```
julia> L=[1 0 0; 8 1 0; 9 10 1]
3×3 Matrix{Int64}:
 1   0  0
 8   1  0
 9  10  1
```
ℓ_1 ℓ_2 ℓ_3

```
julia> U=[2 3 4; 0 5 6; 0 0 7]
3×3 Matrix{Int64}:
 2  3  4   u1ᵀ
 0  5  6   u2ᵀ
 0  0  7   u3ᵀ
```

```
julia> A=L*U
3×3 Matrix{Int64}:
  2   3    4
 16  29   38
 18  77  103
```

goal to factor that back into L and U.

Outer product formula for matrix matrix multiplication

$$A = \sum_{k=1}^{3} \ell_k u_k^T$$

First row of A is $e_1^T A$ $\qquad e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

length of vectors

```
julia> e(n)=1:3 .== n
e (generic function with 1 method)

julia> e(1)
3-element BitVector:
 1
 0
 0

julia> e(2)
3-element BitVector:
 0
 1
 0
```

standard basis vectors

```
julia> e(1)'*A
1×3 adjoint(::Vector{Int64}) with eltype Int64:
 2  3  4
```

first row of A

$$e_1^T \ell_k = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \ell_k = \begin{cases} 1 & \text{for } k=1 \\ 0 & \text{for } k \neq 1 \end{cases}$$

```
1   0  0
8   1  0
9  10  1
ℓ1 ℓ2 ℓ3
```

$$e_1^T A = e_1^T \sum_{k=1}^{3} \ell_k u_k^T = \sum_{k=1}^{3} e_1^T \ell_k u_k^T = u_1^T$$

first row

$$A e_1 = \sum_{k=1}^{3} \ell_k u_k^T e_1 = U_{11} \ell_1 \qquad \ell_1 = \frac{A e_1}{U_{11}}$$

first col of A

first col of L

$$u_k^T e_1 = u_{k,} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{cases} 2 = U_{11} & k=1 \\ 0 & k \neq 1 \end{cases}$$

```
2  3  4   u1ᵀ
0  5  6   u2ᵀ
0  0  7   u3ᵀ
```

```
julia> A*e(1)
3-element Vector{Int64}:
  2
 16
 18
```

first column of A

```
julia> u1=(e(1)'*A)'
3-element Vector{Int64}:
 2
 3
 4

julia> l1=(A*e(1))/u1[1]
3-element Vector{Float64}:
 1.0
 8.0
 9.0
```

know this

know this

$$A = \sum_{k=1}^{3} l_k u_k^T = l_1 u_1^T + l_2 u_2^T + l_3 u_3^T$$

$$l_2 u_2^T + l_3 u_3^T = A - l_1 u_1^T$$

```
julia> A-l1*u1'
3×3 Matrix{Float64}:
 0.0   0.0   0.0
 0.0   5.0   6.0
 0.0  50.0  67.0
```

$A_2 = A - l_1 u_1^T$ work on the 2×2 sub matrix.

$$e_2^T A_2 = e_2^T \left(l_2 u_2^T + l_3 u_3^T \right) = e_2^T l_2 u_2^T + e_2^T l_3 u_3^T = u_2^T$$

2nd row

```
julia> A=L*U
3×3 Matrix{Int64}:
  2   3    4
 16  29   38
 18  77  103
```

not 2nd row of original but of new matrix

$$e_2^T l_k = \begin{cases} 1 & \text{if } k=2 \\ 0 & \text{if } k>2 \end{cases}$$

```
1   0   0
8   1   0
9  10   1
```

only knocks on diagonal

$$u_2 = \left(e_2^T A_2 \right)^T$$

```
julia> U=[2 3 4; 0 5 6; 0 0 7]
3×3 Matrix{Int64}:
 2  3  4   u1ᵀ
 0  5  6   u2ᵀ
 0  0  7   u3ᵀ
```

