Les boucles, comment s'en sortir ?

Les boucles sont avec les tests l'une des caractéristiques fondamentales des ordinateurs. Elles sont faites pour tourner, mais il faut qu'elles se terminent au bon moment.

■ Les boucles sont un outil de programmation très puissant : elles permettent de réaliser rapidement des calculs répétitifs sans que l'utilisateur ait à intervenir. De nombreux types d'algorithmes sont d'ailleurs réalisés de la sorte.

Le plus difficile à obtenir, lorsque l'on a mis en œuvre une boucle, c'est évidemment d'en sortir au moment où le résultat souhaité est atteint. Dans le n° 3 de *l'Op*, nous avons vu que l'on pouvait y parvenir en effectuant des tests dont le résultat arrêtait le « bouclage » en temps utile et branchait le pointeur vers une sortie assurant, par exemple, l'affichage du résultat.

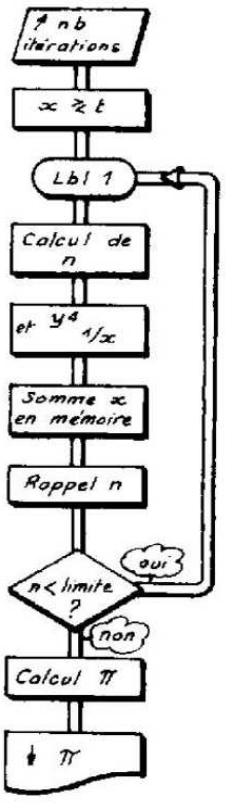
En règle générale, les raisons qui conduisent à sortir d'une boucle sont de deux ordres :

- ou bien l'on attend qu'un certain résultat, connu d'avance, soit obtenu ;
- ou bien l'on décide d'exécuter un certain nombre de tours de piste avant d'examiner le résultat.

00	35	y^x
01	04	4
02	45	÷
03	09	9
04	00	0
05	85	=
06	22	x ≷ t
07	86 1	2nd Lbl 1
08	01	1
09	34 1	SUM 1
10	33 1	RCL 1
11	35	y^x
12	04	4
13	85	=
14	25	$^1/x$
15	34 2	SUM 2
16	33 2	RCL 2
17	- 76	INV 2nd x ⩾ t
18	51 1	GTO 1
19	33 1	RCL 1
20	36	2nd Pause
21	33 2	RCL 2
22	36	2nd Pause
23	55	x
24	09	9
25	00	0
26	85	=
27	35	y^x
28	04	4
29	25	$^1/x$
30	85	=
31	81	R/S
32	- 19	INV 2nd Ct
33	81	R/S
34	71	RST

Fig. 1

Avec le premier programme, on entre pi avec la précision que l'on choisit (3,1 ou 3,141, etc.) et l'on apprend combien de tours de boucles sont nécessaires pour ce calcul. Avec le second programme, on indique le nombre de tours, et l'on obtient pi calculé en conséquence.



00	22	x ≷ t
01	86 1	2nd Lbl 1
02	01	1
03	34 1	SUM 1
04	33 1	RCL 1
05	35	y^x
06	04	4
07	85	=
08	25	$^1/x$
09	34 2	SUM 2
10	33 1	RCL 1
11	- 76	INV 2nd x ⩾ t
12	51 1	GTO 1
13	36	2nd Pause
14	33 2	RCL 2
15	36	2nd Pause
16	55	x
17	09	9
18	00	0
19	85	=
20	35	y^x
21	04	4
22	25	$^1/x$
23	85	=
24	81	R/S
25	- 19	INV 2nd Ct
26	81	R/S
27	71	RST

Fig. 2

Les boucles, comment s'en sortir ?

Dans l'un et l'autre cas, le test à effectuer n'est pas le même. Dans la première hypothèse, le test porte en effet sur le résultat de l'opération lui-même. Dans la seconde hypothèse au contraire, il faut ajouter au programme (s'il n'en contient pas déjà) un index de boucle qui n'est là que pour compter les tours. Quand cet index atteint la valeur désirée, la porte de sortie est ouverte : le test ne porte donc pas sur le résultat de l'opération, mais sur la valeur de l'index.

