```
--With Ada.Text_IO;
With Ada.Integer_Text_IO;
With Alices_Encryption_Numbers;
With Normal_Vector_I_Coefficients;
With Normal_Vector_J_Coefficients;
With Normal_Vector_K_Coefficients;
With Change_of_Origin_I_Coefficients;
With Change_of_Origin_J_Coefficients;
With Change_of_Origin_K_Coefficients;
With Change_of_Origin_II_Coefficients;
With Change_of_Origin_JJ_Coefficients;
With Change_of_Origin_KK_Coefficients;
WITH Text_IO,Basic_Num_IO,Calendar;
USE  Text_IO,Basic_Num_IO,Calendar;
Procedure General_Decryption_Program_Mark_0 IS
-------------------------------------------------------------------------
--| This is an annotated version of the decryption program.
--| Copyright © 2003 Austin O'Byrne.
--| Created 25/03/2018
--| Last modified:- November 2021
-------------------------------------------------------------------------
  MaxName : CONSTANT Positive := 80;
  SUBTYPE NameRange IS Positive RANGE 1 .. MaxName;
  InFileName : String(NameRange) :=(OTHERS => '#');
  OutFileName: String(NameRange) :=(OTHERS => '#');
  InNameLength: NameRange;
  OutNameLength: NameRange;
  InData: Ada.Text_IO.File_Type;
  OutData: Ada.Text_IO. File_Type;

  I               : Integer;
  X               : Integer;
  Epsilon_X       : Integer;
  Epsilon_Y       : Integer;
  Epsilon_Z       : Integer;
  Total           : Integer;
  Line_Number     : Integer;
  Count           : Integer;
  Counter         : Integer;
  Check           : Integer;
  AlphabetChar    : Character;
  View            : Character;

  Gain : Constant Integer:= 2;  --( 1 =< Gain =< 874 )

     ---*****************************************************---

                  -- KEY PAD STARTS HERE --

  SliceNum_1  : CONSTANT Integer := 0;  --Start point in array   -- 0, 450, 2
  StepNum_1   : CONSTANT Integer := 1000;--Upstream placemoves   --19
  RepeatsNum_1: CONSTANT Integer := 1;--Repeats of shift instruction --52
  -- scrambling device in "Load_n_Scramble_Encryption_Numbers" procedure
  -- (SliceNum_1 + stepNum_1*RepeatsNum_1) <= 1000.

  SliceNum_2  : CONSTANT Integer := 0;  --Start point in array  -- 0, 23, 43
```

```
  StepNum_2   : CONSTANT Integer := 1000;--Upstream placemoves
  RepeatsNum_2: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Normal_Vector_I_Coefficients" procedure --Array 11
  -- (SliceNum_2 + stepNum_2*RepeatsNum_2) <= 1000.

  SliceNum_3  : CONSTANT Integer := 0;  --Start point in array     -- 0, 17, 58
  StepNum_3   : CONSTANT Integer := 1000;--Upstream placemoves
  RepeatsNum_3: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Normal_Vector_J_Coefficients" procedure --Array
13
  -- (SliceNum_3 + stepNum_3*RepeatsNum_3) <= 1000.

  SliceNum_4  : CONSTANT Integer := 0;  --Start point in array     -- 0, 19, 52
  StepNum_4   : CONSTANT Integer := 1000;--Upstream placemoves
  RepeatsNum_4: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Normal_Vector_K_Coefficients" procedure --Array
15
  -- (SliceNum_4 + stepNum_4*RepeatsNum_4) <= 1000.

  SliceNum_5  : CONSTANT Integer := 0;  --Start point in array   -- 0, 30, 38
  StepNum_5   : CONSTANT Integer := 14250;--Upstream placemoves
  RepeatsNum_5: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Change_of_origin_Vector_I_Coefficients" procedure
--Array 17
  -- (SliceNum_5 + stepNum_5*RepeatsNum_5) <= 14250.

  SliceNum_6  : CONSTANT Integer := 0;  --Start point in array  -- 0, 45, 316
  StepNum_6   : CONSTANT Integer := 14250;--Upstream placemoves
  RepeatsNum_6: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Change_of_origin_Vector_J_Coefficients" procedure
--Array 19
  -- (SliceNum_6 + stepNum_6*RepeatsNum_6) <= 14250.

  SliceNum_7  : CONSTANT Integer := 0;  --Start point in array  --0, 28, 508
  StepNum_7   : CONSTANT Integer := 14250;--Upstream placemoves
  RepeatsNum_7: CONSTANT Integer := 1;--Repeats of shift instruction
  --scrambling device in "Load_n_Scramble_Change_of_origin_Vector_K_Coefficients" procedure
-- Array 21
  --(SliceNum_7 + stepNum_7*RepeatsNum_7) <= 14250.

  SliceNum_8  : CONSTANT Integer := 0;  --Start point in array  -- o, 75, 190
  StepNum_8   : CONSTANT Integer := 14250;--Upstream placemoves
  RepeatsNum_8: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Change_of_origin_Vector_II_Coefficients"
procedure
  -- (SliceNum_8 + stepNum_8*RepeatsNum_8) <= 14250.

  SliceNum_9  : CONSTANT Integer := 0;  --Start point in array  --0, 37, 385
  StepNum_9   : CONSTANT Integer := 14250;--Upstream placemoves
  RepeatsNum_9: CONSTANT Integer := 1;--Repeats of shift instruction
  -- scrambling device in "Load_n_Scramble_Change_of_origin_Vector_JJ_Coefficients"
procedure
  -- (SliceNum_9 + stepNum_9*RepeatsNum_9) <= 14250.

