ÉLÉMENTS DE PROGRAMMATION EN PL/I

INITIATION et PRATIQUE

PAR

Gerald M. WEINBERG

Directeur à l'Institut IBM européen de Recherches sur les Systèmes

TRADUIT PAR

Jean BERNARD

Ingénieur des Arts et Manufactures

DUNOD
PARIS
1971

Traduction de l'ouvrage publié
en langue anglaise sous le titre

PL/I Programming Primer

par McGRAW-HILL BOOK COMPANY

PRÉFACE

Ce livre est destiné à servir de texte pour un cours d'introduction à la programmation des ordinateurs. Il est aussi conçu comme une initiation à PL/I : tout nouveau langage a besoin d'un ouvrage qui en facilite l'abord et lui donne des chances d'être largement accepté. Le développement escompté de PL/I, de son emploi et de ses compilateurs, incite à choisir ce langage comme base d'un cours de programmation. Mais les qualités propres du langage sont encore plus déterminantes, car PL/I est l'un des premiers langages qui promette d'être raisonnablement universel, tout au moins en ce qui concerne les applications existantes à ce jour.

L'universalité de PL/I le rend particulièrement attrayant comme support des cours de plus en plus nombreux qu'enseignent les universités pour initier les étudiants à la programmation. Ces cours viennent en complément d'un éventail toujours plus étendu de disciplines de base. L'instructeur se trouve en face, non seulement de mathématiciens, de physiciens, d'astronomes et d'ingénieurs, mais aussi de biologistes, de linguistes, de médecins, d'anthropologistes, d'économistes, de psychologues, d'historiens, de sociologues et d'administrateurs. Le texte d'un tel cours doit être suffisemment général pour retenir l'intérêt de tous ces étudiants. Ses exemples et ses problèmes ne doivent pas exiger de formation spéciale dans l'une ou l'autre de ces disciplines, car seul l'instructeur peut fournir les exemples particuliers qui conviennent exactement à la composition de sa classe. Au contraire, le livre doit fournir la fonction d'usage général sur laquelle il puisse construire son cours personnel.

Le livre est organisé en dix chapitres ; chacun d'eux est découpé en sections suivies d'exercices. Les modules de cette organisation sont prévus pour que l'instructeur puisse facilement intégrer le texte dans ses cours, voire dans des travaux de laboratoire. Dans bien des cas, le texte et les exercices

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION

1.1 Calculateur réel et calculateur idéal

Les calculateurs sont des machines qui traitent quelque chose appelée « information ». Comme il y a bien des sortes d'information, il y a aussi beaucoup de traitements différents exécutés par les calculateurs. A peu près tout calculateur numérique peut exécuter, en théorie, n'importe quel traitement d'information à la portée d'un quelconque autre calculateur. Ainsi, si nous avons quelque information que nous désirons traiter, le problème principal est de dire au calculateur ce qu'il doit faire, plutôt que de choisir un calculateur spécialement conçu pour chaque traitement.

En pratique cependant, les calculateurs sont assez loin d'être universels, pour un certain nombre de raisons : les machines réelles font occasionnellement des erreurs ; les machines réelles ne peuvent pas emmagasiner effectivement une quantité infinie d'informations ; les machines réelles prennent un certain temps pour exécuter chaque partie d'un traitement, et le temps peut être critique. Ainsi, nous pourrons avoir à spécifier, non seulement le traitement que nous voulons faire exécuter par un calculateur réel, mais encore des procédures qui surmonteront toute déficience que le calculateur pourrait présenter.

