
RC9000-10/RC8000

SW8585 Compiler Collection

ALGOL Text Procedures

User's Guide

RC Computer

Keywords:

RC9000-10, RC8000, ALGOL, Text Handling, Procedures

Abstract:

This manual describes a set of procedures intended for handling of text strings in ALGOL programs

Date:

January 1989

PN: 991 11289

Copyright © 1988, Regnecentralen a · s/RC Computer a · s
Printed by Regnecentralen a · s, Copenhagen

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without prior notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and shall not be responsible for any damages caused by reliance on any of the materials presented.

Table of Contents

1. Introduction.....	1
2. Procedures.....	2
2.1 Putnumber.....	2
2.2 Putfixed.....	4
2.3 Putchar.....	5
2.4 Puttext.....	6
3. Get Procedures.....	8
3.1 Getnumber.....	8
3.2 Getfixed.....	12
3.3 Getchar.....	13
3.4 Gettext.....	13
4. Text Comparison.....	16
4.1 Comparetext.....	16
5. Error Handling.....	18
5.1 Putgeterror.....	18
5.2 Alarm Messages.....	19
6. Programming Examples.....	20
6.1 Concatenation of Text Strings.....	20
6.2 Extraction of a Substring.....	20
6.3 A Text Compared with a String.....	21
6.4 Conversion of String Format.....	21
6.5 Conversion of Characters in a String.....	21
6.6 Reading of Punched Cards.....	22
Appendix A. References.....	25



1. Introduction

This manual describes a set of procedures intended for handling of text strings in ALGOL programs.

Text strings may be packed into arrays of type boolean, long or real by means of the procedures described in chapter 2.

Chapter 3 describes some procedures for extracting a substring, a single character, or a numeric value from a textarray.

Chapter 4 describes how to compare two text strings.

Chapter 5 contains a general description of error handling and a list of alarm messages.

Finally, chapter 6 shows some programming examples.

The text handling procedures are designed for use in ALGOL programs, but they may also be used directly in a FORTRAN program if no string parameters are specified. If an external ALGOL procedure using the text procedures is called from a FORTRAN program, long text string references must be avoided.

2. Put Procedures

The put procedures putnumber, putfixed, putchar and puttext are intended for generating a text string in an array in the same way as write would output the characters through a zone.

The text may be created as a sequence of 12-bits characters in a boolean array, or as an ordinary string of 8-bits characters in a long or real array.

When the text is generated in a boolean array, one character is stored in each halfword. The character is placed in the 8 least significant bits, and the remaining bits are set equal to zero.

The characters generated by the put procedures may use the full ISO-low and ISO-high alphabet, i.e. $0 \leq \text{char} \leq 255$.

Neither replacechar nor outtable has any influence on the generating of characters by the put procedures, but a character conversion table may be stated as a parameter in the procedure call, giving the same possibilities as a call of outtable before write.

However, the possibility of using different characters for leading space, space in number, and ending space by means of replacechar does not exist in putnumber/putfixed.

2.1 Putnumber

This integer procedure converts a numeric value into a text string in an array, in the same way as a numeric parameter is converted by write.

Call:

```
put_number (dest, pos, [ conv, ] [ layout, ] num)
```

```
put_number  
(return value, integer).
```

Shows the result of the conversion:

```
>=0: no. of characters inserted into dest,  
--1: conversion stopped because dest is full,  
--3: layout exceeded, see below.
```

dest

(call and return value, array of type boolean, long, or real).
The converted text string is inserted into this array, see below.

pos

(call and return value, integer).

As call value pos denotes the character position into which the first character is to be inserted. Pos = 1 means first character in dest(1), so pos may be negative if lower limit of dest is less than 1.

The return value of pos denotes the character position after the last inserted character, so that pos is ready for a following call of a put-procedure. The contents of dest outside this character position interval are unchanged.

conv

(call value, one-dimensional integer array, optional parameter). If this parameter is specified, it is supposed to contain a character conversion table, by which all characters are converted before insertion into dest, see below.

layout

(call value, string, optional parameter).

An ordinary ALGOL layout with the same format and meaning as in write. If this parameter is omitted, the same standard layout is used as in write (<<_d> or <<_dd.dddd>).

num

(call value, integer, long or real). The value of this parameter is converted into character form according to the layout, in the same way as by write.

