```
Open "OUTPUT.txt" For Output As #1

' .................. Config ....................
Dim Shared MATRIX_MAX_SIZE%
Dim Shared ARR_MAX_LEN%
MATRIX_MAX_SIZE% = 50
ARR_MAX_LEN% = arr_len_sq%(MATRIX_MAX_SIZE%)

' .................. Matrices .................'
' Main Matrices
Dim Shared AR%
Dim Shared AC%
Dim Shared matA#(ARR_MAX_LEN%)

Dim Shared matAName$
Dim Shared matBName$
matAName$ = "Matrix-In"
matBName$ = "Matrix-Sub"

' Result Matrix
Dim Shared RR%
Dim Shared RC%
Dim Shared matRes#(ARR_MAX_LEN%)

Dim Shared matResName$
Dim Shared lastMatResName$
matResName$ = "Result"
lastMatResName$ = matResName$

' ........... COMMANDS ...........
Dim Shared COMMAND_PRINT_INFO$
Dim Shared COMMAND_PRINT_MATRIX_A$
Dim Shared COMMAND_PRINT_MATRIX_LAST_RESULT$
Dim Shared COMMAND_SET_MATRIX_A_AS_RESULT$
Dim Shared COMMAND_RESTORE_HARDCODED_DATA_POINTER$
Dim Shared COMMAND_MAT_ADD_SCALER$
Dim Shared COMMAND_MAT_MULT_SCALER$
Dim Shared COMMAND_MAT_DETERMINANT$
Dim Shared COMMAND_MAT_TRANSPOSE$
Dim Shared COMMAND_MAT_MINOR_MATRIX$
Dim Shared COMMAND_MAT_COFACTOR_MATRIX$
Dim Shared COMMAND_MAT_ADJOINT_MATRIX$
Dim Shared COMMAND_MAT_INVERSE_MATRIX$
Dim Shared COMMAND_MAT_POWER_MATRIX$
Dim Shared COMMAND_MAT_ADD_MATRIX$
Dim Shared COMMAND_MAT_MULT_MATRIX$

' 1. Basic Commands
COMMAND_PRINT_INFO$ = "info"
COMMAND_PRINT_MATRIX_A$ = "input"
COMMAND_PRINT_MATRIX_LAST_RESULT$ = "result"
COMMAND_SET_MATRIX_A_AS_RESULT$ = "continue"
COMMAND_RESTORE_HARDCODED_DATA_POINTER$ = "restore data"

' 2. Matrix Unitary Transform Commands
COMMAND_MAT_ADD_SCALER$ = "add scaler"
COMMAND_MAT_MULT_SCALER$ = "mult scaler"
COMMAND_MAT_DETERMINANT$ = "det"
COMMAND_MAT_TRANSPOSE$ = "transpose"
```

```
COMMAND_MAT_MINOR_MATRIX$ = "minor"
COMMAND_MAT_COFACTOR_MATRIX$ = "cofactor"
COMMAND_MAT_ADJOINT_MATRIX$ = "adjoint"
COMMAND_MAT_INVERSE_MATRIX$ = "inverse"
COMMAND_MAT_POWER_MATRIX$ = "pow"

' 3. Matrix Binary Transform Commands
COMMAND_MAT_ADD_MATRIX$ = "add"
COMMAND_MAT_MULT_MATRIX$ = "mult"

'.................................. MAIN LAUNCH ..............................
Cls
Screen 12

lb
print_app_header
lb
lb
print_aux_info
lb

' Input Main MATRIX
label_input_mat_a:
pl "................ " + matAName$ + " .............."

