OCaml

What does this code do?

```
let rec sort n l =
    match n, l with
    | 2, x1 :: x2 :: _ ->
      if cmp x1 x2 <= 0 then [x1; x2] else [x2; x1]
    | 3, x1 :: x2 :: x3 :: _ ->
      if cmp x1 x2 <= 0 then begin
        if cmp x2 x3 <= 0 then [x1; x2; x3]
        else if cmp x1 x3 <= 0 then [x1; x3; x2]
        else [x3; x1; x2]
      end else begin
        if cmp x1 x3 <= 0 then [x2; x1; x3]
        else if cmp x2 x3 <= 0 then [x2; x3; x1]
        else [x3; x2; x1]
      end
    | n, l ->
      let n1 = n asr 1 in
      let n2 = n - n1 in
      let l2 = chop n1 l in
      let s1 = rev_sort n1 l in
      let s2 = rev_sort n2 l2 in
      rev_merge_rev s1 s2 []
```

…

Specification

(noun)

Intended behavior of a piece of code

(verb)

The act of creating such an artifact

Example specification

```
val sort : int list -> int list
```

- Returns a list with elements in ascending order
- 😕 *Can return a list with every element set to 0!*
- Returns a list with elements in ascending order, that is also a permutation of the input
- 😕 *Can return a list whose length is different than the input list!*
- Returns a list with elements in ascending order, that is a permutation of the input and has a same length as the input
- 👍

Specifications are contracts

Benefits

- **Locality:** understand abstraction without needing to read implementation
- **Modifiability:** change implementation without breaking client code
- **Accountability:** clarify who is to blame

Audience of specification

- **Clients**
  - What they must guarantee (preconditions)
  - What they can assume (postconditions)

- **Implementers**
  - What they can assume (preconditions)
  - What they must guarantee (postconditions)

Satisfaction

An implementation **satisfies** a specification if it provides the described behavior

An implementation may satisfy several specifications

- Client has to assume it could be any of them
- Implementer gets to pick one

SPECIFYING FUNCTIONS

A template for spec. comments

```
(** [f x] is ...
     Example: ...
     Requires: ...
     Raises: ...  *)
val f : t1 ... -> u
```

Based on *Abstraction and Specification in Program Development*

(Now *Program Development in Java: Abstraction, Specification, and Object-Oriented Design)*

By Barbara Liskov and John Guttag

Requires clause

```
(** [hd lst] is the head of [lst].
    Requires: [lst] is non-empty. *)
val hd : 'a list -> 'a
```

Precondition: blame client if input is bad

Requires clause

```
(** [hd lst] is the head of [lst].
    Requires: [lst] is non-empty. *)
val hd : 'a list -> 'a
```

Precondition: blame clie

Types are part of the source code not the comment.

Returns clause

```
(** [sort lst] contains the same
    elements as [lst], but sorted
    in ascending order. *)
val sort : int list -> int list
```

Postcondition: blame implementer if output is bad

(unless client violated a precondition)

Example clause

```
(** Examples:
    - [sort [1;3;2;3]] is [[1;2;3;3]].
    - [sort []] is [[]]. *)
val sort : int list -> int list
```

Super helpful to clarify spec for humans.

Raises clause

```
(** [hd lst] is the head of [lst].
    Requires: [lst] is non-empty.
    Raises: [Failure "hd"] if [lst]
      is empty. *)
val hd : 'a list -> 'a
```

Also a postcondition: behavior implementer must provide

Total function:

Well-defined behavior for all inputs.

No requires/raises clause needed.

Partial function:

Some inputs lead to unspecified behavior.

Requires/raises clause needed.

WORKING WITH SPECS

TL;DR: It's hard

Writing good specs is hard:

- the language and compiler do not demand it
- if you're coding only for yourself, neither do you

Reading specs is also hard:

- requires close attention to detail

When to write specifications

- **During design:**
  - as soon as a design decision is made, document it in a specification
  - posing and answering questions about behavior clarifies what to implement

- **During implementation:**
  - update specification during code revisions
  - a specification becomes obsolete only when the abstraction becomes obsolete

Upcoming events

- [Tomorrow] A2 due
- [Tomorrow] A3 *not* going out due to Winter Break
- Essay on *The Pragmatic Programmer* posted on CMS
- Any issues with assignments → discuss with section TAs first

This is hard.

THIS IS 3110