The following updates the print call subroutine given in SETL Newsletter Number 25. There are some differences to be noted:

1 - A Dewey decimal system is used for abbreviations instead of consecutive numbering. Thus, if s contains an abbreviated item, say l*, the first item in l* to be abbreviated will be labelled l*l*. Note that a '*' is used in place of the '.'.

2 - The indentation is continued if an item requires more than one line of printing.

3 - The linelength is initialized to 120 and not 72.

4 - No provision is made for printing sequences in the form \([r_1, r_2, \ldots, r_k]\). The reason is that sequence is not a data type and determining whether a set is a sequence is too time consuming. If the run time library had provision for flagging sets that are sequences, then it would be worthwhile to print sequences in a special format. With the new form for tuples, however, it is unlikely that sequences will play such a major role in SETL algorithms.

Print Routine.

```plaintext
define printcall(obj); /* obj is any SETL object. It will be printed using, essentially, the format described in the SETL notes, pp. 76-78 */
level=0; /* level of nesting on the printed page - used to control the amount of indentation */
dep=4; /* dep is the maximum depth of nesting not requiring abbreviation */
um=3; /* if, in the course of printing an object, more than num-1 lines are required, then all tuples and sets remaining in that object are abbreviated */```
printc(obj,nulc); /* printc is the routine that does the printing - since no label for this object the second argument is nulc. */
return;
end printcall;

define printc(s,dotlabel); /* printc prints an object s labelled by the string in dotlabel */
printcall external level;
post external linenumber,linelength,position;
depth=0; /* depth counts the level of nesting in s */
t=nulc; /* if the level of nesting or the number of printlines becomes excessive, then the items in s are abbreviated and saved in t for subsequent printing */
post('er'); /* end the present line */
position=2*(level//((linelength/4)));
/* indent 2 spaces for each level */
post(dotlabel+'b');
position=position+#dotlabel+1;
lineno=linenumber; /* save the current linenumber so the char routine can determine when to abbreviate items */
char(s); /* char posts the object s with abbreviations. The abbreviated items are queued in the tuple t. */
level=level+1;
(l<>k<>t) printc(t(k), dotlabel+dec k + 'i');; /* Print out each of the abbreviated items */
level=level-1;
return;
end printc;
define post(x); /* the string x is added to output.
Record size is determined by linelength.
Each line of print is preceded by position-1 blanks. The variable linenumber is incremented each time a line is written. When x is the character string 'er' an end of line condition is forced. This condition is recognized on the next entry to post (for which x≠'er'). The value of position is used only when a new line is begun. */

initially linelength=120; linenumber=0; position=1;
p=0; line=nulc; /* line is the current line */

if x eq 'er' then p=linelength; return;;
y=x;
    (while y ne nulc doing p=p+j;y=y(j+1:));
    if p eq linelength then
        output=output+line+'er';
        p=position-1;
        linenumber=linenumber+1;
        line=p#'b'; end if p;
        j=#y min (linelength-p);
        line=line+y(1:j); end while;
    return;
end post;
define char(s); /* This routine posts the representation of the object s with appropriate abbreviations inserted. Abbreviated items are queued in t, to be subsequently printed by the printc routine */

printcall external dep, num, n;
princ external depth, t, linno, dotlabel;
post external linenumber;

iff atomtest?
   printatom, | mtset?
   printmt, abbrtest?
   ifupertest? abbreviate,
   ptupl, pset;

atomtest:=atom s;
mtset:= # s eq 0;
/* test for atom */
/* null set and null tupl treat separately */

printmt: post(if s eq nult then '<<' else '??');
abbrtest: m=linenumber-lineno; = depth lt dep and (m lt num);
/* We replace the item with an abbreviation as soon as the level of nesting exceeds dep or the number of lines already used for this item exceeds num-1 */

abbreviate:k=#t+1;
t(k)=s;
post(dotlabel+dec k+'*');
/* Check density - if more than six abbreviations per line, then increase the depth limit by 2 */

if m gt 0 and (k/m gt 6) then
dep=dep+2;
num=num+1;;

tupltest: sw=t; depth=depth+1;=type s eq tupl;
/* print out tuple */
ptupl: (\k\leq s doing \text{sw=f};)
    post(if \text{sw then} '<' \text{else},');
    char(s(k));
    post('>');
    depth=depth-1;

    /* print out a set - the special case of sequence is not implemented herein */

pset: (\forall x \in s \text{ doing sw=f};)
    post(if \text{sw then} '{' \text{else},');
    char(x);
    post('}');
    depth=depth-1;

printatom: if type s eq cstring then
    post('""');
    (\forall x \in s) post(if x eq "" then "" else x);
post('""');
else if type s eq bstring then
    if \#s eq 0 then post('000');
    else if \#s eq 0 then post('<0', '1', '10', '11'>
                 (bitr(s(1,k)+1)));
        post('B'); k=k+1;
    (1 \leq \forall 1 \leq \#s/3 \text{ doing } j=j+3;)
    post('<0', '1', '2', '3', '4', '5', '6', '7'>
                 (bitr(s(j:3)+1)));
end \forall 1; end if \#s;
else post(s as cstring); end printatom;
end iff;
return;
end char;