# OE10

## 1 SUMMARY

`OE10A` is an **editor program for maintaining files of fixed-length records,** such as programs in Fortran. It reads an ***input file*** and edits it under the control of an ***edit file*** to produce one or more ***output files.*** The files are identified to the subroutine by input-output unit numbers, which can be changed at any time by means of ***commands*** in the edit file, thus enabling a data set to be split into parts or several data sets to be concatenated. The fixed-length record that is handled by `OE10A` consists of a ***record body*** followed by a ***label field,*** which may be null but whose default length is 8 characters.

`OE10A` provides for copying, deleting, inserting, labelling, and numbering records. This may be performed selectively to include or exclude blocks of records identified by a ***select code,*** for example to enable either a Fortran 66 or a Fortran 77 version to be produced from the same input file.

`OE10A` can also maintain an internal buffer containing ***macros,*** which are groups of records (e.g. `COMMON` blocks) that replace ***macro call records.*** Macros may be nested (that is, the macro text may contain references to other macros) and may have ***parameters*** which are given numeric or text values in the macro call. Additional parameters may be defined in a macro as functions of those already known.

`OE10A` can also operate on ‘files’ in memory. Unit number 0 refers to an array supplied as an argument. A unit number greater than 99 specifies a ‘file’ whose records are accessed one by one through subroutines written by the user.

**ATTRIBUTES** — **Version:** 1.0.0. **Types:** `OE10A` **Calls:** None. **Original date:** June 1988. **Origin:** A.R.Curtis, Harwell. **Remark:** Makes `OE03` obsolete.

## 2 HOW TO USE THE PACKAGE

### 2.1 Argument list

```
CALL OE10A (JED,JIN,JOUT,LEND,ISUB,LREC,NISUB,NSET)
```

`JED` is an `INTEGER` variable initialising the unit number for the edit file, from which `OE10A` will start to read commands; `JED` is not changed by `OE10A`, even if a command causes a change of the edit file number. **Restriction:** `JED` $\geq 0$.

`JIN` is an `INTEGER` variable initialising the unit number for the input file; it is also not changed by `OE10A`. **Restriction:** `JIN` $\geq 0$.

`JOUT` is an `INTEGER` variable initialising the unit number for the output file; it is also not changed by `OE10A`. **Restriction:** `JOUT` $\geq 0$.

`LEND` is a `LOGICAL` variable which is set by `OE10A` to `.TRUE.` for a successful edit, or to `.FALSE.` if it detects any error condition.

`ISUB` is a `CHARACTER*1` array of dimensions (`LREC,NISUB`), which `OE10A` uses as an internal buffer addressed as unit 0. It is used to hold the text of macros. The user may, if he or she wishes, load the first `NSET` columns of this array with records before calling `OE10A`. It will be treated as a single file, open for reading at its start and writing at its end. If he or she does not do this, and does not call for use of unit 0, then `NISUB` can be set to 1.

`LREC` is an `INTEGER` variable specifying the logical record length of all the files (including `ISUB`). It is not changed by `OE10A`. **Restriction:** $20 \leq$ `LREC` $\leq 125$.

`NISUB` is an `INTEGER` variable specifying the second dimension of `ISUB`. It is not changed by `OE10A.` **Restriction:** `NISUB` $>$ max(0, `NSET`).

`NSET` is an `INTEGER` variable specifying that the number of records held in `ISUB` is max(0, `NSET`); it must be set by the user before entry, and is changed by `OE10A` to the final number of records in `ISUB`.

**2.2 Commands**

The edit file consists of a sequence of commands to be interpreted by `OE10A`, interspersed with records for insertion in the output. Any record not recognised as a command is inserted in the output. Commands are identified by a colon at the start of an edit file record, followed immediately by one of the command letters defined in Sections 2.2.1 to 2.2.10 and (except as specified below for `:C`, `:D`, and `:U`) at least one blank.

The general separator for parameters on a command record is a comma; this must always immediately follow the preceding parameter (otherwise subsequent ones are assumed absent), but may be followed by one or more blanks if desired. The whole command must lie within a single record body. Commas that are consecutive or have only blanks intervening indicate missing parameters; these are allowed, for example, in the `:A` or `:U` command.

Numerical parameters, indicated by *n*1, *n*2, and so on, must be unsigned integers. For some purposes, a range of character positions within a record must be supplied; this is indicated by *posrange*, and defines two numbers *n*1 and *n*2, a starting and a finishing position. If present, it must have one of the following forms:

1. 0 defines *n*1 and *n*2 to have default values;
2. *n*1 defines *n*2 = *n*1;
3. *n*1–*n*2 defines *n*1 and *n*2 separately; *n*1 must be less than *n*2 and the – sign must immediately follow *n*1.

Text parameters, for example `'`*string*`'`, each consist of the desired character string preceded and followed by the same character (e.g. a prime) that is not a comma or a number and does not occur in the string. A null string, consisting simply of two consecutive occurrences of the same delimiting character, is acceptable in some situations.

**2.2.1 The Alter command**

`:A '`*slid*`', '`*pidstr*`', '`*mcid*`',` *posrange*`, '`*cmid*`', '`*leof*`'`

allows new values to be set for various strings and character positions. Since most applications will use the default settings throughout, we describe the facilities in terms of the defaults in Sections 2.2 to 2.5 and defer explaining the alter command until Section 2.6.

