Character/String Data, Expressions & Intrinsic Functions (CHARACTER Data Type), Part 2

See *Programming in Fortran 90/95*, Chapter 11, section 11.1.

Extra Spaces at the End of a Character String

When we declare a character string, we give it a length attribute:

```
CHARACTER (LEN = 12) :: my_name
```

But, this assumes that we know the exact length of the string's value.

In some cases, that might be so. But in general, it's too much of a pain to count out the number of characters in the string's value, assuming that we even know what the value is when we write the program.

Instead, it's better simply to declare the character string to have the maximum conceivable length; since memory is cheap, we can simply ignore the leftover characters at the end of the string.

```
INTEGER,PARAMETER :: maximum_string_length = 80
CHARACTER (LEN = maximum_string_length) :: my_name
```

The Problem with Putting Extra Spaces at the End of a Character String

```
  INTEGER,PARAMETER :: maximum_string_length = 32
  CHARACTER (LEN = maximum_string_length) :: &
&     my_name = "Henry Neeman"
```

When we declare a string to be of an arbitrarily chosen length, as above, in general we end up with a string that has more space than it has actual data. In Fortran 90, the unused characters are set to blanks.

So, in the above declarations, the string variable `my_name` has the name `"Henry Neeman"` followed by 20 blanks (32 characters total, minus 12 characters in the name).

This can be awkward when outputting the value of a string:

```
% cat charstrassn32.f90
PROGRAM character_string_assign_32
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    CHARACTER (LEN = my_name_length) :: my_name

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
END PROGRAM character_string_assign_32
% f95 -o charstrassn32 charstrassn32.f90
% charstrassn32
 My name is Henry Neeman                    , so there.
```


The `TRIM` Instrinsic Function

In Fortran 90, we can eliminate the problem of trailing blanks by using an intrinsic function named `TRIM`:

```
% cat charstrassntrim.f90
PROGRAM character_string_assign_trim
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    CHARACTER (LEN = my_name_length) :: my_name

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
    PRINT *, "My name is ", TRIM(my_name), &
 &      ", so there."
END PROGRAM character_string_assign_trim
% f95 -o charstrassntrim charstrassntrim.f90
% charstrassntrim
 My name is Henry Neeman                    , so there.
 My name is Henry Neeman, so there.
```

`TRIM` takes a string as its argument, and returns a string as its return value.

The return string has the exact same data as the argument string, except that the return string's length is the length of the part of the argument string before the trailing blanks.

That is, `TRIM` trims off the trailing blanks.

In Fortran 90, whenever you output a string, it's best to use `TRIM`.

The Lengths of a String

```
% cat charstrassntrim.f90
PROGRAM character_string_assign_trim
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    CHARACTER (LEN = my_name_length) :: my_name

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
    PRINT *, "My name is ", TRIM(my_name), &
 &      ", so there."
END PROGRAM character_string_assign_trim
% f95 -o charstrassntrim charstrassntrim.f90
% charstrassntrim
 My name is Henry Neeman                    , so there.
 My name is Henry Neeman, so there.
```

In Fortran 90, you can think of a string as having two different lengths: its length in memory (i.e., the length that it's declared to have), and its trimmed length (i.e., its length excluding trailing blanks).

Thus, in the case of the program above, the string `my_name` has a length in memory of 32 characters, but a trimmed length of 12 characters.

`LEN` and `LEN_TRIM`

Fortran 90 has intrinsic functions for both of these lengths: `LEN` for the length in memory, and `LEN_TRIM` for trimmed length:

```
% cat charstrassnlen.f90
PROGRAM character_string_assign_length
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    CHARACTER (LEN = my_name_length) :: my_name

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
    PRINT *, "My name is ", TRIM(my_name), &
 &      ", so there."
    PRINT *, "LEN(my_name)            = ", &
 &           LEN(my_name)
    PRINT *, "LEN_TRIM(my_name)       = ", &
 &           LEN_TRIM(my_name)
    PRINT *, "LEN(TRIM(my_name))      = ", &
 &           LEN(TRIM(my_name))
    PRINT *, "LEN_TRIM(TRIM(my_name)) = ", &
 &           LEN_TRIM(TRIM(my_name))
END PROGRAM character_string_assign_length
% f95 -o charstrassnlen charstrassnlen.f90
% charstrassnlen
 My name is Henry Neeman                    , so there.
 My name is Henry Neeman, so there.
 LEN(my_name)            =  32
 LEN_TRIM(my_name)       =  12
 LEN(TRIM(my_name))      =  12
 LEN_TRIM(TRIM(my_name)) =  12
```

Substrings

A *substring* is a portion of a string.

