Character Data

Constants

Numeric constants (integer and real)

12 24 86.5 -39.2 -7

Character strings

'A' 'B' '7' 'xyz' '+' '123' 'Fred'

Note: A program can have single character constants or strings of characters. Both are treated as SINGLE values. The quote marks are NOT part of the string, but they must be used to distinguish strings of characters from variable names.

Declarations

CHARACTER (LEN=4) :: name

This allows the character variable **name** to hold a character string of length 4 (no more, no less).

Alternate declaration:

CHARACTER :: name*4, addres*20

This declares the variable **name** to store a character string of length 4 and the variable **address** to store a string of length 20.

The alternate declaration method is used when one wishes to declare character variables of different lengths within the same character declaration statement.

Character vs Numeric Storage

Note: digits stored in character variables cannot be used in arithmetic expressions, (i.e. '1' is logically different from 1)

Character Variables

Assignment

Given the folowing declaration:

CHARACTER (LEN=4) :: name

Character Assignment Rules

1) **IF** the assigned string is < the length of the character variable
THEN the string is LEFTjustified and 'padded' with blanks

LEFT JUSTIFICATION (LJ)

2) **IF** the assigned string is > the length of the character variable
THEN the string is 'truncated' on the RIGHT

RIGHT TRUNCATION (RT)

Constant Examples:

where the **ß** character represents a blank character.

Assignment Statement	Memory NAME	
name = 'FRED'	FRED	← LJ
name = 'JOE'	JOEß	← RT
name = 'WILLIAM'	WILL	

Variable Examples

Given the following declarations:

CHARACTER :: name*6, temp*4

Assignment Statement	Memory NAME	Memory TEMP	
name = 'KOOPER'	KOOPER		← RT
temp = name		KOOP	← LJ
name = temp	KOOPßß		

Character Data Input

Format Code

Use A (**alphanumeric**) format code, similar to I.

Aw where w >= 1 and indicates the width of the field

One must be aware of the way that the character variable is declared.

Input (read)

For input **Aw**, w dictates how many columns of characters will be read.

Character Input Rules

1) **IF** the input format code is < the length of the character variable
THEN the string is LEFTjustified and 'padded' with blanks
LEFT JUSTIFICATION (LJ)

2) **IF** the input format code is > the length of the character variable
THEN the string is 'truncated' on the LEFT
LEFT TRUNCATION (LT)

Examples

Given the following line of input data to be read next: Johnson

Given the following declaration and input statements:

```
CHARACTER ::   name*6
read (9,100)  name
```

Applying the FORMAT statements below yields:

FORMAT statement	Memory name	
100 format (A6)	Johnso	← LJ
100 format (A4)	John**ßß**	← LT
100 format (A7)	ohnson	

Character Data Output

Format Code (write)

For output **Aw**, w dictates how many columns of characters will be written.

Character Output Rules

1) **IF** the output format code is < the length of the character variable
THEN the string is 'truncated' on the RIGHT
RIGHT TRUNCATION (**RT**)

2) **IF** the OUTPUT format code is > the length of the character variable
THEN the string is RIGHT justified & padded with blanks on the LEFT
RIGHT JUSTIFICATION (**RJ**)

Examples

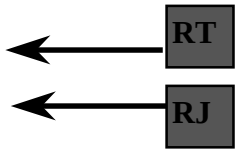
Given the following declaration, assignment and output statements:

```
CHARACTER :: name*6
name = 'Taylor'

write (10,200)  name
```

Applying the FORMAT statements below yields:

	FORMAT statement	OUTPUT column 123456789
200	format (TR1, A6)	Taylor
200	format (TR1, A4)	Tayl
200	format (TR1, A8)	ßßTaylor



Character Comparison

Lexicographic Ordering

A program can compare strings of different lengths. The result is obtained by an alphabetic ordering.

... ' ' < '0' < '1' < ... < '9' < ... < 'A' < 'B' < ... < 'Z' < ... < 'a' < 'b' ...

The above character ordering shows the internal machine ordering.

ASCII Codes

All characters in a computer are stored according to a standard positive numeric ordering called the ASCII codes.

character	code
' ' (ß)	32
'0'	48
'A'	65
'a'	96

The character digits, uppercase and lowercase letters are ordered in sequence.

