```
--> load ( sym ) $
    load ( "sym/compile" ) $
    load ( gcdex ) $

--> /*==========       subroutine 1 ==========================*/

--> /*
    以下は逆元を求めるサブルーチンプログラム
    使い方

    A:    逆元を求めたい元
    Const:  代数体を規定しているリスト

    例えば    Const:[ [v,gv[0]],[a[1],B[1]]] など
    gv[0]:  最小多項式    変数v
    B[1]:   添加される数a[1]が満たす冪根方程式

    具体的には    Const:[[v,v^8-v^6+v^4-v^2+1],[a[1],a[1]^2-5/4]]$

    */

--> Inverse ( A , Const ) : = block (

      [ Nconst , vn , Tn , rwn , rpn ] ,

    Nconst : length ( Const ) ,
    vn : Nconst ,
    varL : [ ] ,
    maxpL : [ ] ,

    Tn : 1 ,
    for i : 1 thru Nconst do (

      con : Const [ i ] ,
      y [ i ] : con [ 1 ] , varL : endcons ( y [ i ] , varL ) ,
      D [ i ] : con [ 2 ] ,
      pw [ i ] : hipow ( D [ i ] , y [ i ] ) , maxpL : endcons ( pw [ i ] , maxpL ) , Tn : Tn * pw [ i ]

      ) ,
      /*    基底の次数の組み合わせ



    Tn:    pw[1]*pw[2]*pw[3]   2*3*2=12
```

pwL:　基底次数の組み合わせのリスト

[[0,0,0],[0,0,1],[0,1,0],[0,1,1],[0,2,0],[0,2,1],
[1,0,0],[1,0,1],[1,1,0],[1,1,1],[1,2,0],[1,2,1]]

rwn:　繰り返し同じ数を書く数(re-wright number)
rpn:　(repetition number)
例えば上の例ではリスト[x1,x2,x3]のx3の例でいうとx2だけをみると
[0,0,1,1,2,2,0,0,1,1,2,2]となっているが、この
[0,0,1,1,2,2]の繰り返しが2回繰り返されている。
この数を指定する数

Tn		pw[i]		rwn	rpr
12	÷	2	=	6	1=Tn/(pw[1]*rwn)
6	÷	3	=	2	2=Tn/(pw[2]*rwn)
2	÷	2	=	1	6=Tn/(pw[3]*rwn)

という関係式があるので下記のプログラムとなる

```
  */

pwL : [ ] ,
for i : 1 thru Tn do ( pwL : endcons ( [ ] , pwL ) ) ,

rwn : Tn ,
for i : 1 thru Nconst do (

  rwn : rwn / pw [ i ] ,
  rpn : Tn / ( pw [ i ] * rwn ) ,
  n : 1 ,

for j : 1 thru rpn do (

for k : 1 thru pw [ i ] do (

for m : 1 thru rwn do (
       pwL [ n ] : endcons ( k - 1 , pwL [ n ] ) , n : n + 1
       )
     )
     )
     ) ,
baseL : [ ] ,

for i : 1 thru Tn do (
  b : 1 ,
```

```
  for j : 1 thru vn do ( b : b * varL [ j ] ^ pwL [ i ] [ j ] ) ,
  baseL : endcons ( b , baseL )

) ,
coefL : makelist ( c [ i ] , i , 0 , Tn - 1 ) ,
F : sum ( coefL [ i ] * baseL [ i ] , i , 1 , Tn ) ,
W : F * A ,

for i : 1 thru Nconst do (
W : remainder ( W , Const [ i ] [ 2 ] , varL [ i ] )
) ,
W : expand ( W ) ,
CeL : [ ] ,
for i : 1 thru Tn do (
  ce : W ,
  for j : 1 thru vn do ( ce : coeff ( ce , varL [ j ] , pwL [ i ] [ j ] ) ) ,

