueurs 2, 4, 8, ...20. }
      for J := 1 to 10 do
        begin
          { Dessiner une strie de longueur (2 * J) centrée sur un côté }
          { de l'octogone. }
          avf(J); avf(−2 * J); avf(J);
          { Passer à la strie suivante (sans tracer le côté de l'octo- }
          { gone). }
          lc; pqtd; av; pqtg; bc;
        end;
      { Tourner le robot perpendiculairement au côté suivant. }
      phtd;
    end;
  st;
end.
```

□

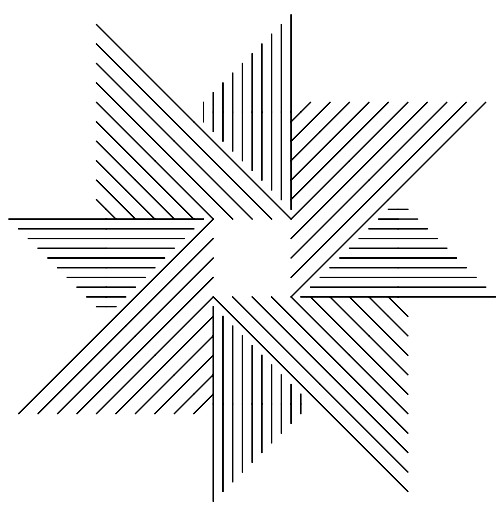


FIG. 9.1 – *Stries sur octogone (Échelle $\frac{1}{3}$)*

9.2 Épaisseur du crayon

La commande `ec` définit l'épaisseur du crayon du $\mathcal{R}\mathcal{o}\mathcal{b}^{o}t$. Son paramètre entier indique l'épaisseur du crayon qui est choisie. Jusqu'ici nous avons employé un crayon d'épaisseur 1 qui est celle qu'utilise le $\mathcal{R}\mathcal{o}\mathcal{b}^{o}t$ par défaut c'est-à-dire quand on ne lui donne pas d'autres précisions. La commande `ec(2)` permet de dessiner avec un trait deux fois plus gros. L'unité de longueur choisie pour mesurer l'épaisseur du crayon est le *pixel* c'est-à-dire, rappelons-le, la taille du plus petit point qui peut être dessiné sur l'écran de l'ordinateur.

Quand on utilise un très gros crayon[1], il faut savoir que le crayon du $\mathcal{R}\mathcal{o}\mathcal{b}^{o}t$ a une mine carrée et que le $\mathcal{R}\mathcal{o}\mathcal{b}^{o}t$ tient cette mine par son coin supérieur gauche.

Exemple 14 (Crayon à grosse mine) Le programme `EpaisseurCrayon` ci-dessous permet d'observer l'effet de l'exécution d'une commande `av` avec une grille standard de 12 pixels quand le $\mathcal{R}\mathcal{o}\mathcal{b}^{o}t$ utilise des épaisseurs successives du crayon de 12 , 24 , 36 et 48 pixels :
□

```
program EpaisseurCrayon;
  uses  ;
  var I : integer;
begin
  eh; pqtd;
  for I := 1 to 4 do
    begin
      lc; pqtd; avf(I); pqtg; bc;
      ag; ec(12 * I); av;
    end;
  dg; st;
end.
```

On observera par exemple que le trait tracé quand le $\mathcal{R}\mathcal{o}\mathcal{b}^{o}t$ exécute une commande `av` avec un crayon de 48 pixels a une longueur apparente de 5 fois la longueur du côté du quadrillage. Ceci s'explique par le fait qu'en position finale le crayon laisse une trace carrée de côté 48 pixels en bas à droite de son point d'arrêt soit 4 fois la longueur du côté du quadrillage de 12 pixels.
□

1. Sur compatibles IBM PC, les seules épaisseurs possibles du crayon sont 1 et 3.

9.3 Vitesse du robot

Un dernier exemple de procédure avec paramètre est la commande `vt` pour choisir la vitesse du *Robot*. Cette vitesse est un nombre entier compris entre 0 et 10. La vitesse 10 correspond à la vitesse maximale par défaut. Donc la commande `vt(10)` est équivalente à la commande `ex` d'exécution normale. La vitesse `vt(1)` correspond à la vitesse minimale. Elle permet d'observer tous les détails des mouvements du *Robot* qui, à cette vitesse, sont très lents. Enfin la vitesse 0 correspond à l'exécution en mode pas à pas. La commande `vt(0)` est donc équivalente à la commande `pp`. Le mode pas à pas est utilisé pour trouver les erreurs dans les programmes. Dans ce mode d'exécution, les commandes du *Robot* s'affichent en bas de l'écran et s'exécutent en tapant sur une touche (ou en cliquant sur le bouton de la souris). Les parcours du *Robot* avec le crayon levé sont marqués en pointillés ce qui permet de suivre parfaitement sa trace.

Exercice 19 *Ecrire un programme* Pascal *pour dessiner :*

- *la lettre H majuscule en relief de la figure 1.12 (page 9) ;*
- *le cube en perspective, le cristal à huit branches et la paire de lunettes de la figure 9.2 (page 107) ;*
- *l'hélicoptère et le train de voyageurs de la figure 1.13 (page 10) ;*
- *les courbes 'spirolatérales' de la figure 9.3 (page 107), obtenues en faisant avancer le Robot une fois puis deux fois puis trois fois, etc., en tournant à chaque fois d'un quart de tour à droite et en répétant le tout quatre fois de suite ([28]) ;*
- *l'arabesque et l'échelle de Jacob de la figure 9.4 (page 107), échelle de Jacob que l'on pourra également essayer de réaliser en entrelaçant une ficelle entre ses doigts.*

□

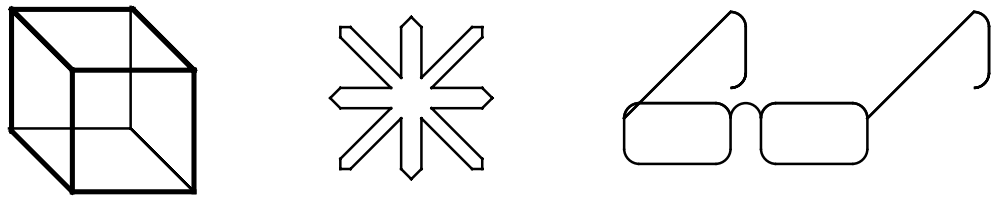


FIG. 9.2 – *Cube en perspective, cristal à huit branches et paire de lunettes*

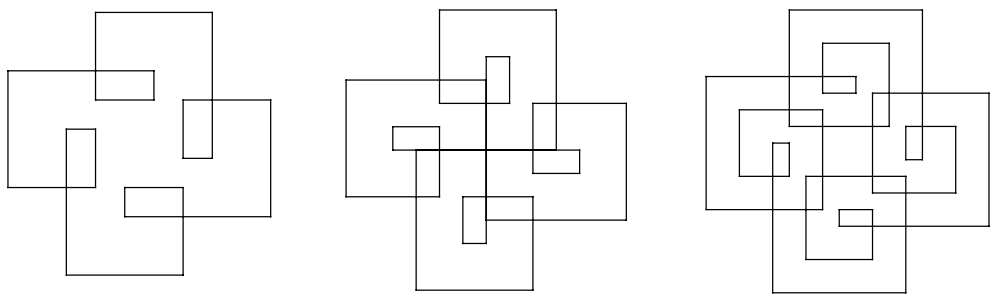


FIG. 9.3 – *Spirolatérales à 90° d'ordre 5, 7 et 9*

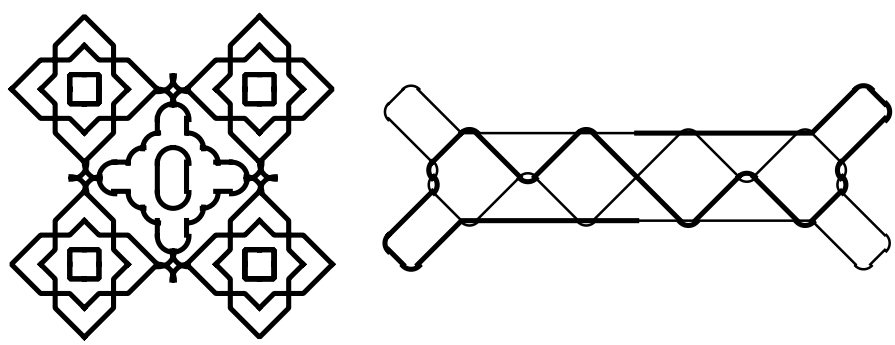


FIG. 9.4 – *Arabesque et échelle de Jacob*

Corrigé 19

```
program Spirolaterale90oDordre9;
  uses  ;
  { Dessiner une courbe spirolatérale de Frank C. Odds à 90 degrés }
  { d'ordre 9.                                                      }
  var
    I : integer;
    J : integer;
begin
  for I := 1 to 4 do
    begin
      for J := 1 to 9 do
        begin
          avf(J); pqtd;
        end;
    end;
  st;
end.
```

□

Addenda

L'usage de paramètres dans les procédures est une des notions de base de la programmation : chaque appel de la procédure dépend d'une ou plusieurs valeurs en fonction desquelles on exprime l'effet de la procédure. Nous ne considérons pour l'instant que le passage de paramètres *par valeur* qui permet de passer une information (un ou plusieurs nombres dans notre cas) à une procédure. Ceci rejoint la notion élémentaire de *fonction* (ou *application*) en mathématiques.

L'emploi de sous-programmes avec paramètres en langage machine est généralement attribué à Alan Turing. Le suisse Heinz Rutishauser [31] et l'italien Corrado Böhm [5] introduisirent les notions de langage de programmation évolué et de compilateur pour traduire les programmes écrits dans ce langage évolué en langage machine (on ne parlait pas à l'époque de compilateur mais de *codification automatique*). Ils incluèrent la notion de sous-programme avec paramètres dans leurs langages sans la dégager très explicitement.

Après ces contributions théoriques liminaires, le premier compilateur qui ait fonctionné sur la machine MARK I à Manchester en Angleterre fût construit par Alick Glennie en 1952. Le language évolué AUTOCODE utilisait des sous-programmes avec paramètres passés dans les registres de la machine.

Ce travail n'eut aucun succès car le travail de programmation était relativement aisé à l'époque comparé aux efforts nécessaires pour adapter les programmes aux dysfonctionnements de la machine. Wilkes [39] introduisit l'usage de bibliothèques de procédures (comme celles que nous utilisons pour les commandes du $\mathcal{R}^o\!\!\smile^o t$) en 1951.

Le premier langage où ces notions apparurent sous leur forme moderne fut FORTRAN (FORmula TRANslator) conçu en 1954 par une équipe dirigée par John Backus [1]. Grâce à de très nombreuses modifications qui incorporent les nouveaux concepts de programmation, aux extensions qui prennent en compte les architectures récentes de machines et à la richesse des bibliothèques disponibles, ce langage prédomine encore dans la programmation scientifique numérique.

10

Grille de déplacement du robot

Jusqu'à présent nous avons tracé nos dessins sur du papier quadrillé à petits carreaux. Le *Robot* a la faculté supplémentaire de changer la taille des carreaux. On peut donc agrandir les dessins en augmentant la taille des carreaux avant de commencer le dessin. Il est également possible de changer la taille des carreaux pendant que le *Robot* dessine. Ceci permet de tracer des segments de longueurs différentes et des arcs de cercles dont le rayon peut varier. En choisissant des carreaux rectangulaires on peut tracer des ellipses et choisir de manière quelconque les angles de déplacement du *Robot*.

10.1 Grille carrée

La grille standard représentée à la figure 2.2 (page 14) a une taille de 12 pixels. On peut changer la longueur du côté d'un carreau de la grille avec la commande `lg` qui prend en paramètre la nouvelle longueur de la grille exprimée en pixels. Par exemple `lg(6)` définit une taille de grille deux fois plus petite que la grille habituelle tandis que `lg(24)` double la taille de la grille.

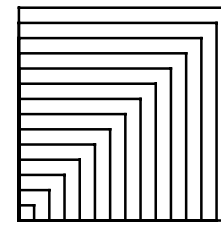
Exemple 15 (Carrés superposés) Le programme ci-dessous permet de dessiner des carrés superposés, le *Robot* partant et arrivant au coin inférieur gauche, orienté vers le nord, la superposition étant obtenue en choisissant des tailles de grille successives de 6, 12, 18, ..., 84 :

```
program CarresSuperposes2;
 uses   ;
```

```
procedure Carre;
begin
  av; pqtd; av; pqtd; av; pqtd; av; pqtd;
end;

var I : integer;

begin
  eb; ag;
  for I := 1 to 14 do
    begin
      lg(6 * I); Carre;
    end;
  st;
end.
```



□

10.2 Grille rectangulaire

Le *Robot* peut se promener sur une grille rectangulaire dont la longueur du côté horizontal est différente de celle du côté vertical. Ceci lui permet de dessiner des rectangles et des ellipses comme ceux-ci :

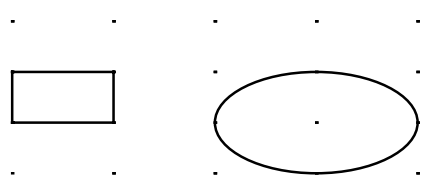


FIG. 10.1 – *Rectangle et ellipse sur un quadrillage* 40×20

La grille initiale a une taille 12×12 comptée en pixels. La commande `lgX(40)` permet de donner une longueur de 40 pixels au côté horizontal de la grille sans changer la longueur du côté vertical. Après cette commande la grille a donc une taille 40×12 pixels. Dans cette notation 40×12 on donne d'abord la longueur du côté horizontal (nommé X) puis la longueur du côté vertical (nommé Y) de la grille. La commande `lgY(20)` permet de donner une longueur de 20 pixels au côté vertical de la grille sans changer la longueur du côté horizontal. Après cette commande la grille a donc une taille 40×20 pixels. On peut donc changer indépendamment la taille horizontale

(avec `lgX`) ou la taille verticale (avec `lgY`) de la grille. On peut revenir à une grille carrée en utilisant la commande `lg`. Par exemple la commande `lg(12)` permet de revenir à une grille carrée standard 12×12. Le rectangle et l'ellipse ci-dessus sont construits par le programme suivant :

```
program RectangleEtEllipse;
 uses  ;
 { Dessiner un rectangle et une ellipse juxtaposés. }
begin
 { Taille de la grille } lgX(40); lgY(20); ag;
 { Grille } dg;
 { Rectangle } av; pqtd; av; pqtd; av; pqtd; av; pqtd;
 { Décalage à droite. } lc; pqtd; av; av; pqtg; bc;
 { Ellipse } vd; vd; vd; vd;
 st;
end.
```

La figure ci-dessous illustre l'effet des commandes `vd`, `av` et `vg` avec une grille rectangulaire de dimensions 30×60.

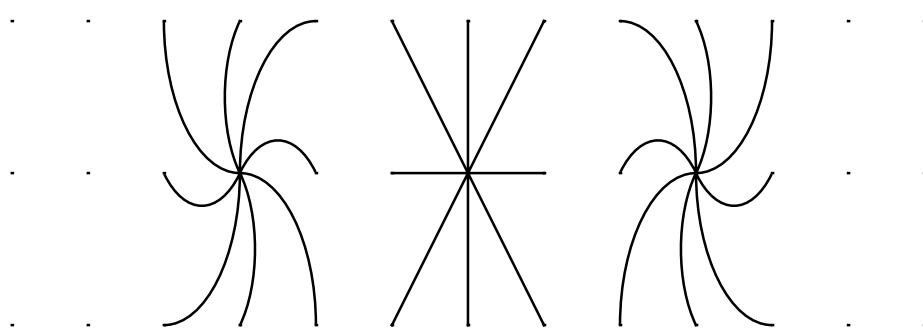


FIG. 10.2 – *Virage à droite, avancer et virage à gauche avec une grille* 30×60

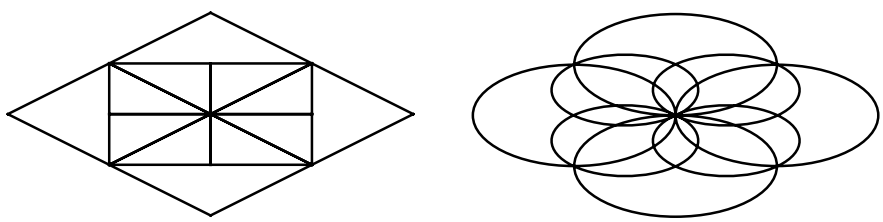


FIG. 10.3 – *Parallélogrammes et ellipses*

Exemple 16 Les divers tracés de segments et d'arcs d'ellipses par les commandes `av`, `vg` et `vd` sont illustrés par les deux programmes ci-dessous qui

dessinent des parallélogrammes et des ellipses par pivotement d'un quart de tour autour du centre de la grille, comme on le voit à la figure 10.3 (page 113) :

```
program Parallelogrammes;
  uses  ;
  var I : integer;
begin
  { Définir la taille de la grille 40 x 20. }
  lgX(40); lgY(20);
  { Dessiner huit parallélogrammes en faisant pivoter leur coin infé- }
  { rieur gauche d'un quart de tour autour du centre de la grille.    }
  for I := 1 to 8 do
    begin
      av; pqtd; av; pqtd; av; pqtd; av; phtd;
    end;
  st;
end.
```

```
program Ellipses;
  uses  ;
  var I : integer;
begin
  { Définir la taille de la grille 40 x 20. }
  lgX(40); lgY(20);
  { Dessiner huit ellipses en faisant pivoter leur grand axe d'un }
  { quart de tour autour du centre de la grille.                  }
  for I := 1 to 8 do
    begin
      phtd; vd; vd; vd; vd;
    end;
  st;
end.
```

□

Exercice 20 *Utiliser les possibilités offertes par les changements de taille de la grille carrée pour dessiner :*

- *les spirales et cercles concentriques de la figure 10.4 (page 115) ;*
- *la frise de la figure 10.5 (page 115) ;*
- *l'hélice et la balle de la figure 10.6 (page 116) ;*
- *l'escargot de la figure 10.7 (page 116) ;*

– *le stégosaure et son bébé de la figure 10.8 (page 116) ;*

– *la famille de chiens de la figure 10.9 (page 117) ;*

– *le pavage de la figure 10.13 (page 118) inspiré par un motif de van de Vecht ([24], page 26) représenté à la figure 10.10 (page 117).* □

Exercice 21 *Utiliser les possibilités offertes par les grilles rectangulaires pour dessiner :*

– *la frise de losanges se chevauchant de la figure 10.11 (page 117) ;*

– *le vélo de la figure 10.12 (page 117) ;*

– *la rose des vents à huit directions 2.1 (page 14) ;*

– *les pavages des figures 10.14 (page 119), 10.15 (page 120) et 10.16 (page 121) obtenues à partir du pavage de la figure 10.13 (page 118) en diminuant progressivement la taille de la grille sur le côté droit du dessin, sur le côté gauche et dans le dernier cas sur les quatre côtés du rectangle à la manière du graveur hollandais M. C. Escher ([24], [10]).*

□

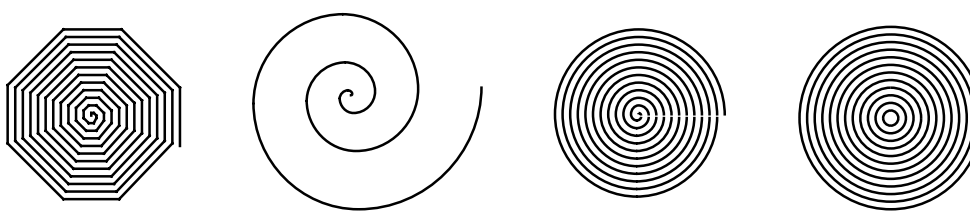


FIG. 10.4 – *Spirales et cercles concentriques*

FIG. 10.5 – *Double frise crantée*

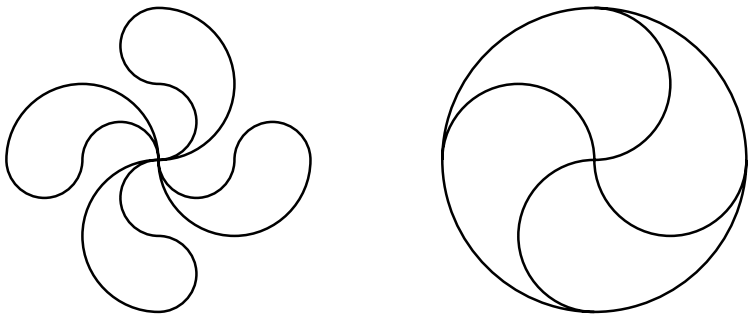


FIG. 10.6 – *Hélice et balle*

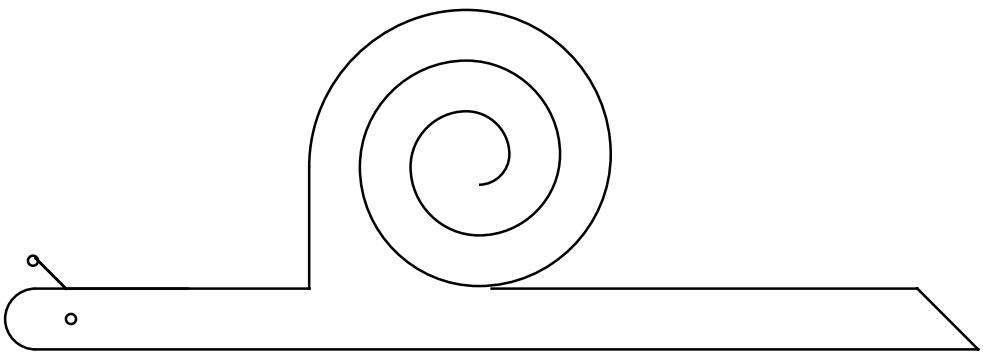


FIG. 10.7 – *Escargot*

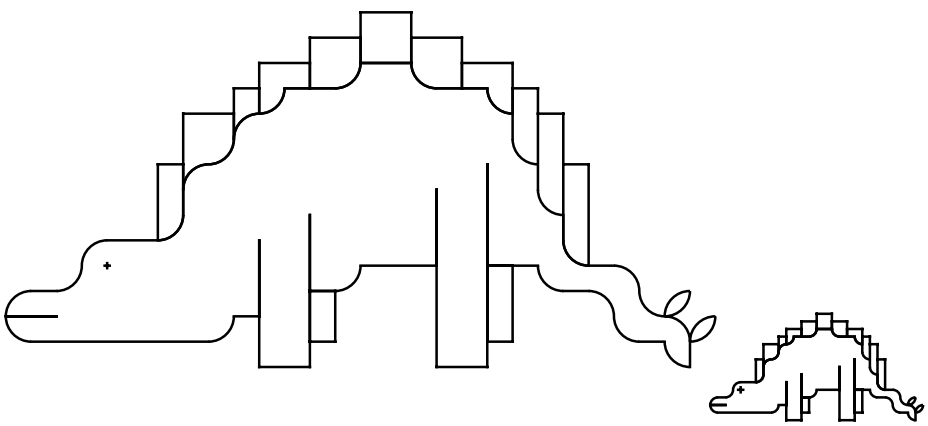


FIG. 10.8 – *Stégosaure et son bébé*

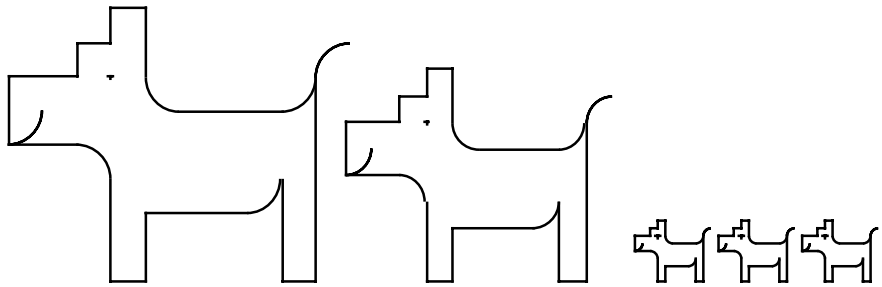


FIG. 10.9 – *Famille de chiens en balade*

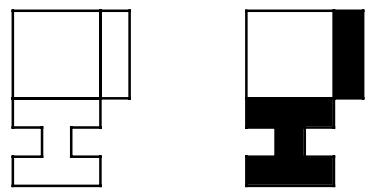


FIG. 10.10 – *Motif ornemental de N. J. van de Vecht*

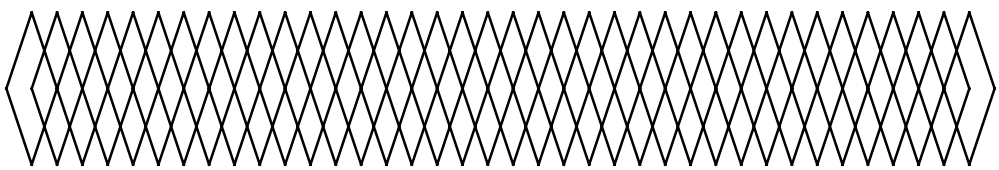


FIG. 10.11 – *Frise de losanges se chevauchant*

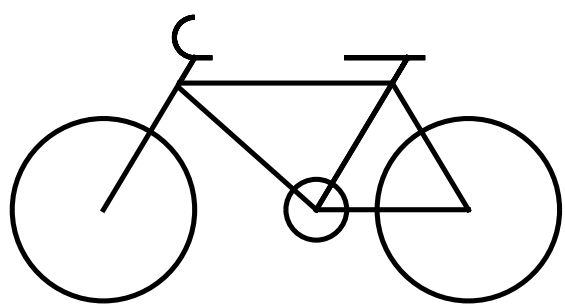


FIG. 10.12 – *Vélo*

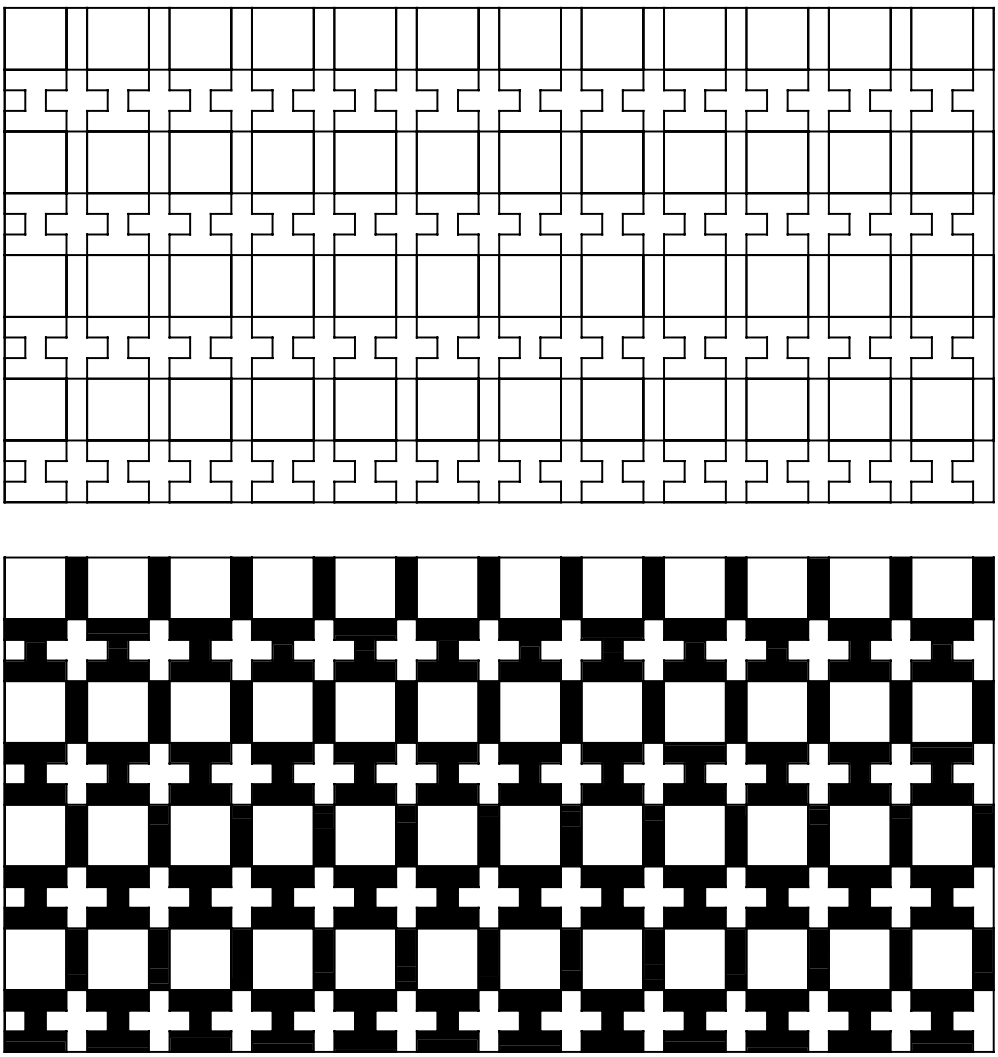


FIG. 10.13 – *Pavage régulier inspiré par un motif ornemental* de N. J. van de Vecht

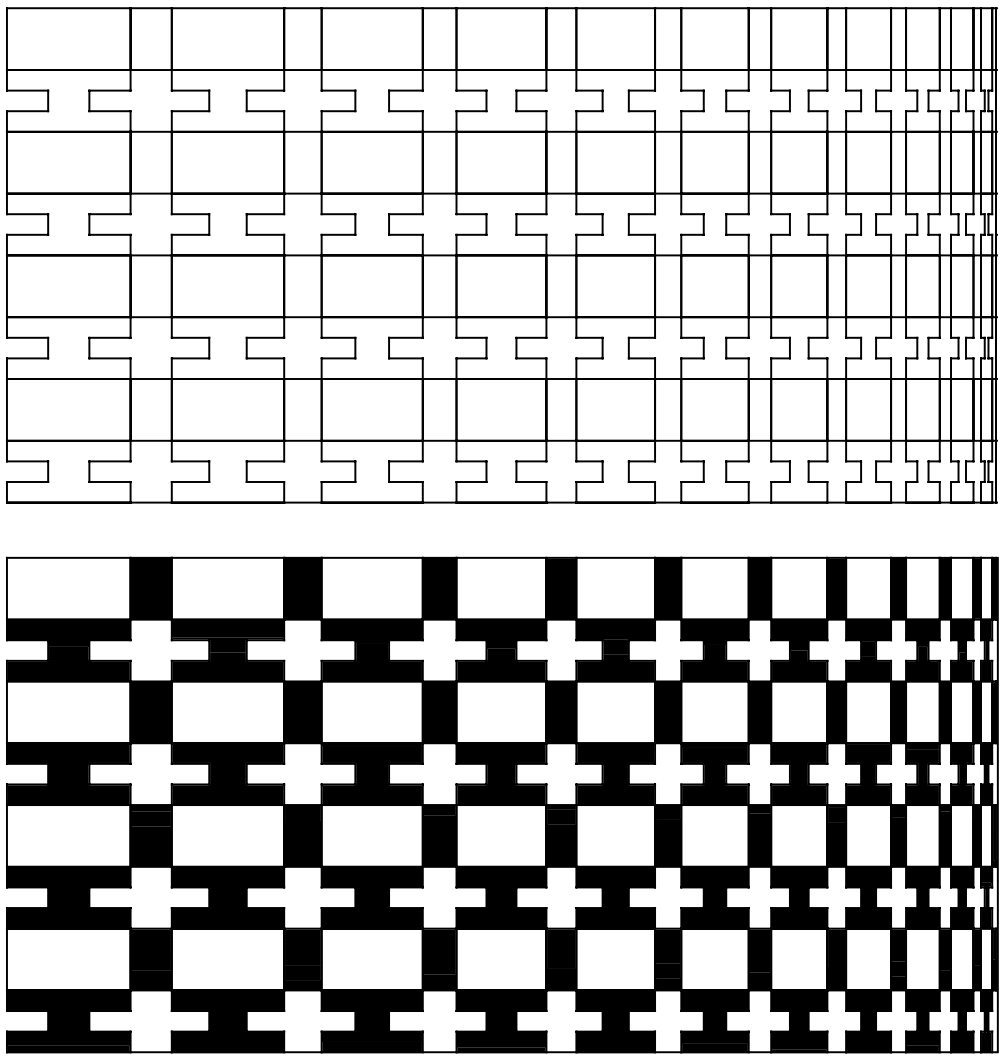


FIG. 10.14 – *Pavage rectangulaire à une limite*

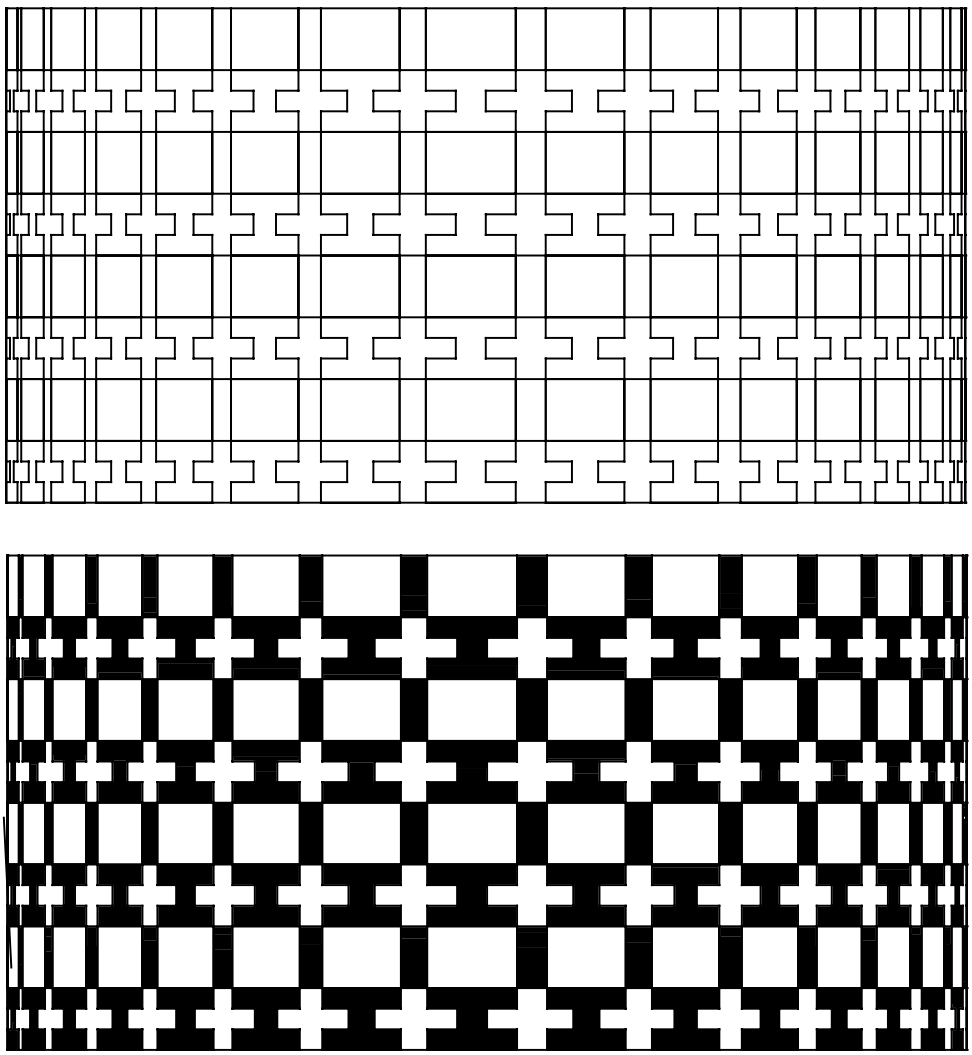


FIG. 10.15 – *Pavage rectangulaire à deux limites*

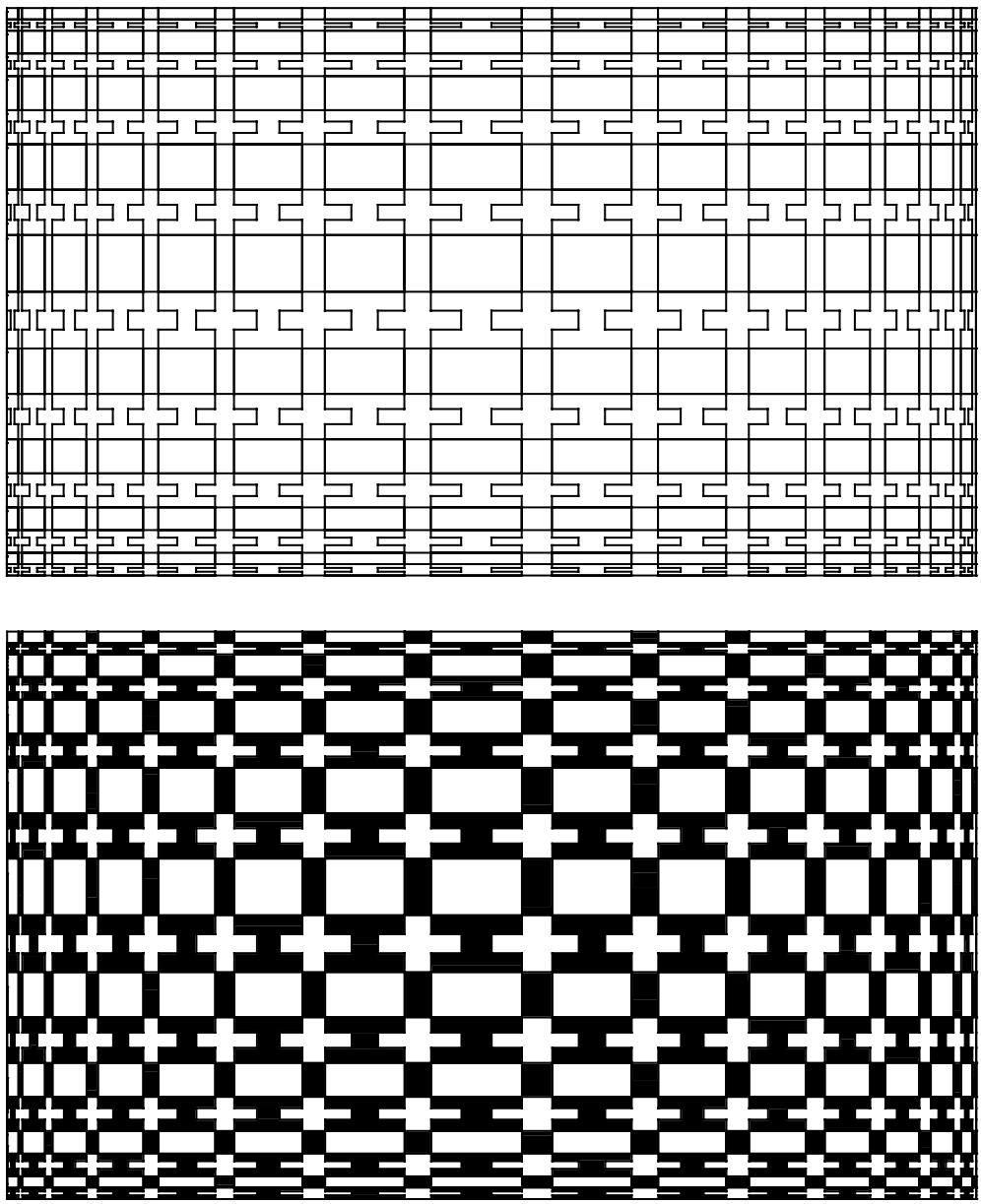


FIG. 10.16 – *Pavage rectangulaire à quatre limites*

Corrigé 20 *Nous donnons quelques programmes, les autres programmes se trouvant sur la disquette d'accompagnement du livre :*

- ```
  program SpiraleGeometrique;
    uses  ;
    var I : integer;
  begin
    pdt;
    for I := 1 to 12 do
      begin
        lg(I * I); vg;
      end;
    st;
  end.
  ```

- ```
  program Escargot2;
    uses  ;
    var I : integer;
  begin
    { Monter au centre de la coquille }
    lc; avf(4); bc; pqtd;
    { Coquille }
    for I := 1 to 11 do
      begin
        vg; lg(12 + (5 * I));
      end;
    lg(12); avf(4);
    { Corps }
    pqtd; avf(9); vg; vg; avf(31); pdt; phtd; avf(2); phtg; avf(14);
    { Rejoindre l'antenne }
    lc; avf(10); bc; avf(4);
    { Antenne }
    phtd; av; lg(2); vg; vg; vg; vg; lg(12); pdt; av;
    { Oeil }
    phtd; lc; av; bc; lg(2); vg; vg; vg; vg;
    st;
  end.
  ```

□

Corrigé 21 *Nous donnons le programme le plus difficile, à savoir le pavage de la figure 10.16 (page 121) obtenue à partir du pavage de la figure 10.13 (page 118) en diminuant progressivement la taille de la grille sur les quatre côtés du rectangle. Les autres programmes se trouvent sur la disquette d'accompagnement du livre :*

```
program PavageVdV4limites;
 uses  ;

 { Pavage irrégulier dans le style de  M. C. Escher,  inspiré d'un }
 { motif de N. J. van de Vecht.                                     }

 procedure Motif;
  { Motif de N. J. van de Vecht, le robot partant en haut à gauche }
  { et arrivant en bas à gauche orienté vers le nord.              }
 begin
  pqtd; avf(4); pqtd; avf(3); pqtd; av; pqtd; avf(3); pdt; avf(3);
  av; pqtd; av; pqtg; av; pqtg; av; pqtd; av; pqtd; avf(3);
  pqtd; av; pqtd; av; pqtg; av; pqtg; av; pqtd; av; pqtd; avf(3);
  pdt; avf(3); pqtd; avf(3); lc; pdt; avf(6); pdt; bc;
 end;

 procedure Colonne;
  var I : integer;
  { Dessiner une colonne de motifs de hauteurs  2, 4, 6, 8, 10, 8, }
  { 6, 4, 2,  le robot partant en haut à gauche et arrivant en bas }
  { à droite (c'est-à-dire en bas à gauche de la colonne suivante) }
  { orienté vers le nord.                                          }
 begin
  lgY(1); eh;
  for I := 1 to 5 do
    begin
      lgY(2 * I); Motif;
    end;
  for I := 1 to 4 do
    begin
      lgY(10 - (2 * I)); Motif;
    end;
  pqtd; avf(4); pqtg;
 end;

 procedure Pavage;
  { Dessiner un pavage irrégulier constitué de colonnes de largeurs }
  { 0, 2, 4, 6, ..., 14, 16, 14, ..., 2, 0.                         }
  var I : integer;
 begin
  lgX(1); ag; lgX(0);
  for I := 1 to 8 do
    begin
      Colonne; lgX(2 * I);
    end;
```

```
    for I := 1 to 8 do
      begin
        Colonne; lgX(16 - (2 * I));
      end;
    Colonne;
  end;
begin
  Pavage; st;
end.
```

□

Addenda

Si toutes les grilles des points d'arrêt du $\mathcal{R}\underline{o}\underline{b}ot$ étaient alignées sur le centre de l'écran, il se pourrait qu'après un changement de grille le $\mathcal{R}\underline{o}\underline{b}ot$ ne soit plus sur un point de la grille. C'est pourquoi la grille est toujours relative au $\mathcal{R}\underline{o}\underline{b}ot$ et non pas à l'écran. En cas de changement de taille de la grille tous les points de la grille (que l'on peut observer grâce à la commande `dg`) changent de place, sauf celui sur lequel est placé le $\mathcal{R}\underline{o}\underline{b}ot$. En conséquence aucune commande du $\mathcal{R}\underline{o}\underline{b}ot$ (sauf une, à savoir `ce`) ne peut lui faire quitter l'un des points de la grille courante. Par exemple les commandes `ag`, `ad`, `eh` et `eb` déplacent le $\mathcal{R}\underline{o}\underline{b}ot$ horizontalement ou verticalement sur le point de la grille courante (respectivement à gauche, à droite, en haut et en bas) le plus près du bord. Pour pouvoir placer le $\mathcal{R}\underline{o}\underline{b}ot$ n'importe où sur l'écran avec précision, il faut disposer d'un point de repère absolu. Ce point de repère absolu est le centre de l'écran. La commande `ce` ramène le $\mathcal{R}\underline{o}\underline{b}ot$ toujours au même point exactement situé au centre de l'écran. Cette commande est donc la seule commande du $\mathcal{R}\underline{o}\underline{b}ot$ qui ne s'effectue pas sur la même grille. Si le centre de l'écran n'est pas situé sur la grille courante, la commande `ce` recentre la grille exactement au centre de l'écran de façon à éviter de faire sortir le $\mathcal{R}\underline{o}\underline{b}ot$ de son quadrillage de déplacement.

Pour éviter d'éventuels problèmes de décalage, il suffit de programmer le dessin comme s'il s'agissait de papier millimétré en choisissant une taille de grille minimale (1 millimètre sur le papier millimétré) de façon à ce que tous les changements de taille de grille soient effectués quand le $\mathcal{R}\underline{o}\underline{b}ot$ est au centre et correspondent à des multiples (5, 10, ...sur le papier millimétré) de la taille de base. On peut également procéder empiriquement par expérimentation, ce qui est quelquefois plus distrayant.

11

Constantes

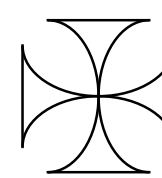
Le nombre 1789 nous fait immédiatement penser à la Révolution française. Quand on lit 3.14159 on sait qu'il s'agit d'une valeur approchée de π (écrit avec la convention anglo-saxonne qui utilise un point à la place de notre virgule). Par contre quand on lit le chiffre 2, on ne pense à rien de précis et il est à peu près sûr qu'on se demande « deux quoi ? ». Tout cela pour dire que lorsqu'on voit un nombre dans un programme il n'est pas toujours facile de comprendre immédiatement de quoi il s'agit. Pour le savoir, on peut, en Pascal nommer le nombre à l'aide d'une déclaration de constante placée au début du programme :

```
const
  RevolutionFrancaise = 1789;  { Date de la révolution }
```

puis utiliser le nom `RevolutionFrancaise` à la place du nombre 1789 dans la suite du programme.

Exemple 17 (Croix de Malte) Dans le programme de dessin d'une croix de Malte ci-dessous, on utilise une constante `Longueur` pour définir la longueur d'une branche (c'est-à-dire la distance de sa base au centre de la croix) et une autre `Largeur` pour définir la largeur d'une branche (c'est-à-dire la distance entre les extrémités de sa base) :

```
program CroixDeMalte_1;
  uses  ;
{ Dessiner une croix de Malte }
  const
    Largeur = 20;  { Largeur d'une branche  }
    Longueur = 30; { Longueur d'une branche }
```



```
procedure BrancheVerticale;
  { Dessiner une branche verticale avec le robot }
  { orienté au nord ou au sud }
begin
  lgX(Largeur); lgY(Longueur); vg; pdt; av; av; pdt; vg; pqtg;
end;

procedure BrancheHorizontale;
  { Dessiner une branche horizontale avec le robot }
  { orienté à l'est ou à l'ouest }
begin
  lgX(Longueur); lgY(Largeur); vg; pdt; av; av; pdt; vg; pqtg;
end;

procedure CroixDeMalte;
  var I : integer;
begin
  for I := 1 to 2 do
    begin
      BrancheVerticale; BrancheHorizontale;
    end;
end;

begin
  CroixDeMalte; st;
end.
```

Pour choisir la meilleure forme possible pour cette croix, il n'y a qu'un seul endroit dans le programme où il est nécessaire de modifier ces constantes. Par exemple pour obtenir la plus petite croix représentée à la figure 11.1 ci-dessous, il suffit de prendre :

```
const
  Largeur = 10;  { Largeur d'une branche  }
  Longueur = 15; { Longueur d'une branche }
```

□

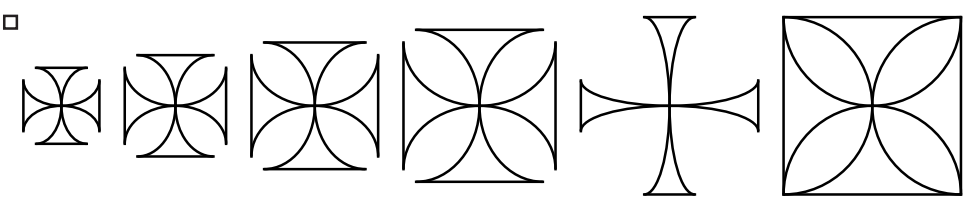


FIG. 11.1 – *Croix de Malte* 10×15, 15×20, 20×25, 25×30, 10×35, 35×35

□

On indique que l'on utilise une ou plusieurs *constantes nommées* par des *déclarations de constantes* :

Déclaration de constantes :

