Computer Aided Number Theory
—Introduction to Pari—

木村 巌*
富山大学理学部数学科

目 次

1 序論 **2**

2 Pari の型システム **3**

2.1 Pari で用意されている型 3

2.2 GEN オブジェクトの生成 4

2.2.1 整数をつくる 4

2.2.2 ベクトルや行列をつくる 6

2.2.3 多項式をつくる 7

2.3 GEN オブジェクトの代入 8

3 ライブラリ関数の呼び出し方 **9**

4 入出力 **11**

4.1 出力 11

4.2 入力 13

4.3 エラー出力 13

5 メモリ管理 **13**

5.1 Pari スタックと avma 13

5.2 avma の書き換えによるゴミ集め 16

5.3 cgiv() によるゴミ集め 17

*iwao@sci.toyama-u.ac.jp

5.4 gerepile() によるゴミ集め 17

6 例 20

6.1 2 次体の場合 20

6.2 一般の代数体の計算 21

6.3 コツ 21

7 その他 24

7.1 サンプルのコンパイル法 24

7.2 GP2C 25

7.2.1 gp2c の入手とコンパイル 25

7.2.2 gp2c の使い方 25

7.2.3 注意 26

7.3 Pari-gp の入手先・情報源 27

概 要

本稿では、H. Cohen, K. Belabas らによって開発・保守が行われている計算機数論用のライブラリ Pari[1]を用いたプログラミングを解説する．用いる言語は C 言語である．

1 序論

Pari ライブラリを用いて C 言語でプログラムを作成するには、まずプログラムの先頭で、`void pari_init(long, long)`を呼ばなければならない（5 節（13 頁）を参照）．この関数は、Pari スタックと呼ばれる領域を確保して初期化する他、1,2, 1/2, 円周率、自然対数の底などの定数を用意し、初めの幾つかの素数の計算などを行う．1 は`gun`, 2 は`gdeux`, 1/2 は`ghalf` と定義されている．

ついで、以下の計算で必要になる初期データを、Pari スタック上に置く．これらを引数として、Pari ライブラリで用意している関数を呼び出すと、結果が Pari スタックに積まれ、そのアドレスが返される．

Pari スタックは、初期データの保持の他、Pari のライブラリ関数が計算の途中に一時的なデータの保存場所に使ったり（勿論そのライブラリ関数から戻る際に、一時的なデータは消去される)、プログラマ自身が、データの一時的な保存場所として使ったりする．従って、計算の規模がある程度以上大きくなると、最初の`pari_init()`の呼び出しで確保したサイズでは足りなくなることがある．それを防ぐために、必要なくなった一時データの消去など、メモリ管理を行わねばならない．Pari ライブラリを用いた C 言語でのプログラミングでは、このメモリ管理がもっとも面倒な部分である．

---

[1]用いるバージョンは 2.1.0.

本稿では、C 言語に関する基礎的な知識を仮定する（例えばカーニハン・リッチー［1］やハービンソン・スティール［4］参照）．また、サンプルプログラムなどはいずれも Pari-gp 2.1.0 がインストールされた FreeBSD 3.4-stable の上で動作確認などを行った．

2 Pari の型システム

Pari ライブラリを用いて計算を行うとき、例えば（多倍長の）整数、有理数、p 進数といった対象を扱う．これらは C 言語には用意されていないので、Pari ライブラリの内部でしかるべく定義されているものを使う．

Pari ライブラリで定義されたこれらの対象は、C 言語の立場からは単一の GEN という型に見える．Pari で用意されたそれぞれの型は、単なるバイト列として long の配列に格納されている．GEN は `long *` の typedef に他ならない．Pari ライブラリで定義された様々な型を持つ対象を、GEN オブジェクトと呼ぶことにする．以下で、Pari が用意している型にはどんなものがあるか、GEN オブジェクトを生成したり、それらを変数に代入したりするにはどうすればよいのかを解説する．

2.1 Pari で用意されている型

Pari ライブラリで用意されている型は、以下の通りである．

- 整数（`t_INT`）、実数（`t_REAL`）、整数を法とする整数（`t_INTMOD`）、有理数（`t_FRAC`，`t_FRACN`）、複素数（`t_COMPLEX`）、p 進数（`t_PADIC`）、2 次体の元（`t_QUAD`）
- 多項式（`t_POL`）、多項式を法とした多項式（`t_POLMOD`）、冪級数（`t_SER`）、有理関数（`t_RFRAC`，`t_RFRACN`）、2 次形式（`t_QFR`，`t_QFI`）
- 行・列ベクトル（`t_VEC`，`t_COL`）、行列（`t_MAT`）、リスト（`t_LIST`）、文字列（`t_STR`）

Pari ライブラリ内では、これらの型に小さい自然数を割り振ることで区別している．それらの整数を覚えやすくするために、マクロが定義されている．括弧内に書いたものがそれである．

これらの型は、次のような意味で recursive である．例えば、多項式（`t_POL`）の係数は、整数、実数、複素数、p 進数のいずれでも良いばかりか、上記の型から作ることができる任意の型で良い．代数体 k 上の多項式を表す場合、k を有理数体上の多項式環 $\mathbf{Q}[x]$ の剰余（`t_POLMOD`）として表し、それを係数とする多項式（`t_POL`）とすることが出来る．

多項式に限らず、数学的に意味があれば、ほぼ全ての組み合わせが可能である[2]．

[2]数学的に意味があって、Pari ライブラリで対応していない組合わせがあれば、開発陣に報告してください．

個々の型は、それぞれの型に固有の値を持つことがある．例えば p 進数ならば p 進的な精度、p に関する付値、また多項式ならば変数や次数、などである．これらの値を調べる・値を定める関数が型に応じて定義されている．詳細は Users guide［2］の 4.5 節を参照して欲しい．

2.2　GEN オブジェクト の生成

GEN オブジェクトを Pari スタック上に生成するための関数は、`GEN cgetg（long length, int Pari_type）`である．また、整数（`t_INT`）と実数（`t_REAL`）を生成するために、それぞれ`GEN cgeti（long length）= cgetg（long length, t_INT）`,`GEN cgetr（long length, t_REAL）`が用意されている．

`cgetg()` の第 2 引数の Pari_type は、上述した各型に割り当てられた自然数である．length は、その GEN オブジェクトを保持するのに、どれだけの Pari スタックを割り付けるかを指定する．この値は、`DEFAULTPREC`, `MEDDEFAULTPREC`, `BIGDEFAULTPREC`, のいずれかのマクロで指定する[3].