  SliceNum_10  : CONSTANT Integer :=  0;  --Start point in array  --0, 17, 9
  StepNum_10   : CONSTANT Integer := 14250;--Upstream placemoves
```

```
 RepeatsNum_10: CONSTANT Integer := 1;--Repeats of shift instruction
 -- scrambling device in "Load_n_Scramble_Change_of_origin_Vector_KK_Coefficients"
procedure
 -- (SliceNum_10 + stepNum_10*RepeatsNum_10) <= 14250.

                    -- KEY PAD ENDS HERE --

     ---*******************************************************---

 SUBTYPE Index_1 IS Positive RANGE 1 .. 1000;
 TYPE EncryptionNumsArray IS ARRAY(Index_1) OF Integer;
 Array_1: EncryptionNumsArray; -- Encryption numbers before scrambling

 SUBTYPE Index_2 IS Positive RANGE 1 .. 1000;
 TYPE EncryptionNumbersArray IS ARRAY(Index_2) OF Integer;
 Array_2: EncryptionNumbersArray; -- Encryption numbers after scrambling

 SUBTYPE IndeX_3 IS Positive RANGE 32 .. 126;
 TYPE NumbersArray IS ARRAY(Index_3) OF Integer;
 Number: NumbersArray;--these numbers provide Alice's many alphabets

 SUBTYPE Index_4 IS Positive RANGE 1 .. 1000;
 TYPE Normals_I_Array IS Array(Index_4) OF Integer;
 Array_4: Normals_I_Array; -- Normal vector I coeffeicients before scrambling

 SUBTYPE Index_5 IS Positive RANGE 1 .. 1000;
 TYPE NormalsAlso_I_Array IS Array(Index_5) OF Integer;
 Array_5: NormalsAlso_I_Array;  -- Normal vector I coefficients after scrambling

 SUBTYPE Index_6 IS Positive RANGE 1 .. 1000;
 TYPE Normals_J_Array IS Array(Index_6) OF Integer;
 Array_6: Normals_J_Array;  -- Normal vector J coefficients before scrambling

 SUBTYPE Index_7 IS Positive RANGE 1 .. 1000;
 TYPE NormalsAlso_J_Array IS Array(Index_7) OF Integer;
 Array_7: NormalsAlso_J_Array;  -- Normal vector J coefficients after scrambling

 SUBTYPE Index_8 IS Positive RANGE 1 .. 1000;
 TYPE Normals_K_Array IS Array(Index_8) OF Integer;
 Array_8: Normals_K_Array;  -- Normal vector K coefficients before scrambling

 SUBTYPE Index_9 IS Positive RANGE 1 .. 1000;
 TYPE NormalsAlso_K_Array IS Array(Index_9) OF Integer;
 Array_9: NormalsAlso_K_Array;  -- Normal vector K coefficients after scrambling

 SUBTYPE INDEX_10 IS Positive RANGE 1 .. 14250;
 TYPE UnitVectorCoeffArray_1 IS ARRAY (Index_10) OF Integer;
 Array_10: UnitVectorCoeffArray_1;  -- Change of origin vector I coefficients before scrambling

 SUBTYPE Index_11 IS Positive RANGE 1 .. 14250;
 TYPE UnitVectorCoeff_1 IS ARRAY(Index_11) OF Integer;
 Array_11: UnitVectorCoeff_1;  -- Change of origin vector I coefficients after scrambling

 SUBTYPE Index_12 IS Positive RANGE 1 .. 14250;
 TYPE UnitVectorCoeffArray_2 IS ARRAY (Index_12) OF Integer;
 Array_12: UnitVectorCoeffArray_2 ; -- Change of origin vector J coefficients before scrambling
```

```
SUBTYPE Index_13 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeff_2 IS ARRAY(Index_13) OF Integer;
Array_13: UnitVectorCoeff_2; -- Change of origin vector J coefficients after scrambling

SUBTYPE INDEX_14 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeffArray_3 IS ARRAY (Index_14) OF Integer;
Array_14: UnitVectorCoeffArray_3; -- Change of origin vector K coefficients Before scrambling

SUBTYPE Index_15 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeff_3 IS ARRAY(Index_15) OF Integer;
Array_15: UnitVectorCoeff_3; -- Change of origin vector K coefficients after scrambling

SUBTYPE Index_16 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeffArray_4 IS ARRAY (Index_16) OF Integer;
Array_16: UnitVectorCoeffArray_4; -- Change of origin vector II coefficients before scrambling

SUBTYPE Index_17 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeff_4 IS ARRAY(Index_17) OF Integer;
Array_17: UnitVectorCoeff_4;  -- Change of origin vector II coefficients after scrambling

SUBTYPE Index_18 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeffArray_5 IS ARRAY (Index_18) OF Integer;
Array_18: UnitVectorCoeffArray_5; -- Change of origin vector JJ coefficients before scrambling

SUBTYPE Index_19 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeff_5 IS ARRAY(Index_19) OF Integer;
