TO: All R & D Personnel

SUBJECT: The Formatted Output Package "loas".

The entry points loas and loasrs perform output conversion and formatting services, and are callable from any user program. Programs running in 64V mode must use the V-mode library version VIOLIB; programs running in any other mode use the R-mode version IOALIB (temporarily, X.IOALIB). The calling sequence for loas and loasrs seen at the user level is the same for either version.

The formal documentation for loas and loasrs is presented below. By way of introduction, however, the following comments are in order. Basically, loas allows the caller to specify a character string called a control string, as well as up to 99 optional arguments. The control string specifies literal text to be output, but intermixed within the literal text can be escape sequences called conversion requests. These control the conversion of the data represented by the optional arguments to character-string form, and specify how the results of the conversions are to be edited into the final output string. Thus, conversion requests specify both the data type of an optional argument and the format of the resulting conversion.

The principal reasons for using loas instead of FORTRAN I/O (with FORTHA statements) are the following. First, loas is a smaller total package than FORTRAN I/O by a factor of about 2.5 seconds; the conversion specifiers available in loas are more flexible than those provided by FORTRAN (for example, loas permits control over justification, zero-filling, and so forth). And finally, the format of an loas control string is particularly convenient in terms of ease of concatenation of conversions with literal text, and of a possible need to specify a conversion format at runtime.

**Loading the loas Package**

To load the V-mode version, use the loader command "library violib".
To load the R-mode version, use "library x.10alib".

```
OR PHA routine: SEG
DYNTO ul:
END
```

*For V-mode*
Overview: subroutine-loa5 is called to generate formatted output text and input it to the user's terminal. Conversions of several data types to various formats can be easily interspersed with literal text in a natural and flexible way.

Present version of loa5 does not support floating-point conversions of any type, nor does it support automatic conversion of entire arrays.

Note that any alphabetic character in a conversion specifier may be upper- or lower-case, equivalently. In the following discussions, the phrase "next argument" refers to the next optional argument in sequence from the list arg1, ..., arg99. If no more arguments exist when one is needed, formatting is terminated. A given argument is used only once, unless the reposition request ("y" conversion) is used. Example: In the specifier %27.1f, %w is 27, <prec> is 1, *z* is present, *z* is omitted, and <type> is *c*.

<fw> specifies the field width, or in some cases a repeat count. If omitted or zero, fields will default to the exact number of positions required to represent the value. An omitted repeat count defaults to 1, while a zero repeat count is honored as such (i.e. as a no-operation). A nonzero <fw> specifies the number of character positions to be occupied by the converted datum, filled with blanks or zeroes as needed. A datum can be left- or right-justified in its field. A constant field width is specified by providing a decimal integer for <fw>. If the field size is to be computed at execution time, a variable field width is specified by providing the character "*" for <fw>. The next argument will then be used as a single precision integer specifying the field width. A negative integer <fw> is processed as if <fw> were omitted.

<prec> is the argument precision. Three different precisions are allowed at present: 0 for fixed binary(16,0) or "single precision unsigned"; 1 for fixed binary(15,0) or "single precision signed"; and 2 for fixed binary(31,0) or "double precision signed". These correspond to the FORTRAN datatypes INTEGER2 (taken as unsigned), INTEGER2 and INTEGER4, respectively. If <prec> is numeric but greater than 2, 2 is assumed; if it is non-numeric, the precision specifier is ignored. Thus "2:1:" will result in a precision of 2. If <prec> is omitted, 1 is assumed.

If specified, means that a numeric field is to be zero-filled instead of blank-filled. Note that "z" will have no visible effect if padding is not actually needed (<fw> omitted or too small to contain the value). *z* has no effect on octal or pointer conversions. The "*z*" and "*t*" conversions use *z* in a special way see below.

If specified, means that the default justification sense for the conversion will be reversed. See the individual conversion request descriptions for specifics.

>Type specifies the control operation, or the conversion to be performed (and hence the datatype of the output argument, if one is used). Implemented <type>s are:

Literal *x*, *z*, *r*, <prec> and <fw> are ignored.
decimal integer. The next argument is the integer of precision specified by <prec>. "z" and <fw> are honored, except that <fw> is ignored if it is too small to contain the value. "r" is honored; the default justification sense is right.

is like "d", except conversion is done in octal radix.

is like "d", except conversion is done in hexadecimal radix.

generate newline characters. <FW> is a repeat count; <FW> newlines will be generated. "z", "r", and <prec> are ignored. Due to a peculiarity of the operating system routine that performs terminal I/O, the character pair CR followed by LF is currently used to represent 'newline'.

generate formfeed characters. <FW> is a repeat count; <FW> formfeeds will be generated. "z", "r", and <prec> are ignored. The ASCII formfeed character (octal 21a) is presently used to represent formfeeds.

generate filler. <FW> is a repeat count; <FW> filler characters are generated. If the "z" flag is present, the filler character used will be 011 else blanks are used. "r" and <prec> are ignored.

reposition in argument list. <FW> specifies the argument number. If <FW> = 1 will position to arg (the first optional argument). If <FW> is less than 1, 1 is assumed; if greater than 99, 99 is assumed. Hence, just "z" repositions to the first argument. "z", "r", and <prec> are ignored.

terminate control string. "z", "r", <prec> and <FW> are ignored.

terminate control string and append single reulline to output. "z", "r", <prec> and <FW> are ignored.