Une façon d'approcher pi

Un exemple va nous permettre de comprendre plus concrètement en quoi consiste cette différence. Ce sera l'occasion de découvrir (ou de retrouver) l'une des suites de Riemann grâce auxquelles on peut retrouver π avec diverses formes de calcul. C'est ainsi que

$$x_n = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \ldots + \frac{1}{n^4}$$

tend vers $\pi^4/90$ lorsque n tend vers l'infini. Autrement dit, plus n est grand, meilleure est l'approximation. C'est une des façons d'obtenir une valeur assez précise de π.

Dans le calcul d'une telle suite, on peut décider d'arrêter le programme lorsque π est obtenu avec 4 décimales par exemple. Nous adoptons alors la première façon de sortir d'une boucle : le test est effectué sur le résultat des calculs. Mais on peut aussi décider de sortir de la boucle après 50 ou 60 itérations pour examiner la valeur de x et en déduire l'approximation de π obtenue. Le compteur utilisera alors un compteur de boucles qui sera le plus souvent de la forme 1 SUM *n*. Le contenu de la mémoire *n* augmentera donc d'une unité à chaque passage. Très souvent d'ailleurs, on fera d'une pierre deux coups : l'index, autrement dit la mémoire *n*, sera utilisé dans le calcul même de la suite. Naturellement, il existe bien des façons d'écrire des programmes correspondant à ces différents algorithmes.

On peut se demander par exemple quelle valeur doit prendre x pour que π soit calculé avec un nombre déterminé de décimales, et entrer ensuite cette valeur dans le registre de test, registre *t* ; à l'intérieur de la boucle, le programme comparera le résultat de x avec ce nombre. Tant que ce résultat sera inférieur à la valeur contenue dans le registre *t*, le programme continuera à boucler, puis le pointeur sortira de la boucle et π sera calculé à partir de x après que le nombre d'itérations ait été affiché. Cette solution est représentée figure 1.

Si l'on veut au contraire effectuer un nombre d'itérations choisi d'avance, on utilisera la méthode de la figure 2. Après avoir entré en *t* le nombre de boucles désirées, on laisse le programme calculer la suite de *n*, puis x. C'est le rappel à l'affichage de *n* et sa comparaison avec *t* qui décident de la poursuite ou non de la boucle. Quand le pointeur sort de la boucle, *n* est affiché, puis x et enfin π tel qu'il a été estimé d'après x.

Que conclure de ces programmes ? Pour obtenir π avec 2 décimales, il suffit de 5 itérations. Pour 3 décimales, il faut 7 itérations. Les 4 décimales sont obtenues avec 14 passages. Pour la suite, je vous laisse le voir d'essayer, mais je vous conseille de brancher votre calculatrice sur le secteur si vous voulez poursuivre les calculs. Le nombre des tours de circuit augmente en effet beaucoup plus vite que celui des décimales !

Un compte-tours pour programmes

Nous venons de voir comment contrôler à l'intérieur d'un programme le nombre d'itérations grâce à un index. Cela fonctionne — et même fort bien. Toutefois, nous n'avons pas utilisé la TI 57 au mieux de ses possibilités : cette calculatrice peut en effet gérer elle-même l'index. Elle dispose pour cela d'une instruction spécifique : 2nd Dsz. Cette instruction utilise la mémoire 0, qui doit contenir l'index, et elle doit être placée à la fin de la boucle. A chaque passage du pointeur, le contenu de la mémoire 0 est décrémenté de l'unité (autrement dit diminué d'un point), puis la machine vérifie si la valeur du nouveau contenu de la mémoire 0 est ou non égale à zéro. Tant que ce contenu n'est pas nul, c'est l'instruction qui suit immédiatement le Dsz qui est exécutée (ici, un GTO qui renvoie au début de la boucle). Dès que la valeur contenue dans la mémoire 0 est nulle, l'instruction de branchement (GTO) qui suit le Dsz est sautée, enjambée, et l'on sort de la boucle. Ce type d'aiguillage est représenté schématiquement figure 3.

