Le présent fichier PDF présente des extraits du document

Le langage PL/1
Tome 1
Eléments fondamentaux
1971

Auteur : Marcus Dornbusch, de la société IBM – International Business Machines
Editeur : Dunod, Paris, France

Ce document de 414 pages, est disponible sous le numéro **AC-22335-01** dans l'inventaire de la collection de l'ACONIT, Association pour un conservatoire de l'informatique et de la télématique, Grenoble.

Ce document daté de 1971, est écrit en français, par un français membre de la société IBM-France, et il vise à populariser, auprès des informaticiens de langue française, un langage de programmation informatique PL/1 (Programming Language – 1) que la société IBM veut généraliser.

Ce langage de haut-niveau a été conçu à partir de 1964 par des utilisateurs d'ordinateurs IBM.
Ce langage PL/1 veut être un langage « universel », unissant les avantages des langages FORTRAN (orienté calculs mathématiques), ALGOL (orienté logique et algorithmes) et COBOL (orienté gestion).
Ce langage PL/1 n'a pas eu le succès rapide escompté.

Par ses efforts de promotion de ce langage en 1971, la société IBM, alors leader des ventes et services en informatique, voulait démocratiser l'usage de l'informatique et partant augmenter ses ventes, en particulier sur ses produits phare les IBM 360 et 370, même s'il ne s'agissait pas forcément alors d'augmenter ses parts de marché déjà très élevées.

Le présent PDF est une copie des pages de ce document avec :

-la page de couverture et la page d'entête,
-l'Avant-propos (en 5 pages), et un encart d'**avertissement**,
-la table des matières (4 pages) : on y trouve les pages énonçant des exercices à programmer, et les pages dans lesquelles se trouvent les réponses. L'ouvrage se veut très pédagogique, avec de très nombreux exemples,
-l'annexe (en 16 pages) indiquant la manière de déclarer les variables, leur longueur, la manière dont doivent être gérés les arrondis, les opérations de conversion entre types de numération, la gestion des fractions.

-un texte ne provenant pas de ce document, essaye de répondre à la question **: le langage PL/1 a-t-il été utile ?**

Fiche rédigée le 19/11/2019 par Jean Ricodeau.

Marcus Dornbusch

le langage PL/1

tome 1 éléments fondamentaux

Dunod

LE LANGAGE

PL/1

Marcus DORNBUSCH
Ancien élève de l'Ecole normale supérieure

TOME 1

Éléments fondamentaux

DUNOD
PARIS
1971

AVANT-PROPOS

Au commencement, il y avait FORTRAN et COBOL...

Depuis l'année 1950 où il devint certain que les ordinateurs allaient connaître un développement considérable, leur complexité s'est sans cesse accrue en même temps que l'éventail des applications où ils trouvaient leur juste place s'est largement ouvert.

Dès 1956, les constructeurs comprirent que la diffusion des ordinateurs était limitée par leur difficulté d'emploi : pour fonctionner, un ordinateur doit exécuter les instructions d'un programme enregistré en langage machine, programme se présentant sous la forme d'une suite de zéros et de uns. L'apprentissage d'un tel langage est laborieux, puisqu'il nécessite la connaissance préalable des structures logiques de la machine et que sa mise en œuvre, même pour des spécialistes, est restée toujours très délicate. A cette date apparaît le langage ASSEMBLEUR qui, par l'emploi de codes mnémoniques pour représenter les instructions et de variables pour repérer les adresses, simplifie la tâche de ces spécialistes mais n'augmente guère leur nombre puisque la connaissance parfaite du fonctionnement interne de la machine est toujours requise.

Vers 1956, IBM met au point le premier langage extérieur, FORTRAN (FORmula TRANslator), qui d'une part permet aux scientifiques de rédiger dans ce langage proche de celui des mathématiques classiques des programmes relativement simples, et d'autre part supprime pour ces utilisateurs l'essentiel de l'apprentissage du fonctionnement interne des ordinateurs. FORTRAN connaît un succès immédiat tant auprès des utilisateurs scientifiques que des autres constructeurs. Il évolue, souvent de façon anarchique, et se stabilise vers 1963 autour de FORTRAN IV.

Parallèlement, un langage COBOL (COmmon Business Oriented Language) est apparu sur le marché, strictement réservé aux applications de gestion (au sens qu'on leur donnait en 1960). Il a connu un développement moins passionné que celui de FORTRAN, dû certainement au fait que COBOL est employé par des programmeurs de métier pour lesquels l'analyse de l'application est déjà faite, alors que FORTRAN est souvent utilisé par l'analyste lui-même (et c'est, en général, une personne au bagage universitaire bien rempli) qui, plus que le résultat du programme, cherche une façon élégante d'obtenir ce résultat.

L'université n'est pas restée absente de ces développements et dès 1959 apparaît ALGOL (ALGOrithmic Language) dont les aspects nobles et séduisants ainsi que la rigueur du formalisme n'ont pas manqué d'impressionner les utilisateurs scientifiques.

En 1963, le point sur les langages extérieurs conduit aux réflexions suivantes :

— ils sont nombreux (près d'une centaine) ;

— chacun est spécialisé (même si, au prix d'acrobaties, on peut par exemple écrire des applications de gestion en FORTRAN) alors que les ordinateurs ne le sont plus (l'ère des ordinateurs scientifiques ou des ensembles de gestion est révolue) ;

— chaque langage existe sous des formes multiples et, le plus souvent, incompatibles entre elles (il n'y a pas un FORTRAN mais une dizaine de FORTRAN IV en plus de celui qui est normalisé) ;

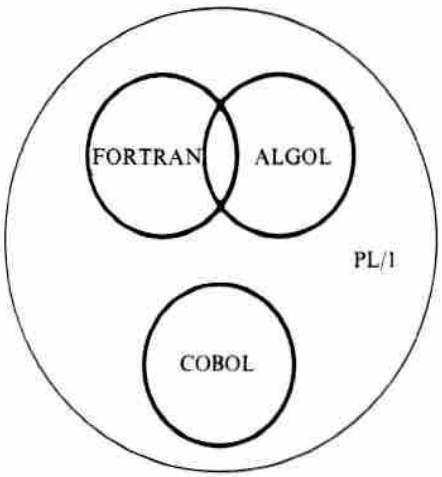
— des fonctions nouvelles, disponibles sur les ordinateurs (gestion fine des interruptions, télétraitement, etc.) ne peuvent être gérées par ces langages ;

— les principaux langages restent d'un accès difficile à une large couche d'utilisateurs potentiels : le traitement des applications non numériques (documentation automatique, banques de données, etc.) ne se fait que par des langages très spécialisés.

