

projets PARI - 68

document de travail n° 1

★ UTILISATEURS POTENTIELS

utilisateurs primaires : arithméticiens
utilisateurs secondaires : algébristes et physiciens
théoriciens (calcul formel)

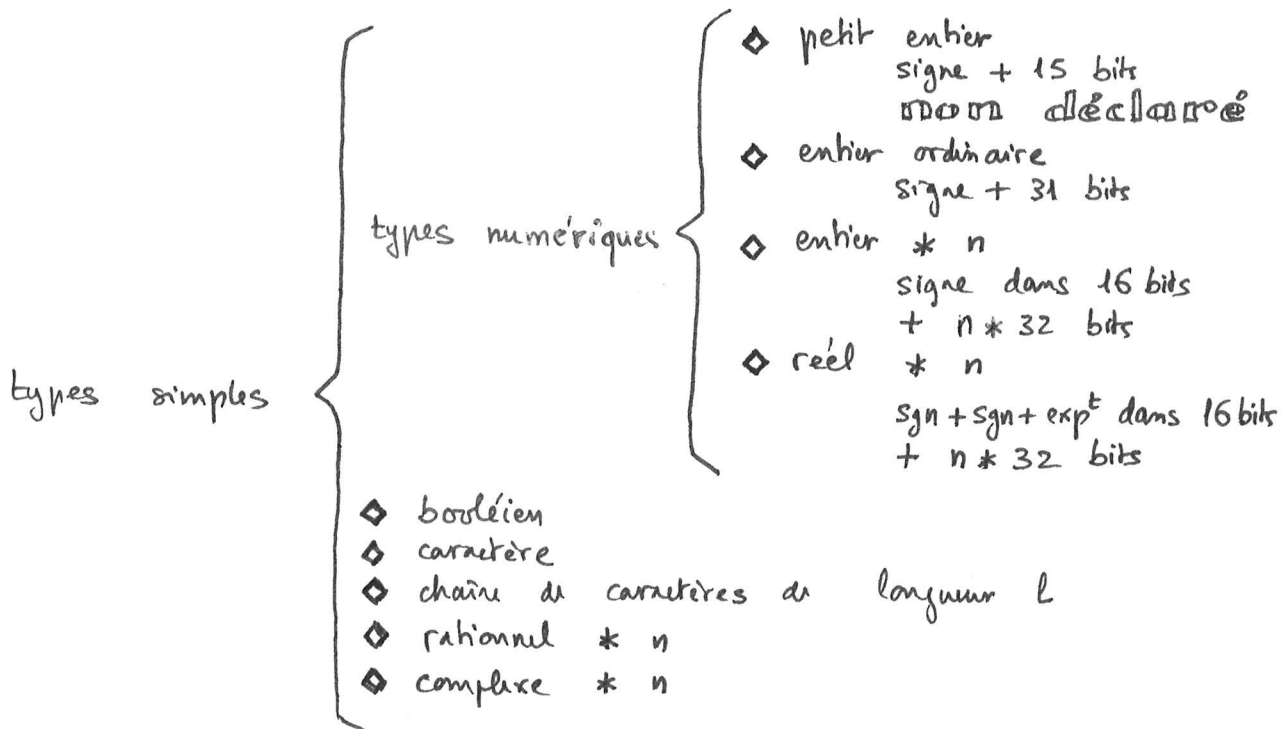
prévoir une présentation sommaire du langage qui sera
envoyée aux utilisateurs potentiels \Rightarrow réactions puis
confection d'un cahier des charges

★ a priori deux projets sont envisageables :

- le pari de Pascal
élaboration d'un langage quasi-Pascal augmenté
de très nombreux types / écriture en assembleur
68000 d'un compilateur

- le pari basique
élaboration d'un langage quasi-Basic augmenté
de très nombreux types / écriture en assembleur
68000 d'un interpréteur

★ LISTE PROVISOIRE DES TYPES ENVISAGÉS (pari de Pascal)



- ◇ tableau (non structuré) de type simple
- ◇ vecteur, matrice de type numérique
- ◇ polynôme à 1 variable, développement limité de type numérique
- ◇ fichier
- ◇ record ??