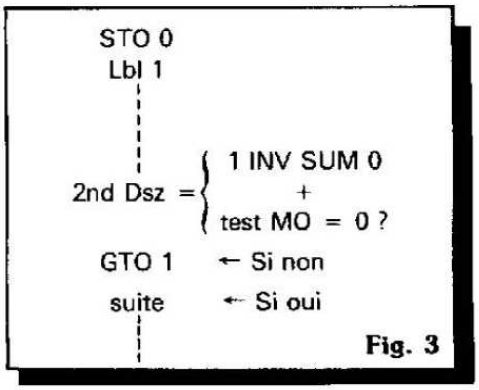


Fig. 3

Habituellement, on fait en sorte que le nombre contenu dans la mémoire 0 soit un entier : il n'est pas question bien sûr que le programme boucle 3 fois 1/4. Cela dit, si l'on introduit dans la mémoire 0 un nombre décimal, il sera interprété comme s'il était augmenté à la valeur entière supérieure. C'est la raison pour laquelle 9,59 en mémoire 0 provoquera 10 bouclages successifs. Le petit programme de la

00	32 0	STO 0	•	entrée en mémoire du contrôleur de boucle
01	86 1	2nd Lbl 1	•	étiquette
02	33 0	RCL 0	•	rappel du contenu de MO qui va décroître de 1
03	36	2nd Pause		à chaque passage
04	01	1	•	opération à l'intérieur de la boucle
05	34 1	SUM 1		
06	56	2nd Dsz	•	enlever 1 à MO et test : MO = 0 ?
07	51 1	GTO 1	•	si non aller à Lbl 1
08	33 1	RCL 1	•	si oui afficher le résultat du calcul en mémoire 1
09	81	R/S	•	arrêt
10	- 19	INV 2nd Ct	•	pour essayer avec d'autres nombres, entrer un
11	81	R/S		nouveau contrôleur de boucle
12	71	RST	•	retour à 00

Fig. 4

figure 4 vous permettra de voir comment opère le Dsz.

Pour commencer, entrer le programme en mode LRN. Faites RST et introduisez le contrôleur de boucle. Dans un premier temps, essayez 10 et appuyez sur R/S : vous voyez l'affichage passer de 10 à 1 puis le résultat de SUM 1 apparaître. Le contenu de la mémoire 0 étant devenu nul au dernier tour, le pointeur est sorti de la boucle et le programme s'est arrêté. Essayez maintenant avec 9,5 et vous constaterez que vous obtenez tout aussi bien 10 parcours de boucle : l'affichage décroît de 9,5

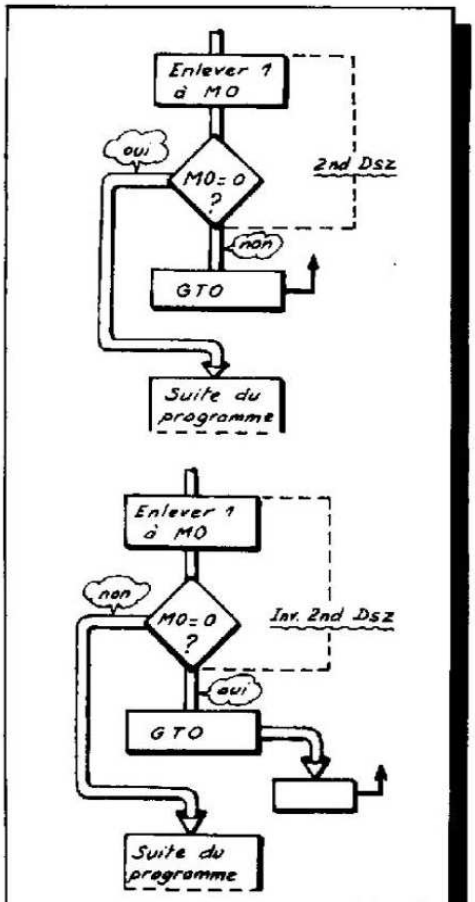


Fig. 5

à 0,5, mais à 0,5 le test conclut fort justement que la mémoire 0 n'est pas nulle et il renvoie le pointeur pour un dernier tour de piste.