*
* *

Enfin PL/1 vint...

Conçu en 1964 par un groupe d'utilisateurs auquel se joint IBM, PL/1 (Programming Language n° 1) peut se situer par rapport aux autres langages à l'aide du schéma suivant :



De façon générale, PL/1 contient ce qu'il y avait de meilleur dans les langages existants et offre des fonctions nouvelles :

— PL/1 n'est pas un langage spécialisé. Tout ce qui s'effectuait de façon simple en FORTRAN, COBOL ou ALGOL reste simple (ou devient plus simple) en PL/1 ;

— PL/1 est un langage modulaire : il n'est nul besoin de le connaître entièrement pour l'employer efficacement ;

— PL/1 permet de réaliser tout ce que le langage ASSEMBLEUR permet de réaliser. Plus on progresse dans l'apprentissage de PL/1, plus on se rapproche du traitement en mémoire ;

— PL/1 est un langage extensible. Il se prête à l'addition de fonctions nouvelles ;

— PL/1 contient en lui-même les outils de mise au point du programme (car rédiger un programme, c'est s'engager à le mettre au point).

*
* *

La traversée du désert...

Le premier compilateur PL/1 est mis en service par IBM en 1966. L'enfant est né avant terme, ses premiers pas sont hésitants et maladroits, mais il ne déçoit pas ses premiers adeptes. Une fois passées les maladies de jeunesse, ses qualités évidentes apparaissent à la plus grande satisfaction de ses utilisateurs. Il est alors l'objet d'une intense campagne de dénigrement, orchestrée surtout par ceux-là même qui n'avaient pas pris la peine de l'examiner de près. Toutes ses qualités lui sont reprochées en bloc. On le traite de « ramassis de FORTRAN, COBOL et ALGOL » alors qu'il contient effectivement le meilleur de ces langages. On l'accuse d'être géré par un compilateur encombrant, alors que sur la série 360/OS,

le compilateur PL/1 est le moins encombrant de tous les compilateurs. On le suspecte de produire un code résultant vaste et peu performant, alors qu'il offre la possibilité de choisir librement pour un encombrement mémoire réduit aux dépens de la vitesse d'exécution ou réciproquement. Enfin et surtout, on en fait le cheval de bataille d'IBM (« passer à PL/1, c'est se lier à IBM ») alors que, d'une part, ce sont essentiellement les utilisateurs qui ont conçu PL/1, IBM n'ayant fait que suivre leurs recommandations, et que, d'autre part, FORTRAN était effectivement un produit IBM, ce qui n'empêcha personne de le copier.

Pendant les années 1968 et 1969, les conférences où l'on traite de langages sont curieusement animées par des groupes qui n'utilisent pas PL/1 et en disent le plus grand mal et par des groupes d'utilisateurs satisfaits de PL/1 où l'on s'étonne de l'animosité des premiers.

PL/1 acquiert ses lettres de noblesse dans l'université où, à l'abri des passions et des querelles, on commence à l'enseigner et à s'en servir (il est tellement plus simple d'enseigner un seul langage là où on devait en enseigner deux). En 1970, toutes les enquêtes faites sur l'utilisation de PL/1 convergent : ses utilisateurs sont satisfaits (le coût global d'une application, depuis la formation du programmeur jusqu'à la maintenance du programme est singulièrement réduit avec PL/1) et leur nombre va sans cesse croissant. La plupart des constructeurs annoncent alors (ou laissent entendre) qu'ils offriront un compilateur PL/1.

*
* *

Une normalisation, pour quoi faire ?

Dans de nombreux pays, il existe un comité chargé de définir une norme de PL/1, c'est-à-dire un ensemble en dessous duquel le produit ne devrait pas s'appeler PL/1. Il est important de noter que la normalisation d'un langage était, jusqu'à présent, un constat à posteriori. Ce n'est qu'après que la majorité des constructeurs ont livré un compilateur FORTRAN qu'on s'est soucié de le normaliser, de façon à mettre de l'ordre dans les spécifications multiples et contradictoires rencontrées dans ses compilateurs. Les diverses normalisations ne se sont pas faites sans mal car elles ont parfois entraîné un effort important des modifications de certains compilateurs existants. Pour PL/1, on tombe dans le cercle vicieux suivant : ou bien il est normalisé, et c'est contraindre les constructeurs hésitants à produire un compilateur, ou bien il n'est pas normalisé, et certains organismes ne peuvent pas l'utiliser librement. D'autre part, ceux qui sont décidés à écrire un compilateur ne savent pas à quelles normes se vouer.

*
* *

POUR UN ENSEIGNEMENT COHÉRENT DE L'INFORMATIQUE

Chaque membre du Corps Enseignant, quelle que soit sa discipline, est convaincu que l'informatique doit prendre sa place dans les programmes d'enseignement secondaire. Laissons aux spécialistes le soin de conclure si l'informatique est, ou non, une science. Reconnaissons que c'est une matière qui ne préjuge en rien les qualités dont on peut faire preuve dans les différentes disciplines. De façon plus précise, on peut être doué en mathématiques et se montrer hermétique à l'informatique. Inversement, les cas ne sont pas rares où ceux qui n'ont fait que des études très moyennes se révèlent de brillants informaticiens.

Contrairement à une opinion communément répandue, l'initiation à l'informatique, ce n'est pas nécessairement le langage binaire, pas plus que le schéma de principe du fonctionnement de l'additionneur. C'est encore moins la description, même sommaire, d'une unité de contrôle. De façon générale,

l'initiation à l'informatique ne doit pas commencer par des cours de technologie. Pour prendre un exemple, combien de candidats au permis de conduire ne rebuterait-on pas en exigeant d'eux la description détaillée du fonctionnement d'un carburateur ou en leur demandant d'admirer le principe du différentiel ? Certes, un informaticien de métier ne devra pas manquer d'avoir suivi ces cours de technologie, mais il aura au préalable choisi sa spécialisation.

Ensuite, lors d'une initiation à l'informatique, on ne doit pas chercher à écrire des programmes très performants ou le moins encombrants possible ou encore très astucieux, de la même façon qu'il n'est pas nécessaire de battre des records de vitesse lorsqu'on apprend à conduire. L'optimisation des programmes doit être confiée à des spécialistes chevronnés.