Une autre différence entre calculateur réel et idéal réside dans les diverses manières de spécifier la « même » procédure. Un calculateur idéal utilisant deux procédures identiques fournirait les mêmes résultats (sortie), étant donnée la même information à traiter (entrée). Mais les calculateurs réels font quelquefois des erreurs. La procédure A et la procédure B peuvent produire la même sortie pour la même entrée, simplement parce que l'une d'elles (ou les deux) a été exécutée avec une erreur en un certain point. Inversement, si elles ne produisent pas la même sortie, nous ne pouvons en déduire qu'elles ne sont pas identiques, car la différence peut être due à une

erreur. Les calculateurs réels prennent aussi du temps pour mener à bien leurs procédures, et différentes procédures peuvent exiger des temps différents pour produire les mêmes résultats.

Enfin, les procédures établies pour un calculateur réel peuvent différer des procédures convenables pour un calculateur idéal. Rarement nous trouverons deux procédures qui soient identiques au sens vrai, c'est-à-dire qui produisent des résultats identiques, dans des temps identiques, avec une égale sensibilité aux erreurs. D'autre part, des traitements différents ne le seront pas toujours de manière significative, car celà dépend de nos exigences. Par exemple, si nous avons besoin de nos résultats dans le courant de la semaine prochaine, nous ne considérons pas que des temps de traitement de sept et quatorze secondes présentent une différence significative (à moins que le coût de sept secondes de traitement soit très élevé), car tous deux satisfont nos exigences. De plus, nous ne considérons pas comme significative une différence entre des temps de traitement de sept et de quatorze *années* (quels qu'en soient les coûts), car ni l'un ni l'autre ne satisfait nos exigences. Naturellement, la différence entre sept et quatorze *jours* est significative, vu les circonstances.

L'action de spécifier la procédure à exécuter par le calculateur s'appelle programmation ; la procédure elle-même s'appelle le *programme*. Sur la plan de la conception, nous pouvons diviser le programme en deux parties, la partie concernant la procédure idéale, et la partie concernant l'adaptation de cette procédure à une machine réelle. Rarement, voire jamais, ces deux parties sont distinguées explicitement; il y a bien des situations où il n'est pas nécessaire de porter beaucoup d'attention au fait que le calculateur n'est pas idéal. Pour apprendre comment spécifier des procédures, comment écrire des programmes, il est avantageux de commencer avec des situations idéales, car la grande part des complications dans les programmes ne survient que si le problème met à l'épreuve la capacité des machines réelles. Bien des programmeurs, la plupart peut-être, ne rencontreront jamais une telle situation, ou si rarement qu'ils pourront alors s'offrir l'assistance d'un professionnel. Ce livre est destiné à ces programmeurs-là.

Même après avoir mis de côté les complications en provenance des limitations du calculateur, il nous reste des difficultés certaines du fait de nos propres limitations. Même le calculateur scientifique idéal n'est pas idéal autant que nous pourrions le souhaiter. En particulier, nous devons spécifier ce que nous voulons qu'il fasse, et nous devons être explicites dans cette spécification. Il est à présumer qu'un calculateur vraiment idéal n'aurait même pas à être instruit que vous voulez traiter quelque information ; il préviendrait vos désirs, trouverait les données nécessaires, et vous fournirait le résultat juste au moment où vous en ressentez le besoin, de la même façon que le domestique idéal prépare les habits de son maître sans qu'on lui dise

quoi ou quand. Moins idéal, mais encore très idéal, serait le calculateur que vous pourriez commander en disant simplement : « Traitez ces données, voulez-vous ; vous savez ce que je veux ».

Dans un bureau, le chef de service peut avoir plusieurs collaborateurs capables de répondre à un ordre aussi vague. Pourquoi un calculateur serait-il incapable d'en faire autant ?

En fait, il n'y a aucune raison valable pour qu'un calculateur ne puisse faire ce qu'un bon employé peut réaliser. L'employé ne lit pas la pensée de son chef, il se rappelle seulement des situations semblables passées pour lesquelles il avait reçu des instructions détaillées. Il se souvient qu'il s'agit de la fin de mois, et voyant les bons mensuels, il réalise que son patron désire qu'il procède à la répartition mensuelle du travail. Un calculateur peut en faire autant, s'il a reçu précédemment le programme qui calcule cette distribution du travail, s'il l'a stocké avec l'annotation appropriée disant quand et par qui cela sera demandé. En fait, le calculateur peut être programmé pour fournir un tel état au responsable, automatiquement à la fin de chaque mois ; et souvent il est ainsi programmé.

