A Note on Parameterized Type Definitions
• A **GML document** consists of:
  – a list of **elements**

• **An element** is either:
  – a **word** or **markup** applied to an element

• **Markup** is either:
  – italicize, **bold**, or a **font name**
A GML document consists of:
- a list of elements

An element is either:
- a word or markup applied to an element

Markup is either:
- italicize, bold, or a font name

```
type markup = Ital | Bold | Font of string

type elt =
  Words of string list
| Formatted of markup * elt

type doc = elt list
```
Last Time: Challenge

- Change all of the “Arial” fonts in a document to “Courier”.
- Of course, when we program functionally, we implement change via a function that
  - receives one data structure as input
  - builds a new (different) data structure as an output
Challenge

• Change all of the “Arial” fonts in a document to “Courier”.

```java
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
| Formatted of markup * elt

type doc = elt list
```
• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
  Words of string list
| Formatted of markup * elt

type doc = elt list
```

• Technique: approach the problem top down, work on `doc` first:

```ocaml
let rec chfonts (elts:doc) : doc =
```
Challenge

• Change all of the “Arial” fonts in a document to “Courier”.

```ml
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
| Formatted of markup * elt

type doc = elt list
```

• Technique: approach the problem top down, work on `doc` first:

```ml
let rec chfonts (elts:doc) : doc =
    match elts with
    | [] ->
    | hd::tl ->
```
• Change all of the “Arial” fonts in a document to “Courier”.

```
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
  | Formatted of markup * elt

type doc = elt list
```

• Technique: approach the problem top down, work on `doc` first:

```
let rec chfonts (elts:doc) : doc =
    match elts with
    | [] -> []
    | hd::tl -> (chfont hd)::(chfonts tl)
```
Changing fonts in an element

• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
  | Formatted of markup * elt

type doc = elt list

let rec chfont (e:elt) : elt =
```

• Next work on changing the font of an element:
Changing fonts in an element

• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
  Words of string list
| Formatted of markup * elt

type doc = elt list
```

• Next work on changing the font of an element:

```ocaml
let rec chfont (e:elt) : elt =
  match e with
  | Words ws ->
  | Formatted(m,e) ->
```
Changing fonts in an element

• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
  | Formatted of markup * elt

type doc = elt list
```

• Next work on changing the font of an element:

```ocaml
let rec chfont (e:elt) : elt =
  match e with
  | Words ws -> Words ws
  | Formatted(m,e) ->
```
• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
  | Formatted of markup * elt

type doc = elt list
```

• Next work on changing the font of an element:

```ocaml
let rec chfont (e:elt) : elt =
  match e with
  | Words ws -> Words ws
  | Formatted(m,e) -> Formatted(chmarkup m, chfont e)
```
Changing fonts in an element

• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
    Words of string list
| Formatted of markup * elt

type doc = elt list
```

• Next work on changing a markup:

```ocaml
let chmarkup (m:markup) : markup =
```
Changing fonts in an element

• Change all of the “Arial” fonts in a document to “Courier”.

```ocaml
type markup = Ital | Bold | Font of string

type elt =
  | Words of string list
  | Formatted of markup * elt

type doc = elt list

let chmarkup (m:markup) : markup =
  match m with
  | Font "Arial" -> Font "Courier"
  | _ -> m
```
Summary: Changing fonts in an element

- Change all of the “Arial” fonts in a document to “Courier”
- Lesson: function structure follows type structure

```ml
let chmarkup (m:markup) : markup =
  match m with
  | Font "Arial" -> Font "Courier"
  | _ -> m

let rec chfont (e:elt) : elt =
  match e with
  | Words ws -> Words ws
  | Formatted(m,e) -> Formatted(chmarkup m, chfont e)

let rec chfonts (elts:doc) : doc =
  match elts with
  | [] -> []
  | hd::tl -> (chfont hd)::(chfonts tl)
```
Consider again our definition of markup and markup change:

```ocaml
type markup =
  Ital | Bold | Font of string

let chmarkup (m:markup) : markup =
  match m with
  | Font "Arial" -> Font "Courier"
  | _ -> m
```
Poor Style

• What if we make a change:

```ocaml
type markup =
  Ital | Bold | Font of string | TTFont of string

let chmarkup (m:markup) : markup =
  match m with
  | Font "Arial" -> Font "Courier"
  | _ -> m
```

the underscore silently catches all possible alternatives

this may not be what we want -- perhaps there is an Arial TT font

it is better if we are alerted of all functions whose implementation may need to change
Better Style

• Original code:

```ocaml
type markup =
    Ital | Bold | Font of string

let chmarkup (m:markup) : markup =
    match m with
    | Font "Arial" -> Font "Courier"
    | Ital | Bold | Font _ -> m
```
Better Style

• Updated code:

```ocaml
type markup =
   Ital | Bold | Font of string | TTFont of string

let chmarkup (m:markup) : markup =
  match m with
  | Font "Arial" -> Font "Courier"
  | Ital | Bold | Font _ -> m

..match m with
  | Font "Arial" -> Font "Courier"
  | Ital | Bold -> m..

Warning 8: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
TTFont _
```
• Updated code, fixed:

```ocaml
type markup =
  Ital | Bold | Font of string | TTFont of string

let chmarkup (m : markup) : markup =
  match m with
  | Font "Arial" -> Font "Courier"
  | TTFont "Arial" -> TTFont "Courier"
  | Font _ | TTFont _ | Ital | Bold -> m
```

• **Lesson**: use the type checker where possible to help you maintain your code
A couple of practice problems

• Write a function that gets rid of immediately redundant markup in a document.
  – Formatted(Ital, Formatted(Ital,e)) can be simplified to Formatted(Ital,e)
  – write maps and folds over markups

• Design a datatype to describe bibliography entries for publications. Some publications are journal articles, others are books, and others are conference papers. Journals have a name, number and issue; books have an ISBN number; All of these entries should have a title and author.
  – design a sorting function
  – design maps and folds over your bibliography entries
To Summarize

Design recipe for writing OCaml code:

- write down English specifications
  - try to break problem into obvious sub-problems
  - we took a top-down approach here
- write down some sample test cases
- write down the types for the code
- use the types to guide construction of the code:
  - tear apart inputs using pattern matching
  - handle each case, building results using data constructor
  - complex data structures defined by multiple type definitions
    - often one function per data type definition
- use your sample tests (and ideally others) to ensure correctness
  - 5 minutes to build testing code up front can help out big time in the long run!