U1ZFF0E5P : when you define a type alias, it also gives you a constructor for free. so with your type alias you can call `Macro2 "string1" "string2" "string3"` to build a record
U1ZFF0E5P : when you define a union type, the "options" are also constructors to build the actual type
U1ZFF0E5P : so you can't have a union type and a type constructor with the same name in the module
U1ZFF0E5P : you can move one or the other to another module and call it with the whole name
U1ZFF0E5P : <@U3LUC6SNS> ^
U3SJEDR96 : <@U3LUC6SNS> you'll need something like `oneOf [ parser1, parser2, parser3 ]`, so in your case `oneOf [ macro1, macro2, environment ]` etc
U3LUC6SNS : Thanks <@U3SJEDR96>, thanks <@U1ZFF0E5P> !
U3SJEDR96 : assuming that `macro1 : Parser Latex` etc
U3LUC6SNS : Got it!
U3LUC6SNS : Oops, I guess I don't understand -- if I define `type Latex` in a module A importing the module B where `Macro1` etc are defined, then the `Latex` will be undefined in module B. No?
U3SJEDR96 : <@U3LUC6SNS> oh, alright, so you have a `macro1 : Parser Macro1`, and you'd like to create a union type `Latex` and a `latex : Parser Latex`?```
module Parsers.Macro exposing (Macro1, macro1)

type alias Macro1 = {...}
macro1 : Parser Mactor1
macro1 = ...

---

module Parsers.Latex exposing (..)
import Parsers.Macro exposing (Macro1, macro1)
import Parser exposing (Parser, map, oneOf)

type Latex = M1 Macro1 | M2 Macro2 | Env Environment

latex : Parser Latex
latex =
  oneOf [map M1 macro1, map M2 macro2, map Env environment]
```
```

U3SJEDR96 : something like that
U3LUC6SNS : <@U3SJEDR96>, that works just great. I want to think about your use of `M1`, `map M1` etc. so as to understand it .. may be back with some questions after I have done my homework
U3SJEDR96 : Yeah, they're not the best names, to be honest. The thing is you need a way to distinguish between the tag for the union type, and the actual data-definition
U3SJEDR96 : as for `map` that's basically just "wrapping" it in that tag - much like you do with decoders, url-parsers, etc. `map : (a -> b) -> Parser a -> Parser b`
U3LUC6SNS : That makes sense ... I can fiddle with the names so that they look good in the final output.
U3LUC6SNS : <@U3SJEDR96>, there is still a problem with `oneOf` -- see session below. Would you mind taking a look when you have the chance?
The code is at <https://github.com/jxxcarlson/koko_elm_client/blob/master/src/LatexParser/Parser.elm>

```
&gt; import LatexParser.Latex
&gt; import LatexParser.Latex exposing(..)
&gt; import LatexParser.Parser exposing(..)
&gt; import Parser exposing(run)

&gt; run latex "\\foo{bar} "
Ok (Macro1 { name = "foo", arg = "bar" })
   : Result.Result Parser.Error LatexParser.Parser.Latex

&gt; run latex "\\foo{bar}{baz} "
Err { row = 1, col = 10, source = "\\foo{bar}{baz} ", problem = BadRepeat, context = [] }
   : Result.Result Parser.Error LatexParser.Parser.Latex
```

> run latex "\\begin{foo}Blah, blah!\\end{foo} "
Err { row = 1, col = 12, source = "\\begin{foo}Blah, blah!\\end{foo} ", problem = BadRepeat, context = [] }
    : Result.Result Parser.Error LatexParser.Parser.Latex
```

U3SJEDR96 : (you can use `inContext "macro1" &amp;lt;| succeed Macro1 |= something |. something else` to add contextual information to your parsers, btw)
U3SJEDR96 : so error messages allow you to track how and where things went wrong
U3LUC6SNS : Ah .. to have good error messages?
U3LUC6SNS : Yay!
U3SJEDR96 : my guess it's committing to stuff before bailing out. You'd need to delay that commit - <https://github.com/elm-tools/parser/blob/master/README.md#delayed-commits>
U3LUC6SNS : That sounds right and is consistent with the information given about the column of the error.  I'll read up on delayed commits.  Thanks!
U3LUC6SNS : And using context to provide better error messages I can confirm that premature commit is the problem.
U3SJEDR96 : in your case, on those statements, you could consider using something like mentioned here: <https://github.com/elm-tools/parser/blob/2f9c3370fe82211b5b4a37166f795face6801326/comparison.md#expressiveness>
U3CUFAX4H : Is there a package for generating the ordinal value of a char? (like ruby's String::ord method)
```