Fortran 90 has a special syntax for indicating a substring:

```
the_string(start_index:end_index)
```

The above string expression means: “the portion of the string named `the_string`, beginning at the index represented by `start_index` and ending at the index represented by `end_index`.”

```
% cat charstrassnsub.f90
PROGRAM character_substring
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    CHARACTER (LEN = my_name_length) :: my_name

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
    PRINT *, "My name is ", TRIM(my_name), &
 &      ", so there."
    PRINT *, "My first name is ", my_name(1:5), "."
    PRINT *, "My last name is ", my_name(7:12), "."
END PROGRAM character_substring
% f95 -o charstrassnsub charstrassnsub.f90
% charstrassnsub
 My name is Henry Neeman                    , so there.
 My name is Henry Neeman, so there.
 My first name is Henry.
 My last name is Neeman.
```

Note that a substring can have length 1:

```
the_string(index:index)
```


Searching a String Using Substrings of Length 1

```
% cat charstrsearch.f90
PROGRAM character_string_search
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    INTEGER,PARAMETER :: first_index    =  1
    INTEGER,PARAMETER :: increment      =  1
    CHARACTER (LEN = my_name_length) :: my_name
    INTEGER :: index

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
    PRINT *, "My name is ", TRIM(my_name), &
 &      ", so there."
    index = first_index
    DO WHILE ((index <= my_name_length) .AND. &
 &            (my_name(index:index) /= " "))
        index = index + increment
    END DO !! WHILE ((index <= my_name_length) ...
    PRINT *, "My first name is ", &
 &      my_name(first_index:index-1), "."
    PRINT *, "My last name is ", &
 &      my_name(index+1:LEN_TRIM(my_name)), "."
END PROGRAM character_string_search
% f95 -o charstrsearch charstrsearch.f90
% charstrsearch
 My name is Henry Neeman                    , so there.
 My name is Henry Neeman, so there.
 My first name is Henry.
 My last name is Neeman.
```

Substrings of Length 1 CANNOT Be Referred to Using Array Index Notation

A substring of length 1 is indexed like so:

```
my_name(index:index)
```

It **CANNOT** be referred to using array index notation:

```
my_name(index) !! ILLEGAL ILLEGAL ILLEGAL
```

```
% cat charstrsearchbad.f90
PROGRAM character_string_search_bad
    IMPLICIT NONE
    INTEGER,PARAMETER :: my_name_length = 32
    INTEGER,PARAMETER :: first_index    =  1
    INTEGER,PARAMETER :: increment      =  1
    CHARACTER (LEN = my_name_length) :: my_name
    INTEGER :: index

    my_name = "Henry Neeman"
    PRINT *, "My name is ", my_name, ", so there."
    PRINT *, "My name is ", TRIM(my_name), &
 &       ", so there."
    index = first_index
    DO WHILE ((index <= my_name_length) .AND. &
 &            (my_name(index) /= " ")) !! ILLEGAL
        index = index + increment
    END DO !! WHILE ((index <= my_name_length) ...
    PRINT *, "My first name is ", &
 &      my_name(first_index:index-1), "."
    PRINT *, "My last name is ", &
 &      my_name(index+1:LEN_TRIM(my_name)), "."
END PROGRAM character_string_search_bad
% f95 -o charstrsearchbad charstrsearchbad.f90
Error: charstrsearchbad.f90, line 15:
    Inconsistent usage of MY_NAME
       detected at )@/=
[f95 terminated - errors found by pass 1]
```


String Concatenation

Given two strings, the operation that creates a new string whose contents are the first string followed immediately by the second string is called *concatenation*, which in Fortran 90 is achieved using the *concatenation operator*:

```
first_string // second_string
```

```
PROGRAM character_string_concatenate
    IMPLICIT NONE
    INTEGER,PARAMETER :: part_name_length = 16
    INTEGER,PARAMETER :: full_name_length = 32
    INTEGER,PARAMETER :: first_index    =  1
    INTEGER,PARAMETER :: increment      =  1
    CHARACTER (LEN = part_name_length) :: &
 &      first_name, last_name
    CHARACTER (LEN = full_name_length) :: &
 &      full_name, trimmed_name, good_name

    first_name = "Henry"
    last_name  = "Neeman"
    full_name    = first_name // last_name
    trimmed_name = &
 &      TRIM(first_name) // TRIM(last_name)
    good_name    = &
 &      TRIM(first_name) // " " // TRIM(last_name)
    PRINT *, "My first name is ", first_name, "."
    PRINT *, "My last  name is ", last_name,  "."
    PRINT *, "My first name is ", &
 &      TRIM(first_name), "."
    PRINT *, "My last  name is ", &
 &      TRIM(last_name),  "."
    PRINT *, "My full name is ", full_name, "."
    PRINT *, "My trimmed name is ", trimmed_name, "."
    PRINT *, "My name is ", good_name, "."
    PRINT *, "My name is ", TRIM(good_name), "."
END PROGRAM character_string_concatenate
```

The full script is on the next page.