Function:

The characters generated by put_number are inserted into dest depending on the type of this array. For an array of more than one dimension the lexicographical ordering is used.

If dest is a boolean array, the characters are generated as so-called 12-bits characters, with one character per array element. The characters are stored in the 8 least significant bits, whereas the remaining bits are set to zero. So, all character values in the range $0 \leq \text{char} \leq 255$ may be used.

In an array of type long or real, and in a zone record, the characters are stored as 8-bits characters with 3 characters per word (24 bits). Here, too, the full character range 0-255 is allowed.

If dest is a zone, a zonerecord must exist, i.e. recordlength > 0.

Character conversion:

In all put procedures character conversion is totally independent of a possible call of outtable and/or replacechar, and the value of outindex is

insignificant as well. If no conv parameter has been specified, the characters are never converted.

If a conv parameter has been specified, each character is converted before insertion into dest, according to the following algorithm:

```
new_char: = conv (char) extract 12;
```

If new_char > 255 or char is not a legal index in conv, no character is generated.

Layout:

A possible layout is specified and used in the same way as in write, with the exception of a layout containing "ending blanks". If the number value exceeds the range of such a layout, write will print a * as error symbol in the last position. Put number will instead insert as many characters as possible within the layout, and then return with the procedure value -3.

Errors:

Too many parameters, or illegal parameter types will cause a run-time alarm (param <paramno>).

A call value of pos outside the index range of dest will cause a run-time alarm (charpos <pos>).

If dest is filled up before the number conversion is finished, the character conversion stops, and the procedure returns with the procedure value -1. In this case the return value of pos denotes the first non-existent character position in dest. So a possible new call of a put procedure (with pos unchanged) will give the charpos alarm above.

A negative procedure value may be changed into a run-time alarm by means of the standard variable 'put_get_error'.

2.2 Putfixed

This integer procedure converts an integer or long value into a text string in an array, giving the possibility of inserting a decimal point in a fixed position (like writeint).

Call:

```
put_fixed (dest, pos, [ conv, 9 [layout,] num)
```

```
put_fixed
dest
pos          see put number.
conv
layout
```

num
(call value, integer or long). The value of this parameter is converted into character form according to layout in the same way as writeint. I.e. if the layout contains a decimal point, a point is inserted into the generated textstring in the corresponding position.

Errors:

As put_number.

2.3 Puchar

This integer procedure inserts a single character into one or more positions of a text string in an array.

Call:

```
put_char {dest, pos, [ conv, ] char [ , rep ] }
```

put_char
(return value, integer).

Denotes the result:

>=0: no. of characters inserted into dest

--1: too few characters inserted, dest is full.

dest

pos

see put number

conv

char

(call value, boolean or integer).

The rightmost 8 bits of char are considered a character, which may be converted by a possible conv parameter. The resulting character is inserted into dest in the position denoted by pos (and possibly following positions, see rep). The contents of the remaining bits of char are insignificant.

rep

(call value, integer, optional parameter). This parameter specifies how many times char is going to be inserted into dest. If the parameter is omitted, char is inserted just once. If rep <= 0, nothing is inserted.

Errors:

As put number.

2.4 Puttext

This integer procedure moves a text string into an array from a string parameter or from another array, possibly after conversion of each single character.

Call:

```
put_text (dest, pos, [ conv, ] text [, length] )
```

```
put_text  
(return value, integer).
```

Denotes the result:

```
>=0: no. of characters inserted into dest,  
--1: transfer stopped, because dest is full,  
--2: transfer stopped, because text is exhausted.
```

```
dest  
post      see put number  
conv
```

```
text  
(call value, string or array of type boolean, long, or real).  
The characters are transferred one by one from text into dest, to be  
stored from character position pos and on (after possible conversion).
```

```
length  
(call value, integer, optional parameter). This parameter controls the  
transfer as described below.
```

Function:

The contents of text are transferred one character at a time into dest.

If text is an array of more than one dimension, the lexicographical ordering is used. The first character transferred is the first character of text(1). If text is a zone, a zone record must exist, i.e. recordlength > 0.

The array is supposed to contain a simple string of characters. A long text string reference (as described in ref.[1]) cannot occur, as it would be interpreted as characters.

When text is a boolean array, the characters are taken one from each array element, using only the 8 least significant bits.