GEN オブジェクトは、実際には long の配列として実現されている．その配列の 0 番目（以下これを 0 番目のコード ワード（code word）と言う）には、GEN オブジェクトとしての型、今の場合なら`t_INT` などの管理情報が含まれている．幾つかの GEN オブジェクトでは、もうひとつ管理用にコード ワードを消費する．例えば`t_INT` はそうであり、そこには、符号、実際に意味のある mantissa が格納されている最大のコード ワードが何処までか、などの情報が格納されている．

次の節で、Pari スタック上に整数をつくる例を見てみよう．

2.2.1　整数をつくる

例えば、`cgeti(DEFAULTPREC)`は、絶対値が 2^{64} 以下の整数を保持するための領域を Pari スタックから割り付け、管理情報を格納するためのコード ワードをしかるべく初期化して、そのアドレスを返す．`t_INT` の各 mantissa は、大きい（significant な）方から順に並んでいるので、それを埋めていくことで必要な整数を得る（リスト 1）．

ただし、この方法で整数や実数を Pari スタック上に生成することは殆んどない．必要な整数が`long` の範囲であれば、`GEN stoi（long）`によって、C の long を`t_INT` 型の GEN オブジェクトに変換できる．

`long` の範囲に収まらない場合は、文字列を GEN オブジェクトに変換する`GEN lisseq（char *）`を用いる．この関数は、与えられた文字列に応じて、適切な GEN オブジェクトを Pari スタックに生成する．

[3]そうでないと、プログラムを実行するマシンが 32bit マシンか 64bit マシンかで計算の精度が異なる、移植性の低いものになってしまう．

リスト 1: 整数を作る例

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN i;
8:   pari_init (200000, 2000);
9:   i = cgeti (DEFAULTPREC); /* allocate space for integer */
10:   setlgefint (i, 4); /* effective length = 4 */
11:   setsigne (i, 1);   /* signature > 0 */
12:   i[2] = 1L;         /* 1 * 2**32 + 1 */
13:   i[3] = 1L;
14:
15:   printf ("i = %s\n", GENtostr (i));
16:   return 0;
17: }
```

これらの関数は、Pari スタックの領域確保を自動的に行うので、予め`cgetg()`などで領域を確保しておく必要はない.

リスト 2: `lisseq()`を使って整数をつくる例

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN i;
8:   pari_init (200000, 2);
9:   i = lisseq ("18446744073709551616"); /* == 2^64 */
10:   outbeaut (i);
11:   return 0;
12: }
```

実数や複素数なども同様である.

2.2.2 ベクトルや行列をつくる

整数や有理数、実数など、それ自身が値であるようなものとは別に、ベクトルや行列と言った、値を入れる器に相当するものがある．これらはしばしばコンテナ（container）とも呼ばれる．Pari スタック上にコンテナを作るにはどうするかを説明する．

リスト 3: ベクトルと行列をつくる

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN a, b, c, d, v, w, m;
8:   pari_init (200000, 2);
9:   a = gzero;       /* gzero = 0 */
10:   b = gun;          /* gun = 1 */
11:   c = gneg (gun);  /* gneg(x) = -x */
12:   d = gzero;
13:   v = cgetg (3, t_COL);
14:   v[1] = (long) a;
15:   v[2] = (long) b;
16:   w = cgetg (3, t_COL);
17:   w[1] = (long) c;
18:   w[2] = (long) d;
19:   m = cgetg (3, t_MAT);
20:   m[1] = (long) v;
21:   m[2] = (long) w;
22:   printf ("m = %s\n", GENtostr (m));
23:   return 0;
24:   /* output is [0, −1; 1, 0] */
25: }
26:
```

リスト 3 で、v, w がそれぞれ 1 列目、2 列目である（Pari の行列は、列ベクトルの行ベクトルとして表される）．`cgetg()` でそれらを確保してから、各成分を埋めている（`GEN` が `long *` であることを思い出そう．この為、v, w の各成分を埋める際に `long` へのキャストが必要になる）．

GEN オブジェクトは `long` の配列なので、行列の各成分へのアクセスは、通常通り配列の参照として可能である．しかし、例えば行列は列ベクトルの配列なので、成分を得るには 2 回の参照が要り、しかも必要に応じて `GEN` へのキャストが必要になる．これを簡略化するために、マクロ `GEN gcoeff (GEN, int, int)` が定義されている．例えば `m` が行列の時、`gcoeff (m, 2, 3)` で 2 行 3 列の成分にアクセスできる．なお、これらは引数の範囲チェックなどは行わないので、注意が必要である．

longを返すlong coeff()もある．いずれもマクロなので、Pari スタックは消費しない．

2.2.3 多項式をつくる

多項式は係数の並びを保持するコンテナと言えるが、使用頻度が高いと思われるので別に述べる．

Pari における多項式は、変数を variable number という非負整数（long）で識別する．多変数の多項式は、係数も多項式であるような多項式として表現する．variable number が最小の変数が、主な変数となる．

多項式（t_POLY）も 2 つ目のコードワードを持ち、そこに符号（setsigne（GEN, long）で指定し、0 のときその多項式は 0, それ以外の時その多項式は非零である．符号を返すマクロはsigne(GEN))、有効長、即ち、実際に使われているワードが何処までか（setlgef(GEN, long)で指定し、lgef(GEN)で読み出す)、変数名（variable number, setvarn(GEN, long)で指定し、varn(GEN)で読み出す）などを保持する．なお、必ずしも多項式ではない GEN オブジェクトに対しても適用可能な、long gvar（GEN）という関数もある．多項式、冪級数に対してはその主変数の variable number を返し、それ以外のスカラーに対しては、BIGINT を返す[4]．

多項式の次数は、有効長から 3 を引いたものである．多項式の次数を得る為の、degree(GEN)というマクロが定義されている．

多項式は（GEN オブジェクトとしての管理情報を保持するコードワードを別にすれば）係数の単なるリストである．よって、各係数を参照するには、配列としてアクセスすればよい．係数が、整数などの即値でない場合（例えば整数を法とする整数、異なる変数に関する多項式……)、その係数の各成分にアクセスするには、やはりGEN へのキャストなどを交えて複数回配列を参照することになる．これを簡易化するために、マクロGEN gmael(GEN, int, int)が定義されている．pol = Mod（1, 3)*x + Mod(2, 3)なら、gmael（pol, 2, 1)で、1 次の項Mod(2,3)の代表元 2 を参照できる．係数がより複雑な構造を持つ場合のために、gmael3, gmael4, gmael5 まで定義されている．

多項式を作る簡単な例をリスト 4 として掲載する．

予め定義された多項式の配列polx[], polun[]が大域的に定義されている．pari_init()の呼び出しの際に、polx[MAXVARN], polun[MAXVARN]のみ初期化されている[5]．他に変数が必要な場合、long fetch_var(void)によって空いている variable number を取得する．この返り値は、Pari ライブラリ内で使われていない変数であることが保証されている．この値をsetvarn（GEN, long）の第 2 引数として指定する．面倒なようだが、このようにしないと意図せず他の多項式を壊すことがありうるためである（特に C で書いた関数を gp にリンクして使う場合)．

この変数には、通常のように対応する文字が存在しない．outbeaut()などで出力すると、変数に相

[4] 32bit の環境では $2^{15}-1$, 64bit の環境では $2^{31}-1$ である．
[5] MAXVARN は、32bit 環境か 64bit 環境かに応じて 16383, 65535 である．

リスト 4: 多項式をつくる

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN poly;
8:   pari_init (20000, 2);
9:   poly = cgetg (5, t_POL);
10:   setsigne (poly, 1);       /* set signature */
11:   setlgef (poly, 5);        /* set effective length */
12:   setvarn (poly, 0);        /* set variable number */
13:   poly[2] = un;
14:   poly[3] = deux;
15:   poly[4] = un;
16:   outbeaut (poly);          /* x^2+2x+1 */
17:   return 0;
18: }
```