En fait, l'initiation à l'informatique ne peut se faire que par l'initiation à la programmation. De même qu'on n'enseigne pas la littérature à des analphabètes, il ne peut pas être question de comprendre quelles sont les possibilités, et donc les limites d'emploi, d'un ordinateur sans être passé par les balbutiements de la programmation. Et il n'y a pas de solution miraculeuse en la matière, il n'existe pas de langage-qu'on-apprend-en-deux-jours-et-sans-peine, car la programmation suppose une somme de connaissances qu'on ne peut pas réinventer. Il existe soit des langages spécialisés, mais qui ne montrent qu'un aperçu des possibilités de la programmation, soit des langages vastes, tels que PL/1, mais on peut être tenté de reprocher leur ampleur. Il est aisé de dégager de PL/1 un sous-ensemble fondamental, contenu dans ce tome 1, et sans la connaissance duquel il n'est pas raisonnable de chercher à progresser. Plus tard, et en fonction des orientations de chacun, on pourra parfaire ses connaissances dans les domaines spécialisés qu'offre PL/1 : gestion des fichiers, traitement des listes, analyse numérique fine, télétraitement, etc. Ce tome 1 cherche à être, en quelque sorte, un permis de programmer. On trouvera dans ce livre plus que le sous-ensemble fondamental, car il est apparu utile de laisser entrevoir quelques-unes des applications réalisables à l'aide de PL/1.

Nous conseillons au lecteur de ne consacrer aux six premiers chapitres qu'une première lecture et de s'attarder sur les chapitres VII et VIII, de ne puiser dans le chapitre IX que les fonctions pour lesquelles il se sent concerné, en revenant alors sur les premiers chapitres.

Certains pourront s'étonner du nombre relativement important des exercices et problèmes. C'est, qu'en fait, l'initiation à la programmation se fait essentiellement par mimétisme. Ce n'est qu'après avoir souvent utilisé un module de programme dans des conditions voisines que l'on comprend réellement le fonctionnement des instructions. Nous ne saurions trop conseiller au lecteur de soumettre à l'ordinateur des programmes s'écartant très peu de ceux qui figurent dans ce livre. En modifiant les bornes d'un tableau ou le jeu des données, il découvrira certaines anomalies qui lui permettront de mieux comprendre comment se déroule un programme. Enfin, le lecteur devra se garder d'exposer toute solution qu'il n'aurait pas soumise en machine. Aucun spécialiste ne se hasarde à dire d'un programme qu'il est correct tant que l'ordinateur n'a pas émis son verdict.

PARIS 1971.

AVERTISSEMENT

Ce livre est un ouvrage d'initiation à la programmation PL/1-F, tel que ce langage est défini sur les séries 360/OS et 370/OS IBM. La rigueur a souvent été sacrifiée au profit d'une recherche de simplification et de clarté. Bien que tous les exemples cités aient été traités en ordinateur, ce livre ne contient en aucune manière un engagement sur les spécifications du langage PL/1.

TABLE DES MATIÈRES

Début de l'annexe (en 16 pages, de page 399 à page 414)

ANNEXE

I. Affectation d'une valeur à un identificateur arithmétique

La zone réservée en mémoire à un identificateur arithmétique peut, schématiquement, être représentée à l'aide d'un filtre dont la forme dépend des attributs de cet identificateur.

1) Attributs `DECIMAL FIXED` .

Le filtre à la forme

`S EEEEEFF`

où l'on distingue

— une position réservée au signe `S`
— des positions réservées aux chiffres constituant la partie entière du nombre `EEEEE`
— des positions réservées aux chiffres constituant la partie fractionnaire du nombre `FF`

Les positions entières et fractionnaires sont indiquées par l'attribut de précision.

Le nombre total des positions réservées aux chiffres (partie entière + partie fractionnaire) est un nombre impair.

L'affectation d'une valeur à un identificateur doté des attributs `DEC FIXED` s'obtient en faisant coïncider le point fractionnaire de la valeur avec la virgule du filtre.

Exemple 1 : `DCL A DEC FIXED(7,2);`

Cette instruction définit le filtre

`S EEEEEFF`

L'instruction `A=+27543.71;` remplit la zone ainsi :

`+ 2754371`

Lorsqu'il n'y a pas assez de chiffres pour la partie entière, des zéros non significatifs sont insérés à gauche. Lorsqu'il n'y a pas assez de chiffres pour la partie fractionnaire, des zéros sont insérés à droite :

`A=-25.83;`	→	`- 0002583`
`A=-37415.2;`	→	`- 3741520`
`A= 3.1;`	→	`+ 0000310`
`A=635;`	→	` 0063500`

Lorsqu'il y a trop de chiffres pour la partie entière, ceux de gauche sont perdus lors de l'affectation. Lorsqu'il y a trop de chiffres pour la partie fractionnaire, ceux de droite sont perdus lors de l'affectation :

`A=1234567.89;`	→	`+ 3456789`
`A=12345.6789;`	→	`+ 1234567`
`A=7654321.1234;`	→	`+ 5432112`

Lors d'une affectation, les deux cas peuvent se présenter :

```
A= 12.3456;      →  + 0001234
A=-123456.7;     →  - 2345670
A= 1000000.1;    →  + 0000010
A=-0.00001;      →  - 0000000
```

Exemple 2 : `DCL B DEC FIXED(5,0);`

Cette instruction définit le filtre

```
S EEEEE
```

d'où il résulte que `B` *ne peut prendre que des valeurs entières*

```
B=-12345;        -  12345
B=+27;           +  00027
B=31.14;         +  00031
B=-0.001;        -  00000
B=1234567;       +  34567
B=-123456.78;    -  23456
```

Rappelons que l'instruction `DCL B DEC FIXED;` dote, par défaut, l'identificateur `B` de l'attribut de précision `(5,0)`, ce qui en fait un nombre entier qui peut valoir, en valeur absolue, de 0 à 99999.

Exemple 3 : `DCL C DEC FIXED(6,3);`

Comme le nombre total des chiffres demandé est pair (on demande 6 chiffres dont 3 à droite du point fractionnaire), le filtre a la forme

```
S EEEEFFF
```

car on réserve un nombre impair de positions pour les chiffres. A noter que la position supplémentaire est située à gauche.

```
C=123.456;       →  + 0123456
C=5743.778;      →  + 5743778
```

L'effet de cette déclaration est donc très voisin de

```
DCL C DEC FIXED(7,3);
```

Remarque. L'anomalie `SIZE` (voir le chapitre des anomalies) permet de détecter la perte de chiffres significatifs lors d'une affectation.

