```
-- Code de Reed Solomon
-- Package : Gestion Fraction
-- Cedric Dos Reis - 03.12.2017
-- Algorithmique

package Gestion_Fractions is
    DIV_PAR_ZERO : exception;

    type T_Fraction is record
        num : Integer := 0;
        den : Integer := 1;
    end record;

    procedure Get(f: out T_Fraction);
    procedure Put(f : T_Fraction);
    procedure Reduire(f : in out T_Fraction);

    function "+"(f1, f2 : T_Fraction) return T_Fraction;
    function "-"(f1, f2 : T_Fraction) return T_Fraction;
    function "*"(f1, f2 : T_Fraction) return T_Fraction;
    function "*"(f : T_Fraction; n: Integer) return T_Fraction;
    function "*"(n: Integer; f : T_Fraction) return T_Fraction;
    function "/"(f1, f2 : T_Fraction) return T_Fraction;
    function "/"(f : T_Fraction; n : Integer) return T_Fraction;
    function "/"(n : Integer; f : T_Fraction) return T_Fraction;
    function "**"(f : T_Fraction; n : Integer ) return T_Fraction;
    function PGCDof(n1, n2: Integer) return Integer;

    private
    procedure Verifie(f : T_Fraction);
end Gestion_Fractions;



-- Code de Reed Solomon
-- Package body : Gestion Fraction
-- Cedric Dos Reis - 03.12.2017
-- Algorithmique

with Text_IO;                       use Text_IO;
WITH Ada.Command_Line;                USE Ada.Command_Line;
with Ada.Integer_Text_IO;           use Ada.Integer_Text_IO;
with Ada.Float_Text_IO;             use Ada.Float_Text_IO;
with Ada.Numerics.Float_Random;     use Ada.Numerics.Float_Random;
with Ada.Numerics.Generic_Elementary_Functions;

package body Gestion_Fractions is
    -- Lit la fraction entrée
    procedure Get(f: out T_Fraction) is
   begin
      get(f.Num);
      get(f.Den);
      Reduire(f);
   end;

    -- Affiche la fraction
    procedure Put(f : in T_Fraction) is
    begin
        Put(Integer'Image(f.num));
        Put(Integer'Image(f.den));
    end Put;

    -- Tranforme la fraction reçu en fraction irreductible
    procedure Reduire(f : in out T_Fraction) is
        pgcd : Integer := 1;
    begin
        -- verifie si la fraction n'est pas nul
        -- sinon reduit normalement la fraction
        if f.num /= 0 then
            pgcd := PGCDof(f.num, f.den);
            f.num := f.num / pgcd;
            f.den := f.den / pgcd;
        end if;
```

```
74      end Reduire;
75
76      ---------- FONCTIONS ----------
77      -- Somme de deux fraction, retourne une fraction irreductible
78      function "+"(f1, f2 : T_Fraction) return T_Fraction is
79          result : T_Fraction;
80      begin
81          Verifie(f1);
82          Verifie(f2);
83          result.num := f1.num * f2.den + f1.den * f2.num;
84          result.den := f1.den * f2.den;
85          Reduire(result);
86          return result;
87      end "+";
88
89      -- Soustraction de deux fractions, retourne une fraction irreductible
90      function "-"(f1, f2 : T_Fraction) return T_Fraction is
91          result : T_Fraction;
92      begin
93          Verifie(f1);
94          Verifie(f2);
95          result.num := f1.num * f2.den - f1.den * f2.num;
96          result.den := f1.den * f2.den;
97          Reduire(result);
98          return result;
99      end "-";
100
101     -- Multiplication de deux frcations, retourne une fraction irreductible
102     function "*"(f1, f2 : T_Fraction) return T_Fraction is
103         result : T_Fraction;
104     begin
105         Verifie(f1);
106         Verifie(f2);
107         result.num := f1.num * f2.num;
108         result.den := f1.den * f2.den;
109         Reduire(result);
110         return result;
111     end "*";
112
113     -- Multiplication d'une fraction par un entier, retourne une fraction irreductible
114     function "*"(f : T_Fraction; n: integer) return T_Fraction is
115         result : T_Fraction;
116     begin
117         Verifie(f);
118         result.num := f.num * n;
119         result.den := f.den;
120         Reduire(result);
121         return result;
122     end "*";
123
124     -- Multiplication d'un entier par une fraction, retourne une fraction irreductible
125     function "*"(n: Integer; f : T_Fraction) return T_Fraction is
126     begin
127         return f*n;
128     end "*";
129
130     -- Division de deux fractions, retourne une fraction irreductible
131     function "/"(f1, f2 : T_Fraction) return T_Fraction is
