Author(s) : J. Zoll
Submitter :
Language : Fortran

Library: KERNLIB
Submitted: 02.06.1989
Revised: 01.04.1994

Utility Routines for Character String Parsing and Construction

The routines of this package analyse and manipulate Fortran `CHARACTER` strings.

Structure:

`SUBROUTINE` and `FUNCTION` subprograms
User Entry Names: `CKRACK`, `CCOPYL`, `CCOPYR`, `CCOPIV`, `CCOSUB`, `CENVIR`, `CFILL`, `CLEFT`, `CRIGHT`, `CSQMBL`, `CSQMCH`, `CLTOU`, `CUTOL`, `CSETDI`, `CSETHI`, `CSETOI`, `CSETVI`, `CSETVM`, `CTRANS`, `ICDECI`, `ICHEXI`, `ICOCTI`, `ICEQU`, `ICFIND`, `ICFILA`, `ICFMUL`, `ICFNBL`, `ICLOC`, `ICLOCL`, `ICLOCU`, `ICLUNS`, `ICNEXT`, `ICNTH`, `ICNTHL`, `ICNTHU`, `ICINQ`, `ICINQL`, `ICINQU`, `ICNUM`, `ICNUMA`, `ICNUMU`, `ICTYPE`, `LNBLNK`, `NCDECI`, `NCHEXI`, `NCOCTI`
`COMMON` Block Names and Lengths: `/SLATE/ 40`

Summary:

`CALL CKRACK`	Read integer or real number from character
`CALL CCOPYL`	Copy string, left shift allowed if overlap
`CALL CCOPYR`	Copy string, right shift allowed if overlap
`CALL CCOPIV`	Copy string, with characters front-to-back
`CALL CCOSUB`	Copy string, replacing a token by text
`CALL CENVIR`	Copy string, replacing environment variables
`CALL CFILL`	Fill
`CALL CLEFT`	Left justify
`CALL CRIGHT`	Right justify
`CALL CSQMBL`	Squeeze multiple blanks
`CALL CSQMCH`	Squeeze multiple character
`CALL CLTOU`	Convert low to up
`CALL CUTOL`	Convert up to low
`CALL CSETDI`	Set decimal integer to character
`CALL CSETHI`	Set hexadecimal integer to character
`CALL CSETOI`	Set octal integer to character
`CALL CSETVI`	Set a vector of integers to character
`CALL CSETVM`	Set a series of generated integers to character
`CALL CTRANS`	Character translation
`IX = ICDECI`	Read decimal integer from character
`IX = ICHEXI`	Read hexadecimal integer from character
`IX = ICOCTI`	Read octal integer from character
`IX = ICEQU`	Compare two strings for equality
`JX = ICFIND`	Find first occurrence, single
`JX = ICFILA`	Find last occurrence, single
`JX = ICFMUL`	Find first occurrence, multiple
`JX = ICFNBL`	Find first non-blank

continue:	`JX = ICLOC`	Locate case sensitive
	`JX = ICLOCL`	Locate case insensitive, up to low
	`JX = ICLOCU`	Locate case insensitive, low to up
	`JX = ICLUNS`	Locate unseen characters
	`JX = ICNEXT`	Delimit next word
	`JX = ICNTH`	Identify choice case sensitive
	`JX = ICNTHL`	Identify choice case insensitive, up to low
	`JX = ICNTHU`	Identify choice case insensitive, low to up
	`JX = ICINQ`	Inquire presence in a list, case sensitive
	`JX = ICINQL`	Inquire presence in a list, case insensitive, up to low
	`JX = ICINQU`	Inquire presence in a list, case insensitive, low to up
	`JX = ICNUM`	Verify numeric
	`JX = ICNUMA`	Verify alpha-numeric
	`JX = ICNUMU`	Verify alpha-numeric or underscore
	`JX = ICTYPE`	Identify type
	`NX = LNBLNK`	Find last non-blank character
	`IX = NCDECI`	Read decimal integer from character
	`IX = NCHEXI`	Read hexadecimal integer from character
	`IX = NCOCTI`	Read octal integer from character

Usage:

General Remarks:

For what follows, let the `CHARACTER` variable `LINE` contain a string of n characters assuming the following declaration:

```
CHARACTER LINE*(n),COL(n)*1
EQUIVALENCE(LINE,COL)
```

thus `COL(j)` is the `j`-th character `LINE(j:j)`. A sub-string of `LINE` is specified by `JL` and `JR`, where `COL(JL)` is the first or left-most, and `COL(JR)` is the last or right-most character.

```
COMMON /SLATE/ ND,NE,NF,NG,NUM(2),DUMMY(34)
```

returns certain search parameters, which are set by some of the routines.