AR% = in_int%(" ROWS: ")
AC% = in_int%(" COLS: ")

If AR% < 1 Or AC% < 1 Or AR% > MATRIX_MAX_SIZE% Or AC% > MATRIX_MAX_SIZE% Then
   pl "ERR: Matrix Rows And Columns must be > 0 and <= " + trim_val$(MATRIX_MAX_SIZE%) + ", given Order: " +
format_order$(AR%, AC%)
   lb
   GoTo label_input_mat_a
End If
Call input_or_read_matrix(matAName$, matA#(), AR%, AC%)
Call print_matrix(matAName$, matA#(), AR%, AC%)

' Enter Commands'
lb
print_commands_info
lb
label_enter_command:
lb
com$ = trim_str$(in_str$("RC> ENTER COMMAND: "))
' Basic Commands
If com$ = COMMAND_PRINT_INFO$ Then
   Call print_info(0)
ElseIf com$ = COMMAND_PRINT_MATRIX_A$ Then
   Call print_matrix_a
ElseIf com$ = COMMAND_PRINT_MATRIX_LAST_RESULT$ Then
   Call print_matrix_result_last
ElseIf com$ = COMMAND_SET_MATRIX_A_AS_RESULT$ Then
   Call set_matrix_a_as_result
ElseIf com$ = COMMAND_RESTORE_HARDCODED_DATA_POINTER$ Then
   Call restore_hardcoded_data

   ' Matrix Unitary Transform Commands
ElseIf com$ = COMMAND_MAT_ADD_SCALER$ Then
   Call main_add_scaler
```

```
ElseIf com$ = COMMAND_MAT_MULT_SCALER$ Then
   Call main_mult_scaler
ElseIf com$ = COMMAND_MAT_DETERMINANT$ Then
   Call main_determinant
ElseIf com$ = COMMAND_MAT_TRANSPOSE$ Then
   Call main_transpose
ElseIf com$ = COMMAND_MAT_MINOR_MATRIX$ Then
   Call main_minor_matrix
ElseIf com$ = COMMAND_MAT_COFACTOR_MATRIX$ Then
   Call main_cofactor_matrix
ElseIf com$ = COMMAND_MAT_ADJOINT_MATRIX$ Then
   Call main_adjoint_matrix
ElseIf com$ = COMMAND_MAT_INVERSE_MATRIX$ Then
   Call main_invert_matrix
ElseIf com$ = COMMAND_MAT_POWER_MATRIX$ Then
   Call main_pow_matrix

   ' Matrices Binary Transforms'
ElseIf com$ = COMMAND_MAT_ADD_MATRIX$ Then
   Call main_add_matrices
ElseIf com$ = COMMAND_MAT_MULT_MATRIX$ Then
   Call main_mult_matrices
ElseIf Len(com$) > 0 Then
   pl "RC> Invalid Command: " + trim_str$(com$) + ". For user guide, enter command: " + COMMAND_PRINT_INFO$
End If

GoTo label_enter_command

Sub print_app_header
   pl "================== MATRIX CALCULATOR (Made by RC) ========================="
End Sub

Sub print_aux_info
   pl "## Restrictions"
   pl "Maximum Matrix Order : " + trim_val$(MATRIX_MAX_SIZE%)
End Sub

Sub print_commands_info
   pl "................... COMMANDS ................."