132         result : T_Fraction;
133     begin
134         Verifie(f1);
135         Verifie(f2);
136         result.num := f1.num * f2.den;
137         result.den := f1.den * f2.num;
138         Reduire(result);
139         return result;
140     end "/";
141
142
143     -- Division d'une fraction par un entier, retourne une fraction irreductible
144     function "/"(f : T_Fraction; n : integer ) return T_Fraction is
145         f2: T_Fraction;
146     begin
```

```
147             f2 := (n,1);
148             return f/f2;
149         end "/";
150
151         -- Division d'un entier  par une fraction, retourne une fraction irreductible
152         function "/"(n : Integer; f : T_Fraction ) return T_Fraction is
153             f2: T_Fraction;
154         begin
155             f2 := (n,1);
156             return f2/f;
157         end "/";
158
159         -- Met la fraction à une puissance entière, retourne une fraction irreductible
160         function "**"(f : T_Fraction; n : integer ) return T_Fraction is
161             result : T_Fraction;
162         begin
163             Verifie(f);
164             result.num := f.num ** n;
165             result.den := f.den ** n;
166             Reduire(result);
167             return result;
168         end "**";
169
170         -- Calcule (recursivement) le Plus Grand Commun Diviseur de deux nombres
171         function PGCDof(n1, n2 : Integer) return Integer is
172             a, b : Integer;
173         begin
174             a := abs n1;
175             b := abs n2;
176             if b = 0 then
177                 return a; -- fin de recursivité
178             elsif  a > b then
179                 return PGCDof(b, a mod b);
180             else
181                 return PGCDof(a, b mod a);
182             end if;
183         end PGCDof;
184
185         -- Vérifie si la fraction n'a pas de division par 0
186         procedure Verifie(f : T_Fraction) is
187         begin
188             if f.den = 0 then
189                 raise DIV_PAR_ZERO;
190             end if;
191         end Verifie;
192     end Gestion_Fractions;
193
194
195
196     -- Code de Reed Solomon
197     -- Programme de test du paquet "Gestion_Fractions"
198     -- Cedric Dos Reis - 03.12.2017
199     -- Algorithmique
200
201     with Text_IO;                       use Text_IO;
202     WITH Ada.Command_Line;                USE Ada.Command_Line;
203     with Ada.Numerics.Float_Random;     use Ada.Numerics.Float_Random;
204     with Ada.Numerics.Generic_Elementary_Functions;
205     with Gestion_Fractions; use Gestion_Fractions;
206
207     Procedure Calcul_Fractions is
208         f1, f2 :T_Fraction;
209         n1 : Integer := Integer'Value(Argument(1));
210         n2 : Integer;
211         chaine : String := Argument(2);
212         chaine2 : string := Argument(3);
213     begin
214         case Argument_Count is
215           -- 3 arguments -> PGCD
216             when 3 =>
217                 n2 := PGCDof(Integer'Value(Argument(1)), Integer'Value(Argument(2)));
218                 put(Integer'Image(n2));
219           -- 4 arguments -> un entier, une fraction et un operateur
```

```
          when 4 =>
        -- L'entier est avant la fraction
          case chaine(1) is
              when '/' =>
                  f2 := (Integer'Value(Argument(3)), Integer'Value(Argument(4)));
                  Put(n1 / f2);
              when 'x' =>
                  f2 := (Integer'Value(Argument(3)), Integer'Value(Argument(4)));
                  Put(n1 * f2);
              when others =>
                  null;
          end case;
        -- La fraction est avant l'entier
          case chaine2(1) is
              when '/' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  Put(f1 / Integer'Value(Argument(4)));
              when 'x' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  Put(f1 * Integer'Value(Argument(4)));
              when 'p' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  Put(f1 ** Integer'Value(Argument(4)) );
              when others =>
                  null;
          end case;
        -- 5 arguments -> 2 fractions et 1 operateur
          when 5 =>
          case chaine2(1)is
              when '+' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  f2 := (Integer'Value(Argument(4)), Integer'Value(Argument(5)));