The types of all variables and functions follow from the Fortran default typing convention, except that `LINE`, `COL`, and variables starting with the letters `CH` are of type `CHARACTER`.

Convention

Typing rules for data to be interpreted by `CKRACK`:

`Binary:`	String of zeros or ones prefixed by `#B` or `#b`.
`Octal:`	String of octal digits prefixed by `#0` or `#O` or `#o`.
`Hex:`	String of hexadecimal digits prefixed by `#X` or `#x`.
`Integer:`	String of decimal digits optionally prefixed by `+` or `-`.
`Real:`	`[+|-] [int] [.]  [fract] [E] [+|-] [exp]` `int, fract, exp` are strings of decimal digits, either the decimal dot or the letter `E` (or `e`) must be present.
`Double:`	`[+|-] [int] [.]  [fract] D [+|-] [exp]` the letter `D` (or `d`) must be present.

Read integer or real number from character:

```
CALL CKRACK(LINE,JL,JR,IFLD)
```

reads the number whose character representation starts with the first non-blank character at or after `COL(JL)` and ends at `COL(JR)` or at the first blank after the number (*normal termination*), or at the first character after the number which cannot be part of it (*special termination*).

`CKRACK` detects the type of the number (bit-pattern, integer, real single, real double) from its representation. The typing rules for data to be interpreted by `CKRACK` are given in the note on the previous page.

The number read is returned in `/SLATE/` in `NUM(1)` or `ANUM(1)` or `DNUM`, for which one will need:

```
REAL ANUM(2)
DOUBLE PRECISION DNUM
EQUIVALENCE (ANUM(1),NUM(1)),(DNUM,NUM(1))
```

The flag in the last parameter is normally given as zero; `IFLD` > 0 demands that single-precision real numbers be handled and returned as double precision numbers; `IFLD` < 0 demands that double-precision numbers be returned in single precision.

Apart from `NUM`, the following parameters are returned in `/SLATE/`:

`ND`	Number of numeric digits seen.
`COL(NE)`	Terminating character in the field; `NE` $=$ `JR` $+ 1$ if terminated by end-of-field.
`NF`	Type of the number read: < 0 : error code for bad data; $= 0$: the whole field is blank; $= 1$: bit pattern (binary, octal, or hexadecimal); $= 2$: integer $= 3$: single precision real; $= 4$: double precision real.
`NG`	$= 0$ for normal termination; special termination otherwise.

Copy string, left shift allowed if overlap:

```
CALL CCOPYL (CHFROM,CHTO,NCH)
```

copies `NCH` characters from `CHFROM(1:NCH)` to `CHTO(1:NCH)`; the characters are copied in order, thus the end of the target may overlap the beginning of the source.

Copy string, right shift allowed if overlap:

```
CALL CCOPYR (CHFROM,CHTO,NCH)
```

copies `NCH` characters from `CHFROM(1:NCH)` to `CHTO(1:NCH)`; the characters are copied in reverse order, thus the beginning of the target may overlap the end of the source. These two routines are useful to copy strings from or into a very large array of type `CHARACTER*1`.

Copy string, with characters front-to-back:

```
CALL CCOPIV(CHFROM,CHTO,NCH)
```

copies `NCH` characters from `CHFROM(1:NCH)` to `CHTO(1:NCH)` inverting the order of the characters such that the last becomes the first, etc.

Copy string, replacing a token by text:

```
CALL CCOSUB(CHFROM,NFR,LINE,JL,JR,CHTOKEN,CHSUB)
```

copies `CHFROM(1:NFR)` to `LINE` starting at `COL(JL)` and not going beyond `COL(JR)`, substituting each occurrence of `CHTOKEN` by `CHSUB`.
The following parameters are returned in `/SLATE/`:

`ND`	Number of characters stored;
`COL(NE)`	is the first character after the last stored;
`NF`	Non-zero if `LINE(JL:JR)` too small to receive the complete copy;
`NG`	Zero if no substitution was done, i.e. `CHTOKEN` did not occur.

