```julia
# methods/inplace.jl
#
# Method-based access to tensor operations using inplace definitions.
tensorcopy!(A, IA, C, IC) = tensorcopy!(A, tuple(IA...), C, tuple(IC...))
tensoradd!(α, A, IA, β, C, IC) = tensoradd!(α, A, tuple(IA...), β, C, tuple(IC...))
tensortrace!(α, A, IA, β, C, IC) = tensortrace!(α, A, tuple(IA...), β, C,
tuple(IC...))
tensorcontract!(α, A, IA, conjA, B, IB, conjB, β, C, IC) =
    tensorcontract!(α, A, tuple(IA...), conjA, B, tuple(IB...), conjB, β, C,
tuple(IC...))
tensorproduct!(α, A, IA, B, IB, β, C, IC) = tensorproduct!(α, A, tuple(IA...), B,
tuple(IB...), β, C, tuple(IC...))


"""
    tensorcopy!(A, IA, C, IC)

Copies `A` into `C` by permuting the dimensions according to the pattern specified
by
`IA` and `IC`. Both iterables should contain the same elements in a different order.
The result of this method is equivalent to `permutedims!(C, A, p)` where `p` is the
permutation such that `IC=IA[p]`. The implementation of tensorcopy! is however more
efficient on average, especially if `Threads.nthreads() > 1`.
"""
tensorcopy!(A, IA::Tuple, C, IC::Tuple) = tensoradd!(1, A, IA, 0, C, IC)

"""
    tensoradd!(α, A, IA, β, C, IC)

Updates `C` to `β*C + α * tensorcopy(A,IA,IC)`, but without creating the temporary
permuted array.

See also: [`tensorcopy`](@ref)
"""
function tensoradd!(α, A, IA::Tuple, β, C, IC::Tuple)
    indCinA = add_indices(IA, IC)
    add!(α, A, :N, β, C, indCinA)
end

"""
    tensortrace!(α, A, IA, β, C, IC)

Updates `C` to `β*C + α tensortrace(A,IA,IC)`, but without creating the temporary
traced array.

See also: [`tensortrace`](@ref)
"""
function tensortrace!(α, A, IA::Tuple, β, C, IC::Tuple)
    indCinA, cindA1, cindA2 = trace_indices(IA, IC)
    trace!(α, A, :N, β, C, indCinA, cindA1, cindA2)
    return C
end

"""
```

```
    tensorcontract!(α, A, labelsA, conjA, B, labelsB, conjB, β, C, labelsC)

Replaces `C` with `β C + α A * B`, where some indices of array `A` are contracted
with corresponding
indices in array `B` by assigning them identical labels in the iterables `labelsA`
and `labelsB`.
The arguments `conjA` and `conjB` should be of type `Char` and indicate whether the
data of
arrays `A` and `B`, respectively, need to be conjugated (value `'C'`) or not (value
`'N'`).
Every label should appear exactly twice in the union of `labelsA`, `labelsB` and
`labelsC`,
either in the intersection of `labelsA` and `labelsB` (for indices that need to be
contracted)
or in the interaction of either `labelsA` or `labelsB` with `labelsC`, for
indicating the order
in which the open indices should be match to the indices of the output array `C`.
"""
function tensorcontract!(α, A, IA::Tuple, conjA, B, IB::Tuple, conjB, β, C,
IC::Tuple)

    conjA == 'N' || conjA == 'C' || throw(ArgumentError("Value of conjA should be
'N' or 'C' instead of $conjA"))
    conjB == 'N' || conjB == 'C' || throw(ArgumentError("Value of conjB should be
'N' or 'C' instead of $conjB"))
    CA = conjA == 'N' ? :N : :C
    CB = conjB == 'N' ? :N : :C

    oindA, cindA, oindB, cindB, indCinoAB = contract_indices(IA, IB, IC)
    contract!(α, A, CA, B, CB, β, C, oindA, cindA, oindB, cindB, indCinoAB)
    return C
end

"""
    tensorproduct!(α, A, labelsA, B, labelsB, β, C, labelsC)

Replaces C with `β C + α A * B` without any indices being contracted.
"""
function tensorproduct!(α, A, IA::Tuple, B, IB::Tuple, β, C, IC::Tuple)
    isempty(intersect(IA, IB)) || throw(LabelError("not a valid tensor product"))
    tensorcontract!(α, A, IA, 'N', B, IB, 'N', β, C, IC)
end
```