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

--> /*==========       main program ==========================*/

--> fx : x ^ 2 + x + 1 ; N : 2 $ NN : 2 $

--> virtL : [ v [ 1 ] = α + 2 * β , v [ 2 ] = β + 2 * α ] ;

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

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

--> pawL : [ α , β ] ; pbwL : [ β , α ] ;

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

--> /*   α, β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 ;

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

--> Part : ev ( Pax , virtL ) ; Part : expand ( Part ) ;
    Pbrt : ev ( Pbx , virtL ) ; Pbrt : expand ( Pbrt ) ;

--> /*     準備終わり       */

--> EL : [ e1 = - 1 , e2 = 1 ] $

--> /*   V(x) の計算   */
    Vrt1 : tcontract ( Vrt , [ α , β ] ) ; Vrt2 : elem ( [ 2 ] , Vrt1 , [ α , β ] ) ; Vrt2 : expand ( Vrt2 ) ;
    Vx : ev ( Vrt2 , EL ) ;

--  /*   Pα(x) の計算   */
>   Part1 : tcontract ( Part , [ α , β ] ) ; Part2 : elem ( [ 2 ] , Part1 , [ α , β ] ) ; Part2 : expand (
    Part2 ) ;
    Pax : ev ( Part2 , EL ) ;

--  /*  Pβ(x)の計算   */
>   Pbrt1 : tcontract ( Pbrt , [ α , β ] ) ; Pbrt2 : elem ( [ 2 ] , Pbrt1 , [ α , β ] ) ; Pbrt2 : expand (
    Pbrt2 ) ;
    Pbx : ev ( Pbrt2 , EL ) ;

--> /* V(x) は既約多項式か？       vの最小多項式の定義*/
```

```
--> 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 ] ) ;

--> /*
    Vxの微分した dV の逆元　IdV を計算する
    これらを使い [α,β] を計算する
      */

--> dVx : diff ( Vx , x ) ; dV : subst ( v , x , dVx ) ;

--> /*      dVの逆元をIDとする*/
    ID : c1 * v + c0 $
    mx : dV * ID ; mx : expand ( mx ) ;

--> mx : remainder ( mx , gv [ 0 ] , v ) ;

--> ansc : solve ( [ 2 * c0 - 3 * c1 = 0 , - 6 * c1 + 3 * c0 = 1 ] , [ c1 , c0 ] ) ;

--> ID : ev ( ID , ansc ) ;

--> check : dV * ID ; check : expand ( check ) ; check : remainder ( check , gv [ 0 ] ) ;

--> /*
    上記check=1となったのでIdVがdVの逆元である事が
    確認された
    */

--> /*      逆元 ID がvの多項式として求まったので、α,β の計算が出来るようになった      */

--> Pav : subst ( v , x , Pax ) ; Pbv : subst ( v , x , Pbx ) ;
    SoL : [ ] $
    αv : remainder ( Pav * ID , gv [ 0 ] , v ) ;   SoL : endcons ( α = αv , SoL ) $
    βv : remainder ( Pbv * ID , gv [ 0 ] , v ) ;   SoL : endcons ( β = βv , SoL ) $
    SoL ;

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

--> /*ここからが本当の計算*/

--> h0 : ( x - v [ 1 ] ) $ h0 : ev ( h0 , VivL ) ;
    h1 : ( x - v [ 2 ] ) $ h1 : ev ( h1 , VivL ) ;
    t0 : ( h0 + h1 ) / 2 ; t1 : ( h0 - h1 ) / 2 ;

--> T1 : expand ( t1 ^ 2 ) ;
    T1 : remainder ( T1 , gv [ 0 ] , v ) ;

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

```
--> h0 : expand ( t0 + t1 ) ; h0 : remainder ( h0 , gv [ 0 ] , v ) $ h0 : expand ( h0 ) ;
    h1 : expand ( t0 - t1 ) ; h1 : remainder ( h1 , gv [ 0 ] , v ) $ h1 : expand ( h1 ) ;

--> gx [ 1 ] : h0 ; gv [ 1 ] : subst ( v , x , h0 ) ; vsoL : solve ( [ gv [ 1 ] = 0 ] , v ) ;

--> SoL : ev ( SoL , vsoL ) $ SoL : expand ( SoL ) ; SoL : remainder ( SoL , B [ 1 ] , a [ 1 ] ) ;
```