Copy string, replacing environment variables:

```
CALL CENVIR(CHFROM,NFR,LINE,JL,JR,IFLAG)
```

copies `CHFROM(1:NFR)` to `LINE` starting at `COL(JL)` and not going beyond `COL(JR)`, substituting each occurrence of `${name}` by the value of the environment variable `"name"` obtained by calling `GETENVF` (Z 265); on machines running UNIX the form `"$name"` is also recognized. The handling of undefined environment variables is defined by `IFLAG`: if zero the string `${name}` is skipped from the copy; if non-zero the string is copied through as is.
The following parameters are returned in `/SLATE/`:

`ND`	Number of characters stored;
`COL(NE)`	is the first character after the last stored;
`NF`	Bit 1 is set if undefined env/v have been encountered;
	Bit 2 is set if syntax error (missing closing bracket);
	Bit 3 is set if `LINE(JL:JR)` is too small to receive the copy;
`NG`	Zero if no substitution was done.

Fill:

```
CALL CFILL(CHI,LINE,JL,JR)
```

fills `LINE(JL:JR)` with as many copies of `CHI` as possible; if $\texttt{JL} + 1 - \texttt{JR}$ is not a multiple of `LEN(CHI)` as many characters of `CHI` as necessary to fill up to `JR` will be taken for the last copy.

Left justify:

```
CALL CLEFT(LINE,JL,JR)
```

left-justifies `LINE(JL:JR)` squeezing blanks to the right.

`ND`	Number of non-blank characters in the field.
`COL(NE)`	First blank character after left-justifying (or $\texttt{NE} = \texttt{JR} + 1$ if there are no blanks).

Right justify:

```
CALL CRIGHT(LINE,JL,JR)
```

right-justifies `LINE(JL:JR)` squeezing blanks to the left.

`ND`	Number of non-blank characters in the field.
`COL(NE)`	Last blank character after right-justifying (or $\texttt{NE} = \texttt{JL} - 1$ if there are no blanks).

Squeeze multiple blanks:

```
CALL CSQMBL(LINE,JL,JR)
```

left-justifies `LINE(JL:JR)` replacing any string of several consecutive blanks by a single blank.

`ND`	Number of characters retained (vacated trailing characters, if any, are blanked).
`COL(NE)`	First blank character after (or `NE` = `JR` + `1` if there are no multiple blanks).

Squeeze multiple characters:

```
CALL CSQMCH(CHIS,LINE,JL,JR)
```

left-justifies `LINE(JL:JR)` reducing any multiple occurrence of the character `CHIS` to this character just once. `CHIS` is of type `CHARACTER*1`.

`ND`	Number of characters retained (vacated trailing characters, if any, are blanked).
`COL(NE)`	First character after the squeezed string (or `NE` = `JR` + `1` if there are no multiple occurrences).

Convert low to up:

```
CALL CLTOU(LINE(JL:JR))
```

converts lower case letters in `LINE(JL:JR)` to upper case.

Convert up to low:

```
CALL CUTOL(LINE(JL:JR))
```

converts upper case letters in `LINE(JL:JR)` to lower case.

Set decimal integer to character:

```
CALL CSETDI(INT,LINE,JL,JR)
```

writes the integer `INT` into `LINE(JL:JR)` right-justified. If `INT` is too large, the most significant characters are lost. Unused positions are not cleared to blank, so that they may be pre-loaded with default characters, such as leading zeros. (One normally clears the whole of `LINE` initially with `LINE` =' ', one could clear the substring with `LINE(JL:JR)`=' ' or preset it before calling `CSETDI`).

`ND`	Number of digits which have been set.
`COL(NE+1)`	Most significant digit set.
`COL(NF+1)`	Most significant character set. `NF` = `NE` if `INT` is positive, `NF` = `NE` − `1` if `INT` is negative and no overflow.
`NG`	= `0` normally, non-zero if field too small.

Set hexadecimal integer to character:

```
CALL CSETHI(INT,LINE,JL,JR)
```

acts like `CSETDI`, except that the hexadecimal rather than the decimal representation of `INT` is stored.

Set octal integer to character:

```
CALL CSETOI(INT,LINE,JL,JR)
```

acts like `CSETDI`, except that the octal rather than the decimal representation of `INT` is stored.