当する部分は#⟨number⟩が出力される．文字を対応させる関数が、`void name_var(long, char *)`である．第 1 引数に variable number, 第 2 引数に対応させる文字を指定する．

`fetch_var()`と`name_var()`を合わせた働きをする関数`long fetch_user_var (char *)`も存在する．引数で指定した文字を持つ変数を作成し、その variable number を返す.

変数を保持する領域は、Pari スタックではなくヒープに取られる．従って通常のゴミ集めではこの領域の不要な箇所は回収されない．必要のなくなった変数を破棄するための関数が、`long delete_var(void)`である．これは、一番最近に割り当てられた変数を破棄し、その variable number を返す.

2.3 GEN オブジェクトの代入

GEN オブジェクトは、Pari スタック上のある領域を指すポインタであることを思い出そう．このポインタが指す領域に、それぞれの型に応じた構造を持ったデータが書き込まれている（例えば多項式ならば、初めの 2 つのコードワードに管理情報、引き続き各係数、などのように）．よって、GEN オブジェクトの代入と言った場合、大別して二通りが考えられる．

- ポインタとしての代入：`x`, `y`が GEN と宣言されているとき、`y = x;`
- GEN オブジェクトとしての代入：`x`, `y`が GEN と宣言されていて、`x`が多項式なら、`y`も同じ多項式を表すようにする

前者は`y`が Pari スタックの同じ場所を指すポインタとなる（従って双方は同じ GEN オブジェクトを表す）．後者では、`x`，`y`は同じ場所を指すのではないが、それらが表す数学的な対象は同じものとなる．

GEN オブジェクトとしての代入をする方法は、2 つある．1 つは`void gaffect(GEN, GEN)`を使う．第 1 引数を第 2 引数にコピーするが、第 2 引数は予め Pari スタック上に確保された領域を指していなければならない．のみならず、第 2 引数が指す領域は、第 1 引数が指す領域をコピーできるよう、様々な条件を満たさねばならない．リスト 5（10 頁）では、整数係数の多項式を`gaffect()`でコピーしている．`y`を初期化していることに注意．

`gaffect()`は非常に使いづらい関数であり、実際これを使うケースは稀である．整数、実数のように、他の GEN オブジェクトを成分として含まないような型については、メモリ管理の目的で使うことがある（メモリ管理で用いる関数`gerepile()`より効率が良い）．また、精度を切り捨てる方向への精度変換でも用いることがある．

もう 1 つは、`GEN gcopy(GEN)`を使う方法．これは、引数として与えられた GEN オブジェクトのコピーを保持できる領域を Pari スタック上に確保し、そこにコピーし、そのアドレスを返す．

最後に、`GEN gclone(GEN)`について説明する．これは、引数として与えられた GEN オブジェクトのクローンを作る．但し、クローンは Pari スタック上に作られるのではない．プログラムを実行中のプロセスのヒープに作られる．従って、Pari スタックを消費せず、後述のゴミ集めの対象にもならない．

クローンは、多くの GEN オブジェクトから参照されるような GEN オブジェクト（例えば整数や多項式の法の部分）を作成するのに用いる．そのような GEN オブジェクトを Pari スタックに作ると、ゴミ集めの際に意図せず回収されてしまうことを避けるようにコーディングしなくてはならないからである．

無暗に`gclone()`を使うと、ヒープを使い果たしてしまう．必要がなくなったクローンは、`void unclone(GEN)`によって消去する．

3 ライブラリ関数の呼び出し方

Pari ライブラリを用いる大きな理由は、その豊富な組み込み関数である．その全てのリストを挙げることはこの小文のよくするところではない[6]．Users guide［2］の 3 章を参照されたい．

Pari ライブラリで用意されている大半の関数は、幾つかの GEN オブジェクトを引数に取り、GEN オブジェクトまたは`long`を返す．後者の場合、Pari スタックを消費しない．前者の場合、返り値が`t_VEC`などのコンテナであることもある．それらの成分は、Pari スタック上隣接していることが保証されている．