```
const
  Identificateur1 = Nombre1;
  Identificateur2 = Nombre2;
  ...
```

Le mot anglais **const** (abréviation de 'constant') doit obligatoirement être séparé de l'identificateur qui le suit par un blanc ␣ ou un retour à la ligne. Après ces déclarations, on peut utiliser les noms de constantes 'Identificateur1' à la place de 'Nombre1' et 'Identificateur2' à la place de 'Nombre2', etc.

Dans un programme Pascal, une déclaration définit des constantes, des variables ou des procédures :

Déclaration :

Déclaration de constantes

ou

Déclaration de variables

ou

Déclaration de procédures

Les déclarations de constantes, variables ou procédures se placent, dans n'importe quel ordre, avant le corps du programme :

Programme Pascal *:*

```
program ␣ NomDuProgramme;
 uses   ;
 ... Déclarations du programme ...
begin
 ... Instructions du programme[a] (séparées par des ';') ...
end.
```

[a] Utilisant les constantes, variables et procédures déclarées.

La principale règle à respecter est que tout identificateur doit être déclaré avant d'être utilisé.

Exercice 22 *Écrire des programmes* PASCAL *comportant des constantes nommées pour dessiner :*

- *l'hélice singulière de la figure 11.2 ci-dessous ;*
- *le canardeau de la figure 11.3 ci-dessous.*

□

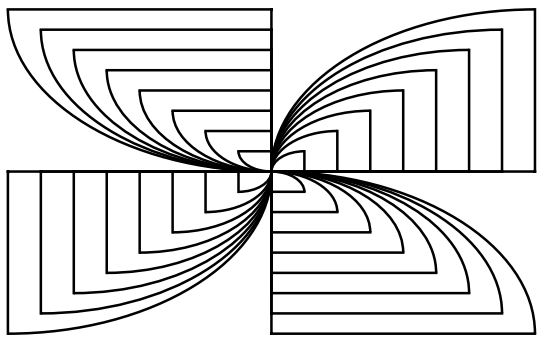


FIG. 11.2 – *Hélice singulière*

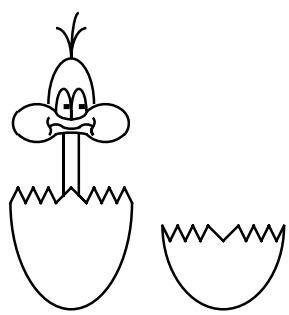


FIG. 11.3 – *Canardeau sortant de son œuf*

Corrigé 22 *Nous donnons un programme, l'autre se trouve sur la disquette d'accompagnement du livre :*

```
program HeliceSinguliere;
  uses   ;
{ Dessiner une hélice singulière }
```

```
const
  Xbase = 13; { Dimension horizontale de la grille de base }
  Ybase = 8; { Dimension verticale de la grille de base }
  NombreDeFeuilles = 8; { Nombre de feuilles dans une pale }
  NombreDePales = 4; { Nombre de pales de l'hélice singulière }

procedure Pale;
  var I : integer;
begin
  for I := 1 to NombreDeFeuilles do
    begin
      { Dessiner un quart d'ellipse, de dimension (Xbase*I, Ybase*I) }
      { sans changer l'orientation initiale du robot                 }
      lgX(Xbase * I); lgY(Ybase * I);
      av; pqtg; av; pqtg; vg; pqtg;
    end;
end;

var J : integer;

begin
  for J := 1 to NombreDePales do
    begin
      Pale; pqtd;
    end;
  st;
end.
```

□

Addenda

Une *déclaration de constante* **const** permet de donner un nom à un nombre particulier. Par la suite le nom, plus significatif, peut être employé à la place du nombre. L'avantage immédiat est que pour changer de nombre il suffit de le faire dans la déclaration de constante alors qu'autrement il faut rechercher toutes ses occurrences dans le programme.

Ceci peut être comparé à la notion de variable en mathématiques c'est-à-dire à l'emploi de lettres x , y , z ...pour désigner des nombres quelconques. La différence est qu'en PASCAL le nom de constante désigne une valeur particulière alors qu'en mathématiques il s'agit d'une valeur quelconque.

Historiquement les premiers problèmes mathématiques traités par les babyloniens (1800 ans av. J.-C.) comportaient une solution algorithmique donnée pour un exemple et des tables numériques séparées pour résoudre d'autres problèmes similaires [30]. Au Moyen Age on utilisait le langage rhétorique comme 'un nombre ajouté à son carré valent 12' pour '$x + x^2 = 12$'. Chuquet [a], auteur du plus ancien traité d'algèbre écrit par un français (1484), utilisait 1^1 pour désigner l'inconnue, 1^2 pour désigner son carré et les symboles $\bar{p}$ et $\bar{m}$ pour l'addition et la soustraction. Il pouvait écrire '$12\ \bar{p}\ 3^1$ egaulx a 4^2' pour '$12 + 3x = 4x^2$'. Viète [b] fit faire un pas important vers la symbolisation en algèbre en introduisant l'usage des lettres pour désigner les quantités inconnues. Il écrivait vers 1590 : '1 QC $-$ 15 QQ $+$ 85 C $+$ 225 Q $+$ 274 N égalent 120', soit si C signifie cube, donc x^3, Q signifie carré, donc x^2, et N signifie nombre, donc x, l'équation $x^5 - 15x^4 + 85x^3 + 225x^2 + 274x = 120$ ([4]). Il fallut attendre quatre mathématiciens français Desargues [c], Descartes [d], Fermat [e] et Pascal [f] (ce dernier ayant été choisi par Wirth pour nommer son langage de programmation) pour entrer dans l'ère moderne des mathématiques.

---

[a] Nicolas Chuquet, mathématicien français, v. 1445-1500.
[b] François Viète, mathématicien français, 1530-1603.
[c] Girard Desargues, mathématicien français, 1591–1661.
[d] René Descartes, mathématicien français, 1596–1650.
[e] Pierre de Fermat, mathématicien français, 1601–1665.
[f] Blaise Pascal, mathématicien français, 1623–1662.

12

Peinture

12.1 Noir et blanc

Le *Robot* peut colorier ses dessins. Nous allons commencer par des coloriages en noir et blanc. La commande `peindre` permet de laisser s'échapper la peinture noire par l'arrière du *Robot*. La peinture s'étale sur la plus grande partie possible de l'écran délimitée par une ligne noire. Un exemple est donné à la figure 12.1 ci-dessous. On commence par dessiner le pourtour de la zone

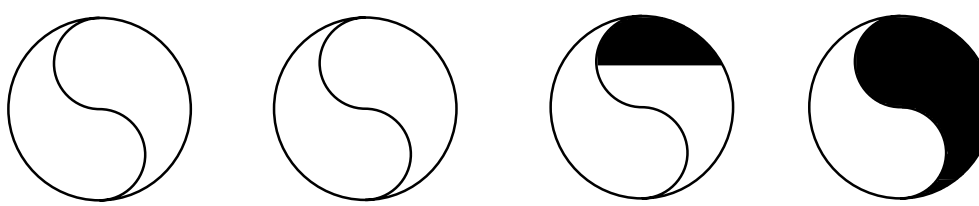
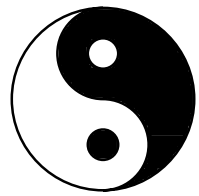


FIG. 12.1 – *Étapes de la peinture d'un dessin*

à peindre puis on place le *Robot* dans la zone à peindre ou sur son bord de façon à ce que l'arrière du *Robot* par où s'échappe la peinture soit dans la zone à peindre. La peinture s'étale ligne par ligne au dessus, à gauche, à droite et en dessous du point arrière du *Robot*. La zone couverte est la plus grande où se trouve l'arrière du *Robot* et qui est délimitée par une ligne noire continue. Le *Robot* peut peindre sans se déplacer, que son crayon soit levé ou baissé.

Exemple 18 (Yin Yang) Le programme ci-après dessine le symbole chinois du Yin Yang :



```
program YinYang;
  uses  ;

  const
    TailleQuadrillage = 3;

  procedure Cercle;
  begin
    vd; vd; vd; vd;
  end;

begin
  { Demi cercle inférieur } lg(6 * TailleQuadrillage); pqtd; vg; vg;
  { Demi cercle supérieur } lg(6 * TailleQuadrillage); vd; vd;
  { Cercle du pourtour } lg(12 * TailleQuadrillage); Cercle;
  { Petit cercle blanc } lg(2 * TailleQuadrillage); lc; pqtd; avf(2);
  pqtg; bc; Cercle;
  { Peindre la zone noire } lc; pqtd; avf(4); peindre;
  { Petit cercle noir } avf(2); pqtg; bc; Cercle; pqtg; peindre;
  st;
end.
```

□

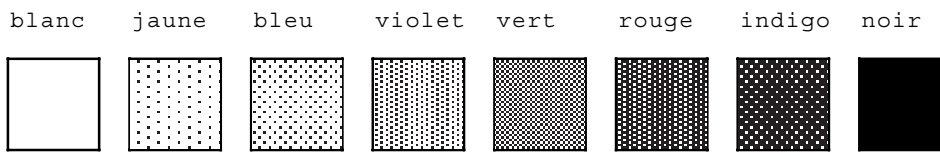
Une erreur fréquente consiste à laisser un trou dans le périmètre de la zone à peindre par lequel la peinture fuira. Un trou d'un seul pixel suffit pour provoquer une fuite. En cas de fuite, on peut arrêter la peinture en tapant sur l'une des touches esc ou § ou #. Le *Robot* continue de dessiner. Taper au moins une seconde fois sur l'une de ces touches pour l'arrêter définitivement.

12.2 Couleur

Pour dessiner et peindre, le *Robot* dispose de crayons de couleurs blanc, jaune, bleu, violet, vert, rouge, indigo ou noir. La commande `cc` permet de choisir la couleur du crayon, que l'on donne en paramètre. Par exemple après la commande `cc(rouge)` tous les tracés et la peinture sont en rouge. La commande `cc(noir)` permet de revenir aux tracés et peinture noirs.

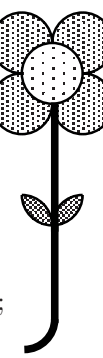
Quand on ne dispose pas d'un écran couleur mais d'un écran à plusieurs niveaux de gris dont chaque pixel peut être plus ou moins foncé, les couleurs sont représentées par des gris, dans l'ordre blanc, jaune, bleu, violet, vert, rouge, indigo et noir du plus clair au plus foncé.

Quand on dispose seulement d'un écran noir et blanc, les couleurs sont représentées par des grisés constitués par des motifs de points blancs et de points noirs juxtaposés. L'ordre est à nouveau blanc, jaune, bleu, violet, vert, rouge, indigo et noir du plus clair au plus foncé. Plus les grisés sont foncés plus les petits points utilisés pour les représenter sur l'écran sont denses comme on le voit ci-dessous :



Quand on peint sur un écran noir et blanc, les bords du motif doivent être tracés en noir (car dans une autre couleur le pourtour serait tracé en pointillés ce qui provoquerait des fuites de peinture).

Exemple 19 (Fleur en couleur) Le programme ci-dessous permet de peindre la fleur ci-contre en couleur :



```
program FleurStyliseeCouleur;
  uses  ;
  { Dessiner et peindre une fleur stylisée en couleur }
  const
    TailleGrille = 12;
    EpaisseurTige = 3;
    EpaisseurBord = 1;
begin
  eb; lg(TailleGrille);
  { Bas de la tige } ec(EpaisseurTige); pqtd; vg; av; av; av;
  { Feuille droite }
    { Dessiner le pourtour } ec(EpaisseurBord); vd; pqtd; vd;
    { Peindre l'intérieur } cc(vert); phtg; peindre; p3htd;
  { Feuille gauche }
    { Dessiner le pourtour } cc(noir); vg; pqtg; vg;
    { Peindre l'intérieur } cc(vert); phtd; peindre; p3htg;
  { Haut de la tige } cc(noir); ec(EpaisseurTige); av; av; av; av;
  { Coeur }
    { Dessiner le pourtour du coeur }
       ec(EpaisseurBord); pqtd; vg; vg; vg; vg;
    { Pétales }
       cc(noir); pqtd; vg; vg; vg; cc(violet); phtd; peindre; phtd;
       cc(noir); pqtd; vg; vg; vg; cc(violet); phtd; peindre; phtd;
```

```
        cc(noir); pqtd; vg; vg; vg; cc(violet); phtd; peindre; phtd;
        cc(noir); pqtd; vg; vg; vg; cc(violet); phtd; peindre; phtd;
    { Peindre l'intérieur du coeur }
        pqtd; cc(jaune); peindre;
  st;
end.
```

□

Exercice 23 *Écrire des programmes* Pascal *pour dessiner et colorier en noir et blanc :*

– *les pavages du carré de la figure 12.2 ci-dessous ;*

– *les mosaïques de la salle des Abencérages (XIV*ème *siècle) de l'Ahlambra, la résidence des rois maures à Grenade en Andalousie telles qu'elles ont été relevées par le peintre Escher en 1936 [24] (figure 12.3 de la page 135).*

□

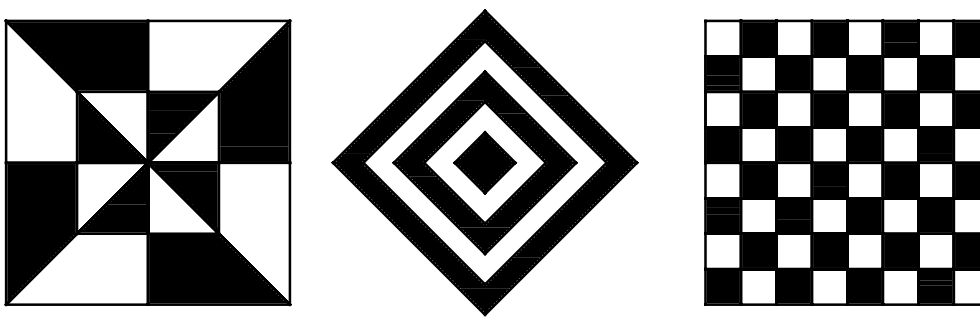


Fig. 12.2 – *Pavages du carré*

Exercice 24 *Écrire des programmes* Pascal *pour dessiner et colorier en couleur :*

– *la balle et le carrelage en trois couleurs de la figure 12.4 (page 135) ;*

– *la mosaïque de l'Ahlambra [24] de la figure 12.5 (page 135) ;*

– *le pavage de carrés disjoints de la figure 12.6 (page 136) ;*

– *les pavages des figures 10.13 (page 118), 10.14 (page 119), 10.15 (page 120) et 10.16 (page 121) utilisant le motif ornemental de N. J. van de Vecht comme il est peint à la figure 10.10 (page 117).*

□

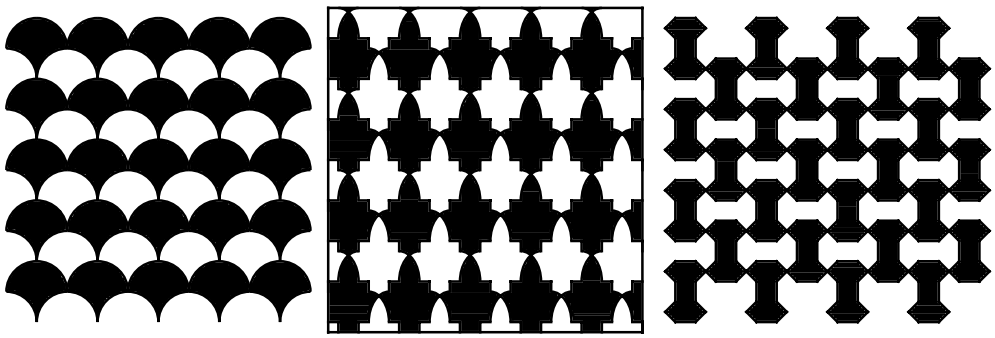


FIG. 12.3 – *Mosaïques de l'Alhambra*

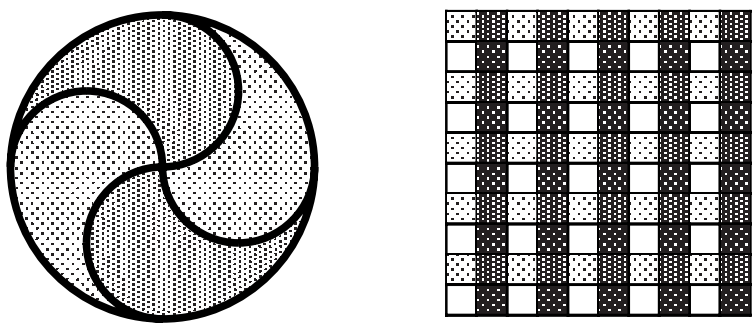


FIG. 12.4 – *Balle, pavage en trois couleurs*

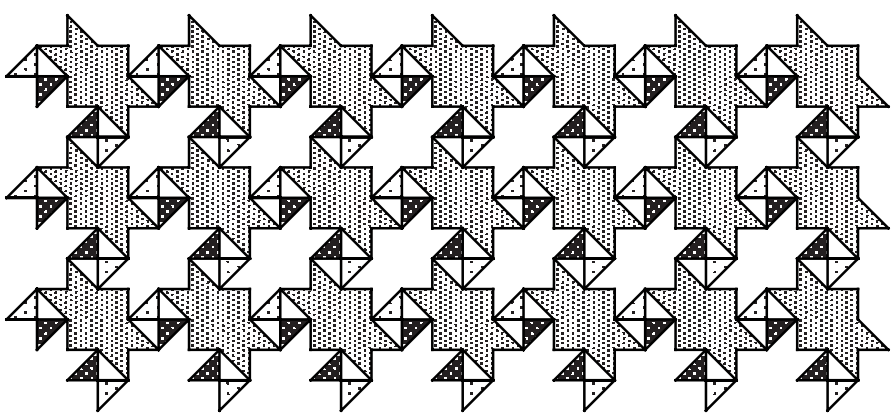


FIG. 12.5 – *Mosaïque de l'Alhambra en couleur*

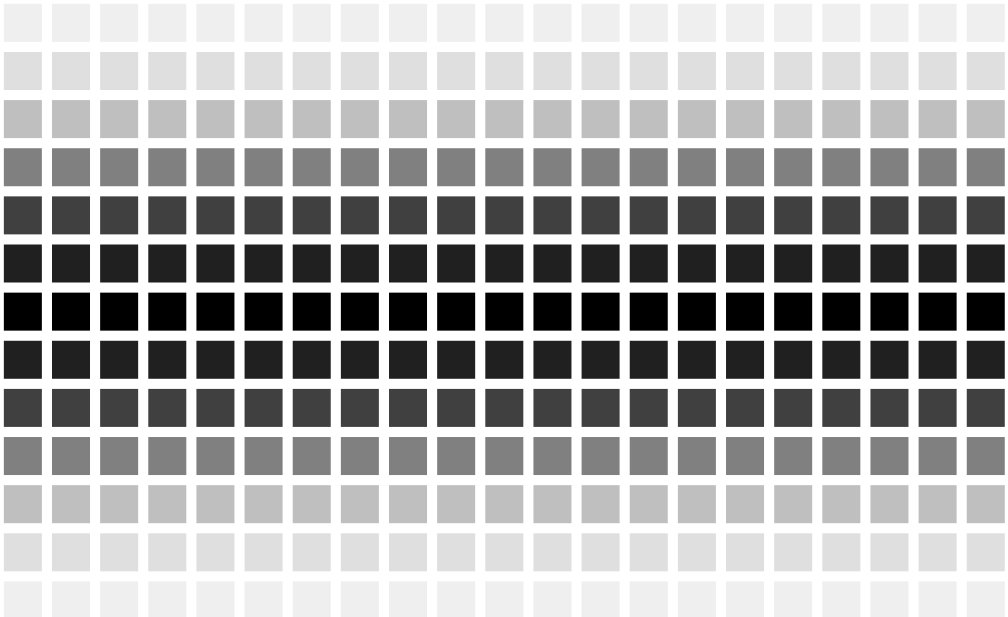


FIG. 12.6 – *Pavage de carrés disjoints en couleur*

Exercice 25 *Quel est le motif utilisé pour réaliser le pavage noir et blanc de la figure 12.7 (page 137) ?* □

Corrigé 23 *Nous donnons un programme, les autres se trouvent sur la disquette d'accompagnement du livre :*

```
program MosaiqueAlhambra3;
 uses  ;

 const
  TailleGrille = 3; { pixels }
  LargeurMotif = 4; { fois la taille de la grille }
  HauteurMotif = 6; { fois la taille de la grille }
  TailleLigne = 7;  { motifs dans une ligne impaire }
  NombreDeLignes = 12; { pair }

 procedure Motif;
  { Dessiner un motif en forme de 8,  en  partant et en arrivant à }
  { l'extrémité  de  la  pointe  inférieure  gauche, avec le robot }
  { orienté vers le nord.                                          }
  var I : integer;
 begin
```

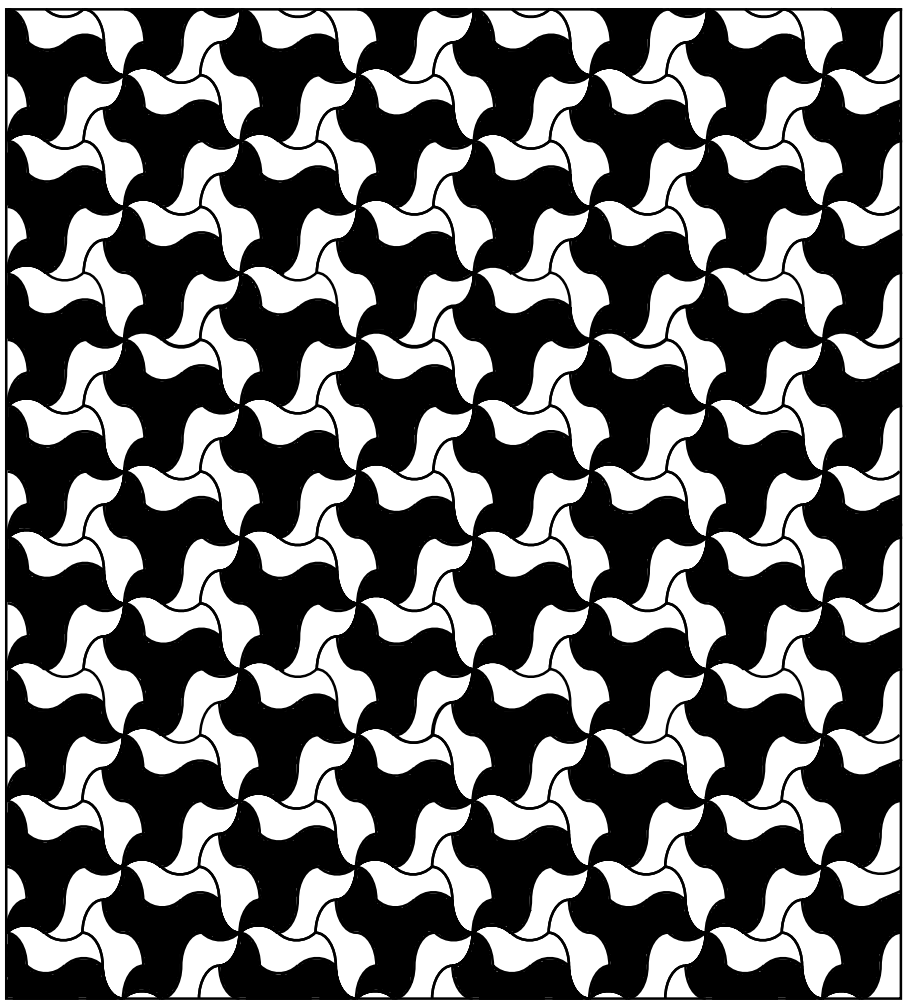


FIG. 12.7 – *Pavage du plan en noir et blanc*

```
  for I := 1 to 2 do
    begin
      { Côté gauche/droit (au dessus/dessous de la pointe inférieure). }
      phtd; av; phtg; avf(HauteurMotif - 4); phtg; av; pqtd; av;
      { Dessus/dessous }
      phtd; avf(LargeurMotif - 2); phtd;
      { Haut/bas du côté droit/gauche (en dessus/dessous de la pointe }
      { supérieure/inférieure).                                       }
      av; phtd;
    end;
  { Colorier et tourner le robot dans l'orientation initiale. }
  pqtg; peindre; pqtd;
end;

procedure MotifSuivant;
  { Passer de la pointe inférieure gauche d'un motif à celle du }
  { motif suivant sur la même ligne horizontale à droite.       }
begin
  lc; pqtd; avf(2 * LargeurMotif); pqtg; bc;
end;

procedure LigneImpaire;
  { Dessiner une ligne horizontale comportant 'TailleLigne' motifs. }
  var I : integer;
begin
  for I := 1 to TailleLigne do
    begin
      Motif; MotifSuivant;
    end;
end;

procedure LignePaire;
  { Dessiner une ligne comportant '(TailleLigne - 1)' motifs. }
  var I : integer;
begin
  for I := 1 to (TailleLigne - 1) do
    begin
      Motif; MotifSuivant;
    end;
end;

procedure RetourDebutDeLigne;
  { Revenir sur la pointe inférieure gauche du premier motif }
  { d'une ligne.                                             }
begin
  lc; pqtg; avf(((2 * LargeurMotif) * TailleLigne) - LargeurMotif);
  pqtd; bc;
```

```
  end;

  procedure LigneSuivante;
    { Passer de la pointe inférieure gauche d'un motif à celle du }
    { motif à la verticale sur la ligne suivante.                 }
  begin
    lc; avf(-(HauteurMotif - 2)); bc;
  end;

  procedure Pavage;
    { Paver avec un nombre de lignes égal à 'NombreDeLignes'. }
    var I : integer;
  begin
    for I := 1 to (NombreDeLignes div 2) do
      begin
        LigneImpaire; RetourDebutDeLigne; LigneSuivante;
        LignePaire;   RetourDebutDeLigne; LigneSuivante;
      end;
  end;

begin
  { Placer le robot à la pointe inférieure gauche du motif gauche }
  { de la première ligne.                                         }
  lc; lg(TailleGrille); ag; eh; pdt; avf(HauteurMotif); pqtg; av;
  pqtg; bc;
  Pavage; st;
end.
```

□

Corrigé 24 *Le motif du pavage du plan en noir et blanc de la figure 12.7 (page 137) est donné à la figure 12.8 (page 140).* □

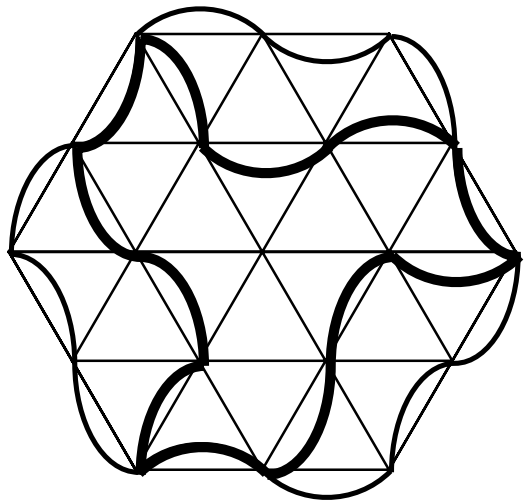


FIG. 12.8 – *Motif du pavage de la figure 12.7*

Corrigé 25 *Nous donnons un programme, les autres se trouvent sur la disquette d'accompagnement du livre :*

```
program BalleEnCouleur;
  uses  ;

  const
    EpaisseurBord = 3;
    DemiRayon = 30;

  procedure Quart_De_Figure;
  begin
    lg(DemiRayon); vg; vg; lg(2 * DemiRayon); vg; ce; pdt;
  end;

  procedure Peindre_Quart_De_Figure;
     { Placer le robot dans la zone à peindre et peindre }
  begin
     lg(DemiRayon div 2); lc; ce; pqtd; av; bc; peindre;
  end;

  var I : integer;

begin
  ec(EpaisseurBord);
```

```
  for I := 1 to 4  do
    begin
      Quart_De_Figure;
    end;
  for I := 1 to 2  do
    begin
      cc(bleu);    Peindre_Quart_De_Figure;
      cc(violet); Peindre_Quart_De_Figure;
    end;
  st;
end.
```

□

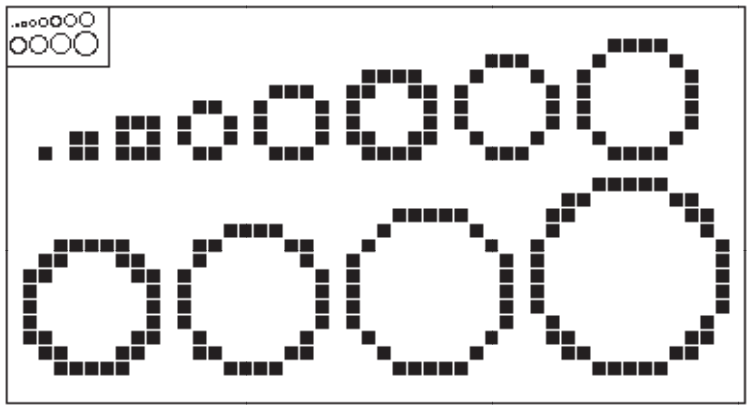
Addenda

On peut imaginer que pour peindre, le $\mathcal{R}^{o}\underline{b}^{o}t$ place un germe sur l'écran, en arrière de son point d'arrêt. Si un germe est placé sur un pixel de l'écran déjà colorié alors ce germe disparaît. En particulier si le germe initial placé par le $\mathcal{R}^{o}\underline{b}^{o}t$ est sur un pixel non blanc, la peinture s'arrête immédiatement. Si le germe est placé sur un pixel blanc de l'écran, alors le germe disparaît, le pixel blanc sur lequel il se trouvait est peint de la couleur du crayon et quatre nouveaux germes sont placés à l'est, à l'ouest, au nord et au sud du pixel qui vient d'être peint. La peinture est terminée quand tous les germes ont disparu.

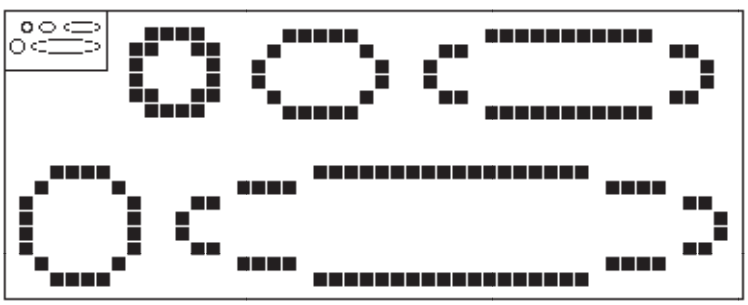
Cet algorithme de peinture est donc basé sur la représentation physique du dessin sur l'écran et non pas sur la représentation logique qui serait celle obtenue en mémorisant toutes les commandes exécutées par le $\mathcal{R}^{o}\underline{b}^{o}t$. De ce fait il se produit parfois des fuites logiquement inexplicables mais qui sont dues à la différence des représentations logiques et physiques.

Prenons l'exemple d'un cercle. Mathématiquement un cercle est une ligne continue constituée par un ensemble infini de points sans dimension. Sur l'écran d'un ordinateur le cercle sera représenté par une ligne discontinue constituée par un ensemble fini de "points" de la dimension d'un pixel. Les notions mathématique et informatique de cercle sont donc complètement différentes.

L'approximation est acceptable quand la taille des pixels est très faible par rapport au diamètre du cercle de la même façon que le compas permet de tracer des cercles qui donnent une bonne image de l'objet mathématique. Cependant pour des cercles très petits, on arrive à quelques aberrations visibles sur la figure ci-après où les cercles représentés dans le petit cadre en haut à gauche de la figure ont été agrandis huit fois. On notera en particulier que les très petits cercles sont carrés !



Sur les écrans ayant des pixels rectangulaires, les cercles ont des formes d'ellipses. Le programme doit donc corriger la déformation en dessinant logiquement des ellipses pour obtenir des cercles sur l'écran. Les cercles sont fermés en ce sens que tout pixel peint sur le cercle a au moins un pixel voisin peint dans l'une des huit directions de la rose des vents. Ce n'est pas le cas pour les ellipses très allongées que l'on peut voir sur la figure ci-dessous :



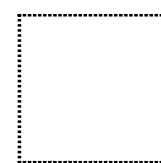
Les trous dans le périmètre des ellipses allongées sont des points de fuite de la peinture quand on veut peindre l'intérieur de l'ellipse. Pour éviter ces fuites il faudrait utiliser des algorithmes de coloriage plus complexes basés sur la structure logique et non pas physique du dessin.

13

Déclarations de procédures avec paramètres

Nous avons appris au chapitre 5 à déclarer et à appeler des procédures sans paramètres nous permettant de répéter plusieurs fois la même séquence de commmandes du *Robot* à des endroits différents dans le programme. Au chapitre 9 nous avons utilisé des commandes du *Robot* qui sont des appels de procédures avec paramètres, l'usage d'un ou plusieurs paramètres permettant à la procédure de faire des séquences de commandes similaires mais différentes selon la valeur des paramètres. Nous allons maintenant voir comment déclarer des procédures avec paramètres en Pascal. Commençons par un exemple simple :

```
program CarrePointille;
  uses   ;
  { Dessiner un carré en pointillé }
  procedure TraitPointille (N : integer);
    { Tracer un trait pointillé de N }
    { points, en supposant N > 1.    }
    var L : integer;
  begin
    lg(2); lc;
    for L := 1 to (N - 1) do
      begin
        dp; av;
      end;
    dp; bc;
```



```
  end;

  procedure Carre (P : integer);
    { Tracer un carré pointillé avec P points par côté (P > 1). }
    var C : integer;
  begin
    for C := 1 to 4 do
      begin
        TraitPointille(P); pqtd;
      end;
  end;

begin
  Carre(30); st;
end.
```

La procédure `TraitPointille` permet de tracer un trait en pointillé constitué par une succession de `N` pixels noirs séparés par (`N`−1) pixels blancs, le nombre `N` de points étant passé en paramètre de cette procédure. La déclaration de la procédure `TraitPointille` indique que le paramètre `N` est un entier (`integer` en anglais). La procédure `Carre` permet de dessiner un carré en pointillé, le nombre `P` de points par côté étant passé en paramètre. L'appel `Carre(30)` indique qu'il faut exécuter la procédure `Carre` avec un paramètre `P` égal à 30. De ce fait la procédure `TraitPointille` est appelée quatre fois avec un paramètre `N` égal à `P` soit 30. On dit que `P` est le *paramètre formel* (dont on ne connaît pas encore la valeur au moment de la déclaration de la procédure et que l'on désigne par un identificateur) et que 30 est le *paramètre effectif* (c'est-à-dire la valeur du paramètre formel pour cet appel de la procédure).

De manière générale, les déclarations et appels de procédures avec paramètres ont la forme suivante :

Déclaration de procédure avec paramètres :

```
procedure ␣ NomDeLaProcedure (ParamFormel1 [a] : integer;
                              ParamFormel2 : integer );
begin
  … Instructions du corps de la procédure [b] (séparées par des ';') …
end;
```

[a] Un paramètre formel est un identificateur.

[b] Les expressions entières figurant dans ces instructions peuvent utiliser les paramètres formels `ParamFormel1`, … de la procédure.

Appel de procédure avec paramètres :

```
NomDeLaProcedure (ParamEffectif1 [a], ParamEffectif2 [b]);
```

[a] Un paramètre effectif est une expression entière.
[b] Les paramètres effectifs sont donnés dans l'ordre où apparaissent les paramètres formels correspondants.

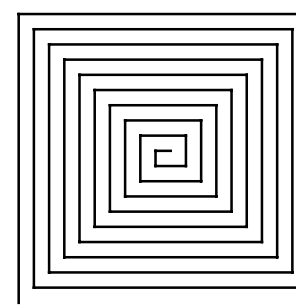
On remarquera que les paramètres formels figurant entre parenthèses dans la déclaration de procédure sont séparés par des points-virgules tandis que les paramètres effectifs figurant entre parenthèses dans l'appel de procédure sont séparés par des virgules.

Une procédure avec paramètres formels peut être appelée plusieurs fois dans un programme avec des paramètres effectifs qui peuvent être différents à chaque fois.

Les paramètres formels d'une procédure ne sont utilisables que dans le corps de la procédure. Par conséquent des procédures différentes peuvent avoir des paramètres formels portant le même nom bien qu'il s'agisse de paramètres différents.

Ces différents points sont illustrés dans l'exemple ci-dessous.

Exemple 20 (Spirale segmentée) Le programme ci-dessous dessine une spirale constituée de paires de segments de droite de longueurs successives 1, 2, 3, ..., 19 :



```
program SpiraleSegmentee;
 uses  ;

 procedure Quart_De_Tour (L : integer);
  { Tourner à gauche puis avancer le }
  { robot tout droit de  L  fois  la }
  { taille  de la grille pour former }
  { un quart de tour.    }
 begin {Quart_De_Tour}
  pqtg; avf(L);
 end; {Quart_De_Tour}

 procedure Demi_Tour (L : integer);
  { Tracer deux quarts de tour succes- }
  { sifs pour former un demi-tour.     }
 begin {Demi_Tour}
    Quart_De_Tour (L);
```

```
    Quart_De_Tour (L);
  end; {Demi_Tour}

  procedure Spirale (N : integer);
  { Tracer une spirale de N demi-tours }
    var L : integer;
  begin {Spirale}
    for L := 1 to N do
      begin
        Demi_Tour (L);
      end;
  end; {Spirale}

begin
  lg(6); Spirale(19); st;
end.
```

□

Exercice 26 *Écrire des programmes* PASCAL *comportant des procédures avec paramètres pour réaliser les dessins suivants :*

– *la croix à trois branches transversales, l'hélice coloriée, le micro-ordinateur, le logo en couleur, la chenille (où le nombre d'anneaux du corps de la chenille est passé en paramètre de la procédure de dessin), la pirogue (où le nombre de rameurs est un paramètre) représentés à la page 147 ;*

– *les carrés inscrits les uns dans les autres ou la version peinte de ce dessin représenté à la figure 13.8 (page 148) ;*

– *l'arabesque inspirée par la coupole du mausolée du Sultan Qayt-Bay représentée à la figure 13.1 (page 147) ;*

– *la péniche de la figure 1.13 (page 10) où le nombre de hublots est donné en paramètre ;*

– *le pavage de Golomb de la figure 8.6 (page 100).*

□

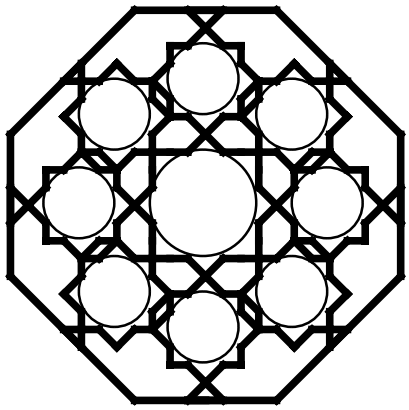


FIG. 13.1 – *Arabesque*

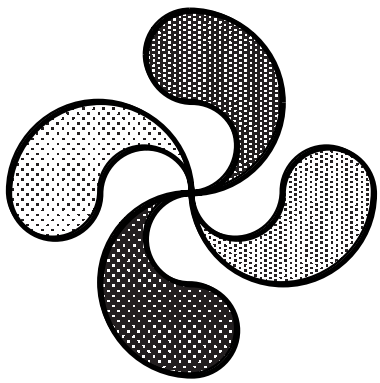


FIG. 13.2 – *Hélice coloriée*

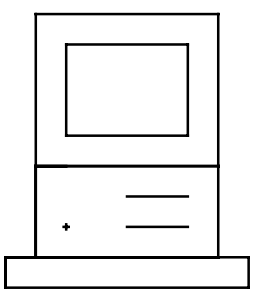


FIG. 13.3 – *Micro-ordinateur*

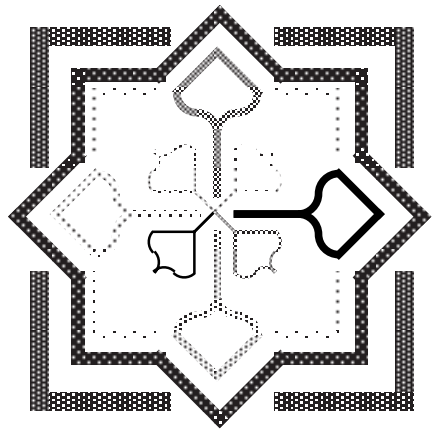


FIG. 13.4 – *Logo en couleur*

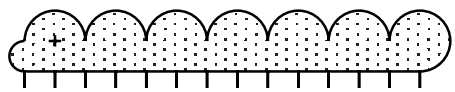


FIG. 13.5 – *Chenille*

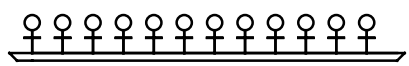


FIG. 13.6 – *Pirogue*

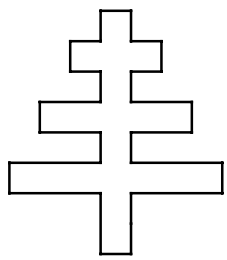
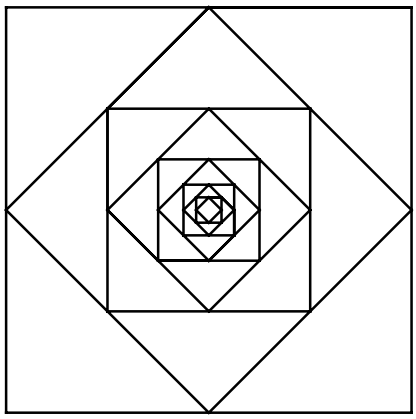


FIG. 13.7 – *Croix à trois branches*



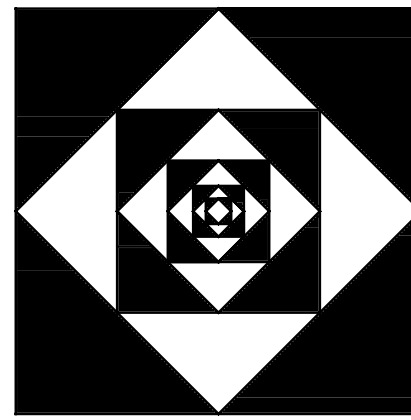


FIG. 13.8 – *Motif de carrés inscrits de Horemis*

Corrigé 26 *Nous donnons un programme, les autres se trouvant sur disquette :*

```
program HeliceEnCouleur;
  uses  ;
  { Dessiner une hélice à quatre pales en couleur }
  const
    TailleGrille = 18;
    EpaisseurDuBord = 2;
  procedure Demi_Cercle_Gauche (R : integer);
  begin
    lg(R); vg; vg;
  end;
  procedure Demi_Cercle_Droit (R : integer);
  begin
    lg(R); vd; vd;
  end;
  procedure Pale(C : integer);
  begin
    { Dessiner le pourtour de la pale }
    ec(EpaisseurDuBord); cc(Noir);
    Demi_Cercle_Gauche (TailleGrille);
    Demi_Cercle_Droit (TailleGrille);
    Demi_Cercle_Droit (2 * TailleGrille);
    { Peindre l'intérieur de la pale }
    pqtd; lg(TailleGrille); lc; avf(3); cc(C); peindre; avf(-3); bc;
  end;
```

```
  procedure Helice;
  begin
    Pale(bleu); Pale(indigo); Pale(violet); Pale(rouge);
  end;
begin
  Helice; st;
end.
```

□

Addenda

On distingue en PASCAL, les définitions des identificateurs de leurs utilisations. Les *occurrences de définition* figurent dans une déclaration de constante, de variable ou de procédure comme nom de procédure ou de paramètre formel. Les *occurrences d'utilisation* de ces identificateurs sont les noms de constante, de variable ou de paramètre formel utilisés dans une expression entière ou les noms de procédure utilisés dans des appels.

Il se peut qu'un même identificateur ait plusieurs occurrences de définition dans un programme. De ce fait il se pose le problème d'établir la correspondance entre les occurrences d'utilisation et de définition d'un même identificateur. Par exemple l'identificateur peut être utilisé pour nommer une constante du programme et des paramètres formels dans des procédures. En PASCAL , ces procédures peuvent être imbriquées les unes dans les autres, comme dans l'exemple ci-après :

```
program Np;
  uses  ;
  const N = 17;
  procedure P1 (N : integer);
    procedure P2 (N : integer);
    begin

      {1}   N

    end;
  begin

    {2}  P2(N);

  end;
begin

  {3}  P1(N);

end.
```

On dit que le programme a une *structure de blocs*. Ces blocs sont formés par les procédures imbriquées les unes dans les autres, le programme étant considéré comme le bloc le plus externe.

La *règle de visibilité* des identificateurs en PASCAL dit que l'occurence d'utilisation d'un identificateur se réfère à l'occurence de définition du bloc le plus interne où figure cette occurrence d'utilisation et qui contient une occurrence de définition de l'identificateur.

Dans l'exemple ci-dessus,

- l'occurence d'utilisation de `N` à la ligne 1 se réfère au paramètre formel de la procédure `P2` ;
- l'occurence d'utilisation de `N` à la ligne 1 se réfère au paramètre formel de la procédure `P1` ;
- l'occurence d'utilisation de `N` à la ligne 3 se réfère à la constante 17.

La structure de bloc de PASCAL est héritée d'ALGOL 60 [27]. Ce langage fut créé par un groupe d'experts européens et américains réunis dans le groupe de travail WG2.1 de l'IFIP (International Federation of Information Processing) qui regroupe toutes les associations nationales d'informaticiens. C'est l'ancêtre d'une famille de langages parmi lesquels on compte PASCAL, ADA et MODULA.

14

Repère cartésien

Pour repérer la position du $\mathcal{R}^{o}\!\!\smile^{o}t$ sur l'écran, nous utilisons un *repère cartésien* tel qu'il est représenté à la figure 14.1 ci-dessous :

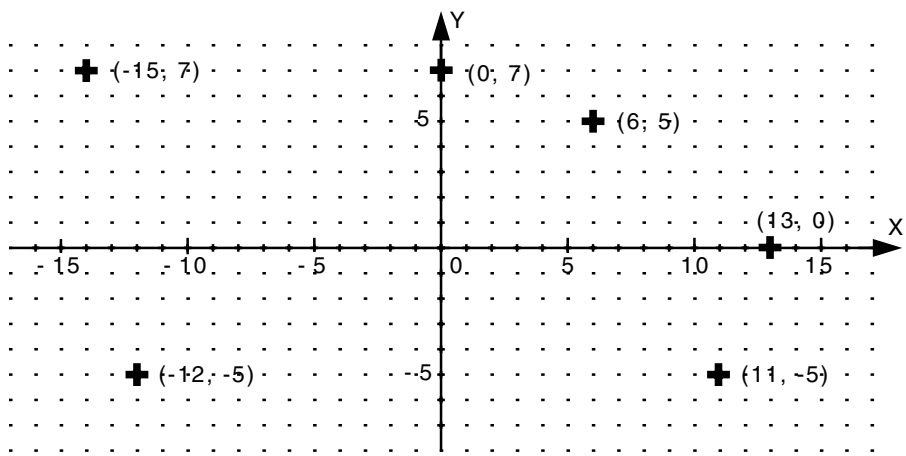


FIG. 14.1 – *Coordonnées dans un repère cartésien*

L'*origine* 0 du repère est située au centre de la grille. L'axe horizontal OX est appelé l'axe des *abscisses*. L'axe vertical OY est appelé l'axe des *ordonnées*. Ces deux axes sont gradués. La longueur des graduations sur l'axe des abscisses est égale à la longueur du côté horizontal d'un carreau de la grille comptée en pixels. De même, la longueur des graduations sur l'axe des ordonnées est égale à la longueur du côté vertical d'un carreau de la grille comptée en pixels.

Un point de la grille est repéré par ses *coordonnées cartésiennes* qui sont deux entiers (X,Y), le premier étant son *abscisse* et le second son *ordonnée*. Par exemple le point de la grille situé au centre de la grille a pour coordonnées (0,0). Les points situés à droite de l'origine O ont une abscisse positive. Les points situés à gauche de l'origine O ont une abscisse négative (précédée du signe moins $-$). De même, les points situés au dessus de l'origine O ont une ordonnée positive tandis que les points situés en dessous de l'origine O ont une ordonnée négative. A titre d'exemple, quelques points de la grille sont repérés sur la figure 14.1 (page 151) par leurs coordonnées.

L'abscisse X d'un point A est la distance à l'origine de sa projection sur l'axe OX (c'est-à-dire du point d'intersection de l'axe OX avec la droite parallèle à l'axe OY passant par A). De même, l'ordonnée Y d'un point A est la distance à l'origine de sa projection sur l'axe OY (c'est-à-dire du point d'intersection de l'axe OY avec la droite parallèle à l'axe OX passant par A). Par exemple sur la figure 14.2, ci-dessous, on a $X = 6$ et $Y = 7$[1].

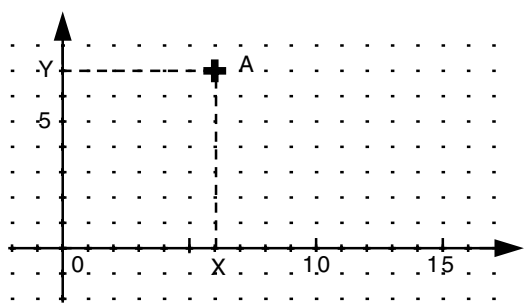


FIG. 14.2 – *Point A d'abscisse X = 6 et d'ordonnée Y = 7*

La commande `deplacer(X, Y)` place le *Robot* sur le point de la grille de coordonnées cartésiennes (X,Y). L'orientation et la position du crayon du *Robot* restent inchangées.

Par exemple, la commande `deplacer(0, 0)` place le *Robot* à l'origine du repère cartésien au centre de la grille[2]. Un exemple est donné ci-après :

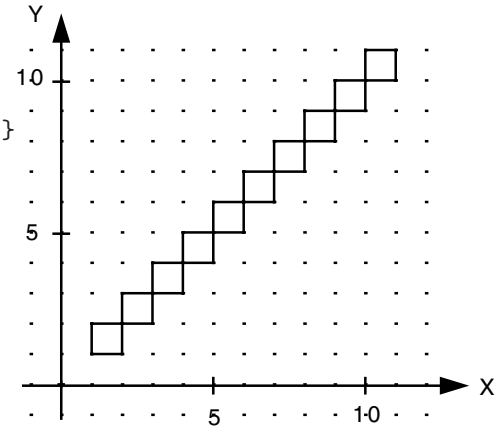
1. On trouvera sur la disquette d'accompagnement du livre un programme PASCAL permettant d'apprendre à repérer des points du plan avec un repère cartésien.
2. La commande `ce` a l'effet supplémentaire de recentrer si nécessaire l'origine du repère exactement au centre de l'écran.

