Outline: `printcall` is a subroutine called once. Its job is to initialize conditions and put the object `s` to be printed on the dot stack. Then it will exhaust the stack, sending the `s` at the highest level, lowest dotlabel to `char`.

`Char` is a subroutine which takes `s` and finds the character string representing it, adding objects to the dot stack if necessary. (A part of `s` is put on the dot stack and replaced by an integer followed by a period (its dotlabel) in the character representation of `s`, whenever it is nested to a depth greater than `dep`.

`Post` is a subroutine which periodically adds pieces of `s` to the output media. It checks line lengths and spaces the actual printout to improve readability.

define printcall(s); dep=4; level=0; dot=nl; string=nulc; n=0; lastcall=0; outs=nulc; linelength=72; dot(level,n)=s; num=3;
(while dot ne nl)nextlevel=[max:y ∈ dot] hd y; depth=0; m=0;
  s=dot(nextlevel, 0 ≤ j[dotlabel] ≤ n / dot(nextlevel,dotlabel) ne ∧)
  dot=dot less s; if nextlevel ne 0 then post(2∗
  (nextlevel // (linelength/4))∗ ' ' + dec dotlabel + '.');
  level=nextlevel;
  char(s,level); lastcall=1; end while;
post(string); return; end printcall;
define char(s,level); printcall external depth, level,
  n,m,outs,dot,dep,num;
  /* Printouts for various atoms are provisional, subject to decision on actual printed representations for them */
if \texttt{atom} \ s then if \texttt{integer} \ s then post(\texttt{dec} \ s) ;
else if \ s \ \texttt{eq} \ t \ then \ post(\texttt{'\$TRUE\$'}) ;
else if \ s \ \texttt{eq} \ \texttt{T} \ then \ post(\texttt{'}\$FALSE\$\texttt{'}) ;
else if \texttt{label} \ s then post(\texttt{'}\$LABEL\texttt{'}) ;
else if \ s \ \texttt{eq} \ \texttt{\textbackslash ~} then post(\texttt{'}\$\\textbackslash ~\texttt{ATOMIC}\texttt{'}\texttt{') ;
else if \texttt{blank} \ s then post(\texttt{'}\$\texttt{BLANK ATOMIC}\texttt{'}\texttt{') ;
else if \texttt{character} \ s then post(\texttt{'}\$\texttt{CHARACTER}"\texttt{') ;
else if \texttt{subroutine} \ s then post(\texttt{'}\$\texttt{SUBROUTINE}\texttt{') ;
else if \texttt{function} \ s then post(\texttt{'}\$\texttt{FUNCTION}\texttt{') ;
else if \texttt{bit} \ s then til (bitstring:); /*All bit
string constants are printed as (binary part) \texttt{b} (octal part),
without parentheses or blank spaces. The empty bit string is
denoted by \texttt{Obb}. */

\begin{verbatim}
bitpart=\langle\texttt{len} \ s\rangle/3)\texttt{first} \ s; \texttt{outbit}=\texttt{nulc} ; \texttt{i}=\texttt{len} \ bitpart ;
bp=1 ; tablebit=\langle0 , '0'\rangle , \langle1 , '1'\rangle ; (\texttt{while} \ bp \ \texttt{le} \ i \\
\texttt{doing} \ bp=bp+1) \texttt{bitdigit}=bp \ \texttt{elt} \ bitpart ;
\texttt{outbit}=\texttt{outbit}+\texttt{tablebit}(\texttt{bitdigit}) ; \texttt{end \ while} ;
\texttt{octpart}=(s-(\texttt{len} \ s/3)) \texttt{last} \ s ; \texttt{outoct}=\texttt{nulc} ;
p=\texttt{len} \ octpart ; \texttt{op}=1 ; tableoct=\langle000 , '0'\rangle \\
\langle001 , '1'\rangle , \langle010 , '2'\rangle , \langle011 , '3'\rangle , \langle100 , '4'\rangle \\
\langle101 , '5'\rangle , \langle110 , '6'\rangle , \langle111 , '7'\rangle ;
(\texttt{while} \ op \ \texttt{le} \ p \ \texttt{doing} \ op=op+3) \texttt{octdigit}=
(optimizer elt octpart)+(op+1 elt octpart)+(op+2 elt octpart) ;
\texttt{outoct}=\texttt{outoct}+\texttt{tableoct}(\texttt{octdigit}) ; \texttt{end \ while} ;
\texttt{if} \ \texttt{outbit} \ \texttt{eq} \ \texttt{nulc} \ \texttt{and} \ \texttt{outoct} \ \texttt{eq} \ \texttt{nulc} \ \texttt{then} \ \texttt{post}('\texttt{Obb}') ;
\texttt{else} \ \texttt{post}(\texttt{outbit}+\texttt{b}+\texttt{outoct}) ; \texttt{end \ if} \ \texttt{outbit} ;
[\texttt{bitstring:}]; \ \texttt{go \ to} [\texttt{out:}];
\texttt{end \ if} \ \texttt{atom};
\end{verbatim}