Array_19: UnitVectorCoeff_5;  -- Change of origin vector JJ coefficients after scrambling

SUBTYPE Index_20 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeffArray_6 IS ARRAY (Index_20) OF Integer;
Array_20: UnitVectorCoeffArray_6; -- Change of origin vector KK coefficients before scrambling

SUBTYPE Index_21 IS Positive RANGE 1 .. 14250;
TYPE UnitVectorCoeff_6 IS ARRAY(Index_21) OF Integer;
Array_21: UnitVectorCoeff_6; -- Change of origin vector KK coefficients after scrambling

SUBTYPE Index_22 IS Positive RANGE 1 .. 3;
TYPE VeeZeroArray IS ARRAY(Index_22) OF Integer;
S: VeeZeroArray;-- briefly holds the current VeeZero

SUBTYPE Index_23 IS Positive RANGE 1 .. 3;
TYPE VeeZero_ZX_Array IS ARRAY(Index_23) OF Integer;
SS: VeeZero_ZX_Array;-- briefly holds the current VeeZero

SUBTYPE Index_24 IS Positive RANGE 1 .. 3;
TYPE VeeZero_XY_Array IS ARRAY(Index_24) OF Integer;
SSS: VeeZero_XY_Array;-- briefly holds the current VeeZero

SUBTYPE Index_25 IS Positive RANGE 1 .. 3;
TYPE CipherText_ItemsArray IS ARRAY(Index_25) OF Integer;
W: CipherText_ItemsArray;

 --holds the finished ciphertext ready for emailing
SUBTYPE Index_26 IS Positive RANGE 1 .. 3;  -- for LargeCrossProduct_ZY
TYPE LargeCrossProduct_1_Array IS ARRAY(Index_26) OF Integer;
```

```
  LargeCrossProduct_1: LargeCrossProduct_1_Array;-- computes Pn x V0_ZY

  SUBTYPE Index_27 IS Positive RANGE 1 .. 3;  -- for LargeCrossProduct_ZX
  TYPE LargeCrossProduct_2_Array IS ARRAY(Index_27) OF Integer;
  LargeCrossProduct_2: LargeCrossProduct_2_Array;-- computes Pn x V0_ZX

  SUBTYPE Index_28 IS Positive RANGE 1 .. 3;  -- for LargeCrossProduct_ZX
  TYPE LargeCrossProduct_3_Array IS ARRAY(Index_28) OF Integer;
  LargeCrossProduct_3: LargeCrossProduct_3_Array;-- computes Pn x V0_ZX

  SUBTYPE Index_29 IS Positive RANGE 1 .. 3;
  TYPE Position_VectorArray IS ARRAY(Index_29) OF Integer;
  Pn: Position_VectorArray;

  SUBTYPE Index_30 IS Positive RANGE 1 .. 3;
  TYPE DecryptedNumberArray IS ARRAY(Index_30) OF Integer;
  n: DecryptedNumberArray; -- brieflty holfds the currnet number in Alice's alpabet

  SUBTYPE Index_31 IS Positive RANGE 1 .. 100000;
  TYPE DecodedSmall_nsArray IS ARRAY(Index_31) OF Character;
  MessageText: DecodedSmall_nsArray; -- holds the messagetext

  SUBTYPE Index_32 IS Positive RANGE 1 .. 10;
  TYPE FigsTimeArray IS ARRAY (Index_32) OF Integer;
  Figs : FigsTimeArray;  -- copies the digits of Time_Ex_1

  SUBTYPE Index_33 IS Positive RANGE 1 .. 10;
  TYPE DatesTimeArray IS ARRAY (Index_33) OF Integer;
  Dates : DatesTimeArray;  -- copies the digits of Time_Ex_2

--       ---*****************************************************---

  PROCEDURE Load_n_Scramble_Encryption_Numbers IS
  -- Pre : The package Alice's_Encryption_Numbers is defined.
  -- Post: User's encryption domain is defined in part.

 BEGIN  -- Load_n_Scramble_Encryption_Numbers

  FOR I IN 1 .. 1000 LOOP
    Array_1(I):= Alices_Encryption_Numbers.ChangeNumber(Numin => I);
    Array_2(I):= Alices_Encryption_Numbers.ChangeNumber(Numin => I);
  END LOOP;
     X := SliceNum_1;
  FOR J IN 1 .. RepeatsNum_1 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_1 LOOP
    X:=X+1;
    Array_2(X) := Array_1(I);
  END LOOP;
  END LOOP;
  FOR Count IN 32 .. 126 LOOP -- this is the alpahabet domain
    Number(Count):= Array_2((Count) + Gain);
  END LOOP;
 END Load_n_Scramble_Encryption_Numbers;

            ----------------------------------------------------
```

```
  PROCEDURE Decipher (Item: IN Integer) IS
  -- pre: n is assigned a value
  -- post: Message Text has the character represented by n

  BEGIN -- Decipher
    I:= 32;
   WHILE Item /= Number(I) LOOP
    I:= I+1;
   END LOOP;
    AlphabetChar:= character 'Val(I);

  END Decipher;

          ----------------------------------------------------

  PROCEDURE Load_n_Scramble_Normal_Vector_I_Coefficients IS
  -- Pre: Normal_Vector_I_Coefficients_ Package is defined.