← correct rows of U.

```
julia> A2=A-l1*u1'
3×3 Matrix{Float64}:
 0.0   0.0   0.0
 0.0   5.0   6.0
 0.0  50.0  67.0

julia> u2=(e(2)'*A2)'
3-element Vector{Float64}:
 0.0
 5.0
 6.0
```

$$A_2 e_2 = \left(l_2 u_2^T + l_3 u_3^T\right) e_2 = l_2 u_2^T e_2 + l_3 u_3^T e_3 = U_{22} l_2$$

$$u_k^T e_2 = \begin{cases} U_{22} e_2 & \text{for } k=2 \\ 0 & \text{for } k>2 \end{cases}$$

$$l_2 = \frac{A_2 e_2}{U_{22}}$$

```
julia> l2=A2*e(2)/u2[2]
3-element Vector{Float64}:
  0.0
  1.0
 10.0
```

know know know ?

$$A = \sum_{k=1}^{3} l_k u_k^T = l_1 u_1^T + l_2 u_2^T + l_3 u_3^T$$

$$A_3 = A - \left(l_1 u_1^T + l_2 u_2^T\right)$$

```
julia> A-l1*u1'-l2*u2'
3×3 Matrix{Float64}:
 0.0  0.0  0.0
 0.0  0.0  0.0
 0.0  0.0  7.0   u3^T
```

$$l_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad u_3 = \begin{bmatrix} 0 \\ 0 \\ 7 \end{bmatrix}$$

Same...

```
julia> L=[1 0 0; 8 1 0; 9 10 1]
3×3 Matrix{Int64}:
 1   0  0
 8   1  0
 9  10  1

julia> U=[2 3 4; 0 5 6; 0 0 7]
3×3 Matrix{Int64}:
 2  3  4
 0  5  6
 0  0  7
```

now express this algorithm in code...

```
function mylu(A)
    m,n=size(A)
    if m!=n
        println("Need a square matrix!")
        throw(exit())
    end
    U=zeros(size(A))
    L=zeros(size(A))
    Ak=copy(A)
    for k=1:n
        U[k,:]=Ak[k,:]
        L[:,k]=Ak[:,k]/U[k,k]
        Ak=Ak-L[:,k]*U[k,:]'
    end
    return L,U
end
```

```
julia> Lnew,Unew=mylu(A)
([1.0 0.0 0.0; 8.0 1.0 0.0; 9.
0])

julia> Lnew
3×3 Matrix{Float64}:
 1.0   0.0  0.0
 8.0   1.0  0.0
 9.0  10.0  1.0

julia> Unew
3×3 Matrix{Float64}:
 2.0  3.0  4.0
 0.0  5.0  6.0
 0.0  0.0  7.0
```

cut off output here.

worked!

```
julia> A=rand(4,4)
4×4 Matrix{Float64}:
 0.593945    0.636468   0.17312   0.137415
 0.00251978  0.541702   0.234918  0.00348068
 0.254154    0.0777767  0.660914  0.380075
 0.885738    0.839848   0.872451  0.935958
```

random matrix may not even have LU fact.

```
julia> L,U=mylu(A);
4×4 Matrix{Float64}:
 0.0   0.0       0.0       0.0
 0.0   0.539002  0.234184  0.00289771
 0.0  -0.194573  0.586835  0.321274
 0.0  -0.109303  0.614281  0.731034
4×4 Matrix{Float64}:
 0.0   0.0           0.0       0.0
 0.0   0.0           0.0       0.0
 0.0  -2.77556e-17  0.671372  0.322321
 0.0   0.0           0.661771  0.731622
4×4 Matrix{Float64}:
 0.0  0.0           0.0  0.0
 0.0  0.0           0.0  0.0
 0.0  0.0           0.0  0.0
 0.0  2.73586e-17   0.0  0.413911
```

```
julia> L
4×4 Matrix{Float64}:
 1.0          0.0       0.0       0.0
 0.00424245   1.0       0.0       0.0
 0.427908    -0.360988  1.0       0.0
 1.49128     -0.202789  0.985699  1.0

julia> U
4×4 Matrix{Float64}:
 0.593945   0.636468     0.17312   0.137415
 0.0        0.539002     0.234184  0.00289771
 0.0       -2.77556e-17  0.671372  0.322321
 0.0        2.73586e-17  0.0       0.413911
```

```
julia> norm(A-L*U)
1.5762137000608 56e-16
```

← error...

Next time compare with built-in LU factorization...