Clarifiez vos programmes : faites des sous-routines !

Si vous devez écrire plusieurs fois la même suite d'instructions, gagnez du temps et de l'espace mémoire : apprenez à vous servir des sous-programmes. Cela vous obligera à plus de rigueur, et c'est tout bénéfice.

■ Dans les programmes que vous avez déjà écrits, il vous est certainement arrivé de retrouver la même séquence d'instructions écrite plusieurs fois. Gâchis de place, surtout lorsque les pas de programme sont peu nombreux (50 sur la TI 57).

Heureusement, il existe un remède efficace contre ce gaspillage de mémoire : on peut réutiliser plusieurs fois et de façon différente la même séquence à l'intérieur d'un programme. Effets secondaires et intéressants du remède : cela oblige à réfléchir à l'organigramme avant de concevoir le programme, et cela facilite ensuite la mise au point.

Aller simple — ou — aller-retour

Dans un précédent article, nous avons déjà étudié les branchements obligés réalisés à l'aide de l'instruction GTO. Nous savons maintenant que le pointeur, quand il rencontre cet ordre, se transporte jusqu'à l'étiquette balisant une portion de programme. Avec GTO cependant, aucun retour n'est prévu : le pointeur poursuit vers d'autres pas sans "se souvenir" d'où il vient.

La figure 1 représente ce type de branchement ; on remarquera que s'il est mis en œuvre de cette façon, tout bêtement, sans être combiné à des tests, il n'a pratiquement aucune utilité en programmation.

Dans le cas d'un sous-programme, le branchement est réalisé avec une autre instruction : sur TI 57, il s'agit de SBR comme SuB-Routine (sous-routine en français). Au début du branchement par SBR, les choses se passent apparemment de la même manière qu'avec GTO. Le pointeur est envoyé à l'étiquette dont le numéro suit le pointeur (Lbl 2, par exemple avec SBR 2). Mais en même temps, une mémoire interne à l'ordinateur, la mémoire de retour, enregistre le numéro du pas où était inscrit SBR.

La séquence d'instructions qui suit l'étiquette est alors exécutée jusqu'à ce que le pointeur rencontre une instruction spéciale, INV SBR, qui lui indique que le sous-programme est terminé et qu'il doit retourner là d'où il était venu.

C'est à ce moment-là que l'enregistrement du numéro de pas où se trouvait l'instruction SBR est utilisé ; le pointeur retourne au pas suivant pour continuer à exécuter le programme principal. On voit donc qu'à n'importe quel autre endroit du programme il pourra se trouver un autre appel du sous-programme avec un retour s'effectuant de la même manière.

Ce nouveau type de circuit est représenté à la figure 2. Au pas 10 sont inscrits SBR et le numéro de l'étiquette du sous-programme, en l'occurrence celui qui occupe les

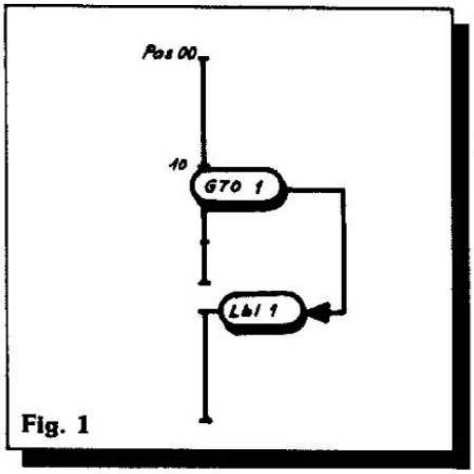

Fig. 1

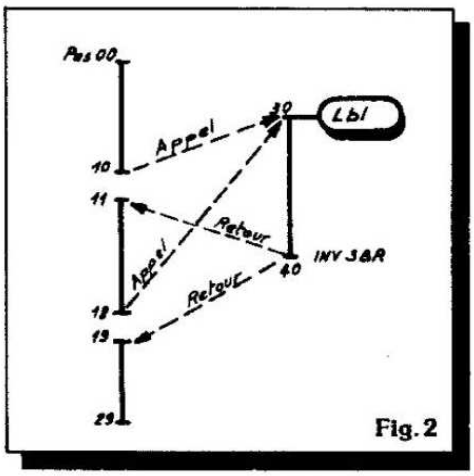

Fig. 2

lignes 30 à 40. Le pas 10 à partir duquel le branchement est demandé est placé en mémoire et le sous-programme est exécuté jusqu'à ce que le pointeur rencontre INV SBR à la ligne 40 : fin du sous-programme.

