```
# overwrite similar from indices, because similar on `Diagonal` tends to create
# `SparseArray` for some horrible reason

function similar_from_indices(T::Type, ind::IndexTuple, A::Diagonal, CA::Symbol)
    sz = similarstructure_from_indices(T, ind, A, CA)
    return similar(A.diag, T, sz)
end
function similar_from_indices(T::Type, poA::IndexTuple, poB::IndexTuple,
                              p1::IndexTuple, p2::IndexTuple,
                              A::Diagonal, B::AbstractArray, CA::Symbol,
CB::Symbol)
    sz = similarstructure_from_indices(T, poA, poB, p1, p2, A, B, CA, CB)
    similar(A.diag, T, sz)
end
function similar_from_indices(T::Type, poA::IndexTuple, poB::IndexTuple,
                              p1::IndexTuple, p2::IndexTuple,
                              A::Diagonal, B::Diagonal, CA::Symbol, CB::Symbol)
    sz = similarstructure_from_indices(T, poA, poB, p1, p2, A, B, CA, CB)
    similar(A.diag, T, sz)
end

function contract!(α, A::AbstractArray, CA::Symbol, B::Diagonal, CB::Symbol,
        β, C::AbstractArray,
        oindA::IndexTuple, cindA::IndexTuple, oindB::IndexTuple, cindB::IndexTuple,
        indCinoAB::IndexTuple, syms::Union{Nothing, NTuple{3,Symbol}} = nothing)

    pA = (oindA...,cindA...)
    (length(pA) == ndims(A) && TupleTools.isperm(pA)) ||
        throw(IndexError("invalid permutation of length $(ndims(A)): $pA"))
    pB = (oindB...,cindB...)
    (length(pB) == ndims(B) && TupleTools.isperm(pB)) ||
        throw(IndexError("invalid permutation of length $(ndims(B)): $pB"))
    (length(oindA) + length(oindB) == ndims(C)) ||
        throw(IndexError("non-matching output indices in contraction"))
    (ndims(C) == length(indCinoAB) && isperm(indCinoAB)) ||
        throw(IndexError("invalid permutation of length $(ndims(C)): $indCinoAB"))

    sizeA = i->size(A, i)
    sizeB = i->size(B, i)
    sizeC = i->size(C, i)

    csizeA = sizeA.(cindA)
    csizeB = sizeB.(cindB)
    osizeA = sizeA.(oindA)
    osizeB = sizeB.(oindB)

    csizeA == csizeB ||
        throw(DimensionMismatch("non-matching sizes in contracted dimensions"))
    sizeAB = let osize = (osizeA..., osizeB...)
        i->osize[i]
    end
    sizeAB.(indCinoAB) == size(C) ||
        throw(DimensionMismatch("non-matching sizes in uncontracted dimensions"))
```

```
    Bd = B.diag
    if CA == :N && CB == :N
        @unsafe_strided A Bd C begin
            _contract!(α, A, Bd, β, C, oindA, cindA, oindB, cindB, indCinoAB)
        end
    elseif CA == :C && CB == :N
        @unsafe_strided A Bd C begin
            _contract!(α, conj(A), Bd, β, C, oindA, cindA, oindB, cindB, indCinoAB)
        end
    elseif CA == :N && CB == :C
        @unsafe_strided A Bd C begin
            _contract!(α, A, conj(Bd), β, C, oindA, cindA, oindB, cindB, indCinoAB)
        end
    elseif CA == :C && CB == :C
        @unsafe_strided A Bd C begin
            _contract!(α, conj(A), conj(Bd), β, C, oindA, cindA, oindB, cindB,
indCinoAB)
        end
    else
        throw(ArgumentError("unknown conjugation flag $CA and $CB"))
    end
    return C
end
function _contract!(α, A::UnsafeStridedView, Bd::UnsafeStridedView,
        β, C::UnsafeStridedView, oindA, cindA, oindB, cindB, indCinoAB)

    sizeA = i->size(A, i)
    csizeA = sizeA.(cindA)
    osizeA = sizeA.(oindA)

    if length(oindB) == 1 # length(cindA) == length(cindB) == 1
        A2 = permutedims(A, (oindA..., cindA...))
