Guide GMP Pour l'Impatient(e)

"GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers.", gmplib.org 2013.
"Le Guide*GMP* Pour l'Impatient(e) permet à chacun de mesurer l'importance des nombres et de leur grande valeur dans les programmes.", nj 2013.

Point de départ : une opération avec des nombres dépassant les valeurs maximales des types primitifs.
Objectif : utiliser les types GMP, grands entiers, rationnels et nombres à virgules avec des opérateurs à précision choisie.

lien utile : https ://gmplib.org/

1 Trace

```
pow(10,i) : 10000000000000000.000000 ou 1e+16
pow(10,i) : 100000000000000000.000000 ou 1e+17
...
pow(10,i) : 1000000000000000000000.000000 ou 1e+22
pow(10,i) : 999999999999999991611392.000000 ou 1e+23
pow(10,i) : 9999999999999999983222784.000000 ou 1e+24
pow(10,i) : 10000000000000000905969664.000000 ou 1e+25
pow(10,i) : 100000000000000004764729344.000000 ou 1e+26
pow(10,i) : 1000000000000000013287555072.000000 ou 1e+27
pow(10,i) : 9999999999999999583119736832.000000 ou 1e+28
pow(10,i) : 99999999999999991433150857216.000000 ou 1e+29
pow(10,i) : 1000000000000000019884624838656.000000 ou 1e+30
pow(10,i) : 9999999999999999635896294965248.000000 ou 1e+31
pow(10,i) : 10000000000000000536616220439 3472.000000 ou 1e+32
1E32 : 10000000000000000536616220439 3472.000000 ou 1e+32
---------------------
pow(10,i) + 0.1 : 10000000000.100000 ou 1e+10
pow(10,i) + 0.1 : 100000000000.100006 ou 1e+11
pow(10,i) + 0.1 : 1000000000000.099976 ou 1e+12
pow(10,i) + 0.1 : 10000000000000.099609 ou 1e+13
pow(10,i) + 0.1 : 100000000000000.093750 ou 1e+14
pow(10,i) + 0.1 : 1000000000000000.125000 ou 1e+15
pow(10,i) + 0.1 : 10000000000000000.000000 ou 1e+16
pow(10,i) + 0.1 : 100000000000000000.000000 ou 1e+17
...
pow(10,i) + 0.1 : 10000000000000000000000.000000 ou 1e+22
pow(10,i) + 0.1 : 999999999999999991611392.000000 ou 1e+23
pow(10,i) + 0.1 : 9999999999999999983222784.000000 ou 1e+24
---------------------
mpz_pow_ui(10, i) : 10000000000000000
mpz_pow_ui(10, i) : 100000000000000000
...
mpz_pow_ui(10, i) : 100000000000000000000000000000000
mpz_pow_ui(10, i) : 1000000000000000000000000000000000
---------------------
mpf_pow_ui(10, i) + 0.1 : 10000000000000000.100000000000000000000000000000000
mpf_pow_ui(10, i) + 0.1 : 100000000000000000.100000000000000000000000000000000
...
mpf_pow_ui(10, i) + 0.1 : 10000000000000000000000000000000000.100000000000000000000000000000000
mpf_pow_ui(10, i) + 0.1 : 100000000000000000000000000000000000.100000000000000000000000000000000
---------------------
```

2 Code source

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include <gmp.h>

#define BP 2048 /* minimal operators' size */

#define PDPV(s,v) printf("%s: %f ou %g\n", s, v, v)
#define PMPZPV(s,v) gmp_printf("%s: %Zd\n", s, v)
#define PMPFPV30(s,v) gmp_printf("%s: %F.30f\n", s, v)
#define PMPFPV50(s,v) gmp_printf("%s: %F.50f\n", s, v)
#define PLINE printf("--------------------\n")

void sansGmp(void);
void avecGmp_z(void);
void avecGmp_f(void);
void avecGmp(void);

/* gcc gmpSimple.c -lgmp -lm */
int main(void) {
  sansGmp();
  PLINE;
  avecGmp_z();
  PLINE;
  avecGmp_f();
  PLINE;
  return 0;
}
void sansGmp(void) {
  int i;
  double dpv;

  /*
   * double: 1 bit s, 52 bits m et 11 bits e
   * m sur 52 bits donc 4.503.599.627.370.496 soit 16 digits en base 10
   * et les problemes viennent apres ces 16 digits pour un affichage decimal
   */

  /* puisances de 10 avec la fonction pow */
  for(i = 16; i < 33;i++)
    dpv = pow(10, i), PDPV("pow(10,i) ", dpv);
  /* et le probleme ne vient pas de pow */
  dpv = 1E32, PDPV("1E32 ", dpv);
  PLINE;

  /* avec m sur 16 digits en base 10
   * pow(10, i) + 0.1 devient ...
   */
  for(i = 10; i < 25;i++)
    dpv = pow(10, i) + 0.1, PDPV("pow(10,i) + 0.1 ", dpv);
}
void avecGmp_z(void) {
  unsigned long int i;
  mpz_t mpzpv, val;

  mpz_init(mpzpv);
  mpz_init(val);
  mpz_set_ui(val, 10);

  for(i = 16; i < 33; i++) {
    mpz_pow_ui(mpzpv, val, i);
    PMPZPV("mpz_pow_ui(10, i) ", mpzpv);
  }
  mpz_clear(mpzpv);
  mpz_clear(val);
}
void avecGmp_f(void) {
  unsigned long int i;
  mpf_t mpfpv, val, zu;

  mpf_init2(mpfpv, BP);
  mpf_init2(val, BP);
  mpf_init2(zu, BP);
  mpf_set_ui(val, 10);
  /* attention, init avec flottant deconseillee
* init avec set_ui 1 et division par 10
*/
  mpf_set_ui(zu, 1);
  mpf_div_ui(zu, zu, 10);

  for(i = 16; i < 33; i++) {
    mpf_pow_ui(mpfpv, val, i);
    mpf_add(mpfpv, mpfpv, zu);
    PMPFPV30("mpf_pow_ui(10, i) + 0.1 ", mpfpv);
  }

  mpf_clear(mpfpv);
  mpf_clear(val);
  mpf_clear(zu);
}
```