```
program LigneDeCarres1;
  uses  ;

  procedure Carre;
    { Dessin d'un carré de côté 1 }
  begin
    av; pqtd; av; pqtd;
    av; pqtd; av; pqtd;
  end;

  var I : integer;

begin
  for I := 1 to 10 do
    begin
      deplacer(I, I);
      Carre;
    end;
  st;
end.
```



La commande `translater(X, Y)` permet de translater le *Robot* de (X,Y) sans changer son orientation[3]. Un exemple de translation d'un rectangle est donné à la figure 14.3 ci-dessous.

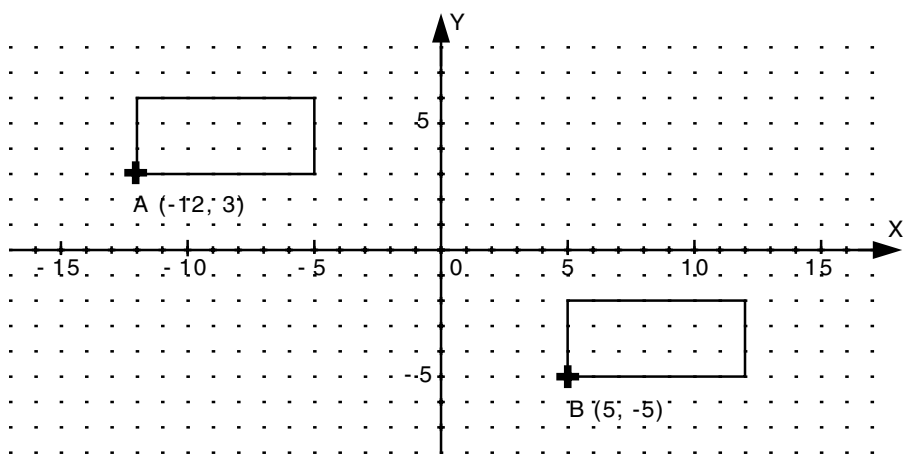


FIG. 14.3 – *Translation (17, -8) de A en B*

3. On trouvera sur la disquette d'accompagnement du livre un programme PASCAL permettant d'apprendre à translater des points du plan repérés par des coordonnées cartésiennes.

Une translation du *Robot* de (X,Y) consiste à augmenter (ou diminuer si X est négatif) son abscisse de X et à augmenter (ou diminuer si Y est négatif) son ordonnée de Y. Si le centre de la grille de coordonnées (0,0) est exactement au centre de l'écran, on peut définir le déplacement à l'aide de la translation comme suit :

```
procedure deplacer(X : integer; Y : integer);
begin
  ce;
  translater(X, Y);
end;
```

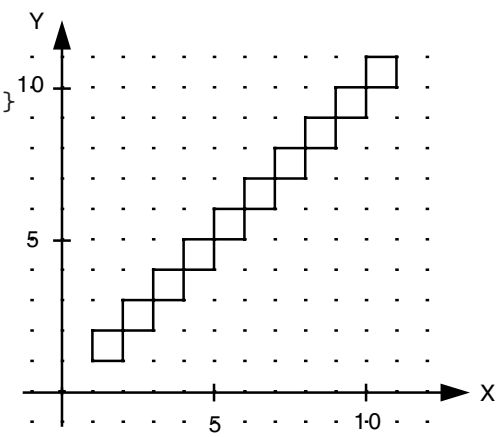
Le programme `LigneDeCarres1` de la page 153 peut être réécrit en utilisant une commande de translation au lieu d'une commande de déplacement comme suit :

```
program LigneDeCarres2;
  uses  ;

  procedure Carre;
    { Dessin d'un carré de côté 1 }
  begin
    av; pqtd; av; pqtd;
    av; pqtd; av; pqtd;
  end;

  var I : integer;

begin
  for I := 1 to 10 do
    begin
      translater(1, 1);
      Carre;
    end;
  st;
end.
```



Avec la commande `translater(X, Y)`, le déplacement du *Robot* est relatif à sa position de départ tandis qu'avec la commande `deplacer(X, Y)`, le déplacement du *Robot* est relatif à l'origine du repère cartésien.

Exercice 27 *Écrire des programmes* Pascal *utilisant les commandes* `deplacer` *et* `translater` *pour réaliser les dessins suivants :*

– *les pavages obtenus par répétition de divers motifs inscrits dans un carré de dimension* (6×6) *de la figure 14.6 (page 157) ;*

– *le pavage de quadrilatères de la figure 14.4 (page 155) ;*

– *la mosaïque de l'Alhambra relevée par Escher [24] de la figure 14.5 (page 156).*

□

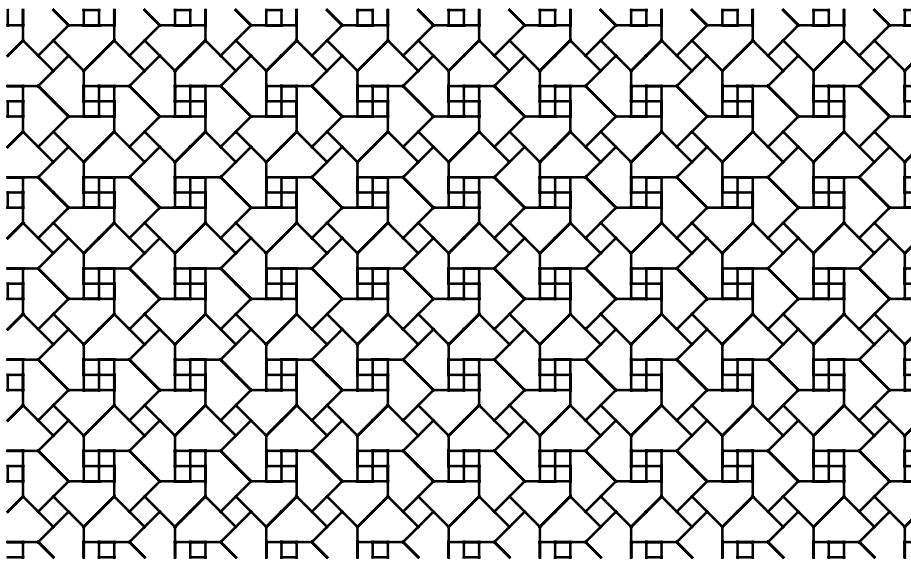


FIG. 14.4 – *Pavage de quadrilatères*

Corrigé 27 *Nous donnons le programme de dessin du pavage de quadrilatères de la figure 14.4 (page 155), les autres programmes se trouvant sur disquette. Il s'agit d'un pavage de l'écran avec des motifs inscrits dans des carrés* (6×6)*, le motif de base étant dessiné à côté de la procédure qui le dessine.*

```
program PavageQuadrilateres;
 uses  ;

 { Pavage de quadrilatères produits par un motif inscrit dans un }
 { carré (6 x 6). }

 const
  TailleGrilleX = 6;
  TailleGrilleY = 6;
  MotifX        = 6; { Largeur du motif }
  MotifY        = 6; { Hauteur du motif }
```

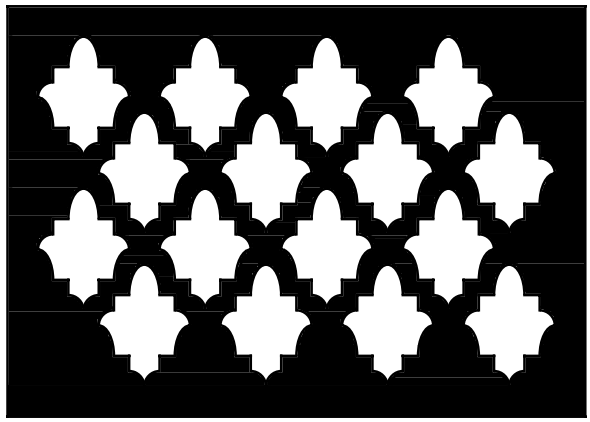
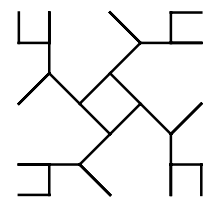


FIG. 14.5 – *Mosaïque de l'Alhambra*



```
procedure Motif;
  { Dessiner un motif dans un  carré }
  { (MotifX x MotifY), le robot par- }
  { tant  et  arrivant  au centre du }
  { motif orienté vers le nord. }
  var I : integer;
begin
  for I := 1 to 4 do
    begin
      lc; pqtg; av; bc;
      p3htd; av; av; pqtg; av; pdt; av;
      phtg; av; av; pdt; av; pqtd; av; pqtd; av;
      lc; p3htd; av; av; av; bc;
      phtd;
  end;
end;

procedure Pavage(X : integer; Y : integer);
  var
    I : integer;
    J : integer;
begin
  { Décentrer  le  dessin  à  gauche  de la moitié de la longueur }
  { d'une ligne.                                                  }
  translater(-(MotifX * (X div 2)), (MotifY * (Y div 2)));
  { Dessiner (Y - 1) lignes de X motifs  (MotifX x MotifY)  (avec }
  { translation en bas à gauche de la ligne suivante).            }
```

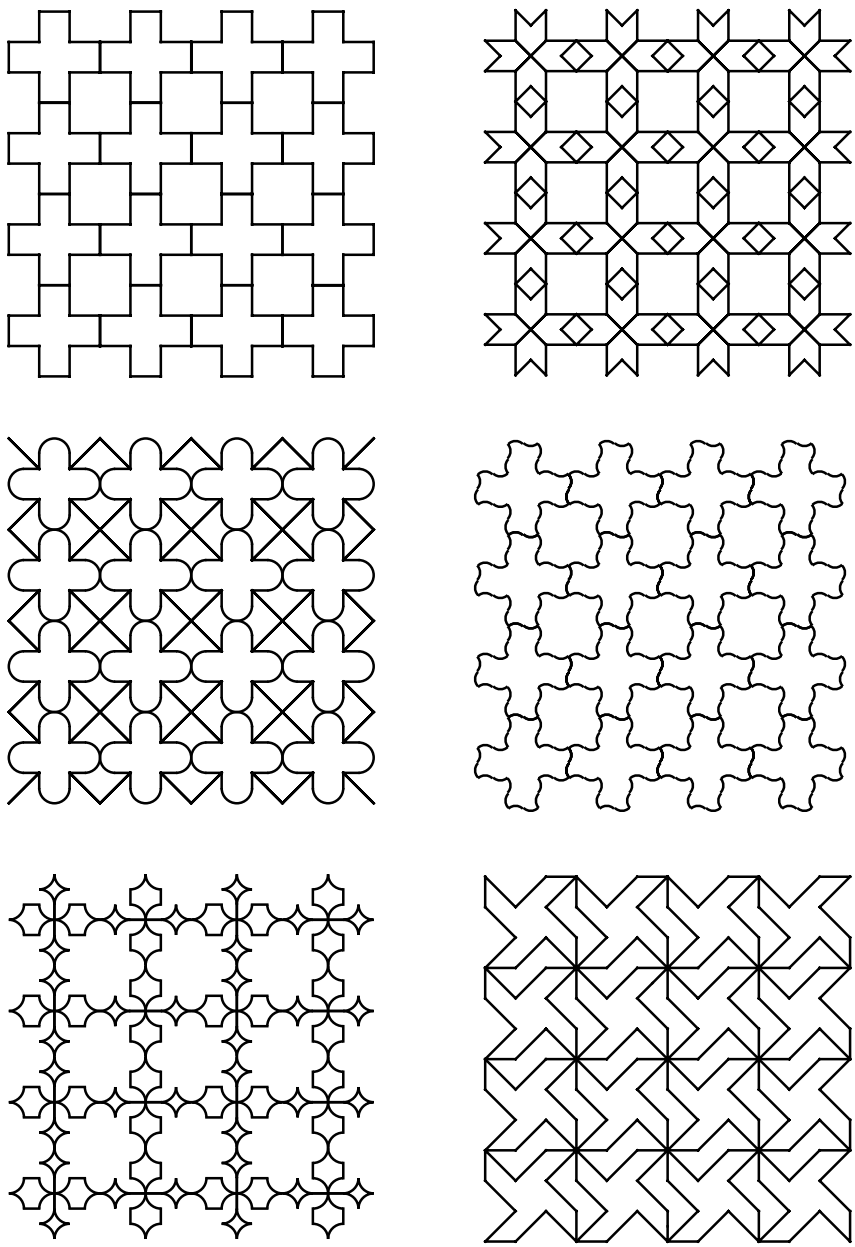


FIG. 14.6 – *Pavages de motifs* (6×6)

```
    for I := 1 to (Y - 1) do
      begin
        { Dessiner une ligne de X motifs (MotifX x MotifY) (sans }
        { translation à droite après le dernier motif de la ligne). }
        for J := 1 to (X - 1) do
          begin
            Motif;
            { Passer au motif suivant à droite dans la même ligne. }
            translater(MotifX, 0);
          end;
        Motif;
        { Passer au début de la ligne suivante. }
        translater(-(MotifX * (X - 1)), -MotifY);
      end;
      { Dessiner la dernière ligne de X motifs (MotifX x MotifY) }
      { (sans translation en bas à gauche de la ligne suivante. }
      for J := 1 to (X - 1) do
        begin
          Motif;
          translater(MotifX, 0);
        end;
      Motif;
  end;

begin
  lgX(TailleGrilleX); lgY(TailleGrilleY);
  Pavage(10, 6);
  st;
end.
```

□

Addenda

Suivant les traces d'Apollonius[a] qui avait donné informellement les équations des coniques (cercles, ellipses, paraboles et hyperboles), Fermat et Descartes ont été les premiers à "mettre en équations" des problèmes géométriques. Descartes a introduit la notion de système de coordonnées planes, chaque point du plan étant défini par son abscisse x et son ordonnée y obtenues en projetant sur des axes OX et OY, rectangulaires ou obliques.

[a] Apollonius de Perge, astronome et mathématicien grec, 295 – 230 av. J.-C.

Une courbe algébrique est alors définie par la relation $f(x,y) = 0$ qui existe entre les coordonnées de chacun de ses points. Par exemple l'équation cartésienne du cercle de centre (a,b) est $x^2 + y^2 - 2ax - 2by + c = 0$ avec le rayon R tel que $R^2 = a^2 + b^2 - c$ en supposant que $a^2 + b^2 \geq c$. Descartes a exposé sa méthode dans un ouvrage publié en 1637 et intitulé "La Géométrie" qui, avec "La Dioptrique" et "Les Météores", constituaient les appendices d'un des textes les plus célèbres de la philosophie, le fameux "Discours de la méthode pour bien conduire sa raison et chercher la vérité dans les sciences" où il expose sa philosophie méthodique et rationelle appelée depuis le *cartésianisme*. L'idée d'équation apparaît plus clairement chez Fermat, qui a découvert indépendamment de Descartes, vers 1629, le principe de base de la *géométrie analytique*. Il ne l'a publié qu'en 1679 dans son "Isagoge (Introduction aux lieux plans et solides)".

Descartes n'avait étudié sérieusement que les coniques du second degré. Newton [a] étudia les cubiques du troisième degré et les courbes de degré supérieur. On lui doit aussi d'avoir employé systématiquement les coordonnées négatives dont Descartes se méfiait. La Hire [b] introduisit en 1679 la considération d'un système de coordonnées spatiales, ce qui revient à construire un troisième axe de coordonnées OZ, perpendiculaire aux axes OX et OY définissant le plan. Un point dans l'espace est alors déterminé par trois nombres réels (x,y,z) et l'équation $f(x,y,z) = 0$ définit une surface gauche. Möbius [c] et Plücker [d] étendirent la notion de coordonnées en introduisant les coordonnées homogènes (X,Y,Z,T) telles que $x = \frac{X}{T}$, $y = \frac{Y}{T}$ et $z = \frac{Z}{T}$. Ceci permet d'introduire les points à l'infini $(X,Y,Z,0)$ et les points imaginaires $X = A_1 + iA_2$, $Y = B_1 + iB_2$, $Z = C_1 + iC_2$ et $T = D_1 + iD_2$ où $i^2 = -1$. On peut alors considérer par exemple des cercles imaginaires tels que $a^2 + b^2 < c$.

---

[a] Sir Isaac Newton, mathématicien, physicien et astronome anglais, 1642–1727.
[b] Philippe de la Hire, astronome et mathématicien français, 1640–1718
[c] August Ferdinand Möbius, astronome et mathématicien allemand, 1790–1868
[d] Julius Plücker, mathématicien et physicien allemand, 1801–1868

15

Test “if”

En réponse à nos questions, notre $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ peut indiquer dans quelle direction il est orienté. N'étant pas doué de la parole, ses réponses sont *booléennes* c'est-à-dire `true` (prononcer “trou”, *vrai* en anglais) ou `false` (prononcer “faulse”, *faux* en anglais). On peut imaginer qu'il donne ses réponses à l'aide d'un voyant lumineux qu'il allume pour répondre `true` (vrai) et qu'il laisse éteint pour répondre `false` (faux).

Pour poser les questions sur l'orientation du $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$, on utilise les commandes de test suivantes :

Commandes de test d'orientation du $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$:	
`tn`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers le nord (n) et `false` sinon.
`tne`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers le nord-est (ne) et `false` sinon.
`te`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers l'est (e) et `false` sinon.
`tse`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers le sud-est (se) et `false` sinon.
`ts`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers le sud (s) et `false` sinon.
`tsw`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers le sud-ouest (sw) et `false` sinon.
`tw`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers l' ouest (w) et `false` sinon.
`tnw`	`true` si le $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ est tourné vers le nord-ouest (nw) et `false` sinon.

TAB. 15.1 – *Tests d'orientation du $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$*

En PASCAL, l'instruction de test “if” permet de n'exécuter une séquence d'instructions que si une certaine *expression booléenne* est vraie (`true`). Pour commencer nous allons utiliser des expressions booléennes simples, à savoir les commandes de test d'orientation du $\mathcal{R}^{o}_{\underline{b}}{}^{o}t$ (`tn`, `tne`, `te`, `tse`, `ts`, `tsw`, `tw`,

`tnw`). Dans sa forme la plus simple, l'instruction de test "if" s'écrit comme suit :

Test "if" à une seule alternative :

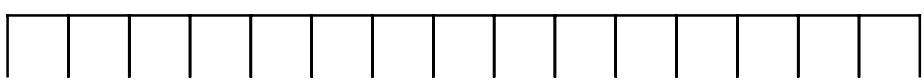
if ␣ expression booléenne[a] ␣ **then**
begin
... Instructions (séparées par des ';') exécutées seulement si l'expression booléenne vaut `true` ...
end;

[a] A résultat `true` (vrai) ou `false` (faux).

On peut traduire en français par `si ... alors dbut ... fin`. Le mot anglais `then` se prononce très approximativement 'zen' (avec un cheveu sur la langue).

Une erreur grave est de mettre un point-virgule ';' après le **then**. Dans ce cas les instructions figurant entre le **begin** et le **end** sont toujours exécutées.

Dans l'exemple ci-dessous, le *Robot* avance deux fois et tourne à droite d'un quart de tour puis d'un quart de tour supplémentaire s'il est orienté à l'ouest :



```
program FriseEnPeigne;
  uses  ;
  const Longueur = 15;
  var I : integer;
begin
  ag;
  for I := 1 to (3 * Longueur) do
    begin
      avf(2); pqtd;
      if tw then
        begin
          pqtd;
        end;
    end;
  st;
end.
```

Dans sa première forme ci-dessus l'instruction de test ne fait rien quand l'expression booléenne vaut `false`. Si l'on veut donner le choix de la séquence d'instructions à exécuter dans les deux alternatives, on utilisera la forme suivante :

Test "if" à deux alternatives :

if ␣ expression booléenne[a] ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne vaut `true` ...
end
else
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne vaut `false` ...
end;

[a] A résultat `true` (vrai) ou `false` (faux).

On peut traduire en français par `si` ... `alors dbut` ... `fin` ... `sinon dbut` ... `fin`.

Une erreur grave est de mettre un point-virgule ';' après le **else**. Dans ce cas les instructions figurant entre le **begin** et le **end** après le **else** sont toujours exécutées. C'est aussi une erreur que de mettre un point-virgule ';' avant le **else** mais elle n'est pas très grave car elle est signalée par le compilateur.

Dans le dessin de la frise en dents de scie ci-dessous, le *Robot* tourne de trois quarts de tour à droite s'il est orienté au nord-est et à gauche dans les autres cas (le seul autre cas possible étant d'ailleurs qu'il soit tourné vers le sud) :

```
program FriseEnDentsDeScie;
  uses   ;
  const NombreDeDents = 31;
  var I : integer;
begin
  ag; phtd;
```

```
  for I := 1 to (2 * NombreDeDents) do
   begin
    av;
    if tne then
     begin
      p3htd;
     end
    else
     begin
      p3htg;
     end;
   end;
  st;
end.
```

Quand il y a plus de deux cas à tester on peut utiliser plusieurs alternatives et ne rien faire quand aucune des expressions booléeennes ne vaut `true` comme dans :

Test "if" à n alternatives :

if ␣ expression booléenne 1 ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne 1 vaut `true` ...
end
else if ␣ expression booléenne 2 ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne 2 vaut `true` ...
end
...
else if ␣ expression booléenne n ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne n vaut `true` ...
end;

On peut aussi prévoir une dernière alternative qui est exécutée quand toutes

les expressions booléeennes testées valent `false` :

Test "if" à $(n+1)$ alternatives :

if ␣ expression booléenne 1 ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne 1 vaut `true` ...
end
else if ␣ expression booléenne 2 ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne 2 vaut `true` ...
end
...
else if ␣ expression booléenne n ␣ **then**
begin
... Instructions (séparées par des ';') exécutées si l'expression booléenne n vaut `true` ...
end
else
begin
... Instructions (séparées par des ';') exécutées si les expressions booléennes 1, ..., n valent `false` ...
end;

Cette forme de test multiple est illustrée ci-dessous dans le programme construisant la frise suivante :

```
program FriseDeVaguelettes;
 uses  ;
 const Longueur = 15;
 var I : integer;
begin
 ag;
 for I := 1 to (2 * Longueur) do
  begin
   if tn then
    begin
     vd; pqtd;
    end
```

```
      else if ts then
        begin
          vg; pqtg;
        end
      else if te then
        begin
          pqtd;
        end
      else
        begin
          pqtg;
        end;
      av;
    end;
  st;
end.
```

Pour terminer nous allons introduire de nouvelles commandes du $\mathcal{R}\text{o}\underline{b}\text{o}t$ qui permettent de faire pivoter le $\mathcal{R}\text{o}\underline{b}\text{o}t$ sur place vers une orientation donnée. Par exemple la commande `pn` qui permet de faire p̲ivoter le $\mathcal{R}\text{o}\underline{b}\text{o}t$ vers le n̲ord peut être définie à l'aide de la procédure Pascal suivante :

```
procedure pn;
begin
  if tne then
    begin
      phtg;
    end
  else if te then
    begin
      pqtg;
    end
  else if tse then
    begin
      p3htg;
    end
  else if ts then
    begin
      pdt;
    end
  else if tsw then
    begin
      p3htd;
    end
```

```
    else if tw then
      begin
        pqtd;
      end
    else if tnw then
      begin
        phtd;
      end;
  end;
```

Nous pourrions définir de la même façon les commandes de pivotement du $\mathcal{R}^{o}_{b}{}^{o}t$ sur place dans l'une des huit directions principales de la rose des vents qui sont données dans la table 15.2 ci-dessous.

Commandes de pivotement du $\mathcal{R}^{o}_{b}{}^{o}t$ sur place :	
`pn`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers le nord (n).
`pne`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers le nord-est (ne).
`pe`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers l'est (e).
`pse`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers le sud-est (se).
`ps`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers le sud (s).
`psw`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers le sud-ouest (sw).
`pw`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers l' ouest (w).
`pnw`	fait pivoter le $\mathcal{R}^{o}_{b}{}^{o}t$ sur place vers le nord-ouest (nw).

TAB. 15.2 – *Pivotement du $\mathcal{R}^{o}_{b}{}^{o}t$ sur place dans une direction donnée*

Exercice 28 *Écrire des programmes* PASCAL *comportant des instructions de test "if" pour réaliser les frises de la figure 15.1 ci-dessous.* □

Corrigé 28 *Nous donnons le programme pour dessiner la frise d'octogones de la figure 15.1 (page 168), les autres se trouvant sur disquette :*

```
program FriseDoctogones;
  uses  ;
  const NombreDoctogones = 15;
  var I : integer;
begin
  ag; pqtd;
  for I := 1 to (8 * NombreDoctogones) do
    begin
```

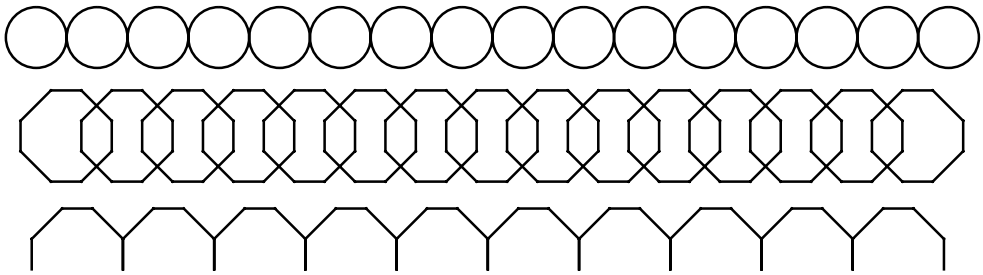


FIG. 15.1 – *Frises de cercles et (semi-) octogones*

```
      if te then
        begin
          lc; avf(2); bc;
        end;
      av; phtd;
    end;
  st;
end.
```

□

Addenda

L'instruction de test `si ...alors ...sinon ...`fut introduite vers la fin des années 1950 dans le langage de programmation ALGOL 60. Ce langage, conçu par un groupe d'experts internationaux, eut une très grande influence dans la conception de langages de programmation ultérieurs notamment ALGOL W, PASCAL, ADA, MODULA et bien d'autres. Une autre forme d'instruction de test par cas fut introduite dans le langage de programmation ALGOL W (également conçu par N. Wirth) et reprise en PASCAL. Cette instruction "case" généralise l'instruction "if" qui peut s'écrire comme suit :

```
case Expression boolenne of
  true  : begin  end;
  false : begin  end;
end;
```

De manière plus générale, l'instruction case permet de choisir quelle séquence d'instructions doit être exécutée en fonction de la valeur d'une expression. Par exemple si `C` est une couleur, on pourra écrire :

```
case C of
  blanc  : begin  end;
  jaune  : begin  end;
  bleu   : begin  end;
  violet : begin  end;
  vert   : begin  end;
  rouge  : begin  end;
  indigo : begin  end;
  noir   : begin  end;
end;
```

Plusieurs cas semblables peuvent être regroupés comme dans l'exemple ci-dessous :

```
case C of
  bleu   : begin  end;
  vert   : begin  end;
  rouge  : begin  end;
  blanc, jaune, violet, indigo, noir : begin  end;
end;
```

Tous les cas possibles doivent être prévus. S'il n'y a rien à faire dans certains cas, il faut le prévoir mais supprimer le **begin end** correspondant. C'est pourquoi il est prévu dans certaines extensions de PASCAL de pouvoir regrouper tous les cas omis dans une même alternative désignée par le mot **otherwise** ('autrement' en anglais) comme dans l'exemple ci-dessous :

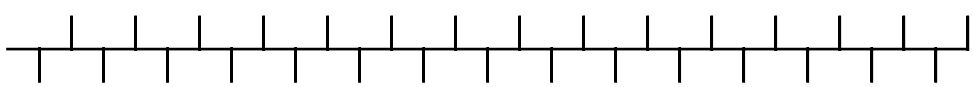
```
case C of
  bleu       : begin  end;
  vert       : begin  end;
  rouge      : begin  end;
  otherwise    begin  end;
end;
```

16

Expressions booléennes

16.1 Parité

Les tests que nous avons faits jusqu'à présent dans les instructions "if" concernaient l'orientation du $\mathcal{R}^{o}_{\smile}{}^{o}t$. En Pascal il est possible de faire des tests concernant les valeurs des variables. Les seules variables que nous utilisons jusque là sont les compteurs de boucles "for". Il peut être utile de faire un test sur le compteur dans le corps d'une boucle pour exécuter des actions différentes à chaque itération. Prenons l'exemple du dessin de la frise en forme de double peigne ci-dessous :



Les dents du peigne étant dessinées de gauche à droite, les $1^{\text{ère}}$, $3^{\text{ème}}$, $5^{\text{ème}}$, ... sont dirigées vers le bas alors que les $2^{\text{ème}}$, $4^{\text{ème}}$, $6^{\text{ème}}$, ... sont dirigées vers le haut. Dans le programme de dessin on utilise un compteur de boucle `I` qui vaut 1 quand on dessine la $1^{\text{ère}}$ dent, 2 quand on dessine la $2^{\text{ème}}$, ...Par conséquent quand la valeur du compteur de boucle est *impaire* (c'est-à-dire 1, 3, 5, ...) il faut dessiner une dent vers le bas tandis que si la valeur du compteur est *paire* (c'est-à-dire 2, 4, 6, ...) il faut dessiner une dent vers le haut. Pour tester que la valeur du compteur `I` est impaire on utilise l'expression booléenne `odd(I)`, le mot anglais 'odd' signifiant précisément 'impair'.

```
program FriseDoublePeigne;
  uses  ;
  const NombreDeDents = 30;
```

```
  var I : integer;
begin
  ag; pe;
  for I := 1 to NombreDeDents do
    begin
      { Trait horizontal } av;
      if odd(I) then
        begin
          { Dent vers le bas } pqtd; av; pdt; av; pqtd;
        end
      else
        begin
          { Dent vers le haut } pqtg; av; pdt; av; pqtg;
        end;
    end;
  st;
end.
```

16.2 Comparaison d'entiers

Si `I` est une variable, par exemple un compteur de boucle, l'expression booléenne (`I > 10`) est `true` (vraie) si la valeur de `I` est strictement plus grande que 10 et `false` (faux) dans le cas contraire c'est-à-dire si la valeur de `I` est inférieure ou égale à 10. Les symboles mathématiques $\leq$ (inférieur ou égal) et $\geq$ (supérieur ou égal) se notent <= et >= en Pascal (car les symboles mathématiques $\leq$ et $\geq$ ne se trouvent pas sur tous les claviers d'ordinateurs alors que <, > et = sont toujours présents). Par exemple les expressions booléennes (10 <= 10) et (10 <= 20) valent `true` (vrai) tandis que (10 <= 9) vaut `false` (faux). En Pascal on note l'égalité avec le symbole mathématique usuel =. Par contre le symbole mathématique $\neq$ (différent) se note <>. Par exemple (10 = 10) et (10 <> 20) valent `true` (vrai) tandis que (10 <> 10) et (10 = 20) valent `false` (faux). On peut utiliser les expressions booléennes de comparaison d'entiers ou de valeurs de variables entières dans le test de l'instruction "if". Par exemple la commande `avf(N)` qui fait avancer le $\mathcal{R}\!\mathit{obot}$ `N` fois si la valeur de `N` est positive, le fait reculer si la valeur de `N` est négative et qui ne change pas sa position quand la valeur de `N` est nulle peut se définir comme suit :

```
procedure Avf (N : integer);
```

```
  var I : integer;
begin
  if (N > 0) then
    begin
      for I := 1 to N do
        begin
          av;
        end;
    end
  else if (N < 0) then
    begin
      pdt;
      for I := 1 to (- N) do
        begin
          av;
        end;
      pdt;
    end;
end;
```

De manière plus générale, on peut utiliser les opérateurs < (strictement inférieur), <= (inférieur ou égal), = (égal), <> (différent), >= (supérieur ou égal) et > (strictement supérieur) pour comparer les valeurs de deux expressions entières comme par exemple (`I <= L`) dans le programme suivant qui trace le serpentin ci-dessous :

```
program Serpentin;
  uses  ;

  procedure DessinSerpentin (L : integer );
    var I : integer;
  begin
    for I := 1 to (2 * L) do
      begin
        if (I <= L) then
          begin
            lg(I);
          end
        else
          begin
            lg(((2 * L) + 1) - I);
          end;
```

```
      if odd(I) then
        begin
          { Demi cercle supérieur } vd; vd;
        end
      else
        begin
          { Demi cercle inférieur } vg; vg;
        end;
    end;
  end;

begin {Robot}
  ag; DessinSerpentin(13); st;
end. {Robot}
```

16.3 Négation, conjonction et disjonction

La *négation* d'une expression booléenne E s'écrit (**not** E), le mot anglais **not** signifiant 'non'. Si la valeur de E est `true` alors celle de (**not** E) est `false` tandis que si la valeur de E est `false` alors celle de (**not** E) est `true`. Par exemple (**not** `odd(5)`) vaut `false` puisque `odd(5)` vaut `true`, `5` étant impair. Ceci peut se résumer par la table de vérité suivante :

E	(**not** E)
`true`	`false`
`false`	`true`

La *conjonction* de deux expressions booléennes E_1 et E_2 s'écrit (E_1 **and** E_2), le mot anglais **and** signifiant 'et'. La valeur de (E_1 **and** E_2) est `true` (vrai) si et seulement si la valeur de E_1 et celle de E_2 sont `true` (vrai). Si l'une des expressions booléennes E_1 ou E_2 vaut `false` (faux) alors la conjonction (E_1**and** E_2) vaut également `false` (faux). Par exemple ((5 >= 0) **and** `odd(5)`) vaut `true` puisque `5` est positif et impair. Ceci peut se résumer par la table de vérité suivante :

E_1	E_2	(E_1**and** E_2)
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

La *disjonction* de deux expressions booléennes E_1 et E_2 s'écrit $(E_1$ **or** $E_2)$, le mot anglais **or** signifiant 'ou'. La valeur de $(E_1$ **or** $E_2)$ est `true` (vrai) si et seulement si la valeur de E_1 ou celle de E_2 est `true` (vrai). Si les valeurs de E_1 et de E_2 sont `false` (faux) alors la disjonction $(E_1$ **or** $E_2)$ vaut également `false` (faux). Par exemple $((4 >= 0)$ **or** `odd(4)`$)$ vaut `true` puisque `4` est positif et bien que `4` soit pair. Ceci peut se résumer par la table de vérité suivante :

E_1	E_2	$(E_1$ **or** $E_2)$
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`

La conjonction et la disjonction peuvent se réaliser simplement avec un circuit électrique comme on le voit à la figure 16.1 ci-dessous avec la convention que `true` représente l'interrupteur fermé et l'ampoule allumée (le courant passe) et `false` représente l'interrupteur ouvert et l'ampoule éteinte (le courant ne passe pas).

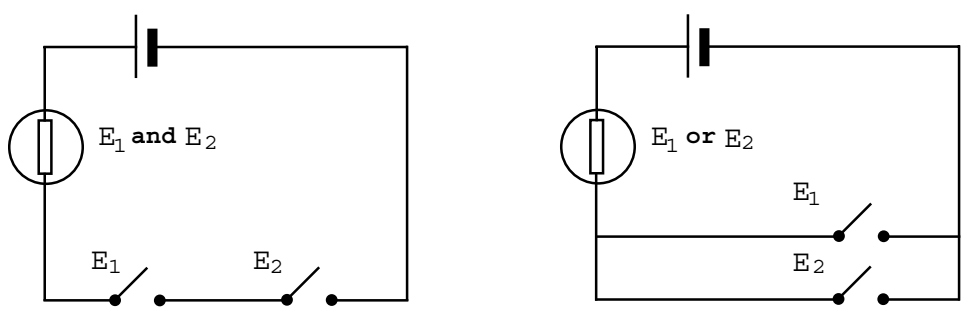


FIG. 16.1 – *Circuits électriques réalisant la conjonction et* la disjonction logiques

Une disjonction est utilisée dans le programme suivant pour tester si le $\mathcal{R}o\underline{b}ot$ est orienté au nord ou à l'est :

```
program FriseEnOr;
  uses   ;
  const Longueur = 31;
```

```
  var I : integer;
begin
  ag;
  for I := 1 to Longueur do
    begin
      if (tn or te) then
        begin
          vd;
        end
      else
        begin
          avf(-1); vg; pqtg;
        end;
    end;
  st;
end.
```

16.4 Expressions booléennes

La table ci-dessous résume les règles d'écriture et les règles de calcul de la valeur des expressions booléennes en PASCAL :

Expressions booléennes :	
`true` `false`	Valeur 'vrai'. Valeur 'faux'.
`odd`(Expr. entière)	`true` si et seulement si la valeur de l'Expr. entière est impaire.
(Expr. entière$_1$ <= Expr. entière$_2$)	`true` si et seulement si la valeur de l'Expr. entière$_1$ est inférieure ou égale à celle de l'Expr. entière$_2$.
(Expr. entière$_1$ < Expr. entière$_2$)	`true` si et seulement si la valeur de l'Expr. entière$_1$ est strictement inférieure à celle de l'Expr. entière$_2$.
(Expr. entière$_1$ = Expr. entière$_2$)	`true` si et seulement si la valeur de l'Expr. entière$_1$ est égale à celle de l'Expr. entière$_2$.
(Expr. entière$_1$ <> Expr. entière$_2$)	`true` si et seulement si la valeur de l'Expr. entière$_1$ est différente de celle de l'Expr. entière$_2$.

(Expr. entière$_1$ > Expr. entière$_2$)	`true` si et seulement si la valeur de l'Expr. entière$_1$ est strictement supérieure à celle de l'Expr. entière$_2$.
(Expr. entière$_1$ >= Expr. entière$_2$)	`true` si et seulement si la valeur de l'Expr. entière$_1$ est supérieure ou égale à celle de l'Expr. entière$_2$.
(`not` Expr. bool.)	`true` si et seulement si la valeur de l'Expr. bool. est égale à `false`.
(Expr. bool.$_1$ `and` Expr. bool.$_2$)	`true` si et seulement si les valeurs de l'Expr. bool.$_1$ et de l'Expr. bool.$_2$ sont égales à `true`.
(Expr. bool.$_1$ `or` Expr. bool.$_2$)	`true` si et seulement si la valeur de l'Expr. bool.$_1$ et ou celle de l'Expr. bool.$_2$ est égale à `true`.

Exercice 29 *Écrire des programmes* Pascal *utilisant des tests de parité pour réaliser les dessins suivants :*

- *la serpentine pyramidale et la grille de la figure 16.2 ci-dessous ;*
- *le pavage octogonal et le pavage d'hexagones entrelacés de la figure 16.3 (page 178).*

□

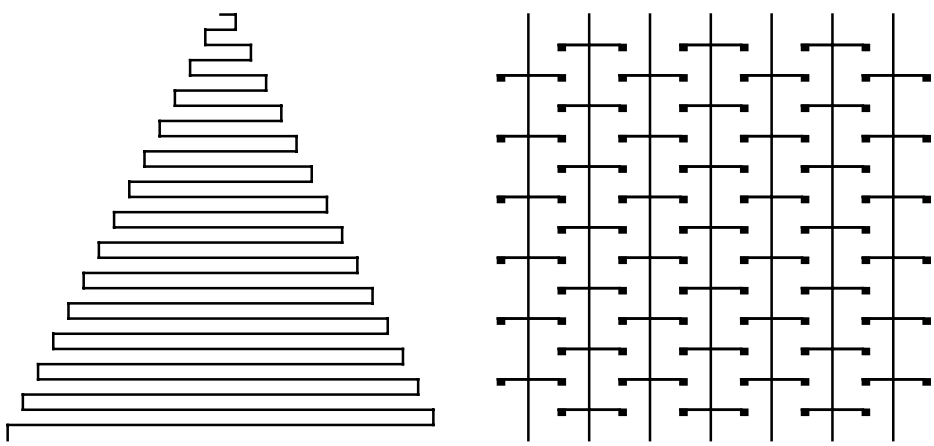


Fig. 16.2 – *Serpentine pyramidale et grille de croix*

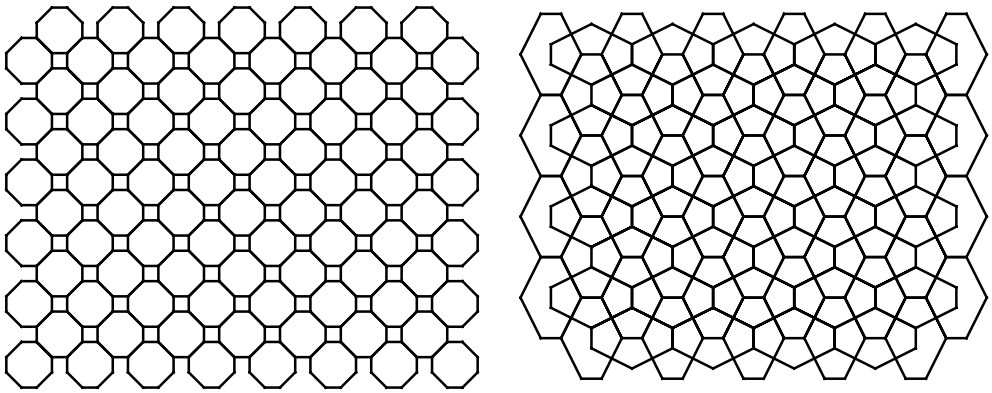


FIG. 16.3 – *Pavages d'octogones juxtaposés et d'hexagones entrelacés*

Exercice 30 *Écrire des programmes* PASCAL *utilisant des expressions booléennes de comparaison de valeurs entières pour réaliser les dessins suivants (le deuxième est difficile) :*

- *les chiffres digitaux de la figure 1.6 (page 8) où le chiffre à dessiner est passé en paramètre de la procédure de dessin ;*
- *le pavage de roues dentées de la figure 16.4 ci-dessous ;*
- *le tapis égyptien de la figure 16.5 (page 179).*

□

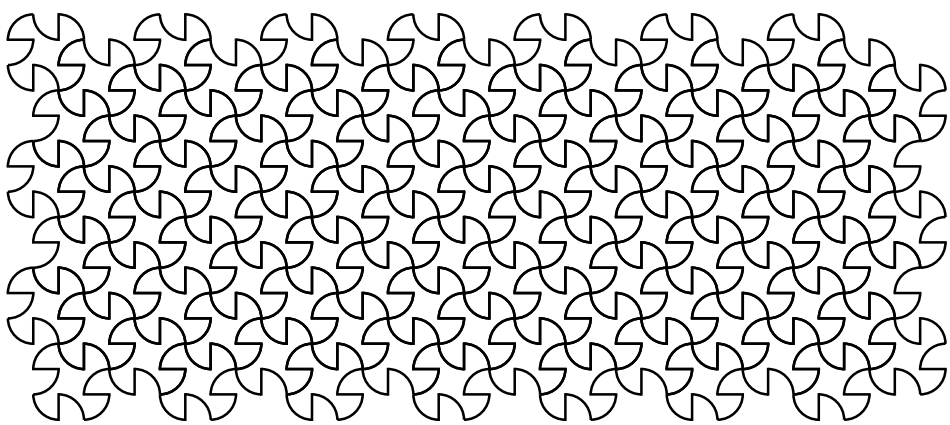


FIG. 16.4 – *Pavage de roues dentées*

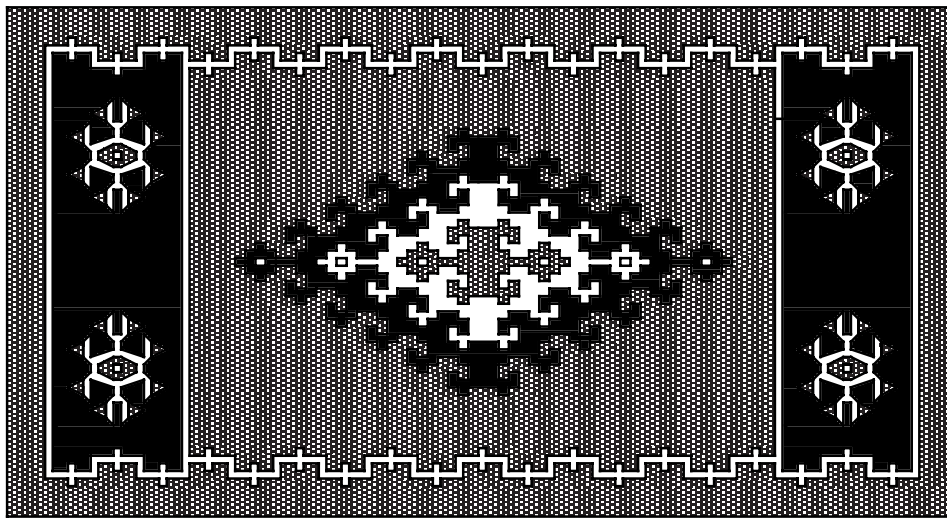


FIG. 16.5 – *Tapis égyptien*

Exercice 31 *Écrire des programmes* PASCAL *utilisant des négations, conjonctions ou disjonctions d'expressions booléennes pour réaliser les dessins suivants (le deuxième est difficile) :*

- *le carré zébré de la figure 16.6 (page 180) ;*
- *la mosaïque de l'Alhambra représentée à la figure 16.7 (page 180).*

□

Corrigé 29 *Nous donnons un programme, les autres se trouvant sur disquette :*

```
program SerpentinePyramidale;
  uses   ;
  const
    TailleGrille = 6;
    Hauteur = 28;
  var I : integer;
begin
  lg(TailleGrille); eh; pqtd;
  for I := 1 to Hauteur do
    begin
      avf(I);
```

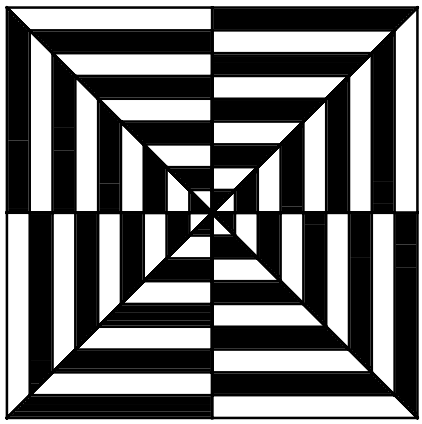


FIG. 16.6 – *Carré zébré*

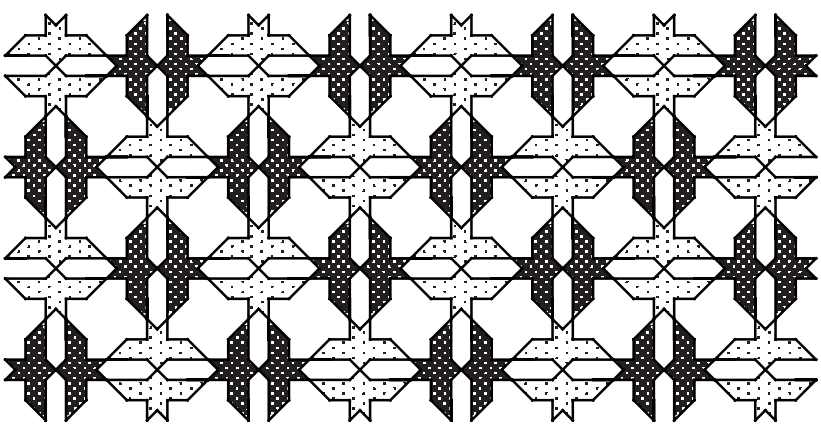


FIG. 16.7 – *Mosaïque de l'Alhambra*

```
      if odd(I) then
        begin
          pqtd; av; pqtd;
        end
      else
        begin
          pqtg; av; pqtg;
        end;
    end;
  st;
end.
```

□

Corrigé 30 *Nous donnons un programme, le deuxième et le troisième se trouvant sur disquette :*

