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 [1] 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.

1 ⟨Format.sig 1⟩≡
```
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`, used in chunks 2 and 3.
- `cond`, used in chunk 2c.
- `create`, never used.
- `info`, never used.

`margin`, never used.
`nl`, used in chunk 3.
`out`, never used.
`uncond`, used in chunk 2c.
`unitAction`, never used.

2 Format implementation

2a ⟨** 2a*⟩≡
```
local open ⟨Modules to open 2b⟩ in
  ⟨Type definitions 2c⟩
  ⟨Variable definitions 2d⟩
end
```

2b ⟨*Modules to open* 2b⟩≡ (2a)
```
BasicIO
```

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

2c ⟨*Type definitions* 2c⟩≡ (2a)
```
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` 1 3a 3a, `cond` 1 3b 3b, and `uncond` 1 3c 3c.

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

2d ⟨*Variable definitions* 2d⟩≡ (2a) 2e▷
```
fun create () = (0, true, 0, 0)
```
Defines:
`create`, never used.

2e ⟨*Variable definitions* 2d⟩+≡ (2a) ◁2d 3a▷
```
fun unitAction i = i
```
Defines:
`unitAction`, never used.

Adjust `lmargin` only.

3a ⟨*Variable definitions* 2d⟩+≡ (2a) ◁2e 3b▷

```
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`, used in chunks 2 and 3.

3b ⟨*Variable definitions* 2d⟩+≡ (2a) ◁3a 3c▷

```
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`, used in chunk 2c.
Uses `at` 1 3a 3a and `nl` 1 3d 3d.

3c ⟨*Variable definitions* 2d⟩+≡ (2a) ◁3b 3d▷

```
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`, used in chunk 2c.
Uses `at` 1 3a 3a and `nl` 1 3d 3d.

3d ⟨*Variable definitions* 2d⟩+≡ (2a) ◁3c 4▷

```
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`, used in chunk 3.
Uses `at` 1 3a 3a.

4 ⟨*Variable definitions* 2d⟩+≡ (2a) ◁3d

```
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.