  CeL : endcons ( ce , CeL )

  ) ,
  CeL [ 1 ] : CeL [ 1 ] - 1 ,
SLV : solve ( CeL , coefL ) ,
IA : sum ( coefL [ i ] * baseL [ i ] , i , 1 , Tn ) ,
IA : ev ( IA , SLV )
) $

--> /*==========       subroutine 2 ==========================*/

--> /*
以下は、代数体の次数を低減させるサブルーチンプログラム

Const:  代数体を規定しているリスト

例えば　Const:[ [v,gv],[a[1],B[1]]] など
gv[i]:  現在の体で使用されている v　の最小多項式
B[1]: 添加される数a[1]が満たす冪根方程式

具体的には　Const:[[v,v^8-v^6+v^4-v^2+1],[a[1],a[1]^2-5/4]]$



   */

--> Reduce ( A , Const ) : = block (

crn : length ( Const ) ,
```

```
for i : 1 thru crn do ( A : remainder ( A , Const [ i ] [ 2 ] , Const [ i ] [ 1 ] ) ) ,

A

) $
--> /*==========        main program ==========================*/
--> fx : x ^ 3 + 3 * x + 1 ;
--> N : 3 ; NN : N ! ;
--> rtL : [ α , β , γ ] ;
    fx1 : divide ( fx , x - α , x ) ;
    fx2 : divide ( fx1 [ 1 ] , x - β , x ) ;
    fx3 : divide ( fx2 [ 1 ] , x - γ , x ) ;
--> /*本当は eiL:makelist(e[i],i,1,Nfpwr);  としたい

       のだが e[i]のようなサフィックスだと根[α,β,γ]の対称式から方程式の
    係数に変換すると他の関数  " tcontract(...),  elem(...)  "
    の出力がe1,e2,e3を使っておりと一致しないので、やめにした       */
--> /*
    3次方程式 fx=0の根を ( α, β, γ  ) とするとき
    (α,β,γ)の全ての置換の組をリスト化する命令

    permutations([α,β,γ])

    */
--> /*  v[i]の計算の時に必要*/
--> cL : [ 1 , 2 , 3 ] ; zL : [ u , w , y ] ;
    V ( cL , rtL ) : = sum ( cL [ i ] * rtL [ i ] , i , 1 , N ) ;
--> V ( cL , rtL ) ;
--> /* α,β,γの計算の時に必要*/
--> prtL : listify ( permutations ( rtL ) ) ;
--> viL : makelist ( v [ i ] , i , 1 , NN ) ;
--> pVrtL : makelist ( V ( cL , prtL [ i ] ) , i , 1 , NN ) ;
--> virtL : map ( lambda ( [ x , y ] , x = y ) , viL , pVrtL ) ;
--> awL : [ 1 , 0 , 0 ] ; bwL : [ 0 , 1 , 0 ] ; cwL : [ 0 , 0 , 1 ] ;
```

```
--> pawL : makelist ( V ( awL , prtL [ i ] ) , i , 1 , NN ) ;
    pbwL : makelist ( V ( bwL , prtL [ i ] ) , i , 1 , NN ) ;
    pcwL : makelist ( V ( cwL , prtL [ i ] ) , i , 1 , NN ) ;

--> vxL : makelist ( x - v [ i ] , i , 1 , NN ) ;

--> awxL : makelist ( pawL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;
    bwxL : makelist ( pbwL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;
    cwxL : makelist ( pcwL [ i ] / ( x - v [ i ] ) , i , 1 , NN ) ;

--> Vx : apply ( "*" , vxL ) ;

--> /*   α, β, γ  の計算　の準備　*/