```
program PavageRouesDentees;
  uses  ;

  const
    TailleGrille = 10;
    NombreDeMotifsParLigne = 7;
    NombreDeLignes = 14;

  procedure EnHaut;
    { Avancer le robot tout droit, jusqu'à deux carreaux du bord nord }
    { du cadre.                                                       }
  begin
    eh; pdt; avf(2);
  end;

  procedure RoueDentee;
    { Dessiner le motif du pavage }
    var I : integer;
  begin
    for I := 1 to 4 do
      begin
        av; pqtd; vd; pqtg; vg; pqtg;
      end;
  end;

  procedure Pavage;
    var
      I : integer;
      J : integer;
      L : integer;
  begin
```

```
    for L := 1 to NombreDeLignes do
      begin
        lc; EnHaut; ag;
        avf(L); pqtg; av;
        { Décalage en début de ligne }
        if (L mod 5) = 2 then
          begin
            avf(2);
          end
        else if (L mod 5) = 3 then
          begin
            avf(4);
          end
        else if (L mod 5) = 4 then
          begin
            av;
          end
        else if (L mod 5) = 0 then
          begin
            avf(3);
          end;
        { Dessiner une ligne de motifs }
        pqtg; bc;
        for J := 1 to (NombreDeMotifsParLigne - 1) do
          begin
            RoueDentee;
            translater(5, 0);
          end;
        RoueDentee;
      end;
  end;

begin
  lg(TailleGrille); Pavage; st;
end.
```

□

Corrigé 31 *Nous donnons un programme, le deuxième se trouvant sur disquette :*

```
program MosaiqueAlhambra6;
  uses   ;
```

```
  const
    TailleDeGrille = 8; { doit être pair }
    NombreDeLignes = 4;
    NombreDeColonnes = 8;

  procedure DemiMotif;
  begin
    phtd; av; phtd; av; pqtg; av;
    lg(TailleDeGrille div 2); p3htd; av; pqtg; av;
    lg(TailleDeGrille); p3htd; av; pqtg; av; phtd; av; p3htd; avf(2);
    lg(TailleDeGrille div 2); phtg; av; pqtd; av;
    lg(TailleDeGrille); phtg; avf(2);
    { Peindre }
    avf(-1); phtg;
    if (tsw or tne) then
      begin
        cc(jaune);
      end
    else
      begin
        cc(indigo);
      end;
    peindre;
    cc(noir); phtd; av;
  { Remettre le robot dans l'orientation initiale }
  pqtd;
end;

  procedure Motif;
  begin
    DemiMotif;
    lc; pqtd; avf(5); pqtd; av; bc;
    DemiMotif;
  end;

  procedure MotifDouble;
  begin
    Motif;
    { Motif suivant }
    lc; phtg; avf(2); p3htd; bc;
    Motif;
  end;
```

```
  var
    I : integer;
    J : integer;

begin
  lg(TailleDeGrille); eh; ag; translater(1, −3);
  for J := 1 to NombreDeLignes do
    begin
      for I := 1 to (NombreDeColonnes div 2) do
        begin
          MotifDouble;
          { MotifDouble suivant }
          lc; phtd; avf(2); p3htg; bc;
        end;
      { Ligne suivante }
        if odd(J) then
          begin
            lc; pdt; avf(6); bc;
          end
        else
          begin
            lc; avf(4); pdt; bc;
          end;
    end;
  st;
end.
```

□

Addenda

En PASCAL les priorités des opérateurs entiers et booléens sont définies comme suit :

Priorité des opérateurs entiers et booléens :		
$-$ (unaire), **not**	opérateurs unaires	priorité forte
$*$, **div**, **mod**, **and**	opérateurs multiplicatifs	priorité mi-forte
$+$ $-$ (binaires), **or**	opérateurs additifs	priorité mi-faible
$<$, $<=$, $=$, $<>$, $>=$, $>$	opérateurs de relation	priorité faible

Par exemple l'expression booléenne '`true` **and** `false` **or** `false`' s'évalue comme '((`true` **and** `false`) **or** `false`)' et est donc égale à `false`.

Ce choix des priorités des opérateurs a été fait pour éviter un trop grand nombre de niveaux. Cependant cette convention conduit à certaines écritures peu naturelles. Par exemple '`0 <= I` **and** `I < 100`' s'évalue comme '((`0 <= (I` **and** `I)) < 100`)' qui est incorrecte si `I` est une variable entière puisque la conjonction n'est définie en PASCAL que pour les booléens. Dans certains langages de programmation cette expression '`0 <= I` **and** `I < 100`' s'évalue comme '((`0 <= I`) **and** (`I < 100`))' qui est correcte et plus proche des conventions mathématiques usuelles. Pour éviter les confusions et les erreurs nous employons systématiquement des parenthèses.

Le terme "booléen" ou "booléien" (`boolean` en anglais) vient de Boole[a], créateur de la logique mathématique moderne.

[a] George Boole, mathématicien britannique, 1815–1864.

17

Boucles “while”

17.1 Le robot est-il au bord du cadre ?

Nous avons vu dans le chapitre précédent que les expressions booléennes `tn`, `tne`, `te`, `tse`, `ts`, `tsw`, `tw`, `tnw` peuvent être utilisées dans les instructions de test “if” pour connaître l'orientation du *Robot*. Une autre expression booléenne, `tb`, permet de tester si le *Robot* est au bord du cadre. Plus précisément `tb` vaut `true` (vrai) si l'exécution supposée de la commande `av` fait sortir le *Robot* de son cadre de déplacement. L'expression booléenne `tb` vaut `false` (faux) si l'exécution supposée de la commande `av` ne déplace le *Robot* qu'à l'intérieur de son cadre de déplacement. Par exemple, on peut définir une commande `AvSansSortir` qui ordonne au *Robot* d'exécuter un `av` si cela ne le fait pas sortir du cadre :

```
procedure AvSansSortir;
begin
  if (not tb) then
    begin
      av;
    end;
end;
```

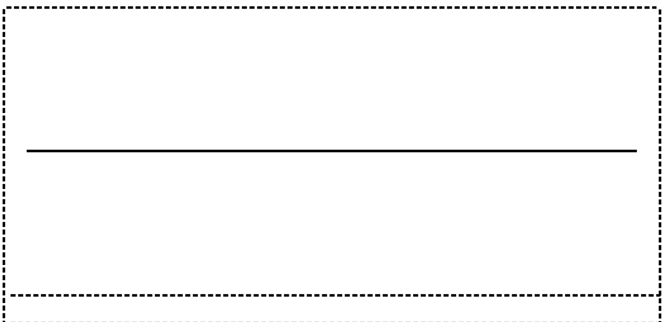
17.2 Boucle “while”

Une boucle “while” permet de répéter l'exécution d'une séquence d'instructions zéro ou plusieurs fois en testant à chaque itération s'il faut s'arrêter

ou continuer. Le mot anglais “while” qui se prononce très approximativement ‘ouaile’ signifie ‘tant que’. La boucle “while” permet de répéter l’exécution d’une séquence d’instructions tant que la valeur d’une expression booléenne est `true` (vrai). Par exemple, le programme suivant permet de tracer un trait horizontal au milieu de l’écran en orientant le *Robot* vers l’est et en répétant des `av` tant que le *Robot* ne sort pas du cadre :

```
program TraitHorizontal;
  uses   ;
begin
  ag; pe;
  while (not tb) do
    begin
      av;
    end;
  st;
end.
```

Le résultat de l’exécution de ce programme est présenté en réduction dans la figure ci-dessous, les cadres de déplacement du *Robot* et d’affichage des messages d’erreurs étant en pointillé :



Une boucle “while” a toujours la forme suivante :

Boucle“while”:

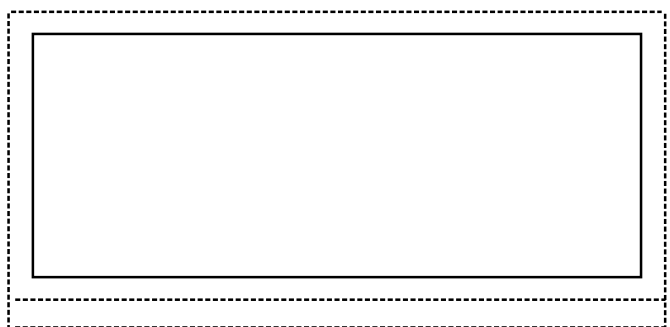
while ␣ Expression booléenne ␣ **do**
begin
… Instructions répétées (séparées par des ‘;’) …
end;

La séquence d'instructions séparées par des ';' qui est répétée tant que la valeur de l'Expression booléenne vaut `true` (vrai) s'appelle le *corps de la boucle*. Si l'Expression booléenne vaut `false` quand on commence l'exécution de la boucle "while" alors cette exécution se termine immédiatement sans que le corps de la boucle ne soit exécuté. Si l'Expression booléenne vaut toujours `true` à chaque itération alors l'exécution de la boucle "while" ne se termine jamais ! On peut tout de même arrêter l'ordinateur en coupant le courant. Des méthodes plus recommandables sont expliquées ci-après.

Une erreur grave est de mettre un point-virgule ';' après le **do**. Si l'Expression booléenne vaut `true`, l'ordinateur boucle éternellement sans rien faire. Si l'Expression booléenne vaut `false`, le corps de la boucle est exécuté une fois.

On utilise de préférence les boucles "for" (dont l'exécution se termine toujours) en réservant l'usage des boucles "while" au cas où le nombre d'itérations n'est pas calculable par une expression entière avant de commencer à exécuter le corps de la boucle.

Exemple 21 (Cadre) Le programme ci-dessous permet de tracer un cadre autour d'un dessin :



```
program TracerCadre;
  uses  ;
  procedure AuBord;
    { Rejoindre le bord en avançant tout droit. }
  begin
    while (not tb) do
      begin
        av;
      end;
  end;
```

```
  procedure Cadre;
  begin
    { Rejoindre le bord supérieur sans rien tracer }
    lc; AuBord;
    { Tracer la moitié droite du bord supérieur }
    pqtd; bc; AuBord;
    { Tracer le bord droit }
    pqtd; AuBord;
    { Tracer le bord inférieur }
    pqtd; AuBord;
    { Tracer le bord gauche }
    pqtd; AuBord;
    { Tracer le bord supérieur }
    pqtd; AuBord;
    { Placer le robot au centre du cadre, orienté au nord }
    ce; pn;
  end;
begin
  Cadre; st;
end.
```

□

Tous les programmes que nous avions écrits jusqu'alors se terminaient. Ceci veut dire que toute exécution du programme prend un temps fini, qui semble parfois long, mais on peut être sûr qu'en attendant suffisamment longtemps, on pourra toujours observer un résultat. Si par exemple on a levé le crayon du *Robot* et oublié de le baisser par la suite, le résultat est une feuille blanche mais l'exécution du programme se termine tout de même.

Avec la boucle "while", on peut écrire des programmes dont l'exécution ne se termine pas. C'est le cas pour le programme suivant :

```
program Boucle1;
  uses  ;
begin
  lc;
  while true do
    begin
      vd;
    end;
  st;
end.
```

Le *Robot* tourne au centre de l'écran, tant que l'ordinateur marche et que l'on n'intervient pas. On peut l'arrêter en tapant deux fois au moins sur l'une

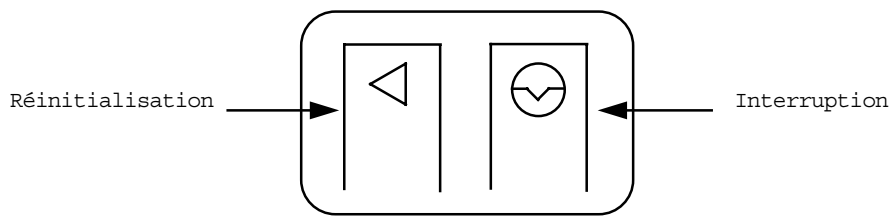
des touches esc, § ou #. L'exécution du programme prend fin car avant d'exécuter une commande, le *Robot* regarde si l'utilisateur n'a pas tapé sur l'une de ces touches pour lui demander de s'arrêter.

Par contre si l'on exécute le programme PASCAL suivant :

```
program Boucle2;
  uses  ;
begin
  while true do
    begin
    end;
end.
```

l'exécution boucle sans s'arrêter même si l'on tape sur les touches d'arrêt esc, § ou # puisque le *Robot* ne reçoit aucune commande pendant que l'ordinateur exécute ce programme. La façon d'arrêter l'exécution de ce programme dépend de l'ordinateur utilisé.

Sur un Macintosh, on peut redonner le contrôle au compilateur (si le programme a été compilé en mémoire) ou au Finder (si le programme a été compilé sur disque) en appuyant sur la touche d'interruption (marquée d'un cercle) placée sur le côté du Macintosh :



Il faut éviter d'appuyer sur la touche de réinitialisation (marquée d'un triangle) qui revient à éteindre puis à rallumer le Macintosh.

Sur un compatible IBM PC, on peut redonner le contrôle au compilateur (si le programme a été compilé en mémoire) ou au système MS-DOS (si le programme a été compilé sur disque) en maintenant la touche contrôle Ctrl enfoncée et en tapant sur la touche Break. On peut également, mais c'est déconseillé, réinitialiser l'ordinateur en maintenant les touches Ctrl et Alt enfoncées puis en tapant sur la touche Del.

Exercice 32 *Écrire des programmes* Pascal *utilisant des boucles "while" pour réaliser les dessins suivants :*

- *la frise de la figure 17.1 ci-dessous ;*
- *la grille des points d'arrêt du Robot de la figure 17.2 (page 193), sans utiliser bien entendu la commande* `dg` *;*
- *le quadrillage de la figure 17.2 (page 193) ;*
- *le grillage de la figure 17.2 (page 193) obtenu en superposant deux quadrillages dont un est tourné d'un huitième de tour.*

Dans tous les cas le dessin doit être le plus grand possible et s'inscrire à l'intérieur du cadre fixé pour les déplacements du Robot sur l'écran de l'ordinateur.
□

Exercice 33 *Définir les commandes* `ag`, `ad`, `eh` *et* `eb` *par des procédures comportant des boucles "while" (on utilisera l'expression booléenne* `valBc` *qui est vraie si et seulement si le crayon du Robot est baissé).* □

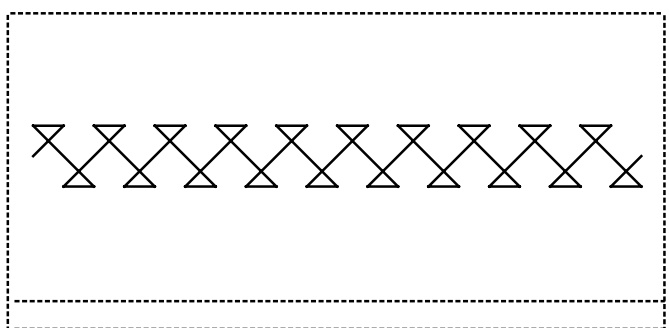


Fig. 17.1 – *Frise de triangles (dans un écran réduit)*

Corrigé 32 *Nous donnons un programme, les autres se trouvant sur disquette :*

```
program FriseDeTriangles;
  uses   ;
```

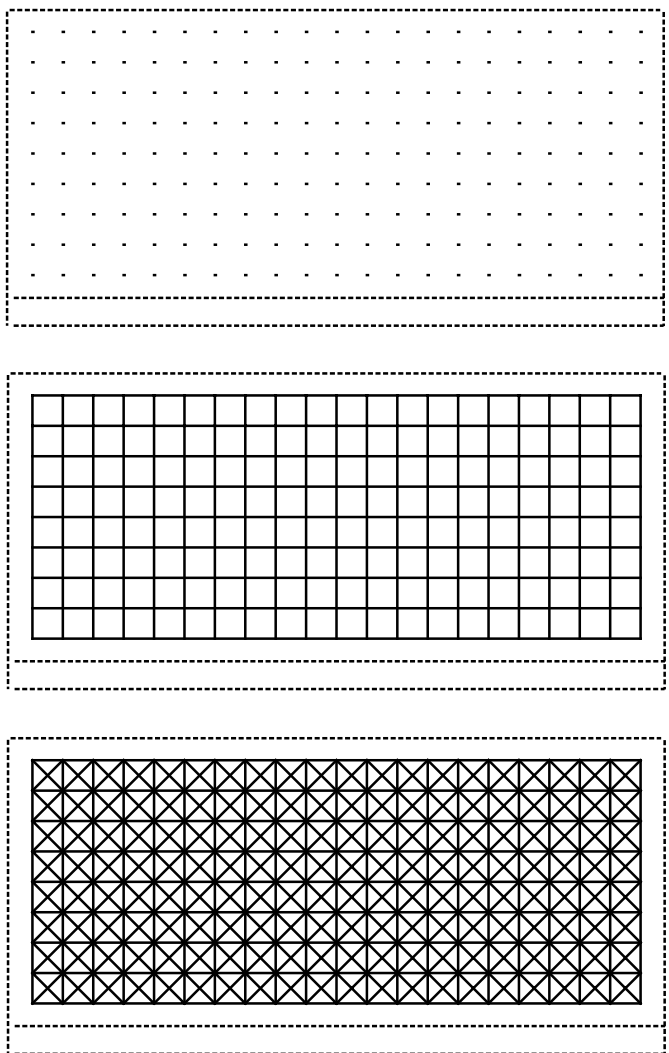


FIG. 17.2 – *Points d'arrêt, quadrillage et grillage sur un écran réduit*

```
  procedure AuBord;
    { Rejoindre le bord en avançant tout droit. }
  begin
    lc;
    while (not tb) do
      begin
        av;
      end;
    bc;
  end;

  procedure Motif;
  begin
    if (not tb) then
      begin
        av; p3htg; av; p3htg; av;
        if (not tb) then
          begin
            av; p3htd; av; p3htd; av;
          end;
      end;
  end;

  procedure Frise;
  begin
    pne;
    while (not tb) do
      begin
        Motif;
      end;
  end;

begin
  pw; AuBord; Frise; st;
end.
```

□

Corrigé 33 *Nous donnons le corrigé pour la commande* `ag`*, les autres se trouvant sur disquette :*

```
{ Définition de la commande ag; }

procedure AuBordOuest;
begin
  lc; pw;
```

```
  while (not tb) do
    begin
      av;
    end;
end;

procedure AlOuest;
begin
  if tn then
    begin AuBordOuest; pn; end
  else if tne then
    begin AuBordOuest; pne; end
  else if te then
    begin AuBordOuest; pe; end
  else if tse then
    begin AuBordOuest; pse; end
  else if ts then
    begin AuBordOuest; ps; end
  else if tsw then
    begin AuBordOuest; psw; end
  else if tw then
    begin AuBordOuest; end
  else
    begin AuBordOuest; pnw; end;
end;

procedure ag;
begin
  if valBC then
    begin AlOuest; bc; end
  else
    begin AlOuest; end;
end;
```

□

Addenda

Une autre forme de répétition non bornée en PASCAL est la boucle "repeat" (qui signifie répéter en anglais). Elle a la forme suivante :

repeat
... Instructions répétées (séparées par des ';') ...
until ␣ Expression booléenne;

Le corps de la boucle est répété (`repeat`) jusqu'à (`until`) ce que la valeur de l'Expression booléenne soit égale à `true`.

On peut exprimer les boucles "while" à l'aide des boucles "repeat" et vice-versa. Une boucle "repeat" :

```
repeat
  Corps de la boucle;
until Expression booléenne;
```

peut s'écrire avec une boucle "while" comme suit :

```
begin
  Corps de la boucle;
  while not Expression booléenne do
    begin
      Corps de la boucle;
    end;
end;
```

De la même façon, une boucle "while" ;

```
while Expression booléenne do
  begin
    Corps de la boucle;
  end;
```

peut se réécrire avec une boucle "repeat" comme suit :

```
if Expression booléenne then
  begin
    repeat
      Corps de la boucle;
    until not Expression booléenne;
  end;
```

L'usage de l'une ou l'autre forme d'itération ne dépend que du fait que le corps de la boucle peut ne jamais être exécuté (boucle "while") ou doit toujours l'être au moins une fois (boucle "repeat").

18

Codage

La mémoire de l'ordinateur ne contient que des zéros et des uns. Il faut donc pouvoir représenter toutes les informations que l'on veut mémoriser par des suites de zéros et de uns. Cette traduction de l'information en binaire s'appelle le *codage binaire*. Les chiffres 0 et 1 utilisés pour représenter l'information s'appellent des *bits*. Nous allons commencer par étudier d'autres formes de codages plus anciennes.

18.1 Code Morse

Le télégraphe électrique, ancêtre de notre téléphone, fut inventé en 1837 par l'américain Samuel Morse [3]. Le télégraphiste émetteur dispose d'un manipulateur qui est un simple levier à ressort permettant d'envoyer des signaux courts (appelés points) et des signaux longs (appelés traits) séparés par des silences ou blancs (correspondant à l'absence de signal). Le signal est transmis le long d'un fil électrique, le retour se faisant par la terre. Chaque mouvement du manipulateur ferme le circuit et provoque un mouvement bref ou long du récepteur qui reproduit les signaux sur une bande de papier. Pour envoyer un message, on commence par l'abréger en *style télégraphique* puis on traduit chaque signe (lettre, chiffre, signe de ponctuation, ...) en une suite de points et de traits séparés par des blancs selon le *code Morse* de la figure 18.1 (page 198). On laisse sept blancs à la fin de chaque mot. Divers autres codes sont utilisés pour indiquer le début et la fin du message, signaler les erreurs, etc. Le télégraphe de Morse permettait de transmettre environ 25 mots à la minute.

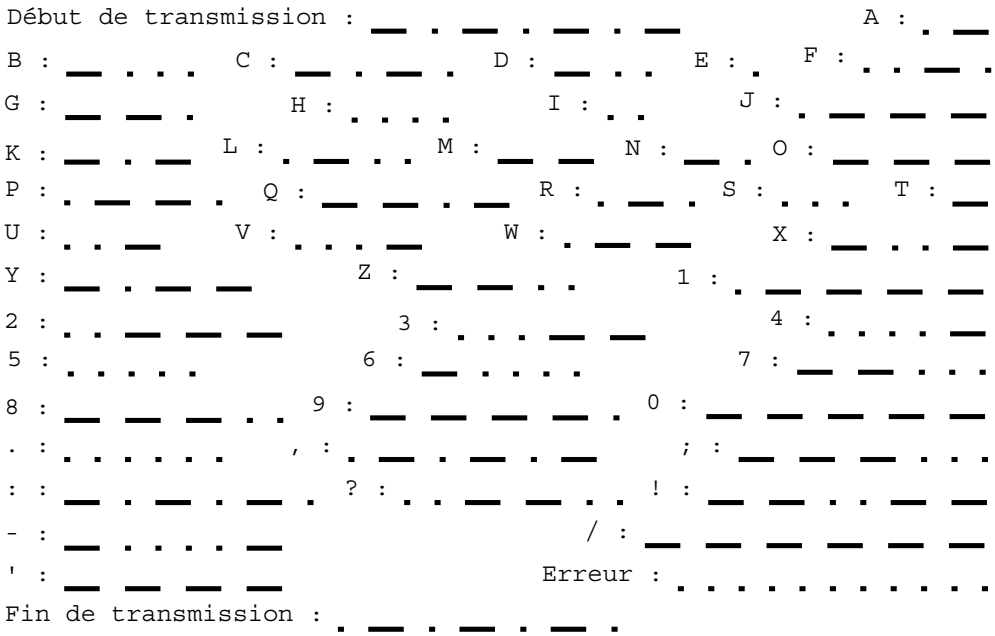
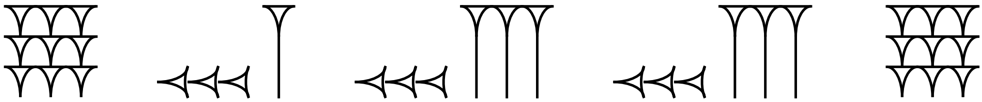
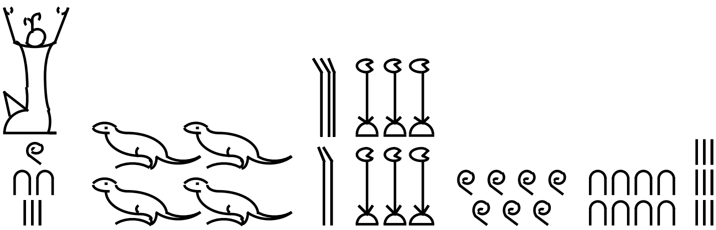


FIG. 18.1 – *Code Morse*

18.2 Numération

Notre codage des entiers naturels (positifs ou nuls) est une numération positionnelle et décimale. C'est une numération décimale car nous utilisons les dix chiffres 0, 1, ..., 9 d'origine indo-arabe. C'est une numération positionnelle car chaque chiffre dans un nombre représente une valeur qui dépend de sa position dans le nombre. Par exemple dans 1789, le 1 représente 1000, le 7 représente 700, le 8 représente 80 et le 9 représente 9. On a donc $1789 = 1000 + 700 + 80 + 9 = (1 \times 10^3) + (7 \times 10^2) + (8 \times 10^1) + (9 \times 10^0)$. Les mathématiciens anciens et modernes ont inventé bien d'autres façons de coder les nombres. La figure 18.2 (page 199) montre différentes façons de coder le nombre 123 456 789, certaines remontant aux civilisations antiques [15]. La supériorité du codage positionnel et décimal des entiers naturels ne tient pas au choix de la base dix mais à la facilité avec laquelle on peut faire les opérations arithmétiques usuelles, par exemple en utilisant un boulier (longtemps en usage et maintenant disparu des écoles communales)





$\lvert\overline{\text{MCCXXXIV}}\rvert\ \overline{\text{LVI}}\ \text{DCCLXXXIX}$

75BCD15

123456789

726746425

221210220202212200

111010110111100110100010101

FIG. 18.2 – *Codages de 123 456 789 en numérations babylonienne, égyptienne, grecque, romaine, hexadécimale, décimale, octale, ternaire et binaire*

18.3 Codage binaire

Toutes les informations rangées dans la mémoire de l'ordinateur ou sur disquette doivent être représentées par des suites de zéros et de uns. Sur les micro-ordinateurs, tels qu'ils sont conçus actuellement, la longueur de ces suites de bits doit être multiple de 8. Selon leur longueur on leur donne des noms spéciaux comme *octet* pour une suite de 8 bits, *demi-mot* pour une suite de 16 bits, *mot* pour 32 bits et *double-mot* pour 64.

Pour montrer que toute information peut être représentée par une suite de bits (à condition de connaître le code) nous allons prendre l'exemple des entiers puis celui des caractères et des textes tapés au clavier.

18.3.1 Entiers naturels

Pour représenter les entiers positifs par une suite de bits, il suffit de les coder en base 2. Par exemple :

$$\begin{aligned} 147 &= 128 + 16 + 2 + 1 \\ &= (1 \times 2^7) + (0 \times 2^6) + (0 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) \\ &\quad +(1 \times 2^1) + (1 \times 2^0) \end{aligned}$$

s'écrit :

$$10010011$$

en base deux. De manière plus générale, le nombre qui s'écrit :

$$a_n a_{n-1} \ldots a_1 a_0$$

en base deux (où $a_n \ldots a_0$ valent 0 ou 1) vaut :

$$(a_n \times 2^n) + (a_{n-1} \times 2^{n-1}) + \ldots + (a_1 \times 2^1) + (a_0 \times 2^0)$$

(où 2^n est égal à 2 multiplié n fois par lui-même si $n > 0$ et $2^0 = 1$). Avec cette convention on peut donc coder sur un octet les entiers naturels compris entre 0 (00000000) et 255 (11111111).

18.3.2 Caractères (code ASCII)

Les programmes PASCAL que nous écrivons sont des suites de caractères. A chaque fois que nous tapons sur une touche du clavier le code correspondant

du caractère est entré dans la mémoire de l'ordinateur. Les caractères correspondant aux chiffres, aux lettres minuscules ou majuscules et aux caractères usuels comme $+$, $*$, $($, $)$, ...s'appellent les caractères "imprimables". Certains autres caractères qui ne peuvent pas s'écrire s'appellent les caractères "spéciaux." C'est le cas par exemple des "caractère de contrôle" et "caractère d'échappement" que l'on obtient respectivement en tapant les touches Ctrl et Esc. Ces caractères (imprimables ou spéciaux) sont souvent représentés sur un octet (8 bits) selon un code international qui s'appelle le code ASCII. Les caractères imprimables dont le code ASCII est compris entre 32 et 126 sont donnés dans la table 18.1 ci-dessous. Par exemple, le code ASCII 65 du caractère `'A'` s'obtient dans cette table en ajoutant le numéro 64 de la ligne où se trouve le caractère `'A'` au numéro 1 de la colonne de ce caractère.

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	¡	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

TAB. 18.1 – *Caractères de codes* ASCII *compris entre 32 et 126*

Quand nous disons que le caractère `'A'` a pour code ASCII 65, il faut comprendre que ce caractère est représenté dans la mémoire de l'ordinateur par l'octet 01000001 qui représente également le nombre 65 écrit en base deux. On remarquera que le caractère zéro a pour code 48. L'entier 0 (codé en machine sur un octet par 00000000) a donc un code différent du caractère zéro `'0'` (codé en machine par 00110000). La table 18.2 (page 201) montre les codes ASCII de quelques caractères spéciaux.

Code ASCII	8	13	27	127
Touche du clavier	←	↩	Esc	Del

TAB. 18.2 – *Code* ASCII *de quelques caractères spéciaux*

18.3.3 Texte

Un texte (comme par exemple un programme PASCAL) est une suite de caractères. Le code d'un texte en machine est la suite des codes des caractères qui le composent. Par exemple le texte `'Pascal'` sera représenté en machine par la suite des codes ASCII des caractères `'P'`, `'a'`, `'s'`, `'c'`, `'a'`, `'l'` soit, d'après la table 18.1 (page 201), la suite 80 (01010000), 97 (00000110), 115 (01110011), 99 (01100011), 97 (00000110), 108 (01101100). Le code du texte `'Pascal'` est donc 01010000 00000110 01110011 01100011 00000110 01101100.

18.3.4 Chaînes de caractères

Un programme permet de faire manipuler par l'ordinateur des booléens, des entiers mais également des textes, tous codés en binaire. Pour ne pas confondre le texte du programme et les textes manipulés par le programme, nous appelerons ces derniers des "chaînes de caractères".

Usage et notation en PASCAL :

En PASCAL on peut manipuler des textes limités à 255 caractères que l'on appelle des "chaînes de caractères" (`string` (prononcer stri-ngue) en anglais). Un paramètre formel de procédure `X` dont la valeur est une chaîne de caractères doit être déclaré `X : string`. Le paramètre effectif correspondant est une chaîne de caractères, comme par exemple :

```
'Ceci est une chaine de caracteres'
```

qui s'écrit entre apostrophes. S'il est facile de distinguer un identificateur ou un entier dans le texte d'un programme, il est plus difficile de reconnaître un texte dans ce texte ! L'usage des apostrophes permet donc de délimiter la partie du texte constituant la chaîne de caractères de celle du reste du programme. Mais ceci pose problème si la chaîne de caractères contient une apostrophe. C'est pourquoi on convient de doubler les apostrophes dans une chaîne de caractères écrite entre apostrophes, comme dans :

```
'Le langage ''pascal'''.
```

Une chaîne de caractères réduite à une seule apostrophe s'écrit donc `''''`.

Concaténation de chaînes de caractères :

Si c_1, c_2, ..., c_n sont des chaînes de caractères alors `concat(`c_1`, `c_2`, , `c_n`)` est la chaîne de caractères obtenue en concaténant (c'est-à-dire en mettant bout à bout) les chaînes de caractères c_1, c_2, ..., c_n. Par exemple :

```
concat('Le ', 'langage ', '''', 'pascal', '''')
```

est égal à la chaîne de caractères qui s'écrit également :

```
'Le langage ''pascal'''
```

Écriture sur le dessin et sous le dessin du robot :

Que son crayon soit levé ou baissé, le *Robot* peut écrire des chaînes de caractères, sans se déplacer et sans changer son orientation. Il peut écrire dans la fenêtre de dessin ou dans le cadre d'affichage des messages situé sous la fenêtre de dessin.

La commande `Ecrire`(c) indique au *Robot* qu'il faut écrire la chaîne de caractères c à droite du point où il se trouve dans sa fenêtre de dessin. La commande `Message`(c) indique au *Robot* qu'il faut écrire la chaîne de caractères c dans le cadre prévu en dessous de la fenêtre de déplacement. Ceci a pour effet d'effacer le message précédent affiché dans ce cadre. Pour laisser à l'utilisateur le temps de lire le message, on peut, si nécessaire, utiliser la procédure `MarquerUnePause` qui attend pour continuer que l'utilisateur tape sur une touche ou clique sur le bouton de la souris. La commande `EffacerMessage` efface ce message. Le choix d'une police, d'un style ou d'une taille de caractères autre que le choix standard est expliqué sur les disquettes d'accompagnement du livre.

Si n est un entier alors `EntierEnChaine`(n) est la chaîne de caractères qui dénote cet entier dans la notation décimale usuelle. Si b est un booléen alors `BooleenEnChaine`(b) est `'Vrai'` si b est égal à `true` et `'Faux'` si b est égal à `false`. Par exemple si `X` est un paramètre entier de procédure égal à 17 alors la commande :

```
ecrire(concat('La valeur de X est ', EntierEnChaine(X), '.'))
```

écrit la concaténation des chaînes de caractères `'La valeur de X est '`, `'17'` et `'.'`, c'est-à-dire :

```
La valeur de X est 17.
```

à la droite du *Robot*. Si `B` est un paramètre booléen de procédure égal à `true` alors la commande :

```
message(concat('B = ', BooleenEnChaine(B)));
```

écrit la concaténation des chaînes de caractères `'B = '` et `'Vrai'`, c'est-à-dire :

```
B = Vrai
```

dans le cadre des messages.

18.4 Musique

Ce paragraphe est destiné aux musiciens qui connaissent le solfège, c'est-à-dire la manière de coder la musique. Le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ peut jouer de la musique, de manière il est vrai assez limitée, puisqu'il ne peut jouer du piano qu'à un seul doigt. Pour coder la musique pour le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$ nous disposons des procédures avec paramètres entiers. Il nous suffit donc d'établir la correspondance avec la théorie classique de la musique [12] pour définir comment écrire de la musique pour le $\mathcal{R}\text{o}\underset{\frown}{\text{b}}\text{o}t$.

18.4.1 Figures de notes

La procédure `Note(h, d)` permet de faire entendre sur le haut-parleur du micro-ordinateur une note de hauteur `h` et de figure `d`. La hauteur de la note est codée par un entier `h` comme indiqué à la figure 18.3 (page 205). La figure de la note indique sa durée qui est codée par une constante `d` qui peut être `ronde`, `ronde_pointee`, `blanche`, `blanche_pointee`, `blanche_en_triolet`, `noire`, `noire_pointee`, `noire_en_triolet`, `croche`, `croche_pointee`, `croche_en_triolet`, `double_croche`, `double_croche_pointee`, `double_croche_en_triolet`, `triple_croche`, `triple_croche_pointee`, `triple_croche_en_triolet`, `quadruple_croche` ou `quadruple_croche_en_triolet`. La durée relative d'une `ronde_pointee` est deux fois et demi celle d'une `ronde`. La durée relative d'une `ronde` est deux fois celle d'une `blanche`, trois fois celle d'une `blanche_en_triolet`, ...

18.4.2 Mouvement

Le mouvement, c'est-à-dire le degré de vitesse ou de lenteur avec lequel doit être exécuté le morceau de musique détermine la durée absolue des différentes figures de notes. Ce mouvement est défini par la procédure `metronome`(n) qui indique le nombre n d'oscillations du métronome par minute, sachant qu'une noire est exécutée à chaque battement. Par exemple `metronome(60)` indique qu'il faut jouer 60 noires à la minute. Les mouvements

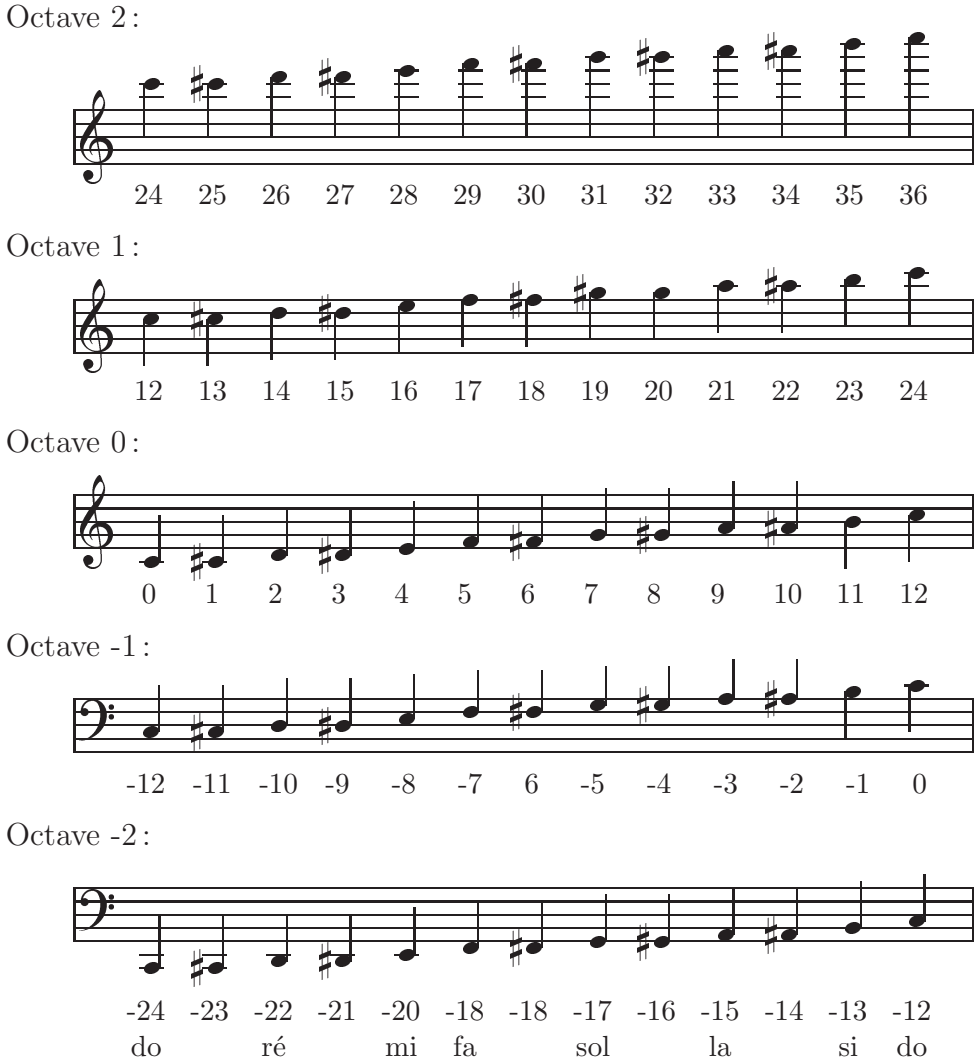


FIG. 18.3 – *Codage de la hauteur des notes de musique par des entiers*

sont généralement définis par des termes italiens (adagio, allegro, prestissimo,...) que l'on place au début d'un morceau et au dessus de la portée. On peut les définir en PASCAL à l'aide des déclarations de constantes suivantes :

```
const
   largo = 50;
   larghetto = 63;
   adagio = 71;
   andante = 92;
   moderato = 114;
   allegro = 144;
   presto = 184;
   prestissimo = 204;
```

18.4.3 Notes

Le numérotage des notes de la figure 18.3 (page 205) n'est pas facile à mémoriser. C'est pourquoi nous allons coder les notes par une expression entière portant sur des constantes dont il est facile de se souvenir.

Une note est définie par son nom n (à savoir do, ré, mi, fa, sol, la, si), son éventuelle altération a (c'est-à-dire un signe qui modifie le son de la note à laquelle il est affecté qui peut être le bémol ♭ (qui abaisse le son de la note d'un demi-ton), le dièse ♯ (qui élève le son de la note d'un demi-ton) ou le bécarre ♮ (qui détruit l'effet du dièse ou du bémol)) et son octave o (qui indique à quelle série de notes portant le même nom appartient cette note). Nous codons les octaves du grave à l'aigu par des entiers -3, -2, ..., 3, 4 , l'octave 0 commençant par le do au centre du clavier du piano. Définissons les constantes suivantes en PASCAL[1] :

```
const
  { notes : }
    ut = 0;
    d0 = 0;
    re = 2;
    mi = 4;
    fa = 5;
    sol = 7;
```

1. On remarquera que la note **do** correspond au mot anglais utilisé dans la boucle "for". Ce mot ne pouvant pas être utilisé pour un autre usage en PASCAL nous sommes dans l'obligation de trouver un autre nom pour la note do. Nous laissons le choix entre `ut` ou `d0` (lettre d suivie du chiffre zéro).

```
la = 9;
si = 11;
{ altérations : }
bemol = −1;
becarre = 0;
diese = 1;
{ octaves : }
octave = 12;
```

Le numéro de la note tel qu'il est défini à la figure 18.3 (page 205) est donné par la valeur de l'expression entière $(((o{*}octave){+}n){+}a)$. Si la note n'est pas altérée, son code est simplement $((o{*}octave){+}n)$. Par exemple le do du milieu du clavier du piano correspond à $o = 0$, $n =$`d0` soit $((0{*}\texttt{octave}){+}0) = 0$. Le ré dièse, une octave plus haut a pour code $(((1{*}\texttt{octave}){+}\texttt{re}){+}\texttt{diese}) = (((1 * 12){+}2){+}1) = 15$. Le ré bémol voisin a pour code $(((1{*}\texttt{octave}){+}\texttt{re}){+}\texttt{bemol}) = (((1 * 12) + 2) + (-1)) = 13$.

18.4.4 Silences

La procédure `silence`(s) permet d'interrompre le son pour une durée relative fixée par la figure de silence s qui peut être l'une des constantes `pause`, `pause_pointee`, `demi_pause_pointee`, `demi_pause`, `soupir`, `soupir_pointe`[2], `demi_soupir`, `demi_soupir_pointe`, `quart_de_soupir`, `quart_de_soupir_pointe`, `huitieme_de_soupir`, `huitieme_de_soupir_pointe` ou `seizieme_de_soupir`.

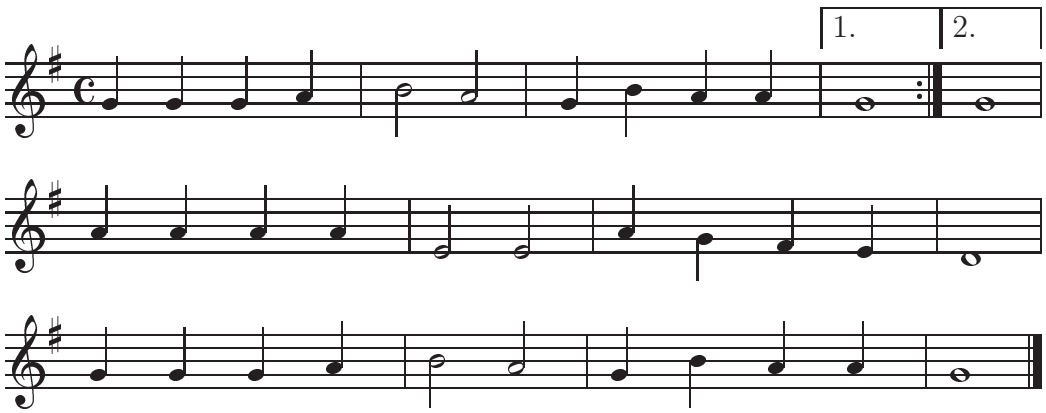
18.4.5 Nuances

Les nuances sont les différents degrés de force par lesquels peuvent passer un ou plusieurs sons, un trait ou un morceau entier. La procédure `nuance`(n) définit la nuance avec laquelle doit être jouée la suite du morceau. Les termes n de nuances possibles sont, du plus faible au plus fort, les suivants : `pianissimo`, `piano`, `mezzo_piano`, `un_poco_piano`, `sotto_voce`, `mezza_voce`, `un_poco_forte`, `mezzo_forte`, `forte` ou `fortissimo`.

Exemple 22 (Au clair de la lune) "Au clair de la lune", qui aurait été composée par le musicien Jean-Baptiste Lulli, est sans doute la plus célèbre

2. Il n'est pas d'usage de pointer le soupir.

des chansons enfantines françaises :



Le programme ci-dessous permet de la faire jouer par l'ordinateur :

```
program AuClairDeLALune;
  uses  ;

  { Au clair de la lune (chanson attribuée à Jean-Baptise Lulli) }

  procedure Strophe;
    var I : integer;
  begin
    metronome(moderato); { mesure 4 / 4 } nuance(piano);
    for I := 1 to 2 do
      begin
        note(sol, noire); { Au / Prê-} note(sol, noire); { clair / -te }
        note(sol, noire); { de / moi } note(la, noire); { la / ta }
        note(si, blanche); { lu- / plu- } note(la, blanche); { ne / -me }
        note(sol, noire); { Mon / Pour } note(si, noire); {a- / é- }
        note(la, noire); { mi / -crire } note(la, noire); { Pier- / un }
        note(sol, ronde); { -rot. / mot. }
      end;
    note(la, noire); { Ma } note(la, noire); { chan- }
    note(la, noire); { -delle }  note(la, noire); { est }
    note(mi, blanche); { mor- } note(mi, blanche); { -te. }
    note(la, noire); {  Je } note(sol, noire); { n'ai }
    note(fa + diese, noire); { plus } note(mi, noire); { de }
    note(re, ronde); { feu. }
    note(sol, noire); { Ou- } note(sol, noire); { -vre }
    note(sol, noire); { moi } note(la, noire); { ta }
    note(si, blanche); { por- } note(la, blanche); { -te, }
    note(sol, noire); { Pour } note(si, noire); { l'a- }
```