**2.2.2 Copy commands**

`:C`

copies records from the input file until an end-of-file condition (the physical end of the file or the identifier `.EOF` starting in column 1) is found in the input file.

`:C` *n*1

copies *n*1 records from the input.

`:C`*x* `'`*string*`',` *posrange*

copies until the next record which contains the specified string starting in a position within the range *posrange*. The string must not be null. The default for *posrange* is the first position of the label field (the label field is the last 8 positions by default, and may be altered by the `:L` command). If the character, designated as *x*, immediately following `:C` is not blank, there is a 'copy up to but not including' action: the record containing the string is not copied to the output, but is saved in an internal one-record buffer for use on the next read from the same input file. Only one such record can be held at a time.

Examples of copy commands are:

| | |
|---|---|
| `:C` | Copy to end of file. |

| | |
|---|---|
| `:C 10` | Copy 10 records. |
| `:C 'SUM', 1` | Copy until `SUM` found starting in column 1. |
| `:C 'SUM',1-8` | Copy until `SUM` found starting one of cols 1-8. |
| `:C /SUM/, 0` | Copy until `SUM` found in label field. |
| `:CX 'SUM'` | Copy until `SUM` found in label field, excluding the card on which it is found. |

### 2.2.3 Delete commands

`:D`
`:D` *n*1
`:D`*x* `'`*string*`',` *posrange*

read and ignore records from the input file. Otherwise, the interpretation is exactly as for `:C`. In particular, if *x* is not blank, the record on which the string is found is saved for later copying or deletion.

### 2.2.4 The Exit command

`:E`

returns to the calling program, with the argument `LEND` set to indicate success. This is the only way to return indicating success. All errors, including reaching the end of the edit file, cause an immediate return.

### 2.2.5 The Identify command

`:I '`*string*`',` *posrange*`, '`*newstring*`'`

reads the next record in the input file and checks whether it contains *string* (e.g. a version date) starting in the position range *posrange*, whose default is the first position of the label field. If it fails the check, an immediate error return occurs. Otherwise, *string* is replaced by *newstring*, whose default is *string* (i.e. no replacement), and the record is output. If the replacement is too short, it is padded on the right with blanks; if it is too long, it is truncated.

An example of the identify command is:

| | |
|---|---|
| `:I '27/4/88',,'28/4/88'` | Check the date in the label field and change it. |

### 2.2.6 The Label command

`:L '`*string*`',` *n*1

causes subsequent output records to be labelled with the specified string, left justified in the label field. The string may be null. Its length must not exceed 12. If the optional parameter *n*1 (maximum: 12, minimum: length of *string*) is present, it is used as a new value for the width of the label field (which always finishes at the end of the record). If *n*1 is absent, the width of the label field is not changed. Initially, the length is 8.

An example of the label command is:

| | |
|---|---|
| `:L 'OE10'` | Change the label to `OE10`. |

### 2.2.7 The Number command

`:N` *n*1, *n*2

causes subsequent output records to be serially numbered in the label field, the initial number being *n*1 and the increment *n*2 (default *n*1, or 10 if *n*1=0). If *n*1 is absent, serial numbering is switched off. The numbers are right adjusted in the field with leading zeros filling the field. Numbering is done before labelling if both are in operation, so that the label over-writes the leading zeros of the number. Initially, numbering is switched off.

Examples of the number command are:

|  |  |
|---|---|
| `:N` | Switch numbering off. |
| `:N 10` | Number from 10 in steps of 10. |
| `:N 100, 10` | Number from 100 in steps of 10. |

### 2.2.8 The Remark command

`:R` *text*

has no effect whatsoever; it may be used to annotate the edit file, and will be listed with all other edit records read, if listing has been specified – see `:U` command in Section 2.2.10.

### 2.2.9 The Select command

`:S '`*string*`', `*n*1

specifies that blocks of output records that are identified by the select code *string* are output if they are also identified with the select value *n*1 and are suppressed if they are identified with other select values. This provides a ***conditional select*** facility, described in more detail in Section 2.5. The string specifying the select code must not be null or longer than 4 characters, and must start with the select code identifier ?. If the value *n*1 is absent, a value is found from the macro whose name is *string* (see Section 2.5.4). In either case, the value must not exceed 8000. Any existing select command with the same code is simply over-written; a value *n*1 = 0 causes this deletion without storing the new command.

`:S '`*string*`', 'C'`
`:S '`*string*`', 'D'`

specify that blocks of records identified by *string* are copied or deleted. This provides an ***absolute select*** facility, also described in Section 2.5.

Examples of the select command are:

|  |  |
|---|---|
| `:S '?D', 1` | Select records identified by `?D` and 1. |
| `:S '?D', 'C'` | Select records identified by `?D.` |
| `:S '?D', 'D'` | Delete records identified by `?D.` |

### 2.2.10 Units commands:

`:U `*n*1`, `*n*2`, ....`

sets new unit numbers for edit (*n*1), input (*n*2), listing of edit records read (*n*3), listing of output records (*n*4), and output (*n*5 onwards). The value 0 for *n*3 or *n*4 indicates no listing. Initially, the number of units is set to 5 and both *n*3 and *n*4 are set to 0; `OE10A` initializes *n*1, *n*2, and *n*5 to `JED`, `JIN`, and `JOUT`, respectively. Subsequently, when a unit command specifies 5 or more, this is the new number of units; if it specifies less than 5, the new number is 5. No more than 14 units are permitted. The default for any omitted unit number is the former value. A trailing comma indicates one default unit. If the same positive value is specified for edit and output listings, the resulting listing traces the full course of the edit process. In both the edit and output listings, each record is preceded on the same line by a count of the number of output records so far.