                  Put(f1 + f2);
              when '-' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  f2 := (Integer'Value(Argument(4)), Integer'Value(Argument(5)));
                  Put(f1 - f2);
              when '/' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  f2 := (Integer'Value(Argument(4)), Integer'Value(Argument(5)));
                  Put(f1 / f2);
              when 'x' =>
                  f1 := (Integer'Value(Argument(1)), Integer'Value(Argument(2)));
                  f2 := (Integer'Value(Argument(4)), Integer'Value(Argument(5)));
                  Put(f1 * f2);
              when others =>
                  null;
          end case;
              when others =>
                  null;
      end case;
  exception
      -- divison par 0
      when DIV_PAR_ZERO => Put_line("Division par 0.");
  end Calcul_Fractions;


  -- Code de Reed Solomon
  -- Package : Gestion Polynomes
  -- Cedric Dos Reis - 03.12.2017
  -- Algorithmique

  with Gestion_Fractions; use Gestion_Fractions;

  package Gestion_Polynomes is
      subtype T_Degre is Natural range 0..10000;
      type T_Coeff is array (T_Degre range<>) of T_Fraction;
      type T_Polynome(Degre : T_Degre := 0) is record
          Coeff : T_Coeff(0..Degre);
      end record;

      procedure Get(p : out T_Polynome);
      procedure Put(p : in  T_Polynome);
```

```
293
294         function "+"(p1, p2 : T_Polynome) return T_Polynome;
295         function "-"(p1, p2 : T_Polynome) return T_Polynome;
296         function "*"(p1, p2 : T_Polynome) return T_Polynome;
297         function "*"(p: T_Polynome; f : T_Fraction) return T_Polynome;
298         function "*"(f : T_Fraction; p: T_Polynome) return T_Polynome;
299         function "/"(dividende, diviseur : T_Polynome) return T_Polynome;
300         function Reste(dividende, diviseur : T_Polynome) return T_Polynome;
301         function Eval(p : T_Polynome; f : T_Fraction) return T_Fraction;
302         function Alloc_Polyn(p : T_Polynome; n : T_Degre) return T_Polynome;
303         function Reduit_Polyn(p : T_polynome) return T_Polynome;
304
305     end Gestion_Polynomes;
306
307
308
309     -- Code de Reed Solomon
310     -- Package body : Gestion Polynomes
311     -- Cedric Dos Reis - 03.12.2017
312     -- Algorithmique
313
314     with Text_IO; use Text_IO;
315     with Ada.Integer_Text_IO;          use Ada.Integer_Text_IO;
316     with Gestion_Fractions; use Gestion_Fractions;
317
318     package body Gestion_Polynomes is
319         ---------- PROCEDURES ----------
320         procedure Get(p : out T_Polynome) is
321         begin
322             for i in 0..p.Degre loop
323              Get(p.Coeff(i).Num);
324              Get(p.Coeff(i).Den);
325           end loop;
326         end Get;
327
328         -- affiche un polynome
329         procedure Put(p : in T_Polynome) is
330         begin
331             for I in 0..p.Degre loop
332                 Put(p.Coeff(i));
333             end loop;
334         end Put;
335
336         ---------- FONCTIONS ----------
337         -- Addition de deux polynomes
338         function "+"(p1, p2 : T_Polynome) return T_Polynome is
339             result : T_Polynome(Integer'Max(p1.Degre, p2.Degre));
340         begin
341             if p1.Degre < p2.Degre then
342                 return p2+p1;
343             elsif p1.Degre > p2.Degre then
344                 -- Additionne les coefficient de meme degre
345                 for i in 0..p2.Degre loop
346                     result.Coeff(i) := p1.Coeff(i) + p2.Coeff(i);
347                 end loop;
348                 -- Ajoute le reste du polynome au resultat
349                 for j in p2.Degre+1..p1.Degre loop
350                     result.Coeff(j) := p1.Coeff(j);
351                 end loop;
352             else
353                 for i in 0..p1.Degre loop
354                     result.Coeff(i) := p1.Coeff(i) + p2.Coeff(i);
355                 end loop;
356             end if;
357             return Reduit_Polyn(result);
358         end "+";
359
360         -- Soustraction de deux polynomes
361         function "-"(p1, p2 : T_Polynome) return T_Polynome is
362             result : T_Polynome(Integer'Max(p1.Degre, p2.Degre));
363         begin