   -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_Normal_Vector_I_Coefficients

  FOR I IN 1 .. 1000 LOOP
    Array_4(I) := Normal_Vector_I_Coefficients.NumberExchange(Numin => I);
    Array_5(I) := Normal_Vector_I_Coefficients.NumberExchange(Numin => I);
  END LOOP;
     X := SliceNum_2;
  FOR J IN 1 .. RepeatsNum_2 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_2 LOOP
    X:=X+1;
    Array_5(X) := Array_4(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Normal_Vector_I_Coefficients;

          ----------------------------------------------------

  PROCEDURE Load_N_Scramble_Normal_Vector_J_Coefficients IS
  -- Pre : Package Normal_Vector_J_Coefficients is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_Normal_Vector_J_Coefficients

  FOR I IN 1 .. 1000 LOOP
    Array_6(I) := Normal_Vector_J_Coefficients.NumberExchange(Numin => I);
    Array_7(I) := Normal_Vector_J_Coefficients.NumberExchange(Numin => I);
  END LOOP;
     X := SliceNum_3;
  FOR J IN 1 .. RepeatsNum_3 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_3 LOOP
    X:=X+1;
    Array_7(X) := Array_6(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Normal_Vector_J_Coefficients;

          ----------------------------------------------------
```

```
  PROCEDURE Load_n_Scramble_Normal_Vector_K_Coefficients IS
  -- Pre : Package Normal_Vector_K_Coefficients is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_Normal_Vector_K_Coefficients

  FOR I IN 1 .. 1000 LOOP
    Array_8(I) := Normal_Vector_K_Coefficients.NumberExchange(Numin => I);
    Array_9(I) := Normal_Vector_K_Coefficients.NumberExchange(Numin => I);
   END LOOP;
     X := SliceNum_4;
  FOR J IN 1 .. RepeatsNum_4 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_4 LOOP
    X:=X+1;
    Array_9(X) := Array_8(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Normal_Vector_K_Coefficients;

          ---------------------------------------------------

  PROCEDURE Load_n_Scramble_Change_of_Origin_Vector_I_Coefficients IS
  -- Pre : The package " Change_Of_Origin _I_Coefficients" is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_I_Coefficients

  FOR I IN 1 .. 14250 LOOP
    Array_10(I):= Change_Of_Origin_I_Coefficients.GetNum(NumIn => I);
    Array_11(I):= Change_Of_Origin_I_Coefficients.GetNum(NumIn => I);
  END LOOP;
     X := SliceNum_5;
  FOR J IN 1 .. RepeatsNum_5 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_5 LOOP
    X:=X+1;
    Array_11(X) := Array_10(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Change_of_Origin_Vector_I_Coefficients;

            ------------------------------------------------
  PROCEDURE Load_n_Scramble_Change_of_Origin_Vector_J_Coefficients IS
  -- Pre : The package " Change_Of_Origin _J_Coefficients" is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_J_Coefficients

  FOR I IN 1 .. 14250 LOOP
    Array_12(I):= Change_Of_Origin_J_Coefficients.GetNum(NumIn => I);
    Array_13(I):= Change_Of_Origin_J_Coefficients.GetNum(NumIn => I);
  END LOOP;
     X := SliceNum_6;
  FOR J IN 1 .. RepeatsNum_6 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_6 LOOP
    X:=X+1;
```

```
    Array_13(X) := Array_12(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Change_of_Origin_Vector_J_Coefficients;

          ------------------------------------------------

  PROCEDURE Load_n_Scramble_Change_of_Origin_Vector_K_Coefficients IS
  -- Pre : The package " Change_Of_Origin _K_Coefficients" is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_K_Coefficients

  FOR I IN 1 .. 14250 LOOP
    Array_14(I):= Change_Of_Origin_K_Coefficients.GetNum(NumIn => I);
    Array_15(I):= Change_Of_Origin_K_Coefficients.GetNum(NumIn => I);
  END LOOP;
     X := SliceNum_7;
  FOR J IN 1 .. RepeatsNum_7 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_7 LOOP
    X:=X+1;
    Array_15(X) := Array_14(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Change_of_Origin_Vector_K_Coefficients;

          ------------------------------------------------

  PROCEDURE Load_n_Scramble_Change_of_Origin_Vector_II_Coefficients IS
  -- Pre : The package " Change_Of_Origin _II_Coefficients" is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_II_Coefficients

  FOR I IN 1 .. 14250 LOOP
    Array_16(I):= Change_Of_Origin_II_Coefficients.GetNum(NumIn => I);
    Array_17(I):= Change_Of_Origin_II_Coefficients.GetNum(NumIn => I);
  END LOOP;
     X := SliceNum_8;
  FOR J IN 1 .. RepeatsNum_8 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_8 LOOP
    X:=X+1;
    Array_17(X) := Array_16(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Change_of_Origin_Vector_II_Coefficients;

          ------------------------------------------------

  PROCEDURE Load_n_Scramble_Change_of_Origin_Vector_JJ_Coefficients IS
   --Pre : The package " Change_Of_Origin _JJ_Coefficients" is defined
   --Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_JJ_Coefficients

  FOR I IN 1 .. 14250 LOOP
```

```
    Array_18(I):= Change_Of_Origin_JJ_Coefficients.GetNum(NumIn => I);
    Array_19(I):= Change_Of_Origin_JJ_Coefficients.GetNum(NumIn => I);
  END LOOP;
     X := SliceNum_9;
  FOR J IN 1 .. RepeatsNum_9 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_9 LOOP
    X:=X+1;
    Array_19(X) := Array_18(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Change_of_Origin_Vector_JJ_Coefficients;

            ---------------------------------------------

  PROCEDURE Load_n_Scramble_Change_of_Origin_Vector_KK_Coefficients IS
  -- Pre : The package " Change_Of_Origin _KK_Coefficients" is defined
  -- Post: These coefficients are stored in arrays for recalling.