★ idées diverses

- fonctions à réaliser :

+ - * / ^ sur tous les types additionnelles
multiplicables, etc.... (et ce globalement pour
les types composés structurés : vecteur, matrice, etc...)

opérations modulo

(prendre des variables et des drapeaux intmod polynmod intmodctrl polynmodctrl)

dérivation, intégration, composition (polynômes et dev. lin) pour les rationnels

fonctions arithmétiques : d, φ, σ, μ, etc....
a priori pour les petits entiers seulement

plus petit facteur premier ≥ ... , indicateur de primalité
seulement pour les entiers ordinaires

- constantes :

π
 γ (sur demande)
 $+\infty, -\infty, \infty$ projectif

projets PARI - C68

document de travail n° 2

* COMPLÉMENT DU PRÉCÉDENT

- fonctions à réaliser

sur matrices et vecteurs : M^{-1} EM MV M^n

sur polynômes : $P \text{ div } P$ $P \text{ mod } P$ $P \wedge n$ éval P

sur développés limités : $F_0 G$ F_{0-1} F/F

- optimisation des algorithmes de multiplication pour * 4
(\Rightarrow pénalisation divisaire en * 2)

- fractions arithmétiques : On peut envisager une factorisation automatique jusqu'à 10^{18} (Shanks)

* DÉCISIONS (provisaires ?)

- écrire le compilateur en C
travailler sous UNIX (même si absent sur le SORD)

- le langage doit être un sur-ensemble d'un sous-ensemble de PASCAL

à supprimer :
type ensemble
type predef
gets et étiquettes (???)

à rajouter :
EXIT, EXITALL
CASE
:= (échange)

et bien sûr tous les types usages précédemment avec leurs fonctions associées

1 messages d'erreurs clairs
1 non arrêt de la compilation si erreur (saut au prochain délimiteur)

* en discussion ...

possibilité pour le programmeur de définir de nouvelles fonctions :

{ syntactiquement :
sémantiquement :

de définir de nouvelles

nouveaux opérateurs binaires (faut-il réserver pour cela quelques caractères comme #, ... ?)
fonctionnement défini comme une FUNCTION ou même une PROCEDURE

projet PARI - C 68

document de travail n°3

★ ET UN PRÉ-PROCESSEUR ?

```
PROGRAM ESSAI ;
CONST UD = 1#2 ;
      UT = 1#3 ;
      DT = 2#3 ;
VAR P2, Q2 : POLY(9) OF RAT ;
BEGIN
  P2[0]:=UD; P2[1]:=UT; P2[2]:=DT;
  Q2 = P2 * P2 ;
  WRITELN (Q2);
END.
```

```
PROGRAM ESSAI (INPUT, OUTPUT) ;
CONST UD0010 = 1; UD0020 = 2 ;
      " " " " ; DT0020 = 3 ;
VAR P2757, Q2757, R2757 : ARRAY [0..9, 0..1]
  OF INTEGER ;
PROCEDURE MUL (A, B, C : INTEGER) : EXTERN ;
PROCEDURE AFFET (A, B : INTEGER) : EXTERN ;
PROCEDURE WRITZ (A : INTEGER) : EXTERN ;
BEGIN
  P2757 [0,0] := UD0010 ; " " " " ; P2757 [2,1] := DT0020 ;
  MUL (↑P2757, ↑P2757, ↑R2757) ;
  AFFET (↑R2757, ↑Q2757) ;
  WRITZ (↑Q2757) ;
END.
```

remarque : 0010 code du numérateur entier simple d'un rationnel
 0020 " " " " dénominateur entier simple d'un rationnel
 757 " " d'un polynôme de rationnels d'entiers

★ MAIS IL Y A UN CONCURRENT :

un PACKAGE d'ADA qui apparemment peut tout faire " en particulier supporter des procédures avec des types " mutants "

pour PASCAL et préprocesseur	pour ADA
① La vitesse d'exécution	④ le principe
② la grande diffusion des compilateurs PASCAL / la rareté (et la mix !!!) des compilateurs ADA	⑤ la rapidité d'écriture du package et la simplicité ultérieure
③ l' "énormité" d'ADA	

un essai sera probablement fait avec ADA (si compilateur disponible). Il est vraisemblable que ①+② ⇒ écriture d'un pré-processeur dans les 2 cas de figure

Rappel : 1 octet = 8 bits ; 1 mot = 16 bits ; 1 long mot = 32 bits.

