1. Crie a função recursiva `inversa(L,I)` que, dada lista `L` e uma lista `I` (inicialmente *vazia*), constrói e devolve a lista inversa de `L`. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = inversa([1,2,3,4],[])
    = inversa([2,3,4],no(1,[]))
    = inversa([2,3,4],[1])
      = inversa([3,4],no(2,[1]))
      = inversa([3,4],[2,1])
        = inversa([4],no(3,[2,1]))
        = inversa([4],[3,2,1])
           = inversa([],no(4,[3,2,1]))
           = inversa([],[4,3,2,1])
           = [4,3,2,1]
        = [4,3,2,1]
      = [4,3,2,1]
    = [4,3,2,1]
  = [4,3,2,1]
```

Observação: *Note que a função deve usar recursão de cauda, acumulando os nós criados no parâmetro* `I`.

2. Crie a função recursiva `rnd(n)`, que constrói e devolve uma lista com `n` itens aleatórios. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = rnd(4)
    = no(7?,rnd(3))
      = no(7?,no(9?,rnd(2)))
        = no(7?,no(9?,no(5?,rnd(1))))
          = no(7?,no(9?,no(5?,no(8?,rnd(0)))))
          = no(7?,no(9?,no(5?,no(8?,NULL))))
          = no(7?,no(9?,no(5?,no(8?,[]))))
        = no(7?,no(9?,no(5?,[8?])))
      = no(7?,no(9?,[5?,8?]))
    = no(7?,[9?,5?,8?])
  = [7?,9?,5?,8?]
```

Observações: *(a) o sinal '?' indica 'incerteza'; (b) suponha que* `n` *é maior ou igual a* `0`*; (c) use a expressão* `rand()%10` *para gerar um número aleatório entre* `0` *e* `9`*; e (d), a função* `rand()` *está declarada em* `stdlib.h`.

3. Crie a função recursiva `progressao(p,u)`, que constrói e devolve uma lista com uma sequência progressiva de itens, iniciando em `p` e aumentando até `u`. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = progressao(2,5)
    = no(2,progressao(3,5))
      = no(2,no(3,progressao(4,5)))
        = no(2,no(3,no(4,progressao(5,5))))
          = no(2,no(3,no(4,no(5,progressao(6,5)))))
          = no(2,no(3,no(4,no(5,NULL))))
          = no(2,no(3,no(4,no(5,[]))))
        = no(2,no(3,no(4,[5])))
      = no(2,no(3,[4,5]))
    = no(2,[3,4,5])
  = [2,3,4,5])
```

4. Crie a função recursiva `regressao(p,u)`, que constrói e devolve uma lista com uma sequência regressiva de itens, iniciando em `p` e diminuindo até `u`. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = regressao(4,1)
    = no(4,regressao(3,1))
      = no(4,no(3,regressao(2,1)))
        = no(4,no(3,no(2,regressao(1,1))))
          = no(4,no(3,no(2,no(1,regressao(0,1)))))
          = no(4,no(3,no(2,no(1,NULL))))
          = no(4,no(3,no(2,no(1,[]))))
        = no(4,no(3,no(2,[1])))
      = no(4,no(3,[2,1]))
    = no(4,[3,1])
  = [4,3,2,1]
```

5. Crie a função recursiva `progressao_passo(p,u,k)`, que constrói e devolve uma lista com uma sequência progressiva de itens, iniciando em `p` e aumentando até no máximo `u`, de `k` em `k`. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = progressao_passo(0,7,2)
    = no(0,progressao_passo(2,7,2))
      = no(0,no(2,progressao_passo(4,7,2)))
        = no(0,no(2,no(4,progressao_passo(6,7,2))))
          = no(0,no(2,no(4,no(6,progressao_passo(8,7,2)))))
          = no(0,no(2,no(4,no(6,NULL))))
          = no(0,no(2,no(4,no(6,[]))))
        = no(0,no(2,no(4,[6])))
      = no(0,no(2,[4,6]))
    = no(0,[2,4,6])
  = [0,2,4,6]
```

6. Crie a função recursiva `regressao_passo(p,u,k)`, que constrói e devolve uma lista com uma sequência regressiva de itens, iniciando em `p` e diminuindo até no mínimo `u`, de `k` em `k`. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = regressao_passo(10,0,-3)
    = no(10,regressao_passo(7,0,-3))
      = no(10,no(7,regressao_passo(4,0,-3)))
        = no(10,no(7,no(4,regressao_passo(1,0,-3))))
          = no(10,no(7,no(4,no(1,regressao_passo(-2,0,-3)))))
          = no(10,no(7,no(4,no(1,NULL))))
          = no(10,no(7,no(4,no(1,[]))))
        = no(10,no(7,no(4,[1])))
      = no(10,no(7,[4,1]))
    = no(10,[7,4,1])
  = [10,7,4,1]
```

7. Crie a função recursiva `unico(L)`, que constrói e devolve uma lista contendo todos os itens de `L`, sem repetição. A função deve funcionar como exemplificado pela simulação a seguir:

```
R = unico([2,3,2,1,3,1,3])
    = unico([3,2,1,3,1,3])
      = unico([2,1,3,1,3])
        = unico([1,3,1,3])
          = unico([3,1,3])
            = unico([1,3])
              = unico([3])
                = unico([])
                = NULL
                = []
              = no(3,[])      {adiciona 3, pois 3∉[]}
              = [3]
            = no(1,[3])       {adiciona 1, pois 1∉[3]}
            = [1,3]
          = [1,3]             {não adiciona 3, pois 1∈[1,3]}
        = [1,3]               {não adiciona 1, pois 1∈[1,3]}
      = no(2,[1,3])           {adiciona 2, pois 2∉[1,3]}
      = [2,1,3]
    = [2,1,3]                 {não adiciona 3, pois 3∈[2,1,3]}
  = [2,1,3]                   {não adiciona 2, pois 2∈[2,1,3]}
```

8. Crie a função recursiva `ordenada(L)`, que devolve `1` se a lista `L` estiver ordenada de forma crescente, podendo ter itens repetidos (ou `0`, caso contrário). A função deve funcionar como exemplificado pela simulação a seguir:

```
R = ordenada([])
  = 1                         {pois a lista vazia é ordenada}

R = ordenada([1,4,9,9])
    = ordenada([4,9,9])       {pois 1≤4}
      = ordenada([9,9])       {pois 4≤9}
        = ordenada([9])       {pois 9≤9}
        = 1                   {pois toda lista unitária é ordenada}
      = 1
    = 1
  = 1

R = ordenada([1,4,5,8,7,9])
    = ordenada([4,5,8,7,9])   {pois 1≤4}
      = ordenada([5,8,7,9])   {pois 4≤5}
        = ordenada([8,7,9])   {pois 5≤8}
        = 0                   {pois 8>7}
      = 0
    = 0
 = 0
```