BEGIN  -- Load_n_Scramble_KK_Coefficients

  FOR I IN 1 .. 14250 LOOP
    Array_20(I):= Change_Of_Origin_KK_Coefficients.GetNum(NumIn => I);
    Array_21(I):= Change_Of_Origin_KK_Coefficients.GetNum(NumIn => I);
  END LOOP;
     X := SliceNum_10;
  FOR J IN 1 .. RepeatsNum_10 LOOP
  FOR I IN REVERSE X+1 .. X+StepNum_10 LOOP
    X:=X+1;
    Array_21(X) := Array_20(I);
  END LOOP;
  END LOOP;
END Load_n_Scramble_Change_of_Origin_Vector_KK_Coefficients;

            ------------------------------------------------

  PROCEDURE Time_Ex_1 IS
  -- Pre: Basic_Num_IO.ads is defined
  -- Post: Time data is available for converting to information
    Now: Time:= Clock;
BEGIN
  Figs(1) := (Year(Now));
  Figs(2) := (Month(Now));
  Figs(3) := (Day(Now));
  Figs(4) := (Integer(Seconds(Now)))/3600;
  Figs(5) := (Integer(Seconds(Now))) REM 3600 / 60;
  Figs(6) := (Integer(Seconds(Now))) REM  60;
  Figs(7) := (Integer(Seconds(Now)));
END TIME_EX_1;

             ------------------------------------------------
  PROCEDURE Time_Ex_2 IS
  -- Pre: Basic_Num_IO.ads is defined
  -- Post: Time data is available for converting to information
  Now: Time:= Clock;
BEGIN
  Ada.Text_IO.New_Line(2);
```

```
  Put(Year(Now), Width => 40);Put('-');
  Put(Month(Now), Width => 1);Put('-');
  Put(Day(Now), Width => 1); New_Line;
  Put(Integer(Seconds(Now))/3600, Width => 38);Put(';');
  PUT(Integer(Seconds(Now)) REM 3600 / 60, Width => 1); Put(';');
  PUT(Integer(Seconds(Now)) REM  60, Width => 1); New_Line;
  Dates(1) := (Year(Now));
  Dates(2) := (Month(Now));
  Dates(3) := (Day(Now));
  Dates(4) := (Integer(Seconds(Now)))/3600;
  Dates(5) := (Integer(Seconds(Now))) REM 3600 / 60;
  Dates(6) := (Integer(Seconds(Now))) REM  60;
  Dates(7) := (Integer(Seconds(Now)));
END TIME_EX_2;

          ---------------------------------------------------

         -- Finds the G.C.D of two cefficients
  FUNCTION GCD (M,N: IN Positive) RETURN Positive IS
  --  Pre: M and N are defined.
  --  Post: Returns the greatest common divisor of M and N.
  Result: Positive;

BEGIN  -- GCD

  IF (N <= M) AND (M REM N = 0) THEN
    Result := N;
  ELSIF M < N THEN
    Result := GCD (N,M);
  ELSE
    Result := GCD(N, M REM N);
  END IF;
  RETURN Result;

END GCD;

           ---------------------------------------------------

  -- finds VeeZero_ZY for the current normal

  PROCEDURE VeeZero_ZY
   (Item_1: IN OUT Integer; Item_2: IN OUT Integer; Item_3: IN OUT Integer) IS
  -- PRE : Normal vector is defined
  -- Post: Vector VeeZero is defined.

BEGIN  -- VeeZero

   Epsilon_X := GCD( M => ABS Item_2, N => ABS Item_3);

   S(1) :=0;
   S(2) := Item_3/Epsilon_X;
   S(3) := -Item_2/Epsilon_X;


END VeeZero_ZY;
```

```
          ------------------------------------------------

  -- finds VeeZero_ZX (<=Y=0) for the current normal

  PROCEDURE VeeZero_ZX
    (Item_1: IN OUT Integer; Item_2: IN OUT Integer;Item_3: IN OUT Integer) IS
  -- PRE : Procedure Normals is defined
  -- Post: Vector VeeZero is defined.

  -- Item_1 := E(U), Item_2:= F(U), Item_3:= G(U) in main program

BEGIN  -- VeeZero_ZX

   Epsilon_Y := GCD( M => ABS Item_1, N => ABS Item_3);

   SS(1):= -Item_3/Epsilon_Y;
   SS(2):= 0/Epsilon_Y;
   SS(3):= Item_1/Epsilon_Y;

END VeeZero_ZX;

          ------------------------------------------------

  -- finds VeeZero_XY (<=Z=0) for the current normal

  PROCEDURE VeeZero_XY
   (Item_1: IN OUT Integer; Item_2: IN OUT Integer;Item_3: IN OUT Integer) IS
  -- PRE : Procedure Normals is defined
  -- Post: Vector VeeZero is defined.

