Listing 1: Implementation of TGW algorithm

```julia
function build_CF( V::Matrix{Float64}, EV::ChainOp, FE::ChainOp)
  m,n,marks,FV = initialization(V, EV, FE)
  FC = mainloop(marks,n,V,EV,FE,FV) # building the [∂_3] matrix FC
  CF = convert(ChainOp, FC')  # boundary [∂_3 → δ_2] coboundary
  return CF # matrix of δ_2 : C_2 → C_3 linear operator between chain spaces
end
```

Listing 2: Initializations

```julia
function initialization(V, EV, FE)
  n,m = size(FE); marks = zeros(Int8,n);
  I,J,X = SparseArrays.findnz(FE * EV)  # δ_2 * δ_1 = FV ⊂ [0]
  FV = SparseArrays.sparse(I,J,ones(Int8,length(X)))    # FV ⊂ [1]
  return m,n,marks,FV # marks: counter array; FV: Boolean matrix of F ⊂ 2^V
end
```

Listing 3: Main loop adding stepwise each FC's column

```julia
function mainloop(marks,n,V,EV,FE,FV)
  choose(marks) = findfirst(x -> x<2, marks) # function definition
  FC = convert(SparseMatrixCSC{Int8,Int64}, spzeros(n,0)) #  void FC
  while sum(marks) < 2n
    σ = choose(marks) # select σ ∈ c_{d-1}, "seed" of column extraction
    c_{d-1} = marks[σ]>0 ? sparsevec([σ],Int8[-1],n) : sparsevec([σ],Int8[1],n)
    c_{d-2} = (c_{d-1}' * FE)' # compute boundary cd2 of seed cell
    c_{d-1} = constructioncycle(c_{d-2},V,EV,FE,FV,c_{d-1}) # corolla ≡ 2-cycle
    FC = newcolumn!(c_{d-1},marks,FC) # update FC
  end
  return FC
end
```

Listing 4: Append C_3 basis column to FC

```julia
function newcolumn!(c_{d-1}, marks, FC)
  for σ ∈ SparseArrays.findnz(c_{d-1})[1] # update counters of used cells
    marks[σ] += 1 end
  return [FC c_{d-1}] # append column to operator: FC += c_{d-1}
end
```

Listing 5: Minimal 2-cycle construction

```julia
function constructioncycle(c_{d-2},V,EV,FE,FV,c_{d-1})
  Fvs,Evs,Fes = map(Lar.cop2lar,[FV,EV,FE])
  m = length(c_{d-2}); W = Matrix(V')
  global corolla
  while nnz(c_{d-2}) ≠ 0 # loop until c_{d-2} becomes empty
    corolla = sparsevec([],Int8[],m) # partial 2-cycle set to empty
    for τ ∈ (.*)(SparseArrays.findnz(c_{d-2})...)
```

```
      tau_1 = sparsevec([abs(\tau)], [sign(\tau)], m) # \tau sparse coord vector
      cbd_2 = FE * tau_1 # compute the \tau coboundary cbd_2 \in C_2 (coord vector)
      cells2D = SparseArrays.findnz(cbd_2)[1]
      # compute the \tau support
      inters = intersect(cells2D, SparseArrays.findnz(c_{d-1})[1])
      pivot = inters \neq [] ? inters[1] : error("no pivot")
      # compute the new aadj_2 cell
      fan = Lar.ord(abs(\tau),cbd_2, W,Fvs,Evs,Fes) # ord(pivot,cbd1)
      adj_2 = \tau > 0 ? thenext(fan,pivot) : theprev(fan,pivot)
      \sigma = c_{d-1}[pivot]; j = abs(\tau)
      corolla[adj_2] = FE[adj_2,j] \neq FE[pivot,j] ? \sigma : -\sigma # orient adj_2
    end
    # insert corolla cells c_2 coords in current c_{d-1} coord vector
    [c_{d-1}[k] = c_2 for (k,c_2) in zip(SparseArrays.findnz(corolla)...)]
    c_{d-2} = (c'_{d-1} * FE)'# compute again the coord vector of boundary of c_{d-1}
  end
  return c_{d-1} # coordinate vector, to append at coboundary matrix
end
```