5) Types scalaires

1.1 Type S : entier court ou non déclaré
1 octet + 1 longmot ou 1 octet + 1 mot.



bit 3 = 0 : mot, bit 3 = 1 : long mot.

bits 2 1 0 :

000	S < 0
001	S = 0
010	S > 0
011	S modulo <u>SMOD</u>
100	S = +∞
101	S = -∞
110	S = ∞ ou ∞ modulo <u>SMOD</u>
111	∞ modulo <u>SMOD</u> est utilisée

Ici SMOD est une variable prédéfinie de type S. Elle doit être déclarée et initialisée (si le type modulo SMOD est utilisé)

1.2 Type I : entier long ou multiprécision
1 mot + n longmots

mot: 1000 + 12 bits	I < 0
1001 + 12 bits	I = 0
1010 + 12 bits	I > 0
1011 + 12 bits	I modulo <u>IMOD</u>

Dans ces quatre cas les 12 bits représentent un de moins que le nombre n de long m.

Entier maximal représenté : $2^{131072} - 1 > 10^{39456}$

Ici IMOD est une variable prédéfinie de type I. Elle doit être déclarée et initialisée (si le type modulo IMOD est utilisé)

1.3 Type R : réel long ou multiprécision

2 mots + n longmots (sauf type spécial 2 mots + 1 longmot)

1 ^{er} mot: 1100 + 12 bits	R < 0
1101 + 12 bits	R de type spécial (voir plus bas)
1110 + 12 bits	R > 0

Dans le 1^{er} et le 3^e cas les 12 bits représentent un de moins que le nombre n de long mots

type spéciale



* = n'importe quoi

bits 210

- 100 R = +∞
- 101 R = -∞
- 110 } R = ∞
- 111 } R = 0
- 001 R = 0

Dans le cas des types spéciales, le 2^e mot et le long mot ne servent à rien mais permettent de conserver une régularité de structure pour le préprocesseur =

On suppose désormais R non spécial

2^e mot: exposant, ~~le 2^e mot est considéré comme~~

n long mots: mantisse de R, considérée comme comprise entre 1 (au sens large) et 2 (strictement).

Réels représentés:

$$2^{-32768} \leq |R| \leq 2^{32768} (1 - 2^{-32n})$$

§2) Types ~~non~~ linéaires courts

2.1. Type C (typiquement complexes ou entiers de Gauss)

1 octet + 2 types ~~de~~ identiques

octet: 0000****

2.2. Type F (typiquement fractions ou fractions rationnelles)

1 octet + 2 types identiques

octet 0000**** sans PGCD automatique

octet 0011**** avec PGCD automatique

§3) Types linéaires

3.1. Type V (typiquement vecteur)

1 long mot + m types identiques où

long mot = 0000 + 28 bits (contenant m)

3.2. Type P (typiquement polynôme)

idem avec 0000 remplacé par 0001 (m est le degré + 1)

3.3 Type PM (typiquement polynôme modulo un polynôme prédéfini POLYMOD)
idem avec 00110

3.4 Type DL 1 long mot + 1 mot + m type identiques. où
long mot = 0011 + 28 bits (contenant m)
mot : contient l'exposant (i.e. si exposant ≥ 0 , on a un polynôme)

54) Type multilinéaire

4.1 Type M (typiquement matrice)
1 long mot + m' types vecteurs identiques.

long mot = 0100 + 28 bits (contenant m')
Vecteurs identiques : représentent les lignes de la matrice M. (Donc m' lignes x m colonnes si vecteur à m composantes)
etc...!

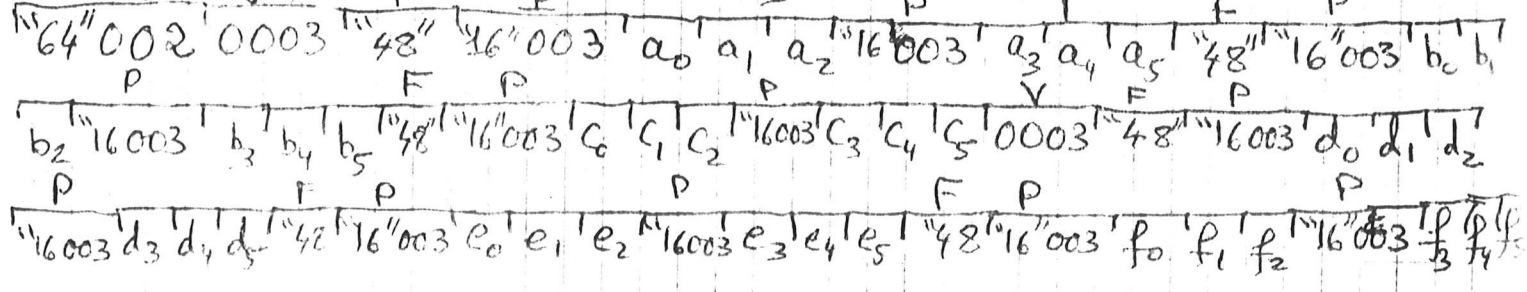


Attention : dans tous les types linéaires ou multilinéaires, les composantes doivent être du même type.

