```
"""
    ncon(tensorlist, indexlist, [conjlist, sym]; order = ..., output = ...)

Contract the tensors in `tensorlist` (of type `Vector` or `Tuple`) according to the
network
as specified by `indexlist`. Here, `indexlist` is a list (i.e. a `Vector` or
`Tuple`) with
the same length as `tensorlist` whose entries are themselves lists (preferably
`Vector{Int}`) where every integer entry provides a label for corresponding
index/dimension
of the corresponding tensor in `tensorlist`. Positive integers are used to label
indices
that need to be contracted, and such thus appear in two different entries within
`indexlist`, whereas negative integers are used to label indices of the output
tensor, and
should appear only once.

Optional arguments in another list with the same length, `conjlist`, whose entries
are of
type `Bool` and indicate whether the corresponding tensor object should be
conjugated
(`true`) or not (`false`). Finally, a `Symbol` can be provided which provides a
hook into
the global LRU cache of TensorOperationsXD.jl to store temporaries. This symbol
should be
unique for every `ncon` call withing a running application.

By default, contractions are performed in the order such that the indices being
contracted
over are labelled by increasing integers, i.e. first the contraction corresponding
to label
`1` is performed. The output tensor had an index order corresponding to decreasing
(negative, so increasing in absolute value) index labels. The keyword arguments
`order` and
`output` allow to change these defaults.

See also the macro version [`@ncon`](@ref).
"""
function ncon(tensors, network,
                conjlist = fill(false, length(tensors)), sym = nothing ;
                order = nothing, output = nothing)
    length(tensors) >= 2 ||
        throw(ArgumentError("do not use `ncon` for less than two tensors"))
    length(tensors) == length(network) == length(conjlist) ||
        throw(ArgumentError("number of tensors and of index lists should be the
same"))
    isnconstyle(network) || throw(ArgumentError("invalid NCON network: $network"))
    outputindices = Vector{Int}()
    for n in network
        for k in n
            if k < 0
                push!(outputindices, k)
            end
        end
```

```
    end
    if output === nothing
        output = sort(outputindices; rev = true)
    else
        for a in output
            a in outputindices ||
                throw(ArgumentError("invalid NCON network: $network -> $output"))
        end
        for a in outputindices
            a in output ||
                throw(ArgumentError("invalid NCON network: $network -> $output"))
        end
    end
    (tensors,network) = resolve_traces(tensors,network);
    tree = order === nothing ? ncontree(network) : indexordertree(network, order)

    if sym !== nothing
        syma = Symbol(sym, "_a")
        symb = Symbol(sym, "_b")
    else
        syma = symb = nothing
    end
    A, IA, CA = contracttree(tensors, network, conjlist, tree[1], syma)
    B, IB, CB = contracttree(tensors, network, conjlist, tree[2], symb)
    IC = tuple(output...)

    oindA, cindA, oindB, cindB, indCinoAB = contract_indices(IA, IB, IC)
    T = promote_type(eltype(A), eltype(B))
    # end result: don't use cache
    C = similar_from_indices(T, oindA, oindB, indCinoAB, (), A, B, CA, CB)
    if sym !== nothing
        symcontract = (Symbol(sym, "_a′"), Symbol(sym, "_b′"), Symbol(sym, "_c′"))
    else
        symcontract = nothing
    end
    contract!(true, A, CA, B, CB, false, C,
                oindA, cindA, oindB, cindB, indCinoAB, (), symcontract)
    return C
end

function contracttree(tensors, network, conjlist, tree, sym)
    @nospecialize
    if tree isa Int
        return tensors[tree], tuple(network[tree]...), (conjlist[tree] ? :C : :N)
    end

    if sym !== nothing
        syma = Symbol(sym, "_a")
        symb = Symbol(sym, "_b")
    else
        syma = nothing
        symb = nothing
    end
    A, IA, CA = contracttree(tensors, network, conjlist, tree[1], syma)
    B, IB, CB = contracttree(tensors, network, conjlist, tree[2], symb)
```

```
    IC = tuple(symdiff(IA, IB)...)
    oindA, cindA, oindB, cindB, indCinoAB = contract_indices(IA, IB, IC)
    T = promote_type(eltype(A), eltype(B))
    if sym !== nothing
        symc = Symbol(sym, "_c")
        C = cached_similar_from_indices(symc, T, oindA, oindB, indCinoAB, (), A, B,
CA, CB)
    else
        C = similar_from_indices(T, oindA, oindB, indCinoAB, (), A, B, CA, CB)
    end
    if sym !== nothing
        symcontract = (Symbol(sym, "_a′"), Symbol(sym, "_b′"), Symbol(sym, "_c′"))
    else
        symcontract = nothing
    end
    contract!(true, A, CA, B, CB, false, C,
                oindA, cindA, oindB, cindB, indCinoAB, (), symcontract)
    return C, IC, :N
end
```