En utilisant ainsi un calculateur, nous n'avons pas réellement éliminé le besoin de spécifier explicitement les procédures, pas plus que nous n'avons éliminé le besoin de former les employés. Nous profitons seulement d'efforts passés qui ont été fournis pour faciliter les spécifications quotidiennes. S'il n'existait que quelques milliers de procédures différentes qu'on veuille faire exécuter par un calculateur, elles pourraient toutes être programmées et stockées une fois pour toutes. Au lieu d'avoir à apprendre à programmer, tout ce que nous aurions à faire serait d'apprendre à se servir de la liste de tous les programmes et ce qu'ils traitent. Malheureusement, ce n'est pas aussi simple. Par ailleurs, ce n'est pas aussi pénible que cela pourrait l'être.

Bien qu'il y ait des millions ou des milliards de procédures que différentes personnes souhaiteraient faire exécuter à différents moments, l'expérience a montré que beaucoup de ces procédures partagent un grand nombre de caractéristiques communes. Par exemple, dans un très grand nombre de problèmes de traitement d'information, on veut additionner deux nombres. Dans un groupe de problèmes sensiblement plus petit, on veut prendre les chiffres de deux nombres, et produire un nouveau nombre en entrelaçant ces chiffres (12345 et 67890 produiraient 1627384950 par exemple). En fait, il se peut que personne n'ait jamais désiré réaliser cet entrelacement, bien que nous ne puissions exclure cette éventualité dans le futur. En étudiant les programmes du passé, nous pouvons apprendre ce que les gens font souvent. Alors nous pouvons écrire une fois pour toutes des programmes qui rendent ces choses aussi faciles que possible. En agissant ainsi, nous pouvons rendre impossible ou non la réalisation de choses moins communes. Naturellement, nous n'aimons pas les interdire absolument. Dans certains cas, il peut nous arriver de faciliter certaines choses peu courantes ; nous

n'y voyons pas d'inconvénient — à moins que ces choses peu courantes soient très probablement des erreurs.

1.2 Langages de programmation

Une des fonction du langage dans la vie humaine est de donner des ordres ou des instructions à d'autres. Dans ce sens, l'utilisation du langage est très voisine de la programmation. Si c'était possible, nous aimerions pouvoir utiliser le langage ordinaire pour commander au calculateur, juste comme nous l'utilisons pour commander à d'autres personnes. En fournissant des programmes appropriés, nous pouvons pratiquement permettre au programmeur de commander au calculateur dans un langage qui paraît très semblable au langage humain ordinaire, bien qu'une grande partie de la ressemblance soit une illusion. Même si ce n'était pas une utopie, cependant, ce ne serait pas nécessairement la meilleure solution que d'utiliser le langage courant dans la programmation. En particulier, tout langage traduit une adaptation. Les choses les plus fréquemment dites deviennent les plus faciles à dire. Mais les choses que nous disons le plus souvent dans notre langue naturelle ne sont pas de cette sorte que nous disons normalement aux calculateurs. (« Allo », « Embrasse-moi », « Dieu vous bénisse »)

Sans parler des calculateurs, nous ne trouvons même pas la langue « naturelle » appropriée à nos interactions avec d'autres personnes. De nombreux langages spécialisés se sont développés. Les gens de loi parlant aux gens de loi, les médecins aux médecins, les ingénieurs aux ingénieurs, ou les mathématiciens aux mathématiciens, ne parlent pas la langue naturelle mais des variantes plus ou moins spécialisées qui simplifient leurs communications ordinaires. Bien entendu, ils doivent subir quelque apprentissage de ces langages spéciaux, mais l'investissement s'amortit rapidement plusieurs fois. Les mathématiques, pour prendre l'exemple le plus frappant, bégayeraient complètement si les mathématiciens étaient empêchés d'employer leurs moyens d'expression particuliers.