   pl "## Basic"
   pl " " + COMMAND_PRINT_INFO$ + " → Information Guide"
   pl " " + COMMAND_PRINT_MATRIX_A$ + " → Input Matrix"
   pl " " + COMMAND_PRINT_MATRIX_LAST_RESULT$ + " → Last Result Matrix"
   pl " " + COMMAND_SET_MATRIX_A_AS_RESULT$ + " → Continue Calculation (Set [Input] = [Result])"
   pl " " + COMMAND_RESTORE_HARDCODED_DATA_POINTER$ + " → Restore Hardcoded Smaple Data"
   lb
   pl "## Unitary Trsnaforms"
   pl " " + COMMAND_MAT_ADD_SCALER$ + " → Add Scaler"
   pl " " + COMMAND_MAT_MULT_SCALER$ + " → Multipliy Scaler"
   pl " " + COMMAND_MAT_DETERMINANT$ + " → Determinant"
   pl " " + COMMAND_MAT_TRANSPOSE$ + " → Transpose"
   pl " " + COMMAND_MAT_MINOR_MATRIX$ + " → Minor Matrix"
   pl " " + COMMAND_MAT_COFACTOR_MATRIX$ + " → Cofactor Matrix"
   pl " " + COMMAND_MAT_ADJOINT_MATRIX$ + " → Adjoint Matrix"
   pl " " + COMMAND_MAT_INVERSE_MATRIX$ + " → Inverse Matrix"
   pl " " + COMMAND_MAT_POWER_MATRIX$ + " → Power Matrix"
   lb
   pl "## Binary Transforms"
   pl " " + COMMAND_MAT_ADD_MATRIX$ + " → Add Matrices (with scale)"
```

```
  pl " " + COMMAND_MAT_MULT_MATRIX$ + " → Multipliy Matrices (with scale)"
End Sub

Sub print_info (with_app_header%)
  If with_app_header% = 1 Then
    lb
    Call print_app_header
    lb
  End If

  lb
  Call print_aux_info
  lb
  Call print_commands_info
  lb
End Sub

Sub print_matrix_a
  Call print_matrix(matAName$, matA#(), AR%, AC%)
End Sub

Sub print_matrix_result (name$)
  lastMatResName$ = name$
  Call print_matrix(name$, matRes#(), RR%, RC%)
End Sub

Sub print_matrix_result_last
  Call print_matrix(lastMatResName$ + " (Last Result)", matRes#(), RR%, RC%)
End Sub

Sub set_matrix_a_as_result
  AR% = RR%
  AC% = RC%
  Call copy_mat(matRes#(), matA#(), RR%, RC%)
  pl "RC> Input Matrix set to last Result Matrix"
End Sub

Sub restore_hardcoded_data
  Restore
  pl "RC> Hardcoded Sample Data Restored"
End Sub

Sub main_add_scaler
  add# = in_doub#("Enter Scaler Addant: ")

  RR% = AR%
  RC% = AC%
  Call mat_add_scaler(matA#(), matRes#(), add#, AR%, AC%)
  Call print_matrix_result("[Result] = " + trim_val$(add#) + " + [" + matAName$ + "]")
End Sub

Sub main_mult_scaler
  mult# = in_doub#("Enter Scaler Multiplier: ")

  RR% = AR%
  RC% = AC%
  Call mat_mult_scaler(matA#(), matRes#(), mult#, AR%, AC%)
  Call print_matrix_result("[Result] = " + trim_val$(mult#) + " * [" + matAName$ + "]")
End Sub
```

```
Sub main_determinant
  If AR% <> AC% Then
    pl "ERR: Determinant is only defined for a square matrix, given matrix order: " + format_order$(AR%, AC%)
  Else
    det# = determinant#(matA#(), AR%)
    pl "DETERMINANT([" + matAName$ + "]): " + trim_val$(det#)
  End If
End Sub

Sub main_transpose
  RR% = AC%
  RC% = AR%
  Call transpose(matA#(), matRes#(), AR%, AC%)
  Call print_matrix_result("[Result] = TRANSPOSE([" + matAName$ + "])")
End Sub

Sub main_minor_matrix
  If AR% <> AC% Then
    pl "ERR: Minor matrix is only defined for a square matrix, given matrix order: " + format_order$(AR%, AC%)
  Else
    RR% = AR%
    RC% = AC%
    Call minor_matrix(matA#(), matRes#(), AR%)
    Call print_matrix_result("[Result] = MINOR([" + matAName$ + "])")
  End If
End Sub

Sub main_cofactor_matrix
  If AR% <> AC% Then
    pl "ERR: Cofactor matrix is only defined for a square matrix, given matrix order: " + format_order$(AR%, AC%)
  Else
    RR% = AR%
    RC% = AC%
    Call cofactor_matrix(matA#(), matRes#(), AR%)
    Call print_matrix_result("[Result] = COFACTOR([" + matAName$ + "])")
  End If
End Sub

Sub main_adjoint_matrix
  If AR% <> AC% Then
    pl "ERR: Adjoint matrix is only defined for a square matrix, given matrix order: " + format_order$(AR%, AC%)
  Else
    RR% = AR%
    RC% = AC%