2) Attributs BINARY FIXED
Le filtre a la forme

S EEEEE,FFF

où l'on distingue

— une position réservée au signe S
— des positions réservées aux chiffres binaires constituant la partie entière du nombre EEEEE
— des positions réservées aux chiffres binaires constituant la partie fractionnaire du nombre FFF

Le nombre total des positions réservées aux chiffres (partie entière + partie fractionnaire) est soit 15, *soit* 31.

Si, dans l'attribut de précision, le nombre total des chiffres binaires demandé est inférieur ou égal à 15, on réserve 15 positions numériques dans le filtre. S'il est supérieur à 15, on réserve 31 positions. L'affectation d'une valeur à un identificateur doté des attributs BIN FIXED s'obtient en faisant coïncider le point fractionnaire de la valeur avec la virgule du filtre.

Exemple 1 : `DCL U BIN FIXED(8,3);`

Cette instruction définit le filtre

S EEEEEEEEEEE,FFF

L'instruction `U=10111.011B;` remplit la zone ainsi :

\+ 00000010111,011

les divers cas d'inégalité de longueurs entre la valeur qu'on affecte et la zone réceptrice sont traités comme pour les identificateurs dotés des attributs DEC FIXED .

`U=-10.101B;`	− 0000000000010,101	(− 2,625)
`U=+10101.1B;`	\+ 0000000101011,00	(21,5)
`U=-10.1B;`	− 0000000000010,100	(− 2,5)
`U=11111111.101B;`	\+ 0000011111111,101	(127,625)
`U=1.10101B;`	\+ 0000000000001,101	(1,625)
`U=-0.00001B;`	− 0000000000000,000	(0)

Exemple 2 : `DCL V BIN FIXED(7,0);`

Cette instruction définit le filtre

S EEEEEEEEEEEEEE,

d'où il résulte que V *ne peut prendre que des valeurs entières.*

`V=-1111111B;`	− 0000000001111111,	(− 127)
`V=+101B;`	\+ 0000000000000101,	(5)
`V=-101.11B;`	− 0000000000000101,	(− 5)
`V=+0.101B;`	\+ 0000000000000000,	(0)
`V=-10101010101B;`	− 0000001010101010,	(− 341)
`V=10110011001.1011B;`	\+ 0000101100110011,	(1 433)

Rappelons que l'instruction `DCL V BIN FIXED;` dote, par défaut, l'identificateur `V` de l'attribut de précision `(15,0)`, ce qui en fait un nombre entier qui peut aller de − 32768 à + 32767.

Exemple 3 : `DCL W BIN FIXED(20,2);`

Cette instruction définit le filtre

`S EEEEEEEEEEEEEEEEEEEEEEEEEEEEEE FF`

et l'instruction `W=1.11B;` le remplit ainsi :

`+ 00000000000000000000000000000011 11`

Quelques considérations sur les identificateurs dotés de l'attribut `FIXED` : Qu'il ait l'attribut `DECIMAL` ou l'attribut `BINARY`, lorsqu'un identificateur `FIXED` est doté, par défaut, d'un attribut de précision `(5,0)` ou `(15,0)`, c'est nécessairement un entier. Cet état des choses peut être soit une source d'erreur pour le programmeur soit une façon simple de manipuler les nombres entiers et les nombres premiers en particulier

Exemple 1 :

```
DCL I DEC FIXED;
DO I=1 TO 10 BY 0.1;
   PUT DATA(I);
END;
```

Dans cette séquence, le programmeur espère faire varier `I` de 1 à 10 avec un pas de variation de 0,1. En fait, ce programme boucle pour la raison suivante : le filtre défini pour `I` est

`S EEEEE`

Lorsqu'on active la boucle, on exécute l'instruction

`I=1;` → `+ 00001`

qui affecte `I` de la valeur 1, et on imprime `I=1;`. Lorsqu'on retourne à `DO`, on exécute

`I=1+0.1;` c'est-à-dire

`I=1.1;` → `+ 00001`

qui affecte à nouveau `I` de la valeur 1, et on imprime `I=1;`. La boucle `DO` ne devant s'arrêter que lorsque `I` dépasse 10, il y a un bouclage.

A noter (ce qui est le cas courant) que lorsque `I` ne fait pas l'objet d'une déclaration explicite, par contexte et par défaut il est doté des attributs `BINARY FIXED(15,0)`, ce qui conduit, une fois les conversions faites, au même bouclage. Pour que le programme précédent soit correct, il suffit de remplacer la première instruction par

```
DCL I DEC FIXED(3,1);
```

Exemple 2 : Voici une façon de savoir si un nombre I est divisible par un nombre J. Si c'est le cas, on donne à un indicateur N la valeur 1. Dans le cas contraire, on lui donne la valeur 0.

```
① DCL (I,J,K) DEC FIXED;
②    I=15;
③    J=3;
④    K=I/J;
⑤    IF I=K*J THEN  N=1;
⑥                   ELSE  N=0;
⑦    PUT DATA(N);
```

La déclaration définit trois filtres

```
I S EEEEE,
J S EEEEE,
K S EEEEE,
```

Une fois exécutée l'instruction de la ligne ④, les filtres sont remplis ainsi :

```
I + 00015,
J + 00003,
K + 00005,
```

L'expression logique `I=K*J` est vraie, puisque 15 = 5 × 3. `N` prend la valeur 1.
Changeons de valeurs :

```
① DCL (I,J,K) DEC FIXED;
②    I=17;
③    J=3;
④    K=I/J;
⑤    IF I=K*J THEN N=1;
⑥                 ELSE N=0;
⑦    PUT DATA(N);
```

Pour exécuter l'instruction de la ligne ④, on a

```
K=17/3;                 I + 00017,
K=5.6666666666;         J + 00003,
                        K + 00005,
```

qui affecte `K` de la valeur 5.

L'expression logique `I=K*J` est fausse, puisque 17 = 5 × 3 est faux. `N` prend la valeur 0.

De façon générale, ou bien `I/J` est une division exacte (au sens des nombres entiers), et la valeur affectée à `K` n'est pas tronquée. En multipliant cette valeur par `J`, on retrouve `I`. On bien `I/J` n'est pas une division exacte, et la valeur affectée à `K` est la partie entière de cette valeur. En multipliant cette valeur par `J`, on ne retrouve pas `I`.