--> Pax : 0 $
    for i : 1 thru NN do ( Pax : Pax + Vx * awxL [ i ] ) $
    Pax ;
    Pbx : 0 $
    for i : 1 thru NN do ( Pbx : Pbx + Vx * bwxL [ i ] ) $
    Pbx $
    Pcx : 0 $
    for i : 1 thru NN do ( Pcx : Pcx + Vx * cwxL [ i ] ) $
    Pcx $

--> Vrt : ev ( Vx , virtL ) ; Vrt : expand ( Vrt ) $

--> Part : ev ( Pax , virtL ) ; Pbrt : ev ( Pbx , virtL ) $ Pcrt : ev ( Pcx , virtL ) $

--> Vrt1 : remainder ( Vrt , fx3 [ 2 ] , γ ) ; Vrt2 : remainder ( Vrt1 , fx2 [ 2 ] , β ) ; Vx : remainder (
    Vrt2 , fx1 [ 2 ] , α ) ;

--> /* P(x)の計算　[α,β,γ] の計算の為の多項式　Pax,Pbx,Pcx を求める　*/

    Part1 : remainder ( Part , fx3 [ 2 ] , γ ) ; Part2 : remainder ( Part1 , fx2 [ 2 ] , β ) ; Pax :
    remainder ( Part2 , fx1 [ 2 ] , α ) ;
    Pbrt1 : remainder ( Pbrt , fx3 [ 2 ] , γ ) $ Pbrt2 : remainder ( Pbrt1 , fx2 [ 2 ] , β ) $ Pbx :
    remainder ( Pbrt2 , fx1 [ 2 ] , α ) ;
    Pcrt1 : remainder ( Pcrt , fx3 [ 2 ] , γ ) $ Pcrt2 : remainder ( Pcrt1 , fx2 [ 2 ] , β ) $ Pcx :
    remainder ( Pcrt2 , fx1 [ 2 ] , α ) ;

--> /* Vxは最小多項式gx[0]の候補となる。 Vx が因数分解されたかどうかチェックする。
    因数分解されないとき　gx[0]=Vx　とする
    Vxが因数分解されたとき　　gx[0]=part(Vx,1)　として因数分解の第１因子をgx[0]とする
    以上の判断をして以下の議論をする。
         参考：式" f "の1番目(n=1)の因数を取り出す命令　part(f,n)

    hipow(Vx,x)
    と言うコマンドも面白い
```

```
    */
--> Vx : factor ( Vx ) ;
--> Vpw : hipow ( Vx , x ) $
    if Vpw = NN then gx [ 0 ] : Vx else gx [ 0 ] : part ( Vx , 1 ) $
    gx [ 0 ] ;
--> gv [ 0 ] : subst ( v , x , gx [ 0 ] ) ;
--> /*ここ重要　拘束条件のリスト作成

    Const:[]$  巡回拡大の際に現れる2項方程式B[i]とその根として添加される数a[i]を
                巡回拡大が現れる度に拘束条件リストに付け加えられるリスト
              Const:cons([a[i],B[i]],const); という命令で積み重ねる

    CurConst:[]$  高速条件Consにその時点での最小多項式の条件を加えるた高速条件
                  CurConst:cons([v,gv[i]],Const);   */

    /*現時点での単拡大体F(v)上での拘束条件作成は以下の通り*/ ;
--> Const : [ ] $
    CurConst : [ ] $
    CurConst : cons ( [ v , gv [ 0 ] ] , Const ) ;
--> /*

    Vxの微分した dV の逆元　IdV を計算する
    これらを使いα,β,γを計算する

    逆元を求める際には、ここではmaximaが持っているgcdexという命令を
    使ってみる

      */
--> dV : diff ( Vx , x ) ;
--> dVx : diff ( Vx , x ) ;  dVv : subst ( v , x , dVx ) ; IdV : Inverse ( dVv , CurConst ) ;
--  /* 逆元がvの多項式としても止まったので、α,β,γ の計算が出来るようになった */
>