La mémoire de retour renvoie alors le pointeur au pas 11 où se poursuit le programme principal jusqu'à la nouvelle instruction SBR du pas 18. La mémoire de retour enregistre 18 et le pointeur repart sur l'étiquette du pas 30. En rencontrant de nouveau INV SBR, il retourne au programme principal, mais cette fois-ci au pas 19 pour en poursuivre l'exécution jusqu'au pas 29.

Et si nous prenions un exemple ?

Nous avons maintenant une bonne idée des trajets effectués par le pointeur lorsqu'il va faire un détour vers un sous-programme. Il nous reste à examiner tout cela de façon plus concrète en décortiquant un programme qui utilise ces nouvelles instructions. A cette fin, nous allons réaliser un jeu de dés avec la TI 57. L'ordinateur devra tirer deux dés et présenter le résultat sous la forme d'un nombre décimal : premier dé avant la virgule, second dé après.

Naturellement, comme il s'agit de dés, les chiffres tirés devront être compris entre 1 et 6 et former une suite aussi imprévisible que possible. Nous aurons donc recours à un générateur de nombres pseudo-aléatoires. Ce générateur est classique, et il pourra être réutilisé dans beaucoup d'autres occasions.

Le nombre-source, que l'on appelle parfois *semence* est ici un nombre fractionnaire quelconque, c'est-à-dire compris entre 0 et 1 exclus. Il sert à amorcer le générateur. On le choisit lui aussi au hasard de façon à ne pas retomber deux fois sur la même série de nombres. En effet, les mêmes semences appliquées à la formule mathématique du générateur conduisent une à une aux mêmes séries, et comme le but recherché est de simuler le hasard...

Dans un premier temps, le nombre-source est ajouté à pi (3,141...) et la somme est élevée à la puissance 5. On prend alors la partie fractionnaire du nombre obtenu,

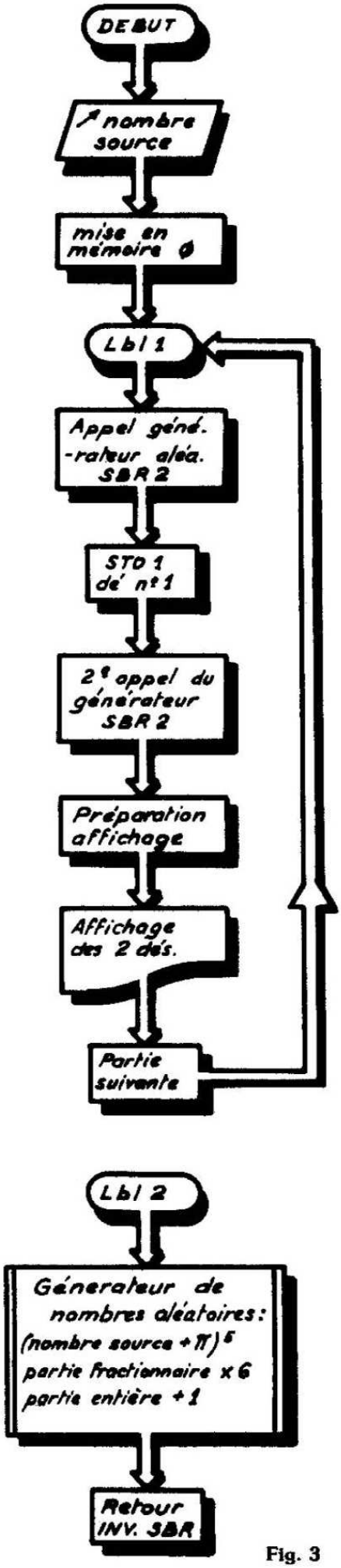

Fig. 3

que l'on conserve (elle servira de nombre-source pour le prochain tirage), et on la traite en faisant en sorte d'en obtenir un nombre compris dans l'intervalle désiré. Pour notre jeu, nous avons besoin de nombres allant de 1 à 6 ; on multiplie donc la partie fractionnaire par 6, on en extrait la partie entière à laquelle on ajoute 1 et l'on a le résultat recherché. Ce générateur donne une distribution assez homogène et il est tout à fait convenable pour des applications telles que la nôtre.

Comme nous avons décidé de lancer deux dés à chaque fois, c'est justement ce générateur qui sera organisé en sous-programme. On l'appellera une première fois dans le cours du programme principal et le nombre obtenu sera mis en mémoire. Un nouvel appel fournira la valeur du second dé qui, divisée par 10, sera ajoutée à la première valeur, et il ne restera plus qu'à afficher le tout.

La structure du programme est donc très simple : elle a été représentée sous forme d'organigramme à la figure 3. Vous remarquerez de quelle façon je symbolise un sous-programme. Etant donné que cette partie de programme est indépendante, il est plus simple de la dessiner en marge du programme principal : l'organigramme y gagne en lisibilité.