```
    note(la, noire); { -mour } note(la, noire); { de }
    note(sol, ronde); { Dieu. } silence(pause);
  end;

  procedure Chanson;
    var J : integer;
  begin
    for J := 1 to 3 do
      begin
        Strophe;
      end;
  end;

begin
  Chanson;
end.
```

□

Exercice 34

_ *Écrire des procédures* `DebutTransmission`, `A`, ..., `Z`, `0`, ..., `9`, `Po` *(point),* `Vg` *(virgule),* `Pv` *(point-virgule),* `Dp` *(deux-points),* `Pi` *(point d'interrogation),* `Pe` *(point d'exclamation),* `Tu` *(trait d'union),* `Ap` *(apostrophe),* `Di` *(barre de division),* `Erreur`, `FinTransmission`, `Fm` *(fin de mot) et* `ALaLigne` *(passer à la ligne suivante) puis les utiliser pour coder des messages en morse, chaque signe du message en clair étant codé en appelant l'une de ces procédures. Par exemple "*message codé*" sera traduit par la suite d'instructions "* `M; E; S; S; A; G; E; Fm; ALaLigne C; O; D; E; Fm;` *". On pourra ajouter le son en utilisant les procédures suivantes du* Robot *:*

- `Bip` *produit un son court ;*
- `Biiip` *produit un son long ;*
- `Delai`(t) *fait attendre pendant* `t` *secondes.*

_ *Avant l'invention de l'électricité, on utilisait en France le télégraphe aérien inventé en 1793 par le citoyen Claude Chappe. Le télégraphe est constitué par trois bras de bois articulés à l'extrémité d'un poteau installé sur une tour au sommet d'une colline ou sur un clocher d'église. Dès qu'il aperçoit à la lunette un signe d'activité au poste le plus proche, le* télégraphier *s'empresse de manœuvrer à son tour les bras de sa machine pour transmettre (sans le comprendre) le message au poste suivant. De proche en proche le message arrive à destination. Il s'agit en quelque sorte d'une version perfectionnée des*

signaux de fumée des Indiens ! Après une première expérience le 12 juillet 1793, la première ligne fut ouverte entre Paris et Lille le 17 août 1794 [3]. Les signaux du télégraphe optique de Chappe pouvaient être décodés selon plusieurs codes. Le code alphabétique est donné à la figure 18.4 (page 211). Écrire un programme qui permet d'afficher successivement, un par un, les signaux et envoyer le message "Paris est tranquille et les bons citoyens sont contents" que Bonaparte fit diffuser immédiatement après le coup d'État du 18 brumaire (9-10 novembre 1799).

Pour laisser apparaître le signal un certain temps on utilisera la procédure `Delai`(t) *qui fait attendre pendant t secondes. Pour effacer l'écran entre chaque signal, on utilisera la procédure* `cf(blanc)` *qui peint le fond de l'écran en blanc (ou dans une autre couleur selon le paramètre) et place le Robot dans les conditions initiales (visible au centre de l'écran, orienté vers le nord, crayon noir d'épaisseur 1 baissé, avec une grille de déplacement (12 × 12)).*

– *Écrire un programme Pascal de traduction des lettres de l'alphabet en alphabet phonétique selon le code international de la table 18.3 (page 211).*

– *Écrire un programme Pascal de cryptage et de décryptage de messages au moyen d'un code secret simple, obtenu par exemple par permutation des lettres de l'alphabet.*

□

Exercice 35

– *Quelle est la représentation binaire sur 16 bits du nombre qui s'écrit 127 en base 10 ?*

– *Quelle est l'écriture décimale du nombre entier positif représenté en binaire sur 16 bits par 1101010101010101 ?*

– *Quel est le texte représenté en code* ASCII *par les octets suivants : 01000100 01100101 01110011 01100011 01100001 01110010 01110100 01100101 0111-0011 ?*

□

Exercice 36 *Les chansons sont des sources inépuisables d'exercices [6], [32], [33], [34]. Écrire par exemple des programmes* Pascal *pour jouer* Greensleeves *(figure 18.5, page 212) et* O Tannenbaum *(figure 18.6, page 213). On pourra également dérouler la partition musicale ligne par ligne sur l'écran en même temps qu'elle est jouée par l'ordinateur en représentant la portée et les notes comme à la figure 18.7 (page 213).* □

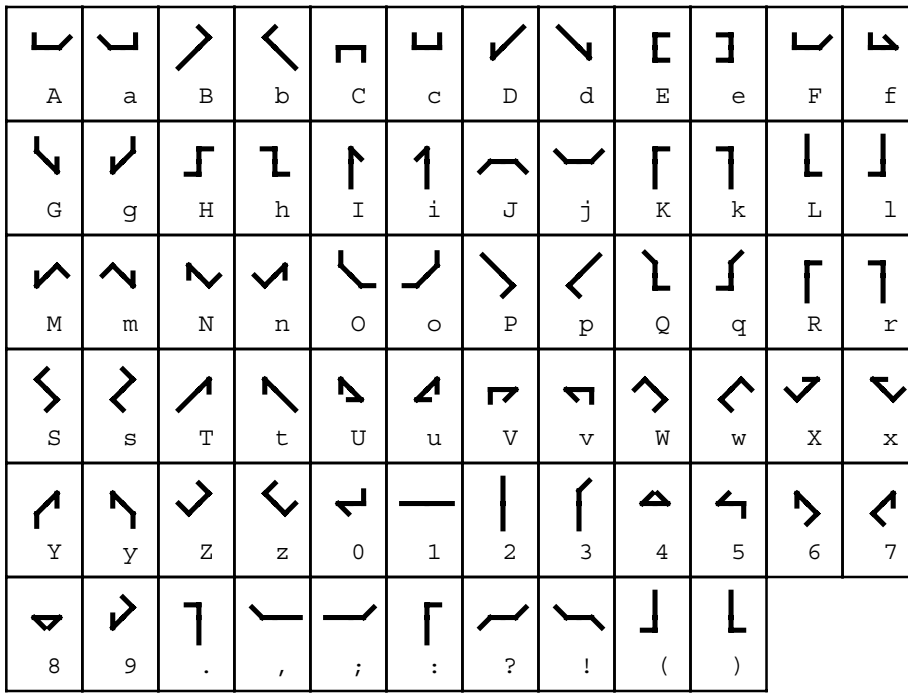


FIG. 18.4 – *Alphabet du télégraphe optique de Chappe*

A	Alpha	B	Bravo	C	Charlie	D	Delta	E	Echo
F	Foxtrot	G	Golf	H	Hotel	I	India	J	Juliet
K	Kilo	L	Lima	M	Mike	N	November	0	Oscar
P	Papa	Q	Quebec	R	Romeo	S	Sierra	T	Tango
U	Uniform	V	Victor	W	Whiskey	X	Xray	Y	Yankee
Z	Zulu								

TAB. 18.3 – *Alphabet phonétique international*

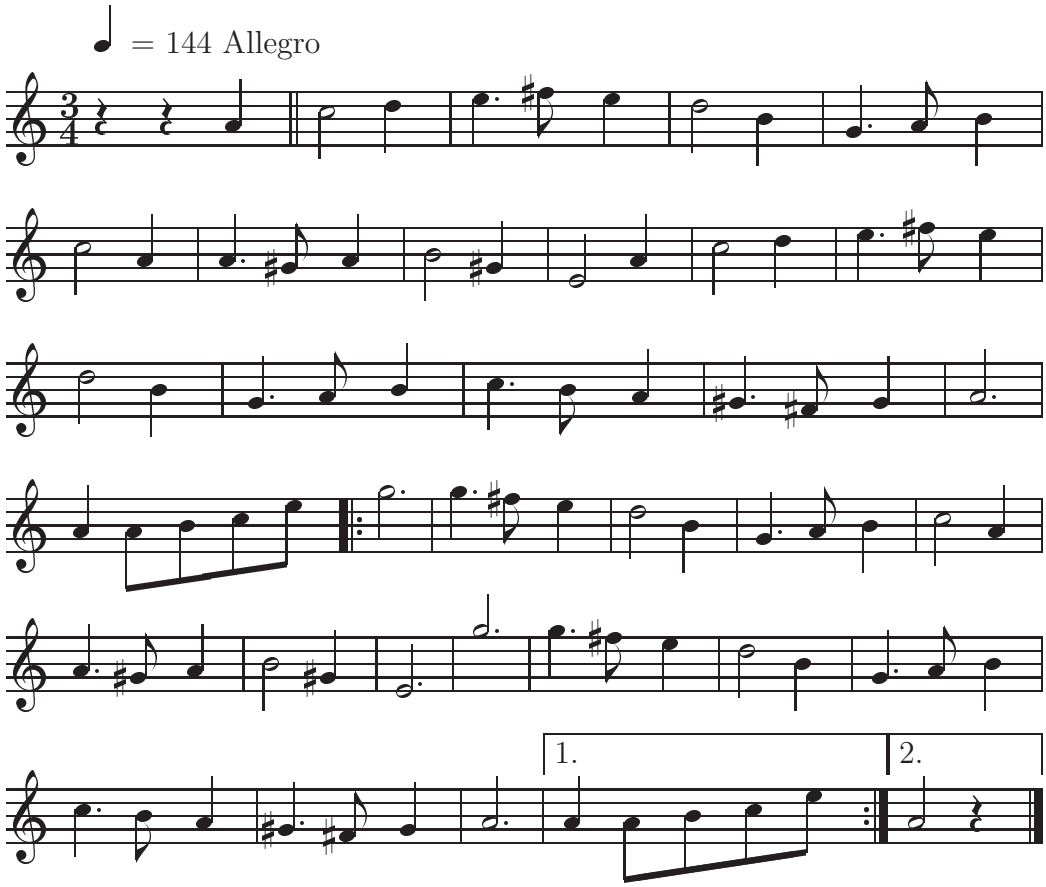


FIG. 18.5 – *Greensleeves (chanson irlandaise)*

Corrigé 34 *Nous donnons un programme, les autres se trouvant sur disquette :*

```
program CodeMorse1;
  uses  ;

  procedure ALaLigne;
    { Passer au debut de la ligne suivante. }
  begin lc; ag; ps; av; pe; av; bc; end;

  procedure Pt; { Point }
  begin dp; bip; lc; av; bc; delai(0.1); end;

  procedure Bl; { Blanc }
  begin lc; av; bc; delai(0.1); end;
```

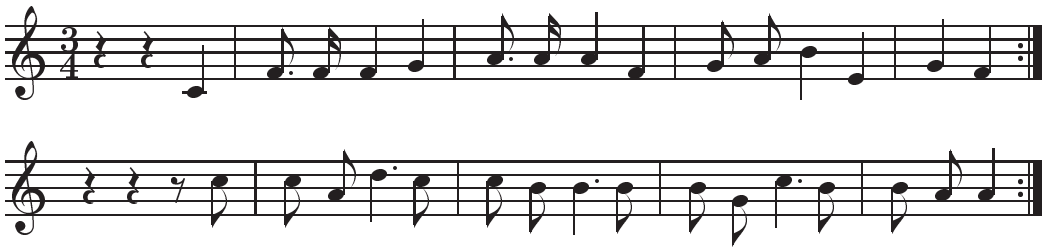


FIG. 18.6 – *O Tannenbaum (chanson allemande)*

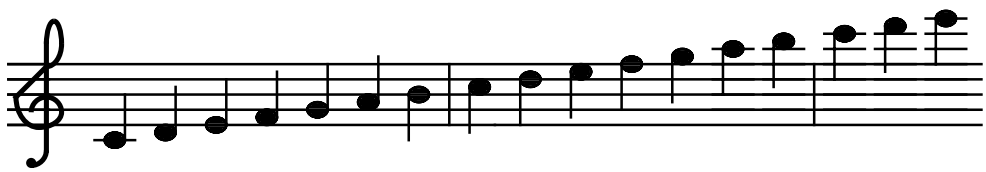


FIG. 18.7 – *Portée et notes représentées sur l'écran de l'ordinateur*

```
procedure Tr; { Trait }
begin av; biiip; lc; av; bc; delai(0.1); end;

{ Alphabet }
procedure A; begin Pt; Tr; Bl; end;

procedure Z; begin Tr; Tr; Pt; Pt; Bl; end;

{ Chiffres }
procedure C1; begin Pt; Tr; Tr; Tr; Tr; Bl; end;

procedure C0; begin Tr; Tr; Tr; Tr; Tr; Bl; end;

procedure Po; { Point } begin Pt; Pt; Pt; Pt; Pt; Pt; Bl; end;

procedure Di; { Barre de division }
  begin Tr; Tr; Tr; Tr; Tr; Tr; Bl; end;
procedure Fm; { Fin de mot } begin lc; av; bc; delai(0.6); end;

procedure Erreur;
begin Pt; Pt; Pt; Pt; Pt; Pt; Pt; Pt; Pt; Pt; Pt; Bl; end;

procedure DebutTransmission;
begin ec(2); eh; ALaLigne; Tr; Pt; Tr; Pt; Tr; Pt; Tr; Bl; end;
```

```
  procedure FinTransmission;
  begin Pt; Tr; Pt; Tr; Pt; Tr; Pt; Bl; end;

begin
  DebutTransmission;
  A; B; C; D; E; ALaLigne;

  C9; C0; Po; Vg; ALaLigne;
  Pv; Dp; Pi; Pe; ALaLigne;
  Tu; Ap; Di; ALaLigne;
  Erreur; FinTransmission; st;
end.
```

On trouvera sur la disquette diverses améliorations possibles de ce programme, notamment un retour automatique à la ligne. □

Corrigé 35

– *La représentation binaire sur 16 bits de 127 est 0000000001111111.*

– *Le nombre représenté par le demi-mot 1101010101010101 s'écrit 13653 en base 10.*

– *La suite d'octets 01000100 01100101 01110011 01100011 01100001 01110010 01110100 01100101 01110011 soit, en notation décimale 68 101 115 99 97 114 116 101 115, représente la chaîne de caractères* `'Descartes'`.
□

Corrigé 36 *Nous donnons un programme, de nombreuses autres chansons se trouvant programmées sur disquette :*

```
program OTannenbaum;
  uses   ;

  procedure Refrain;
  begin
    note(d0, noire); { O } note(fa, croche_pointee); { Tan- }
    note(fa, double_croche); { -nen- } note(fa, noire); { -baum, }
    note(sol, noire); { o } note(la, croche_pointee); { Tan- }
    note(la, double_croche); { -nen- } note(la, noire_pointee);
    { -baum, } note(fa, croche); { wie } note(sol, croche); { treu }
    note(la, croche); { sind } note(si+bemol, noire); { dei- }
    note(mi, noire); { -ne } note(sol, noire); { Blät- }
    note(fa, noire); { -ter! }
  end;
```

```
  procedure Couplet;
  begin
    silence(demi_soupir); note(d0 + octave, croche); { Du }
    note(d0 + octave, croche); { grünst } note(la, croche); { nicht }
    note(re+octave, noire_pointee); { nur } note(d0 + octave, croche);
    { zur } note(d0 + octave, croche); { Som- }
    note(si + bemol, croche); { -mer- } note(si + bemol, noire_pointee);
    { -zeit } note(si + bemol, croche); { Nein, }
    note(si + bemol, croche); { auch } note(sol, croche); { im }
    note(d0+ octave, noire_pointee); { Win- } note(si + bemol, croche);
    { -ter } note(si + bemol, croche); { wenn } note(la, croche); { es }
    note(la, noire); { schneit. }
  end;

  procedure Chanson { allemande };
  begin
    metronome(adagio); nuance(mezza_voce); Refrain; Couplet; Refrain;
  end;

begin
  Chanson;
end.
```

□

Addenda

En complément nous expliquons deux représentations des entiers relatifs, la seconde étant la plus employée.

Représentation signée des entiers relatifs : Pour coder un entier relatif (positif ou négatif) sur n bits, on peut réserver un bit pour le signe et coder la valeur absolue de l'entier sur les $n-1$ bits restants. Prenons la convention de placer le signe en tête en choisissant 0 pour le signe $+$ et 1 pour le signe $-$. Le code de -10 sur un octet sera 10001010 soit 1 pour le signe suivi de 0001010 pour la valeur absolue 10 de -10. Avec cette convention on peut coder sur un octet les entiers compris entre -127 (11111111) et 127 (01111111). L'entier nul est représenté par 00000000 $(+0)$ ou bien par 10000000 (-0).

Représentation en complément à deux des entiers relatifs : Dans la représentation en complément à deux sur n bits, les nombres positifs strictement inférieurs à 2^n sont représentés comme précédemment en base 2 sur $n-1$ bits, le premier bit de signe étant nul. Un nombre négatif $-x$ tel que $x \leq 2^n$ est représenté par un premier bit de signe égal à 1 suivi de la représentation du nombre positif $p < 2^n$ tel que $-x = -2^n + p$ codé sur $n-1$ bits. Par exemple pour coder - 1 sur un octet, on écrit $-1 = -128 + 127$ et l'on code 127 sur 7 bits ce qui donne 1111111 que l'on fait précéder de 1 pour trouver 11111111. Avec cette convention on peut également coder $-128 = -128 + 0$ ce qui donne le code 10000000. Cette méthode permet de coder sur un octet tous les entiers relatifs compris entre -128 et +127. Sur 16 bits, on pourra coder les entiers compris entre -32768 et 32767 ce qui correspond le plus souvent aux variables entières de type `integer` en PASCAL. Sur 32 bits on pourra coder les entiers compris entre -2147483648 et 2147483647 ce qui correspond le plus souvent aux variables entières de type `longint` en PASCAL. Cette représentation conduit à des algorithmes d'addition et soustraction très simples. De plus l'algorithme pour le codage d'un nombre négatif $-x$ est également simple. On code le nombre positif x sur $n-1$ bits puis on inverse les bits 0 en 1 et les bits 1 en 0 puis on ajoute 1. Par exemple pour coder -127, on code 127 en 01111111 puis on inverse tous les bits ce qui donne 10000000 et on ajoute 1 pour trouver 10000001.

19

Expressions rationnelles

19.1 Nombres rationnels

Les paramètres de procédures et les variables `X` que nous avons utilisés jusqu'à présent ont des valeurs entières comprises entre -32768 et 32767, ce que l'on indique par `X : integer` dans leur déclaration. On peut également utiliser des paramètres de procédures ou des variables ayant des valeurs entières comprises entre -2147483648 et 2147483647, ce qu'il faut indiquer par `X : longint` dans la déclaration. Ces deux types d'entiers sont codés en base deux dans la mémoire de l'ordinateur, l'un sur 16 bits et l'autre sur 32.

Nous pouvons également utiliser en Pascal des nombres à virgule comme `3.14159`. Ils s'écrivent selon la convention anglo-saxonne avec un point à la place de notre virgule. Les mathématiciens appellent ces nombres des *nombres rationnels*. En Pascal, on distingue les nombres entiers des nombres rationnels par la présence du point. Par exemple 1000 est un entier alors que 1000.0 est un nombre rationnel. Dans l'écriture d'un nombre rationnel, il faut toujours écrire au moins un chiffre avant et après le point. Par exemple .0 et 1. sont incorrects. Il faut écrire 0.0 et 1.0.

Les nombres entiers et rationnels sont codés de manières différentes dans la mémoire de l'ordinateur.

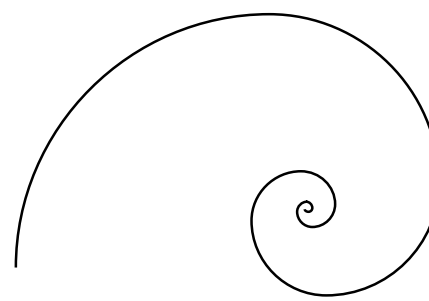
19.2 Paramètres et variables de type “real”

On indique que les valeurs d'un paramètre formel ou d'une variable `X` sont des nombres rationnels par la déclaration `X : real`[1]. Le paramètre effectif correspondant peut être un nombre rationnel ou entier, le compilateur se chargeant dans ce dernier cas de coder l'entier comme doivent l'être les nombres rationnels.

Les procédures de commande du $\mathcal{R}^{o}\!\!\smile^{o}t$ `lg`, `lgX`, `lgY`, `translater` et `deplacer` ont des paramètres formels de type `real` (et peuvent donc avoir des paramètres effectifs qui sont entiers ou rationnels).

Exemple 23 (Spirale arithmétique) La spirale ci-dessous est constituée d'arcs de cercles de rayons successifs 100; $100/1.5 = 66.666\ldots$; $66.666\ldots/1.5 = 44.444\ldots$; $44.444\ldots/1.5 = 29.629\ldots$; etc :

```
program SpiraleArithmetique;
  uses  ;
  procedure Arc(L : real);
  begin
    lg(L); vd;
  end;
begin
  Arc(100); Arc(66.666666667);
  Arc(44.444444444); Arc(29.62962963);
  Arc(19.75308642); Arc(13.16872428);
  Arc(8.7791495199); Arc(5.8527663466);
  Arc(3.9018442311); Arc(2.6012294874);
  Arc(1.7341529916); Arc(1.1561019944);
  st;
end.
```



□

19.3 Expressions rationnelles

L'addition ($+$), la soustraction ($-$) et la multiplication ($*$) de nombres rationnels se notent comme pour les entiers. Par contre on utilise la barre

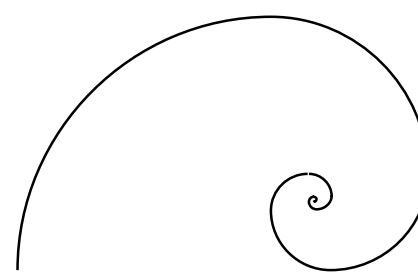
1. Le choix du terme “real” (réel en français) est malheureux dans la mesure où il ne s'agit pas de nombres réels puisque les nombres de type `real` ne peuvent pas avoir une infinité de chiffres différents de zéro après la virgule !

de fraction / pour la division de nombres rationnels. Par exemple $(((2 * (10.3 + 0.7)) - 1.0)/3) = 7.0$. Si r est un rationnel (ou un entier), `sqr`(r) est son carré r^2 (de l'anglais square, carré), `sqrt`(r) est sa racine carrée $\sqrt{r}$ (de l'anglais square root, racine carrée), `abs`(r) est sa valeur absolue, `trunc`(r) est sa partie entière (de l'anglais truncate, tronquer) et `round`(r) est l'entier le plus proche (de l'anglais round, arrondir). Par exemple `trunc`$(0.3) =$ `trunc`$(0.5) =$ `trunc`$(0.9) = 0$ tandis que `round`$(0.3) = 0$ et `round`$(0.5) =$ `round`$(0.9) = 1$.

Dans une expression rationnelle, on peut utiliser la longueur horizontale `valLgX` et la longueur verticale `valLgY` d'un carreau de la grille de déplacement du $\mathcal{R}o\mathcal{b}ot$. Par exemple `lgX(valLgX * 2)` double la taille horizontale d'un carreau de la grille.

Exemple 24 (Spirale d'or) La spirale d'or ci-dessous est constituée d'arcs de cercles successifs dont les rayons ont pour rapport le *nombre d'or* $\Phi = \frac{1}{2}(1 + \sqrt{5}) \simeq 1.618033989$. Pour les tenants d'esthétique traditionnelle, la spirale d'or constituerait une figure géométrique très harmonieuse à l'œil.

```
program SpiraleDor;
  uses  ;
  var I : integer;
begin
  lg(100);
  for I := 1 to 12 do
    begin
      vd;
      lg(valLgX / (1 + sqrt(5)) * 2);
    end;
  st;
end.
```



□

19.4 Erreurs d'arrondi

Pour les nombres réels, qui ont un nombre infini de chiffres après la virgule, comme $\frac{1}{3} = 0.3333333333333333\ldots$ ou $\pi = 3.1415926535897932\ldots$ l'ordinateur ne peut garder dans sa mémoire que quelques chiffres après la

virgule, ce qui conduit, comme pour une calculette, à des erreurs d'arrondi dans les calculs. Par exemple en mathématiques on a :

$$\left(\left(\left(\left(\left(\sqrt{\sqrt{\sqrt{\sqrt{2}}}}\right)^2\right)^2\right)^2\right)^2\right)^2 = \left(\left(\left(\left(\sqrt{\sqrt{\sqrt{\sqrt{2}}}}\right)^2\right)^2\right)^2\right)^2 =$$

$$\left(\left(\left(\sqrt{\sqrt{\sqrt{2}}}\right)^2\right)^2\right)^2 = \left(\left(\sqrt{\sqrt{2}}\right)^2\right)^2 = \left(\sqrt{2}\right)^2 = 2$$

tandis que sur certains ordinateurs on peut trouver que :

`sqr(sqr(sqr(sqr(sqr(sqrt(sqrt(sqrt(sqrt(2)))))))))` `=`
1.99999999999999997.

On retrouve des problèmes d'arrondi similaires avec la grille de déplacement du $\mathcal{R}^{o}_{b}{}^{o}t$. De manière idéale, les coordonnées cartésiennes du $\mathcal{R}^{o}_{b}{}^{o}t$ devraient être réelles. L'ordinateur en conserve en mémoire une approximation rationnelle. Sur l'écran elles doivent être arrondies à l'entier le plus proche pour correspondre à un pixel de l'écran.

Exercice 37 *Écrire des programmes* Pascal *pour réaliser les motifs de la figure ci-dessous (seul le motif central qui utilise les fonctions trigonométriques introduites dans l'addenda du chapitre 21 est difficile).* □

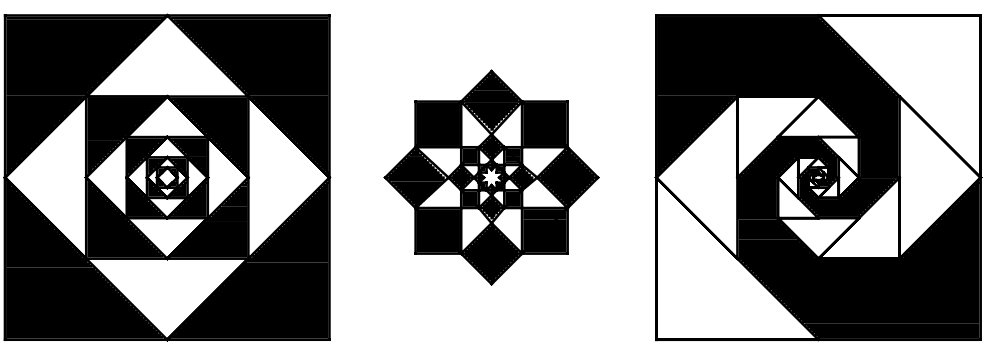


Fig. 19.1 – *Motifs de Horemis et Beard ; spirale de Baravelle*

Corrigé 37 *Nous donnons un programme, les autres se trouvant sur disquette :*

```
program CarresImbriquesPeints;
 uses  ;

 { Motif de S. Horemis, 'Optical and geometrical patterns and }
 { designs', Dover, New York, 1970.                           }

 const
  N = 6;
  DeuxPuissanceN = 64; { 2 à la puissance N }
 var
  I : integer;
  J : integer;
begin
 eb; ag; lg(DeuxPuissanceN);
 for I := 1 to N do
  begin
   for J := 1 to 4 do
    begin
     avf(4); pqtd;
    end;
   avf(2); phtd;
   for J := 1 to 4 do
    begin
     avf(2); avf(-1); pqtd; peindre; pqtg; av; pqtd;
    end;
   av; phtd;
   lg(valLgX / 2);
  end;
 st;
end.
```

□

Addenda

Les nombres rationnels sont qualifiés de réels en PASCAL par abus de langage. Ils peuvent se noter en utilisant la *notation scientifique*, qui consiste à faire suivre la partie fractionnaire par un facteur d'échelle formé de la lettre e ou E (qui se lit "dix à la puissance") suivie d'un exposant.

Par exemple $-3.0\text{e}2 = -300.0$, $3.0\text{e}0 = 3.0$ et $3.0\text{e}-2 = 0.03$. Un nombre réel ne contient pas de blanc. Par conséquent $-3.0␣\text{e}␣2$ est erroné. Sur les micros-ordinateurs, les réels sont généralement représentés avec 7 ou 8 chiffres significatifs, le plus petit réel strictement positif étant de l'ordre de $1.175\text{e}-38$ et le plus grand de l'ordre de $3.403\text{e}38$.

La valeur d'une expression arithmétique peut être de type `longint` ou `real`, selon les règles suivantes :

- Si toutes les constantes et variables de l'expression sont entières (de type `integer` ou `longint`) et si tous les opérateurs de l'expression sont à résultats entiers (comme $+$, $-$, $*$, **div**, **mod**, `sqr`[a], `abs` si leurs arguments sont entiers ou `trunc` et `round`) alors l'expression est à résultat entier de type `longint`.
- Si l'expression contient une constante réelle (avec un point ou un exposant `e` ou `E`) ou une variables réelle (figurant dans une déclaration `identificateur : real`) ou un opérateur à résultat réel (comme $+$, $-$, $*$, `sqr` ou `abs` si l'un des arguments est réel ou / ou `sqrt`) alors l'expression est à résultat de type `real`.

En PASCAL, les paramètres effectifs doivent avoir le même type que les paramètres formels, sauf dans les cas suivants :

- Si le type du paramètre formel est `integer` alors le paramètre effectif peut être de type `longint` à condition que sa valeur soit comprise entre -32768 et 32767 ;
- Si le type du paramètre formel est `real` alors le paramètre effectif peut être de type `longint` ou `integer`.

Dans les deux cas le compilateur se charge de coder l'entier comme doivent l'être les grands entiers ou les nombres rationnels.

---

[a] Nous utilisons des typographies différentes pour les *identificateurs réservés* (comme **program** ou **mod**) et les *identificateurs prédéfinis* (comme `integer` ou `sqr`). Les identificateurs introduits dans un programme PASCAL (comme les noms de variable, de procédure, etc.) doivent être différents des identificateurs réservés mais peuvent être identiques aux identificateurs prédéfinis. Cette pratique est déconseillée car les programmes qui changent le sens habituel des identificateurs prédéfinis sont illisibles.

20

Rotations et translations

20.1 Dimensions de la grille de déplacement du robot en coordonnées polaires

Les dimensions horizontales `X` et verticales `Y` d'un carreau de la grille de déplacement du *Robot* peuvent être définies en coordonnées cartésiennes par les commandes `lgX(X)`. et `lgY(Y)` comme indiqué à la figure 20.1 ci-dessous.

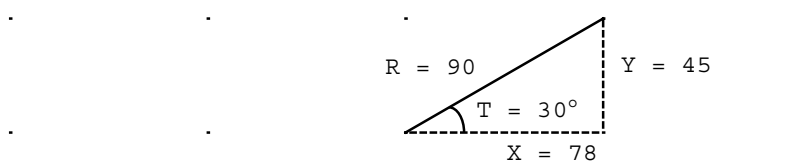


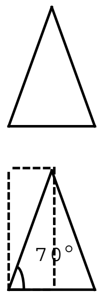
FIG. 20.1 – *Grille de déplacement du Robot définie par* `lgX(X); lgY(Y);` `ou lgRT(R, T);`

On peut également définir les dimensions d'un carreau du quadrillage en *coordonnées polaires* par la commande `lgRT(R, T)` en donnant la longueur `R` de la diagonale de ce carreau mesurée en pixels et l'angle `T` exprimé en degrés que fait cette diagonale avec une horizontale comme illustré à la figure 20.1 ci-dessus.

Exemple 25 (Triangle isocèle) Le programme ci-dessous dessine un tri-

angle isocèle ayant deux angles égaux à 70 degrés et deux côtés égaux à 50 pixels :

```
program TriangleIsocele;
  uses  ;
  const
    LongueurCote = 50;
    Angle = 70;
begin
  lgRT(LongueurCote, Angle);
  phtd; av;
  pqtd; av;
  p3htd; avf(2);
  st;
end.
```



□

20.2 Rotations

La commande `rt`(t) fait effectuer au *Robot* une rotation de t degrés. Si la valeur de l'expression réelle t est positive alors le pivotement sur place se fait dans le sens *contraire* des aiguilles d'une montre[1]. Si la valeur de l'expression réelle t est négative alors le pivotement sur place se fait dans le sens des aiguilles d'une montre. Cette commande ne change pas la position du *Robot* mais elle change son orientation. La longueur du segment tracé par le prochain `av` reste également inchangée. Pour ce faire la taille du carreau du quadrillage doit être convenablement modifiée. Ceci est illustré sur la figure 20.2 (page 225) qui montre un carreau du quadrillage en pointillés avant et après des rotations successives de 15 degrés (le *Robot* partant à chaque fois du coin inférieur gauche du carreau et la rotation étant suivie d'un `av` pour montrer que la longueur du segment tracé ne change pas). Sur cette figure, les schémas successifs sont disposés de haut en bas et de gauche à droite.

Quand on utilise la commande `rt`, nous conseillons d'éviter les commandes `phtd`, `pqtd`, `p3htd`, ..., `pne`, `pse`, ...en les remplaçant par `rt` avec un paramètre approprié (seules les commandes `pn`, `ps`, ...sont utiles pour les positionnements absolus). En utilisant exclusivement la commande `rt`,

1. Sens contraire des aiguilles d'une montre que les mathématiciens appellent *sens trigonométrique*.

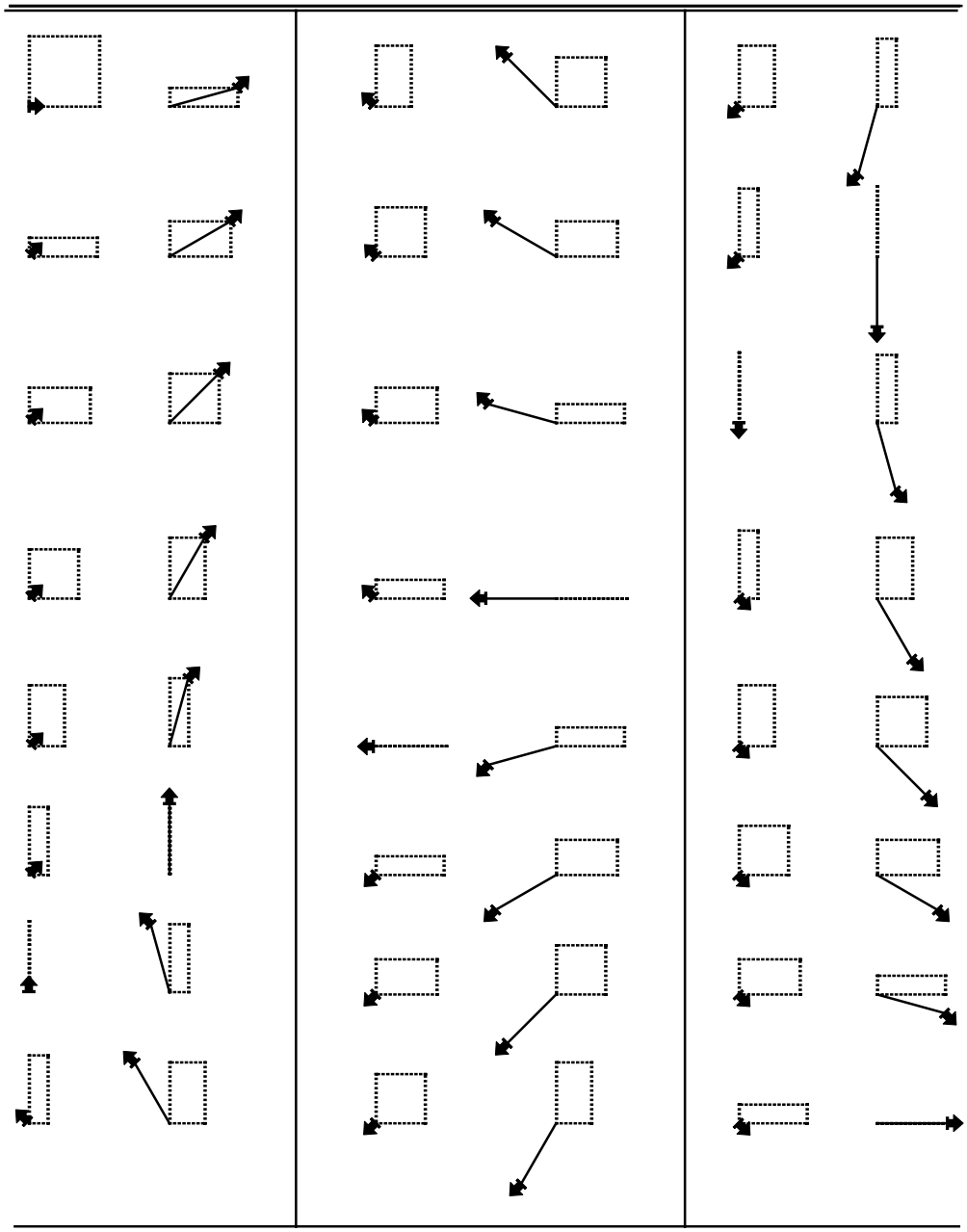
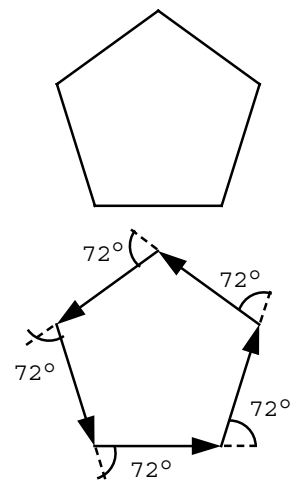


FIG. 20.2 – *Rotations et déplacements successifs* `rt(15); av;`

il n'est pas nécessaire de tenir compte des changements de repères qu'elle induit.

Exemple 26 (Pentagone) Le programme ci-dessous dessine un pentagone en répétant cinq fois le tracé d'un segment suivi d'une rotation de $(360/5) = 72$ degrés :

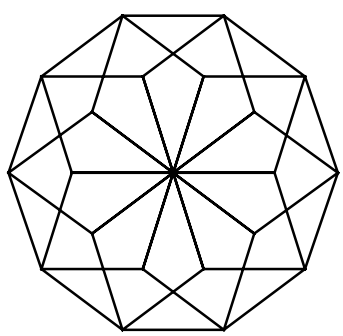
```
program Pentagone_2;
  uses  ;
  const LongueurCote = 50;
  var J : integer;
begin
  lg(LongueurCote); pe;
  for J := 1 to 5 do
    begin
      av;
      rt(72);
    end;
  st;
end.
```



□

Exemple 27 (Rosace de pentagones) La rosace ci-dessous est obtenue par superposition de dix pentagones tournés successivement de 36 degrés :

```
program RosaceDePentagones;
  uses  ;
  const
    LongueurCote = 40;
  var
    I : integer;
    J : integer;
begin
  pqtd; lg(LongueurCote);
  for I := 1 to 10 do
    begin
      rt(36);
      for J := 1 to 5 do
        begin
          av; rt(72);
        end;
    end;
  st;
```



end.

□

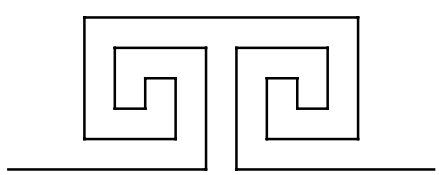
20.3 Translations

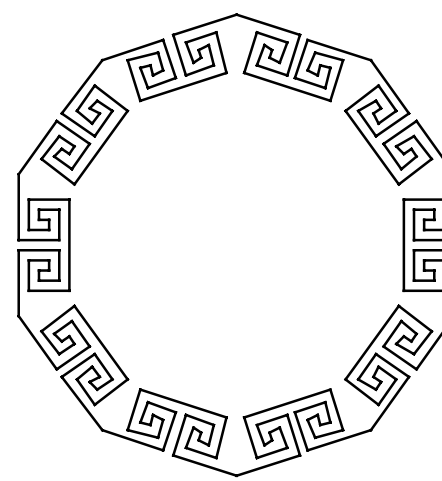
Quelle que soit l'orientation du $\mathcal{R}o\mathcal{b}ot$, l'expression `valLgTl` de type `real` a pour valeur la longueur (mesurée en pixels) d'un segment qui serait tracé par un `av`.

Pour tracer un segment d'une longueur différente, on peut utiliser la commande `tl`(E) qui prend en paramètre une expression E de type `integer` ou `real` dont la valeur est la longueur de ce segment comptée en pixels. Si cette valeur est positive, le segment est tracé en avançant dans la direction courante du $\mathcal{R}o\mathcal{b}ot$. Si cette valeur est négative, le tracé se fait en reculant. Le $\mathcal{R}o\mathcal{b}ot$ est déplacé à l'extrémité du segment et ne change pas son orientation. Les dimensions d'un carreau du quadrillage sont changées pour permettre ce déplacement (si le crayon est levé) ou ce tracé (si le crayon est baissé). Par conséquent, le segment peut être prolongé par un segment d'égale longueur en utilisant une commande `av`.

Par exemple, la séquence de commandes `tl`(20); `rt`(90); `av`; `rt`(90); `av`; `rt`(90); `av`; trace un carré de côté 20 pixels.

Exemple 28 (Grecque polygonale) La frise polygonale ci-dessous est obtenue en remplaçant le tracé d'un côté du polygone par celui de la grecque :





program `GrecquePolygonale`;

```
uses   ;
const
  Fg = 2; { Facteur de grossissement }
  N = 10;  { Nombre de côtés }

procedure Motif;
begin
  { Partie gauche }
  tl(13 * Fg); rt(90); tl(8 * Fg); rt(90); tl(6 * Fg); rt(90);
  tl(4 * Fg); rt(90); tl(2 * Fg); rt(90); tl(2 * Fg); rt(-90);
  tl(2 * Fg); rt(-90); tl(4 * Fg); rt(-90); tl(6 * Fg); rt(-90);
  tl(8 * Fg); rt(-90);
  tl(18 * Fg); rt(-90);
  { Partie symétrique droite }
  tl(8 * Fg); rt(-90); tl(6 * Fg); rt(-90); tl(4 * Fg); rt(-90);
  tl(2 * Fg); rt(-90); tl(2 * Fg); rt(90); tl(2 * Fg); rt(90);
  tl(4 * Fg); rt(90); tl(6 * Fg); rt(90);  tl(8 * Fg); rt(90);
  tl(13 * Fg);
end;

procedure Grecque;
  var I : integer;
begin
  for I := 1 to N do
    begin
      Motif; rt(360 / N);
    end;
end;