`:UD`

deletes the last logical file on unit 0.

`:UE `*n*1`, `*n*2`, .....`

writes a physical end-of-file mark on units *n*1,..., except in the case of unit 0. For unit 0, it causes all the macro header records to be located and a system of pointers to them to be set up.

`:UL n1, n2, .....`

writes a logical end-of-file mark (`.EOF` starting in position 1) on units *n*1,...

`:UR n1, n2, .....`

rewinds units *n*1,..., except in the case of unit 0. For unit 0, it moves the buffer's read pointer back to the start of the file it is in, or back a whole file if it points to an end-of-file record.

Examples of the units command are:

| | |
|---|---|
| `:U ,,6,6` | Change both listing units to 6. |
| `:U 12,5` | Change the edit file to 12 and the input file to 5. |

## 2.3 Unit numbers and files

Unit numbers, by which files are identified to `OE10A`, are of three kinds:

**2.3.1 The special unit number 0** refers to files held in the buffer supplied as argument `ISUB`. Except for the last, each file must be terminated by a logical end-of-file (a record containing the identifier `.EOF` starting in position 1) normally placed there by a `:UL` or `:UE` command. If `:UE` is used, the file is assumed to contain data for use by the macro facility (see Section 2.4), and is structured for that purpose; this can be done for only one file in the buffer. If unit 0 is specified as the first output file (fifth parameter) on the `:U` command, output records are sent to it exactly as read, without macro, select, label or number action, to enable such output to be re-used as input (after writing an end-of-file record by `:UL` or `:UE`). Writing is always done to the end of the file and a separate pointer is maintained for reading.

If the user sets data in `ISUB` and calls `OE10A` with `NSET` positive, the data is treated as a single file that does not have an end-of-file record. It will be ready to be read from the first record or written after the last record.

An end-of-file record during reading from unit 0 has the same effect as a logical end-of-file on a Fortran unit number (see Section 2.3.2) unless it is at the end of the last file in the buffer, when it counts as a physical end-of-file. A `:UR` (rewind) command for unit 0 moves the buffer's read pointer back to the start of the file it is in, or back a whole file if it points to an end-of-file record. A `:UD` command deletes the last file in the buffer by moving its write pointer back; if the read pointer is in this file, a subsequent read generates a physical end-of-file condition.

**2.3.2 Fortran unit numbers 1-99** are operated on by Fortran `READ` or `WRITE` instructions. During reading, if an end-of-file condition is detected as a result of obeying a `:C` or `:D` command with no arguments, it terminates the command without error; in all other cases it causes an error return after a diagnostic. After a legal logical end-of-file (`.EOF` starting in position 1), input can continue without error, on the assumption that the input file consists of concatenated logical files. After a physical end-of-file (detected by the `END=` exit from a Fortran `READ`) any attempt to continue input from the unit causes an error return.

A `:UE` command writes a file mark on a Fortran unit, by means of a Fortran `ENDFILE` instruction; a `:UL` command writes a logical end-of-file record. A `:UR` command issues a Fortran `REWIND` instruction. For suitable physical storage media, this can be used to permit re-input of a file (or a series of logical files) which have been output by `OE10A`; but unlike files on unit `0` (if it is the first output unit), records on such files will have been operated on by the macro, select, label, and number facilities during output.

**2.3.3 Unit numbers in the range 100 to 999** refer to pseudo-files whose records are accessed one by one through two user-written subroutines (`OE10Q` for input, `OE10R` for output) that are called whenever `OE10A` requires a record to be read or written. The dummy versions of these subroutines supplied with `OE10A` merely cause an end-of-file condition, but the user may replace them by subroutines to carry out any action he or she wishes – for example he or she can easily carry out editing between character arrays in memory if desired. The calling sequences are:

```
        CALL OE10Q(NU,REC,LREC,IEND)
        CALL OE10R(NU,REC,LREC,IEND)
```

`NU` is an `INTEGER` variable set before entry to the unit number on which input or output is required; it will be in the range 100 to 999, and should not be changed by either subroutine.

`REC` is a `CHARACTER*1` array of length `LREC`, which contains the record to be output by `OE10R` (which should not change it), or into which `OE10Q` should put the input record.

`LREC` is an `INTEGER` specifying the record length; except that, in the case of `OE10R,` a special meaning attaches to values 1 or 2 for `LREC`, which are illegal record lengths. A value of 1 means that a `:UE` command has called for a physical file mark to be written on the unit, while a value of 2 means that a `:UR` command has called for it to be rewound. If the user's edit file produces such commands, his or her `OE10R` subroutine must carry out suitable actions. The value of `LREC` should not be changed.