    Call adjoint_matrix(matA#(), matRes#(), AR%)
    Call print_matrix_result("[Result] = ADJOINT([" + matAName$ + "])")
  End If
End Sub

Sub main_invert_matrix
  If AR% <> AC% Then
    pl "ERR: Inverse matrix is only defined for a square matrix, given matrix order: " + format_order$(AR%, AC%)
  Else
    det# = determinant#(matA#(), AR%)
    If det# = 0 Then
      pl "ERR: " + matAName$ + " is not invertible since it is SINGULAR (det = 0)"
    Else
      RR% = AR%
```

```
      RC% = AC%
      Call adjoint_matrix(matA#(), matRes#(), AR%)
      Call mat_mult_scaler(matRes#(), matRes#(), 1 / det#, RR%, RC%)
      Call print_matrix_result("[Result] = INVERSE([" + matAName$ + "])")
    End If
  End If
End Sub

Sub main_pow_matrix
  If AR% <> AC% Then
    pl "ERR: Power matrix is only defined for a square matrix, given matrix order: " + format_order$(AR%, AC%)
  Else
    pow% = in_int%("Enter the power to raise matrix to: ")
    pow_copy% = pow%

    RR% = AR%
    RC% = AC%
    Call pow_mat(matA#(), matRes#(), pow_copy%, AR%)
    Call print_matrix_result("[Result] = [" + matAName$ + "] ^ " + trim_val$(pow%))
  End If
End Sub

Sub main_add_matrices
  Dim matB#(arr_len%(AR%, AC%))

  Call input_or_read_matrix(matBName$, matB#(), AR%, AC%)
  Call print_matrix(matBName$, matB#(), AR%, AC%)
  lb
  a_scale# = in_doub#("Enter " + matAName$ + " entries scale: ")
  b_scale# = in_doub#("Enter " + matBName$ + " entries scale: ")

  RR% = AR%
  RC% = AC%
  Call add_mat_with_scale(matA#(), matB#(), matRes#(), AR%, AC%, a_scale#, b_scale#)
  Call print_matrix_result("[Result] = (" + trim_val$(a_scale#) + " * [" + matAName$ + "]) + (" + trim_val$(b_scale#) + " * [" +
matBName$ + "])")
End Sub

Sub main_mult_matrices
  b_cols% = in_int%("Enter " + matBName$ + " COLS: ")
  If b_cols% < 1 Or b_cols% > MATRIX_MAX_SIZE% Then
    pl "ERR: Matrix coulmns should be > 0 and <= " + trim_val$(MATRIX_MAX_SIZE%) + ", given order " + format_order$(AC%,
b_cols%)
    lb
    call main_mult_matrices
  Else
    Dim matB#(arr_len%(AC%, b_cols%))

    Call input_or_read_matrix(matBName$, matB#(), AC%, b_cols%)
    Call print_matrix(matBName$, matB#(), AC%, b_cols%)
    lb
    a_scale# = in_doub#("Enter " + matAName$ + " entries scale: ")
    b_scale# = in_doub#("Enter " + matBName$ + " entries scale: ")

    RR% = AR%
    RC% = b_cols%
    Call mult_mat_with_scale(matA#(), matB#(), matRes#(), AR%, AC%, b_cols%, a_scale#, b_scale#)
    Call print_matrix_result("[Result] = (" + trim_val$(a_scale#) + " * [" + matAName$ + "]) X (" + trim_val$(b_scale#) + " * [" +
matBName$ + "])")
```

```
   End If
End Sub

' Matrix representation as 1D array Utility, Mostly depend on column count of the matrix
Function is_even (num%, even_res, odd_res)
   If (num% Mod 2) = 0 Then
      is_even = even_res
   Else
      is_even = odd_res
   End If
End Function

Function is_even11% (num%)
   is_even11% = is_even(num%, 1, -1)
End Function

Function arr_len% (rows%, cols%)
   arr_len% = rows% * cols%
End Function

Function arr_len_sq% (size%)
   arr_len_sq% = arr_len%(size%, size%)
End Function

Function arr_index% (i%, j%, cols%)
   arr_index% = ((i% - 1) * cols%) + j%
End Function

Function mat_i% (arr_ind%, cols%)
   temp% = (arr_ind% - 1) \ cols%
   mat_i% = temp% + 1
End Function

Function mat_j% (arr_ind%, cols%)
   temp% = (arr_ind% - 1) Mod cols%
   mat_j% = temp% + 1
End Function

Sub mat_ij (arr_ind%, cols%, ij())
   ij① = mat_i%(arr_ind%, cols%)
   ij② = mat_j%(arr_ind%, cols%)
End Sub

Function get_ij# (arr#(), i%, j%, cols%)
   get_ij# = arr#(arr_index%(i%, j%, cols%))
End Function

' ....................... Matrix Unitary Transformations ............................
