1 Carrayz 1D

1.1 The Ring 1

1.1.a rotate_ring.m

There are a couple ways to solve this problem. You can write a loop to move things one step in clockwise and counter clockwise directions, and then repeat this single step action within a for loop as many times as it is required. The array contains n elements.

```matlab
function ring = rotate_ring(ring, s)
    n = length(ring);
    if s > 0
        for j = 1:s
            last = ring(end);
            for k = n:-1:-1
                ring(k+1) = ring(k);
            end
            ring(1) = last;
        end
    elseif s < 0
        for j = 1:abs(s)
            first = ring(1);
            for k = 2:n
                ring(k-1) = ring(k);
            end
            ring(n) = first;
        end
    end
end
```

We can improve this code in many ways. For example abs(s) can be very large, but the end state will be any n of the possible states. Any multiple of a full turn will have no effect. It is better if we used the remainder rem function. This will effectively convert s into a number in the range -n+1:n-1. We can add the following line.

```matlab
s = rem(s, n);
```

Since there is no difference between rotating the ring s steps in clockwise direction and n – s steps in counter clockwise direction, we can pick any one of these actions. Let’s convert the counterclockwise rotations to clockwise rotations, and keep only the loop for positive steps. mod function does exactly what we need. Notice the difference between rem and mod functions! mod will map s into the range 0:n-1.
It looks shorter, but moving elements one step at a time is not necessary. We can think of the result consisting of two blocks. Think of a separator which marks the end of the array. You can think of the separator as an extra element which should be moved together with other elements. The end state of the ring will have two parts separated by this imaginary separator. Using this idea we can write the function as follows:

```matlab
function ring = rotate_ring(ring, s)
    s = mod(s, length(ring));
    ring = [ring(end-s+1:end) ring(1:end-s)];
end
```

1.1.b \texttt{circshift}

The MATLAB function we can use is called \texttt{circshift}. \texttt{circshift} works similarly on column vectors. Since our rings are defined with row vectors we have to be careful when we want to compare our results.

1.1.c \texttt{test_rotate_ring.m}

Let’s use \texttt{circshift} to check whether our function works as expected. Notice the operator ‘ used twice to convert an array with size $1 \times n$ to $n \times 1$ and vice versa. We used \texttt{all} function to make sure all elements in these arrays are elementwise equal. Similar to \texttt{all} function there is also \texttt{any} which you might find useful elsewhere. For arrays you can also use the function \texttt{isequal} which is a shorthand for what we do here.

```matlab
%% TEST1
% The trivial test is to use 0 steps
ring = 1:12;
result = rotate_ring(ring, 0);
expected = circshift(ring',0)';
assert(all(result == expected));

%% TEST2
```
% We should test for clockwise rotations
ring = 1:12;
result = rotate(ring,5);
expected = circshift(ring,5)';
assert(all(result == expected));

%% TEST3
% We should also test for counter clockwise rotations
ring = 1:12;
result = rotate(ring,-8);
expected = circshift(ring,-8)';
assert(all(result == expected));

%% TEST4
% It should work for multiple turns around clockwise
ring = 1:12;
result = rotate(ring,101);
expected = circshift(ring,101)';
assert(all(result == expected));

%% TEST5
% It should work for multiple turns around counter clockwise
ring = 1:12;
result = rotate(ring,-1729);
expected = circshift(ring,-1729)';
assert(all(result == expected));

1.2 The Ring 2

1.2.a hungarian.m

We can use our rotate function to change only one of the rows in the matrix. After rotation, we have to update the intersections on the other ring. Notice the function num2str which converts numbers to strings. You might find this function useful in the next homework.

```matlab
function rings = hungarian(rings, r, s)
    rings(r,:) = rotate(ring(rings(r,:),s));
    if r == 1
        rings(2,7) = rings(1,5);    rings(2,11) = rings(1,1);
    elseif r == 2
        rings(1,5) = rings(2,7);    rings(1,1) = rings(2,11);
    else
        error(['There is no ring number ' num2str(r)]);
    end
```
1.2.b  alternating rings

Any function which alternate between values 1 and 2 could be used. Or we can keep a temporary variable to remind us which ring we’d used previously.

```matlab
function rings = hungarian(rings, initial, steps)
    n = length(steps);
    r = initial;
    for s = steps
        rings(r,:) = rotate_ring(rings(r,:),s);
        if r == 1
            rings(2,7) = rings(1,5);
            rings(2,11) = rings(1,1);
        else
            rings(1,5) = rings(2,7);
            rings(1,1) = rings(2,11);
        end
        r = 3 - r;
    end
```

1.3  The Ring 3

1.3.a  celebrimbor.m

This problem can be solved just like the previous problem.

```matlab
function rings = celebrimbor(rings, r, s)
    rings(r,:) = rotate_ring(rings(r,:),s);
    if r == 1
        rings(2,7) = rings(1,5);
        rings(2,11) = rings(1,1);
        rings(3,1) = rings(1,3);
        rings(3,9) = rings(1,7);
    elseif r == 2
        rings(1,5) = rings(2,7);
        rings(1,1) = rings(2,11);
        rings(3,3) = rings(2,5);
        rings(3,11) = rings(2,9);
    elseif r == 3
        rings(1,3) = rings(3,1);
        rings(1,7) = rings(3,9);
        rings(2,5) = rings(3,3);
        rings(2,9) = rings(3,11);
    else
        error(['There is no ring ' num2str(r)]);
    end
```
1.3.b successive rotations

```matlab
function rings = celebritomor(rings, which, steps)

n = length(steps);
assert(n == length(which));

j = 0;
for s = steps
    j = j + 1;
    r = which(j);
    rings(r,:) = rotate(ring(rings(r,:),s));
    if r == 1
        rings(2, 7) = rings(1,5);
        rings(2,11) = rings(1,1);
        rings(3, 1) = rings(1,3);
        rings(3, 9) = rings(1,7);
    elseif r == 2
        rings(1, 5) = rings(2, 7);
        rings(1, 1) = rings(2,11);
        rings(3, 3) = rings(2, 5);
        rings(3,11) = rings(2, 9);
    elseif r == 3
        rings(1, 3) = rings(3, 1);
        rings(1, 7) = rings(3, 9);
        rings(2, 5) = rings(3, 3);
        rings(2, 9) = rings(3,11);
    else
        error(['There is no ring ' num2str(r)]);
    end
end
end
```