String Concatenation Example

```
% cat charstrconcat.f90
PROGRAM character_string_concatenate
    IMPLICIT NONE
    INTEGER,PARAMETER :: part_name_length = 16
    INTEGER,PARAMETER :: full_name_length = 32
    INTEGER,PARAMETER :: first_index    =  1
    INTEGER,PARAMETER :: increment      =  1
    CHARACTER (LEN = part_name_length) :: &
 &      first_name, last_name
    CHARACTER (LEN = full_name_length) :: &
 &      full_name, trimmed_name, good_name

    first_name = "Henry"
    last_name  = "Neeman"
    full_name    = first_name // last_name
    trimmed_name = &
 &      TRIM(first_name) // TRIM(last_name)
    good_name    = &
 &      TRIM(first_name) // " " // TRIM(last_name)
    PRINT *, "My first name is ", first_name, "."
    PRINT *, "My last  name is ", last_name,  "."
    PRINT *, "My first name is ", &
 &      TRIM(first_name), "."
    PRINT *, "My last  name is ", &
 &      TRIM(last_name),  "."
    PRINT *, "My full name is ", full_name, "."
    PRINT *, "My trimmed name is ", trimmed_name, "."
    PRINT *, "My name is ", good_name, "."
    PRINT *, "My name is ", TRIM(good_name), "."
END PROGRAM character_string_concatenate
% f95 -o charstrconcat charstrconcat.f90
% charstrconcat
 My first name is Henry           .
 My last  name is Neeman          .
 My first name is Henry.
 My last  name is Neeman.
 My full name is Henry           Neeman          .
 My trimmed name is HenryNeeman                     .
 My name is Henry Neeman                   .
 My name is Henry Neeman.
```


String Comparisons

Just as numeric values can be compared, so can string values.

However, strings aren't scalars, and strings aren't arrays, exactly.

In Fortran 90, two strings are defined to be equal if they have the exact same contents, except for trailing blanks.

String comparison is **case sensitive**.

Thus, if two strings are identical, except that, in a single character, they differ by case — for example, an `"H"` for one string corresponds to an `"h"` for the other — then they will not be equal.

For example:

```
"Henry" /= "henry"
```

String Equality Example

```
% cat charstrequal.f90
PROGRAM character_string_equal
    IMPLICIT NONE
    INTEGER,PARAMETER :: part_name_length = 16
    INTEGER,PARAMETER :: full_name_length = 32
    INTEGER,PARAMETER :: first_index    =  1
    INTEGER,PARAMETER :: increment      =  1
    CHARACTER (LEN = part_name_length) :: &
 &      first_name, last_name, lower_first_name
    CHARACTER (LEN = full_name_length) :: &
        long_first_name

    first_name = "Henry"
    last_name  = "Neeman"
    lower_first_name = "henry"
    long_first_name = first_name
    IF (first_name == first_name) THEN
        PRINT *, "The string is equal ", &
 &          "to itself."
    ELSE   !! (first_name == first_name)
        PRINT *, "The string is not equal ", &
 &          "to itself."
    END IF !! (first_name == first_name)...ELSE
    IF (first_name == lower_first_name) THEN
        PRINT *, "The string is equal ", &
 &          "to its lower case version."
    ELSE   !! (first_name == lower_first_name)
        PRINT *, "The string is not equal ", &
 &          "to its lower case version."
    END IF !! (first_name == lower_first_name)...ELSE
    IF (first_name == long_first_name) THEN
        PRINT *, "The string is equal ", &
 &          "to its longer version."
    ELSE   !! (first_name == long_first_name)
        PRINT *, "The string is not equal ", &
 &          "to its longer version."
    END IF !! (first_name == long_first_name)...ELSE
    IF (first_name == last_name) THEN
        PRINT *, "first_name is equal ", &
 &          "to last_name."
    ELSE   !! (first_name == last_name)
        PRINT *, "first_name is not equal ", &
 &          "to last_name."
    END IF !! (first_name == last_name)...ELSE
END PROGRAM character_string_equal
% f95 -o charstrequal charstrequal.f90
% charstrequal
 The string is equal to itself.
 The string is not equal to its lower case version.
 The string is equal to its longer version.
 first_name is not equal to last_name.
```


Other String Comparisons

Comparisons other than equality and inequality are more complicated. We need to define the notion of “less than” and “greater than,” in order to make these comparisons.

