```
:::::::::::::
words
:::::::::::::
with STRINGS
package words is

-- maximum length of word is 72 characters
type WORD is private;

constant eof_word:WORD;

-- either the next three or the three after that and within a threesome
-- the first and (a lot of the second or the third)

function make_empty_word return WORD;
function add_char_to_word(c:CHAR;w:WORD)return WORD;
function make_word_from_string(s:STRING)return WORD;

procedure make_empty_word(w:var WORD);

-- if word is already 72 characters, flags the word as overstuffed and
-- adds no characters to the word
procedure add_char_to_word(c:CHAR;w:var WORD);

-- puts only the first 72 characters of string into word, flagging the
-- word as overstuffed if there are more than 72 characters in the
-- string
procedure set_word_from_string(w:var WORD;s:STRING);

-- undefined if n<=0 or n>72
function nth_char_of_word(n:INTEGER;w:WORD)return CHAR;
function string_of_word(w:WORD)return STRING;

-- one of the following two
function litlize(w:WORD)return WORD;
procedure litlize(w:var WORD);

function is_eof_word(w:WORD)return BOOLEAN;
function was_longer_than_72_characters(w:WORD)return BOOLEAN;

-- since only the first 72 characters of a word are remembered, these
-- work on the basis of the first 72 characters of the words!
function "="(w1,w2:WORD)return BOOLEAN;
function ">"(w1,w2:WORD)return BOOLEAN; -- alphabetically
function "<"(w1,w2:WORD)return BOOLEAN; -- alphabetically
function ">="(w1,w2:WORD)return BOOLEAN; -- alphabetically
function "<="(w1,w2:WORD)return BOOLEAN; -- alphabetically

end words;
```

```
::::::::::::::
::::::::::::::
conc_db
::::::::::::::
::::::::::::::
with STRINGS, words, alphabetical_list, locations
package concordance_DB is

type ITERATOR is private;

-- declares local to the package an alphabetical_list of WORDS paired
-- with locations
procedure create_CBD;

procedure add_word_to_CDB(w:WORD);
function is_word_present_in_CDB(w:WORD)return BOOLEAN;
procedure add_location_to_word_in_CDB(w:WORD;loc:LOCATION);

-- attempt to avoid searching for word twice, first to see if present, and
-- then to add location
procedure add_word_if_needed_and_location_to_word_in_CDB(w:WORD;loc:LOCATION);
-- OR could keep word and pointer to cell for word found by is_word_present
-- so that if next application of add_location_to_word is for this word, the
-- pointer is used directly

-- this includes the flag for too long words, if necessary
function get_printable_line_for_word_from_CDB(w:WORD)return STRING;

-- secret: the database maintains the words in alphabetical order

-- each iterator keeps track of one looping so that multiple loops are
-- possible
function create_iterator_into_CDB return ITERATOR

-- the order of delivery of words is alphabetical
function get_first_word_from_CDB(i:ITERATOR) return WORD;
function get_next_word_from_CDB(i:ITERATOR) return WORD;
-- for both, when there are no words left, the returned word is the
-- eof_word

end concordance_DB;

-- uses from words:
-- function nth_char_of_word(n:INTEGER;w:WORD)return CHAR;
-- function string_of_word(w:WORD)return STRING;
-- function "="(w1,w2:WORD)return BOOLEAN;
-- function ">"(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- function "<"(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- function ">="(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- function "<="(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- constant eof_word:WORD;
-- function is_eof_word(w:WORD)return BOOLEAN;
-- function was_longer_than_72_characters(w:WORD)return BOOLEAN;
```

```
::::::::::::::
listword
::::::::::::::
with words, alphabetical_list
package list_of_words is

type LIST_OF_WORDS is private;
-- secret: stores words in alphabetical order for faster lookup
-- uses the alphabetical_list to provide basic alphabetization functions

procedure initialize_list(l:var LIST_OF_WORDS);
procedure add_word_to_list_if_not_already_present(w:WORD;l:var LIST_OF_WORDS);
procedure is_word_present_in_list(w:WORD;l:LIST_OF_WORDS)return BOOLEAN;

end list_of_words;

-- uses from words:
-- function nth_char_of_word(n:INTEGER;w:WORD)return CHAR;
-- function string_of_word(w:WORD)return STRING;
-- function "="(w1,w2:WORD)return BOOLEAN;
-- function ">"(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- function "<"(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- function ">="(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- function "<="(w1,w2:WORD)return BOOLEAN; -- alphabetically
-- constant eof_word:WORD;
-- function is_eof_word(w:WORD)return BOOLEAN;
::::::::::::::
alph_lst
::::::::::::::
generic package alphabetical_list(e: ELEM_TYPE) is

type ELEM_TYPE is private;
type ALPHBETIZED_LIST is private;

procedure initialize_list(l:var ALPHBETIZED_LIST);
procedure add_elem_to_list(w:ELEM;l:var ALPHBETIZED_LIST);
procedure is_elem_present_in_list(w:ELEM;l:ALPHBETIZED_LIST)return BOOLEAN;

end alphabetical_list;
```