begin
  Grecque; st;
end.
```

□

Exercice 38 *Écrire des programmes* Pascal *utilisant la commande* `lgRT` *pour réaliser les dessins suivants :*

- *un triangle équilatéral ;*
- *le pentagone, les croix ou la toile d'araignée de la figure 20.3 (page 229) ;*
- *les motifs géométriques d'après Horemis [16] de la figure 20.4 (page 229) ;*
- *les pavages coloriés de la figure 20.6 (page 230).*

Les très nombreux pavages donnés par Grünbaum et Shephard [14] sont la source d'innombrables exercices. □

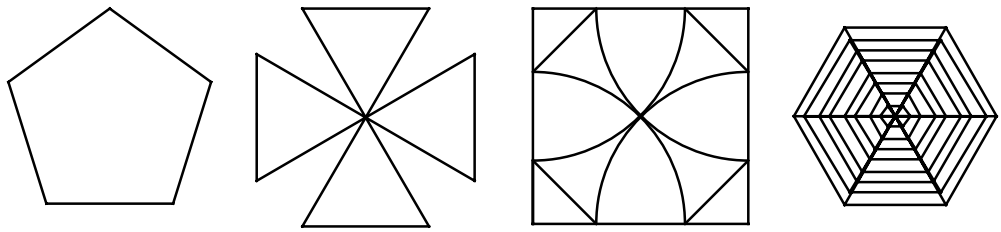


FIG. 20.3 – *Pentagone, croix et toile d'araignée*

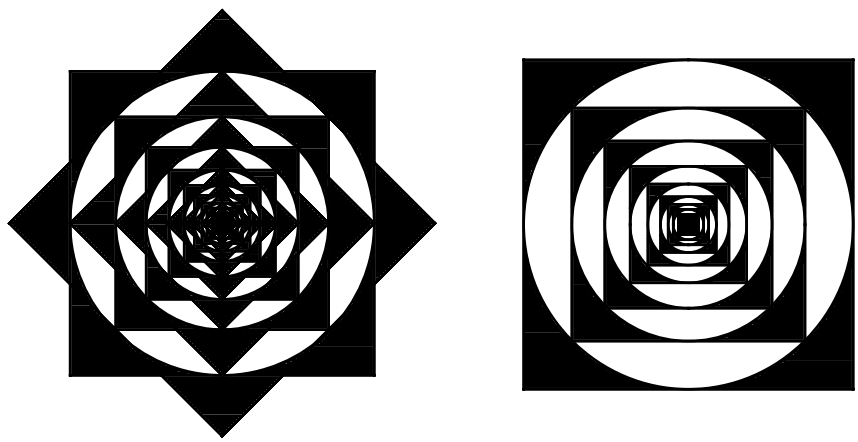


FIG. 20.4 – *Motifs géométriques d'après Horemis*

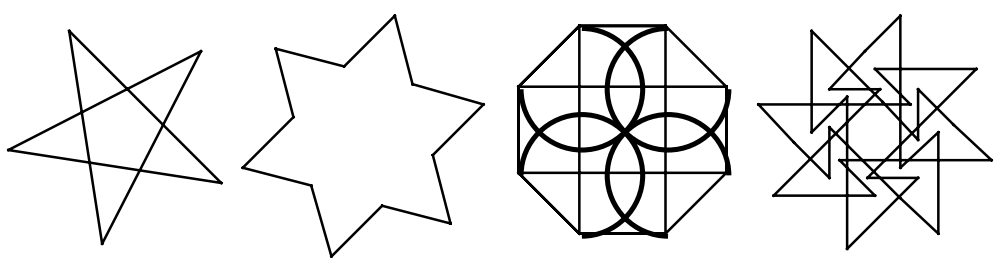


FIG. 20.5 – *Étoiles, octogone décoré et spirolatérale*

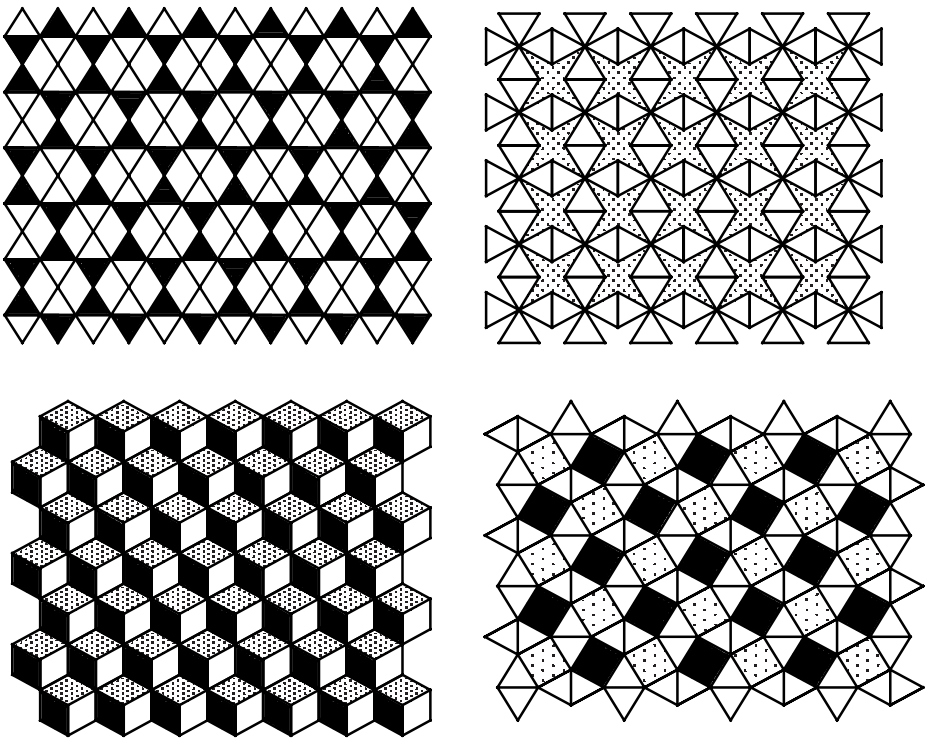


FIG. 20.6 – *Pavages de triangles, losanges et carrés*

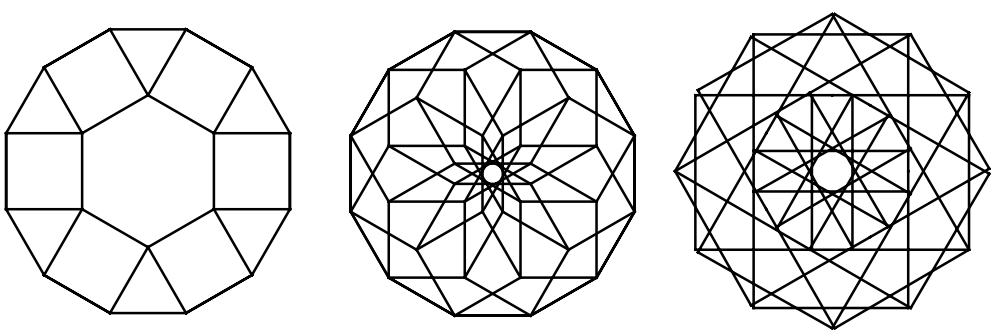


FIG. 20.7 – *Rosaces polygonales*

Exercice 39 *Écrire des programmes* Pascal *utilisant la commande* `rt` *pour réaliser les dessins suivants :*

– *les étoiles, l'octogone décoré ou la courbe spirolatérale d'ordre 3 à 135 degrés de la figure 20.5 (page 229), cette dernière étant obtenue en avançant une, puis deux, puis trois fois, en tournant à chaque fois de 135 degrés et en répétant ces déplacements jusqu'à ce que la courbe soit fermée ;*

– *les polygones réguliers de la figure 20.8 (page 232), le nombre de côtés étant un paramètre de la procédure de dessin ;*

– *les étoiles de la figure 20.9 (page 232), le nombre de branches étant un paramètre de la procédure de dessin ;*

– *le rapporteur de la figure 20.10 (page 232), en utilisant la commande* `Ecrire(EntierEnChaine(`E`))` *qui écrit la valeur de l'expression entière de* E *à droite de la position courante du* Robot *(l'irrégularité des graduations est due aux erreurs d'arrondi en coordonnées d'écran entières qui ne permettent pas de représenter des droites de pentes quelconques) ;*

– *le pavage du plan avec des hexagones ou des étoiles de la figure 20.11 (page 233) ;*

– *le pavage à la Escher [24] du disque et du triangle de la figure 20.12 (page 233) ;*

– *le pavage du disque utilisant le pavé de Voderberg [38] de la figure 20.13 (page 234) ;*

– *les spirales utilisant le pavé de Voderberg [38] de la figure 20.14 (page 234).* □

Exercice 40 *Écrire des programmes* Pascal *utilisant les commandes* `lgRT` *et* `rt` *pour réaliser les développements des solides suivants :*

– *solides de Platon (cube, tétraèdre, octaèdre, dodécaèdre et isocaèdre réguliers) de la figure 20.15 (page 235) ;*

– *les grand et petit rhombicuoctaèdres et le tétraèdre tronqué de la figure 20.16 (page 235).*

Les développements des solides donnés par Cundy et Rollett [11] sont la source de nombreux exercices. □

Exercice 41 *Écrire des programmes* Pascal *utilisant les commandes* `rt` *et* `tl` *pour réaliser les dessins suivants :*

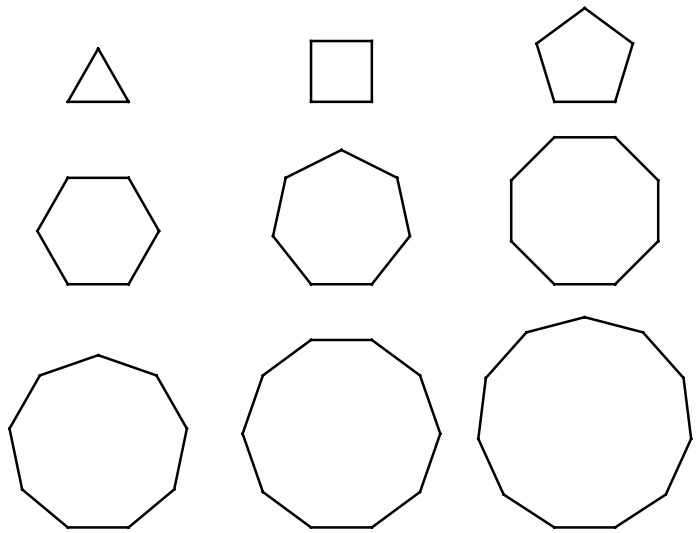


FIG. 20.8 – *Polygones réguliers*

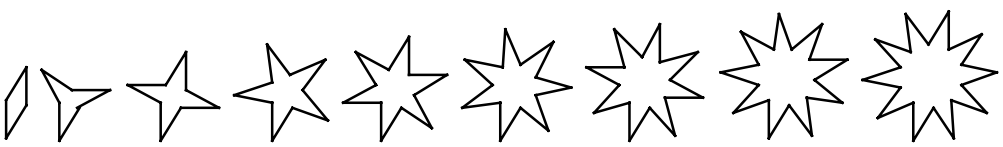


FIG. 20.9 – *Étoiles*

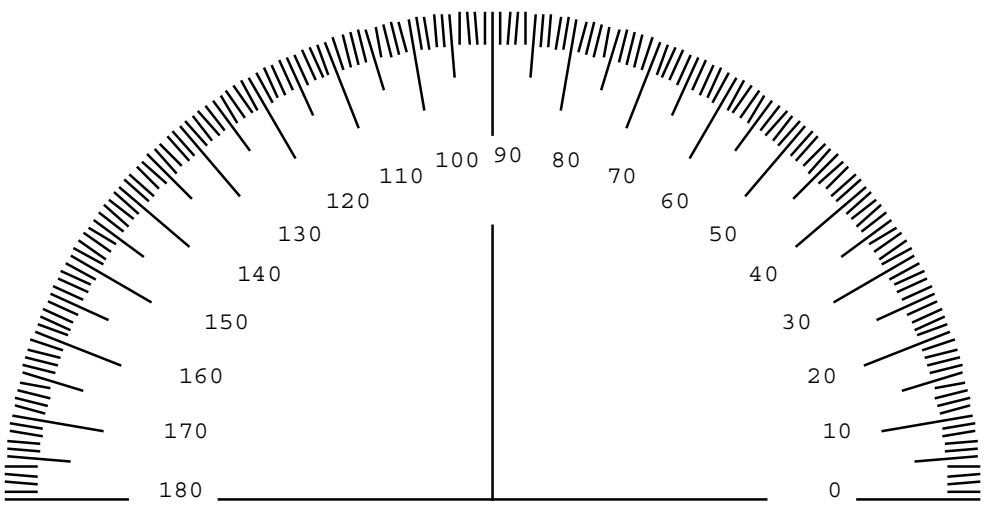


FIG. 20.10 – *Rapporteur gradué en degrés*

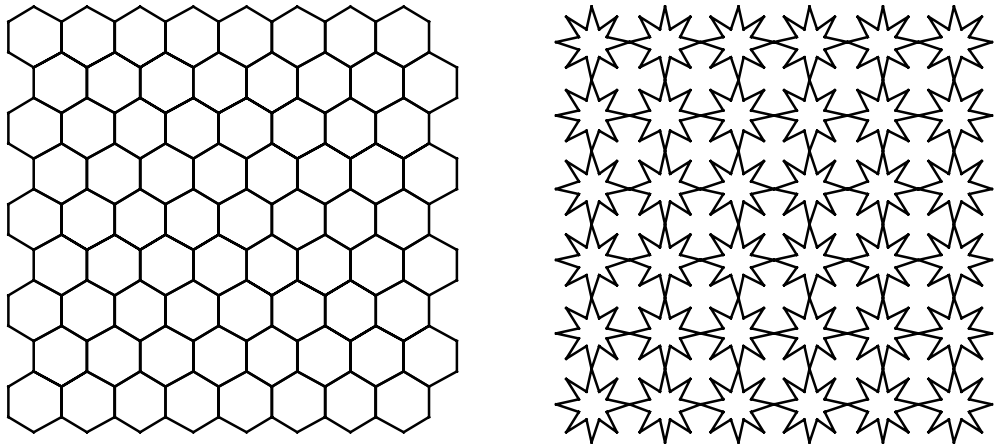


FIG. 20.11 – *Pavage du plan avec des hexagones et des étoiles*

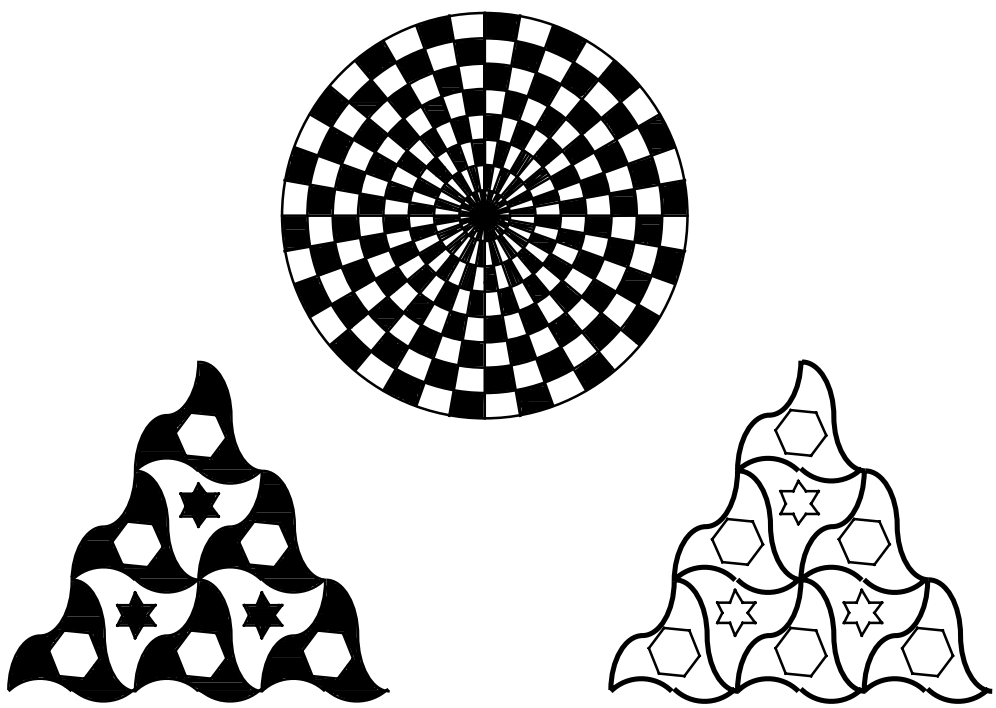


FIG. 20.12 – *Pavage à la Escher du disque et du triangle*

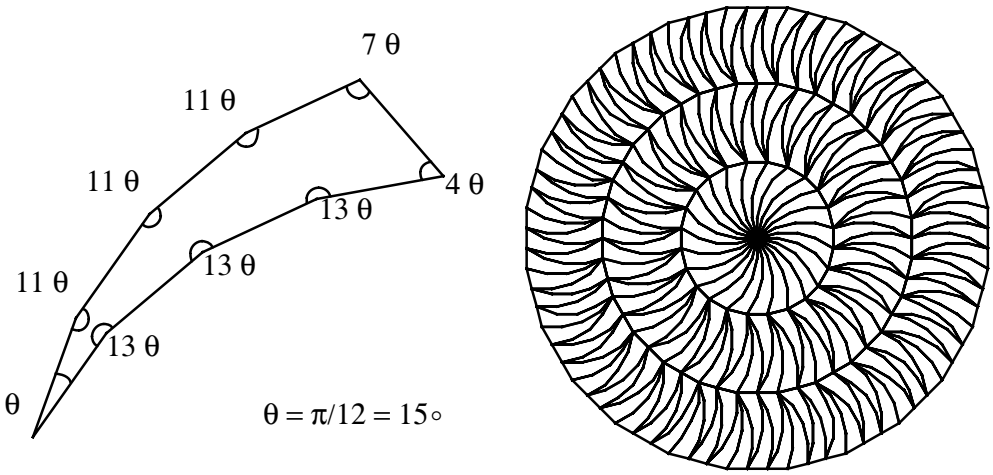


FIG. 20.13 – *Pavé et pavage du disque de Voderberg*

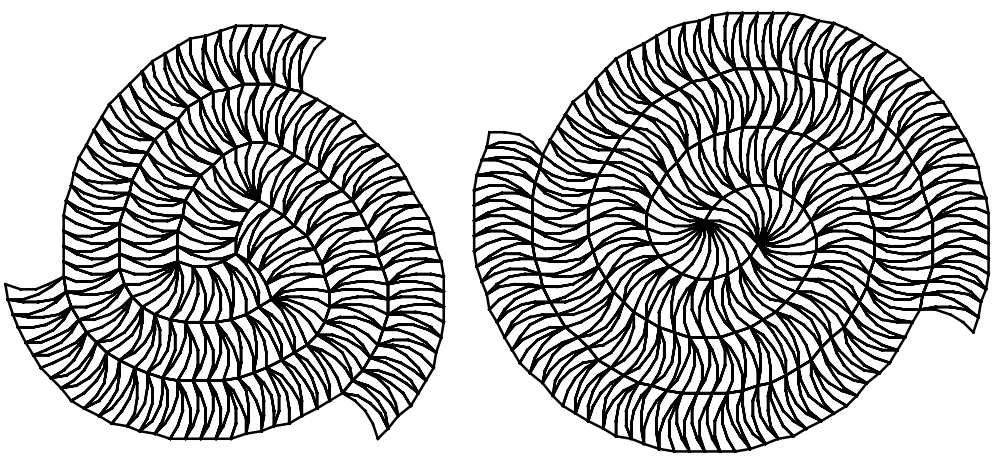


FIG. 20.14 – *Spirales de Voderberg*

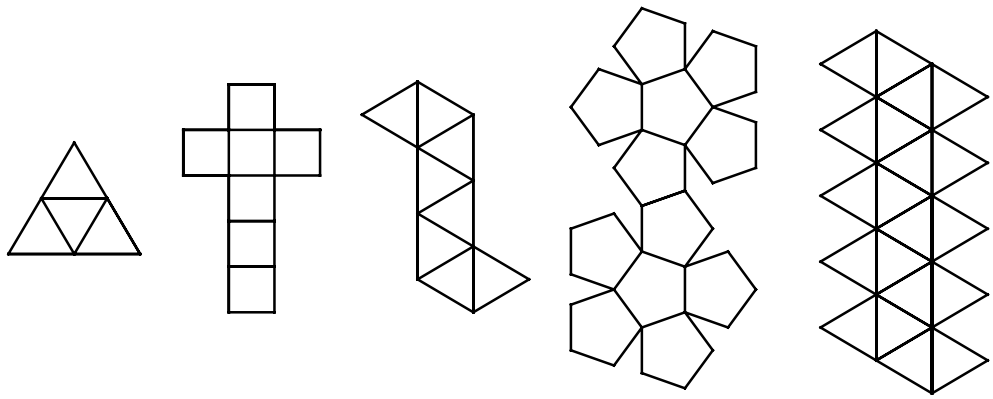


FIG. 20.15 – *Développements des solides de Platon*

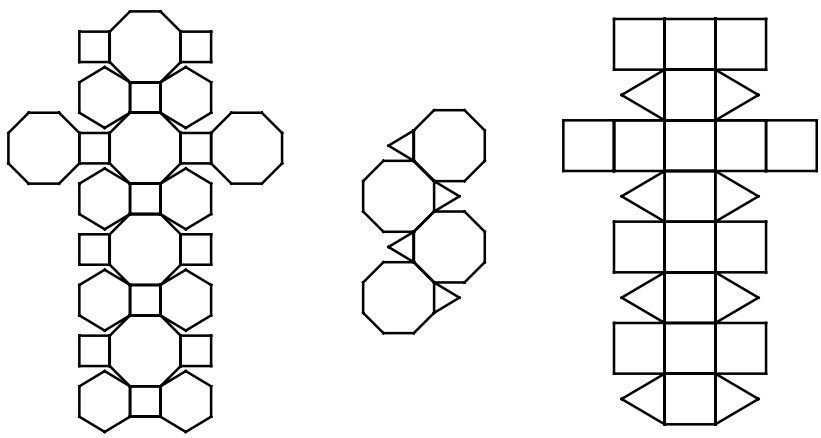


FIG. 20.16 – *Développements des grand et petit rhombicuoctaèdres* et du tétraèdre tronqué

– *les rosaces polygonales de la figure 20.7 (page 230) ;*

– *les "jolygones" de la figure 20.18 (page 237) obtenus en traçant des segments successifs dont la longueur diminue à chaque fois de 5 %, deux segments successifs faisant un angle de θ degrés ; θ étant choisi respectivement égal à 15, 60, 72, 76, 89, 92, 118, 175 et 144 degrés pour les jolygones considérés de gauche à droite et de haut en bas ;*

– *les courbes planes de la figure 20.19 (page 237) définies par une équation polaire, comme suit :*

- *le limaçon de Pascal, $\rho = a + b\cos\theta$ avec $a \neq 0$, $b \neq 0$ et $\theta \in [0,2\pi]$ (sur l'exemple, $a = 40$ et $b = 80$),*
- *le rhodoneæ à trois pétales, $\rho = a\cos n\theta$ avec $a = 60$, $n = 3$ et $\theta \in [0,\pi]$,*
- *la courbe de Moritz, $\rho = a\cos\frac{9}{10}\theta$ avec $a = 60$ et $\theta \in [-10\pi,10\pi]$;*

– *les courbes planes définies par les équations cartésiennes $y = \sin x, x \in [-2\pi,2\pi]$ (figure 20.17 ci-dessous) et $y = \cos x, x \in [-2\pi,2\pi]$ en utilisant les commandes* `deplacer` *et* `tracer`. *La commande* `tracer(X, Y)`, *trace un segment (si le crayon est baissé) de la position courante du Robot au point de coordonnées cartésiennes (X, Y) et déplace le Robot dans cette nouvelle position (X, Y). De nombreux autres exemples de tracés de courbes sont donnés dans [9].*

□

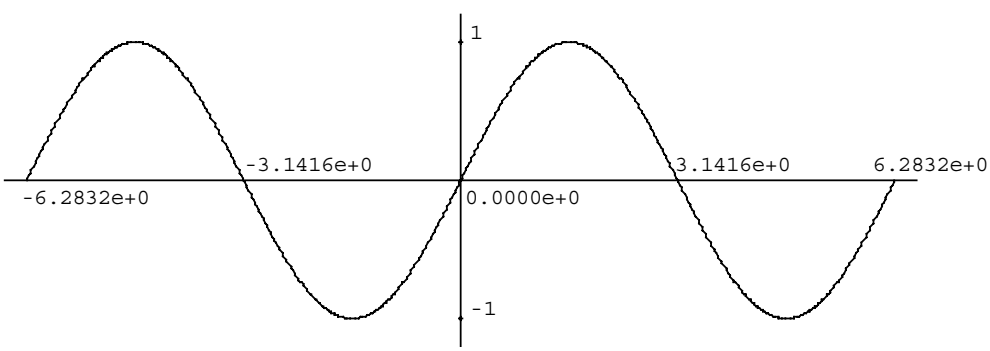


FIG. 20.17 – *Courbe cartésienne $y = \sin x, x \in [-2\pi,2\pi]$*

Corrigé 38 *Nous donnons le programme de dessin de la toile d'araignée de la figure 20.3 (page 229), les autres programmes se trouvant sur disquette :*

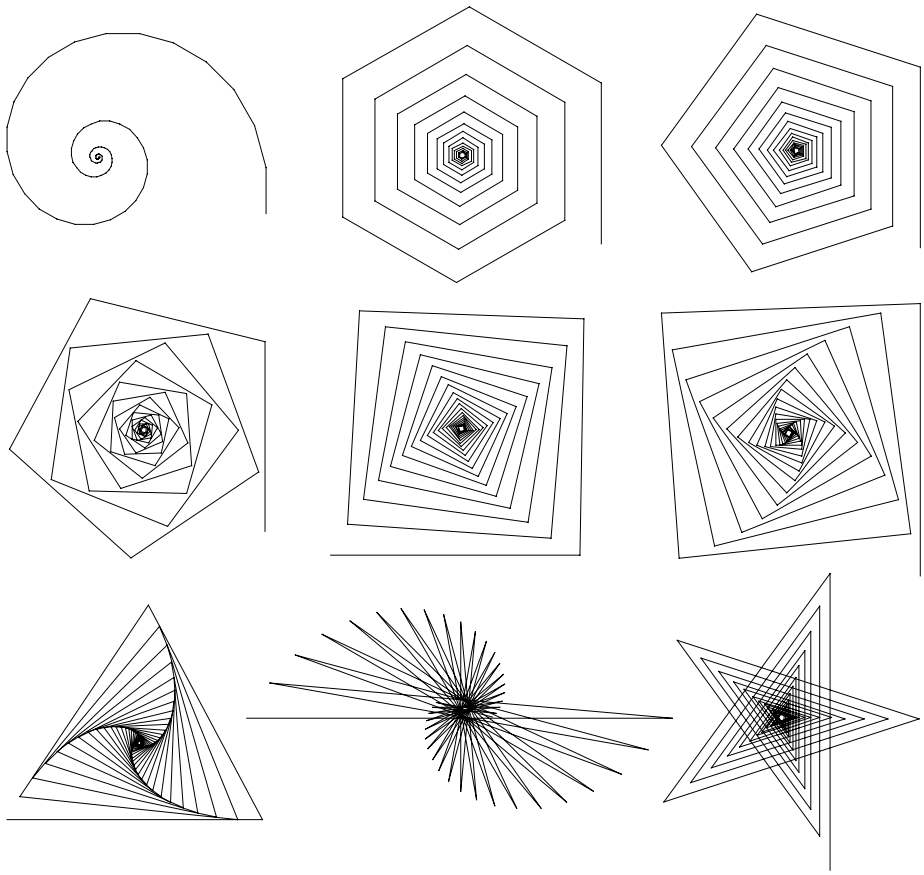


FIG. 20.18 – *Jolygones à 15, 60, 72, 76, 89, 92, 118, 175 et 144 degrés*

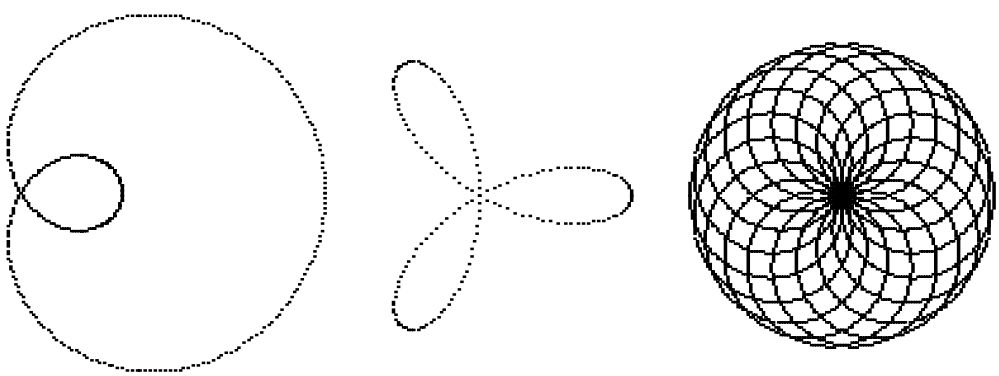


FIG. 20.19 – *Limaçon de Pascal, rhodoneæ à trois pétales, courbe de Moritz*

```
program ToileDaraignee;
  uses   ;

  procedure Hexagone (R : integer);
    var I : integer;
  begin
    lgRT(R, 60);
    ce; pw;
    for I := 1 to 2 do
      begin
        lc; av; av; bc; p3htd; av; phtd; av; av; phtd; av; ce; pe;
      end;
  end;

  procedure Diagonales (R : integer);
    var I : integer;
  begin
    lgRT(R, 60);
    psw;
    for I := 1 to 4 do
      begin
        av; ce; pqtd;
      end;
    ce; pw; avf(2); ce; pe; avf(2);
  end;

  procedure Toile;
    var I : integer;
  begin
    for I := 1 to 8 do
      begin
        Hexagone (5 * I);
        Diagonales(5 * I);
      end;
  end;

begin
  Toile; st;
end.
```

□

Corrigé 39 *Nous donnons la procédure de dessin d'un polygone régulier de n côtés (figure 20.8, page 232), les autres programmes se trouvant sur disquette :*

```
program PolygoneRegulier;
  uses   ;
```

```
const
  TailleQuadrillage = 30;
  NombreDeCotes = 7;

procedure Polygone(N : integer);
  { Dessiner un polygone régulier de  N >= 1 cotés }
    var I : integer;
begin
  for I := 1 to N do
    begin
      av; rt(360 / N);
    end;
end;

begin
  lg(TailleQuadrillage); Polygone(NombreDeCotes); st;
end.
```

□

Corrigé 40 *Nous donnons le programme de dessin du développement du tétraèdre tronqué, les autres programmes se trouvant sur disquette :*

```
program TetraedreTronque;
  uses  ;

  const LgCote = 24;

  procedure FaceEtCoin;
    var I: integer;
  begin
    { Hexagone }
    pqtd; lgRT(LgCote, 60); phtg;
    for I:=1 to 6 do
      begin
        av; rt(60);
      end;
    { Triangle équilatéral }
    pdt; av; pqtd; av;
  end;

  procedure DeveloppementDuTetraedreTronque;
    var I: integer;
  begin
```

```
    for I :=1 to 2 do
      begin
        { Face et coin en bas }
        FaceEtCoin; lc; p3htd; avf(2); phtg; av; p3htd; bc;
        { Face et coin en haut }
        FaceEtCoin; lc; av; phtg; avf(2); pqtg; bc;
      end;
  end;

begin
  DeveloppementDuTetraedreTronque; st;
end.
```

□

Corrigé 41 *Nous donnons le programme de dessin du jolygone à 175 degrés, les autres jolygones sont obtenus en changeant la valeur de la constante* `Angle`. *Les autres programmes se trouvent sur disquette.*

```
program Jolygone175;
  uses  ;
  const
    LongueurInitiale = 350;
    Angle = 175;
    NbrSegments = 40;
  var
    I : integer;
    J : integer;
begin
  ag;
  lg(LongueurInitiale); pqtd; av;
  for I := 1 to (NbrSegments - 1) do
    begin
      rt(Angle); tl(valLgTl * 0.95);
    end;
  st;
end.
```

□

Addenda

La commande `lgRT(R, T)` définit la longueur `R` de la diagonale d'un carreau du quadrillage (exprimée en pixels) et l'angle `T` exprimé en degrés que fait cette diagonale avec une horizontale. Elle est équivalente à :

```
lgX(R * cos((T / 180) * pi)); lgY(R * sin((T / 180) * pi));
```

mais évite l'emploi des fonctions trigonométriques. L'angle `T`, compris entre 0 et 90, définit l'orientation nord-est qui est confondue avec l'est quand `T = 0` et avec le nord quand `T = 90`.

La commande `rt(T)` fait pivoter le $\mathcal{R}^{o}\underset{\smile}{b}^{o}t$ sur place d'un angle de `T` degrés à gauche, dans le sens trigonométrique ou sens inverse des aiguilles d'une montre, si $\texttt{T} > 0$, et à droite, dans le sens des aiguilles d'une montre, si $\texttt{T} < 0$. La commande `tl(L)` fait avancer (si $\texttt{L} > 0$) ou reculer (si $\texttt{L} < 0$) le $\mathcal{R}^{o}\underset{\smile}{b}^{o}t$ tout droit de `L` fois la taille d'un côté d'un pixel de l'écran.

Ces commandes `rt` et `tl` sont directement inspirées du graphisme "tortue" de LOGO. Elles changent le repère et l'orientation du $\mathcal{R}^{o}\underset{\smile}{b}^{o}t$ pour rester compatibles avec l'emploi de coordonnées cartésiennes ou polaires.

Le langage LOGO fut conçu par Seymour Papert à la fin des années 1970 [29]. Il présente de nombreuses similitudes avec LISP [26], en particulier l'interprétation de données représentant des instructions, la portée dynamique des identificateurs et la structure de liste. L'interprétation des données comme des programmes implique quelques subtilités dans l'écriture des programmes (comme l'emploi de la fonction `quote`, [40]) qui ne sont pas à la portée de tous les débutants en informatique. La portée dynamique des identificateurs [40] est largement abandonnée et la structure de liste est presque toujours complétée par des tableaux dans les LISP modernes comme COMMON LISP qui suivent en cela les langages descendant d'ALGOL 60 [27] comme PASCAL ou ADA.

21

Récursivité

En Pascal, une procédure peut appeler une autre procédure précédemment déclarée. Une procédure peut également s'appeler elle-même. On dit alors qu'elle est *récursive*. Les procédures récursives sont utiles pour faire des dessins comme ceux de la figure 21.1 (page 244). Ces dessins récursifs ont la particularité de se retrouver indéfiniment en réduction à l'intérieur d'eux-mêmes (exactement comme l'image infinie obtenue dans deux miroirs se faisant face). Comme en pratique il n'est pas possible de dessiner un trait plus petit qu'un pixel de l'écran, le tracé s'arrête pour les dessins très petits.

21.1 Récursivité simple

La structure des dessins récursifs simples se comprend par *récurrence* sur la taille du dessin. Il faut d'abord comprendre la *base*, c'est-à-dire comment faire un dessin de taille 0 (ou 1). Ensuite il faut comprendre l'*induction*, c'est-à-dire comment construire un dessin de taille n en utilisant des dessins identiques de tailles $n-1$, $n-2$, ...strictement inférieures à n.

Prenons l'exemple des pavés récursifs de la figure 21.2 (page 245). Un pavé de taille 0 est réduit à un point (plus précisément un pixel de l'écran). Un pavé de taille n est constitué d'un pavé de taille $n-1$ entouré d'un carré de côté n fois la taille du quadrillage, comme expliqué à la figure 21.3 (page 245). La procédure récursive `Pave` de dessin d'un pavé récursif reçoit en paramètre la taille `N` du pavé. Si `N` est nul, elle dessine un point. Si `N` est strictement positif alors elle dessine un carré de côté `N` puis s'appelle récursivement pour

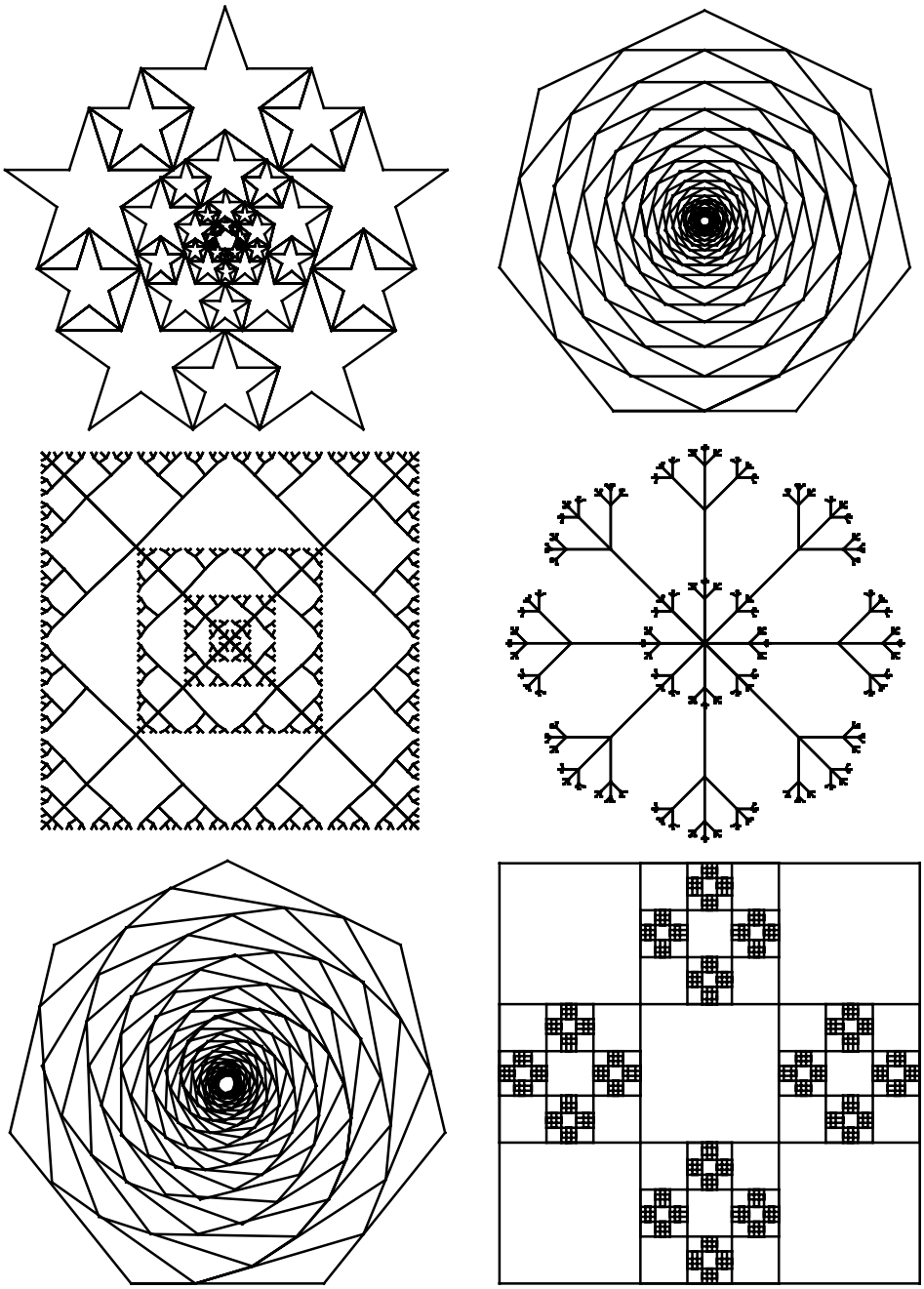


FIG. 21.1 – *Dessins récursifs*

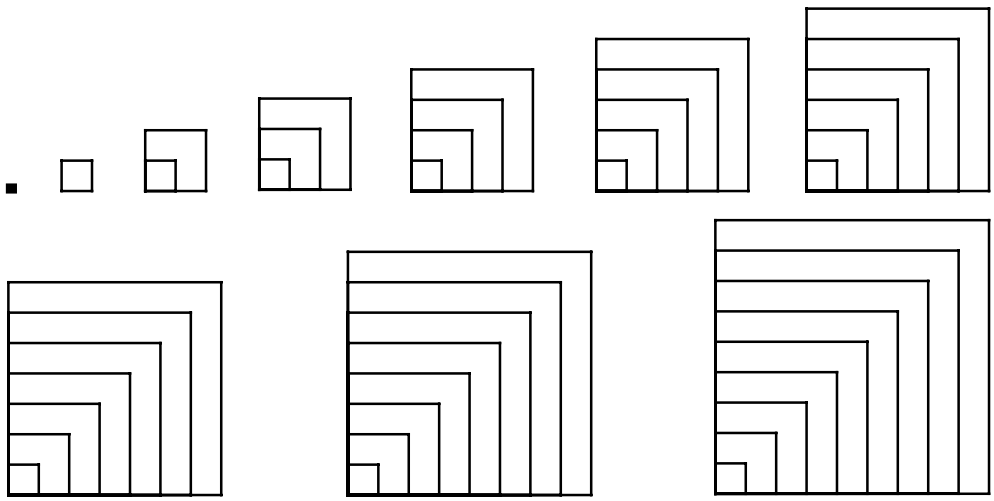


FIG. 21.2 – *Pavés récursifs*

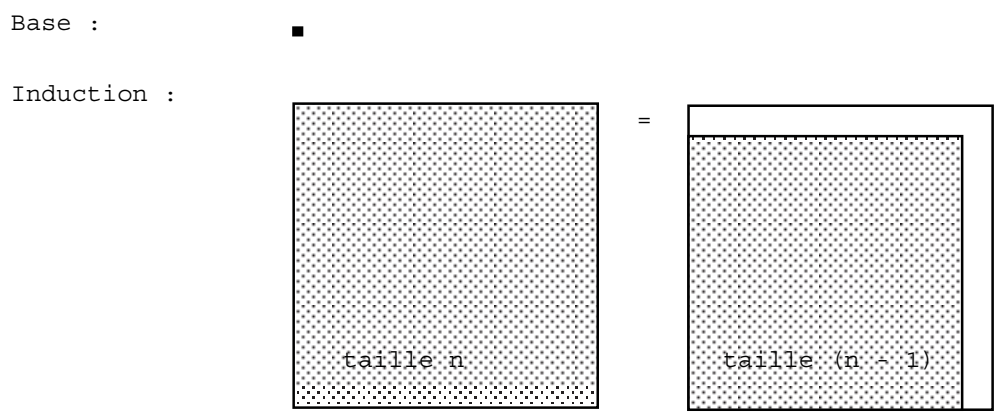


FIG. 21.3 – *Règle de construction récurrente des pavés récursifs*

dessiner un pavé de taille `N − 1`.

```
program PavesRecursifs;
 uses  ;

 procedure Pave(N : integer);
 begin
  if N = 0 then
   begin
    { Dessiner un carré de côté 0 }
    dp;
   end
  else
   begin
    { Dessiner un carré de côté N }
    avf(N); pqtd; avf(N); pqtd; avf(N); pqtd; avf(N); pqtd;
    { Paver l'intérieur }
    Pave(N − 1);
   end;
 end;

begin
 eb; ag; Pave(10); st;
end.
```

Le dessin des pavés récursifs peut se faire *itérativement*, c'est-à-dire en utilisant une boucle "for" ou "while". Ce serait plus difficile avec des dessins comme le tableau d'un tournoi de tennis représenté à la figure 21.4 ci-dessous.

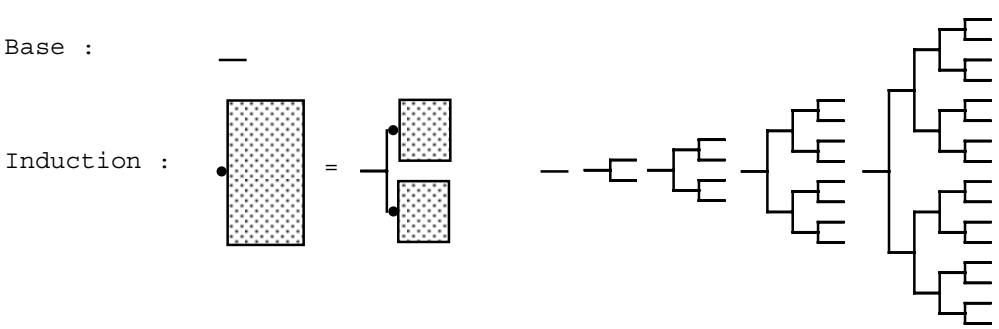
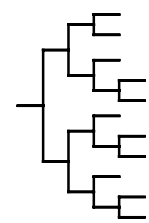


FIG. 21.4 – *Règle de construction récurrente d'un tableau* de tournoi de tennis

Un tableau de tournoi de tennis ne comprenant qu'un seul inscrit comporte un trait horizontal prévu pour écrire le nom de cet inscrit qui est également le vainqueur. Un tableau de tournoi de tennis comprenant `N` inscrits s'obtient en traçant un trait horizontal prévu pour écrire le nom du vainqueur en face de la pointe d'une accolade reliant les sous-tableaux servant à désigner les finalistes. Chacun des sous-tableaux contient à peu près le même nombre de joueurs c'est-à-dire respectivement `N` **div** `2` et `N` − (`N` **div** `2`) joueurs. La procédure récursive de dessin du tableau d'un tournoi de `N` joueurs est donnée ci-dessous et illustrée dans le cas `N = 11`.



```
program TableauDeTournoi;
  uses  ;

  const
    L = 5; { Longueur d'un trait horizontal }

  procedure Tableau(N : integer);
    { Dessiner un tableau pour un tournoi de N }
    { joueurs                                  }
  begin
    { Pointe de l'accolade, pour le vainqueur }
    avf(L);
    if (N > 1) then
      begin
        { Branche supérieure de l'accolade }
        pqtg; avf(N); pqtd;
        { Sous-tableau supérieur, contenant la }
        { moitié des joueurs                   }
        Tableau(N div 2);
        { Revenir au centre de l'accolade }
        lc; pqtg; avf(−N); pqtd; bc;
        { Branche inférieure de l'accolade }
        pqtd; avf(N); pqtg;
        { Sous-tableau inférieur, contenant le }
        { reste des joueurs                    }
        Tableau(N − (N div 2));
        { Revenir au centre de l'accolade }
        lc; pqtd; avf(−N); pqtg; bc;
      end;
    { Revenir à l'extrémité de la pointe }
    avf(−L);
  end;

begin
```

```
  lg(3); pe; Tableau(16); st;
end.
```

21.2 Invariants

Quand on écrit une procédure récursive il est important de réfléchir à l'*invariant* de cette procédure c'est-à-dire aux hypothèses que doivent satisfaire les paramètres effectifs de la procédure et le Robot pour que le dessin soit correct. Prenons l'exemple d'un arbre binaire (voir la figure 21.5 ci-dessous), une structure de base de la programmation, que nous avons déjà rencontrée sous la forme du tableau de tournoi de tennis.

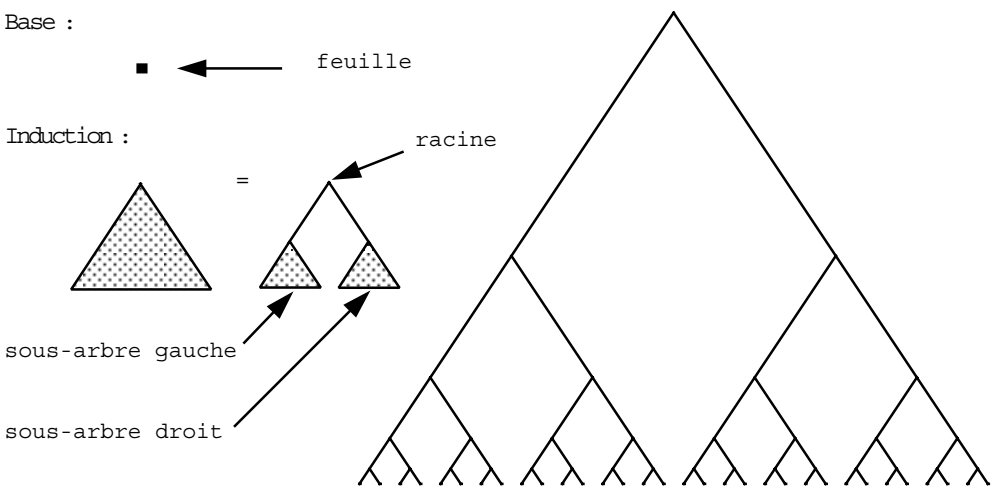


FIG. 21.5 – *Règle de construction récurrente d'un arbre binaire*

La procédure de dessin d'un arbre binaire est la suivante :

```
program ArbreBinaire;
  uses  ;

  procedure Arbre(L : integer);
    { Dessiner un arbre à L feuilles, la racine en haut, les feuilles }
    { en bas, en supposant que L >= 1,  l'arbre  étant  réduit  à  un }
    { point quand L = 1. Avant et après le dessin, le robot doit être }
    { placé  à  la racine de l'arbre, orienté au nord avec son crayon }
    { baissé.                                                         }
  begin
```

```
    if (L = 1) then
      begin
        { Dessiner  un  arbre  réduit  à un seul point (sans changer la }
        { position et l'orientation du robot).                          }
        dp;
      end
    else { L > 1 }
      begin
        { Le robot étant placé à la racine de l'arbre,  orienté au nord }
        { avec son crayon baissé, dessiner la branche sud-ouest.        }
        p3htg; avf(L);
        { Orienter le robot au nord.                                    }
        p3htd;
        { Dessiner le sous-arbre gauche ayant  (L div 2) >= 1 feuilles, }
        { le  robot  étant  placé à la racine du sous-arbre, orienté au }
        { nord avec son crayon baissé.                                  }
        Arbre(L div 2);
        { Sachant que le robot est placé à la racine du sous-arbre gau- }
        { che, orienté au nord et crayon baissé, revenir à la racine de }
        { l'arbre, sans rien dessiner.                                  }
        lc; phtd; avf(L);
        { Le robot étant placé à la racine de l'arbre, orienté au nord- }
        { est avec son crayon levé, dessiner la branche sud-est.        }
        bc; pqtd; avf(L);
        { Orienter le robot au nord.                                    }
        p3htg;
        { Dessiner le  sous-arbre  droit  ayant  (L - (L div 2))  >=  1 }
        { feuilles,  le  robot  étant  placé à la racine du sous-arbre, }
        { orienté au nord avec son crayon baissé.                       }
        Arbre(L - (L div 2));
        { Placer le robot à la racine de l'arbre,  orienté au nord avec }
        { son crayon baissé.                                            }
        lc; phtg; avf(L); phtd; bc;
      end;
  end;

begin
  lgX(2); lgY(3); eh; Arbre(32); st;
end.
```

Le premier invariant est que le paramètre `L` de la procédure `Arbre` doit être supérieur ou égal à 1. C'est vrai pour l'appel principal puisque $\texttt{L} = 32$. Un appel quelconque ne fait des appels supplémentaires que si $\texttt{L} > 1$. Comme $\texttt{L} \geq 2$, on a $(\texttt{L}\ \textbf{div}\ 2) \geq 1$ et l'invariant est vrai quand on appelle récursivement la procédure pour dessiner le sous-arbre gauche. Comme $\texttt{L} = (2 * (\texttt{L}\ \textbf{div}\ 2))$ si

`L` est pair et `L` $= (2*($`L` **div** $2))+1$ si `L` est impair, on a `L` $\geq (2*($`L` **div** $2))$ donc (`L` $-($`L` **div** $2)) \geq ($`L` **div** $2)$ de sorte que (`L` $-($`L` **div** $2)) \geq 1$ puisque (`L` **div** $2) \geq 1$. L'invariant est donc également vrai quand on appelle récursivement la procédure `Arbre` pour dessiner le sous-arbre droit. Comme l'invariant est vrai pour l'appel principal et qu'il reste vrai pour tous les appels suivants, il est toujours vrai.

On vérifierait de la même façon l'invariant concernant le $\mathcal{R}^{o}_{\smile}{}^{o}t$, à savoir qu'avant et après le dessin de l'arbre binaire, le $\mathcal{R}^{o}_{\smile}{}^{o}t$ se trouve placé à la racine de l'arbre, orienté vers le nord avec son crayon baissé.

21.3 Non-terminaison

Quand on utilise des procédures récursives, il peut arriver qu'elles s'appellent les unes les autres sans fin, exactement comme dans le cas de boucles "while" dont l'exécution ne se termine pas. L'exemple le plus simple est celui de la procédure `Boucler` ci-dessous.

```
program NonTerminaison;
  uses  ;

  procedure Boucler;
  begin
    Boucler;
  end;

begin
  Boucler; st;
end.
```

Un appel de `Boucler` provoque immédiatement un nouvel appel de `Boucler`. Comme à chaque appel, l'ordinateur utilise un peu de sa mémoire, il finit par se produire une erreur quand il n'y a plus suffisamment de mémoire disponible pour faire l'appel suivant. Cette erreur est signalée par un message du genre "Stack overflow error" ("Débordement de la pile") avant l'arrêt de l'exécution du programme. En général l'erreur est moins flagrante et ne se produit que pour certaines valeurs des paramètres.

21.4 Récursivité mutuelle

Considérons les serpentines de la figure 21.6 (page 251). Il y en a de deux sortes. Les serpentines $\mathtt{E}_n$ ont une hauteur n et se dessinent à partir de leur

coin inférieur ouest, en traçant un trait horizontal d'ouest en est puis un trait vertical pour passer à l'est de l'étage supérieur. Les serpentines W_n ont une hauteur n et se dessinent à partir de leur coin inférieur est, en traçant un trait horizontal d'est en ouest puis un trait vertical pour passer à l'ouest de l'étage supérieur.

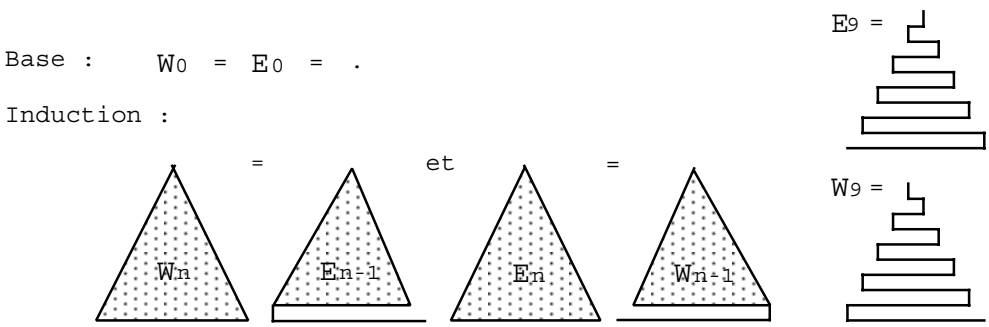


FIG. 21.6 – *Règle de construction récurrente d'une serpentine*

Les procédures `SerpentineE` et `SerpentineW` dans le programme ci-dessous dessinent respectivement une serpentine E_n et W_n, leur paramètre formel `N` ayant pour valeur effective la hauteur n de la serpentine à dessiner.