if \texttt{pair} \ s then if(\texttt{depth} \ \texttt{lt} \ \texttt{dep} \ \texttt{and} \ \texttt{m} \ \texttt{lt} \ \texttt{num})
then post('\texttt{<}'); \texttt{depth}=\texttt{depth}+1 ; (\texttt{while} \ \texttt{pair} \ \texttt{s}
\texttt{doing} \ \texttt{s}+\texttt{tl} \ \texttt{s}) \ \texttt{post} (\texttt{char}(\texttt{hd} \ \texttt{s},\texttt{level})+\texttt{'}\texttt{,}'\texttt{);} \ \texttt{end \ while} ;
\texttt{post}(\texttt{char}(\texttt{s},\texttt{level})+\texttt{'}\texttt{>}\texttt{');} \texttt{depth}=\texttt{depth}+1 ; \ \texttt{go \ to} [\texttt{out:}];
\texttt{else} \ \texttt{n}=\texttt{n}+1 ; \ \texttt{dot}(\texttt{level}+1,\texttt{n})=\texttt{s} ; \ \texttt{post}(\texttt{dec} \ \texttt{n}+\texttt{'}\texttt{.}'\texttt{);} ;
\texttt{go \ to} [\texttt{out:}]; \ \texttt{end \ if} \ \texttt{pair} ;
if \( \forall x \in s/(\text{pair } x \text{ and integer } hd \ x) \text{ and } 1 \leq \forall k \leq \#s/ (s(k) \neq \bot) \text{ then go to [seq:]; else go to [set:];} \end{if}

[seq:] if (depth \text{ lt } dep)\text{ and } m \text{ lt } num \text{ then post('['); depth=depth+1; \( (1 \leq \forall k \leq (\#s-1)) \text{post}(\text{char}(s(k),\text{level})+',') \text{ end if; }

[set:] if (depth \text{ lt } dep \text{ and } m \text{ lt } num) \text{ then post('}'); depth=depth+1; define\ comma(a); if not(l last a \in \{', '\}'\}) \text{ then } b=','; \text{ else } b=nulc;; \text{ return } b; \text{ end comma; } (\forall x \in s)\text{post}(\text{comma}(outs)+\text{char}(x,\text{level}));\text{end } \forall x; \text{ post('}'); \text{depth=depth-1; go to [out:]; else } n=n+1; \text{dot}(\text{level+1},n)=s;\text{post(dec } n + '.');\text{go to [out:];} \text{end if;}

[out:] return; end char;

define\ post(obj); /* If nesting of sets, sequences or pairs is greater than 4 deep, the nested sets, sequences or pairs will be assigned abbreviated designators consisting of an integer followed by a decimal point, and it will be printed on a separate line, with indentations to improve readability. If there are too many abbreviators per line (for now, say an average of 6), we increase allowable nesting by 2. Additional tests may be provided later to refine printout. See also comment below. */

printcall external linelength, outs, dep, lastcall, depth, m, n, num; if (len outs + len obj gt linelength) or (linelength-len outs le 2) or (lastcall eq 1) then output = output + er + outs; m=m+1; outs=nulc; lastcall=0; end if;

outs = outs+obj; if m \text{ ne } 0 \text{ and } (n/m) \text{ gt } 6 then dep=dep+2; num=num+1; return; end post;

/* After (num-1) lines of printing per dotlabel, all sets, sequences, and pairs are replaced by abbreviated designators. */