Set a vector of integers to character:

```
CALL CSETVI(NI,INTV,NBIAS,LINE,JL,JR,NCOL,IFLSQ)
```

sets the NI integers INTV(J) + NBIAS into LINE(JL:JR) in decimal representation, every NCOL columns, each right-justified within its field of NCOL − 1 columns; squeeze multiple blanks to single blanks in the resulting LINE(JL:JR) if IFLSQ non-zero. Like the other CSETxx routines, this routine does not clear unused positions to blank.

COL(NE)	Last character of the last integer stored.
NG	$=$ 0 normally, $=$ N $>$ 0 if there is not enough room to store INTV(N).

Set a series of generated integers to character:

```
CALL CSETVM(NI,INC,IGO,LINE,JL,JR,NCOL,IFLSQ)
```

acts like CSETVI, but the NI integers are IGO $+ n *$ INC, $n = 0, 1, \ldots,$ NI − 1.

Character translation:

```
CALL CTRANS(CHO,CHN,LINE,JL,JR)
```

replaces each occurrence in LINE(JL:JR) of the character CHO by the character CHN. CHO and CHN are of type CHARACTER*1.

Read decimal integer from character:

```
IX = ICDECI(LINE,JL,JR)
```

reads the decimal integer whose character representation starts at COL(JL) and stops on the first non-numeric character or at COL(JR+1), returning its value in IX. Leading blanks are ignored, a leading minus or plus sign is recognized. Note that a blank after the number, or after '+' or '-', is taken as terminator.

ND	Number of digits read ('-' or '+' do not count).
COL(NE)	Terminating character in the field; NE $=$ JR $+$ 1 if pure numeric or if the whole field is blank (in which case ND $=$ 0).
NG	$=$ 0 if the number is terminated by 'blank' or by end-of-field, non-zero otherwise.

Read hexadecimal integer from character:

```
IX = ICHEXI(LINE,JL,JR)
```

acts like ICDECI, but reads a hexadecimal rather than a decimal representation.

Read octal integer from character:

```
IX = ICOCTI(LINE,JL,JR)
```

acts like ICDECI, but reads an octal rather than a decimal representation.

Compare two strings for equality:

```
IX = ICEQU(CHA,CHB,N)
```

checks that CHA(1:N) and CHB(1:N) are identical and returns zero if so, otherwise the ordinal number of the first non-matching character is returned.

Note: this and many other routines of this package are handy when manipulating text stored in an area declared with CHARACTER TEXT(big)*1, which will explain some of the maybe unexpected calling sequences.

Find first occurrence, single:

```
JX = ICFIND(CHIS,LINE,JL,JR)
```

returns in JX the position in LINE of the first occurrence of the single character CHIS in LINE(JL:JR).

JX $=$ JR $+$ 1 if CHIS is not contained in LINE(JL:JR), or if JL $>$ JR.

NG $=$ 0 if not found, $=$ JX otherwise.

Find last occurrence, single:

```
JX = ICFILA(CHIS,LINE,JL,JR)
```

returns in JX the position in LINE of the last occurrence of the single character CHIS in LINE(JL:JR).

JX $=$ JR $+$ 1 if CHIS is not contained in LINE(JL:JR), or if JL $>$ JR.

NG $=$ 0 if not found, $=$ JX otherwise.

Find first occurrence, multiple:

```
JX = ICFMUL(CHI,LINE,JL,JR)
```

returns in JX the position in LINE of the first occurrence in LINE(JL:JR) of any one of the characters CHI(j:j), where j $=$ 1, 2, ..., n $=$ LEN(CHI).

JX $=$ JR $+$ 1 if none of CHI is found in LINE(JL:JR), or if JL $>$ JR.

ND $= j$, i.e. COL(JX) is CHI(j:j) if found.

NG $=$ 0 if not found, $=$ JX otherwise.

Find first non-blank:

```
JX = ICFNBL(LINE,JL,JR)
```

returns in JX the position in LINE of the first non-blank character in LINE(JL:JR).

JX $=$ JR $+$ 1 if LINE(JL:JR) is all blank, or if JL $>$ JR.

NG $=$ 0 if all blank, $=$ JX otherwise.

Locate, case sensitive:

```
JX = ICLOC(CHI,NI,LINE,JL,JR)
```

locates the first occurrence of the complete string CHI(1:NI) within LINE(JL:JR), it returns in JX the position in LINE of the first character of the string found. JX $=$ 0 if CHI is not contained in LINE(JL:JR).