    Pav : subst ( v , x , Pax ) ; Pbv : subst ( v , x , Pbx ) ; Pcv : subst ( v , x , Pcx ) ;
    wa1 : expand ( Pav * IdV ) ;  /*HTML用*/
    av : Reduce ( Pav * IdV , CurConst ) ; bv : Reduce ( Pbv * IdV , CurConst ) ; cv : Reduce ( Pcv *
    IdV , CurConst ) ;
    SoL : [ α = av , β = bv , γ = cv ] ;
```

```
--> VivL : expand ( ev ( virtL , SoL ) ) ;

--> check : [ ] $ GgrL : [ ] $
    for i : 1 thru NN do (
      z : remainder ( subst ( rhs ( VivL [ i ] ) , x , Vx ) , gv [ 0 ] ) ,
      check : endcons ( z , check ) , if z = 0 then GgrL : endcons ( i , GgrL )
                    ) $
    check ; GgrL ;

--> /* 第１ステップ

        6次式のg[0]を分解して3次のg[1]を計算する
        ガロア群はGal0= [1,2,3,4,5,6]からGal1=[1,4,5]に縮小
                Gal0/Gail1=2    の計算である
    */

--> h0 : ( x - v [ 1 ] ) * ( x - v [ 4 ] ) * ( x - v [ 5 ] ) ; h1 : ( x - v [ 2 ] ) * ( x - v [ 3 ] ) * ( x - v [ 6 ] ) ;

    h0 : ev ( h0 , VivL ) $ h0 : expand ( h0 ) ;
    h1 : ev ( h1 , VivL ) $ h1 : expand ( h1 ) ;

    h0 : Reduce ( h0 , CurConst ) ;
    h1 : Reduce ( h1 , CurConst ) ;

--> t0 : ( h0 + h1 ) / 2 $ t0 : expand ( t0 ) ;
    t1 : ( h0 - h1 ) / 2 $  t1 : expand ( t1 ) ;

--> T1 : expand ( t1 ^ 2 ) ;
    T1 : Reduce ( T1 , CurConst ) ;

--> A [ 1 ] : T1 ; B [ 1 ] : a [ 1 ] ^ 2 - A [ 1 ] ;  t1 : a [ 1 ] $

--> h0 : expand ( t0 + t1 ) ;

--> gx [ 1 ] : h0 ; gv [ 1 ] : subst ( v , x , h0 ) ;

--> Const : cons ( [ a [ 1 ] , B [ 1 ] ] , Const ) ;
    CurConst : cons ( [ v , gv [ 1 ] ] , Const ) ;

--> /* 以下は念のためのチェック   */
    h1 : t0 - t1 ;
    h01 : expand ( h0 * h1 ) ; h01 : remainder ( h01 , B [ 1 ] , a [ 1 ] ) ;

--> /*  第2ステップ
        3次多項式のg[1]をさらに分解してゆき1次多項式のg[2]を求める
        ガロア群はGal0= [1,2,3,4,5,6]からGal1=[1,4,5]になり
            以下の計算は  Gal1=[1,4,5]から  Gal2=[1] への最後の
            計算である
```

```
     Gal1/Gal2=3 である
    */

--> Ω : ω ^ 2 + ω + 1 $
    Const : cons ( [ ω , Ω ] , Const ) ; CurConst : cons ( [ v , gv [ 1 ] ] , Const ) ;

--> h0 : ev ( x - v [ 1 ] , VivL ) ;
    h1 : ev ( x - v [ 4 ] , VivL ) ; h1 : Reduce ( h1 , CurConst ) $ h1 : expand ( h1 ) ;
    h2 : ev ( x - v [ 5 ] , VivL ) ; h2 : Reduce ( h2 , CurConst ) $ h2 : expand ( h2 ) ;

--> t0 : expand ( ( h0 + h1 + h2 ) / 3 ) $  t0 : Reduce ( t0 , CurConst ) $ t0 : expand ( t0 ) ;