`IEND` is an `INTEGER` variable used by the subroutine to return a value to the calling program. A negative value should be returned normally, but `IEND` should be set equal to `NU+1000` by `OE10Q` to signify a physical end-of-file, where further reading from the unit would be impossible; or to `NU` to signify a logical end-of-file; however, if `IEND` is negative on return from `OE10Q,` the record is still checked for logical end-of-file. `OE10R` should set `IEND` to `NU`, or to `NU+1000`, to indicate an error condition (e.g. an array in memory, used for output, having insufficient room).

## 2.4 The macro facility

The macro facility allows a single ***macro call*** record sent to the output stream to be replaced before output by a block of ***macro text*** records from array `ISUB`, with any ***macro parameters*** which occur replaced by values from the macro call record. The macro call record carries the identifier `.MCL` in positions 1 to 4 and a 4-character ***macro name*** *mnam* in fixed positions 5 to 8. Macro calls may be nested to a depth of 10.

The macro text in `ISUB` is preceded by a ***macro header*** record, identified by the code `.MAC` in positions 1 to 4, the macro name *mnam* in positions 5 to 8, optionally followed immediately by a comma and then a list of parameter names in the remainder of the record body; this list may be continued on further records if necessary. The macro text is delimited by the next macro header record, or optionally by a special terminator record. The latter choice may improve readability; and remarks records may be included in the macro text for the same purpose.

The macro call record contains actual values for the parameters, in the same order as on the macro header; this list can also be continued on to further records if necessary. The parameter values may be numeric or text. The macro text may contain ***calculated assignment*** statements, each of which defines a new parameter by means of an expression involving existing ones; such statements may also be continued on to more than one record.

Any record in a macro starting with the combination &`REM` is treated as a remark, and is ignored when using the macro. A record starting &`END` acts as a terminator for the macro.

### 2.4.1 An outline example

A macro called `ABCD` in array `ISUB` might have the form

```
.MACABCD,  (list of parameter names)
(records containing text of macro,
     :
     referring to the parameters)
&END            (optional)
.MAC       (next macro header)
```

and a macro call record for this macro would have the form

```
    .MCLABCD,  (list of parameter values)
```

In either case, the comma must immediately follow the macro name if a parameter list is present, or be omitted if not.

### 2.4.2 Setting up a file of macros

A file on unit 0 (i.e. in array `ISUB`) can be designated for use by the macro facility by issuing a `:UE` command; or if the appropriate material has been placed in `ISUB` before calling `OE10A`, and argument `NSET` has been set to its record count, a `:UE` command for unit 0 will cause it to be organized for this purpose.

The first 4 characters of the first record of the file must be the macro header identifier `.MAC`. The identifier `.MAC` must not occur at the start of a record that is not a macro header. Every record in the file starting with `.MAC`, and with positions 5 to 8 not all blank, is a macro header, preceding a macro (which may be null). The file must be terminated by a record with `.MAC` in positions 1 to 4 and blanks in positions 5 to 8.

The `:UE` command for unit 0 causes all these records with `.MAC` to be located, and a system of pointers to them set up.

### 2.4.3 Macro calls

When a record is about to be output (other than on unit 0 if this is the first output unit specified), its first 4 characters are checked against `.MCL`. If they match, positions 5 to 8 are taken as a macro name and the corresponding macro header record is located in the macro data. If the macro name is followed immediately by a comma, the header record is scanned for parameter names, each one as it is found being entered in a dictionary, together with a value (numeric or text) that is taken from the the corresponding position on the macro call record. The parameter values are separated as usual by commas. If the last non-blank character on a call record is a comma, the list of parameter values must continue on the next record.

### 2.4.4 Macro parameters

Parameter names on a macro header record (and perhaps on continuation records) are entered as three-character groups; each is preceded by a comma, and optionally by a number of blanks; the first non-blank character after the comma signals the start of the name. A parameter name (or the macro name, if there are no parameters) not followed by a comma signals the end of the parameter list; a comma at the end of a record, or followed only by blanks, indicates that a continuation header record follows. The special three-character combinations `END` and `REM` must be avoided as parameter names.

Thus in the example of Section 2.4.1, if the macro `ABCD` had parameters named `IJK`, `LMN`, `UVW`, and `XYZ`, the full form of its header record could be

```
.MACABCD, IJK, LMN, UVW, XYZ
```

with the absence of a comma after the last parameter fixing the number as 4. A macro call record could have the form

```
.MCLABCD, 37, 5, 629, /'ERROR MESSAGE NUMBER 23'/
```

which would set the first three parameters to numeric values and the fourth to the text value `'ERROR MESSAGE NUMBER 23'.` Note that in this example the slashes are string delimiters.

### 2.4.5 Parameter references

Parameter references inside the macro take the form &..*nam* where &.. stands for a string of ampersands, and *nam* is the three-character name of a previously-defined parameter. The number of ampersands preceding the name must be sufficient to take up enough space for the longest actual value (where the length of a number is its number of digits) , i.e. at least $n-3$ to accommodate actual parameters of length $n$ characters. If too much space is allocated in this way for a parameter with a numeric value, it is padded on the left with blanks. If too much space is allocated for a parameter with a text value, subsequent characters in the record are moved to the left to reduce the space to that needed.