  -- Item_1 := E(U), Item_2:= F(U), Item_3:= G(u) in main program

BEGIN  -- VeeZero_XY

   Epsilon_Z := GCD( M => ABS Item_1, N => ABS Item_2);

   SSS(1):= Item_2/ Epsilon_Z;
   SSS(2):= -Item_1/Epsilon_Z;
   SSS(3):= 0/Epsilon_Z;

END VeeZero_XY;

          ------------------------------------------------

  -- Large CrossProduct ZY - Current PnOne x Current VeeZero

  PROCEDURE LargeCrossProduct_ZY IS
  -- Pre: VeeZeroThree(SThree) is defined
  -- Pre: PnOne Is define
  -- it is verified the product (VeeOne x VeeZero) = the current normal

BEGIN  -- Large_CrossProduct

  LargeCrossProduct_1(1):= Pn(2)*S(3) - Pn(3)*S(2);
  LargeCrossProduct_1(2):= -(Pn(1)*S(3) - Pn(3)*S(1));
  LargeCrossproduct_1(3):= Pn(1)*S(2) - Pn(2)* S(1);
```

```
END LargeCrossProduct_ZY;

        ------------------------------------------------------

  -- Large CrossProduct ZX - Current PnOne x Current VeeZero

  PROCEDURE LargeCrossProduct_ZX IS
  -- Pre: VeeZeroThree(SThree) is defined
  -- Pre: PnOne Is define
  -- it is verified the product (VeeOne x VeeZero) = the current normal

BEGIN  -- Large_CrossProduct

  LargeCrossProduct_2(1):= Pn(2)*SS(3) - Pn(3)*SS(2);
  LargeCrossProduct_2(2):= -(Pn(1)*SS(3) - Pn(3)*SS(1));
  LargeCrossproduct_2(3):= Pn(1)*SS(2) - Pn(2)*SS(1);

END LargeCrossProduct_ZX;

        ------------------------------------------------------

  -- Large CrossProduct XY- Current PnOne x Current VeeZero

  PROCEDURE LargeCrossProduct_XY IS
  -- Pre: VeeZeroThree(SThree) is defined
  -- Pre: PnOne Is define
  -- it is verified the product (VeeOne x VeeZero) = the current normal

BEGIN  -- Large_CrossProduct

  LargeCrossProduct_3(1):= Pn(2)*SSS(3) - Pn(3)*SSS(2);
  LargeCrossProduct_3(2):= -(Pn(1)*SSS(3) - Pn(3)*SSS(1));
  LargeCrossproduct_3(3):= Pn(1)*SSS(2) - Pn(2)*SSS(1);

END LargeCrossProduct_XY;

        ------------------------------------------------------
Begin -- General_Decryption_Program_Mark_0
  Load_n_Scramble_Encryption_Numbers;
  Load_n_Scramble_Normal_Vector_I_Coefficients;
  Load_n_Scramble_Normal_Vector_J_Coefficients;
  Load_n_Scramble_Normal_Vector_K_Coefficients;
  Load_n_Scramble_Change_of_Origin_Vector_I_Coefficients;
  Load_n_Scramble_Change_of_Origin_Vector_J_Coefficients;
  Load_n_Scramble_Change_of_Origin_Vector_K_Coefficients;
  Load_n_Scramble_Change_of_Origin_Vector_II_Coefficients;
  Load_n_Scramble_Change_of_Origin_Vector_JJ_Coefficients;
  Load_n_Scramble_Change_of_Origin_Vector_KK_Coefficients;
  TIME_EX_1;
  TIME_EX_2;
  Ada.Text_IO.Put
  (Item =>  " A demonstration program of decryption at work > ");
  Ada.Text_IO.New_Line(2);
  Ada.Text_IO.Put(Item => " Prepared Test Programs in files called :");
  Ada.Text_IO.New_Line(2);
```

```
Ada.Text_IO.Put(Item=> "  CipherTextFile_1.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_9.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_100.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_500.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_1000.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_2000.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_4000.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_10000.dat >");
Ada.Text_IO.New_Line;
Ada.Text_IO.Put(Item=> "  CipherTextFile_30000.dat >");
Ada.Text_IO.New_Line(2);
--get input file name and open it
Ada.Text_IO.Put(Item => " Please enter the name of the file to decrypt >");
Ada.Text_IO.New_Line(2);
Ada.Text_IO.Put(Item => " ");
Ada.Text_IO.Get_Line(Item => InFileName, Last=> InNameLength);
Ada.Text_IO.New_Line(2);
Ada.Text_IO.Open(File => Indata,
Mode => Ada.Text_IO.In_File,Name => InFileName(1 .. InNameLength));
-- get output file name and create it
Ada.Text_IO.Put
 (Item => " Please enter the name of the decrypted file >");
Ada.Text_IO.New_Line(2);
Ada.Text_IO.Put
 (Item=> " Call it MessageTextFile_n.dat for whatever 'n' may be >");
Ada.Text_IO.New_Line(2);
Ada.Text_IO.Put(Item => " ");
Ada.Text_IO.Get_Line(Item => OutFileName,Last => OutNameLength);
Ada.Text_IO.Create(File => OutData,
Mode => Ada.Text_IO.Out_File,Name => OutFileName(1 .. OutNameLength));
-- copy and encrypt input file to outputfile, character by character
Ada.Text_IO.New_Line;
Time_Ex_1;
Counter:= 0;  --initialising normals counter
Count  := 0;  --initialising change-of-origin Count
Total  := 0;  -- Initialising Total
Line_Number:= 0; --Initialising Line counter
LOOP
 BEGIN  -- exception block
  EXIT WHEN Ada.Text_IO.End_of_File(File => InData);
   LOOP
    EXIT WHEN Ada.TexT_IO.End_of_Line(File => Indata);