    t1 : ( h0 + ω * h1 + ω ^ 2 * h2 ) / 3 $  t1 : Reduce ( t1 , CurConst ) $ t1 : expand ( t1 ) ;
    t2 : ( h0 + ω ^ 2 * h1 + ω * h2 ) / 3 $ t2 : Reduce ( t2 , CurConst ) $ t2 : expand ( t2 ) ;

--> /* T1=t1^3 を求める計算*/
    T1 : expand ( t1 ^ 3 ) $ T1 : Reduce ( T1 , CurConst ) $ T1 : expand ( T1 ) ;

--> /*  T1の逆数を求める計算
    IAと言う逆元多項式を仮定して係数を決定する方式　個の逆数を求めるやり方は重要 */

    IA : Inverse ( T1 , CurConst ) ;

--> CurConst ;

--> T12 : t1 * t2 $ T12 : expand ( T12 ) ;   T12 : Reduce ( T12 , CurConst ) $  T12 : expand ( T12 )
    ;

--> t2 : T12 * a [ 2 ] ^ 2 * IA ;  t2 : expand ( t2 ) ;   t2 : factor ( t2 ) ;

--> /* ここまでのまとめが以下の式*/

    A [ 2 ] : T1 ; B [ 2 ] : a [ 2 ] ^ 3 - A [ 2 ] ;  t1 : a [ 2 ] $
    Const : cons ( [ a [ 2 ] , B [ 2 ] ] , Const ) ;

--> t0 ; t1 ; t2 ;
    h0 : t0 + t1 + t2 $
    gx [ 2 ] : h0 ; gv [ 2 ] : subst ( v , x , h0 ) ;
    CurConst : cons ( [ v , gv [ 2 ] ] , Const ) ;

--> W : solve ( gx [ 2 ] , x ) ; W : expand ( W ) ;

--> /* 以下はg[1]が以下に分解されたかのチェック*/

--> h1 : t0 + t1 * ω ^ 2 + t2 * ω ; h2 : t0 + t1 * ω + t2 * ω ^ 2 ;
    h012 : expand ( h0 * h1 * h2 ) $
    h012 : remainder ( h012 , B [ 2 ] , a [ 2 ] ) $ h012 : remainder ( h012 , Ω , ω ) $
    h012 : remainder ( h012 , B [ 1 ] , a [ 1 ] ) ;


--> SoL ; Rt : subst ( v , x , W ) ;
```

```
--> /*
    以下はgx[2]=0で求まるvの値を、vの多項式で表現された[α,β,γ]に
    代入して、添加数a[1],a[2]で3根を表す計算
    */

--> SoL : ev ( SoL , Rt ) ;

--> AnsL : [ ] $
    for i : 1 thru 3 do (
        z : rhs ( SoL [ i ] ) ,
        z : Reduce ( z , Const ) ,
        AnsL : endcons ( z , AnsL )
    ) $
    AnsL ;

--> /*    以下の式が添加数a[1],a[2],ωで記述されたf(x)の3根の数式である    */

--> AnsL : map ( lambda ( [ x , y ] , x = y ) , rtL , AnsL ) ;

--> /*  実際に数値的に正しいか計算してみる。 maximaで数値で解を求める命令はallroots()である*/

--> root : [ ] $
    ωr : allroots ( Ω ) $ root : endcons ( ω = rhs ( ωr [ 1 ] ) , root ) ;
    a1r : allroots ( B [ 1 ] ) $ root : endcons ( a [ 1 ] = rhs ( a1r [ 1 ] ) , root ) ;
    b2 : ev ( B [ 2 ] , root ) $ b2r : allroots ( b2 ) $ root : endcons ( a [ 2 ] = rhs ( b2r [ 1 ] ) , root ) ;

--> FansL : ev ( AnsL , root ) $ FansL : expand ( FansL ) ;

--> allroots ( fx ) ;

--> /*FansLとallroota(fx)の実数値で一致している事が判ったので、AnsLの数式は正しい事が判った*/
```