Relational Comparisons

Given the following declarations and assignments:

```
CHARACTER ::  name*6, chars*4
name = 'TAYLOR'
chars = 'WILL'
```

Examples:

character expression	**relational result**
(name < chars)	true
('M' == 'M ')	true
('ME' < 'RA')	true
('SMITH, J' > 'SMITH, A')	true
('Zebra' < 'apple')	true
('Fred' < 'Freddy')	true
(' ' < 'D')	true
('a' < 'A')	false

Substrings

String Subscripting

FORTRAN allows a program to refer to a (partial) segment of a character variable's contents, termed a substring.

The formed substring can be used in manner as a regular string.

Syntax:

string (first:last)

where first and last are integers such that:

$1 \leq first \leq last \leq$ length of the string

Simple Examples:

Given the following declaration and assignments:

```
CHARACTER ::  name1*6, name2*6
name1 = 'freddy'
name2 = 'billy'
```

The substring **name1(2:4)** has the value **'red'**

The substring assignment:

name1(3:4) = name2(2:3)

causes **name1** to take on the value **'frildy'**

Substring Program

```
    PROGRAM substring
    IMPLICIT NONE
    INTEGER :: i
    CHARACTER :: name*7='WILLIAM'
1   WRITE(*,*) 'The name is ',name
2   WRITE(*,*) name (2:4)
3   WRITE(*,*) name (7:7)
4   WRITE(*,*) name ( :4)
5   WRITE(*,*) name (5: )
    DO i = 1, 7
6       WRITE(*,*) name (i:i)
    END DO
    DO  i = 1, 7
7       WRITE(*,*) name ( :i )
    END DO
    DO  i = 1, 7
8       WRITE(*,*) name ( i: )
    END DO
    DO  i = 1, 4
9       WRITE(*,*) name ( i:8 - i )
    END DO
    name (2:5) = 'FRED'
10  WRITE(*,*) name
    name = 'WILLIAM'
    name(2:5) = 'FR'
11  WRITE(*,*) name
    name = 'WILLIAM'
    name(2:5) = 'FREDDY'
12  WRITE(*,*) name
13  WRITE(*,100) name
100     FORMAT(TR1, A8)
14  WRITE(*,200) name
200     FORMAT(TR1, A6)
    STOP
    END PROGRAM substring
```

First substring subscript defaults to 1 if omitted.

(1:4)

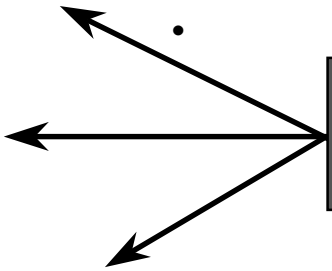
(5:7)

Last substring subscript defaults to the last character position if omitted.

(1:i)

(i:7)

(1:7)
(2:6)
•
•



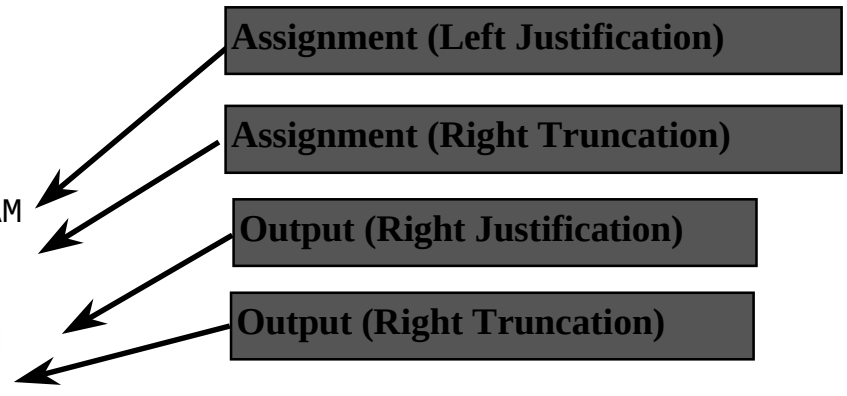
Substring assignment only affects the subscripted characters.

Note: To compile this program using ELF90, omit all labels except labels for FORMAT statements.