Counter := Counter+1;
  IF Counter REM 1000 = 0 THEN --Normals resetting
    Counter:= 1;
  END IF;
  Count:= Count+1;
IF Count REM 14250 = 0 THEN    -- change-of-origin resetting
   Count:= 1;
```

```
  END IF;
  Total := Total + 1;
    FOR I IN 1 .. 3 LOOP
      Ada.Integer_Text_IO.Get(File => InData, Item => W(I));
      Ada.Integer_Text_IO.Put(Item => W(I), Width => 8);
    END LOOP;
  Ada.Text_IO.Put(Item => "      - Current ciphertext item for decryption");
  Ada.text_IO.New_Line;
  Ada.Integer_Text_IO.Put(Item => Array_10(Count), Width => 8);
  Ada.Integer_Text_IO.Put(Item => Array_17(Count), Width => 8);
  Ada.Integer_Text_IO.Put(Item => Array_15(Count), Width => 8);
  Ada.Text_IO.Put(Item => "      - Current change_of_origin vector");
  Ada.text_IO.New_Line;
  Pn(1):= W(1) - Array_17(Count);-- 11  18  10  12 19 21 20 17 16 13 18 20 11 21 19 16 18 17
  Pn(2):= W(2) - Array_14(Count);-- 19  17  17  16 14 18 13 19 11 21 19 17 21 14 10 21 13 14
  Pn(3):= W(3) - Array_11(Count);-- 21  13  15  21 17 11 15 14 19 18 12 13 16 11 15 13 10 11
    For I in 1 .. 3 Loop
     Ada.integer_Text_IO.Put(Item => Pn(I), Width => 8);
    End Loop;
    Ada.text_IO.Put(Item => "      - This is the associated position vector  ");
    Ada.Text_IO.New_Line;
    If Total Rem 3 = 1 Then
      VeeZero_ZY(Item_1 => Array_4(Counter), Item_2 => Array_6(Counter), Item_3 =>
Array_8(Counter));
      Ada.Integer_Text_IO.Put(Item => S(1), Width => 8);
      Ada.Integer_Text_IO.Put(Item => S(2), Width => 8);
      Ada.Integer_Text_IO.Put(Item => S(3), Width => 8);
      Ada.Text_IO.Put(Item => "      - VeeZero_ZY for this Normal");
      LargeCrossproduct_ZY;
      Ada.Text_IO.New_Line;
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_1(1), Width => 4);
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_1(2), Width => 10);
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_1(3), Width => 10);
      Ada.Text_IO.Put(Item => "    - Pn cross VeeZero_ZY");
      n(1):= LargeCrossproduct_1(1)/Array_4(Counter);
      n(2):= LargeCrossproduct_1(2)/Array_6(Counter);
      n(3):= LargeCrossproduct_1(3)/Array_8(Counter);
      Ada.Text_IO.New_Line;
      Ada.Integer_Text_IO.Put(Item => n(1), Width => 8);
      Ada.Integer_Text_IO.Put(Item => n(2), Width => 8);
      Ada.Integer_Text_IO.Put(Item => n(3), Width => 8);
      Ada.Text_IO.Put(Item => "      - Current 'n ' for decryption");
      Ada.Text_IO.New_Line;
      Decipher(Item => n(1));
      MessageText(Total):= AlphabetChar;
      Ada.Text_IO.Put(Item => "          ");
      Ada.Text_IO.Put(Item =>MessageText(Total));
      Ada.Text_IO.Put(File => OutData, Item => MessageText(Total));
      Ada.Text_IO.Put(Item => "                  - Deciphered 'n'- this is the messagetext ");
      Ada.Text_IO.New_Line(1);
      Ada.Text_IO.Put(Item => "                              - Character number ");
      Ada.Integer_Text_IO.Put(Item => Total, Width => 1);
      Ada.Text_IO.New_Line(1);
      Ada.Text_IO.Put(Item => " ------------------------------------------------------------------------------");
      Ada.Text_IO.New_Line(1);
      End If;
```

```
   If Total Rem 3 = 2 Then
      VeeZero_ZX(Item_1 => Array_4(Counter), Item_2 => Array_6(Counter), Item_3 =>
Array_8(Counter));
      Ada.Integer_Text_IO.Put(Item => SS(1), Width => 8);
      Ada.Integer_Text_IO.Put(Item => SS(2), Width => 8);
      Ada.Integer_Text_IO.Put(Item => SS(3), Width => 8);
      Ada.Text_IO.Put(Item => "    - VeeZero_ZX for this Normal");
      LargeCrossproduct_ZX;
      Ada.Text_IO.New_Line;
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_2(1), Width => 4);
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_2(2), Width => 10);
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_2(3), Width => 10);
      Ada.Text_IO.Put(Item => "   - Pn cross VeeZero_ZX");
      n(1):= LargeCrossproduct_2(1)/Array_4(Counter);
      n(2):= LargeCrossproduct_2(2)/Array_6(Counter);
      n(3):= LargeCrossproduct_2(3)/Array_8(Counter);
      Ada.Text_IO.New_Line;
      Ada.Integer_Text_IO.Put(Item => n(1), Width => 8);
      Ada.Integer_Text_IO.Put(Item => n(2), Width => 8);
      Ada.Integer_Text_IO.Put(Item => n(3), Width => 8);
      Ada.Text_IO.Put(Item => "      - Current 'n ' for decryption");
      Ada.Text_IO.New_Line;
      Decipher(Item => n(1));
      MessageText(Total):= AlphabetChar;
      Ada.Text_IO.Put(Item => "          ");
      Ada.Text_IO.Put(Item =>MessageText(Total));
      Ada.Text_IO.Put(File => OutData, Item => MessageText(Total));