364             -- le polynome p1 doit etre avoir un degre egale ou supérieur au degré de p2
365             if p1.Degre < p2.Degre then
```

```
366             return p2-p1;
367         elsif p1.Degre > p2.Degre then
368             -- soustrait les coefficient dont le degre est identique
369             for I in 0..p2.Degre loop
370                 result.Coeff(I) := p1.Coeff(I) - p2.Coeff(I);
371             end loop;
372 
373             -- Ajoute au polynome les coefficients qui n'ont pas de pair
374             for J in p2.Degre+1..p1.Degre loop
375                 result.Coeff(J) := p1.Coeff(J);
376             end loop;
377         else
378             -- polynomes de meme degré
379             for i in 0..p1.Degre loop
380                 result.Coeff(i) := p1.Coeff(i) - p2.Coeff(i);
381             end loop;
382         end if;
383         return Reduit_Polyn(result);
384     end "-";
385 
386     -- Multiplication de deux polynomes
387     function "*"(p1, p2 : T_Polynome) return T_Polynome is
388         produit : T_Polynome(p1.Degre+p2.Degre);
389     begin
390         produit.Coeff := (others => (0,1));
391         for I in 0..p1.Degre loop
392             for J in 0..p2.Degre loop
393                 produit.Coeff(I+J) := produit.Coeff(I+J) +  (p1.Coeff(I) *
                    p2.Coeff(J));
394             end loop;
395         end loop;
396         return produit;
397     end "*";
398 
399     -- Multiplication d'un polynome par une fraction
400     function "*"(p: T_Polynome; f : T_Fraction) return T_Polynome is
401         produit : T_Polynome(p.Degre);
402     begin
403         for i in 0..p.Degre loop
404             produit.Coeff(i) := p.Coeff(i) * f;
405         end loop;
406         return Reduit_Polyn(produit);
407     end "*";
408 
409     -- Multiplication d'une fraction par un polynome
410     function "*"(f : T_Fraction; p: T_Polynome) return T_Polynome is
411     begin
412         return p*f;
413     end "*";
414 
415     -- Division de deux polynomes, retourne le quotient
416     function "/"(dividende, diviseur : T_Polynome) return T_Polynome is
417         quotient : T_Polynome(abs (dividende.Degre - diviseur.Degre));
418         newDividende : T_Polynome;
419     begin
420         -- Impossible de diviser le dividende si le diviseur à un degré plus grand
421         if dividende.Degre >= diviseur.Degre then
422             -- divise le coeff de degre sup du dividende par le coeff de degre
                superieur du diviseur
423             quotient.Coeff(dividende.Degre - diviseur.Degre) :=
                dividende.Coeff(dividende.Degre) / diviseur.Coeff(diviseur.Degre);
424             -- Calcul le nouveau dividende au format reduit
425             newDividende := Reduit_Polyn(dividende - (diviseur * quotient));
426             -- recomence la division avec le nouveau dividende
427             return quotient + (newDividende / diviseur); -- recursif
428         else
429             return Reduit_Polyn(quotient);-- retourne un polynome dont tous les
                coefficient sont à 0/1
430         end if;
431     end "/";
432 
433     -- Calcule le reste de la division de deux polynomes
434     function Reste(dividende, diviseur : T_Polynome) return T_Polynome  is
```

```
435         reste, quotient: T_Polynome;
436     begin
437         -- calcul le quotient de la division de deux polynomes
438         quotient := dividende / diviseur;
439         reste := Reduit_Polyn(dividende - (quotient * diviseur));
440         return reste;
441     end Reste;
442
443     -- Evalue un polynome sur une fraction
444     function Eval(p : T_Polynome; f : T_Fraction) return T_Fraction is
445         result : T_Fraction;
446     begin
447         result := p.Coeff(0);
448         -- la boucle commence à 1 car le premier coéfficient de degré 0 n'est pas
            evalué
449         for i in 1..p.Degre loop
450             result := result + (p.Coeff(i) * (f**i));
451         end loop;
452         return result;
453     end Eval;
454
455     -- Retourne le polynome de degre n dont les coeff sont tous à 0/1
456     function Alloc_Polyn(p : T_Polynome; n :  T_Degre) return T_Polynome is
457         result : T_Polynome(n);
458     begin
459         return result;
460     end Alloc_Polyn;
