Character strings

Arjen Markus

Deltares

June 6, 2018

Character strings

Character strings in Fortran generally have a fixed length:

```
character(len=20) :: string

string = '123'

write(*,*) '>', string, '<'

==> >123                  <
```

The string is padded with spaces.

Character strings in subroutines/functions

You can use the length of an argument to define a local string variable of the right length:

```
function toupper( string )
    character(len=*)              :: string   ! * means inherit the length
                                              ! from the actual argument

    character(len=len(string)) :: toupper  ! Reserve the right length
                                              ! to return the string
    ...
end function toupper
```

Deferred-length character strings

So-called deferred-length strings have a flexible length:

```
character(len=:), allocatable                  :: string
character(len=:), dimension(:), allocatable :: str_array

allocate( character(len=20) :: string )
allocate( character(len=10) :: str_array(100) )
    ...

write(*,*) 'Length: ', len(string)
end function toupper

==> Length: 20
```

Note: each element of the array gets the same length

Deferred-length character strings – caveat!

Since deferred-length strings are allocatable, this code may have a surprising outcome:

```
character(len=:), allocatable                    :: string

allocate( character(len=10) :: string )

string         = '1'
string(10:10) = '2'

write(*,*) '>', string, '<'

==> >1<
```

The reason: the left-hand side is allocatable, so it is adjusted to the size of the right-hand side.

Deferred-length character strings – another caveat?

What about this code:

```
character(len=:), dimension(:), allocatable :: array

allocate( character(len=10) :: array(10) )

array(1)       = '1234567890'
array(2)       = '1'

write(*,*) '>', array(1), '<'
write(*,*) '>', array(2), '<'

==> ???
```

Some character functions

The standard defines a handful of functions to work on character strings:

- `len()` returns the total length of the string and `len_trim()` returns the length of the string without trailing spaces.
- `adjustl()` and `adjustr()` return a string adjusted to the left or to the right – leaving out leading spaces or inserting leading spaces.
- `trim()` returns a string with trailing spaces.

```
character(len=10) :: string
         !1234567890
string = '    xyz   '
                                          !1234567890
write(*,*) len(string)                 !==> 10
write(*,*) len_trim(string)            !==> 7
write(*,*) '>', trim(string), '<'      !==> >    xyz<
write(*,*) '>', adjustl(string), '<'   !==> >xyz       <
write(*,*) '>', adjustr(string), '<'   !==> >       xyz<
```

Some character functions (2)

Finding substrings/characters:

- `index()` returns the index of a string inside another string.
- `scan()` scans a string and returns the first/last index of a character from a given set.
- `verify()` scans a string and returns the first/last index of a character *not* in a given set.

```
character(len=10) :: string
         !1234567890
string = 'fortran'

write(*,*) index(string, 'r')                  !==> 3
write(*,*) index(string, 'r', back = .true.)   !==> 5
write(*,*) scan(string, 'abc' )                !==> 6
write(*,*) verify(string, 'abc' )              !==> 1
write(*,*) verify(string, 'for' )              !==> 4
```

Non-default characters

Limited support for other character sets than ASCII:

```
integer, parameter :: alternative = ??
character(len=10, kind=alternative) :: string

string = alternative_'xyz'
```

You can specify the character set for a file:

```
open( 10, file = 'some_file.inp', encoding = 'UTF-8' )
```

Unfortunately:

- Not all compilers provide alternative character sets, for instance, UTF-8 or UNICODE.
- No inquiry functions are defined for the kind of a character string.