[6]そうでなかったら Pari の Users guide［2］なみの厚さになってしまう．

リスト 5: 代入

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN x, y;
8:   pari_init (20000, 2);
9:   x = cgetg (4, t_POL); /* x is polynomial of degree 1 */
10:   setsigne (x, 1);
11:   setlgef (x, 4);
12:   setvarn (x, 0);
13:   x[2] = un;
14:   x[3] = deux; /* x=2x+1 */
15:
16:   y = x; /* y points same place as x */
17:   printf ("x = %s,\ny = %s\n",
18:           GENtostr (x), GENtostr (y));
19:   x[2] = (long) gneg (gun); /* If we modify x, y is modified */
20:   printf ("x = %s,\ny = %s\n\n",
21:           GENtostr (x), GENtostr (y));
22:
23:   y = cgetg (4, t_POL); /* allocate another memory */
24:   setsigne (y, 1);
25:   setlgef (y, 4);
26:   setvarn (y, 0);
27:   y[2] = lgeti (DEFAULTPREC);
28:   y[3] = lgeti (DEFAULTPREC); /* y is poly with int coeff. */
29:   gaffect (x, y);         /* copy whole of x to y */
30:   printf ("x = %s,\ny = %s\n",
31:           GENtostr (x), GENtostr (y));
32:   x[2] = un;              /* y is unchanged though we modified x */
33:   printf ("x = %s,\ny = %s\n\n",
34:           GENtostr (x), GENtostr (y));
35:
36:   y = gcopy (x);          /* make a copy of x, y point it */
37:   printf ("x = %s,\ny = %s\n",
38:           GENtostr (x), GENtostr (y));
39:   x[2] = (long) gneg (gun); /* y is unchanged though we modified x */
40:   printf ("x = %s,\ny = %s\n\n",
41:           GENtostr (x), GENtostr (y));
42:   return 0;
43: }
```

Users guide の 3 章には、gp の関数と、Pari ライブラリを用いた C プログラムを作成するときに呼び出すべき関数名とが記述してある．引数に現れる`prec`, `flag` は、常に C の`long` である．`prec` には`DEFAULTPREC` などのマクロで精度を指定する．`flag` の意味は個々の関数によって異なるが、多くの場合、その数を 2 進表記したとき、ある桁が 1 か 0 かで、オプションを指定するのである．

ある計算を行いたいときに、どのライブラリ関数を呼び出せばそれが実行できるのかは、初めは明らかではない．意図からその意図を実現してくれるライブラリ関数の索引のようなものは恐らく存在しないので、どうにかして「発見」しなければならない．

ひとつの方法は、gp でその計算を実現する関数を見つけることである．gp では、関数名は一定の規則にしたがって命名されているので、推測しやすい[7]．例えば代数体に関する計算は、`bnf...`（big number field）もしくは`nf...`（number field）の名前で始っている．もし gp に GNU Readline ライブラリがリンクしてあれば[8]、タブキーによる補完が使える．`bnf` とタイプしてから、続けてタブキーを二回押すと、`bnf` で始る関数名を持つ関数の一覧が表示される．

gp の関数名からライブラリ関数の名称を見つけるには、Users guide の第 3 章を調べるのだが、gp が extended help をサポートするようにコンパイルされていれば（これも`\v` で分かる）、gp のプロンプトに対し、`??`関数名とタイプすることで、Users guide の第 3 章の、その関数の部分のみが TEX で処理されて表示される．

4 入出力

Pari ライブラリを用いた C プログラムでの入出力について解説する．

4.1 出力

Pari ライブラリの出力関数のデフォルトの出力先は、`FILE *pari_outfile` という大域変数に格納されている値である．これは`pari_init()` の呼び出しの際に、`stdout` に初期化されている．

`void switchout(char *)` は、これら出力用関数の出力先を、与えられた文字列をファイル名にもつファイルに変更する．既にそのファイルが存在した場合は、アペンドオープンされる．`switchout(NULL)` によって、ファイルがクローズされ、初期状態の`stdout` に戻る．

`void outbrute(GEN)` は、引数の GEN オブジェクトを標準出力に書き出す．実数を出力する際には、その絶対値が 2^{-32} までならば固定小数点形式で（0.0000000141592 のように）、それ以下ならば指数を伴った科学形式で（$1.41592e{-8}$ のように）ある．また、ベクトルなどは $[1, 2; 2, 1]$ のように、

[7] 一方ライブラリ関数の命名規則は極めて混沌としており、歴史的な事情 (と、手間) もあって、改訂される見込みは薄い．

[8] これは、gp のプロンプトに`\v` とタイプしてバージョンを表示せると分かる．readline が組み込まれていれば、readline 4.1 enabled のように表示される．

一行で表示する．

void outmat(GEN)は、行列を表の形で表示する他、行末に改行文字を付加し、出力バッファのフラッシュを行う他は、outbrute()と同じである．

void texe(GEN, char, long)は、第 1 引数の GEN オブジェクトを、TEX の形式で出力する．実数を出力する際、第 2 引数のcharがf ならば固定小数点形式で、e ならば科学形式で、g ならば上記outbrute()と同様に桁数に応じて表示する．小数点以下何桁までを表示するかを、第 3 引数のlongで指定する．-1 のときは、その時点での精度分全ての桁を出力する．

リスト 6: 出力

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN m, v, w;
8:   pari_init (20000, 2);
9:   v = cgetg (3, t_COL);
10:   w = cgetg (3, t_COL);
11:   v[1] = un;
12:   v[2] = deux;
13:   w[1] = deux;
14:   w[2] = un;
15:   m = cgetg (3, t_MAT);
16:   m[1] = (long) v;
17:   m[2] = (long) w;  /* m=(1 2; 2 1) */
18:
19:   outbrute (m);
20:   outmat (m);          /* m is written to stdout */
21:
22:   switchout ("output_test.tex");
23:   texe (m, 'g', -1);  /* written to the file in TEXformat */
24:   switchout (NULL);
25:
26:   printf ("m = %s\n", GENtostr (m));
27:
28:   return 0;
29: }
```

この節までの例もしばしば用いてきたchar *GENtostr(GEN)は、GEN オブジェクトを然るべく文字列に変換する．文字列を保持する領域はGENtostr()内部で確保される．この領域を解放するのはプログラマの責任である．

4.2 入力

gp のプログラムの断片である文字列を、実行時に解釈して GEN オブジェクトを得ることが出来る．1 つの命令だけからなる場合、`GEN flisexpr(char)` を用いる．`GEN lisseq(char)` は複数の命令の場合に用いるが、空白を予め全て取り除いておかねばならない[9]．

予めファイルに書き込まれた文字列を一行読み込んで、gp のプログラムとして評価し、結果となる GEN オブジェクトを返す関数が、`GEN lisGEN(FILE *)` である．ただし、この関数はファイルの末尾の取り扱いに難があり[10]使いづらい．

筆者はリスト 7, 8 のようなものを使っている[11]．このサンプルは[12]、引数を与えずに実行された場合、標準入力から一行読み取り、それを gp のプログラムとして解釈し、結果を表示する．`Ctl-D` か、入力にエラーがあった場合、終了する．非常に簡略化された gp である．引数を指定した場合、そのファイルの各行を gp のプログラムとして解釈し、一行毎に結果を返す．ファイルの末尾に達するか、エラーに遭遇すると終了する．

4.3 エラー出力

`void err(long, ...)` は、エラーを報告する為の Pari ライブラリ関数である．第 1 引数で報告すべきエラーを指定する．各エラーはヘッダファイル `parierr.h` で `enum` として宣言されている．エラーの種類によって、単に警告を表示するだけで実行を続けたり、警告を表示して実行を中止したりする．

リスト 8 の末尾では、`talker` を指定している．この場合、第 2 引数の文字列を表示して、実行を中止する．`warner` を指定すると、第 2 引数の文字列を表示して実行を続ける．

5 メモリ管理

5.1 Pari スタックと avma

はじめに述べたように、`void pari_init(long, long)` が Pari スタックを初期化する．第 1 引数が Pari スタックのサイズを決める．実際には、第 1 引数に `sizeof(long)` をかけたバイト分のメモリが確保される．Pari ライブラリに収められている関数はこれ以降、基本的にはこの領域を用いて計算をすすめる．

---

[9]自動的に文字列の空白を取り除き、得られた文字列を評価する `GEN flisseq(char)` という関数があったのだが、あるバージョンから何故かなくなった．

