```
functor MkSeqUtil(structure S : SEQUENCE) : SEQUENCE_UTIL =
struct
  structure Seq = S
  open Seq

  type 'a hist = ('a * int) seq

  fun tokens (cp : char -> bool) (s : string) : string seq =
      let
        val n = String.size s
        val chars = tabulate (fn i => (i, String.sub (s, i))) n
        val idx = map (fn (i,_) => i) (filter (fn (_,c) => cp c) chars)

        (* grab substrings in between delimiters *)
        val subs = map2 (fn (i,i') => String.substring (s, i, i' - i))
                        (append (singleton 0, map (fn i => i + 1) idx))
                        (append (idx, singleton n))
      in filter (fn s => size s > 0) subs
      end

  fun histogram (cmp : 'a ord) (s : 'a seq) : 'a hist =
      map (fn (a, c) => (a, length c))
          (collect cmp (map (fn a => (a, ())) s))

  fun choose (hist : 'a hist) (p : real) : 'a =
      let
        val (partial, total) = scan op+ 0 (map (fn (_,c) => c) hist)
        val offset = Real.floor (p * Real.fromInt total)
        val below = filterIdx (fn (i,_) => nth partial i <= offset) hist
        val (x, _) = nth below (length below - 1)
      in x
      end
end
```

```
functor MkTableKGramStats(structure Util : SEQUENCE_UTIL
                          structure T : TABLE
                            where type Key.t = string Util.Seq.seq
                          sharing T.Seq = Util.Seq) : KGRAM_STATS =
struct
  structure Table = T
  structure Seq = T.Seq
  open Util
  open Seq

  type token = string
  type kgram = token seq
  type kgramstats = (token * int) seq Table.table

  fun makeStats corpus maxK =
      let
        val toks = Util.tokens (not o Char.isAlphaNum) corpus
        fun kTable k =
            let
              fun gramExt i = (subseq toks (i,k), nth toks (i+k))
              val gramExts = tabulate gramExt (length toks - k)
              val gramCmp = collate String.compare
              fun makeHist (kgram, exts) =
                  (kgram, Util.histogram String.compare exts)
            in map makeHist (collect gramCmp gramExts)
            end
        val maxK' = Int.min (length toks, maxK + 1)
      in Table.fromSeq (flatten (tabulate kTable maxK'))
      end

  fun lookupExts stats kgram =
      getOpt (Table.find stats kgram, empty ())
end
```

```
functor MkBabble(structure R : RANDOM210
                 structure KS : KGRAM_STATS
                 structure Util : SEQUENCE_UTIL
                 sharing KS.Seq = Util.Seq
                 sharing KS.Seq = R.Seq) : BABBLE =
struct
  structure Rand = R
  structure Stats = KS
  open Stats.Seq

  exception NoData

  fun randomSentence stats n seed =
      let
        fun genWord ((ws, prefix), r) =
            let val exts = KS.lookupExts stats prefix
            in case (length exts, length prefix)
                 of (0, 0) => raise NoData
                  | (0, _) => genWord ((ws, drop (prefix, 1)), r)
                  | _ => let val word = Util.choose exts r
                         in (word::ws, append (prefix, singleton word))
                         end
            end

        val randomReals = R.randomRealSeq seed (SOME (0.0, 1.0)) n
        val (words, _) = iter genWord ([], empty ()) randomReals
      in String.concatWith " " (List.rev words) ^ "."
      end

  fun randomDocument stats n seed =
      let
        fun genSentence ((ss, seed), len) =
            (randomSentence stats len seed :: ss, R.next seed)

        val randomLengths = R.randomIntSeq seed (SOME (5, 10)) n
        val (sentences, _) = iter genSentence ([], seed) randomLengths
      in String.concatWith " " sentences
      end
end
```