Exemple : représentation interne de la matrice

$$\begin{pmatrix} a_0 + a_1 x + a_2 x^2 & b_0 + b_1 x + b_2 x^2 & c_0 + c_1 x + c_2 x^2 \\ a_3 + a_4 x + a_5 x^2 & b_3 + b_4 x + b_5 x^2 & c_3 + c_4 x + c_5 x^2 \\ d_0 + d_1 x + d_2 x^2 & e_0 + e_1 x + e_2 x^2 & f_0 + f_1 x + f_2 x^2 \\ d_3 + d_4 x + d_5 x^2 & e_3 + e_4 x + e_5 x^2 & f_3 + f_4 x + f_5 x^2 \end{pmatrix}$$

où les a_i, b_i, \dots sont des réels γ -précision.



où chaque a_i, b_i, \dots occupe 2 mot + 7 longmots
= 32 octets comme expliqué en 1.3 -

Occupation mémoire totale: 1218 octets dont
138 (un peu plus de 10%) en codage - La proportion
codage / occupation totale est de

$$138 / (210 + 144n)$$

où n est la précision des a_i, \dots

Si $n = 2$ cela représente 28% ce qui n'est pas
négligeable -

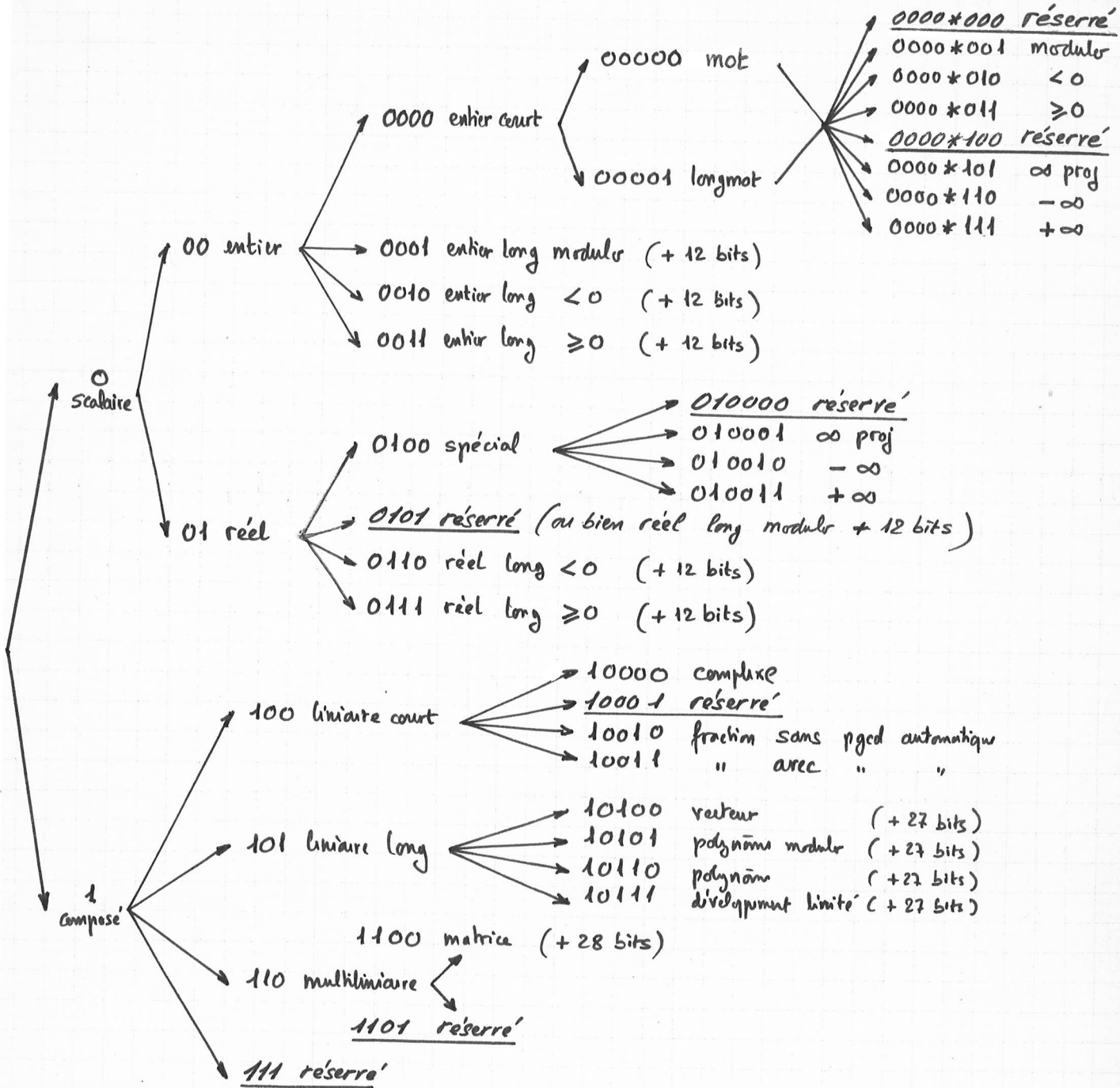
projet PARI - C68

document de travail n° 5

- ★ discussion sur la portabilité
accord sur
 - mise en attente de la réalisation d'un package ADA
 - pari sur C et Unix
- ★ discussion sur le codage des types, la "régularité" des représentations mémoire
- ★ discussion embrouillée autour des types génériques et de la définition des opérateurs sur les nouveaux types, resp. la définition de nouveaux opérateurs
discussion tout aussi embrouillée sur le rôle exact du préprocesseur, le problème de l'écrire en C pour générer du Pascal, la possibilité de lui demander d'étendre la bibliothèque, etc ...
- ★ décision de définir d'ici 8 à 15 jours les étapes du fonctionnement du préprocesseur

15 nov 1983

DRESS → COHEN : nouvelle proposition de codage des types numériques sur 1 octet



17 (12/11/83)

Représentation interne des différents types.

Rappel: 1 octet = 8 bits; 1 mot = 16 bits; 1 longmot = 32 bits.

B3 = bloc de 3 bits ayant la signification suivante

000	∞ projectif
001	∞ projectif
010	modulo
011	< 0
100	$= 0$
101	> 0
110	$+\infty$
111	$-\infty$

3.1) Types scalaires

1.1. Type S (entier court)

1 octet + 1 mot ou 1 octet + 1 longmot

octet : 00 B3 010 mot
 00 B3 100 longmot

