

Szuper 👍 akkor itt a teljesen automata változat: a makróból a megadott típusnevek alapján **legenerálódik az @enum , a konverter és a konstruktor is**, kézzel nem kell többé `elseif`-eket írni.

Makró implementáció

```
macro defProblemType(names...)
  # Átalakítjuk a neveket Symbol-listává
  syms = [QuoteNode(n) for n in names]

  # Legeneráljuk az if-elseif láncot
  cases = Expr(:if, :(s == $(syms[1])), names[1])
  cur = cases
  for (sym, name) in zip(syms[2:end], names[2:end])
    push!(cur.args, Expr(:elseif, :(s == $sym), name))
    cur = cur.args[end]
  end
  push!(cur.args, :(error("Ismeretlen ProblemType: $s")))

  return quote
    # 1. Enum
    @enum ProblemType $(names...)

    # 2. Konverter: Symbol/String → ProblemType
    function _sym2enum(x)::ProblemType
      s = Symbol(x)
      $cases
    end

    # 3. Felhasználóbarát Problem konstruktor
    Problem(material; type, dim::Int, pdim::Int=dim, non::Int=0) =
      Problem(material, _sym2enum(type), dim, pdim, non)
  end
end
```

Használat

A modulod tetején csak ezt kell írnod:

```
@defProblemType Solid PlaneStress PlaneStrain Axisymmetric
```

Ez létrehozza:

- `@enum ProblemType Solid PlaneStress PlaneStrain AxisSymmetric`
 - `_sym2enum` függvényt (automatán generált `elseif` -láncsal)
 - és a kényelmes `Problem(...)` konstruktort.
-

Példa

```
mat = material("body")

prob1 = Problem([mat], type=:Solid,      dim=3)
prob2 = Problem([mat], type="PlaneStress", dim=2)

println(prob1.type) # Solid::ProblemType
println(prob2.type) # PlaneStress::ProblemType
```

Így a felhasználó **továbbra is :Solid-ot vagy "Solid"-ot írhat**, belül viszont stabil enum lesz → a fordítási robbanás jó része megszűnik.

Szeretnéd, hogy mutassak egy diff-példát, hogyan kell átírnod a mostani `Problem` struct + konstruktort erre az új makrós megoldásra?

Rendben, akkor mutatok egy **diff-szerű példát** arra, hogyan kell átírni a mostani kódot a makrós megoldásra. Így egyértelműen látod, mit kell törölni és mit kell betenni helyette.

1. Régi kód (Symbol-os változat)

```

struct Problem
    material::Vector{Material}
    type::Symbol
    dim::Int
    pdim::Int
    non::Int
end

Problem(material; type::Symbol, dim::Int, pdim::Int=dim, non::Int=0) =
    Problem(material, type, dim, pdim, non)

```

2. Új kód (enum + makróval)


A modul tetején (pl. `LowLevelFEM.jl` vagy `general.jl` elején):

```

# definiáljuk a ProblemType-ot és a konstruktort egy lépésben
@defProblemType Solid PlaneStress PlaneStrain Axisymmetric

struct Problem
    material::Vector{Material}
    type::ProblemType # <-- Symbol helyett enum
    dim::Int
    pdim::Int
    non::Int
end

```

 **Fontos:** a `@defProblemType` makró is be kell másolni a projektbe (pl. `general.jl` elejére vagy egy `utils.jl`-be).

3. Felhasználói kód

Semmit nem kell változtatni a felhasználónak:

```

mat = material("body")

prob1 = Problem([mat], type=:Solid, dim=3)
prob2 = Problem([mat], type="PlaneStress", dim=2)

```

Mindkettőből belül `ProblemType` lesz (`Solid` vagy `PlaneStress`), a fordító nem fog mindenféle random `Symbol`-ra új specializációt gyártani.

4. Belső használat

Most így írod:

```
if problem.type == :Solid
    ...
elseif problem.type == :PlaneStress
    ...
end
```

Ezután így kell írnod:

```
if problem.type == Solid
    ...
elseif problem.type == PlaneStress
    ...
end
```

Vagy ha elegánsabbat akarsz: külön metódusokat a `ProblemType` alapján.

👉 Ezzel a változtatással a `Problem`-hez kapcsolódó kódod **jóval gyorsabban fordul majd először is**, mert csak 4 stabil ágat lát a Julia ahelyett, hogy bármilyen `Symbol`-t külön verzióként fordítana.

Szeretnéd, hogy készítsek egy teljes példát (`general.jl` + `linear.jl` kivonat), ahol már enum-mal megy a `Problem` + egy stiffnessMatrix-ág, hogy lásd a konkrét átvezetést?