461
462     -- Reduit le polynome jusqu'au premier coefficient différent de 0
463     -- EXEMPLE : 0x³+2x²+0x+1 -> 2x²+0x+1
464     function Reduit_Polyn(p : T_polynome) return  T_Polynome is
465         reduit : T_polynome(Integer'Max(0, p.Degre-1));
466     begin
467         if p.Degre /= 0 and then p.Coeff(p.Degre).num = 0 then
468             -- Transfere les coefficient du polynome dans un nouveau polynome de
                degre-1
469             for I in 0..reduit.Degre loop
470                 reduit.Coeff(I) := p.Coeff(I);
471             end loop;
472             if reduit.Degre = 0 then
473                 return reduit; -- ce polynome est reduit au max
474             else
475                 return Reduit_Polyn(reduit);
476             end if;
477         else
478             return p; --le polynome reçu n'a pas besoin d'etre reduit
479         end if;
480     end Reduit_Polyn;
481 end Gestion_Polynomes;
482
483
484
485 -- Code de Reed Solomon
486 -- Programme de test du paquet "Gestion_Polynomes"
487 -- Cedric Dos Reis - 05.12.2017
488 -- Algorithmique
489
490 with Text_IO;                       use Text_IO;
491 WITH Ada.Command_Line;                USE Ada.Command_Line;
492 with Ada.Numerics.Float_Random;     use Ada.Numerics.Float_Random;
493 with Ada.Numerics.Generic_Elementary_Functions;
494 with Gestion_Fractions; use Gestion_Fractions;
495 with Gestion_Polynomes; use Gestion_Polynomes;
496
497 procedure Calcul_Polynomes is
498     p0 : T_Polynome(0);
499     p1, p2 : T_Polynome(1);
500     p3 : T_Polynome(3);
501     f0,f1, f2, f3, f4, f5, f6, f7 : T_Fraction;
502 begin
503     f0 := (0,1);
504     f1 := (1,1);
505     f2 := (2,1);
```

```
506         f3 := (3,1);
507         f4 := (4,1);
508         f5 := (5,1);
509         f6 := (6,1);
510         f7 := (7,1);
511
512         p0.Coeff(0) := f1;
513
514         p1.Coeff(0) := f1;
515         p1.Coeff(1) := f2;
516
517         p2.Coeff(0) := f3;
518         p2.Coeff(1) := f2;
519
520         p3.Coeff(0) := f1;
521         p3.Coeff(1) := f3;
522         p3.Coeff(2) := f0;
523         p3.Coeff(3) := f4;
524
525         Put(p1); put(" +"); Put(p3); put(" ="); Put(p3+p1); New_Line; New_Line;
526         Put(p2); put(" -"); Put(p2); put(" ="); Put(p2-p2); New_Line; New_Line;
527         Put(p3); put(" *"); Put(p2); put(" ="); Put(p3*p2); New_Line; New_Line;
528         Put(p2); put(" *"); Put(f3); put(" ="); Put(p2*f3); New_Line; New_Line;
529         Put("Evalue "); Put(p3); Put(" à"); put(f2); Put(" :"); Put(Eval(p3, f2));
            New_Line; New_Line;
530         Put("Division de "); put(p3); put(" par "); put(p1); put(" = ");
            put(p3/p1);New_Line; New_Line;
531         Put("Reste de "); put(p3); put(" diviser par "); put(p1); put(" = ");
            put(Reste(p3,p1));New_Line; New_Line;
532     exception
533         when Div_Par_Zero => Put("Division par zero !");
534     end Calcul_Polynomes;
535
536     -- Code de Reed Solomon
537     -- Cedric Dos Reis - 11.12.2017
538     -- Algorithmique
539
540     with Text_IO;                       use Text_IO;
541     WITH Ada.Command_Line;                USE Ada.Command_Line;
542     with Ada.Numerics.Float_Random;     use Ada.Numerics.Float_Random;
543     with Ada.Numerics.Generic_Elementary_Functions;
544     with Gestion_Fractions; use Gestion_Fractions;
545     with Gestion_Polynomes; use Gestion_Polynomes;
546
547     Procedure Reed_Solomon is
548
549         data, total, parity : Integer := 0;
550
551         subtype T_Octet is Natural range 0..255;
552         type T_Point is record
553             x : Integer := 0;
554             y : T_Octet := 0;
555         end record;
556         type T_Points is array(Integer range<>) of T_Point ;
557
558         function Produit return T_Polynome is
559             polynome : T_Polynome;
560         begin
561             return polynome;
562         end Produit;
563
564         function Interpolate(points : T_Points) return T_Polynome is
565             polynome : T_Polynome;
566         begin
567             return polynome;
568         end Interpolate;
569     begin
570         -- REMARQUE : énoncé pas très compréhensible
571
572         --polynome := Interpolate(points);
573         --Put(polynome);
574
575     end Reed_Solomon;
```