### 2.4.6 Numeric calculated assignments

The macro text may contain statements each defining a new parameter with a numeric value calculated as an expression. A numeric calculated assignment statement has the form

&*num* = *expression*;        (optional comment)

where *expression* involves unsigned integer constants and references to parameters having numeric values, connected by the binary operators listed in Table 1. It may continue on further records if necessary, until the terminating semicolon; but any comment must be confined to this last record of the statement. A single item (a number or parameter reference) must not be split over more than one record, and continuation after the = sign is not permitted.

The parameter name *num* (preceded by an ampersand) must occupy the first positions of the record, but spaces are allowed between items after that. All parameter references in the expression will have been replaced by their values before the expression is evaluated; the fields in which they occur must therefore be made wide enough to avoid field overflow by using sufficient ampersands.

| Operator symbol | Operation represented |
|---|---|
| + | add |
| − | subtract |
| * | multiply |
| / | integer divide |
| > or ) | take the greater of |
| < or ( | take the lesser of |

Table 1. Operators allowed in numeric calculated assignment statements.

The expression is evaluated in calculator mode, from left to right without operator hierarchy; that is, each operator takes the partially evaluated expression as its left operand, and the following value as its right operand, its result replacing the partially evaluated expression. The terminating semicolon causes assignment of this value to *num*.

An attempt to assign to a parameter with a non-null text value or a nonzero numeric value is treated as an error.

As examples of expression evaluation, the expression

20 + 6 * 3 ) 50

evaluates to 78, through the intermediate values

20 + 6 = 26, 26 * 3 = 78, max(78,50) = 78.

However, the expression

6 * 3 + 20 ) 50

evaluates to 50, through the intermediate values

6 * 3 = 18, 18 + 20 = 38, max(38,50) = 50.

### 2.4.7 Text calculated assignments

The macro text may also contain statements each defining a new parameter with a text value that is obtained by concatenation. Their form is:

&*txt* = '*string*1' '*string*2' ... ; (optional comment)

where *string*1, *string*2, etc are text strings or references to parameters, and *txt* is the name of a new parameter. The rules for &*txt* and for continuation records are the same as for numeric calculated assignments. The new parameter is given the value obtained by concatenation with no intervening blanks (but with blanks forming part of text strings preserved). A reference to a parameter with a numeric value is replaced with the text string that represents its decimal value.

Note that the assignment

```
&txt = '&num' ;
```

permits a parameter with a numeric value to be used as text so that when it is shorter than its reference, the remainder of the record moved is to the left.

An attempt to assign to a parameter with a non-null text value or a non-zero numeric value is treated as an error.

### 2.4.8 A detailed example

The full text of the macro ABCD might take the form

```
.MACABCD, IJK, LMN, UVW, XYZ
&SIZ = &IJK * &LMN ;
&ID1 = '&IJK' ;
&ID2 = 'BY ' '&LMN' ;
&FNO = '&UVW' ;
      WRITE (6,&FNO) (ARR(I),I=1,&SIZ)
&&UVW FORMAT(' ARRAY DIMENSIONS &ID1 &ID2'/
     1 1X,&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&XYZ/
     2 (1X,5E14.5))
```

The macro call record

```
.MCLABCD, 37, 5, 629, /'ERROR MESSAGE NUMBER 23'/
```

would then be replaced by

```
      WRITE (6,629) (ARR(I),I=1, 185)
  629 FORMAT(' ARRAY DIMENSIONS 37 BY 5'/
     1 1X,'ERROR MESSAGE NUMBER 23'/
     2 (1X,5E14.5))
```

Note that the parameter UVW has a numeric value and occupies the same space as its reference, being padded on the left with blanks, but that the parameters with text values, including FNO, occupy only the space needed to accommodate them. Errors are diagnosed if FNO is given a value with more than four digits or XYZ is given a value with more than 49 characters.

## 2.5 The select facility

The select facility allows single records or blocks of records to be included or not in the output under the control of select commands in the edit file. Nesting is allowed, but we delay describing the nested case until Section 2.5.3. A record in the output stream is identified as a select record by commencing with the select string of an active select command. For example,

```
?D    DOUBLE PRECISION A,B,C
```

commences with the select string ?D. A select block is delimited by special records commencing with the select string of an active select command and immediately followed by the command identifier :, with no intervening blank or comma. For example,

```
?D:C
```

```
          DOUBLE PRECISION A,B,C
          A = 0.0D0
    ?D:E
```

is a select block with select string `?D`. The `C` indicates the start of a copy block and the `E` indicates the end of the block.

Select processing is performed on all records in the output stream, including those obtained from the macro facility.

### 2.5.1 Single select records

For a single select record that is not nested within a select block (see Section 2.5.3) and is under the control of an absolute select command, the select string is replaced by blanks and the record is output for `C` or deleted for `D`. For example, the absolute select command

```
    :S '?D', 'C'
```

results in the inclusion of the above single select record.

If such a record is under the control of a conditional select command, for example,

```
    :S '?P', 1
```

a numerical value is obtained from immediately ahead of the label field. There must be no blanks between it and the label field and it must be preceded by at least one blank. This value is matched with that of the `:S` command. If they agree, the select code and numerical value are blanked from the record and it is then output; if the values do not agree, the record is omitted from the output. For example, under the control of the select command of this paragraph, the two records

```
    ?P    REAL A,B,C                                                    1
    ?P    DOUBLE PRECISION A,B,C                                        2
```

in the input would yield the single output record

```
          REAL A,B,C
```

### 2.5.2 Block select records

A record that is a leading delimiter of a select block contains `:C` or `:D` immediately after the select code. A record that is a trailing delimiter of a select block contains `:E` immediately after the select code. A delimiting record is never output.