      Ada.Text_IO.Put(Item => "                  - Deciphered 'n'- this is the messagetext ");
      Ada.Text_IO.New_Line(1);
      Ada.Text_IO.Put(Item => "                              - Character number ");
      Ada.Integer_Text_IO.Put(Item => Total, Width => 1);
      Ada.Text_IO.New_Line(1);
      Ada.Text_IO.Put(Item => " ------------------------------------------------------------------------------");
      Ada.Text_IO.New_Line(1);
      End If;
   If Total Rem 3 = 0 Then
      VeeZero_XY(Item_1 => Array_4(Counter), Item_2 => Array_6(Counter), Item_3 =>
Array_8(Counter));
      Ada.Integer_Text_IO.Put(Item => SSS(1), Width => 8);
      Ada.Integer_Text_IO.Put(Item => SSS(2), Width => 8);
      Ada.Integer_Text_IO.Put(Item => SSS(3), Width => 8);
      Ada.Text_IO.Put(Item => "      - VeeZero_XY for this Normal");
      LargeCrossproduct_XY;
      Ada.Text_IO.New_Line;
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_3(1), Width => 4);
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_3(2), Width => 10);
      Ada.Integer_Text_IO.Put(Item => LargeCrossproduct_3(3), Width => 10);
      Ada.Text_IO.Put(Item => "   - Pn cross VeeZero_XY");
      n(1):= LargeCrossproduct_3(1)/Array_4(Counter);
      n(2):= LargeCrossproduct_3(2)/Array_6(Counter);
      n(3):= LargeCrossproduct_3(3)/Array_8(Counter);
      Ada.Text_IO.New_Line;
      Ada.Integer_Text_IO.Put(Item => n(1), Width => 8);
      Ada.Integer_Text_IO.Put(Item => n(2), Width => 8);
      Ada.Integer_Text_IO.Put(Item => n(3), Width => 8);
      Ada.Text_IO.Put(Item => "      - Current 'n ' for decryption");
```

```
      Ada.Text_IO.New_Line;
      Decipher(Item => n(1));
      MessageText(Total):= AlphabetChar;
      Ada.Text_IO.Put(Item => "          ");
      Ada.Text_IO.Put(Item =>MessageText(Total));
      Ada.Text_IO.Put(File => OutData, Item => MessageText(Total));
      Ada.Text_IO.Put(Item => "                  - Deciphered 'n'- this is the messagetext ");
      Ada.Text_IO.New_Line(1);
      Ada.Text_IO.Put(Item => "                              - Character number ");
      Ada.Integer_Text_IO.Put(Item => Total, Width => 1);
      Ada.Text_IO.New_Line(1);
      Ada.Text_IO.Put(Item => " ------------------------------------------------------------------------------");
      Ada.Text_IO.New_Line(1);
    End If;
  Ada.Text_IO.New_Line(1);
  END LOOP;
    Ada.Text_IO.Skip_Line(File => InData);
    Ada.Text_IO.New_Line(File => OutData);
  EXCEPTION
  WHEN Ada.Text_IO.End_Error =>
  EXIT;
  END;  --exception block
  END LOOP;
  Ada.Text_IO.Close(File => Indata);
  Ada.Text_IO.Close(File => Outdata);
  Ada.Text_IO.New_Line(1);
  Ada.Text_IO.Put(Item => "           To see the messagetext - press any key/return to continue >
");
  Ada.Text_IO.Get(Item => View);
  Ada.Text_IO.New_Line(1);
   --  reopen the Messagetext file, read and display the contents on screen
  Ada.Text_IO.Set_Line_Length(77);
  Ada.Text_IO.Open (File => Outdata, Mode=>Ada.Text_IO.In_File,
  Name=>OutFileName(1 .. OutNameLength));
  Line_Number:=0;
  Check := 0;
   While NOT Ada.Text_IO.End_of_File(File => OutData) LOOP
    BEGIN  -- Exceptions Block
     While NOT Ada.TEXT_IO.End_of_Line(File => OutData) LOOP
     Line_Number:= Line_Number +1;
        Check := Check +1;
        Ada.Text_IO.Get(File => OutData, Item => MessageText(I));
        Ada.Text_IO.Put(Item => MessageText(I)); -- View MessageText
      IF Check REM 5000 = 0 THEN  -- Halts the output for viewing
      --IF LineNumber = 60  THEN  -- say -like linenumber 60 for example
        Ada.Text_IO.New_Line(2);
        Ada.Text_IO.Put(Item => "   ");
        Ada.Integer_Text_IO.Put(Item => Check, Width =>1);
        Ada.Text_IO.Put(Item =>" so far - ");
        Ada.Integer_Text_IO.Put(Item => (Total - Check), Width => 1);
        Ada.Text_IO.Put(Item =>" still to go. ");
        Ada.Text_IO.New_Line(1);
        Ada.Text_IO.Put(Item =>" to continue-press any key/return >");
        Ada.Text_IO.Get(Item => View);
        Ada.Text_IO.New_Line(1);--
      END IF;
```

```
      END LOOP;
       Ada.Text_IO.Skip_Line(File => Outdata);
       Ada.Text_IO.New_Line;
  EXCEPTION
  WHEN Ada.Text_IO.End_Error =>
  EXIT;
  END;  --exception block
  END LOOP;
  Ada.Text_IO.Close(File => Outdata);
  Ada.Text_IO.New_Line(2);--
  Ada.Text_IO.New_Line(1);
  Ada.integer_Text_IO.Put(Item => Check, Width => 30);
  Ada.Text_IO.Put(Item => " Items of MessageText.");
  Ada.Text_IO.New_Line;
End General_Decryption_Program_Mark_0;
```