' CAUTION: To overwrite source, pass ource as Result Array/Matrix (NOT RECOMMENDED)

Sub fill_arr (arr#(), num#, length%)
   For i% = 1 To length%
      arr#(i%) = num#
   Next i%
End Sub

Sub fill_mat (mat#(), num#, rows%, cols%)
   Call fill_arr(mat#(), num#, arr_len%(rows%, cols%))
End Sub
```

```
Sub copy_arr (src#(), dest#(), length%)
   For i% = 1 To length%
      dest#(i%) = src#(i%)
   Next i%
End Sub

Sub copy_mat (src#(), dest#(), rows%, cols%)
   Call copy_arr(src#(), dest#(), arr_len%(rows%, cols%))
End Sub

Sub arr_add_scaler (arr#(), result#(), addant#, length%)
   For i% = 1 To length%
      result#(i%) = arr#(i%) + addant#
   Next i%
End Sub

Sub arr_mult_scaler (arr#(), result#(), multiplier#, length%)
   For i% = 1 To length%
      result#(i%) = arr#(i%) * multiplier#
   Next i%
End Sub

Sub mat_add_scaler (mat#(), result#(), addant#, rows%, cols%)
   Call arr_add_scaler(mat#(), result#(), addant#, arr_len%(rows%, cols%))
End Sub

Sub mat_mult_scaler (mat#(), result#(), multiplier#, rows%, cols%)
   Call arr_mult_scaler(mat#(), result#(), multiplier#, arr_len%(rows%, cols%))
End Sub

Sub transpose (mat#(), result#(), src_rows%, src_cols%)
   For i% = 1 To src_rows%
      For j% = 1 To src_cols%
         result#(arr_index%(j%, i%, src_rows%)) = mat#(arr_index%(i%, j%, src_cols%))
      Next j%
   Next i%
End Sub

' Result matrix must be atleast of len = (rows - 1) * (cols - 1)
Sub delete_row_col (mat#(), result#(), row%, col%, rows%, cols%)
   cur_i% = 1
   For i% = 1 To rows%
      If i% <> row% Then
         For j% = 1 To cols%
            If j% <> col% Then
               result#(cur_i%) = mat#(arr_index%(i%, j%, cols%))
               cur_i% = cur_i% + 1
            End If
         Next j%
      End If
   Next i%
End Sub

' SQUARE MATRIX
Function determinant# (mat#(), size%)
   If size% = 1 Then
      determinant# = mat#①
   ElseIf size% = 2 Then
```

```
    determinant# = (mat#① * mat#④) - (mat#② * mat#③)
  Else
    sum# = 0
    Dim temp#(arr_len%(size% - 1, size% - 1))

    For i% = 1 To size%
      Call delete_row_col(mat#(), temp#(), 1, i%, size%, size%)
      sum# = sum# + (get_ij#(mat#(), 1, i%, size%) * is_even11%(1 + i%) * determinant#(temp#(), size% - 1))
    Next i%

    determinant# = sum#
  End If
End Function

Function minor# (mat#(), i%, j%, size%)
  Dim temp#(arr_len%(size% - 1, size% - 1))
  Call delete_row_col(mat#(), temp#(), i%, j%, size%, size%)
  minor# = determinant#(temp#(), size% - 1)
End Function

Function cofactor# (mat#(), i%, j%, size%)
  cofactor# = is_even11%(i% + j%) * minor#(mat#(), i%, j%, size%)
End Function

Function is_singular# (mat#(), size%, true_val#, false_val#)
  det# = determinant#(mat#(), size%)
  If det# = 0 Then
    is_singular# = true_val#
