Formatter

Franklin Chen

July 28, 1996

This module provides the support needed for the pretty-printing program.

1 `Format` signature

We begin by providing the desired signature. Whereas [?] implement their corresponding C module as functions modifying the state of variables in static scope, we pass around the formatting information explicitly from the parser actions.

Warning: this module is still not referentially transparent, as output to standard output is immediately performed.

```
??    ⟨Format.sig ??⟩≡
        type info

        datatype margin = IN                        (* inward *)
          | EX                                      (* outward *)
          | AT                                      (* as is *)

        type action = info -> info

        (* Constructor *)
        val create : unit -> info
        val unitAction : action

        val nl : margin -> action
        val out : string -> action

        val at : margin -> action
        val cond : margin -> action
        val uncond : margin -> action
```

Defines:
- `action`, never used.
- `at`, never used.
- `cond`, never used.
- `create`, never used.
- `info`, never used.

`margin`, never used.
`nl`, never used.
`out`, never used.
`uncond`, never used.
`unitAction`, never used.

2 Format implementation

?? ⟨* ??⟩≡
```
local open ⟨Modules to open ??⟩ in
  ⟨Type definitions ??⟩
  ⟨Variable definitions ??⟩
end
```

?? ⟨*Modules to open* ??⟩≡ (? 0—1)
```
BasicIO
```

Start with the abstract data type `info` that holds all the state information.

?? ⟨*Type definitions* ??⟩≡ (? 0—1)
```
type info =
  int *                                   (* left margin, in tabs *)
  bool *                                  (* are we at left margin? *)
  int *                                   (* managed by cond *)
  int                                     (* managed by uncond *)

datatype margin = IN                      (* inward *)
  | EX                                    (* outward *)
  | AT                                    (* as is *)


type action = info -> info
```
Defines:
`action`, never used.
`info`, never used.
`margin`, never used.
Uses `at`, `cond`, and `uncond`.

The constructor returns an initialized `info` suitable for the beginning of processing a source file.

?? ⟨*Variable definitions* ??⟩≡ (? 0—1) ??▷
```
fun create () = (0, true, 0, 0)
```
Defines:
`create`, never used.

?? ⟨*Variable definitions* ??⟩+≡ (? 0—1) ◁?? ??▷
```
fun unitAction i = i
```
Defines:
`unitAction`, never used.

Adjust `lmargin` only.

?? ⟨*Variable definitions* ??⟩+≡ (? 0—1) ◁?? ??▷

```
fun at AT i = i
  | at IN (lmargin, atmargin, condflag, uncdflag) =
    (lmargin+1, atmargin, condflag, uncdflag)
  | at EX (lmargin, atmargin, condflag, uncdflag) =
    (lmargin-1, atmargin, condflag, uncdflag)
```

Defines:
`at`, never used.

?? ⟨*Variable definitions* ??⟩+≡ (? 0—1) ◁?? ??▷

```
and cond IN (lmargin, atmargin, condflag, uncdflag) =
  nl IN (lmargin, atmargin, condflag + 1, uncdflag)
  | cond EX (i as (_, _, 0, _)) = i
  | cond EX (i as (lmargin, atmargin, _, uncdflag)) =
  let
    val (lmargin', atmargin', condflag', uncdflag') = at EX i
  in
    (lmargin', atmargin', 0, uncdflag')
  end
```

Defines:
`cond`, never used.
Uses `at` and `nl`.

?? ⟨*Variable definitions* ??⟩+≡ (? 0—1) ◁?? ??▷

```
and uncond AT (lmargin, atmargin, condflag, uncdflag) =
  nl AT (lmargin, atmargin, condflag, uncdflag + 1)
  | uncond EX (i as (_, _, _, 0)) = at EX i
  | uncond EX i = i
```

Defines:
`uncond`, never used.
Uses `at` and `nl`.

?? ⟨*Variable definitions* ??⟩+≡ (? 0—1) ◁?? ??▷

```
and nl delta (i as (lmargin, atmargin, condflag, uncdflag)) =
  let
    val (lmargin', _, condflag', uncdflag') = at delta i
  in
    output(std_out, "\n");
    (lmargin', true, condflag', uncdflag')
  end
```

Defines:
`nl`, never used.
Uses `at`.

?? ⟨*Variable definitions* ??⟩+≡ (? 0—1) ◁??

```
and out s (i as (lmargin, atmargin, _, _)) =
  (
   if atmargin then
     let
       fun rep 0 = ()
         | rep n = (output(std_out, "\t"); rep (n-1))
     in
       rep lmargin
     end
   else
     ();
   output(std_out, s);
   (lmargin, false, 0, 0)
   )
```

Defines:
`out`, never used.

3 Indices

3.1 Chunks

3.2 Identifiers

References

[1] Axel T. Schreiner and H. George Friedman, Jr. *Introduction to Compiler Construction with UNIX*[1] . Prentice-Hall, Inc., New Jersey, 1985.

[1]UNIX is a trademark of Bell Laboratories.