Ici modulo signifie modulo SMOD où SMOD est une variable prédefinie de même type, qui doit être initialisée si le type modulo est utilisé.

1.2. Type L (entier long)

1 mot + n longmots

mot : 01 B3 + 11 bits où les 11 bits représentent un de moins que le nombre n de longmots.

Entier maximal représenté $2^{65536} - 1 > 10^{19728}$

Ici modulo signifie modulo IMOD où IMOD est une variable prédefinie de type I.

1.3. Type R (réel long)

2 mots + n longmots.

1^{er} mot : 10 [B3] + 11 bits où les 11 bits représentent un de moins que le nombre n de long mots

2^e mot (significatif seulement si B3 = 011 (<0) ou 101 (>0))
exposant

n long mots (significatif comme ci-dessus) : mantisse de R , considérée comme comprise entre 1 (au sens large) et 2 (strictement)

Reels représentés : 0 et

$$2^{-32768} \leq |R| \leq 2^{32768} (1 - 2^{-32n})$$

§2) Types linéaires courts

2.1 Type C (typiquement complexes ou entiers de Gauss)

1 octet + 2 types identiques

$$\text{octet} = 11000000$$

2.2 Type F (typiquement nombres ou fractions rationnelles)

1 octet + 2 types identiques

$$\text{octet} = 11001000 \text{ sans PGCD automatique}$$

$$= 11001100 \text{ avec PGCD automatique}$$

§3) Types linéaires

3.1 Type V (typiquement vecteur)

1 long mot + m types identiques, où

$$\text{long mot} = 11010000 + 24 \text{ bits (contenant } m)$$

1 pour vecteurs horizontaux
0 pour vecteurs verticaux

3.2 Type P (typiquement polynôme)

idem avec 11010000 remplacé par 11011000

3.3 Type PM (typiquement polynôme modulo un polynôme prédéfini POLYMOD)

idem avec 11010000 remplacé par 11011000

3.4 Type DL (Développement limité généralisé)

1 long mot + 1 mot + m types identiques

$$\text{long mot} = 11011100 + 24 \text{ bits (contenant } m)$$

mot : contient l'exposant de X (i.e. si exposant ≥ 0
on a un polynôme.

§4) Type multilinéaire

4.1 Type M (matrice)

1 long mot + m' types vecteurs identiques

long mot = 11100000 + 24 bits (contenant m')

vecteurs identiques : représentent les lignes de la
matrice M .