Prog Substring Output

```
1    The name is WILLIAM
2    ILL
3    M
4    WILL
5    IAM
6    W
     I
     L
     L
     I
     A
     M
7    W
     WI
     WIL
     WILL
     WILLI
     WILLIA
     WILLIAM
8    WILLIAM
     ILLIAM
     LLIAM
     LIAM
     IAM
     AM
     M
9    WILLIAM
     ILLIA
     LLI
     L
10   WFREDAM
11   WFR    AM
12   WFREDAM

13   ßWFREDAM
14   WFREDA
```



String Operations/Functions

String Concatenation // forms one string from two

The expression **'ABC' // 'XYZ'** yields the string **'ABCXYZ'**
Can be used in assignments such as:

```
name = last // first
```

INDEX (integer function)

loc = **INDEX (arg1, arg2)**

Returns the integer position of arg2 within arg1, where arg1 and arg2 can be either a character constant, character variable, or substring name.

examples:

length 15, padded with 5 ßs

```
CHARACTER :: s*20
s = 'the end is near'
loc = INDEX ( s, 'end' )
loc get the value 5
loc = INDEX ( s, 'NEAR' )
```

loc gets the value 0 because 'NEAR' is not in s

case sensitive

example: (What value does i get?)

```
CHARACTER::name*4
name = 'FRED'
i= INDEX ( name (2:4), 'E' )
```

LEN (integer function)

L = **LEN (arg1)**

Returns the integer length of the string arg1 where arg1 can be either a character constant, character variable, or substring name.

examples:

(10:20)

```
L = LEN ( s(10: ) )
```

since (10:20) is 11 characters

L gets the value 11

```
L = LEN ( s // 'FRED' //  s( 2:9 ) )
```

20 4 8 = 32

String Functions (cont)

CHAR (character function)

ch = **CHAR (arg)**

Returns the character that corresponds to the **ASCII** code of arg, where arg must be an integer expression.

example:

ch = CHAR (**65**)

ch gets the ASCII character that corresponds to 65, i.e. '**A**'

ch = CHAR(**32**)

ch gets the ASCII character of 32, i.e. ' ', which is a **blank**

ICHAR (integer function)

i = **ICHAR (arg1)**

Returns the integer value **(ASCII** code) that corresponds to the character arg1, where arg1 must be a character expression of length one.

example:

```
i = ICHAR ( 'B' )
```

i gets the integer value **66**

```
i = ICHAR( ' ' )
```

i gets the integer value **32**

Center Lines

Problem

Read each line of an input file and print it out verbatim, unless the line begins **'.CE'**, which requires the next line of the file to be output centered on a 80 character line.

Program

```
    PROGRAM center
    IMPLICIT NONE
    INTEGER :: ispaces, i, Llen, ios
    CHARACTER line*80
    OPEN ( .......)
    READ (9,100,IOSTAT=ios) line
100 FORMAT ( A80 )
    DO WHILE (ios >= 0)
      IF ( line(1:3) /= '.CE' ) THEN
         WRITE (*,200 ) line
200      FORMAT (TR1, A80 )
      ELSE       ! Center Next Line
         READ (9,100) line
! Find rightmost non-blank character in the line
         Llen = 80
         DO WHILE (( line(Llen:Llen) == ' ' ) .AND.  &
                  (Llen > 1))
              Llen = Llen - 1
         END DO
! Determine number of spaces to center
         ispaces = ( 80 - Llen ) / 2
! Concatenate centering spaces to the left of the line
         DO  i = 1, ispaces
           line = ' ' // line
          END DO
! Output centered line
         WRITE (*,200 ) line
      END IF
    READ(9,100,IOSTAT=ios)
    END DO
    STOP
    END PROGRAM center
```

String Fn/Oper Examples

Given:

```
CHARACTER :: st*10, first*10, mi, last*10, name*21, fullna*12
st = 'WASHINGTON'
```

Concatenation Examples:

Expression	Result
`st // '  STATE'`	'WASHINGTON STATE'
`'GEORGE  ' // st`	'GEORGE WASHINGTON'
`st( :4) // st(5:7) // st(8: )`	'WASHINGTON'
`st(:3) // ' ' // st(5:5) // st(8:8) // '?'`	'WAS IT?'
`st(10:10) // st(9:9) // st(8:8)`	'NOT'