Exemple 3 : Voici une façon d'isoler les chiffres d'un nombre entier

```
① DCL Z DEC FIXED(3),
      (U,D,C) DEC FIXED(1);
②    Z=845;
③    U=Z;
④    D=Z/10;
⑤    C=Z/100;
```

La déclaration définit 4 filtres

```
Z S EEE,
U S   E,
D S   E,
C S   E,
```

A la ligne ②, on remplit `Z` → `Z + 845,`

A la ligne ③, on obtient → `U + 5,` , puisqu'on fait coïncider le point décimal et la virgule du filtre.

A la ligne ④, on exécute `D=84.5;` → `D + 4,`

et à la ligne ⑤, `C=8.45;` → `C + 8,`

Exercice 1. Deux tableaux sont remplis conformément au schéma ci-contre.
Calculer les restes obtenus en divisant les couples de termes de même rang.

I(1)	00101	J(1)	00003
I(2)	05432	J(2)	00011
I(3)	68527	J(3)	05871
I(4)	10000	J(4)	00020
I(5)	00100	J(5)	03574
I(6)	02394	J(6)	02394

Solution :

```
DCL (I(6) DEC FIXED INIT(101,5432,68527,10000,100,2394),
     J(6) DEC FIXED INIT(3,11,5871,20,3574,2394),
    (K,RESTE) DEC FIXED;
  DO N=1 TO 6;
   K=I(N)/J(N);
   RESTE=I(N)-K*J(N);
   PUT SKIP DATA(I(N),J(N),RESTE);
  END;
```

On obtient les résultats suivants :

```
I(1)=  101  J(1)=    3  RESTE=    2;
I(2)= 5432  J(2)=   11  RESTE=    9;
I(3)=68527  J(3)=5871  RESTE=3946;
I(4)=10000  J(4)=   20  RESTE=    0;
I(5)=  100  J(5)=3574  RESTE=  100;
I(6)= 2394  J(6)=2394  RESTE=    0;
```

Exercice 2. Le nombre 209 est-il premier ?

Solution. Nous divisons successivement 209 par tous les nombres entiers impairs qui le précèdent. Si l'on obtient un reste égal à zéro, c'est que le nombre n'est pas premier

```
P:PROC OPTIONS(MAIN);
    DCL (I,J,K,RESTE) DEC FIXED;
        I=209;
        DO J=3 TO 207 BY 2;
        K=I/J;
        RESTE=I-K*J;
        IF RESTE=0 THEN GO TO Z;
        END;
    PUT LIST('209 EST PREMIER');
    GO TO FIN;
  Z:PUT LIST('209 EST DIVISIBLE PAR',J);
FIN:END P;
```

Lors de l'exécution, pour J égal à 3, on a

```
K=209/3;
```

c'est-à-dire

```
K=69.222222222;
```

qui donne à K la valeur 69. Puis `RESTE=209-69*3;` c'est-à-dire `RESTE=2;` . Comme RESTE ne vaut pas zéro, on saute le groupe THEN , et on retourne à DO où J prend les valeurs 5, 7 et 9 où il en est de même. Lorsque J prend la valeur 11, on a `K=209/11;` c'est-à-dire `K=19;` puis

```
RESTE=209-19*11;
```

c'est-à-dire `RESTE=0;` . On entre dans le groupe THEN qui renvoie à l'étiquette Z où on imprime : `209 EST DIVISIBLE PAR 11` .

— **3)** Attributs DECIMAL FLOAT

Le filtre a la forme

```
S CCMMMMMM
```

où l'on distingue

— une position réservée au signe S
— une zone réservée à la caractéristique CC
— des positions réservées à la mantisse MMMMMM

Il s'agit d'une forme flottante hexadécimale. Lorsque le filtre contient `+ CCABCDEF` , il représente le nombre

$$16^{CC-64} \times \left(\frac{A}{16} + \frac{B}{16^2} + \frac{C}{16^3} + \frac{D}{16^4} + \frac{E}{16^5} + \frac{F}{16^6} \right)$$

A, *B*, *C*, *D*, *E* et *F* représentent des nombres qui peuvent aller chacun de 0 à 15, et CC va de 0 à 127. Lorsque le nombre le plus à gauche, représenté ici par *A*, est différent de zéro, on dit que la représentation est normalisée.

Le nombre des positions hexadécimales réservées à la mantisse est soit 6 (forme courte), soit 14 (forme longue). Voici quelques exemples :

+ 65 1 0 0 0 0 0 → $16^{65-64} \times \left(\frac{1}{16} + \frac{0}{16^2} + \cdots\right) = \frac{16}{16} = 1$ = 1.00000E+00

+ 65 2 0 0 0 0 0 → $16^{65-64} \times \left(\frac{2}{16} + \frac{0}{16^2} + \cdots\right) = \frac{32}{16} = 2$ = 2.00000E+00

- 66 1 0 0 0 0 0 → $-16^{66-64} \times \left(\frac{1}{16} + \frac{0}{16^2} + \cdots\right) = -\frac{16^2}{16} = -16$ =-1.60000E+00

+ 66 6 5 0 0 0 0 → $16^{66-64} \times \left(\frac{6}{16} + \frac{5}{16^2} + \frac{0}{16^3} + \cdots\right) = 6 \times 16 + 5 = 101$ = 1.01000E+02

+ 67 7 6 0 0 0 0 → $16^{67-64} \times \left(\frac{7}{16} + \frac{6}{16^2} + \frac{0}{16^3} + \cdots\right) = 7 \times 16^2 + 6 \times 16 = 1888$ = 1.88800E+03

+ 68 1 3 3 0 0 0 → $16^{68-64} \times \left(\frac{1}{16} + \frac{3}{16^2} + \frac{3}{16^3}\right) = 1 \times 16^3 + 3 \times 16^2 + 3 \times 16$ = 4.91200E+03

+ 65 3 2 4 315 3 → $16^{65-64} \times \left(\frac{3}{16} + \frac{2}{16^2} + \frac{4}{16^3} + \frac{3}{16^4} + \frac{15}{16^5} + \frac{3}{16^6}\right)$ = 3.14158E+00

+ 65 211 714 1 3 → $16^{65-64} \times \left(\frac{2}{16} + \frac{11}{16^2} + \frac{7}{16^3} + \frac{14}{16^4} + \frac{1}{16^5} + \frac{3}{16^6}\right)$ = 2.71827E+00

+ 63 2 815 512 2 → $16^{63-64} \times \left(\frac{2}{16} + \frac{8}{16^2} + \frac{15}{16^3} + \frac{5}{16^4} + \frac{12}{16^5} + \frac{2}{16^6}\right)$ = 9.99999E-03

Pour affecter un identificateur doté des attributs DEC FLOAT d'une valeur, on la transforme en forme flottante hexadécimale normalisée. Les diverses conversions nécessaires peuvent provoquer, lorsque le nombre n'est pas entier, une perte de précision.