Nous pouvons maintenant regarder dans le détail ce qui est indispensable à l'insertion d'un sous-programme dans le corps du programme.

Dans le programme principal tout d'abord, nous devrons avoir obligatoirement des instructions d'appel. Ce sera pour la TI 57 SBR suivi d'un chiffre de 0 à 9 qui correspond à l'étiquette repérant le début du sous-programme (2nd Lbl 0 à 9). On placera cette instruction à chaque endroit où l'on veut faire exécuter le tirage au sort d'un nombre.

Autre élément indispensable : on doit soigneusement isoler le programme principal, surtout si le ou les sous-programmes sont placés à la fin de la liste. On évitera ainsi d'exécuter le sous-programme de façon intempestive. Le plus simple est de faire en sorte que le programme principal se termine par R/S, mais il y a d'autres façons de procéder que vous pouvez imaginer (RST, par exemple, peut très bien faire l'affaire).

Voyons maintenant ce que les

Fig. 4 – Liste du programme de dés

Programme principal				Commentaires
00	32	0	STO 0	Entrée du nombre-source et mise en mémoire
01	86	1	2nd Lbl 1	Etiquette pour renvoi en fin de programme
02	61	2	SBR 2	Appel sous-programme
03	32	1	STO 1	Mise en mémoire du lancer de dé
04	61	2	SBR 2	2ème appel
05	75		+	Le deuxième nombre est additionné au
06	33	1	RCL 1	premier après qu'il ait été divisé par 10
07	45		÷	
08	01		1	A l'affichage les deux dés seront présentés
09	00		0	sous la forme A, B
10	85		=	
11	81		R/S	Arrêt pour affichage
12	51	1	GTO 1	Lancer de dés suivant
Sous-programme				**Commentaires**
13	86	2	2nd Lbl 2	Etiquette du début de sous-programme
14	33	0	RCL 0	Rappel du nombre-source
15	75		+	+
16	30		π	3,14159
17	85		=	
18	35		y^x	Elevé à la puissance 5
19	05		5	
20	85		=	
21	-49		INV 2nd Int	Extraction de la partie fractionnaire
22	32	0	STO 0	Mise en mémoire pour servir comme nombre-
23	55		X	source suivant. La partie fractionnaire est
24	06		6	multipliée par 6. Cela donne un nombre
25	85		=	compris entre 0 et 5,9...
26	49		INT	... dont on ne garde que la partie entière
27	75		+	pour y ajouter 1.
28	01		1	Le résultat final sera bien compris entre 1 et 6
29	85		=	
30	-61		INV SBR	Fin de sous-programme, retour au programme principal

sous-programmes doivent nécessairement comporter. En premier lieu, il nous faudra une étiquette pour repérer le début du sous-programme, et une instruction INV SBR qui en indique la fin. Entre ces deux bornes, le sous-programme proprement dit ne comporte rien de particulier : il est écrit comme le reste de la liste à quelques restrictions près toutefois.

C'est ainsi que RST (renvoi au premier pas) sera proscrit dans presque tous les cas, car il ne renvoie pas seulement le pointeur au début de la ligne, mais il efface aussi les mémoires de retour.

Par ailleurs, les transferts directs (GTO) devront être maniés avec précaution : ils ne pourront guère être utilisés que pour effectuer des branchements à l'intérieur même de la sous-routine.

Enfin, dans le cas de calculs avec parenthèses, il faudra surveiller de près le signe =, car il peut arriver qu'il déclenche l'exécution de calculs commencés sous parenthèses dans le programme principal.

Regardons maintenant, en figure 4, la liste de notre jeu de dès. On s'aperçoit que la routine est en fait plus longue que le programme principal et c'est logique puisque c'est elle qui fait l'essentiel du travail. Le programme principal ne réalise en fait que les appels et prépare les deux nombres pour leur affichage conjoint.

On aurait d'ailleurs pu confier cette dernière tâche à un autre sous-programme qui n'aurait été appelé qu'une fois. Cela nous donne une idée d'une technique d'organisation des programmes qui consiste à n'utiliser le programme principal que comme le lieu d'appel d'une série de routines.

Certes, cette méthode est plus dépensière en pas de programme, mais elle permet de mettre au point les routines indépendamment les unes des autres. Elle présente aussi l'avantage de la clarté, puisque le programme principal est alors construit de façon très structurée. De plus, il devient possible de se constituer une bibliothèque de sous-programmes qui pourront chacun résoudre un type particulier de problèmes et qui seront réutilisables.

Avec une routine de génération de nombres aléatoires et une autre qui effectue le calcul de la distance entre deux points, il devient par exemple facile de réaliser rapidement un jeu de bataille navale.