[10]関数内部で用いられている `fgets()` がブロックしてしまう．Pari 開発陣には報告済みである．

[11]後者は Pari ライブラリ内の lisGEN() と言う関数を僅かに修正したもの

[12]前者を lisGEN-ex.c というファイルに、後者を lisGEN.c というファイルに打ち込んだとすると、コンパイルするには cc -o lisGEN-ex lisGEN-ex.c lisGEN.c -I/usr/local/include/pari -L/usr/local/include -lpari -lm のようにする．

リスト 7: lisGEN() を用いた入力の例

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <pari.h>
4:
5: int main (int, char *[]);
6: extern GEN mylisGEN (FILE *);
7:
8: int
9: main (int ac, char *av[])
10: {
11:   FILE *f;
12:   GEN a;
13:   size_t len;
14:   char *s;
15:
16:   if (ac == 1)
17:     {
18:       f = stdin;
19:     }
20:   else
21:     {
22:       f = fopen (av[1], "r");
23:       if (f == NULL)
24:         {
25:           perror ("fopen");
26:           exit (EXIT_FAILURE);
27:         }
28:     }
29:
30:   pari_init (200000, 20000);
31:
32:   while (1)
33:     {
34:       a = mylisGEN (f);
35:       if (a != NULL)
36:         outbeaut (a);
37:       else
38:         break;
39:     }
40:   fclose (f);
41:   return 0;
42: }
```

リスト 8: lisGEN() を用いた入力の例（続）

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <pari.h>
4:
5: GEN
6: mylisGEN(FILE *fi)
7: {
8:   long size = 512, n = size;
9:   char *buf = gpmalloc(n), *s = buf;
10:
11:   for(;fgets(s, n, fi);)
12:     {
13:       if (s[strlen(s)-1] == '\n')
14:       {
15:         GEN x = flisexpr(buf);
16:         free(buf); return x;
17:       }
18:       buf = gprealloc(buf, size<<1, size);
19:       s = buf + (size-1); n = size+1; size <<= 1;
20:     }
21:   if (feof (fi))
22:     return NULL;
23:   else
24:     err (talker, "failed reading from file");
25: }
```

第 2 引数は、予め計算する素数の上限である．例えば 100000 を指定すれば、そこまでの素数を予め計算し、保存しておく[13].

大域変数`bot`, `top`が定義されており、それぞれ Pari スタックの最初と最後を表す．（`top - bot`）`/ sizeof(long)`が第 1 引数に等しい．また、現時点での Pari スタックの位置を示す大域変数`avma`が定義されている．**av**ailable **m**emory **a**dress の略である．Pari スタックに GEN オブジェクトが積まれるにつれ、`avma`は減少していく．これら 3 つの変数はいずれも`long`である．

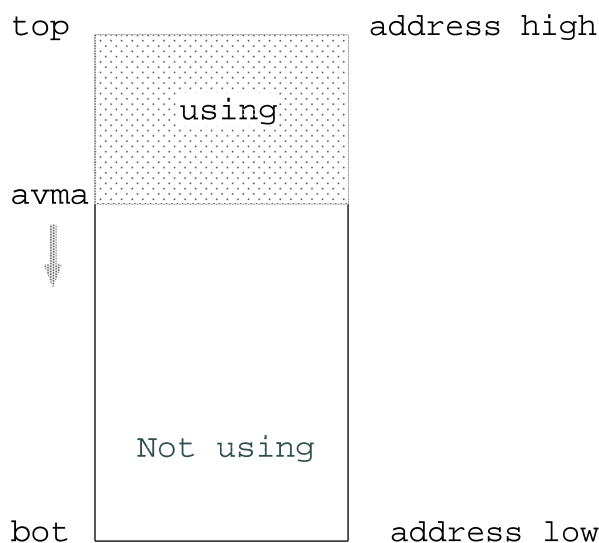

図 9: Pari スタックの模式図

`pari_init()`を呼んだ直後は、`avma = top`であり、`cgetg()`や、その他 Pari ライブラリ関数の呼び出しにつれて`avma`は減少していく．また、`avma`は常に最新の GEN オブジェクトを指している．（`GEN`）`avma`によって、該当する GEN オブジェクトにアクセスできる．

`avma`が`bot`を下回ると、`the Pari stack overflows !`というエラーメッセージが出力され、プログラムは停止する．これを防ぐために、計算の過程で必要なくなった GEN オブジェクトを Pari スタックから回収する作業が必要になる．

5.2 avma の書き換えによるゴミ集め

もっとも単純なゴミ集めの方法が、`avma`を直接書き換えることである．プログラムのある時点での`avma`を記録しておき（`long ltop = avma;`）、別のある時点でそれを`avma`に代入する（`avma`