(1:4) **(8:10)**

INDEX & ICHAR Examples

first = 'Fred'
mi = 'G'
last = 'Flintstone'

name = **first // 'ß' // mi // '.ß' // last**
name gets the value 'FredßßßßßßßG.ßFlintst'

'Fred'

name = **first(1:INDEX(first,'ß')-1 // mi // last**
name gets the value 'FredGFlintstoneßßßßßßß'

fullna = **name**
fullna gets the value 'FredGFlintst'

first= 'allen'
mi = 'b'
last = 'davis'

name = **first // 'ß' // mi // '.ß' // last**
OR
name = 'allenßßßßßßb.ßdavisßß'
name gets the value 'allenßßßßßßb.ßdavisßß'

Cryptograms

Problem†

A cryptogram is a coded message formed by substituting a code character for each letter of an original message. The substitution is performed uniformly throughout the original message, that is, all A's might be replaced by Z's, all B's by Y's, and so on. We will assume that all punctuation (including blanks between words) remains unchanged.

Overview

The program must examine each character in a message, MESSAGE, and insert the appropriate substitution for that character in the cryptogram, CRYPTO. This can be done by using the position of the original character in the alphabet string ALPHABET as an index to the string of code symbols, CODE (e.g., the code symbol for the letter A should always be the first symbol in CODE; the code symbol for the letter B should be the second symbol in CODE, etc.).

Method

1. Enter code string (CODE) and message (MESSAGE).
2. Form cryptogram (CRYPTO) by replacing each letter in MESSAGE with the corresponding code symbol.
   - 2.1 DO for each character in MESSAGE
   - 2.2 Locate position, POSCHR, of next message character in alphabet string, ALPHABET.
   - 2.3 IF POSCHR is not equal to O THEN
   - 2.4 Insert corresponding code symbol into CRYPTO ELSE
   - 2.5 Insert the next message symbol into CRYPTO END IF

   END DO
3. Print CRYPTO.

† "Problem Solving and Structured Programming in FORTRAN 77" 4th ed., Koffman, E.B. & Friedman, F.L., Addison-Wesley, 1990.

Cryptogram Program

```
  PROGRAM cryptogram
    IMPLICIT NONE
! Declarations
     CHARACTER,PARAMETER::ALPHABET*26='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
     INTEGER::POSCHR, NEXT
     CHARACTER(LEN=26):: CODE
     CHARACTER(LEN=80)::MESSAGE, CRYPTO

! Enter code and message
     WRITE(*,*)'Enter code symbol for each letter under the letter'
     WRITE(*,*) ALPHABET
     READ(*,*) CODE
     WRITE(*,*)
     WRITE(*,*) 'Enter a message and I will display its cryptogram:
     READ(*,*) MESSAGE

! Substitute the code symbol for each letter in the message
       DO  NEXT = 1, LEN(MESSAGE)
!         Locate the current message character in ALPHABET
          POSCHR = INDEX(ALPHABET, MESSAGE(NEXT : NEXT)
          IF (POSCHR /= O) THEN
!            Insert code for letter in CRYPTO
             CRYPTO(NEXT : NEXT) = CODE(POSCHR : POSCHR)
          ELSE
!            Insert non-letter in CRYPTO
             CRYPTO(NEXT : NEXT) = MESSAGE(NEXT : NEXT)
          END IF
       END DO

! Print the cryptogram
     WRITE(*,*) CRYPTO

     STOP
     END PROGRAM cryptogram
```

Program Cryptogram

Discussion

The statement:

POSCHR = INDEX(ALPHABET, MESSAGE(NEXT:NEXT))

locates the current message symbol in the string ALPHABET and the statement:

CRYPTO(NEXT:NEXT) = CODE(POSCHR:POSCHR)

inserts the corresponding code symbol in the cryptogram. The statement:

CRYPTO (NEXT:NEXT) = MESSAGE (NEXT:NEXT)

inserts any message symbol that is not a letter directly into the cryptogram.

Output

```
Enter the code symbol for each letter under that
letter:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA

Enter a message and I will display its
cryptogram:
ENCODE THIS *$?+ MESSAGE!
VMXLWV GSRH *$?+ NVHHZTV!
```