For a block that is not nested within another block and an absolute select command, the block is output if the command and the block are both labelled `C` or both labelled `D;` it is not output if they differ.

For a conditional select command, the rest of a `:C` or `:D` block select record contains one or more numerical values, separated by commas, and a match occurs only if one of them matches the value on the select command. A non-nested `:C` block is included when the match occurs and a non-nested `:D` block is included if a match does not occur.

For example, if the input file contains the records

```
    ?P:C  1
          REAL FUNCTION SNRM2(N,SX,INCX)
          REAL SX(1),S
          S=0.0
    ?P:E
    ?P:C  2
          DOUBLE PRECISION FUNCTION DNRM2(N,SX,INCX)
          DOUBLE PRECISION SX(1),S
          S=0.0D0
    ?P:E
```

and is processed under the control of the conditional select command

```
    :S '?P', 1
```

the output is

```
          REAL FUNCTION SNRM2(N,SX,INCX)
          REAL SX(1),S
          S=0.0
```

### 2.5.3 Nesting select blocks

Select blocks may be nested to a maximum depth of 10. If a block is not selected, then all select blocks and single select records within it are not selected. The rules for selecting the outermost block are exactly the same as in the non-nested case. The rules for an inner block or single record when all the blocks within which it lies have been selected are also exactly the same as in the non-nested case.

An example of nested select blocks is given in Section 5.3.

### 2.5.4 Select macros

A select command such as

```
    :S '?DDD'
```

is a conditional select command whose numerical value is obtained from the macro file (which must in this case have been defined) as follows: the select code on the `:S` command is located as a macro name in positions 5 to 8 of one of the macro header records containing `.MAC` in positions 1 to 4; the required numerical value must be supplied on the remainder of this record (no actual macro should follow).

An example of such a macro is

```
    .MAC?DDD 1
```

## 2.6 Altering default identifiers and values

The alter command has the form

`:A '`*slid*`', '`*pidstr*`', '`*mcid*`',` *posrange*`, '`*cmid*`', '`*leof*`'`

The parameters are:

*slid* is a string of zero, one, or two characters that specifies the select identifier (initially `?`).

*pidstr* is a string of one, two, or three characters. The first character is used for the macro parameter identifier *pid* (initially &). The two further optional characters must each be a blank or a digit. The first alters the minimum number of occurrences of *pid* needed to identify a parameter name, or leaves it unchanged if blank; a zero value is not allowed. The initial value is 1. The second (that is, the third character of the string) also has no effect if blank; if zero, it permits references to any parameter of a currently active macro, and if nonzero it permits references only to those of the macro itself. Initially, references are permitted only to the parameters of the macro itself.

*mcid* is a string of one to four characters, specifying a new macro call identifier (initially `.MCL`); if it has less than four characters, it is padded on the right with blanks to make its length exactly four.

*posrange* specifies the character position range for macro names (initially 5-8).

*cmid* is a string of one to four characters that specifies the command identifier (initially `:`).

*leof* is a string of one to four characters that specifies the logical end-of-file identifier (initially `.EOF`).

If a parameter is missing, as indicated by consecutive commas, the corresponding item is left unchanged.

The macro header identifier, usually `.MAC`, actually has no default setting, but rather is automatically taken from the first record of the macro file. For example, the macro data of Section 2.4.1 would have set it to `.MAC`.

## 2.7 Error messages and returns

An error return is preceded by a diagnostic message on unit 6. The last line of the message is

`** OE10 ERROR RETURN FOR ABOVE REASON AT OUTPUT COUNT` *nnn*

The bodies of the possible messages are listed below. Where (*record*) is appended, the record causing the error is listed. Numerical values that occur in messages are indicated by *nnn* and text values by *text*. Most of the messages are self-explanatory; a brief explanation is given of any that may not be.

1 `** ILLEGAL ENTRY PARAMETERS TO OE10` *nnn nnn* `...`
(arguments `JED`, `JIN`, `JOUT`, `LREC`, `NISUB`, `NSET` are listed)

2 `** IDENTIFICATION CHECK FAILURE – RECORDS (`*edit record*`,` *input record*`)`

3 `** UNEXPECTED END FILE ON EDIT STREAM` *nnn*

4 `** MISSING OR ILLEGAL PARAMETER ON MACRO CALL RECORD (`*record*`)`

5 `** ILLEGAL NULL STRING ON EDIT RECORD (`*record*`)`

6 `** ERRONEOUS COLUMN NUMBERS ON EDIT RECORD (`*record*`)`

7 `** ILLEGAL MACRO HEADER RECORD DETECTED AT POSITION` *nnn* `(`*record*`)`

8 `** STRING TOO LONG ON EDIT RECORD (`*record*`)`

9 `** MACRO NESTING TOO DEEP`

10 `** UNEXPECTED END FILE ON INPUT STREAM` *nnn*

11 `** SELECT CODE DOES NOT START WITH CORRECT ID:`*slid* `(`*record*`)`

12 `** BLANK MHID FIELD ON FIRST MACRO HEADER RECORD (`*record*`)`

13 `** TOO MANY SELECT COMMANDS, MAXIMUM` *nnn* `(`*record*`)`
(default maximum 30)