Lorsque l'attribut de précision est inférieur ou égal à 6 (chiffres décimaux), la mantisse est constituée de 6 positions (hexadécimales). Lorsque cet attribut est supérieur à 6, la mantisse est constituée de 14 positions hexadécimales.

Exemple 1 : `DCL A DEC FLOAT(4);`

Cette instruction définit le filtre

`S CCMMMMMM`

— Lorsqu'on écrit

`A=2.304E+03;`

on obtient :

`+ 67900000`

En effet, la valeur contenue est $16^{67-64} \times \frac{9}{16} = 9 \times 16^2 = 2304$.

A noter que lorsque le filtre contient

`+ 68090000`

il s'agit de la même valeur $\left(16^{68-64} \times \frac{9}{16^2}\right)$ mais dont la représentation n'est pas normalisée.

— Lorsqu'on écrit

```
A=3.14159E+00;
```

on obtient en mémoire

```
+ 65 3, 2, 4, 3, 15, 3
```

Calculons cette valeur

$$16^{65-64} \times \left(\frac{3}{16} + \frac{2}{16^2} + \frac{4}{16^3} + \frac{3}{16^4} + \frac{15}{16^5} + \frac{3}{16^6}\right)$$

soit encore

$$3 + \frac{2}{16} + \frac{4}{16^2} + \frac{3}{16^3} + \frac{15}{16^4} + \frac{3}{16^5}$$

$$\begin{aligned}
&3 && \rightarrow 3\\
+\ &\frac{2}{16} = \frac{1}{8} && \rightarrow 0{,}125\\
+\ &\frac{4}{16^2} = 4 \times 0{,}00390625 && \rightarrow 0{,}015625\\
+\ &\frac{3}{16^3} = 3 \times 0{,}000244140625 && \rightarrow 0{,}000732421875\\
+\ &\frac{15}{16^4} = 15 \times 0{,}0000152587890625 && \rightarrow 0{,}0002288818359375\\
+\ &\frac{3}{16^5} = 3 \times 0{,}00000095367431640625 && \rightarrow 0{,}00000286102294921875\\
&&& = 3{,}14158916473388671875
\end{aligned}$$

La valeur réelle contenue en mémoire est donc 3,141589... mais l'instruction `PUT LIST(A);` entraîne l'impression de `A=3.14158E+00;`

— Lorsqu'on écrit `A=3.14158E+00;` on obtient en mémoire

```
+ 653,2,4,3,1,4,9
```

Calculons cette valeur

$$16^{65-64} \times \left(\frac{3}{16} + \frac{2}{16^2} + \frac{4}{16^3} + \frac{3}{16^4} + \frac{14}{16^5} + \frac{9}{16^6}\right)$$

soit encore

$$3 + \frac{2}{16} + \frac{4}{16^2} + \frac{3}{16^3} + \frac{14}{16^4} + \frac{9}{16^5}$$

$$\begin{aligned}
&3 && \rightarrow 3\\
+\ &\frac{2}{16} = \frac{1}{8} && \rightarrow 0{,}125\\
+\ &\frac{4}{16^2} = 4 \times 0{,}00390625 && \rightarrow 0{,}015625\\
+\ &\frac{3}{16^3} = 3 \times 0{,}000244140625 && \rightarrow 0{,}000732421875\\
+\ &\frac{14}{16^4} = 14 \times 0{,}0000152587890625 && \rightarrow 0{,}000213623046875\\
+\ &\frac{9}{16^5} = 9 \times 0{,}00000095367431640625 && \rightarrow 0{,}00000858306884765625\\
&&& 3{,}14157972799072265625
\end{aligned}$$

La valeur réelle contenue en mémoire est donc 3,141579... mais l'instruction `PUT LIST(A);` entraîne l'impression de `A=3.14157E+00;`

Exemple 2 : `DCL B DEC FLOAT(16);`

Cette instruction définit le filtre

`S CCMMMMMMMMMMMMMM`

Lorsqu'on écrit

`B=3.1415926E+07;`

on obtient

`+ 71 11315 514 7 6 0 0 0 0 0 0 0`

qui représente exactement 3,1415926.

Si on écrit

`B=3.14159E+00;`

on obtient

`+ 65 3 2 4 315 314 0 3 7 0121312`

dont la valeur est `3.141589999999999E+00`

De façon générale, pour affecter une valeur à un identificateur doté des attributs `DECIMAL FLOAT`, on donne à cette valeur une représentation flottante hexadécimale normalisée, courte ou longue selon l'attribut de précision de l'identificateur. Puis on fait entrer dans le filtre le signe, la caractéristique et 6 ou 14 positions pour la mantisse.

A noter que du fait des conversions, 14 positions hexadécimales donnent naissance à 16 positions décimales.

4) Attributs `BINARY FLOAT`

La représentation interne est la même que pour les attributs `DEC FLOAT`. Lorsque l'attribut de précision est égal ou inférieur à 21 (chiffres binaires), la mantisse est constituée de 6 positions (hexadécimales). Lorsque cet attribut est supérieur à 21, la mantisse est constituée de 14 positions (hexadécimales).

Exemple : `DCL X BIN FLOAT(21);`

Cette instruction définit le filtre

`S CCMMMMMM`

— Lorsqu'on écrit

`X=1.E-100B;` (c'est-à-dire $1 \times 2^{-100} = 2^{-100}$)

on obtient

`+ 40100000`

La valeur contenue est $16^{40-64} \times \frac{1}{16} = \frac{16^{-24}}{16} = 16^{-25} = 2^{-100}$.

L'instruction `PUT DATA(X)` entraîne l'impression de

`X=7.888609E-31;`

— Lorsqu'on écrit

`X=1.E+100B;` (c'est-à-dire $1 \times 2^{100} = 2^{100}$)

on obtient

`+ 901,0,0,0,0`

La valeur contenue est $16^{90-64} \times \frac{1}{16} = \frac{16^{26}}{16} = 16^{25} = 2^{100}$.

