```
> (* Mantej Sokhi - MATH 801 - Course Project *)
```

Part I: Chinese Remainder

```
> restart:
> CRT := proc(var_A::list, var_B::list, t, q::prime)

  local U:
  local M:
  local g:
  local v;
  local temp:

  U := var_A:
  M := var_B:

  if nops(U) <> nops(M) then
      error "FAIL":
  elif nops(U) = 1 then
      return U[1], M[1]:
  fi:

  g := Gcdex(M[1], M[2], t, 'a', 'b') mod q:
  v := U[1]*b*M[2] + U[2]*a*M[1]:
  temp := M[1]*M[2] mod q:
  v := expand(Rem(v, temp, t) mod q):

  U := U[3..-1]:
  U := [op(U), v]:
  M := [op(M), Expand(temp) mod q]:
  M := M[3..-1]:

  return CRT(U, M, t, q):
  end proc:
> U := [t^2, t^2 + t + 1, t^3]:
  M := [t^3 + t + 1, t^3 + t^2 + 1, t^4 + t + 1]:

  printf("\nINPUTS:\n\nU -> %a.\nM -> %a.\n", U, M):

  cong_U, prod_M := CRT(U, M, t, 2):

  printf("\nOUTPUTS:\n\nU -> %a.\nM -> %a.\n", cong_U, prod_M):

INPUTS:
```

```
U -> [t^2, t^2+t+1, t^3].
M -> [t^3+t+1, t^3+t^2+1, t^4+t+1].

OUTPUTS:

U -> t^9+t^8+t^7+t^5+t^4+t^3+1.
M -> t^10+t^9+t^8+t^6+t^5+t^4+1.
```

Part II: Modular Algorithm

```
> gcd_Quo := proc(a1, a2, p, q)

  local a, g:
  a := RootOf(p) mod q:
  g := Gcd(subs(t = a, a1), subs(t = a, a2)) mod q:

  if not type(a, integer) then
      g := subs(a = t, g):
  fi:

  return g:
  end proc:
```

```
> algo_Imp := proc(pol_A, pol_B, prime_Num)

  local a := pol_A:
  local b := pol_B:
  local q := prime_Num:
  local p := t:
  local k := 1:
  local v := 1:
  local j := 1:
  local bound:
  local sigma:
  local g_Comp:
  local min_Deg:

  local G:
  local V:
  local U := []:
  local M := []:

  local bad_Prime := []:
  local good_Prime := []:
  local new_BP := []:
```

```
local new_GP := []:
local unlucky_Prime := []:
local gcd_List := []:
local gcd_Deg := []:

sigma := Gcdex(lcoeff(a, x), lcoeff(b, x), t) mod q:
bound := min(degree(a, t), degree(b, t)) + degree(sigma, t):

(* Computation for potential min. degree of GCD ~ 8 is chosen at
random *)

while nops(good_Prime) <> 8 do

    if Divide(lcoeff(a, x), p) mod q then
        bad_Prime := [op(bad_Prime), p]:
        p := Nextprime(p, t) mod q:
        next:
    fi:

    good_Prime := [op(good_Prime), p]:
    g_Comp := gcd_Quo(a, b, good_Prime[k], q):
    gcd_List := [op(gcd_List), g_Comp]:
    gcd_Deg := [op(gcd_Deg), degree(g_Comp, x)]:
    k := k + 1:
    p := Nextprime(p, t) mod q;

od:

min_Deg := min(gcd_Deg):

while degree(v, t) <= bound do

    if Divide(lcoeff(a, x), p) mod q then
        new_BP := [op(new_BP), p]:
        p := Nextprime(p, t) mod q:
        next:
    fi:

    if Gcd(p, v) mod q = 1 then
        v := v*p:
        new_GP := [op(new_GP), p]:
        g_Comp := gcd_Quo(a, b, new_GP[j], q):

        if degree(g_Comp, x) > min_Deg then
            p := Nextprime(p, t) mod q;
            next;
        fi:
```

```
            g_Comp := Rem(sigma*g_Comp, p, t) mod q:
            U := [op(U), g_Comp]:
            M := [op(M), new_GP[j]]:
            j := j + 1:
        fi:

        p := Nextprime(p, t) mod q:

    od:

    G, V := CRT(U, M, t, q);
    G := Primpart(G, x) mod q:

    if Divide(a, G) mod q and Divide(b, G) mod q then
        return G:
    else
        error "FAIL":
    fi:

    end proc:
```

```
> g1 := (t^3 - t)*x^5 - t^11*x^3 + t^7*x + t^9 + 1:
  g2 := x^6 + t^6:
  a_Bar_1 := t*x^5 - t^6*x^2 + 1:
  a_Bar_2 := x^3 + t*x^2 + t^2 + 1:
  b_Bar_1 := t*x^4 + x^2 + t^7:
  b_Bar_2 := x^3 + t^2:
  q1 := 3:
  q2 := 2:

  a1 := Expand(g1 * a_Bar_1) mod q1:
  b1 := Expand(g1 * b_Bar_1) mod q1:
  a2 := Expand(g2 * a_Bar_2) mod q2:
  b2 := Expand(g2 * b_Bar_2) mod q2:

  printf("\nINPUTS:\n\na1 -> %a.\nb1 -> %a.\nq1 -> %a.\n", a1, b1,
  q1):

  comp_G := algo_Imp(a1, b1, q1):

  printf("\nOUTPUTS:\n\nG -> %a.\n", comp_G):
```

```
INPUTS:

a1 -> t^17*x^5+2*t^12*x^8+2*t^15*x^2+2*t^13*x^3+2*t^9*x^7+t^10*x^5+2*
t^11*x^3+t^8*x^6+t^7*x^7+t^4*x^10+2*t^2*x^10+t^9+t^7*x+2*t^6*x^2+t^3*
x^5+1.
```

```
b1 -> 2*t^18*x^3+2*t^12*x^7+t^16+2*t^11*x^5+t^14*x+t^10*x^5+t^10*x^4+
t^4*x^9+t^9*x^2+2*t^2*x^9+t^7*x^3+t^3*x^7+2*t*x^7+t^7+t*x^4+x^2.
q1 -> 3.

OUTPUTS:

G -> 2*t^11*x^3+t^9+t^7*x+t^3*x^5+2*t*x^5+1.
```

```
> printf("\nINPUTS:\n\na2 -> %a.\nb2 -> %a.\nq2 -> %a.\n", a2, b2,
  q2):

  comp_G := algo_Imp(a2, b2, q2):

  printf("\nOUTPUTS:\n\nG -> %a.\n", comp_G):
```

```
INPUTS:

a2 -> t^7*x^2+t^6*x^3+t*x^8+x^9+t^8+t^2*x^6+t^6+x^6.
b2 -> t^6*x^3+x^9+t^8+t^2*x^6.
q2 -> 2.

OUTPUTS:

G -> t^6+x^6.
```