14 `** TOO MANY MACRO HEADERS,` *nnn* `NEEDED,` *nnn* `AVAILABLE`
(default maximum 100)

15 `** NON-BLANK NAME FIELD ON LAST MACRO DATA RECORD (`*record*`)`

16 `** TOO MANY INTERNAL FILES, MAXIMUM ALLOWED IS` *nnn*
(default maximum 10)

17 `** CODE ON SELECT COMMAND NOT FOUND (`*record*`)`
(in macro headers, when no numerical value on command)

18 `** SELECT COMMAND (OR REFERENCE) HAS VALUE MISSING OR ILLEGAL` *nnn* `(`*record*`)`

19 `** ILLEGAL SYNTAX ON UNIT COMMAND` *nnn nnn* `... (`*record*`)`
(values are sequence no. of unit, old no. of units, list of units)

20 `** ILLEGAL SYNTAX ON ALTER COMMAND AT POSITION` *nnn* `(`*record*`)`

21 `** ILLEGAL SECOND MACRO FILE SPECIFIED`
(a second physical end-of-file specified for unit 0)

22 `** END OF FILE ON OUTPUT UNIT` *nnn*

23 `** CANNOT FIND MACRO NAME IN MACRO FILE` (*record*)

24 `** ILLEGAL COMMAND ON BLOCK SELECT RECORD` (*record*)

25 `** NO VALUE NUMBER ON SELECT RECORD` (*record*)

26 `** ILLEGAL NESTING OF BLOCK SELECT RECORDS` *nnn nnn nnn* (*record*)
(values are: sequence no. of active select command, pointer to and contents of top of stack; default maximum stack depth is 30)

27 `** ILLEGAL OPERAND IN CALCULATED ASSIGNMENT` (*record*)

28 `** MISSING OR ILLEGAL FIRST OPERAND IN ASSIGNMENT` (*record*)

29 `** PARAMETER VALUE TOO LONG, VALUE SO FAR IS` (*text*)
(default maximum 100 characters)

30 `** ILLEGAL OPERATOR IN CALCULATED ASSIGNMENT` (*record*)

31 `** ILLEGAL TERMINATOR IN ASSIGNMENT` (*record*)

32 `** NUMERICAL PARAMETER VALUE HAS BECOME OUT OF RANGE` *nnn*

33 `** PARAMETER REFERENCE OVERFLOWS END OF RECORD` (*record*)

34 `** PARAMETER REFERENCE TO UNDEFINED PARAMETER` (*record*)

35 `** LENGTH OF REPLACEMENT PARAMETER` *nnn* `EXCEEDS FIELD WIDTH` (*record*)

36 `** PARAMETER DICTIONARY OVERFLOW, SIZE IS` *nnn*
(default 100)

37 `** NO VALUE FOR PARAMETER TO BE ENTERED IN DICTIONARY`

38 `** PARAMETER VALUE ARRAY OVERFLOW, SIZE IS` *nnn*
(default 1000)

39 `** ILLEGAL REDEFINITION OF PARAMETER` (*record*)

40 `** ILLEGAL SYNTAX ON EDIT RECORD` (*record*)

## 2.8 COMMON blocks

There are four named `COMMON` blocks which are used to pass information between the subroutines of the package.

```
      COMMON /OE10V/ PVAL(1000), CODSE(4,30), TEXT(100)
```

holds `CHARACTER*1` arrays.

```
      COMMON /OE10W/ IVAL(100), IPTR(100), LNG(100)
     1, MSTACK(3,10), IFILE(10), ISUPNT(100)
     2, ILKSE(10), IVLSE(30), IUNIT(14)
```

holds `INTEGER` arrays.

```
      COMMON /OE10Y/ BLANK, COMMA, PID, EQLS, MINUS, LEFID(4)
     1, MCLID(4), CMDID(4), MHDID(4), SELID(2), DGT(11), OPRTR(10)
     2, NAMES(3,100)
```

holds `CHARACTER*1` variables and arrays.

```
        COMMON /OE10Z/ LIST, NNMAX, NPID, NMPVAL, LCMDID, LMHDID
       1, LMCLID, LSELID, NSTACK, LLEFID, NFILIM, NSULIM, NSELIM
       2, NOPRTR, NMTXT, NSEL, NLAB, ISSTK, IBSEL, IDSEL, KOUNT
       3, N80, N69, N70, N71, N72, N73, NPVAL, IMSTK, NAM1, NAM2
       4, IOP, NOPRND, NFILE, JFILE, NPOINT, NSUPNT, LLABL, JENDF
       5, NOMACR, NOEXPR, ALLMAC, NOPARM, LNUM, NUNIT, IBUF
```

holds INTEGER and LOGICAL variables.

## 3 GENERAL INFORMATION

**Use of common:** See §2.8.

**Other routines called directly:** OE10B, OE10C, OE10D, OE10E, OE10F, OE10G, OE10H, OE10I, OE10J, OE10K, OE10L, OE10M, OE10N, and OE10O are private subroutines. OE10P is a subroutine without arguments that initializes variables in COMMON. OE10Q and OE10R are explained in §2.3.3.

