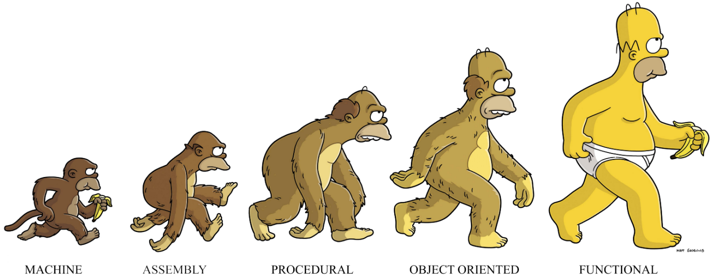
COMP302: Programming Languages and Paradigms

Prof. Brigitte Pientka (Sec 01)
bpientka@cs.mcgill.ca

Francisco Ferreira (Sec 02)
fferre8@cs.mcgill.ca

School of Computer Science
McGill University

Week 2-1, Fall 2017



Functional Tidbit: Evolution of Homer Simpson

$\mathbf{T} = \Lambda\alpha.\lambda x^\alpha \lambda y^\alpha .x$
$\mathbf{F} = \Lambda\alpha.\lambda x^\alpha \lambda y^\alpha .y$
OCaml

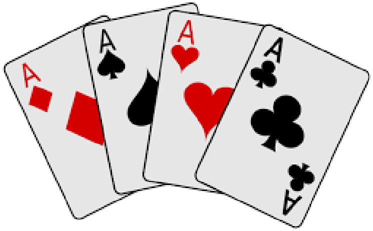
“ Pattern matching is so powerful and elegant! [...] it’s hard for me to return to languages without pattern-matching capabilities.” (Aliya Hameer)

Topic

Data Types and Pattern Matching
– Continued –

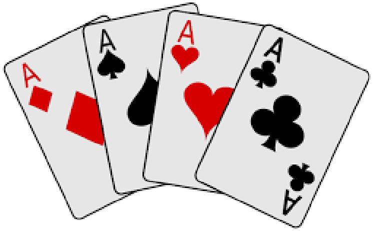
Recap: User-Defined (Non-Recursive Data Types

How can we model a collection of cards?



Recap: User-Defined (Non-Recursive Data Types

How can we model a collection of cards?



Declare a new type together with its elements

```
type suit = Clubs | Spades | Hearts | Diamonds
```

Recap: How Do We Work with User-Defined Data?

```
type suit = Clubs | Spades | Hearts | Diamonds
```

Recap: How Do We Work with User-Defined Data?

```
type suit = Clubs | Spades | Hearts | Diamonds
```

Pattern Matching

```
(* dom : suit*suit -> bool
   dom(s1,s2) = true
   iff suit s1 beats or is equal to suit s2
   relative to the ordering S > H > D > C
*)

let rec dom (s1, s2) = match (s1, s2) with
  | (Spades, _)          -> true
  | (Hearts, Diamonds) -> true
  | (Hearts, Clubs)     -> true
  | (Diamonds, Clubs)   -> true
  | (s1, s2)             -> s1 = s2
```

What's in your hand?

What's in your hand?

Describe the collection of cards in a `hand` inductively.

What's in your hand?

Describe the collection of cards in a `hand` inductively.

- `Empty` is of type `hand`
- If `c` is a `card` and `h` is of type `hand`, then `Hand(c, h)` is of type hand.
- Nothing else is of type hand.

What's in your hand?

Describe the collection of cards in a `hand` inductively.

- `Empty` is of type `hand`
- If `c` is a `card` and `h` is of type `hand`, then `Hand(c, h)` is of type hand.
- Nothing else is of type hand.

```
type hand = Empty | Hand of card * hand
```

Sample Hands

```ocaml
let hand0:hand = Empty
let hand1:hand = Hand((Ace, Hearts), Empty)
let hand2:hand = Hand((Queen, Diamonds), hand1)
let hand5:hand = Hand((Ace, Spades),
                  Hand((Ten, Diamonds),
                   Hand((Seven, Clubs),
                    Hand((Queen, Spades),
                     Hand((Eight, Clubs), Empty)))))
```

Task: Extract it!

Write a function `extract: suit -> hand -> hand`

`extract s h` returns a `hand` containing all cards from `h` of suit `s`.

Demo

Task: Find it!

Write a function `find` which when given a `rank` and a `hand`, finds the first card in `hand` of the specified `rank` and returns its corresponding `suit`.

Task: Find it!

Write a function `find` which when given a `rank` and a `hand`, finds the first card in `hand` of the specified `rank` and returns its corresponding `suit`.

$\Longrightarrow$ **Problem:** what to do if no such card exists?

Task: Find it!

Write a function `find` which when given a `rank` and a `hand`, finds the first card in `hand` of the specified `rank` and returns its corresponding `suit`.

$\Longrightarrow$ **Problem:** what to do if no such card exists?

Optional Data Type (predefined)

```
type 'a option = None | Some of 'a
```

Task: Find it!

Write a function `find: rank * hand -> suit option`.

Given a rank `r` and a hand `h`, `extract r h`

- finds the first card with rank `r` in `h` and return its corresponding suit `s` as `Some s`.
- returns `None`, if there is no card with rank `r`.

Demo