::::::::::::::
::::::::::::::
ignored
::::::::::::::
::::::::::::::
with words, list_of_words
package ignored_words

procedure initialize_IW(ignore_case_flag:BOOLEAN);
−− if IW is initialized to ignore case distinctions, then each word is
−− litlized as it is added
procedure add_word_to_IW_if_not_already_present(w:WORD);
procedure is_word_present_in_IW(w:WORD)return BOOLEAN;

end ignored_words;

−− uses from words:
−− function litlize(w:WORD)return WORD;
::::::::::::::
::::::::::::::
proper
::::::::::::::
::::::::::::::
with words, list_of_words
package proper_names is

procedure initialize_PN;
procedure add_word_to_PN_if_not_already_present(w:WORD);
procedure is_word_present_in_PN(w:WORD)return BOOLEAN;

end proper_names;

::::::::::::::
::::::::::::::
doc_out
::::::::::::::
::::::::::::::
```
with STDIO
package document_output is

procedure initilize_document_output;
procedure new_page;
procedure new_line;
procedure add_character_to_current_line(c:CHAR);

function current_line_no;
function current_page_no;
-- document_output keeps the count for the input. Since the output is a
-- copy of the input, it's no problem and since the output is more convenient
-- for this purpose, we put it here

end document_output;

-- decided against having
-- package line into which one puts words
-- package page into which one puts lines

-- Do not put in words because a hyphenated word straddles lines
-- must put in characters.

-- Also circularity problem: since line must insert itself into page,
-- line calls page. Since starting new page means resetting line counter
-- to 1, page calls line. This is an indication that the two packages must
-- be one.
```