La nécessité de communiquer avec les calculateurs n'est pas présente depuis aussi longtemps que la nécessité de résoudre les problèmes mathématiques. Néanmoins, un énorme volume d'efforts a déjà été dépensé en essayant de simplifier les tâches de communication des procédures aux calculateurs. Et puis, aussi, nous avons comme bases ces autres langages spécialisés, particulièrement celui des mathématiques. Nous avons déjà parcouru un long chemin depuis le temps où, pour commander au calculateur l'addition de deux nombres, nous étions forcés d'écrire quelque chose comme :

```
00101100000011001110101010101010110101
11011111001001001000000111010110011 1
10100100110100000011011111110010101 1
```

ou même quelque chose comme :

```
CLA A
FAD B
STO C
```

Aujourd'hui, de nombreux langages sont disponibles qui réduisent la commande de l'addition à la notation d'apparence mathématique :

C = A+B ;

et nombre de tentatives couronnées de succès ont étendu la puissance et la simplicité de tels « langages de programmation ». Comme les langues naturelles, les langages de programmation évoluent et se développent au cours d'essais continuels, de modifications, d'emprunts d'éléments utiles à d'autres langages. Un livre intéressant pourrait être écrit sur la « glottochronologie » des langages de programmation, bien que nous n'ayons pas ici de place pour un tel sujet.

De même que se sont développés les langages techniques spécialisés, de même des langages de programmation spéciaux ont été élaborés pour des domaines particuliers d'activité, tels que le génie civil, le calcul des systèmes optiques, l'analyse des réseaux électriques, les analyses linguistiques ; il existe même des langages pour écrire des programmes qui traduisent d'autres langages de programmation dans le système de codage propre au calculateur. Dans ce livre, cependant, nous négligerons ces langages, et concentrerons notre attention sur un seul qui est appelé *Programming Language* I, ou PL/I en abrégé.

Le nom « Programming Language I » implique par sa généralité que PL/I n'est pas destiné à la spécification d'un type particulier de procédure. Au contraire, il prétend être adapté à la spécification d'à peu près n'importe quelle procédure. Cette généralité en fait le langage de programmation à choisir pour qui ne veut en connaître qu'un seul, tout comme l'anglais ou le français peut être la langue unique la plus utile au voyageur. Cependant si vous n'avez qu'une seule série de problèmes bien particuliers à traiter, ou tout juste un problème unique, un langage plus spécialisé, s'il en existe un dans votre domaine, peut être préférable, tout comme l'*enga* peut être la langue à connaître si vous devez passer le restant de votre vie dans une certaine partie des hauts plateaux de la Nouvelle Guinée.

Le « I » de PL/I n'est autre que le chiffre romain *un* (il subsiste encore de ces vieilles notations encombrantes). Il implique que PL/I est le premier d'une série de langages qui se développeront probablement en même temps que notre connaissance des langages de programmation. Comme en biologie dans l'évolution des espèces, le premier d'une série de langages peut aussi être considéré comme le dernier d'autres séries ; c'est aussi le cas de PL/I. Sous cet aspect, il possède bien des traits dont la présence est difficile à comprendre sans connaître les langages qui l'ont précédé. De place en place dans le texte, nous essaierons d'apprécier cette évolution.

1.3 Organisation du livre

La suite du livre est divisée en chapitres, eux-mêmes subdivisés en sections. Le sujet est disposé de manière à développer les idées dans une séquence naturelle, à utiliser et à réutiliser celles qui sont les plus importantes. En d'autres termes, il s'agit d'un ouvrage d'enseignement et non d'un ouvrage de référence. Il peut être très difficile de localiser la réponse à une question spécifique, bien que l'index doive y aider. Au lieu de rechercher ainsi la réponse à des questions de détail, le lecteur aurait intérêt à se procurer le manuel standard de référence du langage.