        C2 = permutedims(C, TupleTools.invperm(indCinoAB))
        B2 = sreshape(Bd, ((one.(osizeA))..., csizeA...))
        totsize = (osizeA..., csizeA...)
        if α != 1
            if β == 0
                Strided._mapreducedim!((x,y)->α*x*y, +, zero, totsize, (C2, A2, B2))
            elseif β == 1
                Strided._mapreducedim!((x,y)->α*x*y, +, nothing, totsize, (C2, A2,
B2))
            else
                Strided._mapreducedim!((x,y)->α*x*y, +, y->β*y, totsize, (C2, A2,
B2))
            end
        else
            if β == 0
                return Strided._mapreducedim!(*, +, zero, totsize, (C2, A2, B2))
            elseif β == 1
                Strided._mapreducedim!(*, +, nothing, totsize, (C2, A2, B2))
            else
                Strided._mapreducedim!(*, +, y->β*y, totsize, (C2, A2, B2))
            end
        end
    elseif length(oindB) == 0
```

```
        strideA = i->stride(A, i)
        newstrides = (strideA.(oindA)..., strideA(cindA[1]) + strideA(cindA[2]))
        totsize = (osizeA..., csizeA[1])
        A2 = UnsafeStridedView(A.ptr, totsize, newstrides, A.offset, A.op)
        B2 = sreshape(Bd, ((one.(osizeA))..., csizeA[1]))
        C2 = permutedims(C, TupleTools.invperm(indCinoAB))

        if α != 1
            if β == 0
                Strided._mapreducedim!((x,y)->α*x*y, +, zero, totsize, (C2, A2, B2))
            elseif β == 1
                Strided._mapreducedim!((x,y)->α*x*y, +, nothing, totsize, (C2, A2,
B2))
            else
                Strided._mapreducedim!((x,y)->α*x*y, +, y->β*y, totsize, (C2, A2,
B2))
            end
        else
            if β == 0
                return Strided._mapreducedim!(*, +, zero, totsize, (C2, A2, B2))
            elseif β == 1
                Strided._mapreducedim!(*, +, nothing, totsize, (C2, A2, B2))
            else
                Strided._mapreducedim!(*, +, y->β*y, totsize, (C2, A2, B2))
            end
        end
    else # length(oindB) == 2
        if β != 1
            rmul!(C, β)
        end
        A2 = sreshape(permutedims(A, (oindA..., cindA...)), (osizeA..., 1))
        C2 = permutedims(C, TupleTools.invperm(indCinoAB))
        B2 = sreshape(Bd, ((one.(osizeA))..., length(Bd)))

        sC = strides(C2)
        newstrides = (Base.front(Base.front(sC))..., sC[end-1] + sC[end])
        totsize = (osizeA..., length(Bd))

        C3 = UnsafeStridedView(C2.ptr, totsize, newstrides, C2.offset, C2.op)
        if α != 1
            Strided._mapreducedim!((x,y)->α*x*y, +, nothing, totsize, (C3, A2, B2))
        else
            Strided._mapreducedim!(*, +, nothing, totsize, (C3, A2, B2))
        end
    end
    return C
end

function contract!(α, A::Diagonal, CA::Symbol, B::AbstractArray, CB::Symbol,
        β, C::AbstractArray,
        oindA::IndexTuple, cindA::IndexTuple, oindB::IndexTuple, cindB::IndexTuple,
        indCinoAB::IndexTuple, syms::Union{Nothing, NTuple{3,Symbol}} = nothing)

    indCinoAB′ = map(i->i <= length(oindA) ? length(oindB)+i : i-length(oindA),
indCinoAB)
```

```
    contract!(α, B, CB, A, CA, β, C, oindB, cindB, oindA, cindA, indCinoAB′)
    return C
end
```