From arrays of type long or real the characters are taken as 8-bits characters with 3 characters per word (24 bits).

If the text parameter is of type string, this must be a simple string, i.e. a string constant (<:...:>), or a simple variable referring to a string constant (a:= long <:...:>). From a string expression like "string ra(increase(i))" only the characters in the first array element would be transferred.

Length:

The length parameter controls the transfer in the following way:

If length is not specified or equals zero, characters are transferred until either a null-character is read from text or text is exhausted. The null-character is not transferred.

If length > 0, exactly length characters are transferred regardless of possible null-characters. So the procedure works as a sort of "text-tofrom". However, this is senseless when text is of type string, in which case length > 0 is handled as length < 0.

When length < 0, at most abs length characters are transferred, the transfer stopping if a null-character is read from text.

The character transfer may stop prematurely if text is exhausted, or dest becomes full. In this case puttext returns a negative procedure value.

See also the description of procedure gettext, which transfers a text controlled by character classes instead of by null-characters.

Errors:

As put number.

Furthermore, the run-time alarm "index 1" will occur, if the text parameter is an array of which the element number one does not exist.

And the run-time alarm "string 0" occurs, if the text parameter is of type string, but contains an impossible long text string reference. This alarm will also occur when a legal long text string reference is used from a FORTRAN program (because of the differences between the ALGOL and FORTRAN running systems).

3. Get Procedures

The get procedures `getnumber`, `getfixed`, `getchar`, and `gettext` are intended for reading from a text string in an array, instead of reading through a zone.

The reading may be controlled by character classes in the same way as in the ordinary read procedures (`read`, `readstring`), or a fixed number of characters may be read by specifying a "layin" for number reading or a length for text reading.

A possible call of `intable` has no influence on the get procedures, but a character conversion table may be specified directly as an optional parameter in the procedure call.

The array from which the characters are read may contain 12-bits characters in a boolean array, or 8-bits characters in an array of type `long` or `real`.

If a proper conv table is specified, characters within the full ISO-low and ISO-high alphabet may be read, i.e. $0 \leq \text{char} \leq 255$.

3.1 Getnumber

This integer procedure converts a (part of a) text string in an array into a numeric value.

Call:

```
get_number (source, pos, [ conv, ] [ layin, ] num)
```

`get_number`
(return value, integer).

Denotes the result of the conversion:

- >=0: char class shift 12 + char value extract 12 of the last read character.
- 4: conversion stopped because source exhausted.
- 6: syntax error or number too great.

source

(call value, array of type boolean, long, or real).

This array is supposed to contain the string of characters to be converted, see below.

pos

(call and return value, integer).

As call value pos denotes the character position from which the first character is to be read. Pos=1 means first character in source(1), so pos may be negative if lower limit of source is less than 1.

The return value of pos denotes the character position after the last read character, so that pos is ready for a following call of a get procedure.

conv

(call value, one-dimensional integer array, optional parameter).

If this parameter is specified, it is supposed to contain a character class table, see below. If the parameter is omitted, the ISO standard character classes are used.

layin

(call value, string, optional parameter).

This parameter specifies, how the contents of source should be interpreted, see below.

num

(return value, integer, long, or real).

The number read from source is returned in this parameter, which however remains unchanged, if the result value of getnumber is negative.

Function:

Characters are read from source depending on the array type, and converted into a numeric value which is returned in the parameter num. If source is a zone, a zone record must exist, i.e. record length > 0. For an array of more than one dimension the lexicographical ordering is used.

From a boolean array characters are read as 12-bits characters, one from each array element. Only the 8 least significant bits are used:

char:= source(index) extract 8

From an array of type long or real characters are read as 8-bits characters, three characters from each word. A long text string reference cannot occur, as it would be interpreted as characters.

All character values in the interval $0 \leq \text{char} \leq 255$ are allowed, but if the ISO standard character classes are used, character values > 127 will give an index alarm, as will a character value which is not a legal index in a specified conv table.

Character conversion:

A possible call of intable has no influence on the character conversion, nor has the value of tableindex.

If a conv parameter is specified, the array must contain a character class table, each element composed of a character class and a character value:

`conv(char) = class shift 12 + value extract 12`

Where class is an integer $0 \leq \text{class} \leq 4095$, and value is an integer $-2048 \leq \text{value} \leq 2047$. (See the description of intable, ref. [2]).