L'instruction `PUT DATA(X);` entraîne l'impression de

`X=1.267650E+30`

II. Rappels sur les conversions décimal binaire

1) Un nombre écrit dans le système binaire ne contient que les chiffres 0 ou 1.

Exemple 1 :

$1\ 0\ 1\ 1\ 0\ 1_2$

$$\left.\begin{array}{l} 1 \times 2^0 \\ + \ 0 \times 2^1 \\ + \ 1 \times 2^2 \\ + \ 1 \times 2^3 \\ + \ 0 \times 2^4 \\ + \ 1 \times 2^5 \end{array}\right\} \rightarrow \left\{\begin{array}{r} 1 \\ + \ 0 \\ + \ 4 \\ + \ 8 \\ + \ 0 \\ + \ 32 \\ \hline = 45 \end{array}\right.$$

Exemple 2 :

$0\ ,\ 1\ 0\ 1\ 1_2$

$$\left.\begin{array}{l} 1 \times 2^{-4} \\ + \ 1 \times 2^{-3} \\ + \ 0 \times 2^{-2} \\ + \ 1 \times 2^{-1} \end{array}\right\} \rightarrow \left\{\begin{array}{r} 0{,}0625 \\ + \ 0{,}125 \\ + \ 0{,}00 \\ + \ 0{,}5 \\ \hline = 0{,}6875 \end{array}\right.$$

Exemple 3 :

$1\ 1\ 1\ ,\ 1\ 1_2$

$$\left.\begin{array}{l} 1 \times 2^{-3} \\ + \ 1 \times 2^{-2} \\ + \ 1 \times 2^{-1} \\ + \ 1 \times 2^0 \\ + \ 1 \times 2^1 \\ + \ 1 \times 2^2 \end{array}\right\} \rightarrow \left\{\begin{array}{r} 0{,}125 \\ + \ 0{,}25 \\ + \ 0{,}5 \\ + \ 1 \\ + \ 2 \\ + \ 4 \\ \hline = 7{,}875 \end{array}\right.$$

n	2^n	représentation binaire de 2^n
1	2	00000000000000000000000000000010B
2	4	00000000000000000000000000000100B
3	8	00000000000000000000000000001000B
4	16	00000000000000000000000000010000B
5	32	00000000000000000000000000100000B
6	64	00000000000000000000000001000000B
7	128	00000000000000000000000010000000B
8	256	00000000000000000000000100000000B
9	512	00000000000000000000001000000000B
10	1024	00000000000000000000010000000000B
11	2048	00000000000000000000100000000000B
12	4096	00000000000000000001000000000000B
13	8192	00000000000000000010000000000000B
14	16384	00000000000000000100000000000000B
15	32768	00000000000000001000000000000000B
16	65536	00000000000000010000000000000000B
17	131072	00000000000000100000000000000000B
18	262144	00000000000001000000000000000000B
19	524288	00000000000010000000000000000000B
20	1048576	00000000000100000000000000000000B
21	2097152	00000000001000000000000000000000B
22	4194304	00000000010000000000000000000000B
23	8388608	00000000100000000000000000000000B
24	16777216	00000001000000000000000000000000B
25	33554432	00000010000000000000000000000000B
26	67108864	00000100000000000000000000000000B
27	134217728	00001000000000000000000000000000B
28	268435456	00010000000000000000000000000000B
29	536870912	00100000000000000000000000000000B
30	1073741824	01000000000000000000000000000000B

Tableau des valeurs décimales et binaires de 2^n lorsque n va de 1 à 30

n	2^n	représentation binaire de 2^n
-1	0.500000000000000	0.1000000000000000000000000000000B
-2	0.250000000000000	0.0100000000000000000000000000000B
-3	0.125000000000000	0.0010000000000000000000000000000B
-4	0.062500000000000	0.0001000000000000000000000000000B
-5	0.031250000000000	0.0000100000000000000000000000000B
-6	0.015625000000000	0.0000010000000000000000000000000B
-7	0.007812500000000	0.0000001000000000000000000000000B
-8	0.003906250000000	0.0000000100000000000000000000000B
-9	0.001953125000000	0.0000000010000000000000000000000B
-10	0.000976562500000	0.0000000001000000000000000000000B
-11	0.000488281250000	0.0000000000100000000000000000000B
-12	0.000244140625000	0.0000000000010000000000000000000B
-13	0.000122070312500	0.0000000000001000000000000000000B
-14	0.000061035156250	0.0000000000000100000000000000000B
-15	0.000030517578125	0.0000000000000010000000000000000B
-16	0.000015258789062	0.0000000000000001000000000000000B
-17	0.000007629394531	0.0000000000000000100000000000000B
-18	0.000003814697265	0.0000000000000000010000000000000B
-19	0.000001907348632	0.0000000000000000001000000000000B
-20	0.000000953674316	0.0000000000000000000100000000000B
-21	0.000000476837158	0.0000000000000000000010000000000B
-22	0.000000238418579	0.0000000000000000000001000000000B
-23	0.000000119209289	0.0000000000000000000000100000000B
-24	0.000000059604644	0.0000000000000000000000010000000B
-25	0.000000029802322	0.0000000000000000000000001000000B
-26	0.000000014901161	0.0000000000000000000000000100000B
-27	0.000000007450580	0.0000000000000000000000000010000B
-28	0.000000003725290	0.0000000000000000000000000001000B
-29	0.000000001862645	0.0000000000000000000000000000100B
-30	0.000000000931322	0.0000000000000000000000000000010B
-31	0.000000000465661	0.0000000000000000000000000000001B

Tableau des valeurs décimales (avec 14 chiffres exacts) et binaires de 2^n lorsque n va de -1 à -31

2) Pour convertir un nombre décimal en un nombre binaire, on sépare la partie entière de la partie fractionnaire.

Exemple 1

Soit 23,6875 à convertir en base 2.

— La partie entière 23 est *divisée* successivement par 2, et on garde les restes successifs et le dernier quotient, qui lus dans l'ordre inverse où on les a trouvés, donnent le résultat

```
23 | 2
03 | 11 | 2
 1 |  1 | 5 | 2
       1 | 2 | 2
           0 | 1
```

$23 = 10111_2$

— La partie fractionnaire est *multipliée* successivement par 2, et on garde les chiffres 0 et 1 qui « sortent » du cadre

```
0 ,6875
    × 2
1 | 3750
    × 2
0 | 7500
    × 2
1 | 5000
    × 2
1 | 0000
```

$0{,}6875 = 0{,}1011_2$.