```
program SerpentineRecursive;
  uses  ;

  procedure SerpentineW (N : integer);
  forward;

  procedure SerpentineE (N : integer);
    { Construire une serpentine de hauteur N en partant de son coin }
    { inférieur ouest.                                               }
  begin
    if N > 0 then
      begin
        { Construire la base de largeur N, d'ouest en est. }
        pe; avf(N); pn; av;
        { Construire la serpentine supérieure de hauteur  (N - 1)  en }
        { partant de son coin inférieur est.                         }
        SerpentineW(N − 1);
      end;
  end;
```

```
procedure SerpentineW;
  { Construire une serpentine de hauteur N en partant de son coin }
  { inférieur est.                                                 }
begin
  if N > 0 then
    begin
      { Construire la base de largeur N, d'est en ouest. }
      pw; avf(N); pn; av;
      { Construire la serpentine supérieure de hauteur  (N - 1)  en }
      { partant de son coin inférieur ouest.                       }
      SerpentineE(N - 1);
    end;
end;

begin
  lg(6); eb; ad; SerpentineW(18); eb; ag; SerpentineE(18); st;
end.
```

Une serpentine $\mathtt{E}_0$ ou $\mathtt{W}_0$ de hauteur nulle étant vide, les procédures `SerpentineE` et `SerpentineW` ne font rien quand la valeur n de leur paramètre `N` est négative ou nulle. Pour dessiner une serpentine $\mathtt{E}_n$ ou $\mathtt{W}_n$ dont la hauteur $n \geq 1$ est la valeur effective de leur paramètre formel `N`, les procédures `SerpentineE` et `SerpentineW` dessinent la base puis au-dessus une serpentine $\mathtt{W}_{n-1}$ ou $\mathtt{E}_{n-1}$ dont la hauteur est la valeur de (`N` -1). Comme la procédure `SerpentineE` appelle la procédure `SerpentineW` et que la procédure `SerpentineW` appelle la procédure `SerpentineE`, on dit que ces procédures sont *mutuellement récursives*.

Quand on utilise des procédures mutuellement récursives en Pascal, il faut respecter la règle que *tout identificateur doit être déclaré avant d'être utilisé*. C'est pourquoi on commence par écrire l'entête de l'une des procédures suivie de l'identificateur `forward` (prononcer "forouarde") qui signifie "en avant" et indique que le corps de la procédure sera défini plus tard, puis on déclare normalement la seconde procédure avec son corps et enfin on définit le corps de la première procédure à l'aide d'une déclaration de procédure *où les paramètres formels sont omis*.

De manière générale, les déclarations de procédures mutuellement récursives ont la forme suivante :

Déclaration de procédures mutuellement récursives :

procedure ␣ NomDeLaProcedure$_1$ [a] (ParamFormel$_1^1$ [b] : type$_1^1$ [c];
ParamFormel$_1^2$: type$_1^2$);
forward;

procedure ␣ NomDeLaProcedure$_2$ (ParamFormel$_2^1$: type$_2^1$;
ParamFormel$_2^2$: type$_2^2$);
begin
... Instructions du corps de la procédure [d], [e] (séparées par des ';') ...
end;

procedure ␣ NomDeLaProcedure$_1$;
begin
... Instructions du corps de la procédure [f], [g] (séparées par des ';') ...
end;

[a] Un nom de procédure est un identificateur.
[b] Un paramètre formel est un identificateur.
[c] Un type de paramètre est `boolean`, `integer`, `real`, `string`, ...
[d] Y compris les appels des procédures NomDeLaProcedure$_1$ et NomDeLaProcedure$_2$ précédemment déclarées.
[e] Les expressions figurant dans ces instructions peuvent utiliser les paramètres formels ParamFormel$_2^1$, ParamFormel$_2^2$, ... de la procédure NomDeLaProcedure$_2$.
[f] Y compris les appels des procédures NomDeLaProcedure$_1$ et NomDeLaProcedure$_2$ précédemment déclarées.
[g] Les expressions figurant dans ces instructions peuvent utiliser les paramètres formels ParamFormel$_1^1$, ParamFormel$_1^2$, ... de la procédure NomDeLaProcedure$_1$.

Exercice 42 *L'exercice consiste à dessiner des* fractales *[2] [25], c'est-à-dire des dessins que l'on retrouve à l'infini, en réduction à l'intérieur d'eux-mêmes.*

␣ *Écrire une procédure* Pascal *récursive pour construire une courbe de Von Koch*[1]*, comme indiqué à la de la figure 21.7 (page 255). La courbe K_0 est un segment de droite. La courbe K_1 est obtenue en divisant le segment K_0 en trois segments égaux et en remplaçant celui du milieu par un triangle équilatéral sans base. La courbe K_n est obtenue de même en divisant en trois chacun des segments de la courbe K_{n-1}. Écrire un programme de dessin d'un*

1. Von Koch, mathématicien suédois, 1870–1924.

flocon de Von Koch, obtenu en fragmentant ainsi vers l'extérieur les côtés d'un triangle équilatéral. Faire de même pour un anti-flocon de Von Koch pour lequel la fragmentation se fait vers l'intérieur du triangle équilatéral initial. Utiliser diverses fragmentations d'un segment pour obtenir d'autres types de flocons comme ceux de la figure 21.10 (page 256).

_ *Programmer les dessins des motifs récursifs de la figure 21.1 (page 244).*

_ *Écrire un programme* Pascal *de dessin des carrés (figure 21.11 page 257), triangles (figure 21.12 page 258) et pentagones (figure 21.13 page 259) de Sierpinski*[2].

_ *Dessiner les végétaux de la figure 21.14 (page 260).*

_ *Dessiner les carrés de Sedgewick [35] de la figure 21.15 (page 261).*

_ *Dessiner la courbe du dragon de la figure 21.16 (page 262) selon les règles de construction de la figure 21.8 (page 255). La courbe du dragon d'ordre n est celle que l'on observe en regardant par la tranche une feuille de papier qui a été repliée n fois sur elle-même puis dépliée à angles droits.*

_ *Dessiner la courbe de Hilbert*[3] *de la figure 21.17 (page 263) selon les règles de construction de la figure 21.9 (page 255).*

_ *Dessiner la courbe de Peano*[4] *de la figure 21.19 (page 265) selon les règles de construction de la figure 21.18 (page 264).*

_ *Écrire une procédure* Pascal *qui écrit en base 2 un nombre passé en paramètre en remarquant que l'écriture binaire $\beta(n)$ d'un nombre n est définie par :*

- $\beta(-n) = -\beta(n)$ *si* $-n < 0$;
- $\beta(0) = 0$;
- $\beta(1) = 1$;
- $\beta(n) = \beta(n \texttt{ div } 2)\beta(n \texttt{ mod } 2)$ *si* $n > 1$.

Faire de même en bases 3, 8 et 16 (en utilisant les digits `A`, `B`, `C`, `D`, `E`, `F` *pour dénoter respectivement les nombres 10, 11, 12, 13, 14, 15 et "*`ecrire('d');translater(1,0);`*" pour écrire le digit* `d`*).*

□

2. Waclaw Sierpinski, mathématicien polonais, 1882–1969.
3. David Hilbert, mathématicien allemand, 1862–1943.
4. Giuseppe Peano, mathématicien italien, 1858–1932.

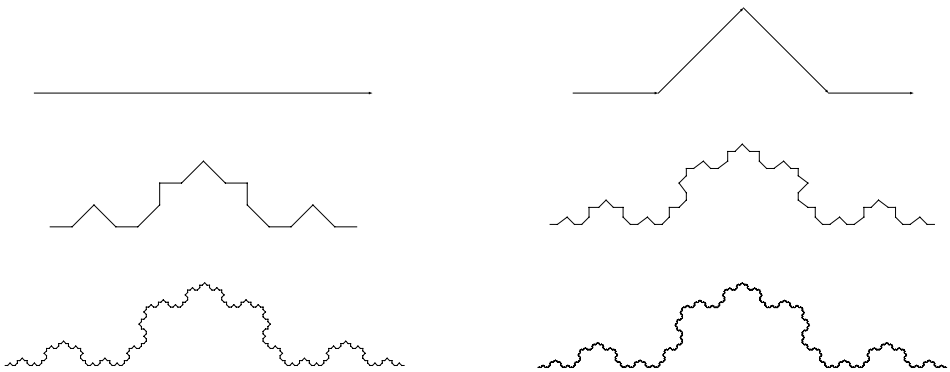


FIG. 21.7 – *Courbe de Von Koch*

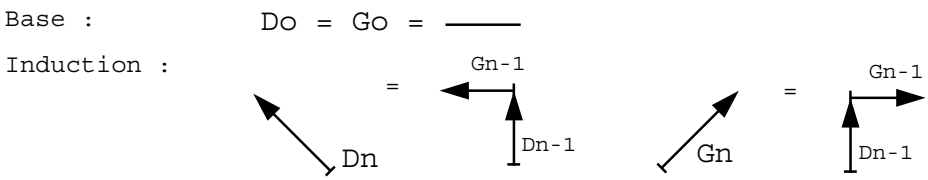


FIG. 21.8 – *Règles de construction de la courbe du dragon*

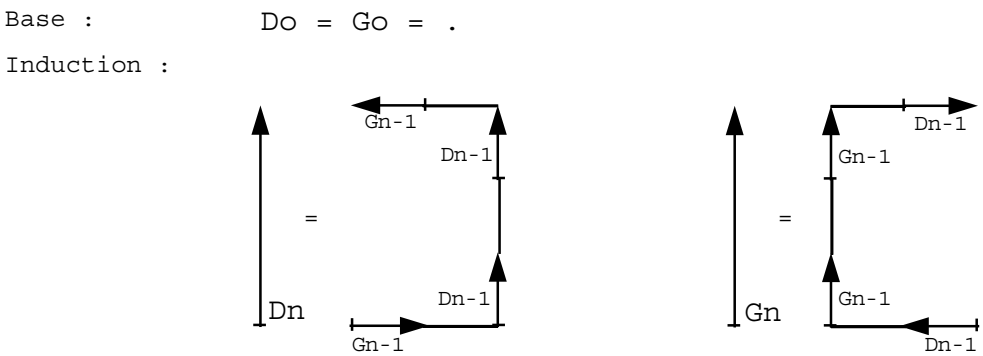


FIG. 21.9 – *Règles de construction des courbes de Hilbert*

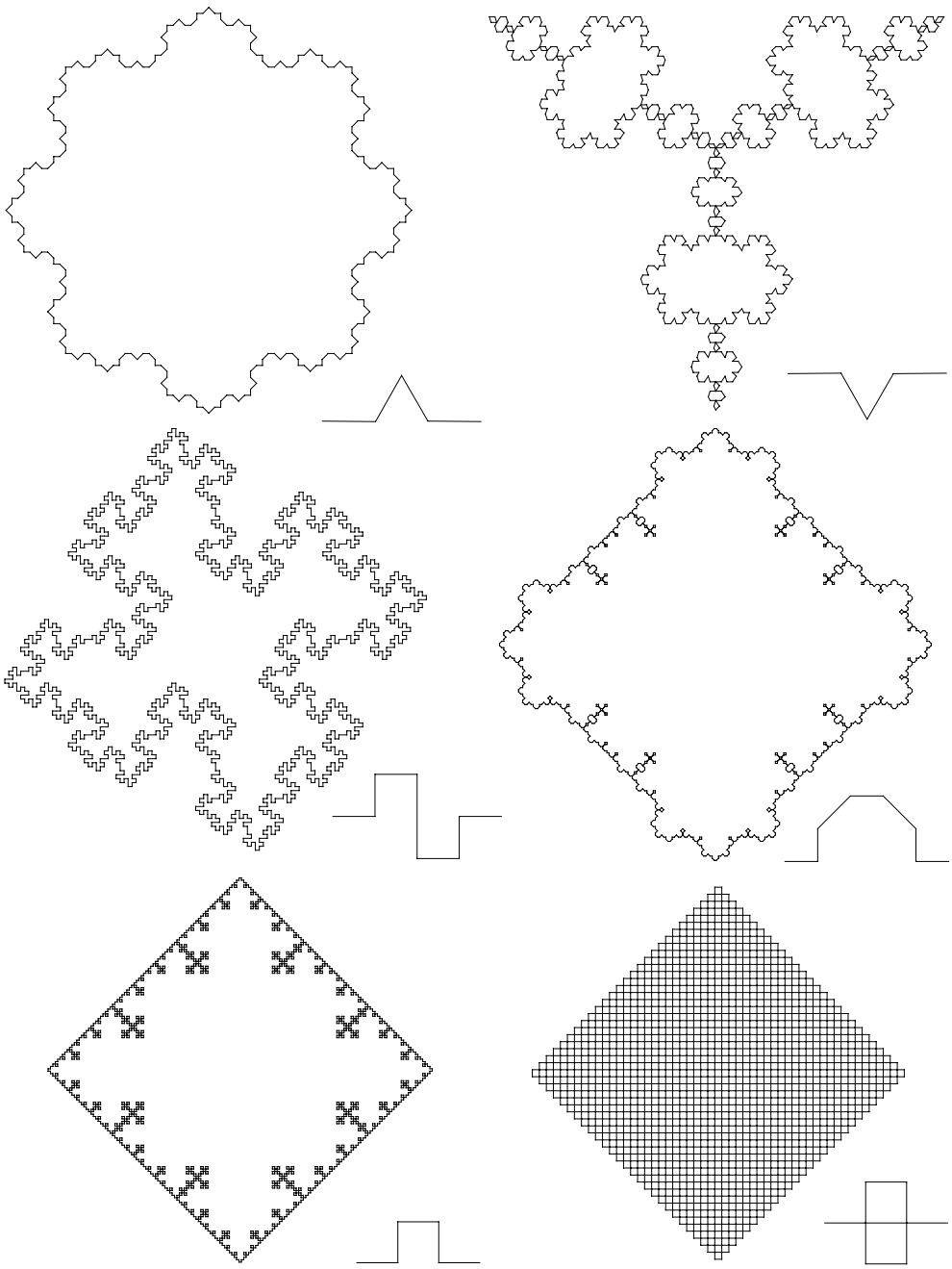


FIG. 21.10 – *Flocons de Von Koch*

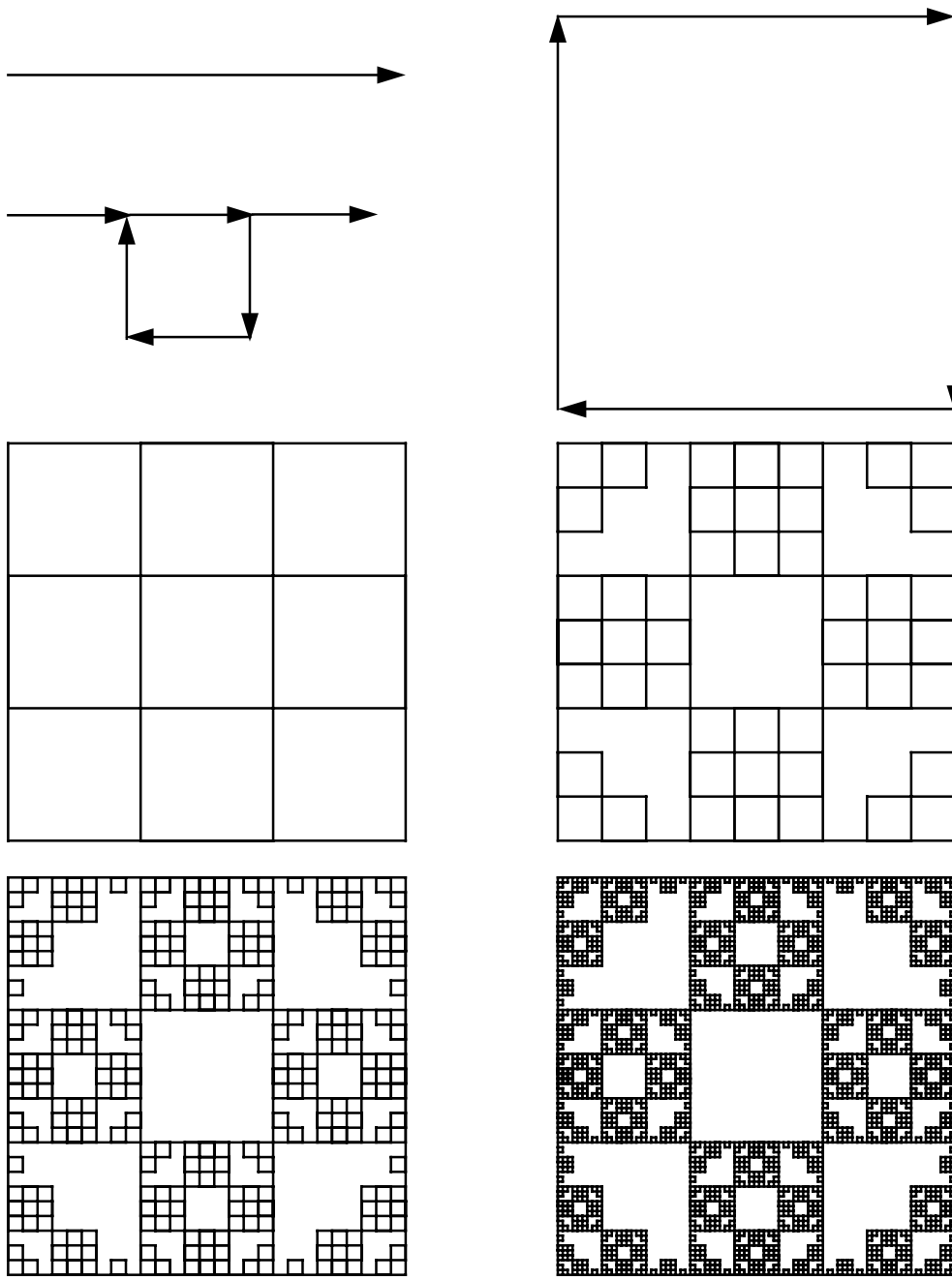


FIG. 21.11 – *Carrés de Sierpinski*

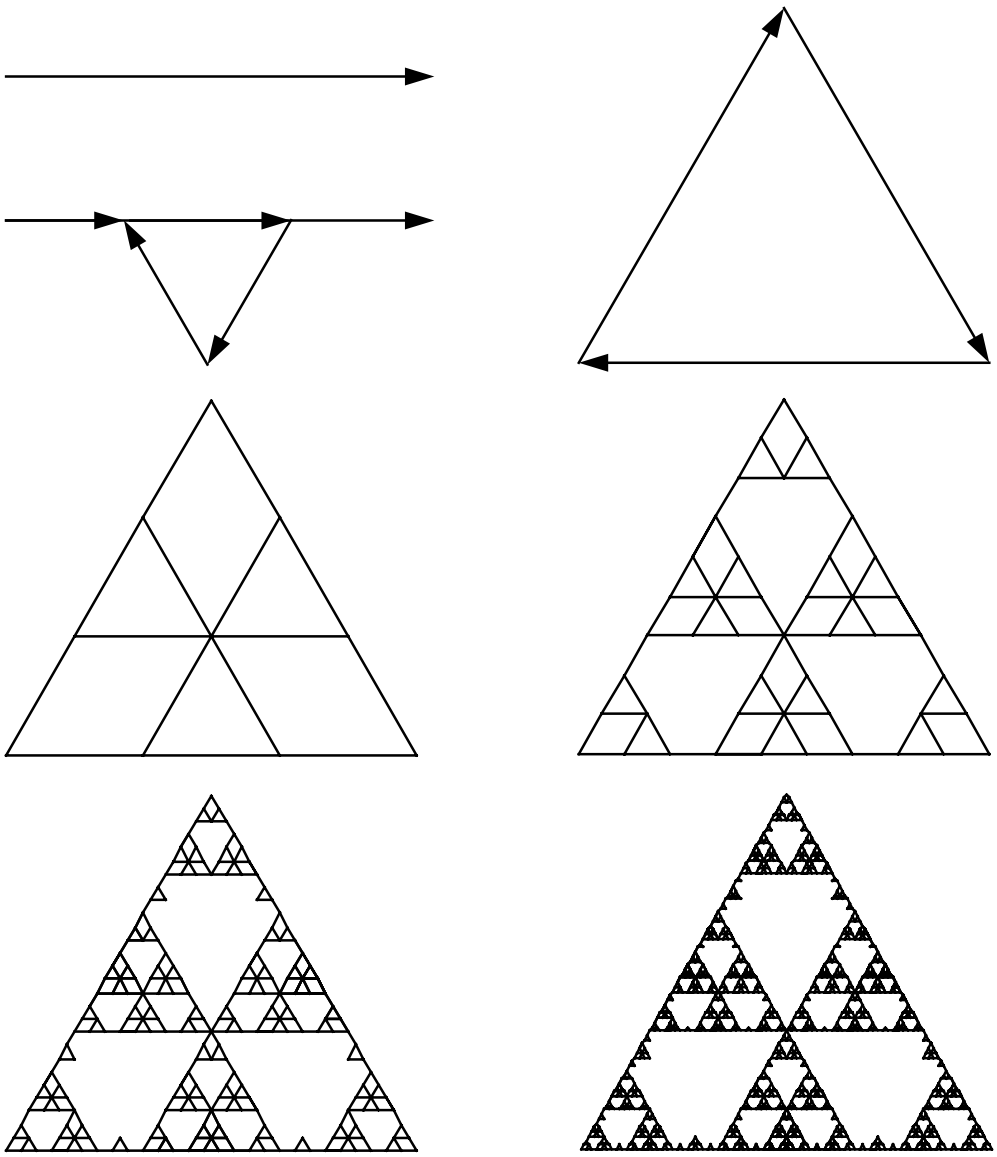


FIG. 21.12 – *Triangles de Sierpinski*

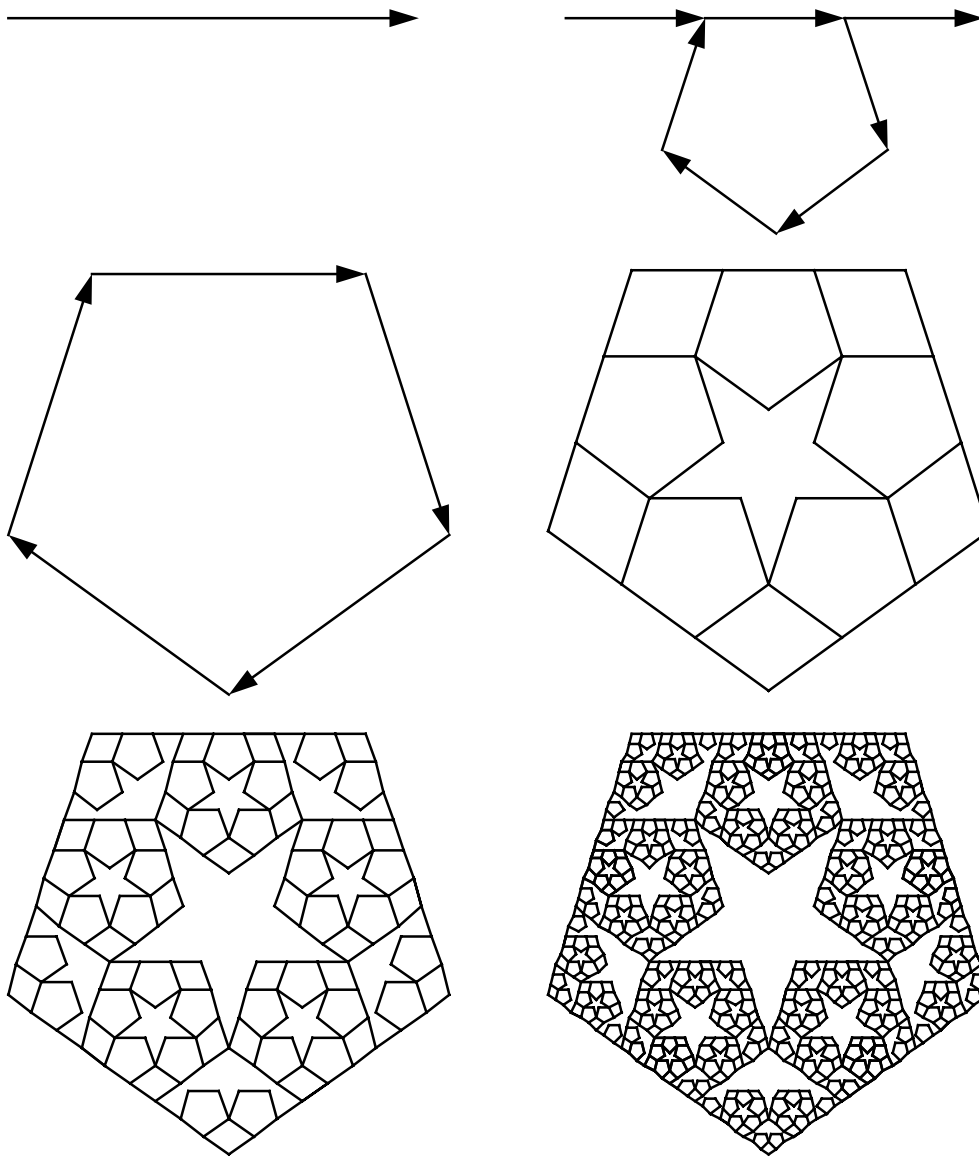


FIG. 21.13 – *Pentagones de Sierpinski*

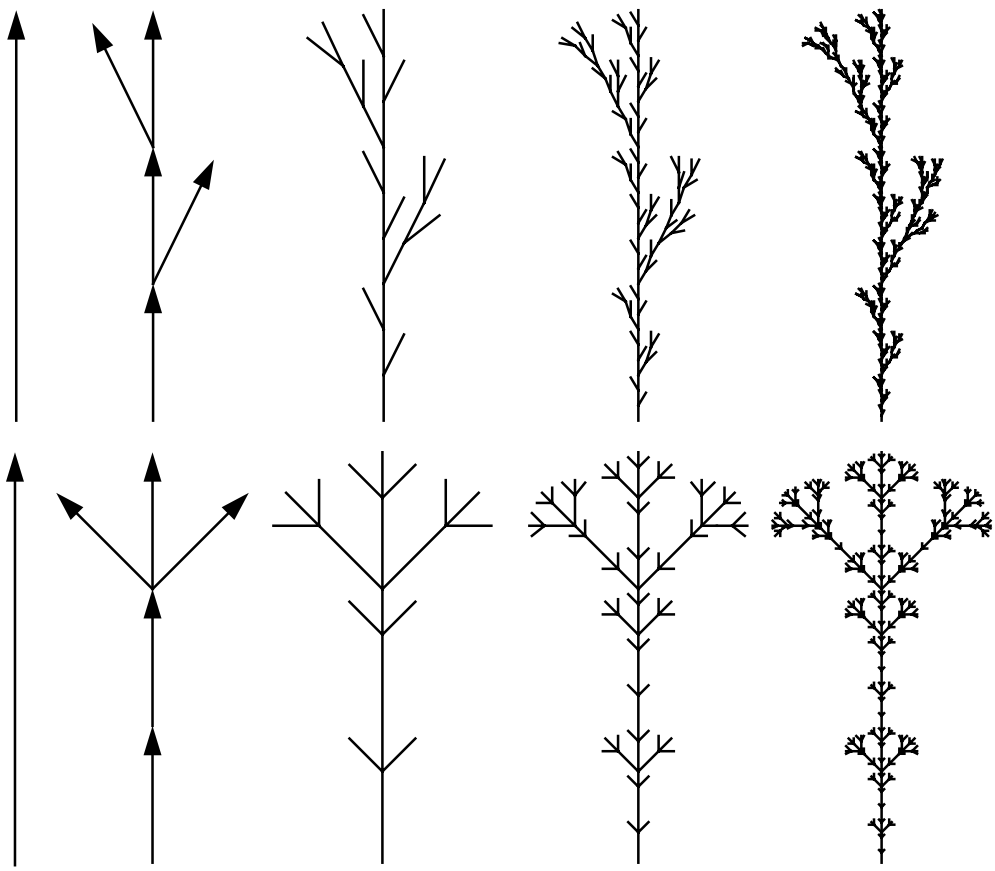


FIG. 21.14 – *Végétaux fractals*

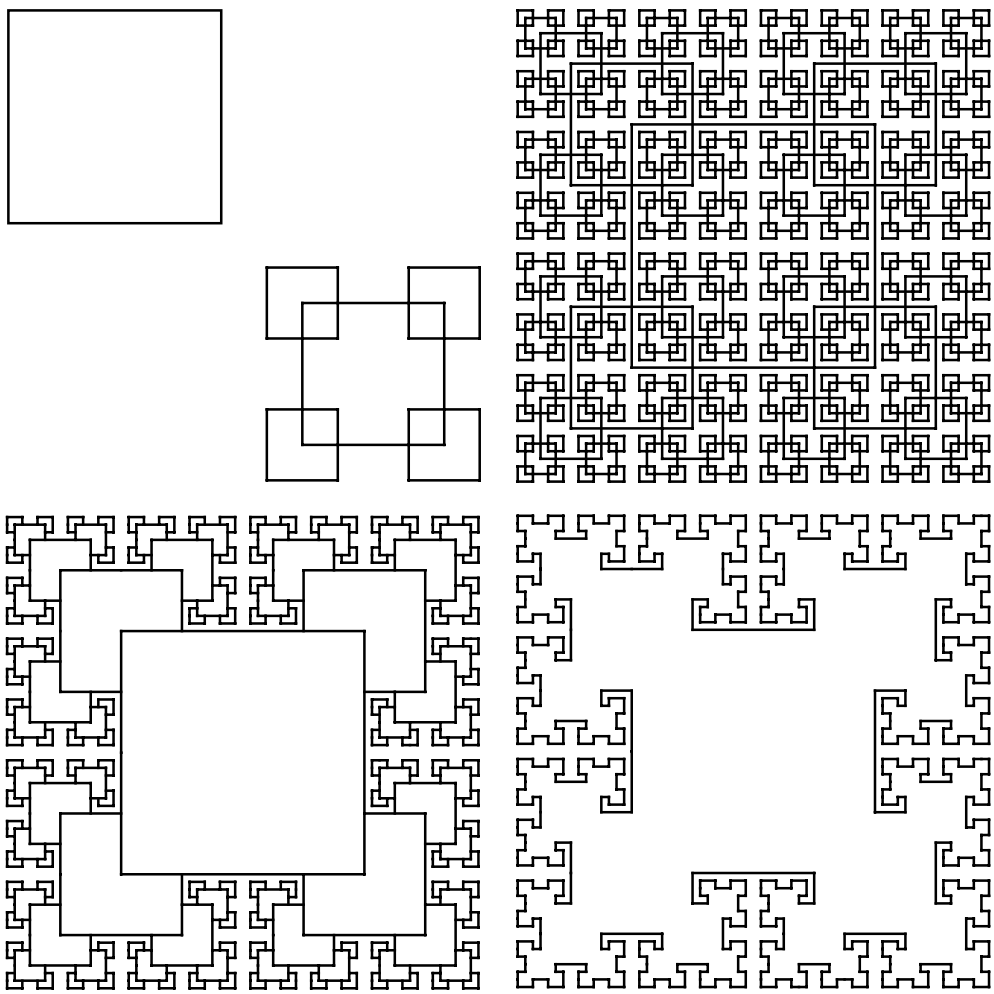


FIG. 21.15 – *Étoile fractale de Sedgewick*

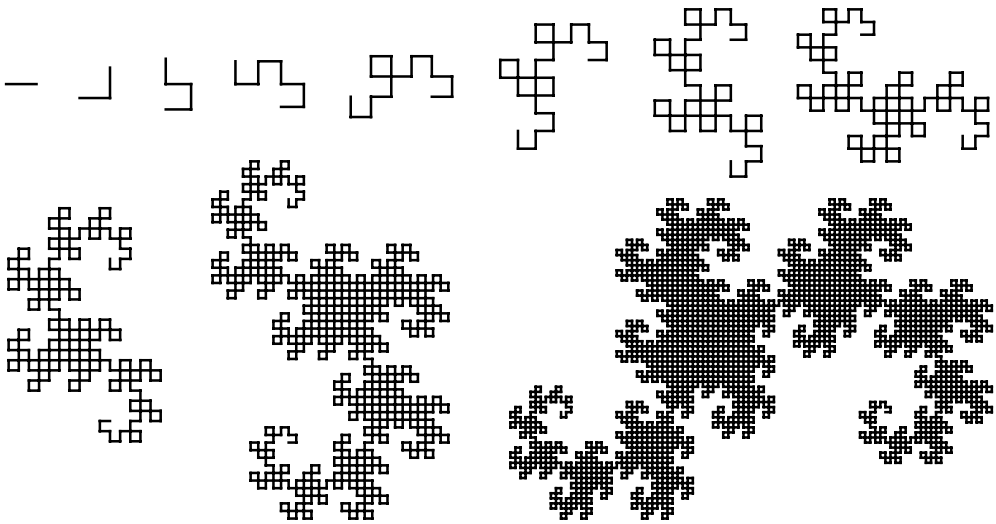


FIG. 21.16 – *Courbes du dragon d'ordre 1, 2, ..., 8, 10 et 12*

Corrigé 42 *Nous donnons deux programmes, les autres se trouvant sur disquette :*

```
program SegmentDeVonKoch;
  uses  ;

  procedure Segment(N : integer);
  begin
    if N = 0 then
      begin av; end
    else
      begin
        Segment(N - 1);
        phtg;
        Segment(N - 1);
        pqtd;
        Segment(N - 1);
        phtg;
        Segment(N - 1);
      end;
  end;

begin
   lg(6); ag; pe; Segment(3); st;
end.
```

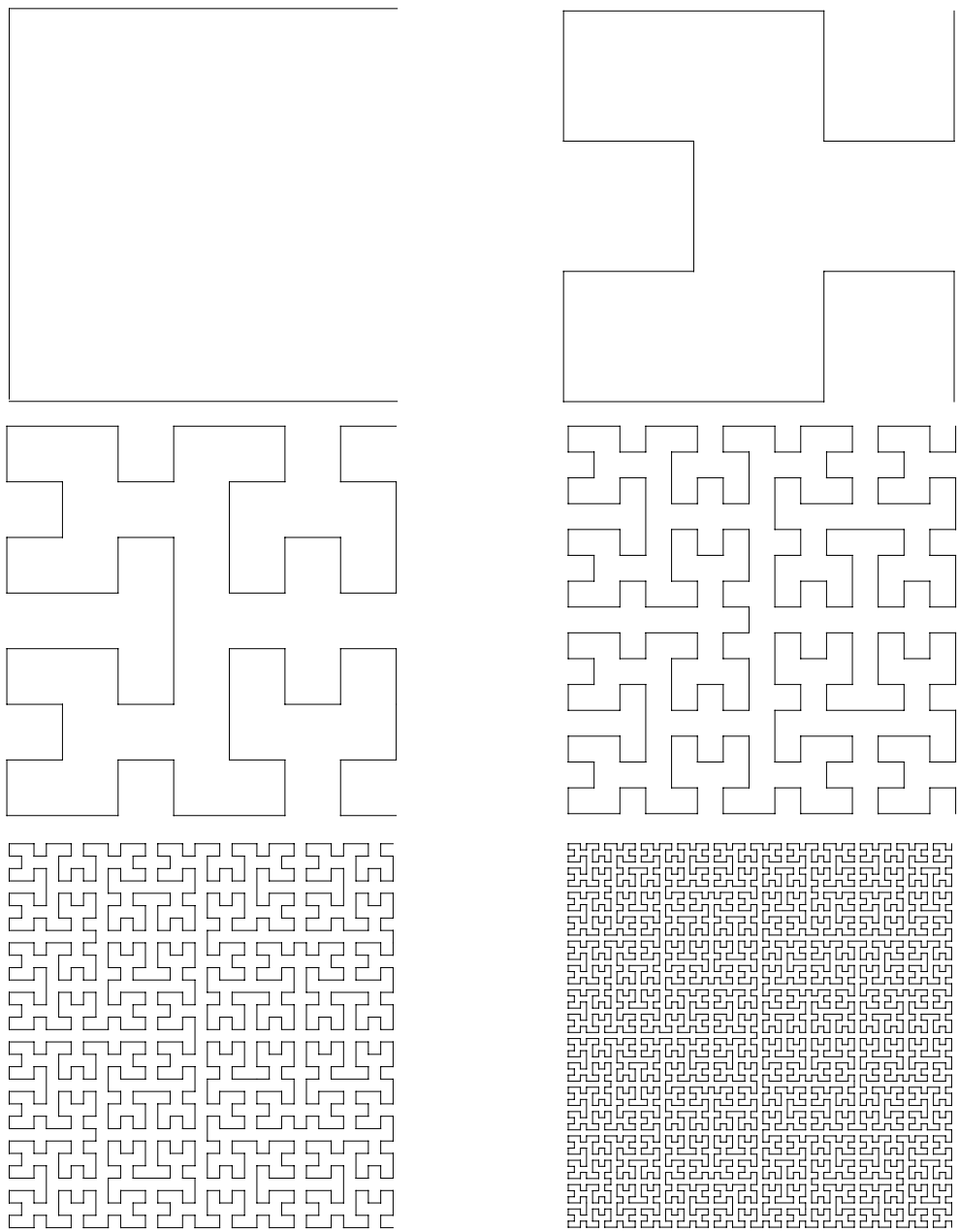


FIG. 21.17 – *Courbes de Hilbert d'ordres 1 à 6*

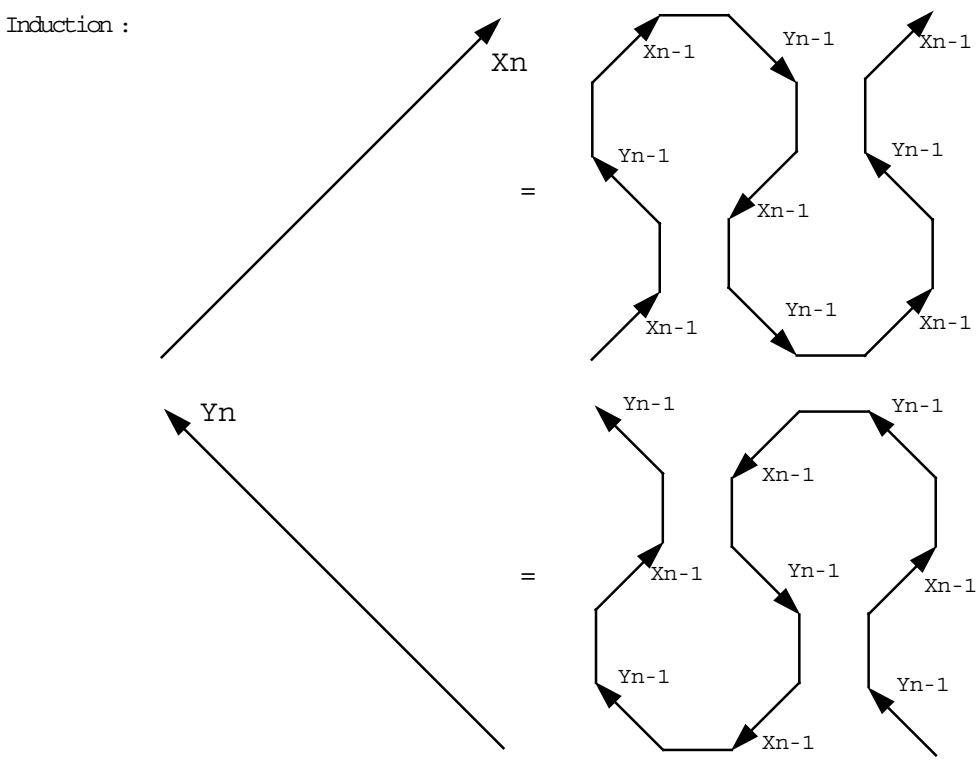


FIG. 21.18 – *Règles de construction des courbes de Peano*

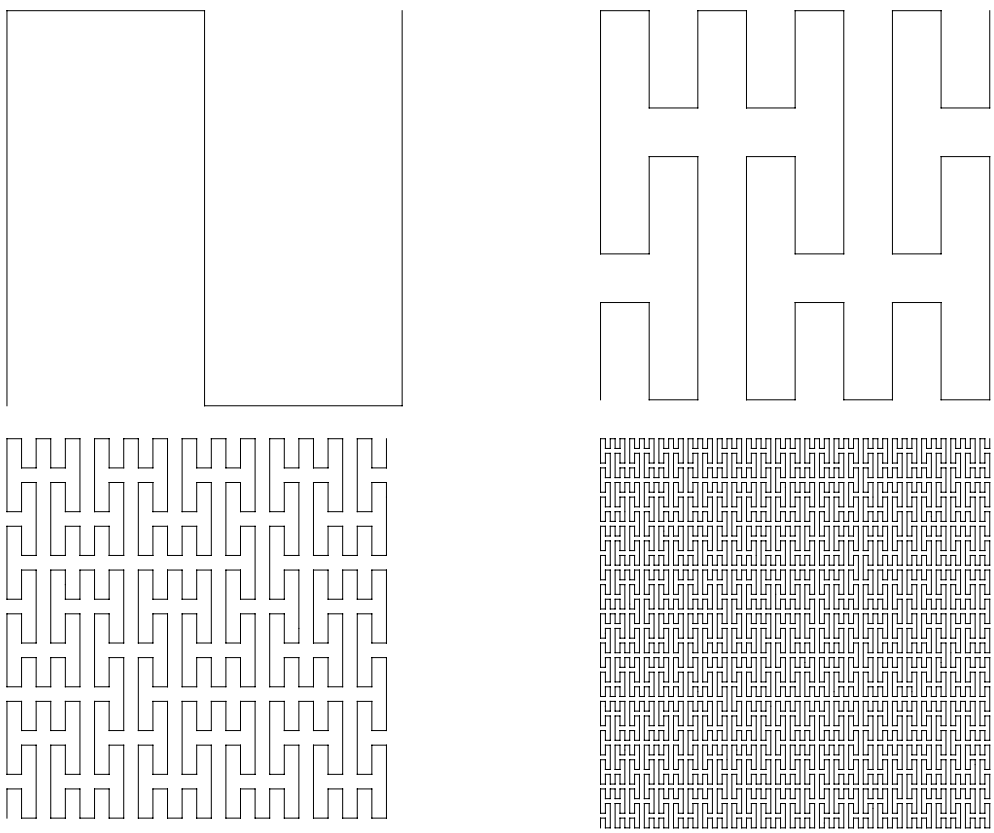


FIG. 21.19 – *Courbes de Peano*

```
program ConversionDecimalEnBinaire;
 uses  ;

 procedure DecimalEnBinaire(N : longint);
 begin
  if (N < 0) then
    begin
      ecrire('-'); translater(1, 0);
      DecimalEnBinaire(-N);
    end
  else if (N = 0) then
    begin
      ecrire('0'); translater(1, 0);
    end
  else if (N = 1) then
    begin
      ecrire('1'); translater(1, 0);
    end
 else {N >= 2}
    begin
      DecimalEnBinaire(N div 2);
      DecimalEnBinaire(N mod 2);
    end;
 end;

begin
 lg(7); DecimalEnBinaire(123456789); st;
end.
```

□

Addenda

Les langages ALGOL60 [27] puis LISP [26] furent les premiers à introduire l'usage de la récursivité. La polémique initiale sur l'efficacité des procédures récursives est maintenant dépassée, les techniques de compilation étant bien au point. L'idée de récursivité fut à la base de nombreux progrès en informatique, que ce soit les structures de données récursives (comme les arbres), les algorithmes récursifs (comme les algorithmes rapides de tri) ou plus récemment, la programmation fonctionnelle.

La façon la plus simple de comprendre une procédure récursive est d'imaginer qu'à chaque appel l'ordinateur exécute une nouvelle *copie* de la procédure. Pour montrer que cette succession d'appels se termine, il suffit de montrer qu'il existe une fonction entière de la valeur des paramètres qui est positive et décroit strictement lors d'appels successifs. Par exemple la procédure `Segment` du programme `SegmentDeVonKoch` (page 262) doit être appelée avec un paramètre `N` positif ou nul. Si c'est le cas, les appels récursifs sont de la forme `Segment(N - 1)` avec `N` > 0 de sorte que lors d'appels successifs les paramètres effectifs décroissent strictement en restant positifs. Ceci n'est pas possible indéfiniment. Par conséquent la procédure `Segment` ne boucle pas quand elle est appelée avec un paramètre initial positif ou nul.

Les *fonctions* sont aux expressions ce que les procédures sont aux instructions. Une déclaration de fonction est identique à une déclaration de procédure sauf que dans l'en-tête on indique le type du résultat et que dans le corps de la fonction, la dernière instruction exécutée de la forme "`NomDeLaFonction` := Expression" indique que la valeur retournée par l'évaluation de la fonction appliquée à ses paramètres est celle de l'"Expression". Par exemple, une constante est une fonction sans paramètres de sorte qu'une déclaration comme :

```
const TailleGrille = 15;
```

peut être remplacée par une déclaration de fonction :

```
function TailleGrille : integer;
begin
  TailleGrille := 15;
end;
```

Comme second exemple considérons le calcul de 2^n avec $n \geq 0$ en observant pour la base que $2^0 = 1$ et pour l'induction que $2^n = 2 * 2^{n-1}$:

```
function DeuxPuissance(N : integer) : longint;
begin
  if (N = 0) then
    begin
      DeuxPuissance := 1;
    end
  else
    begin
      DeuxPuissance := (2 * DeuxPuissance(N - 1));
    end
end;
```

Après cette déclaration, l'expression `DeuxPuissance(2)` vaut 4 et l'expression `DeuxPuissance(DeuxPuissance(2))` vaut 16.

En PASCAL on peut utiliser les fonctions trigonométriques `sin` (sinus), `cos` (cosinus) et `arctan` (arc tangente). Les angles sont exprimés en radians. Comme pour les fonctions `exp` (exponentielle) et `ln` (logarithme népérien), le paramètre est de type `integer`, `longint` ou `real` et le résultat est de type `real`. Pour la puissance, on utilise $a^x = e^{x \ln a}$. On pourra définir la fonction tangente par :

```
function Tg(X : real) : real;
  const
    MinPosReal = 1.2e−38;
    MaxPosReal = 3.4e+38;
begin
  if abs(cos(X)) <= MinPosReal then
    begin
      Tg := MaxPosReal;
    end
  else
    begin
      Tg := (sin(X) / cos(X));
    end;
end;
```

22

Variables

Nous avons utilisé des variables comme compteurs de boucles et dans les expressions. L'instruction d'affectation permet de les utiliser également pour mémoriser des résultats intermédiaires dans les calculs.

22.1 Variables et affectation

Une variable est utilisée pour conserver une valeur d'un certain type dans la mémoire de l'ordinateur. Une déclaration de variable indique le nom de la variable (qui est un identificateur) et son type :

```
var
  B : boolean;
  I : integer;
  R : real;
  S : string;
```

C'est le compilateur qui décide quelle partie de la mémoire et quel codage sera utilisé pour représenter la valeur de la variable. Comme la mémoire contient toujours quelque chose (puisque chaque bit qui la constitue vaut 0 ou 1), la variable a toujours une valeur. On peut utiliser cette valeur dans une expression (comme $((2 * \texttt{I}\ ) + 1)$) en la notant simplement par le nom de la variable (`I` dans l'exemple). Au début de l'exécution du programme, la valeur d'une variable est *indéterminée* en ce sens qu'elle correspond à l'état initial de la mémoire qui peut être quelconque. L'instruction d'*affectation* permet de changer la valeur d'une variable. Par exemple l'instruction :

```
B := true;
```

affecte la valeur "vrai" à la variable booléenne `B`. De manière générale, une instruction d'affectation a la forme suivante :

Instruction d'affectation :

`NomDeVariable` := Expression[a];

[a] Le type de l'expression est *compatible* avec celui de la variable.

Le symbole := se lit "devient égal à". L'exécution de cette instruction consiste à évaluer l'"Expression" pour calculer sa valeur qui devient celle de la variable nommée `NomDeVariable`. Le type de l'expression est compatible avec celui de la variable dans les cas suivants :

Compatibilité de type pour l'affectation :

Type de la variable	Type de l'expression
boolean	boolean
integer	integer
integer	longint[b]
longint	integer
longint	longint
real	integer
real	longint
real	real
string	string

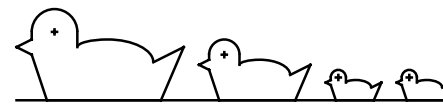
[b] C'est une erreur à l'exécution si la valeur de l'expression n'est pas comprise entre −`maxint` et `maxint`.

Quand les types de l'expression et de la variable sont compatibles mais ne sont pas les mêmes, le compilateur effectue des *conversions* c'est-à-dire les changements de codage appropriés.

22.2 Sauvegarde de l'état du robot

Une première utilisation des variables est la sauvegarde de l'état du $\mathcal{R}\text{o}\underline{b}\text{o}t$. Imaginons par exemple que nous voulions concevoir une procédure pour faire un dessin qui nécessite divers changements de la taille de la grille. Pour que l'appel de cette procédure dans un programme ne modifie pas la taille de la

grille utilisée dans le programme, il faut sauvegarder cette taille au début de la procédure puis la restituer à la fin, comme dans l'exemple suivant :



```
program Canards;
 uses ;
 procedure Canard(L : real);
  { Dessiner un canard sur une }
  { grille de taille L sans }
  { changer la grille originelle. }
  var LX : real; LY : real;
 begin
  { Mémoriser les dimensions de la grille originelle }
     LX := valLgX; LY := valLgY;
  { Dessiner un canard avec une grille de taille L } lg(L);
     { Ligne de flottaison } avf(19);
     { Queue } lgX(3 * L); lgY(7 * L); phtg; av; pdt; lgX(4 * L);
     lgY(2 * L); av;
     { Dos } lgX(6 * L); lgY(3 * L); p3htd; vg;
     lgX(4 * L); lgY(L); vg; pdt;
     { Tête } lg(L); avf(2); lg(3 * L); vg; vg;
     { Oeil } lc; pqtg; av; dc; pdt; av; pqtg; bc;
     { Bec } lgX(2 * L); lgY(L); av; phtd; av; pqtg; av;
     { Poitrail } lgX(2 * L); lgY(6 * L); av; phtg;
     { Placer le robot à l'arrière du canard } lg(L); avf(18);
  { Restituer la grille originelle }
     lgX(LX); lgY(LY);
 end;

begin
 lg(6); pe; Canard(3); av; Canard(2); av; Canard(1); av; Canard(1); st;
end.
```

22.3 Compter

Supposons que nous voulions compter le nombre d'itérations dans une boucle "while". On utilise une variable déclarée de type entier :

```
var I : integer;
```

Avant d'exécuter la boucle "while", la variable doit être initialisée à 0 :

```
I := 0;
```

Après chaque tour de boucle, la valeur de la variable doit être augmentée de 1 :

```
I := (I + 1);
```

Supposons que la valeur de la variable `I` soit égale à 5 *avant* l'affectation `I := (I + 1);`. La valeur de l'expression `(I + 1)` est $(5 + 1) = 6$ de sorte que la valeur de la variable `I` *après* l'affectation sera égale à 6.

Par exemple, ceci peut nous servir pour déterminer la taille de la fenêtre dans laquelle le *Robot* se déplace sur l'écran de l'ordinateur :