Locate, case insensitive, up to low:

```
JX = ICLOCL(CHI,NI,LINE,JL,JR)
```

acts like ICLOC, but upper case characters from LINE are converted to lower case for the comparison.

Locate, case insensitive, low to up:

```
JX = ICLOCU(CHI,NI,LINE,JL,JR)
```

acts like ICLOC, but lower case characters from LINE are converted to upper case for the comparison.

Locate unseen characters:

```
JX = ICLUNS(LINE,JL,JR)
```

returns in JX the position in LINE of the first 'unseen' character in LINE(JL:JR), i.e. any character which will not show on the terminal, except 'blank'. JX $=$ 0 if LINE(JL:JR) does not contain unseen characters.

Delimit next word:

```
JX = ICNEXT(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-blank character in `LINE(JL:JR)` and in `NE` the position of the first blank character after `COL(JX)`, if any.

`JX` Position of the first character of the 'word'.

`NE` Position of the first 'blank' after the 'word' or `NE` = `JR` + 1.

`ND` Number of characters in the 'word'.
`JX` = `NE` = `JR` + 1, `ND` = 0 if `LINE(JL:JR)` is all blank.

Identify choice, case sensitive:

```
JX = ICNTH(CHACT,CHPOSS,NPOSS)
```

compares the character string `CHACT` against the strings stored in the character array `CHPOSS(NPOSS)`, and returns in `JX` the ordinal number of the first match found, or `JX` = 0 if no match. Neither the strings of `CHPOSS` nor of `CHACT` may contain embedded blanks: the first blank, if any, is the string terminator.
To allow matching a shortened key-word given in `CHACT` one may insert (à la VAX) a `'*'` in the text of `CHPOSS(J)` to mark the separation between the obligatory and further possible characters; a second `'*'` may be given to signal that `CHACT` may have any other characters beyond this point, this is implied if the string in `CHPOSS(J)` is not blank terminated.
For example:

```
      PARAMETER  (NPOSS=6)
      CHARACTER*8 CHPOSS(NPOSS)
      DATA CHPOSS /'del*ete ', 'add     ', 'adb*efor',
     +             'rep*lace', 'ch*ange ', 'c*ol*   '/
```

Calling the above with the following strings will give these results:

```
CHACT = 'add'       JX = 2   exact match
        'delete'         1   exact match
        'del'            1   short match
        'del  '          1
        'delphi'         0   wrong optional characters
        'deleted'        0   CHPOSS(1) is terminated
        'replaced'       4   CHPOSS(4) is not terminated
        'chan'           5   short match
        'channel'        0   wrong optional characters
        'c'              6   short match
        'columns'        6   abritrary trailing characters allowed
        'cols'           6
```

Identify choice, case insensitive, up to low:

```
JX = ICNTHL(CHACT,CHPOSS,NPOSS)
```

acts like `ICNTH` converting upper case characters from `CHACT` to lower case for the comparison, hence the `CHPOSS` array must be given in lower case.

Identify choice, case insensitive, low to up:

```
JX = ICNTHU(CHACT,CHPOSS,NPOSS)
```

acts like `ICNTH` converting lower case characters from `CHACT` to upper case for the comparison, hence the `CHPOSS` array must be given in upper case.

Inquire presence in a list, case sensitive:

```
JX = ICINQ(CHLOOK,CHHAVE,NHAVE)
```

like ICNTH this compares the character string CHLOOK against the strings stored in the character array CHHAVE(NHAVE), and returns in JX the ordinal number of the first match found, or JX = 0 if no match. Again, neither the strings of CHHAVE nor of CHLOOK may contain embedded blanks: the first blank, if any, is the string terminator.

As opposed to ICNTH, a '*' may be given in CHLOOK, but not in CHHAVE(J), to allow wild-card checking on the presence of a string in the list of CHHAVE(J). The '*' divides the string into the characters which must be present in the looked-for object of CHHAVE(J), and additional restricting characters which can be absent, but if present they must be right. Again a second '*' can be given in CHLOOK, but this is not useful, since any characters beyond the string terminator both in CHLOOK and in CHHAVE(J) are assumed to be allowed anyway, unlike as with ICNTH.

For example:

