Recursive declarations

Generally SML has *linear visibility*, i.e. *“an id must be declared before it is used”*. However, in certain cases *recursion* is allowed.

Function declarations are allowed to be

- single recursive
- mutual recursive when combined with and, e.g.:

```
fun
 (* definition of f using g *)
   ⋮
 and
 (* definition of g using f *)
   ⋮
```

Recursive declarations

Type declarations are *not* allowed to be single or mutually recursive.

Datatype declarations are allowed to be

- single recursive: `datatype dt = ...  dt ...`
- mutual recursive when combined with and, e.g.:

```
datatype
 dt1 = ...  dt2 ...
 and
 dt2 = ...  dt1 ...
```

A group of datatype declarations is allowed to be mutually recursive with a group of (non recursive) type declarations, e.g.:

```
 datatype dt = ...  t ...

 withtype t = ...  dt ...
```

Mutual recursion, example: file system

Mutually recursive type declarations:

```
datatype elem =
    File of string
  | Catalogue of string*contents
withtype contents = elem list;
```

Mutually recursive function definitions:

```
fun nameElems(File s)             = [s]
  | nameElems(Catalogue(s, cnt)) =
      s::(nameContents cnt)

and nameContents [] = []
  | nameContents (e::es) =
      nameElems e @ (nameContents es);
```

Mutual recursion, example: file system

```
- val fs =
   Catalogue("ah",
             [File "readme",
              Catalogue("02153",
                        [File "foils.tex",
                         File "foils.pdf"]),
              Catalogue("papers",
                        [File "forms07.pdf"])]);
> ...

- nameElems fs;

> val it =
    ["ah", "readme", "02153", "foils.tex",
     "foils.pdf", "papers", "forms07.pdf"]
    : string list
```