```
# Final exam programming prep:
#     1- write a subprogram: set_element. this subprogram will receive as argument double matrix base address,
#        dimension and row and column index and also a new value to be place into the matrix. if row and column
#         index are out of range, then print an error message and do not modify the matrix. everything should be
#         done in column major format.
#
#         $sp+0   Holds array base address (IN)
#         $sp+4   Holds array base height (IN)
#         $sp+8   Holds array base width (IN)
#         $sp+12  Holds row index (IN)
#         $sp+16  Holds column index (IN)
#         $sp+20  new matrix element value (IN)
#
#     2- write a subprogram: create_identity_matrix. this subprogram will receive as argument IN some value N. ther
#        it will allocate a two dimensional integer matrix (N x N). then it will fill the matrix with 1's if i == j
#        and 0's if i != j. for example:
#
#        N = 2, then identity matrix:
#             [1 0]
#             [0 1]
#
#         $sp+0   Holds number N (IN)
#         $sp+4   Holds base address of identity matrix (OUT)
#         $sp+8   Holds array base height of identity matrix (OUT)
#         $sp+12  Holds array base width of identity matrix(OUT)
#
# Final exam programming prep solution:
        .data
set_element_invalid_index_p:    .asciiz     "Invalid index in set_element subprogram\n"
############################################################
        .text
set_element:
# save arguments so we do not lose them
    lw $t0, 0($sp)          # load array base address
    lw $t1, 4($sp)          # load array height
    lw $t2, 8($sp)          # load array width
    lw $t3, 12($sp)         # load row index
    lw $t4, 16($sp)         # load column index
    l.d $f4, 20($sp)        # load new matrix value

    bge $t3, $t1, set_element_invalid_index # index is invalid if row index is greater than or equal to height
    bge $t4, $t2, set_element_invalid_index # index is invalid if column index is greater than or equal to width

set_element_valid:
    mul $t5, $t4, $t1       # $t5 <-- e * k
    add $t5, $t5, $t3       # $t5 <-- e * k + n'
    sll $t5, $t5, 3         # $t5 <-- s * (e * k + n')
    add $t5, $t0, $t5       # $t5 <-- b + s * (e * k + n') = i
    s.d $f4, 0($t5)         # load array element at given address into register $f4

    b set_element_end       # skip printing error message

set_element_invalid_index:
    li $v0, 4               # print error message
    la $a0, set_element_invalid_index_p
    syscall

set_element_end:
    jr $ra                  # jump back to the main

############################################################
############################################################

        .text
create_identity_matrix:
# save arguments so we do not lose them
    lw $t0, 0($sp)          # load number N

# allocate space for the transposed matrix
    mul $a0, $t0, $t0       # $a0 <-- height * width    || note that we are creating a square matrix
    sll $a0, $a0, 2         # $a0 <-- 4 * (height * width)
    li $v0, 9
    syscall                 # allocate matrix using system call 9

    move $t9, $v0           # store the address of transposed matrix into register $t9
    sw $v0, 4($sp)          # store base address of transposed matrix for return

    sw $t0, 8($sp)          # store height of transposed matrix for return
    sw $t0, 12($sp)         # store width of transposed matrix for return


    li $t1, 0               # initialize outer-loop counter to 0
create_identity_matrix_loop_outer:
    bge $t1, $t0, create_identity_matrix_loop_outer_end
```

```
    li $t2, 0               # initialize inner-loop counter to 0
create_identity_matrix_loop_inner:
    bge $t2, $t0, create_identity_matrix_loop_inner_end

# address calculation for original matrix
    mul $t3, $t2, $t0       # $t3 <-- e * k
    add $t3, $t3, $t1       # $t3 <-- e * k + n'
    sll $t3, $t3, 2         # $t3 <-- s * (e * k + n')
    add $t3, $v0, $t3       # $t3 <-- b + s * (e * k + n') = i

# check if i == j
    beq $t1, $t2, create_identity_matrix_loop_identity_diagonal     # check if i index == j index

create_identity_matrix_loop_identity_nondiagonal:
    li $t9, 0               # non-diagonal element, thus: $t9 <-- 0

    b create_identity_matrix_loop_identity_end  # skip else part of IF statement

create_identity_matrix_loop_identity_diagonal:
    li $t9, 1               # diagonal element, thus: $t9 <-- 1

create_identity_matrix_loop_identity_end:
    sw $t9, 0($t3)          # memory[$t3 + 0] <-- $t9

    addi $t2, $t2, 1        # increment inner-loop counter

    b create_identity_matrix_loop_inner # branch unconditionally to beginning of inner-loop

create_identity_matrix_loop_inner_end:
    addi $t1, $t1, 1        # increment outer-loop counter

    b create_identity_matrix_loop_outer # branch unconditionally to beginning of outer-loop

create_identity_matrix_loop_outer_end:

create_identity_matrix_end:
    jr $ra                  # jump back to the main
```