Chaque section d'un chapitre est construite pour présenter une ou deux nouvelles idées et pour développer une ou deux idées présentées dans une section précédente. Du fait que chaque section est basée sur toutes les précédentes, il est préférable de ne passer à la suivante qu'après avoir maîtrisé suffisemment la section courante. Pour aider le lecteur à savoir s'il a maîtrisé ou non une section, chacune est suivie d'un « contrôle ». Ces contrôles de connaissances sont faits d'exercices courts ou de questions qui ne devraient pas présenter de difficulté au lecteur s'il a maîtrisé le sujet. Lorsque des difficultés apparaissent, il est préférable de reprendre la section et de la relire soigneusement.

En plus des exercices de contrôle, des exercices de programmation sont présentés, toutes les deux sections en moyenne. Ces exercices sont particulièrement utiles au lecteur qui dispose d'un calculateur équipé pour PL/I, car il peut alors valider (ou invalider) ses solutions. A défaut, un instructeur ou un collègue qui puisse lire et critiquer les exercices rendra les mêmes services qu'un calculateur, avec moins d'efficacité. Même si ni l'une ni l'autre de ces méthodes de contrôle n'est disponible, le lecteur aura encore intérêt à essayer d'écrire les programmes suggérés. Tout au moins, il découvrira de temps à autre qu'il n'a aucune idée de la manière de traiter une partie d'un exercice, auquel cas une nouvelle lecture des sections précédentes est recommandée.

Tout au long des sections, nous avons essayé de faciliter la tâche du débutant, en évitant de discuter des détails ou des exceptions. Pour le programmeur déjà expérimenté qui veut apprendre PL/I, ces omissions seront plus souvent une gêne. Le seul moyen d'éviter ce défaut eût été de parsemer le livre de notes en pied de page, notes qui risquent d'embrouiller le novice. Nous avons choisi de ne pas le faire, en considérant que le programmeur déjà expérimenté peut se tirer d'affaire tout seul, tandis que le débutant a besoin de toute l'aide disponible. Si le livre est utilisé dans un cours où le niveau des élèves l'exige, l'instructeur peut facilement compléter chaque section avec les détails correspondants tirés du manuel de référence.

ou même quelque chose comme :

```
CLA A
FAD B
STO C
```

Aujourd'hui, de nombreux langages sont disponibles qui réduisent la commande de l'addition à la notation d'apparence mathématique :

C = A+B ;

et nombre de tentatives couronnées de succès ont étendu la puissance et la simplicité de tels « langages de programmation ». Comme les langues naturelles, les langages de programmation évoluent et se développent au cours d'essais continuels, de modifications, d'emprunts d'éléments utiles à d'autres langages. Un livre intéressant pourrait être écrit sur la « glottochronologie » des langages de programmation, bien que nous n'ayons pas ici de place pour un tel sujet.

De même que se sont développés les langages techniques spécialisés, de même des langages de programmation spéciaux ont été élaborés pour des domaines particuliers d'activité, tels que le génie civil, le calcul des systèmes optiques, l'analyse des réseaux électriques, les analyses linguistiques ; il existe même des langages pour écrire des programmes qui traduisent d'autres langages de programmation dans le système de codage propre au calculateur. Dans ce livre, cependant, nous négligerons ces langages, et concentrerons notre attention sur un seul qui est appelé *Programming Language* I, ou PL/I en abrégé.