[13]`pari_init()`の第 2 引数には、436273009 より大きい数は指定できない．また、実際に保存されるのは隣接する素数の差である．

リスト 10: avma, bot, top

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN i;
8:   pari_init (200000, 2);
9:   printf ("bot = %ld, top = %ld, avma = %ld\n",
10:           bot, top, avma); /* reports initial state */
11:   i = cgeti (DEFAULTPREC); /* This takes 4 codewords */
12:   printf ("avma = %ld\n", avma); /* decrese 4 * sizeof(long) */
13:   return 0;
14: }
```

= ltop;）．これによって、次の GEN オブジェクトはltopとして記録された地点の次に置かれ、もともとそこにあった GEN オブジェクトは上書きされる．

即ち、プログラムを書いていて、ある時点からある時点までに消費される Pari スタックが、ある時点で全て不必要になることが分かっているときに、この方法でのゴミ集めが可能なのである．

5.3　cgiv() によるゴミ集め

Pari スタックに一番最近作られた GEN オブジェクトを破棄する関数が、void cgiv(GEN) である．全ての GEN オブジェクトは、第 1 のコードワードに、自分自身が Pari スタックをどれだけ消費しているかを記録している（long lg(GEN) によりアクセスできる）．cgiv() は、この情報を用いてavma を巻き戻しているだけであり、実質は上記のavma の書き換えによるものと同じである．

この方法は、ある時点でもっとも新しい GEN オブジェクトがゴミである（例えばそのオブジェクトをファイルに書出し、以下の計算では必要ない）という場合にのみ有効である．

5.4　gerepile() によるゴミ集め

「残念ながら人生はそのように単純ではないので[14]」、Pari スタックに最近積まれた幾つかの GEN オブジェクトは使える状態のまま、不必要になった幾つかのオブジェクトをシステムに返却したい、ということがままある．

GEN gerepile (long, long, GEN) を使うと、第 1 引数ltop から第 2 引数lbot までの区間

[14]“Pari’s users guide”［2］のgerepile() の項の書出し．

をゴミとして返却し（従って`ltop > lbot`でなければならない）、`lbot`から`avma`までを空いた領域にスライドさせることができる．返り値は第3引数のGENオブジェクトが、この操作によって更新されたそのアドレスである（第3引数のGENオブジェクトが`ltop`と`lbot`の間にある場合は、エラーメッセージが表示される）．

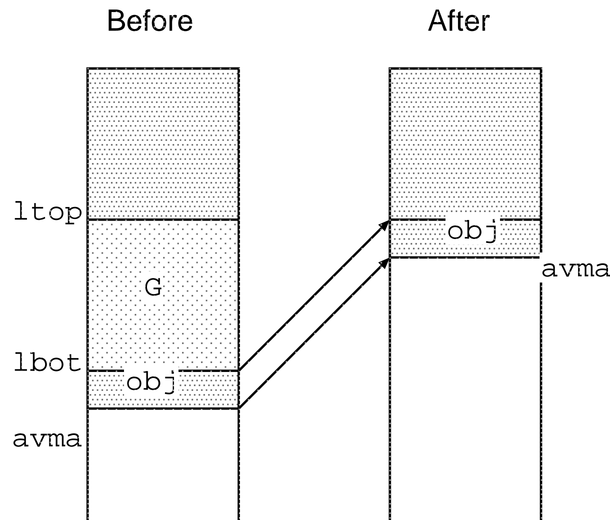

図 11: gerepile() の前後．Gがゴミ．

リスト12では、ベクトル$(3,4)$の長さの二乗を求めている．`a`，`b`が成分の二乗だが、その計算の前に`ltop`、後で`lbot`に、`avma`を保存している．17行目で、この区間にあるGENオブジェクトをゴミとしてシステムへ返却し、`gadd`の返り値であるGENオブジェクトの更新されたアドレスを受け取っている．

このように、`gerepile()`は連続した領域のゴミしか回収することができない．従って、ゴミが連続するように計算の順序を工夫することが望ましい．GENオブジェクト`x`，`y`について、ベクトル(x^2+y, y^2+x)を計算することを考える（リスト13）[15]．

何も考えないと、必要のない`p1`，`p3`が、必要な結果の1つである`p2`で分断されてしまう．そこで、`p1`，`p3`を先に計算し、Pariスタックで隣接するように順番を変えると、`gerepile()`で返却出来るのである．

`gerepile()`は、ある領域をゴミとしてシステムへ返却し、第3引数のGENオブジェクトが更新されて何処へ行ったかを返す関数であった．では、ゴミ集めの前後で複数のGENオブジェクトの行方を追いたい場合にはどうすれば良いだろうか？

その為の関数が`void gerepilemanysp (long, long, GEN *[], long)`である．第1引数

[15] Users guide［2］にある例．

リスト 12: gerepile() を使う一番簡単な例

```
1: #include <stdio.h>
2: #include <pari.h>
3:
4: int
5: main (void)
6: {
7:   GEN a, b, c, v;
8:   long ltop, lbot;
9:
10:   pari_init (200000, 2);
11:
12:   v = lisseq ("[3,4]");
13:   ltop = avma;
14:   a = gsqr ((GEN) v[1]); /* v[1]^2 */
15:   b = gsqr ((GEN) v[2]); /* v[2]^2 */
16:   lbot = avma;
17:   c = gerepile (ltop, lbot, gadd (a, b));
18:   printf ("|v|^2 = %s\n", GENtostr (c));
19:   return 0;
20: }
```

リスト 13: gerepile() を使うために計算の順序をかえる