**Input/output:** See the arguments JED, JIN, and JOUT and the units command (§2.2.10).

**Restrictions:** JED $\geq 0$, JIN $\geq 0$, JOUT $\geq 0$, $20 \leq$ LREC $\leq 125$, NISUB $>$ max(0, NSET).

## 4 METHOD

The edit file is the first to be read and throughout controls the actions. A record is provisionally chosen for output if it is a record in the edit file that is not a command or is copied from the input file under the control of an :C or :I command in the edit file. For principal output to unit 0, the record is copied unchanged. For other output, the record is replaced by its expansion if it is a macro call and the resulting record or records are then subject to select action; the selected records are labelled and numbered, then listed and output. During macro expansion, any reference to a parameter is replaced by its value and any parameter assignment is executed after the parameters have been replaced by values.

Any error condition causes a message to be printed and an immediate return to be executed.

## 5 EXAMPLES OF USE

### 5.1 Single select records

The following file is a master copy of code for both the single and double precision version of a function. Records only needed for the single precision version are labelled ?S and records only needed for the double precision version are labelled ?D.

```
?S    REAL FUNCTION SNRM2(N,SX,INCX)
?D    DOUBLE PRECISION FUNCTION DNRM2(N,SX,INCX)
?S    REAL SX(1),S
?D    DOUBLE PRECISION SX(1),S
?S    S=0.0
?D    S=0.0D0
      IF(N.LE.0)GO TO 20
      IX=1
      DO 10 I=1,N
         S=S+SX(IX)**2
         IX=IX+INCX
   10 CONTINUE
?S 20 SNRM2=SQRT(S)
?D 20 DNRM2=DSQRT(S)
      RETURN
      END
```

The program

```
           CHARACTER ISUB(80,1)
           LOGICAL LEND
           JED=12
           JIN=5
           JOUT=6
           LREC=80
           NISUB=1
           NSET=0
           CALL OE10A(JED,JIN,JOUT,LEND,ISUB,LREC,NISUB,NSET)
           STOP
           END
```

will make the double precision version if given the edit data

```
      :S '?S', 'D'
      :S '?D', 'C'
      :C
      :E
```

which selects ?S records for deletion and ?D records for insertion. It produces the following output:

```
      DOUBLE PRECISION FUNCTION DNRM2(N,SX,INCX)
      DOUBLE PRECISION SX(1),S
      S=0.0D0
      IF(N.LE.0)GO TO 20
      IX=1
      DO 10 I=1,N
         S=S+SX(IX)**2
         IX=IX+INCX
   10 CONTINUE
   20 DNRM2=DSQRT(S)
      RETURN
      END
```

### 5.2 Macro

An alternative is to hold the function as a macro in the input file:

```
.MACNRM2, PRC, NME, ZRO, SRT
      &&&&&&&&&&&&&PRC FUNCTION &&NME(N,SX,INCX)
      &&&&&&&&&&&&&PRC SX(1),S
      S=&&ZRO
      IF(N.LE.0)GO TO 20
      IX=1
      DO 10 I=1,N
         S=S+SX(IX)**2
         IX=IX+INCX
   10 CONTINUE
   20 &&NME=&&SRT(S)
      RETURN
      END
.MAC
```

The program

```
      CHARACTER ISUB(80,20)
      LOGICAL LEND
      JED=12
      JIN=5
      JOUT=0
      LREC=80
      NISUB=20
      NSET=0
      CALL OE10A(JED,JIN,JOUT,LEND,ISUB,LREC,NISUB,NSET)
      STOP
      END
```

run with the edit data

```
:C '.MAC ', 1
:UE 0
:U ,,,,6
.MCLNRM2, 'DOUBLE PRECISION', 'DNRM2', '0.0D0', 'DSQRT'
:E
```

copies the macro into unit 0, formats it, switches the output to unit 6, and calls the macro with parameter values suitable for the double precision version. The output is exactly as in Section 5.1.

### 5.3 Block select

Another possibility is to group the statements needed for each precision into select blocks, labelled with 1 for single and 2 for double precision. If there is a possibility of not needing the function at all, perhaps because on some systems there is an assembly version automatically available, we might enclose the whole text in another select block. This gives the file:

```
?F:C
?P:C  1
      REAL FUNCTION SNRM2(N,SX,INCX)
      REAL SX(1),S
      S=0.0
?P:E
?P:C  2
      DOUBLE PRECISION FUNCTION DNRM2(N,SX,INCX)
      DOUBLE PRECISION SX(1),S
      S=0.0D0
?P:E
      IF(N.LE.0)GO TO 20
      IX=1
      DO 10 I=1,N
         S=S+SX(IX)**2
         IX=IX+INCX
   10 CONTINUE
?P:C  1
   20 SNRM2=SQRT(S)
?P:E
?P:C  2
   20 DNRM2=DSQRT(S)
?P:E
      RETURN
      END
?F:E
```

Running with the program of Section 5.1 and the edit data:

```
:S '?F', 'C'
:S '?P', 2
:C
:E
```

yields the same output as in Section 5.1. If `'C'` on the first record is changed to `'D'`, no output at all is obtained.