```
# Fusion trees:
#==============================================================================#
"""
    struct FusionTree{G,N,M,L,T}

Represents a fusion tree of sectors of type `G<:Sector`, fusing (or splitting) `N`
uncoupled
sectors to a coupled sector. This fusion tree has `M=max(0,N-2)` inner lines.
Furthermore,
for `FusionStyle(G) isa DegenerateNonAbelian`, the `L=max(0,N-1)` corresponding
vertices
carry a label of type `T`. If `FusionStyle(G) isa Union{Abelian,SimpleNonAbelian}`,
`T = Nothing`.
"""
struct FusionTree{G<:Sector,N,M,L,T}
    uncoupled::NTuple{N,G}
    coupled::G
    isdual::NTuple{N,Bool}
    innerlines::NTuple{M,G} # M = N-2
    vertices::NTuple{L,T} # L = N-1
    function FusionTree{G,N,M,L,T}(uncoupled::NTuple{N,G},
                                    coupled::G,
                                    isdual::NTuple{N,Bool},
                                    innerlines::NTuple{M,G},
                                    vertices::NTuple{L,T}) where
                                    {G<:Sector,N,M,L,T}
        new{G,N,M,L,T}(uncoupled, coupled, isdual, innerlines, vertices)
    end
end
FusionTree{G}(uncoupled::NTuple{N,Any},
                coupled,
                isdual::NTuple{N,Bool},
                innerlines,
                vertices = ntuple(n->nothing, StaticLength(N)-StaticLength(1))
                ) where {G<:Sector,N} =
    fusiontreetype(G, StaticLength(N))(map(s->convert(G,s),uncoupled),
        convert(G,coupled), isdual, map(s->convert(G,s), innerlines), vertices)
FusionTree(uncoupled::NTuple{N,G},
            coupled::G,
            isdual::NTuple{N,Bool},
            innerlines,
            vertices = ntuple(n->nothing, StaticLength(N)-StaticLength(1))
            ) where {G<:Sector,N} =
    fusiontreetype(G, StaticLength(N))(uncoupled, coupled, isdual, innerlines,
vertices)

function FusionTree{G}(uncoupled::NTuple{N}, coupled = one(G),
                        isdual = ntuple(n->false, StaticLength(N))) where
{G<:Sector, N}
    FusionStyle(G) isa Abelian ||
        error("fusion tree requires inner lines if `FusionStyle(G) <: NonAbelian`")
    FusionTree{G}(map(s->convert(G,s), uncoupled), convert(G, coupled), isdual,
                _abelianinner(map(s->convert(G,s),(uncoupled...,
dual(coupled)))))
```

```julia
end
function FusionTree(uncoupled::NTuple{N,G}, coupled::G = one(G),
                            isdual = ntuple(n->false, StaticLength(N))) where
{G<:Sector, N}
    FusionStyle(G) isa Abelian ||
        error("fusion tree requires inner lines if `FusionStyle(G) <: NonAbelian`")
    FusionTree{G}(uncoupled, coupled, isdual, _abelianinner((uncoupled...,
dual(coupled))))
end

# Properties
sectortype(::Type{<:FusionTree{G}}) where {G<:Sector} = G
FusionStyle(::Type{<:FusionTree{G}}) where {G<:Sector} = FusionStyle(G)
BraidingStyle(::Type{<:FusionTree{G}}) where {G<:Sector} = BraidingStyle(G)
Base.length(::Type{<:FusionTree{<:Sector,N}}) where {N} = N

sectortype(f::FusionTree) = sectortype(typeof(f))
FusionStyle(f::FusionTree) = FusionStyle(typeof(f))
BraidingStyle(f::FusionTree) = BraidingStyle(typeof(f))
Base.length(f::FusionTree) = length(typeof(f))

# Hashing, important for using fusion trees as key in a dictionary
function Base.hash(f::FusionTree{G}, h::UInt) where {G}
    h = hash(f.isdual, hash(f.coupled, hash(f.uncoupled, h)))
    if FusionStyle(G) isa SimpleNonAbelian
        h = hash(f.innerlines, h)
    end
    if FusionStyle(G) isa DegenerateNonAbelian
        h = hash(f.vertices, h)
    end
    return h
end
function Base.isequal(f1::FusionTree{G,N}, f2::FusionTree{G,N}) where {G<:Sector,N}
    f1.coupled == f2.coupled || return false
    @inbounds for i = 1:N
        f1.uncoupled[i] == f2.uncoupled[i] || return false
        f1.isdual[i] == f2.isdual[i] || return false
    end
    if FusionStyle(G) isa SimpleNonAbelian
        @inbounds for i=1:N-2
            f1.innerlines[i] == f2.innerlines[i] || return false
        end
    end