In Fortran 90, we define these comparisons *lexically*.

That is, we perform the test for whether one string is less than (respectively, greater than) another by comparing each pair of associated characters in turn, until the ASCII value of the first is not equal than the ASCII value of the second.

If the ASCII value of the given character in the first is less than that of the associated character in the second, then the first string is *lexically less than* the second string; likewise, if the ASCII value of the given character in the first is greater than that of the associated character in the second, then the first string is *lexically greater than* the second string. If the ASCII value of the given character in the first is less than that of the associated character in the second, then the first string is *lexically less than* the second string;

Note that trailing blanks are ignored.

If the two strings are identical in every character (except trailing blanks), then they are *lexically equal to* each other.

If the two strings are identical in every character, but one of them is shorter (except trailing blanks) that the other, then the shorter string is lexically less than the longer string.

Lexical Comparison Example

```
% cat charstrlex.f90
PROGRAM character_string_lexical
    IMPLICIT NONE
    INTEGER,PARAMETER :: part_name_length = 16
    INTEGER,PARAMETER :: full_name_length = 32
    INTEGER,PARAMETER :: first_index    =  1
    INTEGER,PARAMETER :: increment      =  1
    CHARACTER (LEN = part_name_length) :: &
 &       first_name, last_name, lower_first_name
    CHARACTER (LEN = full_name_length) :: &
        long_first_name, good_name

    first_name = "Henry"
    last_name  = "Neeman"
    lower_first_name = "henry"
    long_first_name = first_name
    good_name = &
 &      TRIM(first_name) // " " // TRIM(last_name)
    PRINT *, "LLT(", TRIM(lower_first_name), &
 &           ", ", TRIM(first_name), "):  ", &
             LLT(lower_first_name, first_name)
    PRINT *, "LGT(", TRIM(lower_first_name), &
 &           ", ", TRIM(first_name), "):  ", &
             LGT(lower_first_name, first_name)
    PRINT *, "LLT(", TRIM(first_name),     &
 &           ", ", TRIM(last_name), "):  ", &
             LLT(first_name, last_name)
    PRINT *, "LGT(", TRIM(first_name),     &
 &           ", ", TRIM(last_name), "):  ", &
             LGT(first_name, last_name)
    PRINT *, "LLT(", TRIM(first_name), &
 &           ", ", TRIM(good_name), "):  ", &
             LLT(first_name, good_name)
    PRINT *, "LGT(", TRIM(first_name), &
 &           ", ", TRIM(good_name), "):  ", &
             LGT(first_name, good_name)
    PRINT *, "LLT(", TRIM(long_first_name), &
 &           ", ", TRIM(first_name), "):  ", &
             LLT(long_first_name, first_name)
    PRINT *, "LLE(", long_first_name, &
 &           ", ", first_name, "):  ", &
             LLE(long_first_name, first_name)
END PROGRAM character_string_lexical
% f95 -o charstrlex charstrlex.f90
% charstrlex
 LLT(henry,Henry):   F
 LGT(henry,Henry):   T
 LLT(Henry,Neeman):   T
 LGT(Henry,Neeman):   F
 LLT(Henry,Henry Neeman):   T
 LGT(Henry,Henry Neeman):   F
 LLT(Henry,Henry):   F
 LLE(Henry                         ,Henry           ):   T
```


Character String Expressions in Fortran 90

Fortran 90 has some pretty powerful intrinsic string operators and functions:

1. Substrings: `string(start:end)`
2. Concatenation: `str1 // str2`
3. Creation: `REPEAT(the_string,count)`
4. Modification
   (a) Trailing blank removal: `TRIM(the_string)`
   (b) Left justify: `ADJUSTL(the_string)`
   removes leading blanks, adds an equal number of trailing blanks
   (c) Right justify: `ADJUSTR(the_string)`
   removes trailing blanks, adds an equal number of leading blanks
5. Length
   (a) Length in memory: `LEN(the_string)`
   (b) Trimmed length before trailing blanks: `LEN_TRIM(the_string)`
6. Encoding/Decoding
   (a) Encode character to ASCII: `IACHAR(chr)`
   (b) Encode character to native code: `ICHAR(chr)`
   (c) Encode character from ASCII: `ACHAR(integ)`
   (d) Encode character from native code: `CHAR(integ)`
7. Comparison
   (a) Greater than: `LGT(string1,string2)`
   (b) Greater than or equal to: `LGE(string1,string2)`
   (c) Less than: `LLT(str1,str2)`
   (d) Less than or equal to: `LLE(string1,string2)`

You can find a complete listing of string manipulators in *Programming in Fortran 90/95*, pages 113-117.