Finalement,

$23{,}6875 = 10111{,}1011_2$.

Exemple 2

Soit à convertir 1024,625 en base 2.

— Partie entière *divisée* successivement par 2

```
1024 | 2
  02 | 512 | 2
  04 | 11  | 256 | 2
   0 | 12  | 05  | 128 | 2
       0 | 16  | 08  | 64 | 2
           0 | 0 | 04  | 32 | 2
               0 | 12  | 16 | 2
                   0 | 0 | 8 | 2
                       0 | 4 | 2
                           0 | 2 | 2
                               0 | 1
```

$1024 = 10000000000_2$ $\quad (2^{10} = 1024)$.

— Partie fractionnaire multipliée successivement par 2

$\boxed{0}$,625
× 2
$\boxed{1}$ 250
× 2
$\boxed{0}$ 500
× 2
$\boxed{1}$ 000

$$0{,}625 = 0{,}101_2$$

Finalement

$$1024{,}625 = 10000000000{,}101_2.$$

La conversion de la partie entière se fait toujours exactement. Lorsque celle de la partie fractionnaire ne tombe pas juste, on doit décider où arrêter le développement.

Exemple. Soit à convertir 0,3 en base 2

$\boxed{0}$, 3
× 2
$\boxed{0}$ 6
× 2
$\boxed{1}$ 2
× 2
$\boxed{0}$ 4
× 2
$\boxed{0}$ 8
× 2
$\boxed{1}$ 6
× 2
$\boxed{1}$ 2
× 2
$\boxed{0}$ 4
× 2
$\boxed{0}$ 8
× 2
$\boxed{1}$ 6

on obtient $0{,}3 = 0{,}\underbrace{0100110011001}\ldots_2$

Si on décide de ne garder que dix chiffres binaires, on écrira

$$0{,}3 = 0{,}0100110011_2$$

commettant ainsi une légère erreur puisque le nombre de droite vaut

$$\begin{array}{lcl} & 2^{-2} & \rightarrow & 0{,}25 \\ + & 2^{-5} & \rightarrow & + \ 0{,}03125 \\ + & 2^{-6} & \rightarrow & + \ 0{,}015625 \\ + & 2^{-9} & \rightarrow & + \ 0{,}001953125 \\ + & 2^{-10} & \rightarrow & + \ 0{,}0009765625 \\ & & & \overline{0{,}2998046875} \end{array}$$

3) Pour multiplier un nombre binaire par 2,

— on décale la virgule d'une position sur la droite
— s'il est entier, on ajoute un 0 à droite

$101{,}11_2 \times 2 = 1011{,}1_2$

$10111_2 \times 2 = 101110_2$.

De façon générale, pour multiplier un nombre entier binaire par 2^n, on ajoute à sa droite n zéros

$1011_2 \times 2^3 = 1011000_2$.

Imprimé en France
Imprimerie JOUVE, 12, Rue de Tournon, PARIS (6e)
Dépôt légal : N° 7138. — 3e trimestre 1971

Le langage PL/I a-t-il été utile ?

Pour répondre à cette question il faut d'abord s'interroger si son utilité vise d'abord à répondre à un besoin des utilisateurs, ou bien à soutenir la démarche commerciale d'un constructeur[1].

La réponse est mitigée.

Selon Dornbusch (1971, page vi), Il y a eu une « traversée du désert … Le premier compilateur PL/I est mis en service par IBM en 1966. L'enfant est né avant terme, ses premiers pas sont hésitants et maladroits**, mais il ne déçoit pas ses premiers adeptes**. Une fois passées les maladies de jeunesse, ses qualités évidentes apparaissent à la plus grande satisfaction de ses utilisateurs. Il est ensuite l'objet d'**une intense campagne de dénigrement**, orchestrée surtout par ceux-là même qui n'avaient pas pris la peine de l'examiner de près. Toutes ses qualités lui sont reprochées en bloc. On le traite de 'ramassis de FORTRAN, COBOL, et ALGOL', alors qu'il contient effectivement le meilleur de ces langages. On l'accuse d'être géré par un compilateur encombrant, alors que sur la série 360/0S, le compilateur PL/I est le moins encombrant de tous les compilateurs. On le suspecte de produire un code résultant vaste et peu performant, alors qu'il offre la possibilité de choisir librement pour un encombrement mémoire **réduit aux dépens de la vitesse d'exécution** ou réciproquement. Enfin et surtout, on en fait le cheval de bataille d'IBM (« passer à PL/I, c'est se lier à IBM ») alors que, d'une part ce sont essentiellement les utilisateurs qui ont conçu PL/I, IBM n'ayant fait que suivre leurs recommandations, et que d'autre part FORTRAN était effectivement un produit IBM, ce qui n'empêcha personne de le copier. Pendant les années 1968-1969, les conférences où l'on traite de langages sont curieusement animées par des groupes qui n'utilisent pas PL/I et en disent le plus grand mal, et par des groupes d'utilisateurs satisfaits de PL/I où l'on s'étonne de l'animosité des premiers. PL/I acquiert ses lettres de noblesse dans l'université où, à l'abri des passions et des querelles, on commence à l'enseigner et à s'en servir (il est tellement plus simple d'enseigner un seul langage là où on devait en enseigner deux). En 1970, toutes les enquêtes faites sur l'utilisation de PL/I convergent : ses utilisateurs sont satisfaits (le coût global d'une application, depuis la formation du programmeur jusqu'à la maintenance du programme est singulièrement réduit avec PL/I) et leur nombre va sans cesse croissant. La plupart des constructeurs annoncent alors (ou laissent entendre) qu'ils offriront un compilateur PL/I ».

En positif, PL/I a donc eu un grand nombre d'utilisateurs, et des utilisateurs qui l'ont bien aimé, cela en dépit des faiblesses de sa structuration comme langage manquant de bases théoriques, une critique lancée par les universitaires qui lui préféraient ALGOL[2]. Algol un langage qui a été définit **avant** PL/I comme en témoigne les appellations bien connues, telles l'ALGOL 58, l'Algol-60 où le 60 signifie l'année 1960, donc bien avant l'arrivée de PL/I.

[1] Voir Lecht (1968, page xii)

[2] Voir Lecht (1968, page xiii)