  Else
    is_singular# = false_val#
  End If
End Function

Function is_singular10% (mat#(), size%)
  is_singular10% = is_singular#(mat#(), size%, 1, 0)
End Function

Sub minor_matrix_internal (mat#(), result#(), size%, do_cofactor%, do_transpose%)
  For i% = 1 To size%
    For j% = 1 To size%
      If do_transpose% = 1 Then
        ind% = arr_index%(j%, i%, size%)
      Else
        ind% = arr_index%(i%, j%, size%)
      End If

      If do_cofactor% = 1 Then
        result#(ind%) = cofactor#(mat#(), i%, j%, size%)
      Else
        result#(ind%) = minor#(mat#(), i%, j%, size%)
      End If
    Next j%
  Next i%
End Sub

Sub minor_matrix (mat#(), result#(), size%)
  Call minor_matrix_internal(mat#(), result#(), size%, 0, 0)
End Sub
```

```
Sub cofactor_matrix (mat#(), result#(), size%)
  Call minor_matrix_internal(mat#(), result#(), size%, 1, 0)
End Sub

Sub adjoint_matrix (mat#(), result#(), size%)
  Call minor_matrix_internal(mat#(), result#(), size%, 1, 1)
End Sub

' Matrix should be invertible (i.e det is non zero)
Sub invert_matrix (mat#(), result#(), size%)
  Call adjoint_matrix(mat#(), result#(), size%)
  det# = determinant#(mat#(), size%) ' TODO: caheck non-singular

  Call mat_mult_scaler(result#(), result#(), 1 / det#, size%, size%)
End Sub

'............................ Matrix Binary Transformations ............................

Sub add_arr_with_scale (arr_a#(), arr_b#(), result#(), length%, scale_a#, scale_b#)
  For i% = 0 To length%
    result#(i%) = (scale_a# * arr_a#(i%)) + (scale_b# * arr_b#(i%))
  Next i%
End Sub

Sub add_arr (arr_a#(), arr_b#(), result#(), length%)
  Call add_arr_with_scale(arr_a#(), arr_b#(), result#(), length%, 1, 1)
End Sub

Sub subtract_arr (arr_a#(), arr_b#(), result#(), length%)
  Call add_arr_with_scale(arr_a#(), arr_b#(), result#(), length%, 1, -1)
End Sub

Sub add_mat_with_scale (mat_a#(), mat_b#(), result#(), rows%, cols%, scale_a#, scale_b#)
  Call add_arr_with_scale(mat_a#(), mat_b#(), result#(), arr_len%(rows%, cols%), scale_a#, scale_b#)
End Sub

Sub add_mat (mat_a#(), mat_b#(), result#(), rows%, cols%)
  Call add_mat_with_scale(mat_a#(), mat_b#(), result#(), rows%, cols%, 1, 1)
End Sub

Sub subtract_mat (mat_a#(), mat_b#(), result#(), rows%, cols%)
  Call add_mat_with_scale(mat_a#(), mat_b#(), result#(), rows%, cols%, 1, -1)
End Sub

' Result matrix is of order  (rows_a% * cols_b% )
Sub mult_mat_with_scale (mat_a#(), mat_b#(), result#(), rows_a%, cols_a%, cols_b%, scale_a#, scale_b#)
  For i% = 1 To rows_a%
    For j% = 1 To cols_b%
      e# = 0
      For k% = 1 To cols_a%
        e# = e# + ((scale_a# * mat_a#(arr_index%(i%, k%, cols_a%))) * (scale_b# * mat_b#(arr_index%(k%, j%, cols_b%))))
      Next k%
      result#(arr_index%(i%, j%, cols_b%)) = e#