Les applications de l'instruction Dsz sont très nombreuses et elles sont simples à mettre en œuvre. Une seule précaution à prendre : il ne faut utiliser la mémoire 0 à l'intérieur de la boucle qu'à bon escient.

L'instruction Dsz possède son inverse que l'on entre au clavier par INV 2nd Dsz et dont le fonctionnement est identique en ce qui concerne la décrémentation de la mémoire 0. Ce qui est inversé, c'est le branchement effectué après le test inclus dans l'instruction. Si la réponse à la question *M0=0 ?* est positive, le pas suivant est exécuté (et non pas enjambé). Si la réponse est « non », le pas suivant est sauté et le programme se poursuit. La figure 5 illustre la différence existant entre les deux instructions.

Nous n'avons parlé jusqu'ici que de ce qui se produisait lorsque la mémoire 0 contenait au départ une valeur positive. Si cette mémoire contient un nombre négatif avant le début de la boucle, l'effet obtenu est inverse : la valeur de M0 est incrémentée de l'unité à chaque passage sur Dsz ou INV Dsz jusqu'à ce que le contenu de la mémoire soit nul.

Avec tout ce que vous connaissez maintenant sur ces instructions de contrôle de boucles, vous êtes maintenant à même de concevoir toute sorte de programmes calculant des suites. Je vous suggère de vous entraîner en adaptant la suite de Riemann décrite au début de cet article. Vous en trouverez l'organigramme à la figure n° 6. Dans le même ordre d'idées, je vous

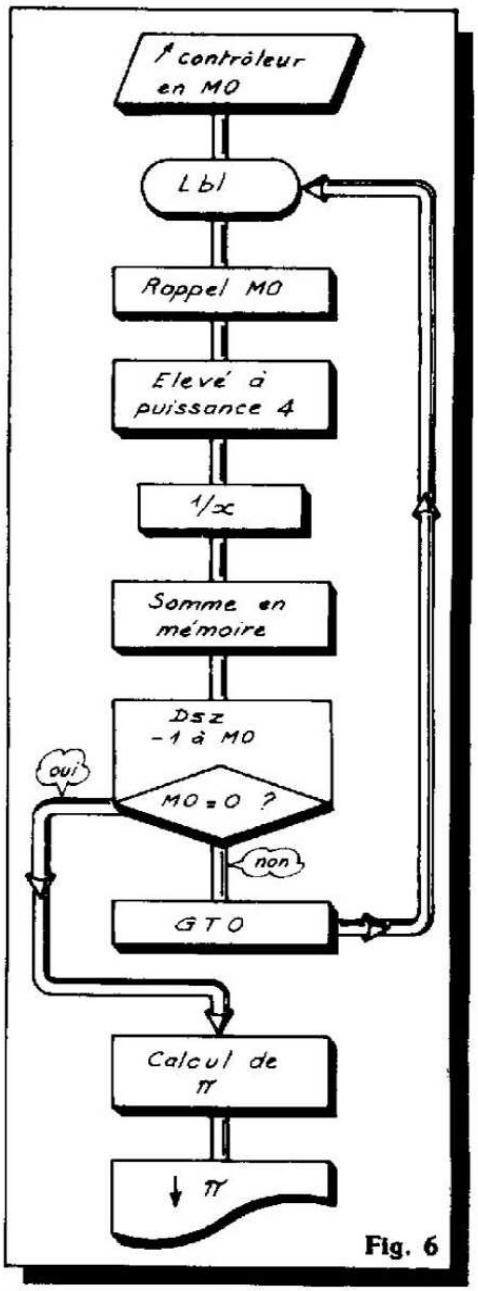


Fig. 6

propose un autre exercice : écrivez votre programme de calcul de factorielle en vous souvenant que 0! = 1, 1! = 1 et n! = n × (n − 1) × (n − 2) × ... × 3 × 2 × 1.

Bonnes boucles !

☐ Xavier de La Tullaye