Logical. <Prec> is ignored; the next argument is a single 16-bit word representing PL/I vocal type "01(16) aligned" (Fortran: LOGICAL). The datum is considered "true" if any bit is 1; else "false". The result of the conversion is one of the letters "T" or "F", unless the "z" flag is present, in which case "TRUE" and "FALSE" are used instead. The default justification sense is right.

word. Equivalent to :oa (unsigned, single precision zcrofilled octal). "z" and <prec> are ignored. Justification is handled as for the "a" converter.

ASCII. The next argument is a nonvarying character string (Fortran: array containing packed characters); the following argument is a precision t integer containing the length of the string in characters. A negalive or zero string length is interpreted as representing the null string. The string length is adjusted downward by stripping all trailing blanks (e.g., all blanks to the right of the rightmost nonblank character). The default justification sense is left. "z" and < prec > are ignored.

ASCII. Like "e", except that trailing blanks are not stripped; that is, if the string length is N, exactly N characters will be output.

Varying character string. The next argument is a varying character string (Fortran: array whose first element contains the string length and remaining elements contain packed characters). No length argument is needed; as the string is self-describing. The string is converted as for the "a" conversion above.

pointer. In R-mode, the next argument is a single-word address value. The address value is converted to a non-zero-filled unsigned octal integer. In V-mode, the next argument is a pointer and may be either 2 or 3 words long, depending on the bit extension flag of the pointer. If the pointer has its fault bit set, the result of the conversion has the format "FAULT/AAAAAA", where A is the first word of the pointer in octal; otherwise, the format used is "SSSS(R)/AAAAAA(6B)", where R is the ring number, S are the segment number, A is the word number and (6B) the bit offset (if any). All values octal except 6B, which is in decimal if present. The default justification sense is left.

start repeat group. <FW> is the repeat count, and must be nonzero. If <FW> is omitted, 1 is assumed. All text and conversions between the opening \Xt and the closing \Xt (see below) are repeated <FW> times on successive optional arguments if any is used. If no closing \Xt is encountered, then the repeats will not be performed. Nested repeat groups are not presently allowed; if a \Xt is encountered inside a repeat group, the result of the inner \Xt conversion will be two question marks. "z", "r" and <prec> are ignored.

end repeat group. The group is repeated until the repeat count specified in the opening \Xt is exhausted. <FW>, "z", "r" and <prec> are ignored.

Notes:
If an illegal <type> is found, the result of the conversion is the two characters "??". If the output buffer overflows, or if the input argument list is exhausted when a new argument is needed, conversion is terminated (without a newline appended).

> Use an internal working buffer of capacity 410 characters. Thus,
400 characters may be generated by a single call to loa$.

---

**Fortran**

```fortran
CALL IDATS(VALUE= X:2C.X',*100,455809) 
DATA VALUE/455809.
CALL IDATS*LOCATION XP CONTAINS X6X.X',*100,XCC(I),1) 
produce 
LOCATION 4DF(3)/2707 CONTAINS 163022.
CALL IDATS*X12A IS X5*?L.I',*15,VERITAS*,7, 
TRUE.) 
VERITAS IS TRUE.
```

---

**PL/I**

```pli
declare loa$s entry
a char(14) var; b char(32) aligned; 
c ptr; d fixed bin; e fixed bin(31); 
f bitt(16) aligned
```

---

```pli
a = 'hello';
b = *x';
c = addr (a);
a = 16; e = -a;
f = (d = 15);
call loa$s (*%6r2, X5zXlv1XpX2d, X5zXv, 
100, b, 32, d, a, c, e, f)
```

---

**Example**

```pli
declare buf char(80) aligned;
len fixed bin;
loa$s entry

call loa$s (buf, 80, len, *%6dX', 6, -123)
```

would set <len> to 6 and <buf> to *00123* (blank padded).

---

The entry point loa$s provides the same kind of output conversionces as loa$, except that the formatted text is returned to the user in a memory buffer, instead of being output to the user's terminal. The buffer is provided by the caller, and hence the capacity of the buffer is determined by the caller. Note that loa$ has an enhancement restriction or output volume, because its internal capacity is only 400 characters large.

---

declare loa$s entry

call loa$s (buffer, bufsize, buflen, control,
```