```
1: 何も考えない場合
2: p1 = gsqr (x); p2 = gadd (p1, y);
3: p3 = gsqr (y); p4 = gadd (p3, x);
4: z = cgetg (3, t_VEC);
5: z[1] = (long) p2;
6: z[2] = (long) p3;
7:
8: p1 と p3 が必要ないので、先に計算してしまう
9: p1 = gsqr (x); p3 = gsqr (y);
10: p2 = gadd (p1, y); p4 = gadd (p3, x);
11: z = cgetg (3, t_VEC);
12: z[1] = (long) p2;
13: z[2] = (long) p3;
```

の`ltop`から第 2 引数の`lbot`までをゴミとしてシステムに返す．第 3 引数の GEN オブジェクトへのポインタの配列によって、このゴミ集めの後も必要なオブジェクトを指定する（第 4 引数は、第 3 引数の配列が幾つの要素からなるのかを指定する）．第 3 引数の配列の各成分が指す GEN オブジェクトは、ゴミ集め後然るべく更新される．`gerepile()`同様、ゴミ集め後も必要な GEN オブジェクト達は、`ltop`から`lbot`の間にあってはいけない．

また、`void gerepilemany (long, GEN *[], long)`という関数もある．第 1 引数の`ltop`から`avma`までをゴミとしてシステムに返す．第 2 引数の GEN オブジェクトへのポインタの配列によって、このゴミ集めの後も必要なオブジェクトを指定する（第 3 引数は、第 2 引数の配列が幾つの要素からなるのかを指定する）．

`gerepilemany()`を使う場合は、ゴミ集め後も必要な GEN オブジェクトがゴミ集めの対象になる区間に置かれていても良い．`gerepilemany()`内部で一旦安全な領域にコピーが作成され、ゴミ集め後に Pari スタックに書き戻されるのである．

この関数の第 2 引数を 1 つの元からなる配列とすれば、`gerepile()`の堅牢なバージョンとして使うこともできるが、毎回コピーが発生することから分かるように、効率面でペナルティがある．

6 例

この節では、幾つかの例を挙げ、簡単な解説を行う．

6.1 2 次体の場合

リスト 14 は、指定した範囲で虚 2 次体の類数を計算し、それが 3 で割りきれるもののリストをつくるプログラムである．簡単のため、判別式の範囲と、結果を書出すファイル名は固定している[16]．

32 行目の`hclassno(GEN)`が（Hurwitz-Kronecker の）類数を、簡約された 2 次形式の個数を数えることで計算する関数である．29 行目で基本判別式の場合のみ計算しているので、対応する虚 2 次体の類数と等しい．Pari ライブラリには、Shanks の baby-step giant-step 法で 2 次体の類数を計算する`GEN classno(GEN)`という関数があるが、虚 2 次体に対しては、類群のランクが大きい場合に結果が不正確になるという制限がある．

36 行目からの条件文は、`avma`が Pari スタックの半分を過ぎたら、`avma`の書き換えによるゴミ集めをする、という意味である．毎回`avma`を書き換えれば必要なメモリ領域は少なくてすむが、実行時間が増してしまう．

23 行目で`gmodulo()`を、33 行目で`gmodulcp()`を使っている．前者は結果をヒープに作成し、後者は Pari スタックに作成するという違いがある．

[16]引数で指定できるようにするのも簡単です．

23 行目で使われている関数long gegal(GEN, GEN)は、2 つの GEN オブジェクトが等しい（帰り値 1）か否か（帰り値 0）を返す．実数（t_REAL の比較にはlong gcmp(GEN, GEN)を用いる．これはx-y の符号を返す）．また、long gcmp0 (GEN)はgegal (GEN, gzero)と同じ意味で、long gcmp1 (GEN)はgegal (GEN, gun)と同じ意味で、long gcmp_1 (GEN)はgegal (GEN, gneg (gun))（即ち −1 との比較）と同じ意味である．

リスト 15 は、関数GEN quadclassunit0 (GEN, long, GEN, long)の使用例である．第 1 引数として与えられた判別式を持つ 2 次体の、類数、類群、基本単数などを計算する．引数についての詳細は、Users guide［2］のquadclassunit の項を参照．返り値は 6 成分からなるベクトルで、類数、類群（巡回群の積として）、類群の巡回部分群の生成元（2 次形式で表現される）、単数基準などからなる．13 行目で、第 3 引数のベクトルを作っている．このパラメタは、計算時間、計算に必要なメモリの総量、計算の精度に影響する．詳細は Cohen［3］を参照のこと．

6.2 一般の代数体の計算

有理数係数の多項式を与えて、それが定義する代数体を計算するための関数が、GEN bnfclassunit0 (GEN, long, GEN, long)である．返り値は 10 成分からなるベクトルで、実・虚素点の個数からなるベクトル、判別式、整数底、類数・類群・各巡回部分群の生成元、単数基準、考えている体に含まれる 1 の冪根の群、基本単数などからなる．引数、返り値に関する詳細は Users guide［2］のbnfclassunit の項を参照.

リスト 16 では、判別式 −23 の虚 2 次体の Hilbert 類体を定義する多項式をまず求め（11 行目、quadhilbert()）、15 行目で実際にその体を計算している．

6.3 コツ

これまでもサンプルプログラムの中で見てきたように、flisexpr(),lisseq() などの、文字列を gp のスクリプトとして評価し、その結果を GEN オブジェクトとして返す関数を使うと、プログラムを簡略化できる．引数によってこれらの関数へ渡す文字列を変更することも、C のレベルで文字列操作を行うことで（sprintf() などを用いて）可能である．ただし、flisexpr(),lisseq() に gp のfor(), sum() などを渡すと、パフォーマンスが非常に悪くなる．

今まで見てきた例は、いずれもmain() 関数のみからなる小さな例だった．複数の関数からなる自家製のライブラリをつくる場合は、それを構成する関数それぞれの中で、引数と返り値の型チェックを行うことを勧める．Pari ライブラリが提供する型は種々あるが、C のレベルではいずれも同じ GEN であることは繰り返し述べた通りである．従って、自分で定義した関数へ渡されているオブジェクトの型が、予期した通りのものであるか否かは、C コンパイラの型チェックでは検出できない．

リスト 14: 類数が 3 で割れる虚 2 次体をリストする

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <pari.h>
4:
5: int
6: main (void)
7: {
8:   long from = -4, upto = -100000, ltop, lim, i;
9:   GEN gtrois, d, h, zeromodtrois;
10:   FILE *fp;
11:
12:   fp = fopen ("qfbex.txt", "w");
13:   if (fp == NULL)
14:     {
15:       perror ("fopen()");
16:       exit (EXIT_FAILURE);
17:     }
18:
19:   pari_init (500000, 20000);
20:   fprintf (fp, "# top = %ld, bot = %ld\n", top, bot);
21:   timer ();
22:   gtrois = stoi (3L);
23:   zeromodtrois = gmodulo (gzero, gtrois);
24:   lim = top - ((top - bot) / 2);
25:   ltop = avma;
26:   for (i = from; i > upto; i--)
27:     {
28:       d = stoi (i);
29:       if (!isfundamental (d))
30:         continue;
31:
32:       h = hclassno (gneg (d));
33:       if (gegal (gmodulcp (h, gtrois), zeromodtrois))
34:         fprintf (fp, "h(%s) = %s\n", GENtostr (d), GENtostr (h));
35:
36:       if (avma < lim)
37:         {
38:           avma = ltop;
39:           fprintf (stderr, "GC done (avma = %ld)\n", avma);
40:         }
41:     }
42:   fprintf (fp, "# total comsumption time = %ld\n", timer ());
43:   fclose (fp);
44:   return 0;
45: }
```

リスト 15: 虚 2 次体の計算

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <pari.h>
4:
5: int
6: main (void)
7: {
8:   GEN d, qcu, data, tech;
9:   int i;
10:
11:   pari_init (500000, 10000);
12:   d = stoi (-23L);
13:   tech = cgetg (1, t_VEC);
14:   qcu = quadclassunit0 (d, 0, tech, DEFAULTPREC);
15:
16:   for (i = 1; i < 6; i++)
17:     printf ("%s\n", GENtostr ((GEN) qcu[i]));
18:
19:   return 0;
20: }
```

リスト 16: 虚 2 次体の Hilbert 類体の計算

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <pari.h>
4:
5: int
6: main (void)
7: {
8:   GEN d, poly, bnfcu, tech;
9:   pari_init (500000, 10000);
10:   d = stoi (-23L);
11:   poly = quadhilbert (d, 0, DEFAULTPREC);
12:   printf ("poly = %s\n", GENtostr (poly));
13:
14:   tech = cgetg (1, t_VEC);
15:   bnfcu = bnfclassunit0 (poly, 0, tech, DEFAULTPREC);
16:   printf ("bnfclassunit() = %s\n", GENtostr (bnfcu));
17:   return 0;
18: }
```

例えば、整数（t_INT）を取るはずの関数に、間違えて実数（t_REAL）を渡すようなプログラムを書いてしまっても、コンパイラには GEN が渡されていることしか分からないので、コンパイルは問題なく通ってしまう．リスト 17 のように、assert() を用いて関数内部で型チェックをしていれば、実行時ではあるが、このような型の不整合を検出することができる．

リスト 17: assert() による型チェック

```
1: #include <assert.h>
2:
3: GEN
4: afunction (GEN a, GEN b)
5: {
6:   GEN retval;
7:
8:   assert (typ (a) == t_INT && typ (b) == t_INT);
9:
10:   /* 以下関数の本体のコード */
11:
12:   assert (typ (retval) == t_INT);
13:   return (retval);
14: }
```

assert() ではなく、if (typ() != t_INT) err (talker, ...); のような型チェックでも勿論よい．

7　その他

7.1　サンプルのコンパイル法

本文中で述べたサンプルは、リスト 18 のようにしてコンパイル出来る．サンプルを file.c というファイルに打ち込んだと仮定している．

リスト 18: サンプルのコンパイル

```
$ cc -o file file.c -I/usr/local/include/pari -L/usr/local/lib -lpari -lm
```