As character class = 1 (shift character) is useless when table- index is not used, such characters are considered blind (like class = 0).

A possible EM character has no special meaning in the get procedures, but is handled like any other terminator.

Layin:

The layin parameter is an ALGOL layout, allowing only a part of the possibilities given in write. A layin is composed of

`<first letter> <d's> <dec.point> <d's> <fill>`

`<fill>`:

If `<fill>` is non-empty, the layin is called an "open layin", and the reading is controlled exclusively by character classes (like procedure read). In this case the number of d's in the layin is insignificant.

When `<fill>` is empty, the layin is a "closed layin", which means that exactly so many characters are read as indicated by the total number of positions in the layin. This may be useful for reading position controlled data, such as punched cards. The data field may contain leading and/or ending delimiters (character class ≥ 6), whereas any delimiter between digits is considered a data error.

Blind characters (class ≤ 1) are skipped, and they are not counted.

`<first letter>`:

The first letter of the layin must be either b or d.

When `<first letter>` is b, the field may be empty, in which case the num parameter remains unchanged. If the layin is open, reading stops immediately after the first delimiter. If the layin is closed, an empty field may contain all delimiters.

When `<first letter>` is d, a closed layin requires the field to contain at least one digit. If the layin is open, leading delimiters are skipped like in procedure read.

`<dec.point>`

A possible decimal part in the layin is allowed, but its only significance in putnumber is to tell the field length when the layin is a closed one.

If no layin parameter is specified, the open layin <<d_> is used.

For illustration of the effect of various layins, see the example below.

Number syntax:

The number is read according to the syntax of ALGOL numbers (see ref. [1]), i.e. a number may contain a sign, decimals and/or an exponent part, which are not specified in the layin.

Example:

Suppose long array la contains the following string from index 1 and on:

ab1234x567

The statement

```
get_number (la, 1, string layin, num)
```

will read the following values for various layins:

Open layins:		last char.
<	num unchanged	a
<<d >	num:= 1234	x
<<ddd.ddd >	num:= 1234	x

Closed layins:

<	num unchanged	a
<<bd>	- -	b
<<bdd>	num:= 1	1
<<d>	syntax error	a
<<dddd>	num:= 12	2
<<ddd.dd>	num:= 1234	4
<<ddddddd>	num:= 1234	x
<<ddddddd>	syntax error	5

Errors:

Too many parameters, or illegal parameter types will cause the run-time alarm "param <paramno>".

A call value of pos outside the index range of source will cause the run-time alarm "charpos <pos>".

A layin containing illegal layout components will cause the run-time alarm "layin 0".

If a character value is read which is not a legal index in a specified conv table (or in the standard ISO table), the run-time alarm "index <char value>" occurs.

If source is exhausted before the number reading is finished, the procedure returns with the procedure value -4, and the num parameter remains unchanged. In this case the return value of pos denotes the first non-existent character position in source. So a possible new call of a get procedure (with pos unchanged) will give the charpos alarm above.

When a syntax error is found in source, or when the number is too great for the actual num parameter, the return value of the procedure is -6 and the num parameter remains unchanged. If the layin was a closed one, the specified number of characters have always been read. Otherwise reading continues up to and including the first delimiter.

A negative procedure value may be changed into a run-time alarm by means of the standard variable 'put_get_error'.

3.2 Getfixed

This integer procedure converts a (part of a) text string in an array into a numeric value to be stored in an integer or long variable. A decimal fraction may be scaled i.e. stored as an integral number after multiplication by a power of ten.

Call:

```
get_fixed (source, pos, [ conv, ] [ layin, ] num)
```

```
get_fixed
source      see get number
pos
conv
```

```
layin
as get_number, see also below.
```

```
num
(return value, integer or long).
```

The number read from source is returned in this parameter, which however remains unchanged when the result value of get fixed is negative.

Function:

The procedure works like get number except for handling of decimals in the layin.

In get fixed the decimals in the layin are significant in an open as well as in a closed layin. If the layin contains "dec" decimals, the number read from source is multiplied by 10^{**dec} . So the values

12, 12.3, 12.345, -12.345

will, when read by the layin < <dd.dd > be stored as

1200, 1230, 1235, -1235

Errors:

As get number.