A la figure 5, on voit une autre version de notre jeu de dés ; il a été remanié de telle sorte que le programme principal ne consiste plus qu'à organiser l'appel des sous-routines. Cette modification coûte 3 pas de mémoire, mais il y aurait économie de pas si l'on décidait de tirer 3 dés au lieu de 2, et l'adaptation se trouverait simplifiée.

Pour tester un sous-programme ou pour l'utiliser indépendamment du programme principal, il est possible sur la TI 57 de l'appeler au clavier, comme on le fait en mode LRN, en appuyant sur la touche SBR et sur le numéro de l'étiquette concernée.

Fig. 5 – Une autre version du jeu de dés

00	32	0	STO 0	┐
01	86	1	2nd Lbl 1	│
02	61	2	SBR 2	├ Début identique à celui du premier programme
03	32	1	STO 1	│
04	61	2	SBR 2	┘
05	61	3	SBR 3	Appel du sous-programme de préparation de
06	51	1	GTO 1	l'affichage
07	86	2	2nd Lbl 2	┐
08	33	0	RCL 0	├ Sous-programme générateur de nombres aléatoires identique à ce qu'il était. Seuls les numéros de pas changent
24	-61		INV SBR	┘
25	86	3	2nd Lbl 3	Etiquette : début du sous-programme de
26	75		+	préparation d'affichage
27	33	1	RCL 1	Le contenu est identique à ce qu'il était
28	45		÷	mais c'est maintenant un morceau de
29	01		1	programme autonome qui peut être testé
30	00		0	séparément
31	85		=	
32	81		R/S	
33	-61		INV SBR	Retour au programme principal

Jusqu'à présent, nous n'avons parlé que des routines appelées par un programme principal, mais une routine peut en appeler une autre. On parle alors en terme de niveau : le sous-programme de premier niveau appelle un sous-programme de deuxième niveau. On ne peut pas aller plus loin avec la TI 57 qui ne dispose en effet que de deux mémoires de retour. On trouvera figure 6 un schéma représentant ces deux niveaux de sous-programmes.

Au pas 10, le programme principal appelle le 1er niveau de sous-programme qui lui-même appelle au pas 25 le second niveau renvoyant le pointeur au pas 31. Le retour s'effectue lui aussi en deux étapes : au pas 38, INV SBR renvoie le pointeur en 26, et donc au premier niveau, puis au pas 30 INV SBR déclenche à son tour le retour au programme principal.

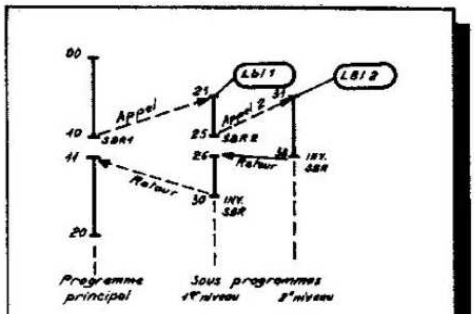

Fig. 6

Rien n'empêche de faire plus compliqué en donnant, par exemple, au sous-programme Lbl 2 le rôle de routine du premier niveau pour le programme principal. Les détours deviennent alors moins faciles à suivre ; ils sont représentés à la figure 7 : la routine Lbl 2 est exécutée d'abord comme étant du second niveau, puis du premier.

Nous avons vu que les sous-programmes autorisent une grande souplesse ; ils demandent seulement un peu d'entraînement et du soin dans la réalisation de l'organigramme. Mais le bénéfice que l'on en tire est important : puissance et

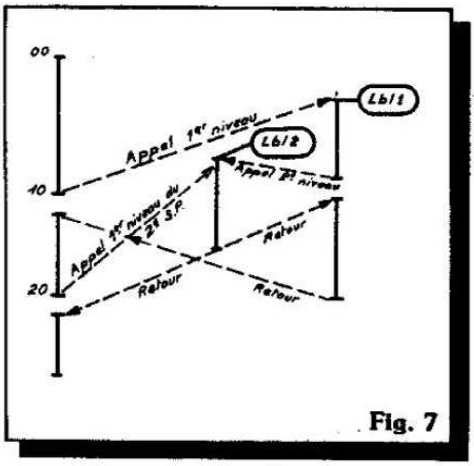

Fig. 7

simplification dans l'écriture des instructions, le véritable travail se trouvant reporté dans la phase initiale, celle de la conception du programme.

Mais tout est là, car c'est cette phase de conception qui fait pratiquement tout l'intérêt de la programmation, le reste n'étant qu'une affaire de routine.

☐ Xavier de La Tullaye