勿論、コンパイラのファイル名や、コンパイラに与えるオプションは個々の環境に依存する．例では、cc が C コンパイラであり、-I で指定しているのが Pari ライブラリのヘッダファイルを置いたディレクトリ、-L で指定しているのが Pari ライブラリの本体の置いてあるディレクトリである[17]．

[17] $はシェル（この場合 bash）のプロンプトです．

7.2 GP2C

Bill Allombert 氏が、gp のスクリプトを C に変換する、gp2c というトランスレータを公開している．2000 年 12 月 23 日に、メーリングリスト`pari-dev@cr.yp.to`にアナウンスが流れた．これは、次の 2 つの方法のいずれかで入手可能である．

7.2.1 gp2c の入手とコンパイル

Pari のソースツリーを既に CVS で入手している場合は、リスト 19 のような操作でソースツリーが展開される．

リスト 19: gp2c を CVS で入手する

```
1:  $ cd pari-source-tree
2:  $ cvs -z 3 checkout gp2c
```

リスト 20: gp2c をビルドする

```
1:  $ cd gp2c
2:  $ ./cvsinit
3:  $ ./configure
4:  $ make
```

リスト 20 のようにして、実行できるものが出来る．但し、automake, autoconf, m4, yacc, lex, perl が（若しくはそれらの相当品が）必要である．

あるいは、`ftp://megrez.math.u-bordeaux.fr/pub/pari/GP2C`から FTP によっても入手可能である．こちらは展開すれば直ぐにビルドできる．

7.2.2 gp2c の使い方

リスト 21 は、類数が 3 で割りきれる虚 2 次体で、その Hilbert 類体の類数も 3 で割りきれるものを指定された範囲で探索する gp のスクリプトである．これを C に変換するには、次のようにする（リスト 22）．上記の gp のスクリプトを、`cfqf.gp`というファイルに保存したとして、 これで、上記の関数が C に変換されたものがダイナミックリンクできるライブラリとしてコンパイルされた[18]．なお、gp2c へのオプション`-g`は、ゴミ集めに関するコードも生成するという意味である．

[18]コンパイルオプションなどは環境によってかわるので注意されたい．gp2c が生成した C のソースの先頭部分にコメントとして、コンパイルの仕方、gp への組み込み方が書いてある．

リスト 21: 虚 2 次体の Hilbert 類体に関する計算

```
1: {cf_qf(from,to)=local(D,cf,cfh);
2: for(D=from,to,
3:       if(isfundamental(-1*D) \
4:          && Mod(qfbhclassno(-1*D),3)==Mod(0,3),
5:               cf=bnfinit(quadhilbert(-1*D));
6:               cfh=cf[8][1][1];
7:                 if(Mod(cfh,3)==Mod(0,3),
8:                       print("D= ",D," h= ",cfh))))
9: }
```

リスト 22: gp スクリプトを C に変換する

```
1:   $ cd gp2cの置いてあるディレクトリ
2:   $ ./gp2c -g cfqf.gp > cfqf.c
3:   $ cc -c cfqf.c -I/usr/local/include
4:   $ ld -o cfqf.gp.so cfqf.o -shared
```

これを gp に組み込むには、以下のようにする．まず gp を起動して、次のように`install`する．実行するのは通常の gp の組み込み関数の起動と同じである（リスト 23）．なお、gp2c が生成した C のソースのコンパイルの仕方と、それを gp へ組み込む際の`install`への引数は、そのソースファイルの先頭部分にコメントとして書いてある[19].

リスト 23: gp に組み込む

```
1: $ gp
2: ? install("cf_qf","vD0,G,D0,G,p","cf_qf_c","./cfqf.gp.so")
3: ? cf_qf_c(1, 400)
4: D= 391 h= 3
```

7.2.3 注意

公開されて間もないので、筆者も余り gp2c を使い込んでいないが、多少制限もある[20]．直ぐに気が付くのは、gp のメンバー関数が使えないことである．これは、例えば`bnfinit()`の返り値`bnf`に対して、`bnf.clgp`でイデアル類群、`bnf.no`で類数を参照できるというものである[21]．リスト 21 の 6 行目で`cfh`を`cf.no`としなかったのはその為である．

[19] `install()` への第 2 引数のコードについては、`Users guide` [2] の `4.9` 節を参照されたい.

[20] ソースツリーに含まれる、BUGS を見よ.

[21] gp を起動して、`?.` とタイプするとメンバー関数の一覧が表示される

しかし、gp でプロトタイプを作り、gp2c で C のコードに変換したものが意図した通りに動けば、極めて生産性を上げることが出来るものと思われ、今後の発展が非常に楽しみである.

7.3 Pari-gp の入手先・情報源

Pari-gp は、開発陣が運営する web ページ`http://www.parigp-home.de/`から入手できる他、例えば東京都立大学 TNT サーバ`ftp://tnt.math.metro-u.ac.jp/`にもミラーされている. インストールの仕方などは、同梱のドキュメントにある.

Pari-gp に関する情報は、上記 web ページの他、開発者らが運営している 3 つのメーリングリストからも得られる. `pari-announce@list.cr.yp.to`には、正式バージョンが公開された場合などに案内が流れる. `pari-users@list.cr.yp.to`では、gp や Pari ライブラリの使い方などの質問と解答が寄せられている. `pari-dev@list.cr.yp.to`は開発者向けのメーリングリストで、バグ情報やパッチ、α, β バージョンのアナウンスなどが流される. これらのメーリングリストに参加するには、例えば pari-announce に参加する場合、`pari-announce-subscribe@list.cr.yp.to`に空メールを送ればよい. メーリングリストサーバから自動応答が返ってくるので、以下の手続きはそのメッセージに書いてある通りにする. メーリングリストのログは、上記 Pari-gp の web ページに蓄積されており、参加せずとも内容を読むことができる.

本稿は、報告集の原稿として訂正後も、少しづつ手を入れて、例えば Pari のバージョンアップなどに追随していく予定である. 筆者の運営する Web ページ`http://www.sci.toyama-u.ac.jp/~iwao/pari-gp.html` を参照されたい.

参考文献

[1] D. M. リッチー B. M. カーニハン. プログラミング言語 C 第 2 版. 共立出版, 1994.

[2] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. *User's Guide to PARI / GP*. 2000.

[3] Henri Cohen. *A course in computational algebraic number theory*. Springer-Verlag, Berlin, 1993.

[4] S. P. ハービンソン, G. L. スティール. 新・詳説 C 言語 H&S リファレンス. ソフトバンク, 1994.

木村 巌 ⟨`iwao@sci.toyama-u.ac.jp`⟩
富山大学理学部数学科
〒 930-8555 富山市五福 3190
`http://www.sci.toyama-u.ac.jp/~iwao/`