3.3 Getchar

This integer procedure fetches one single character from a text string in an array.

Call:

`get_char (source, pos, [ conv, ] char)`

`get_char`
(return value, integer).

Contains char class shift 12 + char_value extract 12 after possible conversion of the character.

source

pos see `get_number`

conv

char

(return value, integer).

The character value (after possible conversion) of the character in source denoted by pos. Even a blind character (class ≤ 1) will be returned by `getchar`.

Errors:

Only the run-time alarms "charpos <pos>" and "index <charvalue>" (as described by `get_number`) can occur.

3.4 Gettext

This long procedure moves a text string from one array to another, possibly with conversion of each character.

Call:

```
get_text (source, pos, [ conv, ] text [, length] )
```

```
get_text
```

```
(return value, long).
```

Denotes the result of the transfer:

>= 0: the value is a combination of two integers:

```
no_of_chars shift 24 + last_char.
```

no_of_chars is the number of characters packed into the new textstring. last_char is

```
class shift 12 + value extract 12
```

(after possible conversion) of the last character read from source.

= -4: Transfer stopped, because source is exhausted.

= -5: Transfer stopped, because text is full.

source

pos see get number

conv

text

(call value, array of type boolean, long, or real). The text string is transferred from source into this array, after possible conversion of each character.

length

(call value, integer, optional parameter).

Denotes how many characters are to be transferred, see below.

Function:

The contents of source, from character position pos and on, are transferred one character at a time into text, where they are stored from index 1 and on. If text is a zone, a zonerecord must exist, i.e. recordlength > 0. If text is an array of more than one dimension, the lexicographical ordering is used.

The characters are stored in text according to the type of this array. If text is a boolean array, the characters are inserted as 12-bits characters, one in each array element. The characters are stored in the 8 least significant bits, whereas the remaining bits are set to zero.

In an array of type long or real the characters are stored as 8-bits characters with 3 characters per word (24 bits).

Character conversion:

If a conv parameter is specified, each character read from source will be converted. The character value is found by

new char: = conv (char) extract 8.

(See also getnumber).

Length:

The length parameter controls the transfer in the following way:

If length is not specified or equals zero, characters are transferred in the same way as by readstring: Leading delimiters (class ≥ 7) are skipped, and the following characters are packed into text until a final delimiter is recognized. This character has been read, but is not stored in text. All blind characters (class ≤ 1) are skipped.

If length > 0 , exactly length characters are transferred regardless of the character class. Even blind characters are transferred, so the procedure works as a sort of "text-tofrom". This is also suitable for extracting a substring of a text.

If length < 0 , at most abs length characters are transferred, the transfer stopping if a terminator (class ≥ 8) is read from text. A possible terminator has been read, but is not packed into text. Blind characters are neither transferred nor counted.

After the last character a NULL-character is inserted in text. If text contains 8-bits characters the remaining part of the last word (of 3 characters) is set equal to zero. The return value no of chars in the procedure value does not count this NULL-character.

The character transfer may stop prematurely if source is exhausted or text becomes full. In this case gettext returns a negative procedure value, and no final NULL-character is inserted in text.

Errors:

As getnumber.

Besides the run-time alarm "index 1" will occur, if the text parameter is an array of which the element number one does not exist.

4. Text Comparison

Two text strings may be compared by the procedure `compare_text`, when they are both stored as 8-bits characters in long arrays, starting in first character of a word. These start conditions are easily obtained by means of `gettext`.

4.1 Comparetext

This integer procedure compares two text strings.

Call:

`compare_text (text1, text2, chars)`

compare_text
(return value, integer).

The result of the comparison:

- 0: text1 = text2
- -1: text1 < text2
- +1: text1 > text2

text1, text2

(call values, long arrays).

Each of the two arrays is supposed to contain a string of 8-bits characters starting at the first character in the element with index 1.

chars

(call and return value, integer).

The call value denotes the maximum number of characters to be compared. The call value may be ≤ 0 meaning all characters. The return value of `chars` is the field address of the word at which the comparison stopped.

Function:

The contents of `text1` and `text2` are compared word by word (24 bits). The comparison involves at most 'chars' characters (stored in

(chars+2)//3 words), but the comparison stops as soon as a difference between the strings has been encountered, or the last character of a word contains a null-character, or when the upper index of one of the arrays is reached.