::::::::::::::
::::::::::::::
getword
::::::::::::::
::::::::::::::
```
with STDIO, words, document_output, locations, Delimiters
procedure get_next_word_and_its_location_from_input_and_add_it_to_output
                                    (w:var WORD;loc: var LOCATION);

-- Read in the next word from the input and get its location (page number
--              and line number from document_output,
--              where input is a possibly empty document, consisting of a list of
--              possibly empty pages, consisting of a list of
--              possibly empty lines, containing
--              possibly hyphenated words that may straddle lines and pages
--                                  and that are built from an unrestricted character set,
-- add the word to the printing of the document with the pages and lines
--              numbered in a special format and the pages separated by a line
--              of equal signs, and mark the containing line if the word is too long,
-- return the word and its location through the parameters

-- there are NO errors to detect. There is no limit on the length of the
-- document, a page, and a line. Also there is no limit on the length of an
-- input word, although only the first 72 count for distinguishing words.
-- That aspect is handled automatically by the words module so that it can be
-- ignored in this module, where such errors are normally detected.

-- while it adds the entire word character by character to the output, it
-- ends up making w have only the first 72 characters of the word, since
-- words simply ignores the 73rd and beyond characters

-- uses from words:
-- either the next three or the three after that and within a threesome
-- the first and (a lot of the second or the third)
-- function make_empty_word return WORD;
-- function add_char_to_word(c:CHAR;w:WORD)return WORD;
-- function make_word_from_string(s:STRING)return WORD;

-- procedure make_empty_word(w:var WORD);
-- procedure add_char_to_word(c:CHAR;w:var WORD);
-- procedure set_word_from_string(w:var WORD;s:STRING);

-- uses from document_output:
-- procedure new_page;
-- procedure new_line;
-- procedure add_character_to_current_line(c:CHAR);

-- function current_line_no;
-- function current_page_no;
```

```
::::::::::::::
location
::::::::::::::
package locations is

type LOCATION is private;

-- a single data item specifying the location of a word, a pair consisting
-- of a page number and a line number

function make_location(page_no, line_no:INTEGER)return LOCATION;

-- makes a location of a page_no, line_no pair

function get_page_no (l:LOCATION)return INTEGER
function get_line_no (l:LOCATION)return INTEGER

-- extract the components of a location

end locations;
::::::::::::::
delimiter
::::::::::::::
with STRINGS
package Delimiters is

procedure create_delimiters;

-- initializes Delimiters with correct set of whitespace and punctuation
-- characters, so that they can be changed if necessary

function is_white_space (c:CHARACTER)return BOOLEAN;
function is_punctuation (c:CHARACTER)return BOOLEAN;

end Delimiters;
```

```
::::::::::::::
rd_ignor
::::::::::::::
with STDIO, ignored_words, words, STRINGS, Delimiters
procedure read_in_ignored_words_and_insert_them_possibly_litlized_into_IW
                                        (ignore_case−flag:BOOLEAN;ignored_words_file_name:STRING);

−− intializes ignored words database possibly to ignore case and then fills it

−− uses from words:
−− either the next three or the three after that and within a threesome
−− the first and (a lot of the second or the third)
−− function make_empty_word return WORD;
−− function add_char_to_word(c:CHAR;w:WORD)return WORD;
−− function make_word_from_string(s:STRING)return WORD;

−− procedure make_empty_word(w:var WORD);
−− procedure add_char_to_word(c:CHAR;w:var WORD);
−− procedure set_word_from_string(w:var WORD;s:STRING);

−− function nth_char_of_word(n:INTEGER;w:WORD)return CHAR;
−− function string_of_word(w:WORD)return STRING;
−− function "="(w1,w2:WORD)return BOOLEAN;
−− function ">"(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function "<"(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function ">="(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function "<="(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function litlize(w:WORD)return WORD;
−− constant eof_word:WORD;
−− function is_eof_word(w:WORD)return BOOLEAN;
::::::::::::::
rd_propr
::::::::::::::
with STDIO, proper_names, words, STRINGS, Delimiters
procedure read_in_proper_names_and_them_insert_into_PN
                                        (proper_names_file_name:STRING);

−− intializes proper names data base and then fills it

−− uses from words:
−− either the next three or the three after that and within a threesome
−− the first and (a lot of the second or the third)
−− function make_empty_word return WORD;
−− function add_char_to_word(c:CHAR;w:WORD)return WORD;
−− function make_word_from_string(s:STRING)return WORD;

−− procedure make_empty_word(w:var WORD);
−− procedure add_char_to_word(c:CHAR;w:var WORD);
−− procedure set_word_from_string(w:var WORD;s:STRING);

−− function nth_char_of_word(n:INTEGER;w:WORD)return CHAR;
−− function string_of_word(w:WORD)return STRING;
−− function "="(w1,w2:WORD)return BOOLEAN;
−− function ">"(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function "<"(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function ">="(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− function "<="(w1,w2:WORD)return BOOLEAN; −− alphabetically
−− constant eof_word:WORD;
−− function is_eof_word(w:WORD)return BOOLEAN;
::::::::::::::
prn_head
::::::::::::::
with STDIO
procedure print_header;
```

```
::::::::::::::
concord
::::::::::::::
with STDIO, STRINGS, document_output, concordance_DB, words, GETARGS,
                ignored_words, proper_names, locations
procedure concordance is

separate procedure
                get_next_word_and_its_location_from_input_and_add_it_to_output
                (w:var WORD,loc:var LOCATION);
separate procedure
                read_in_ignored_words_and_insert_them_possibly_litlized_into_IW
                (ignore_case_flag:BOOLEAN;ignored_words_file_name:STRING);
separate procedure
                read_in_proper_names_and_them_insert_into_PN
                (proper_names_file_name:STRING);
separate procedure
                print_header;

w:WORD
loc:LOCATION;
input:FILE;
i:ITERATOR;

ignored_words_flag:BOOLEAN;
ignored_words_file_name:STRING;
ignore_case_flag:BOOLEAN;
proper_name_flag:BOOLEAN;
proper_name_file_name:STRING;

read in, aborting if error, the command line arguments and set
                                ignored_words_flag
                                ignored_words_file_name
                                ignore_case_flag
                                proper_name_flag
                                proper_name_file_name
                and determine input file−either stdin or named file−and
                set input to name it;

create_CDB;
initialize_document_output;

if ignored_words_flag then
                read_in_ignored_words_and_insert_them_possibly_litlized_into_IW
                                                (ignore_case_flag,ignored_words_file_name);
end if;

if proper_names_flag then
                read_in_proper_names_and_them_insert_into_PN(proper_names_file_name);
end if;
```

```
get_next_word_and_its_location_from_input_and_add_it_to_output(w,loc);
                                        −− w, and loc are passed by reference.
while (not is_eof_word(w)) loop
                        if ignore_case_flag then
                                        if (not is_word_present_in_PN(w)) then
                                                        w := litlize(w);
                                        end if;
                        end if;
                        if (not is_word_present_in_IW(w)) then
                                        if (not is_word_present_in_CDB(w)) then
                                                        add_word_to_CDB(w);
                                        end if;
                                        add_location_to_word_in_CDB(w,loc);
                        end if;
                        −− OR
                        −− if (not is_word_present_in_IW(w)) then
                        −−                      add_word_if_needed_and_location_to_word_in_CDB(w,loc);
                        −− end if;
                        get_next_word_and_its_location_from_input_and_add_it_to_output(w,loc);

end loop;

print_header;

i := create_iterator_into_CDB;
w := get_first_word_from_CDB(i);
while (not is_eof_word(w)) loop
                        put(get_printable_line_for_word_from_CDB(w));
                        w := get_next_word_from_CDB(i);
end loop;
end concordance;
```