```
program CalculerTailleFenetreRobot;
 uses  ;
 { Calcul de la taille de la fenêtre de déplacement du robot. }
 var Largeur : integer; Hauteur : integer;

 procedure LargeurFenetreRobot;
 begin
  lg(1); lc; ce; ag; pe; Largeur := 0;
  while not tb do
    begin Largeur := Largeur + 1; av; end;
 end;

 procedure HauteurFenetreRobot;
 begin
  lg(1); lc; ce; eb; pn; Hauteur := 0;
  while not tb do
    begin Hauteur := Hauteur + 1; av; end;
 end;

begin
 LargeurFenetreRobot; HauteurFenetreRobot;
 message(concat('(', EntierEnChaine(Largeur), ' x ',
    EntierEnChaine(Hauteur), ')'));
 MarquerUnePause;
end.
```

22.4 Entrée d'une valeur d'une variable

On peut changer la valeur d'une variable en entrant sa nouvelle valeur au clavier de l'ordinateur. Pour ce faire la commande du *Robot* `modeTexte;` fait apparaître la fenêtre prévue pour les entrées-sorties en Pascal (cette fenêtre est cachée par la fenêtre de déplacement du *Robot*). Ensuite l'instruction `readln(X);` permet d'entrer la valeur de la variable `X` au clavier, cette valeur s'inscrivant en même temps dans la fenêtre d'entrées-sorties de Pascal. La procédure `readln(X);` est *polymorphe* ce qui veut dire que le type de la variable `X` peut être `boolean`, `integer`, `longint`, `real` ou `string` (alors que la

plupart des autres procédures en PASCAL sont *monomorphes* en ce sens que leurs paramètres ont toujours le même type [1]). La valeur que l'on tape au clavier s'écrit comme les constantes du type de `X` sauf que pour les chaînes de caractères, il ne faut pas utiliser d'apostrophes. On peut corriger les fautes de frappe avec la touche ⟵ d'effacement. Il faut terminer l'entrée de la valeur par la frappe de la touche ↩ de retour à la ligne.

L'instruction `readln`; (sans paramètre) permet d'attendre que la touche ↩ de retour à la ligne soit enfoncée. Elle est donc utile pour éviter que la fenêtre prévue pour les entrées-sorties en PASCAL ne disparaisse à la fin de l'exécution du programme ou lorsque l'exécution d'une commande quelconque du *Robot* fait réapparaître la fenêtre de dessin.

22.5 Affichage de la valeur d'une expression

Pour afficher la valeur d'une expression E dans la fenêtre d'entrées-sorties de PASCAL, on peut utiliser l'instruction polymorphe `write`(E);. Les écritures se font sur la même ligne, les unes à la suite des autres, en passant à la ligne suivante en fin de ligne. L'instruction `writeln`; permet de forcer le passage à la ligne. Par exemple le programme ci-dessous lit un entier n puis écrit sa factorielle $n! = (1 * 2 * \ldots * (n-1) * n)$:

```
program Factorielle;
  uses  ;
  const Nmin = 0; Nmax = 14;
  var N : integer; I : integer; F : longint;
begin
  { Montrer la fenêtre d'entrées-sorties de Pascal }
  modeTexte;
  { Lire la valeur de N (tel que Nmin <= N <= Nmax) au clavier }
  repeat
    write('N = '); readln(N);
    if ((N < Nmin) or (N > Nmax)) then
      begin
        write('N doit etre compris entre '); write(Nmin); write(' et ');
        write(Nmax); write('! Recommencer.'); writeln;
      end;
  until ((Nmin <= N) and (N <= Nmax));
  { Calculer F = N! }
```

1. Le type du paramètre formel et le type du paramètre effectif doivent être compatibles pour l'affectation.

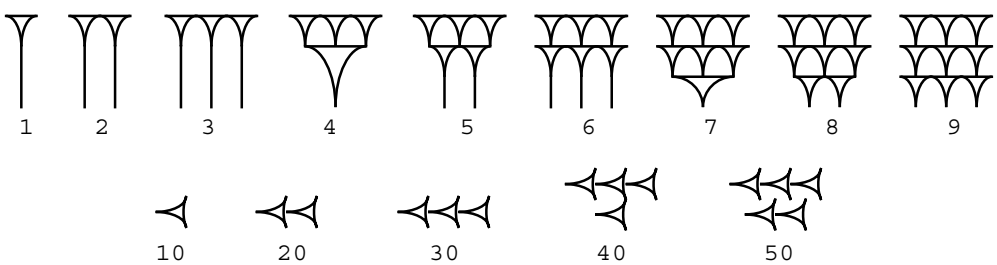
```
  F := 1;
  for I := 1 to N do
  begin
    F := F * I;
  end;
  { Ecrire F }
  write('N! = '); write(F); readln;
end.
```

Exercice 43

– *Écrire un programme qui dessine un cadre le plus près possible du bord de la fenêtre de déplacement du Robot.*

– *Écrire un programme qui esquisse les dessins à compléter de la figure 1.4 (page 7).*

– *La numération babylonienne [15], qui date d'environ 2000 ans av. J.-C., est en base 60. C'est donc une numération sexagésimale tout comme celle que nous utilisons encore pour noter l'heure. Les unités, de 1 à 59, sont notées de manière additive en accumulant des "clous" (pour 1) et des "chevrons" (pour 10) regroupés et superposés comme indiqué sur la figure ci-dessous.*

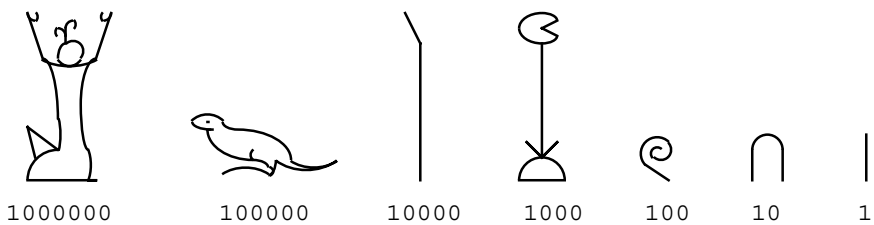


Au delà de 60, la notation est positionnelle, la valeur d'un chiffre occupant la $p^{ème}$ position (comptée de droite à gauche à partir de zéro) étant multipliée par 60^p. L'absence d'unités sexagésimales en position médiale (que nous noterions par 0) fut d'abord marquée par un espace puis, vers le $3^{ème}$ siècle av. J.-C., par "deux clous inclinés". Les Babyloniens ont donc mis 15 siècles pour inventer une notation pour le zéro sans toutefois en comprendre la valeur calculatoire ! Par exemple le nombre $3657 = 1 \times 60^2 + 0 \times 60^1 + 57 \times 60^0$ est représenté par la juxtaposition des chiffres 1, 0 et $57 = 50 + 7$:

"zéro" babylonien 3657

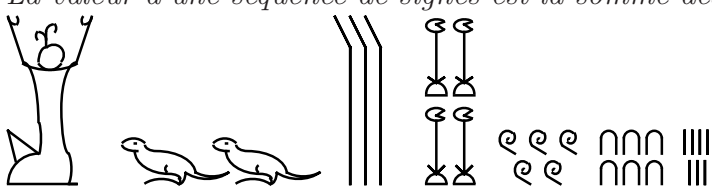
Écrire un programme Pascal *de conversion d'un nombre positif quelconque dans la numération babylonienne.*

– *La numération égyptienne (3ème millénaire av. J.-C.) est décimale et additive [15]. Elle utilise des hiéroglyphes pour noter les puissances de 10 (le trait vertical pour 1, l'anse pour 10, la spirale pour 100, le lotus pour 1000, l'index pour 10 000, le têtard pour 100 000 et le génie pour 1 000 000) :*



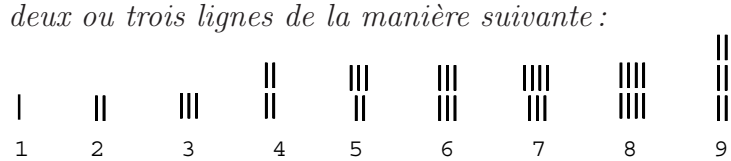
1000000 100000 10000 1000 100 10 1

La valeur d'une séquence de signes est la somme des valeurs de ces signes :



1 234 567

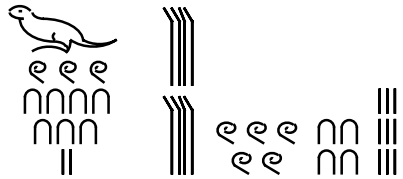
On écrit de gauche à droite ou de droite à gauche, la lecture se faisant dans le sens où sont orientés les hiéroglyphes. Les signes identiques se regroupent sur une, deux ou trois lignes de la manière suivante :



1 2 3 4 5 6 7 8 9

Entre -2000 et -1600 ans av. J.-C., une représentation hybride (additive et partiellement positionnelle) est apparue. Un nombre inférieur à 1000 situé avant, au dessus ou au dessous du signe du génie du million, du têtard, de l'index ou du lotus représente le produit de ce nombre et de la valeur de

ce hiéroglyphe. Par exemple, le nombre 37 280 549 peut se décomposer en $((372 * 100000) + 80549)$ *et s'écrit :*



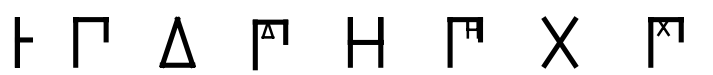
37 280 549

Écrire un programme Pascal *de conversion d'un nombre dans la numération égyptienne.*

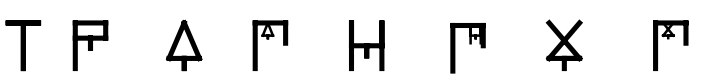
– *La numération acrophonique grecque [15] fut utilisée à partir du Vème siècle av. J.-C. à Athènes. C'est une numération additive décimale utilisant la base auxiliaire 5 pour diminuer les répétitions. Les chiffres supérieurs à un sont notés par des lettres (ou combinaisons de lettres) correspondant chacune à l'initiale (ou à l'abréviation) du nom du nombre correspondant :*

Écrire un programme Pascal *de conversion d'un nombre inférieur à 100 000 dans la numération grecque. Pour représenter les grands nombres, utiliser la numération acrophonique pour les monnaies :*
Oboles :

Drachmes (1 drachme = 6 oboles) :



Talents (1 talent = 6 000 drachmes) :



Par exemple le nombre 239 686 966 se représente par :

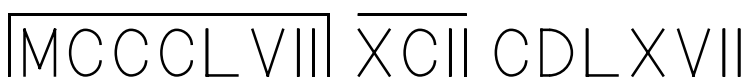
6657 Talents 5827 Drachmes 4 Oboles

239 686 966

_ *La numération romaine [15] est additive et soustractive (tout signe placé à gauche d'un signe de valeur supérieure s'en soustrait). Ses chiffres sont bien connus :*

1 5 10 50 100 500 1000

Pour les grands nombres, la barre horizontale multiplie par 1 000 la valeur du nombre qu'elle surligne et le rectangle sans base multiplie par 100 000 la valeur du nombre qu'il enferme, comme dans l'exemple 135 792 468 ci-dessous :



1 3 5 7 9 2 4 6 8

Écrire un programme Pascal *de conversion d'un nombre dans la numération romaine.*

_ *Écrire un programme de dessin de la courbe de Sierpinski à un ordre quelconque de la figure 22.1 (page 278).*

_ *Un carré magique d'ordre n est un tableau carré de n^2 nombres tel que la somme des nombres de chaque ligne, de chaque colonne et de chaque diagonale est la même. Des exemples sont donnés à la figure 22.2 (page 278). Notons $C_{i,j}$ le nombre figurant à la $i^{ème}$ ligne et la $j^{ème}$ colonne du carré comptées à partir de 1 et numérotées de bas en haut pour les lignes et de gauche à droite pour les colonnes. Écrire un programme* Pascal *qui construit des carrés magiques d'ordre n impair en commençant par placer 1 dans $C_{i,j} = C_{\frac{n+1}{2},n}$ puis affecte les nombres 2, 3, ..., n^2 aux éléments $C_{i,j}$ obtenus en augmentant i et j de 1 modulo n à chaque fois pendant $n-1$ pas, et en diminuant j de 1 tout en laissant i inchangé à chaque $n^{ème}$ pas.*

□

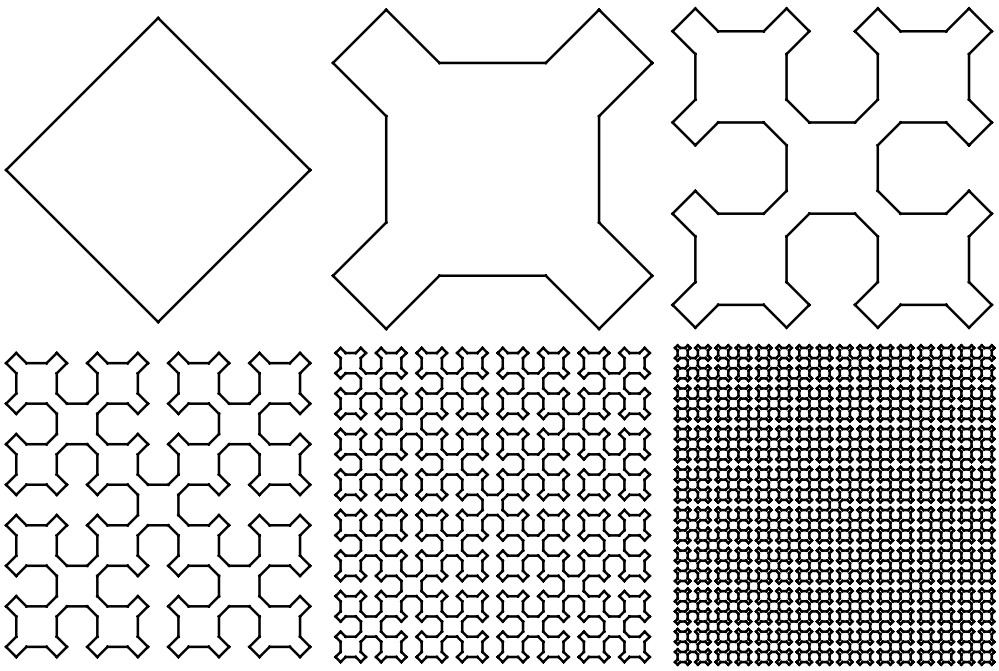


FIG. 22.1 – *Courbes de Sierpinski d'ordres 0 à 5*

8	1	6
3	5	7
4	9	2

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

30	39	48	1	10	19	28
38	47	7	9	18	27	29
46	6	8	17	26	35	37
5	14	16	25	34	36	45
13	15	24	33	42	44	4
21	23	32	41	43	3	12
22	31	40	49	2	11	20

FIG. 22.2 – *Carrés magiques d'ordres 3, 5 et 7*

Corrigé 43 *Nous donnons un programme, les autres se trouvant sur disquette :*

```
program CarreMagique;
 uses  ;
 var
  N : integer;  { Ordre du carré magique (impair) }
  I : integer;  { Indice des lignes (de 1 à N de bas en haut) }
  J : integer;  { Indice des colonnes (de 1 à N de gauche à droite) }
  C : integer;  { Elément de la ligne I et de la colonne J }

 procedure Afficher(I : integer; J : integer; C : integer);
  { Afficher l'élément C au point de coordonnées (I, J) du carré }
 begin
  eb; ag; translater((2 * (I - 1)), ((2 * J) - 1));
  Ecrire(concat(' ', EntierEnChaine(C)));
 end;

begin
 { Lire l'ordre du carré magique }
 modetexte; { Montrer la fenêtre d'entrées-sorties de Pascal }
 repeat
  write('Ordre du carré magique (impair), N = ');
  readln(N);
 until ((N > 0) and odd(N));
 { Dessiner la matrice dans la fenêtre de dessin du robot }
 lg(12); CacherRobot;
 for I := 0 to N do
  begin
   eb; ag; translater((2 * I), 0); pn; avf(2 * N);
   eb; ag; translater(0, (2 * I)); pe; avf(2 * N);
  end;
  { Remplir le carré magique }
 I := (N + 1) div 2; J := N; Afficher(I, J, 1);
 for C := 2 to sqr(N) do
  begin
   if (C mod N) <> 1 then
    begin
     I := (I mod N) + 1;
     J := (J mod N) + 1;
    end
   else
    begin
     J := J - 1;
    end;
      Afficher(I, J, C);
```

```
    end;
  st;
end.
```

□

Addenda

Une variable sert à donner un *nom* à une mémoire qui conserve une *valeur* représentée selon un code déterminé par le *type* de la variable. Pour un codage binaire, on peut imaginer que la mémoire est une suite de lampes éteintes (pour représenter le bit 0) ou allumées (pour le bit 1). La valeur initiale de la variable est *indéterminée* puisqu'elle dépend de l'état de la mémoire au moment du début de l'exécution du programme (certains compilateurs l'initialisent à zéro). L'instruction d'affectation permet de conserver une nouvelle valeur en modifiant physiquement l'état de cette mémoire. La nouvelle valeur est donnée par une expression (qui peut utiliser l'ancienne valeur). La notion informatique de variable est donc très différente de la notion mathématique homonyme. En mathématiques une variable désigne une valeur inconnue fixée une fois pour toutes. Pour un mathématicien, une variable informatique serait la suite des valeurs qu'elle prend à sa création puis après chaque affectation.

On peut déclarer des variables *locales* à une procédure (ou fonction), en insérant la déclaration de la variable entre l'en-tête et le corps. Dans ce cas la variable n'est utilisable que dans le corps de la procédure. A l'exécution, elle n'existe (en ce sens qu'un espace mémoire lui est alloué) que pendant la durée de l'exécution du corps de la procédure. Pour une procédure récursive, il s'agit à chaque appel d'une nouvelle variable sans aucun lien avec les instances correspondant aux appels récursifs antérieurs.

Dans le corps de la procédure on peut également utiliser des variables *globales* dont la déclaration figure dans le programme avant celle de la procédure. Ces variables globales existent pendant toute la durée de l'exécution du programme. Par conséquent en cas d'appels récursifs d'une procédure, il s'agit toujours de la même variable globale.

Enfin des variables locales et globales peuvent porter le même nom. Dans ce cas c'est la variable locale qui est visible dans la procédure, la variable globale continuant à exister mais étant inaccessible.

Bibliographie

[1] J. W. Backus, R.J. Beeber, S. Best, R. Goldberg, L. M. Haibt, H. L. Herick, R. A. Nelson, D. Sayre, P. B. Sheridan, H. Stern, I. Ziller, R. A. Hugues & R. Nutt. *The Fortran automatic coding system*, Proceedings Western Joint Computer Conference, Volume 11, pages 188–198, 1957.

[2] M. Barnsley, R. Devaney, B. Mandelbrot, H.-O. Peitgen, D. Saupe & R. Voss. *The science of fractal images*, Springer-Verlag, Berlin, 312 pages, 1988.

[3] C. Bertho. *Télégraphes et téléphones, de Valmy au microprocesseur*, Le Livre de Poche, Paris, 541 pages, 1981.

[4] P. Benoit. "Calcul, algèbre et marchandise", dans [36], pages 197–221, 1989.

[5] C. Böhm. "Macchina calcolatrice digitale a programma con programma preordinato fisso con tastiera algebrica ridotta atta a comporre formule mediante la combinazione dei singoli elementi simbolici", Dépôt de brevet N° 13567, Milan, 1^{er} octobre 1952, 26 pages.

[6] M. Boutet de Monvel. *Chansons de France pour les petits enfants*, Gautier-Languereau, Paris, 77 pages, 1981.

[7] Commission ministérielle de terminologie de l'informatique. *Glossaire des termes officiels de l'informatique*, AFNOR, Tour Europe, 92080 Paris La Défense Cedex 7, 21 pages, 1989 ($2^{\text{ème}}$ édition).

[8] P. Cousot & R. Cousot. "Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints", Conference Record of the 4^{th} ACM Symposium on Principles of Programming Languages, Los Angeles, Californie, U.S.A., p. 238–252, 1977.

[9] P. Cousot. *Introduction à l'algorithmique numérique et à la programmation en Pascal*, McGraw-Hill, Paris, 621 pages, 1988.

[10] H. S. M. Coxeter, M. Emmer, R. Penrose & M. L. Teuber. *M. C. Escher: Art and Science*, North-Holland, Amsterdam, 402 pages, 1986.

[11] H. M. Cundy & A. R. Rollett. *Mathematical models*, Tarquin publications (Stradbroke, Diss, Norfolk, G.B.), 286 pages, 1981 (3ème édition).

[12] A. Danhauser. *Théorie de la musique (édition revue et corrigée par H. Rabaud)*, Éditions Henry Lemoine, Paris, 128 pages, 1929.

[13] S. W. Golomb. "A geometric proof of a famous identity", *Mathematical gazette*, Vol. 69, p. 198–200, 1965.

[14] B. Grünbaum & G. C. Shephard. *Tilings and patterns*, W. H. Freeman & Cie, New York, 700 pages, 1987.

[15] G. Guitel. *Histoire comparée des numérations écrites*, Flammarion, Paris, 857 pages, 1975.

[16] S. Horemis. *Optical and geometrical patterns and designs*, Dover, New York, 1970.

[17] K. E. Iverson. *A Programming Language*, J. Wiley & Sons, 1962.

[18] K. Jensen & N. Wirth. *Pascal, user manual and report*, Springer-Verlag (Heidelberg), 167 pages, 1976 (2ème édition).

[19] B. W. Kernighan & D. M. Ritchie. *The C programming language*, Prentice-Hall (Englewood Cliffs, New Jersey, U.S.A.), 228 pages, 1978.

[20] D. E. Knuth. *The art of computer programming, Volume 1 / Fundamental algorithms*, Addison-Wesley (Reading, Massachusetts, USA), 634 pages, 1973 (2ème édition).

[21] D. E. Knuth. *The TEXbook*, Addison-Wesley (Reading, Massachusetts, USA), 483 pages, 1987 (12ème édition).

[22] L. Lamport. *LATEX a document preparation system*, Addison-Wesley (Reading, Massachusetts, USA), 242 pages, 1986.

[23] P. Lévy. "L'invention de l'ordinateur", dans [36], pages 515–535, 1989.

[24] J. L. Locher. *The world of M. C. Escher*, Abrams (New York, USA), 1971; traduction française : J. L. Locher, *Le monde de M. C. Escher*, Éditions du Chêne (Paris), 293 pages, 1986 (12ème édition).

[25] B. Mandelbrot. *Les objets fractals* suivi de *Survol du langage fractal*, Flammarion, Nouvelle collection scientifique (Paris), 1989 (3ème édition).

[26] J. McCarthy, P. W. Abrahams, D. J. Edwards, T. P. Hart & M. I. Levin. *LISP 1.5 Programmer's manual*, MIT Press, 1965.

[27] P. Naur *et al.* (Eds.). "Revised report on the algorithmic language Algol 60", *Communications of the Association for Computing Machinery*, volume 6, pages 1–17, 1963.

[28] F. C. Odds. "Spirolaterals", *Mathematics teacher*, volume 66, pages 121–124, 1973.

[29] S. Papert. *Mindstorms: children, computers and powerful ideas*, Harvester Press, 1980.

[30] J. Ritter. "Babylone -1800", dans [36], pages 17–37, 1989.

[31] H. Rutishauser. "Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen", *Mitteilungen aus dem Inst. für angew. Math. an der ETH Zürich*, No. 3, Birkhäuser, Bâle, 1952, 45 pages.

[32] R. Sabatier. *Le livre des chansons de France*, Gallimard, Paris, 157 pages, 1984.

[33] R. Sabatier. *Deuxième livre des chansons de France et d'ailleurs*, Gallimard, Paris, 165 pages, 1986.

[34] C. Sabatier & R. Sabatier. *Troisième livre des chansons de France*, Gallimard, Paris, 165 pages, 1987.

[35] R. Sedgewick. *Algorithms*, Addison-Wesley, Reading, Massachusetts, U.S.A., 650 pages, 1988 ($2^{ème}$ édition), traduit en français par J.-M. Moreau : R. Sedgewick, *Algorithmes*, InterÉditions, Paris, 1990.

[36] M. Serres. *Éléments d'histoire des sciences*, Bordas, Paris, 576 pages, 1989.

[37] S. Thévenet, A. Garioud & N. Pitot. *Mathématiques CM2*, Bordas, Paris, 160 pages, 1986.

[38] H. Voderberg. "Zur Zerlegung der Umgebung eines ebenen Bereiches in kongruente", *Jahresber. Deutsch. Math.-Verein.*, volume 46, pages 229–231, 1936 ; "Zur Zerlegung der Ebene in kongruente Bereiche in Form einer Spirale", Ibid., volume 47, pages 159–160, 1937

[39] M. V. Wilkes, D. J. Wheeler & S. Gill. "The preparation of programs for an electronic digital computer: with special reference to the EDSAC and the use of a library of subroutines", Addison-Wesley, Cambridge, Massachusetts, 170 pages, 1951.

[40] L. B. Wilson & R. G. Clark. *Comparative programming languages*, Addison-Wesley publishing company, Reading, Massachusetts, 379 pages, 1988.

[41] N. Wirth. “The programming language Pascal”, *Acta Informatica*, Vol. 1, Springer-Verlag (Heidelberg), pages 35–63, 1971.

Résumé des commandes du robot

Grille des points d'arrêt du $\mathcal{R}^o\!\underline{b}^o t$:	
`dg`	dessine la grille des points d'arrêt du $\mathcal{R}^o\!\underline{b}^o t$ (peut être interrompue par Esc, § ou #).
`lg`(n)	définit la nouvelle longueur n des côtés d'un carreau de la grille ((12.0×12.0) pixels par défaut, $n \geq 0.0$).
`lgX`(n)	définit la nouvelle longueur n du côté horizontal (X) d'un carreau de la grille (12.0 pixels par défaut, $n \geq 0.0$).
`lgY`(n)	définit la nouvelle longueur n du côté vertical (Y) d'un carreau de la grille (12.0 pixels par défaut, $n \geq 0.0$).
`lgRT`(r,t)	définit les nouvelles longueurs des côtés d'un carreau de la grille par les coordonnées polaires r et t (la longueur $r \geq 0.0$ de la diagonale du carreau est exprimée en pixels et l'angle t que fait cette diagonale avec le côté horizontal est exprimé en degrés ($0.0 \leq t \leq 90.0$)).
`valLgX`	est égal à la valeur (de type `real`) de la longueur du côté horizontal (X) d'un carreau de la grille (comptée en pixels).
`valLgY`	est égal à la valeur (de type `real`) de la longueur du côté vertical (Y) d'un carreau de la grille (comptée en pixels).
`valLgR`	est égal à la valeur (de type `real`) de la longueur de la diagonale (R) d'un carreau de la grille (comptée en pixels).
`valLgT`	est égal à la valeur (de type `real`) de l'angle (T) de la diagonale d'un carreau de la grille avec son côté horizontal (comptée en degrés).

`valX`	est égal à la valeur (de type `real`) de l'abscisse absolue (X) du $\mathcal{R}\!\mathit{obot}$ (comptée en pixels).
`valY`	est égal à la valeur (de type `real`) de l'ordonnée absolue (Y) du $\mathcal{R}\!\mathit{obot}$ (comptée en pixels), de sorte que `X := valX; Y := valY`; et plus tard `lg(1.0); deplacer(X, Y)`; replace le $\mathcal{R}\!\mathit{obot}$ à sa place initiale sur l'écran).

Pivotements relatifs sur place :	
`phtd`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place d'un huitième de tour à droite.
`pqtd`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place d'un quart de tour à droite.
`p3htd`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place de 3 huitièmes de tour à droite.
`phtg`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place d'un huitième de tour à gauche.
`pqtg`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place d'un quart de tour à gauche.
`p3htg`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place de 3 huitièmes de tour à gauche.
`pdt`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place d'un demi-tour.

Pivotements absolus sur place :	
`pn`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers le nord (N).
`pne`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers le nord-est (NE).
`pe`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers l'est (E).
`pse`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers le sud-est (SE).
`ps`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers le sud (S).
`psw`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers le sud-ouest (SW).
`pw`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers l'ouest (W).
`pnw`	fait pivoter le $\mathcal{R}\!\mathit{obot}$ sur place vers le nord-ouest (NW).
`pivoter`(o)	fait pivoter le $\mathcal{R}\!\mathit{obot}$ dans l'orientation indiquée par la valeur de l'entier o modulo 8 égale à `nord` $= 0$, `nordEst` $= 1$, `est` $= 2$, `sudEst` $= 3$, `sud` $= 4$, `sudOuest` $= 5$, `ouest` $= 6$ ou `nordOuest` $= 7$.
`rt`(t)	change le repère et l'orientation du $\mathcal{R}\!\mathit{obot}$ pour qu'il effectue une rotation sur place d'un angle de t degrés (dans le sens trigonométrique si $t > 0$ et dans le sens des aiguilles d'une montre si $t < 0$).
`valOrientation`	est égal à la valeur (de type `integer`) de l'orientation du $\mathcal{R}\!\mathit{obot}$ codée par `nord` $= 0$, `nordEst` $= 1$, `est` $= 2$, `sudEst` $= 3$, `sud` $= 4$, `sudOuest` $= 5$, `ouest` $= 6$ et `nordOuest` $= 7$. Les directions `nord`, `est`, `sud` et `ouest` sont parallèles aux bords de l'écran, le `nord` en haut. Les directions `nordEst`, `sudEst`, `sudOuest` et `nordOuest` correspondent à l'orientation des diagonales d'un carreau de la grille.

Crayon :	
`bc`	baisse le crayon pour dessiner lors des déplacements.
`lc`	lève le crayon pour éviter de dessiner lors des déplacements.
`dc`	dessine une petite croix (de l'épaisseur du crayon), sans déplacer le $\mathcal{R}^{o}_{b}{}^{o}t$.
`dp`	dessine un point (de l'épaisseur du crayon), sans déplacer le $\mathcal{R}^{o}_{b}{}^{o}t$.
`ec`(n)	définit la nouvelle épaisseur n du crayon (1 pixel par défaut).
`valEc`	est égal à la valeur (de type `integer`) de l'épaisseur du crayon.
`valBc`	est égal à la valeur booléenne `true` (vrai) si et seulement si le crayon est baissé, sinon à `false` (faux).

Déplacements (avec ou sans tracé selon que le crayon est levé ou baissé) :	
`av`	fait avancer le $\mathcal{R}^{o}_{b}{}^{o}t$ tout droit jusqu'au prochain point d'arrêt sur la grille.
`vd`	fait tourner le $\mathcal{R}^{o}_{b}{}^{o}t$ par un virage d'un quart de cercle à droite.
`vg`	fait tourner le $\mathcal{R}^{o}_{b}{}^{o}t$ par un virage d'un quart de cercle à gauche.
`avf`(n)	fait avancer le $\mathcal{R}^{o}_{b}{}^{o}t$ tout droit n fois (sans rien faire si $n = 0$ et en reculant si $n < 0$).
`tl`(l)	change le repère (mais pas l'angle des directions NE-E) et fait avancer le $\mathcal{R}^{o}_{b}{}^{o}t$ par une translation de l pixels dans la direction où il se trouve (si $l > 0$) ou dans la direction opposée (si $l < 0$).
`tracer`(X,Y)	trace un segment de droite (si le crayon est baissé) de la position courante du $\mathcal{R}^{o}_{b}{}^{o}t$ au point de coordonnées (X,Y) et déplace le $\mathcal{R}^{o}_{b}{}^{o}t$ en ce point.
`valLgTl`	est égal à la valeur (de type `real`) de la longueur du segment tracé par la dernière translation par `av` ou `tl` (comptée en pixels).

Positionnement au centre et sur les bords du cadre :	
`ce`	place le $\mathcal{R}^{o}_{b}{}^{o}t$ et *le centre de la grille* au centre de *l'écran*, sans changer l'orientation du $\mathcal{R}^{o}_{b}{}^{o}t$ et sans rien dessiner.
`eh`	place le $\mathcal{R}^{o}_{b}{}^{o}t$ sur la grille en haut, sans changer sa position verticale, ni son orientation et sans rien dessiner.
`eb`	place le $\mathcal{R}^{o}_{b}{}^{o}t$ sur la grille en bas, sans changer sa position verticale, ni son orientation et sans rien dessiner.
`ag`	place le $\mathcal{R}^{o}_{b}{}^{o}t$ sur la grille à gauche, sans changer sa position horizontale, ni son orientation et sans rien dessiner.
`ad`	place le $\mathcal{R}^{o}_{b}{}^{o}t$ sur la grille à droite, sans changer sa position horizontale, ni son orientation et sans rien dessiner.

Repère cartésien :
`translater`(X,Y) translate le *Robot* de X carreaux de la grille horizontalement (à droite si $X > 0$ et à gauche si $X < 0$) et Y carreaux de la grille verticalement (en haut si $Y > 0$ et en bas si $Y < 0$), sans changer son orientation et sans rien dessiner.
`deplacer`(X,Y) déplace le *Robot* au point de coordonnées (X, Y) sur la grille, sans changer son orientation ni rien dessiner, l'origine du repère cartésien étant placée au centre de la grille.

Test booléen de l'orientation du *Robot* :	
`tn`	teste si le *Robot* est orienté au nord (N).
`tne`	teste si le *Robot* est orienté au nord-est (NE).
`te`	teste si le *Robot* est orienté à l'est (E).
`tse`	teste si le *Robot* est orienté au sud-est (SE).
`ts`	teste si le *Robot* est orienté au sud (S).
`tsw`	teste si le *Robot* est orienté au sud-ouest (SW).
`tw`	teste si le *Robot* est orienté à l'ouest (W).
`tnw`	teste si le *Robot* est orienté au nord-ouest (NW).

Test booléen de la position du *Robot* :	
`tb`	teste si le *Robot* est au bord du cadre (un déplacement du *Robot* en avant par `av` le faisant sortir du cadre).

Temps :
`heure(H, Mi, S)` affecte aux variables `H`, `Mi` et `S` trois entiers qui sont l'heure (entre 0 et 23), la minute (entre 0 et 59) et la seconde (entre 0 et 59) présentes.
`date(J, Mo, A)` affecte aux variables `J`, `Mo` et `A` trois entiers qui sont le jour (entre 1 et 31), le mois (entre 1 et 12) et l'année courante.
`delai`(t) attendre pendant un délai de t secondes (t de type `real`).

Interruption de l'exécution du programme :
[Esc] ou [§] ou [#] stoppe le *Robot* au cours de l'exécution de l'une de ses commandes.

Mise au point du programme :	
`st`	stoppe le $\mathcal{R}^{o}_{b}{}^{o}t$ qui s'arrête définitivement en tapant l'une des touches [Esc], [§] ou [#]) ou redémarre en enfonçant une autre touche du clavier ou le bouton de la souris. Avant de redémarrer, l'utilisateur a la possibilité de diminuer (avec la touche [-]) ou d'augmenter (avec la touche [+]) la vitesse du $\mathcal{R}^{o}_{b}{}^{o}t$ (entre 0 et 10, la vitesse 0 faisant passer en mode pas à pas).
`pp`	exécute la suite du programme en mode pas à pas, à la vitesse 0.
`ex`	exécute la suite du programme en mode exécution normale, à la vitesse 10.
`vt`(v)	exécute la suite du programme à la vitesse v (en pas à pas si $v \leq 0$, avec un maximum de 10 si $v > 10$).
`montrerRobot`	rend le $\mathcal{R}^{o}_{b}{}^{o}t$ visible sur l'écran pendant le dessin (option par défaut).
`cacherRobot`	rend le $\mathcal{R}^{o}_{b}{}^{o}t$ invisible sur l'écran pendant le dessin.
`InterdireSorties`	interdit les sorties du $\mathcal{R}^{o}_{b}{}^{o}t$ en dehors du cadre dans la suite du programme (option par défaut).
`autoriserSorties`	autorise les sorties du $\mathcal{R}^{o}_{b}{}^{o}t$ en dehors du cadre dans la suite de l'exécution du programme.
`valVt`	est égal à la valeur (de type `integer`) de la vitesse du $\mathcal{R}^{o}_{b}{}^{o}t$ (entre 0 (mode pas à pas) et 10).
`valVisible`	est égal à la valeur booléenne `true`, (respectivement `false`) si le $\mathcal{R}^{o}_{b}{}^{o}t$ est visible (respectivement invisible) sur l'écran.
`valAutoriserSorties`	est égal à la valeur booléenne `true` (respectivement `false`) si les sorties en dehors du cadre sont autorisées (respectivement interdites).

Couleur :	
Les couleurs sont `blanc` $= 0$, `jaune` $= 1$, `bleu` $= 2$, `violet` $= 3$, `vert` $= 4$, `rouge` $= 5$, `indigo` $= 6$, `noir` $= 7$ (dans l'ordre du plus clair au plus foncé sur un écran ayant au moins 8 niveaux de gris) et `inverse = 8`. Sur un écran noir et blanc, et selon le modèle de l'ordinateur, toutes les couleurs (sauf blanc) apparaissent en noir ou les couleurs sont représentées par des motifs répétés.	
`cc`(c)	définit la couleur c du crayon. Si la couleur est `inverse`, les tracés se feront en vidéo inverse (par exemple noir sur fond blanc et blanc sur fond noir).

`peindre` peint de la couleur du crayon la région blanche de surface maximale où se trouve le $\mathcal{R}^{o}_{b}{}^{o}t$ (la peinture peut être interrompue par [Esc], [§] ou [#], le $\mathcal{R}^{o}_{b}{}^{o}t$ continuant ensuite son travail normalement).

`cf`(c) efface complètement le dessin puis remplit l'écran de la couleur de fond c, `inverse` n'étant pas autorisé. Le $\mathcal{R}^{o}_{b}{}^{o}t$ est placé au centre de l'écran dans les conditions initiales (grille (12×12), visible, tourné vers le nord, sorties hors du cadre interdites, crayon noir de taille 1 baissé et exécution normale à la vitesse 10).

`valCf` est égal à la valeur (de type `integer`) de la couleur du fond.

`valCc` est égal à la valeur (de type `integer`) de la couleur du crayon.

`ecranNoirEtBlanc` est égal à `true` si et seulement si l'écran est noir et blanc (sinon `false`).

Écriture dans la fenêtre de dessin du $\mathcal{R}^{o}_{b}{}^{o}t$:

`ecrire`(c) écrit la chaîne de caractères c (entre apostrophes ′ et ′, où toute apostrophe ′ doit être doublée) à droite du $\mathcal{R}^{o}_{b}{}^{o}t$ sans changer sa position ni celle de son crayon.

`entierEnChaine`(i) est la chaîne de caractères représentant l'entier i en base 10.

`reelEnChaine`(r) est la chaîne de caractères représentant le réel r sous la forme $d.dddd$`E`$\pm ee$ où d est un digit de la mantisse, e est un digit de l'exposant et `E` se lit "fois 10 à la puissance".

`booleenEnChaine`(b) est la chaîne de caractères (′`Vrai`′ ou ′`Faux`′) représentant le booléen b.

`concat`$(c_1,c_2,\ldots,c_n)$ est la chaîne de caractères obtenue en concaténant les chaînes c_1, c_2, ..., c_n les unes derrière les autres.

`policeTexte`(p) choisit la police p du texte (qui dépend du type d'ordinateur utilisé).

`tailleTexte`(t) choisit la taille t du texte (qui dépend du type d'ordinateur utilisé).

`styleTexte`(s) choisit le style du texte (qui dépend du type d'ordinateur utilisé).

`valPoliceTexte` valeur de la police courante du texte.

`valTailleTexte` valeur de la taille courante du texte.

`valStyleTexte` valeur du style courant du texte.

Messages affichés en dessous de la fenêtre de déplacement du *Robot* :
`message`(c) efface le message dans le cadre prévu en dessous de la fenêtre de déplacement du *Robot* puis y écrit la chaîne de caractères c.
`marquerUnePause` marque une pause (en faisant clignoter le symbole ↩ dans la fenêtre des messages) en attendant que le bouton de la souris ou qu'une touche du clavier soit enfoncé.
`effacerMessage` efface le message dans le cadre prévu en dessous de la fenêtre de déplacement du *Robot*.
`lireCar`(`C`) attend (en faisant clignoter le symbole ↩ dans la fenêtre des messages) qu'une touche du clavier ou que le bouton de la souris soit enfoncé, puis lit le caractère correspondant à la touche enfoncée (ou le caractère nul `chr`(`0`) si le bouton de la souris a été enfoncé) et l'affecte à la variable `C` (de type caractère `char`).
`mementoCommandesClavier` retourne une chaîne de caractères rappelant les commandes de pilotage interactif du *Robot*.
`valMessage` est égal à valeur (de type `string`) du message (de type `string`) inscrit dans le cadre sous la fenêtre de déplacement du *Robot* ou la chaîne vide '' s'il n'y a aucun message.

Lecture et écriture dans la fenêtre texte de Pascal :
`modeTexte` fait apparaître la fenêtre "Texte" de Pascal (pour réaliser les entrées-sorties avec les instructions `write`, `writeln`, `read` et `readln`). La réapparition de la fenêtre de dessin du *Robot* est automatique à la prochaine utilisation d'une commande du *Robot*.

Impression et conservation du dessin du *Robot* :
`definirFormatImpression`(f) définit le format f = `reduction` = 0, `standard` = 1 ou `agrandissement` = 2 de la prochaine impression.
`definirTitreImpression`(t) définit la chaîne de caractères t (entre apostrophes ' et ', où toute apostrophe ' doit être doublée) comme le titre pour la prochaine impression.
`imprimer` imprime le dessin dans le format et avec le titre (en dessous du dessin) tels qu'ils ont été dernièrement définis (`standard` et '' par défaut).
`valFormatImpression` est égal à la valeur (de type `integer`) du format d'impression.
`valTitreImpression` est égal à la valeur (de type `string`) du titre d'impression.
`copier` copie le dessin sur disque (dépend de l'ordinateur utilisé).

Sons :

`bip` émet un son bref.
`biiip` émet un son long.
`son`(f,t) émet un son de fréquence f (comprise entre 15 et 15000) pendant t secondes (f et t de type `real`, peut être interrompu par Esc, § ou #).

Musique :

`metronome`(n) règle le métronome à n battements par seconde (`largo` = 50, `larghetto` = 63, `adagio` = 71, `andante` = 92, `moderato` = 114, `allegro` = 144, `presto` = 184, `prestissimo` = 204).

`nuance`(n) règle la nuance n (`pianissimo` = 1, `piano` = 2, `mezzo_piano` = 3, `un_poco_piano` = 4, `sotto_voce` = 5, `mezza_voce` = 6, `un_poco_forte` = 7, `mezzo_forte` = 8, `forte` = 9, `fortissimo` = 10).

`note`(h,d) joue une note de hauteur $h = (((o * octave) + n) + a)$ où $o = -3$, -2, ..., 3 est l'octave de la note (`octave` = 12), n = `d0` ou `ut` = 0, `re` = 2, `mi` = 4, `fa` = 5, `sol` = 7, `la` = 9, `si` = 11 est le nom de la note, a = `double_bemol` $= -2$, `bemol` $= -1$, `becarre` = 0, `diese` = 1, `double_diese` = 2 est l'altération de la note et d = `note_carree` = 8.0, `ronde` = 4.0, `ronde_pointee` = 6.0, `blanche` = 2.0, `blanche_pointee` = 3.0, `blanche_en_triolet` = 4/3, `noire` = 1.0, `noire_pointee` = 1.5, `noire_en_triolet` = 2/3, `croche` = 0.5, `croche_pointee` = 0.75, `croche_en_triolet` = 1/3, `double_croche` = 0.25, `double_croche_pointee` = 0.375, `double_croche_en_triolet` = 0.5/3, `triple_croche` = 0.125, `triple_croche_pointee` = 0.1875, `triple_croche_en_triolet` = 0.25/3, `quadruple_croche` = 0.0625, ou `quadruple_croche_en_triolet` = 0.125/3 est la durée de la note.

`silence`(d) marque un silence de durée d = `baton_de_deux_pauses` = 8.0, `pause` = 4.0, `pause_pointee` = 6.0, `demi_pause_pointee` = 3.0, `demi_pause` = 2.0, `soupir` = 1.0, `soupir_pointe` = 1.5, `demi_soupir` = 0.5, `demi_soupir_pointe` = 0.75, `quart_de_soupir` = 0.25, `quart_de_soupir_pointe` = 0.375, `huitieme_de_soupir` = 0.125, `huitieme_de_soupir_pointe` = 0.1875, `seizieme_de_soupir` = 0.0625.

Interaction avec l'utilisateur du $\mathcal{R}\underset{\smile}{o}\underset{\smile}{o}t$:

`interaction`(`X`, `Y`, `Ch`, `Coordonnees`, `Pivotements`, `Dessin`, `FinSouris`, `Sauver`) retourne les coordonnées (x,y) du $\mathcal{R}\underset{\smile}{o}\underset{\smile}{o}t$ déplacé par l'utilisateur avec la souris ou les flèches de défilement ↑, ↓, ← ou → au moment où l'interaction avec l'utilisateur se termine.

L'interaction se termine quand le bouton de la souris est enfoncé (auquel cas la variable `Ch` de type `char` prend pour valeur le caractère nul `chr(0)`) ou quand une touche est tapée (le caractère correspondant étant affecté à `Ch`). Les coordonnées (x,y) du $\mathcal{R}^{o}_{b}{}^{o}t$ sont relatives à sa position au moment de l'appel de la procédure et comptées sur la grille. Elles sont affectées aux variables `X` et `Y` de type `integer`. L'interaction peut être interrompue par les touches Esc, § ou #.

– Si la valeur booléenne `Coordonnees` est `true` (respectivement `false`) alors les coordonnées courantes du $\mathcal{R}^{o}_{b}{}^{o}t$ déplacé par la souris ou les flèches de défilement sont (ne sont pas) affichées à l'écran pendant l'interaction.

– Si la valeur booléenne `Pivotements` est `true` alors l'utilisateur a la possibilité de faire pivoter interactivement le $\mathcal{R}^{o}_{b}{}^{o}t$ (touches – (`phtg`), G (`pqtg`), + (`phtd`) et D (`pqtd`)) avant la fin de l'interaction. Dans ce cas l'interaction peut se terminer uniquement en enfonçant le bouton de la souris ou la touche ↩.

– Si les valeurs booléennes `Pivotements` et `Dessin` sont égales à `true` alors l'utilisateur a la possibilité de dessiner avant la fin de l'interaction (touches A (`av`), ((`vg`),) (`vd`), L (`lc`), B (`bc`), P (`dp`), C (`dc`), . (`dg`) et * (changement de la couleur du crayon) en corrigeant les erreurs avec ←). Dans ce cas l'interaction peut se terminer en enfonçant le bouton de la souris ou la touche ↩.

– Si la valeur booléenne `FinSouris` est `true` alors l'utilisateur a la possibilité de terminer en enfonçant le bouton de la souris sinon il doit terminer en enfonçant une touche du clavier.

– Si les valeurs booléennes `Dessin` et `Sauver` sont `true` alors un programme `Dessin_du_robot.p` dont l'exécution reproduit le dessin interactif est engendré sur disque. S'il y a une erreur (disquette vérrouillée,...) alors `dessinDuRobotEngendre` sera égal à `false` après l'exécution de `interaction`.

Index

Les programmes du *Robot* sont disponibles sur la toile à l'adresse :
`http://www.di.ens.fr/~cousot/books/LE.shtml`

Bon de commande

Veuillez me faire parvenir les disquettes d'accompagnement du livre :

> "*Premières leçons de programmation en Turbo Pascal*"
> *Laurent, Patrick, Radhia et Thibault Cousot*
> *McGraw–Hill, Paris, 1991.*

Je joins mon paiement par chèque postal ou bancaire à l'ordre de McGraw-Hill.

Format des disquettes (cochez la ou les cases appropriées) :

- ☐ 800 K pour Turbo Pascal 1.1 sur Macintosh 512K, Plus, SE, Classique ou II de Apple ; 2 disquettes, 150 FF TTC.
- ☐ 5 pouces 1/4, format MS-DOS pour Turbo Pascal 5.5 ou 6.0 sur compatibles IBM PC ; 2 disquettes, 150 FF TTC.

A renvoyer à :

McGraw-Hill
28, rue Beaunier
75014 Paris

Nom .....

Prénom .....

Adresse .....

.....

Code Postal ..... Ville .....

.....

Offre valable jusqu'au 31 mars 1992.

IMPRIMERIE LOUIS-JEAN
BP 87 — 05003 GAP Cedex
Tél. : 92.51.35.23
Dépôt légal : 9 — Janvier 1991
Imprimé en France

Les auteurs de cet ouvrage utilisent le dessin comme support intuitif pour présenter sous une forme simple et originale les notions de base de la programmation en PASCAL [structuration des programmes en procédures, itérations bornées (`for`), expressions entières, déclarations de constantes, de paramètres (passés par valeur), tests (`if`,`case`), expressions booléennes, itérations non bornées (`while`, `repeat`), codages de l'information, expressions réelles, récursivité, déclarations de variables de types simples et affectations].

Du pilotage d'un "robot" qui se déplace sur l'écran pour exécuter des dessins stylisés jusqu'à la manipulation de concepts plus abstraits comme la récursivité, chacun, de l'écolier à l'adulte débutant, pourra progresser à son propre rythme jusqu'à développer des applications personnelles.

Les auteurs : Laurent Cousot est collégien ; Patrick Cousot est docteur ès sciences mathématiques, professeur d'informatique à l'École polytechnique, directeur du laboratoire d'informatique de l'École polytechnique ; Radhia Cousot est docteur ès sciences mathématiques, chargée de recherche au CNRS en informatique ; Thibault Cousot est écolier.

8 782704 212393

ISBN : 2-7042-1239-2
ISSN : 0-992-5880