Example:

If the long arrays la1 and la2 contains the following strings

la1 = <:abcdefghij:>

la2 = <:abcd:>

then the comparison

compare text (la1, la2, 4)

will give the result

0 (equal),

but the comparison

compare text (la1, la2, 0)

will give the result

1 (la1 > la2).

5. Error Handling

Parameter errors in a call of a put or get procedure (e.g. illegal parameter type) will always cause a run-time alarm.

An error in the contents of a text parameter is handled as a data error, causing a negative return value of the procedure. This may however be changed into a run-time alarm by means of the standard variable 'put_get_error'.

5.1 Putgeterror

This integer variable controls the handling of data errors by the put and get procedures.

Each possible negative procedure value corresponds with a bit in put_get_error. If this bit has the value 1 the procedure will provoke a run-time alarm instead of returning, in case the corresponding data error occurs. The alarm may be trapped, so that data errors can be handled centrally in the program.

The possible data errors and the corresponding bits in put_get_error are listed below:

proc val	putgeterror bit	alarm message	explanation	used by procedure
-1	1 shift 1	put full -1	dest array full	all put procedures
-2	1 shift 2	put exh. -2	text array exhausted	puttext
-3	1 shift 3	p.layout -3	layout exceeded	putnumber/putfixed
-4	1 shift 4	get exh. -4	source array exhausted	all get procedures
-5	1 shift 5	get full -5	text array full	gettext
-6	1 shift 6	getvalue -6	syntax or num too great	getnumber/getfixed

The default value of put_get_error is 0, i.e. no run-time alarms on data error.

5.2 Alarm Messages

The following parameter alarm messages may occur from the put and get procedures. The alarm address is in any case the name "text procs"

alarm text	explanation	procedure
charpos <n>	The character position specified by pos (2nd parameter) is outside dest/source. <n> shows the position.	all put and get procedures
index 1	The array element with index 1 of the text parameter (3rd or 4th parameter) does not exist.	puttext or gettext
index <n>	A character value outside the used alphabet is read from source. <n> is the character value.	all get procedures
layin 0	A layin containing illegal layout components (sign, first letter f or z, exponent part, leading spaces, or space between d's) is specified.	getnumber or getfixed
string 0	The string parameter contains an impossible long text string reference. Or a long text string parameter is used in an external ALGOL procedure, which is called from a FORTRAN program.	puttext

6. Programming Examples

This section shows some examples of use of the put and get procedures.

6.1 Concatenation of Text Strings

A number of different texts may be concatenated into one text string by means of the put procedures. The following program part forms a text string consisting of a string, a number and some single characters.

```
begin
  integer p, num; long array la (1:10);
  ...
  p:= 1;
  puttext (la, p, <:this is a text:>);
  putchar (la, p, '- ',3);
  putnumber (la, p, <<d>, num);
  putchar (la, p, 0);
  ...
  write (out, la);
```

If num = 123 the write statement will produce the following output:

```
this is a text---123
```

6.2 Extraction of a Substring

A part of a text string may be extracted by means of gettext:

```
if gettext (la, 5, subla, 4) < 0 then error;
```

This call will extract a substring consisting of 4 characters from position 5, 6, 7, and 8 of la.

6.3 A Text Compared with a String

As the procedure `comparetext` requires array parameters the contents of a text array can be compared with a string constant in this way:

```
puttext (compare, 1, <:...:>);
if comparetext (la, compare, 0) = 0 then
begin <*equal*>
...
end;
```

6.4 Conversion of String Format

A text string may be converted from 8-bits characters to 12-bits characters (and vice versa) by means of `puttext` or `gettext`:

```
length:= ...; <*no. of chars, > 0 *>
puttext (ba, 1, la, length); <*from 8 to 12*>
gettext (la, 1, ba, length); <*from 8 to 12*>
```

and

```
puttext (la, 1, ba, length); <*from 12 to 8*>
gettext (ba, 1, la, length); <*from 12 to 8*>
```

Whether you should use `puttext` or `gettext` depends on the wanted startposition in from-array or to-array, as a startposition can be specified only for the first parameter.

6.5 Conversion of Characters in a String

The character values in a text string may be converted by means of `puttext` or `gettext` using a conversion table:

```
isotable (conv);
<*change the wanted characters in conv*>
...
length:= ...; <*no. of chars > 0 *>
puttext (new la, 1, conv, la, length);
```

or

```
gettext (la, 1, conv, new la, length);
```

(See section 6.4 concerning whether to use `puttext` or `gettext`).