    if FusionStyle(G) isa DegenerateNonAbelian
        @inbounds for i=1:N-1
            f1.vertices[i] == f2.vertices[i] || return false
        end
    end
    return true
end
Base.isequal(f1::FusionTree, f2::FusionTree) = false

# Facilitate getting correct fusion tree types
Base.@pure fusiontreetype(::Type{G}, ::StaticLength{0}) where {G<:Sector} =
```

```
    FusionTree{G, 0, 0, 0, vertex_labeltype(G)}
Base.@pure fusiontreetype(::Type{G}, ::StaticLength{1}) where {G<:Sector} =
    FusionTree{G, 1, 0, 0, vertex_labeltype(G)}
Base.@pure fusiontreetype(::Type{G}, ::StaticLength{2}) where {G<:Sector} =
    FusionTree{G, 2, 0, 1, vertex_labeltype(G)}
Base.@pure fusiontreetype(::Type{G}, ::StaticLength{N}) where {G<:Sector, N} =
    _fusiontreetype(G, StaticLength(N),
        StaticLength(N) - StaticLength(2), StaticLength(N) - StaticLength(1))
Base.@pure _fusiontreetype(::Type{G}, ::StaticLength{N}, ::StaticLength{M},
                            ::StaticLength{L}) where {G<:Sector, N, M, L} =
    FusionTree{G,N,M,L,vertex_labeltype(G)}

# converting to actual array
function Base.convert(::Type{Array}, f::FusionTree{G, 0}) where {G}
    T = eltype(fusiontensor(one(G), one(G), one(G)))
    return fill(one(T), 1)
end
function Base.convert(::Type{Array}, f::FusionTree{G, 1}) where {G}
    c = f.coupled
    dc = dim(c)
    if f.isdual[1]
        Zcbartranspose = sqrt(dc)*reshape(fusiontensor(conj(c), c, one(c)), (dc,
dc))
        return convert(Array, conj(Zcbartranspose))
    else
        convert(Array, reshape(fusiontensor(c, one(c), c), (dc, dc)))
    end
end

function Base.convert(::Type{Array}, f::FusionTree{G,2}) where {G}
    a, b = f.uncoupled
    isduala, isdualb = f.isdual
    c = f.coupled
    da, db, dc = dim.((a,b,c))
    μ = f.vertices[1]
    X = reshape(fusiontensor(a, b, c, μ), da*db, dc)
    Za = convert(Array, FusionTree((a,), a, (isduala,), ()))
    Zb = convert(Array, FusionTree((b,), b, (isdualb,), ()))
    return convert(Array, reshape(kron(Zb, Za)*X, (da, db, dc)))
end

function Base.convert(::Type{Array}, f::FusionTree{G}) where {G}
    tailout = (f.innerlines[1], TupleTools.tail2(f.uncoupled)...)
    isdualout = (false, TupleTools.tail2(f.isdual)...)
    ftail = FusionTree(tailout, f.coupled, isdualout,
                        Base.tail(f.innerlines), Base.tail(f.vertices))
    Ctail = convert(Array, ftail)
    f1 = FusionTree((f.uncoupled[1], f.uncoupled[2]), f.innerlines[1],
                    (f.isdual[1], f.isdual[2]), (), (f.vertices[1],))
    C1 = convert(Array, f1)
    dtail = size(Ctail)
    d1 = size(C1)
    C = reshape(C1, d1[1]*d1[2], d1[3]) *
        reshape(Ctail, dtail[1], prod(Base.tail(dtail)))
```

```
    return reshape(C, (d1[1], d1[2], Base.tail(dtail)...))
end

# Show methods
function Base.show(io::IO, t::FusionTree{G,N,M,K,Nothing}) where {G<:Sector,N,M,K}
    print(IOContext(io, :typeinfo => G), "FusionTree{", G, "}(",
        t.uncoupled, ", ", t.coupled, ", ", t.isdual, ", ", t.innerlines, ")")
end
function Base.show(io::IO, t::FusionTree{G}) where {G<:Sector}
    print(IOContext(io, :typeinfo => G), "FusionTree{", G, "}(",
        t.uncoupled, ", ", t.coupled, ", ", t.isdual, ",",
        t.innerlines, ", ", t.vertices, ")")
end

# Manipulate fusion trees
include("manipulations.jl")

# Fusion tree iterators
include("iterator.jl")

# auxiliary routines
# _abelianinner: generate the inner indices for given outer indices in the abelian
  case
_abelianinner(outer::Tuple{}) = ()
_abelianinner(outer::Tuple{G}) where {G<:Sector} =
    outer[1] == one(G) ? () : throw(SectorMismatch())
_abelianinner(outer::Tuple{G,G}) where {G<:Sector} =
    outer[1] == dual(outer[2]) ? () : throw(SectorMismatch())
_abelianinner(outer::Tuple{G,G,G}) where {G<:Sector} =
    first(⊗(outer...)) == one(G) ? () : throw(SectorMismatch())
function _abelianinner(outer::NTuple{N,G}) where {G<:Sector,N}
    c = first(outer[1] ⊗ outer[2])
    return (c, _abelianinner((c, TupleTools.tail2(outer)...))...)
end
```