```
      PARAMETER  (NHAVE=7)
      CHARACTER*8 CHHAVE(NHAVE)
      DATA CHHAVE /'apo     ', 'apol    ', 'apollo  ', 'irs6000 ',
     +             'decra1  ', 'decra2  ', 'decra3  '/
```

Calling the above with the following strings will give these results:

```
CHLOOK = 'apo'        JX = 1
         'apo*'            1
         'ap*ollo'         1
         'ap*'             1
         'ap'              0
         'apol'            2
         'apoll'           0
         'apoll*'          3
         'ir*s60'          4
         'ir*s70'          0
         'dec*'            5
         'dec*ra'          5
         'dec*ra*'         5
         'dec*ra3'         7
```

In spite of the similarity, the operations of ICINQ and ICNTH serve really very different functions:

With ICNTH we have a key word CHACT which we try to identify; CHPOSS(N) is most likely a fixed table built into the program which gives the possible key words and allowed abbreviations à la VAX. The return value from ICNTH tells us which key word we have.

With ICINQ we inspect a table CHHAVE(N), which most likely has been built up at execution time, to see whether it contains an object according to the specifications given in CHLOOK. The interesting thing about the return value from ICINQ is mainly whether it is zero or not, the position of the found object in the table is of secondary importance.

Inquire presence in a list, case insensitive, up to low:

```
JX = ICINQL(CHLOOK,CHHAVE,NHAVE)
```

acts like ICINQ converting upper case characters from CHLOOK to lower case for the comparison, hence CHHAVE must be held in lower case.

Inquire presence in a list, case insensitive, low to up:

```
JX = ICINQU(CHLOOK,CHHAVE,NHAVE)
```

acts like `ICINQ` converting lower case characters from `CHLOOK` to upper case for the comparison, hence `CHHAVE` must be held in upper case.

Verify numeric:

```
JX = ICNUM(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-numeric character in `LINE(JL:JR)`; blanks are ignored. Note that `'+', '-'` or `'.'` are not considered numeric.

`JX` = `JR` + `1` if `LINE(JL:JR)` is all numeric.

`ND` Number of digits seen in `LINE(JL:JX-1)`.

`NG` = `0` if all numeric, = `JX` otherwise.

Verify alpha-numeric:

```
JX = ICNUMA(LINE,JL,JR)
```

returns in `JX` the position in `LINE` of the first non-alphanumeric character in `LINE(JL:JR)`; blanks are ignored. Note that `'+', '-'` or `'.'` are not considered alpha-numeric.

`JX` = `JR` + `1` if `LINE(JL:JR)` is all alpha-numeric.

`ND` Number of alpha-numeric characters seen in `LINE(JL:JX-1)`.

`NE` Position of the first numeric character, = `0` if none.

`NF` Position of the first alphabetic character, = `0` if none.

`NG` = `0` if all alpha-numeric, = `JX` otherwise.

Verify alpha-numeric or underscore:

```
JX = ICNUMU(LINE,JL,JR)
```

acts like `ICNUMA`, but the character ”underscore” is considered alphabetic.

Identify type:

```
JX = ICTYPE(CHIS)
```

returns in `JX` the type of the single character `CHIS`:

`JX` = `0` : *Unseen*, i.e. a character not showing on an ASCII terminal.
= `1` : Anything else.
= `2` : Numeric character.
= `3` : Lower case character.
= `4` : Upper case character.

Find last non-blank character:

```
NX = LNBLNK(CHV)
```

returns the non-blank length of the string in `CHV(1:LEN(CHV))`, i.e. the characters `NX+1` to `LEN(CHV)` are all blank. (Note that this is an intrinsic function of several compilers.) If there are many trailing blanks the routine `LENOCC` of `M507` is faster; depending on the machine the break-even point with `LENOCC` is around 25 trailing blanks.

Read decimal integer from character:

```
IX = NCDECI(CHTEXT)
```

acts like `ICDECI`, with `JL` $=$ `1` and `JR` $=$ `LEN(CHTEXT)`.

Read hexadecimal integer from character:

```
IX = NCHEXI(CHTEXT)
```

acts like `ICHEXI`, with `JL` $=$ `1` and `JR` $=$ `LEN(CHTEXT)`.

Read octal integer from character:

```
IX = NCOCTI(CHTEXT)
```

acts like `ICOCTI`, with `JL` $=$ `1` and `JR` $=$ `LEN(CHTEXT)`.

•