6.6 Reading of Punched Cards

The get procedures are particularly suited for reading of punched card images. Each card consisting of 81 characters (80 characters followed by a NL character) may be read by a call of inrec6 and subsequently interpreted by means of the get procedures.

Suppose some of the cards contain the following information:

column	datatype	contents
1- 3	text	cardtype = emp
4- 7	num	employee number (4 digits)
8-36	text	employee name (29 chars)
37-65	text	employee address (29 chars)
66-68	num	department
69-77	num	salary (with 2 decimals)
78-79	num	rate of taxation
80-80	char	status

The card deck is terminated by a card with cardtype = fin.

These cards may be read and used for creation of employee records by the program listed on the following pages.

*p=algol list.yes

c

```

1 begin
2   zone card (128,1,stderror),
3     emp (128,1,stderror);
4   long array    cardtype(1:1), err(1:15);
5   real array    rec(1:20);
6   integer       p, cardno, i;
7   integer field emp_no, tax_pct, department,
8     emp_length, status;
9   long field    salary;
10  long array field emp_name, emp_addr;
11
12  <*init field variables*>
13  emp_length:= 2;
14  emp_no    := 6;
15  department:= 8;
16  emp_name  := 8;
17  emp_addr  := 28;
18  salary    := 52;
19  tax_pct   := 54;
20  status    := 56;
21
22  open (card, 4, <:cards:>, 0;
23    <*cards have been loaded into a disc file*>
24  opensq (emp, <:empfile:>, 0, 3);
25
26  trap (error_card);
27  put_get_error:= -1; <*alarm on all text errors*>

```

```

25  cardno:= 0;
26
27  next_card:
28  cardno:= cardno + 1;
29  inrec6 (card, 54);  <*81 chars = 54 halfwords*>
30  gettext (card, 1, cardtype, 3);
31
32  if cardtype(1) = long<:emp:> then
33  begin <*create employee record*>
34    rec.emp_length:= 56;
35    p:= 4;
36    getnumber (card, p, <<ddd>>, rec.emp_no);
37    gettext (card, p, rec.emp_name, 29);
38    gettext (card, p, rec.emp_addr, 29);
39    getnumber (card, p, <<ddd>>, rec.department);
40    getfixed (card, p, <<dddddd.dd>>, rec.salary);
41    getnumber (card, p, <<dd>>, rec.tax_pct);
42    getchar (card, p, rec.status);
43    outvar (emp, rec);
44    goto next_card;
45  end
46  else
47  if cardtype(1) = long<:fin:> then
48    goto finis
49  else
50    <*other card types may be inserted here*>
51    ;
52
53  error_card:
54  if alarmcause extract 24 = -11 then
55    writealarm (out) <*see ex.1 of getalarm in ref.2*>
56  else
57  begin
58    put_get_error:= 0;
59    write (out, <:<10>error in card number:>, cardno,
60          <: at position:>, p, "nl",1);
61    p:=1
62    if gettext (card, p, err, 81) < 0 then
63      putchar (err, p, 0, 3);
64      write(out, err, "nl",1);
65      putgeterror:=-1;
66      trap(errorcard);
67      goto next_card;
68    end;
69
70  finis:
71  trap (0);
72  close (card, true);
73  closesq (emp, true, true);
74
75  <*print empfile*>
76  p:= opensq (emp, <:empfile:>,0,0);
77  for i:= 1 step 1 until p do
78  begin
79    invar (emp);
80    write(out, emp.empno, emp.emp_name, <:..>,
81          emp.emp_addr, <:. <10>   :>,

```

```
77      emp.department, emp.salary, emp taxpct,  
78      "sp",2, false add emp.status,1, "nl",1);  
79  end;  
80  closesq(emp,true,true);  
81  end;
```

Appendix A. References

Part numbers in references are subject to change as new editions are issued and are listed as an identification aid only. To order, use package number.

- 1) PN: 991 11278
ALGOL8, Reference Manual
Included in SW8585, Compiler Collection
- 2) PN: 991 11280
ALGOL8, User's Guide, Part 2
Included in SW8585, Compiler Collection