    Next j%
  Next i%
End Sub

Sub mult_mat (mat_a#(), mat_b#(), result#(), rows_a%, cols_a%, cols_b%)
  Call mult_mat_with_scale(mat_a#(), mat_b#(), result#(), rows_a%, cols_a%, cols_b%, 1, 1)
```

```
End Sub

Sub pow_mat (mat#(), result#(), pow%, size%)
  If pow% = 0 Then
    Call fill_mat(result#(), 1, size%, size%)
  Else
    If pow% < 0 Then
      Dim inv#(arr_len%(size%, size%))
      Call invert_matrix(mat#(), inv#(), size%)
      Call copy_mat(inv#(), mat#(), size%, size%)
      pow% = Int(Abs(pow%))
    End If

    If pow% = 1 Then
      Call copy_mat(mat#(), result#(), size%, size%)
    Else
      Dim temp#(arr_len%(size%, size%))
      Call copy_mat(mat#(), temp#(), size%, size%)

      For i% = 2 To pow%
        Call mult_mat(mat#(), temp#(), result#(), size%, size%, size%)
        Call copy_mat(result#(), temp#(), size%, size%)
      Next i%
    End If
  End If
End Sub

' .................................. Formatting ...............................
' Prints a given string WITHOUT line break
Sub p (s$)
  Print s$;
  Print #1, s$;
End Sub

' Prints a line break
Sub lb
  Print
  Print #1, ""
End Sub

' Prints given string WITH line break
Sub pl (s$)
  Print s$
  Print #1, s$
End Sub

Function trim_str$ (s$)
  trim_str$ = LTrim$(RTrim$(s$))
End Function

Function trim_val$ (i#)
  trim_val$ = trim_str$(Str$(i#))
End Function

Function format_ij$ (i%, j%, delimiter$)
  format_ij$ = trim_str$("(" + trim_val$(i%) + delimiter$ + trim_val$(j%) + ")")
End Function

Function format_order$ (rows%, cols%)
```

```
   format_order$ = format_ij$(rows%, cols%, "x")
End Function

Sub print_matrix (matrix_name$, matrix_arr#(), rows%, cols%)
   If rows% < 1 Or cols% < 1 Then
      pl "RC> " + matrix_name$ + " NOT SET"
   Else
      lb
      pl "............ " + matrix_name$ + " ..............."
      lb
      For i% = 1 To rows%
         For j% = 1 To cols%
            p trim_val$(get_ij#(matrix_arr#(), i%, j%, cols%)) + "  "
         Next j%
         lb
      Next i%
      lb
   End If
End Sub

'...................................... INPUT ............................................
Function in_str$ (caption$)
   p (caption$)
   Input "", v$
   Print #1, v$
   in_str$ = v$
End Function

Function in_int% (caption$)
   p (caption$)
   Input "", v%
   Print #1, trim_val$(v%)
   in_int% = v%
End Function

Function in_float! (caption$)
   p (caption$)
   Input "", v!
   Print #1, trim_val$(v!)
   in_float! = v!
End Function

Function in_doub# (caption$)
   p (caption$)
   Input "", v#
   Print #1, trim_val$(v#)
   in_doub# = v#
End Function

Sub input_matrix (matrix_name$, matrix_arr#(), rows%, cols%)
   lb
   pl "............ Input " + matrix_name$ + " " + format_order$(rows%, cols%) + " ..............."