Le nom « Programming Language I » implique par sa généralité que PL/I n'est pas destiné à la spécification d'un type particulier de procédure. Au contraire, il prétend être adapté à la spécification d'à peu près n'importe quelle procédure. Cette généralité en fait le langage de programmation à choisir pour qui ne veut en connaître qu'un seul, tout comme l'anglais ou le français peut être la langue unique la plus utile au voyageur. Cependant si vous n'avez qu'une seule série de problèmes bien particuliers à traiter, ou tout juste un problème unique, un langage plus spécialisé, s'il en existe un dans votre domaine, peut être préférable, tout comme l'*enga* peut être la langue à connaître si vous devez passer le restant de votre vie dans une certaine partie des hauts plateaux de la Nouvelle Guinée.

Le « I » de PL/I n'est autre que le chiffre romain *un* (il subsiste encore de ces vieilles notations encombrantes). Il implique que PL/I est le premier d'une série de langages qui se développeront probablement en même temps que notre connaissance des langages de programmation. Comme en biologie dans l'évolution des espèces, le premier d'une série de langages peut aussi être considéré comme le dernier d'autres séries ; c'est aussi le cas de PL/I. Sous cet aspect, il possède bien des traits dont la présence est difficile à comprendre sans connaître les langages qui l'ont précédé. De place en place dans le texte, nous essaierons d'apprécier cette évolution.

1.3 Organisation du livre

La suite du livre est divisée en chapitres, eux-mêmes subdivisés en sections. Le sujet est disposé de manière à développer les idées dans une séquence naturelle, à utiliser et à réutiliser celles qui sont les plus importantes. En d'autres termes, il s'agit d'un ouvrage d'enseignement et non d'un ouvrage de référence. Il peut être très difficile de localiser la réponse à une question spécifique, bien que l'index doive y aider. Au lieu de rechercher ainsi la réponse à des questions de détail, le lecteur aurait intérêt à se procurer le manuel standard de référence du langage.

Chaque section d'un chapitre est construite pour présenter une ou deux nouvelles idées et pour développer une ou deux idées présentées dans une section précédente. Du fait que chaque section est basée sur toutes les précédentes, il est préférable de ne passer à la suivante qu'après avoir maîtrisé suffisemment la section courante. Pour aider le lecteur à savoir s'il a maîtrisé ou non une section, chacune est suivie d'un « contrôle ». Ces contrôles de connaissances sont faits d'exercices courts ou de questions qui ne devraient pas présenter de difficulté au lecteur s'il a maîtrisé le sujet. Lorsque des difficultés apparaissent, il est préférable de reprendre la section et de la relire soigneusement.

En plus des exercices de contrôle, des exercices de programmation sont présentés, toutes les deux sections en moyenne. Ces exercices sont particulièrement utiles au lecteur qui dispose d'un calculateur équipé pour PL/I, car il peut alors valider (ou invalider) ses solutions. A défaut, un instructeur ou un collègue qui puisse lire et critiquer les exercices rendra les mêmes services qu'un calculateur, avec moins d'efficacité. Même si ni l'une ni l'autre de ces méthodes de contrôle n'est disponible, le lecteur aura encore intérêt à essayer d'écrire les programmes suggérés. Tout au moins, il découvrira de temps à autre qu'il n'a aucune idée de la manière de traiter une partie d'un exercice, auquel cas une nouvelle lecture des sections précédentes est recommandée.

Tout au long des sections, nous avons essayé de faciliter la tâche du débutant, en évitant de discuter des détails ou des exceptions. Pour le programmeur déjà expérimenté qui veut apprendre PL/I, ces omissions seront plus souvent une gêne. Le seul moyen d'éviter ce défaut eût été de parsemer le livre de notes en pied de page, notes qui risquent d'embrouiller le novice. Nous avons choisi de ne pas le faire, en considérant que le programmeur déjà expérimenté peut se tirer d'affaire tout seul, tandis que le débutant a besoin de toute l'aide disponible. Si le livre est utilisé dans un cours où le niveau des élèves l'exige, l'instructeur peut facilement compléter chaque section avec les détails correspondants tirés du manuel de référence.