   lb
   For i% = 1 To rows%
      pl "# ROW " + Str$(i%)
      For j% = 1 To cols%
         e# = in_doub#(matrix_name$ + " " + format_ij$(i%, j%, ",") + " : ")
         matrix_arr#(arr_index%(i%, j%, cols%)) = e#
      Next j%
```

```
      lb
   Next i%
   lb
End Sub

Sub read_array (arr#(), length%)
   For i% = 1 To length%
      Read e#
      arr#(i%) = e#
   Next i%
End Sub

Sub read_matrix (mat#(), rows%, cols%)
   Call read_array(mat#(), arr_len%(rows%, cols%))
End Sub

Sub input_or_read_matrix (matrix_name$, mat#(), rows%, cols%)
   lb
   pl "INPUT MODES "
   pl "  1 → READ from hardcoded data"
   pl "  Any → INPUT MANUALLY"
   lb
   mode% = in_int%("RC> Choose Input Mode for [" + matrix_name$ + "] " + format_order$(rows%, cols%) + " : ")
   If mode% = 1 Then
      Call read_matrix(mat#(), rows%, cols%)
   Else
      Call input_matrix(matrix_name$, mat#(), rows%, cols%)
   End If
End Sub

' TODO: Change Smaple Data

Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,-5,2,-4,2,1,3,4,7,2,1,4,2,5,2,5
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,2,5,2,4,6,2,6,2,6,2,6,2,-3.24,12
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,6,1,-4,9,0.5,2,5,2,8,4,66,2,8,3,6
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,3,6,6,2,6,9,0,1,6,2,6,-1,4,2,5,2
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,3,5,-2,0.6,2,0.8,4,4,8,,5,2,12,3,5
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,1,4,6,3,7,-6,2,9,4,9,4,7,-1,4,2,5,2
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,-5,2,-4,2,1,3,4,7,2,1,4,2,5,2,5
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,2,5,2,4,6,2,6,2,6,2,6,2,-3.24,12
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,6,1,-4,9,0.5,2,5,2,8,4,66,2,8,3,6
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,3,6,6,2,6,9,0,1,6,2,6,-1,4,2,5,2
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,3,5,-2,0.6,2,0.8,4,4,8,,5,2,12,3,5
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,1,4,6,3,7,-6,2,9,4,9,4,7,-1,4,2,5,2
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,-5,2,-4,2,1,3,4,7,2,1,4,2,5,2,5
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,2,5,2,4,6,2,6,2,6,2,6,2,-3.24,12
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,6,1,-4,9,0.5,2,5,2,8,4,66,2,8,3,6
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,3,6,6,2,6,9,0,1,6,2,6,-1,4,2,5,2
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,3,5,-2,0.6,2,0.8,4,4,8,,5,2,12,3,5
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,1,4,6,3,7,-6,2,9,4,9,4,7,-1,4,2,5,2
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,-5,2,-4,2,1,3,4,7,2,1,4,2,5,2,5
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,2,5,2,4,6,2,6,2,6,2,6,2,-3.24,12
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,6,1,-4,9,0.5,2,5,2,8,4,66,2,8,3,6
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,3,6,6,2,6,9,0,1,6,2,6,-1,4,2,5,2
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,3,5,-2,0.6,2,0.8,4,4,8,,5,2,12,3,5
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,1,4,6,3,7,-6,2,9,4,9,4,7,-1,4,2,5,2
Data 1,3,4,2,23,12,12,2,12,12,3,5,-3.24,12,6,3,2,2,4,2,3,23,5,4,5,2,12,3,5,1,7,2,9,2,5,-5,2,-4,2,1,3,4,5,3,6,6,2,6,9,0,1,6,2,6,-1,4,2,5,2
Data 5,2,4,2,54,2,6,4,6,2,5,2,4,66,2,8,3,6,-1,4,6,3,7,-6,2,9,4,9,4,7,3,0,2,5,2,6,3,7,3,7,3,7,3,8,3,-5,3,5,-2,0.6,2,0.8,4,4,8,,5,2,12,3,5
Data 7,2,1,6,2,6,3,6,-3,2,6,2,8,3,9,3,6,-2,5,9,2,3,6,-1,4,2,5,2,6,3,6,2,7,3,8,3,8,3,7,2,8,3,8,2,5,8,2,6,1,4,6,3,7,-6,2,9,4,9,4,7,-1,4,2,5,2
```