11. Fortran – Characters

The Fortran language can treat characters as single character or contiguous strings.

Characters could be any symbol taken from the basic character set, i.e., from the letters, the decimal digits, the underscore, and 21 special characters.

A character constant is a fixed valued character string.

The intrinsic data type **character** stores characters and strings. The length of the string can be specified by **len** specifier. If no length is specified, it is 1. You can refer individual characters within a string referring by position; the left most character is at position 1.

Character Declaration

Declaring a character type data is same as other variables:

```
type-specifier :: variable_name
```

For example,

```
character :: reply, sex
```

you can assign a value like,

```
reply = ‘N’
sex = ‘F’
```

The following example demonstrates declaration and use of character data type:

```
program hello
implicit none

   character(len=15) :: surname, firstname
   character(len=6) :: title
   character(len=25)::greetings

   title = 'Mr. '
   firstname = 'Rowan '
   surname = 'Atkinson'
   greetings = 'A big hello from Mr. Beans'

   print *, 'Here is ', title, firstname, surname
   print *, greetings

end program hello
```

When you compile and execute the above program it produces the following result:

```
Here is Mr. Rowan Atkinson
A big hello from Mr. Bean
```

Concatenation of Characters

The concatenation operator //, concatenates characters.

The following example demonstrates this:

```
program hello
implicit none

   character(len=15) :: surname, firstname
   character(len=6) :: title
   character(len=40):: name
   character(len=25)::greetings

   title = 'Mr. '
   firstname = 'Rowan '
   surname = 'Atkinson'

   name = title//firstname//surname
   greetings = 'A big hello from Mr. Beans'

   print *, 'Here is ', name
   print *, greetings

end program hello
```

When you compile and execute the above program it produces the following result:

```
Here is Mr.Rowan Atkinson
A big hello from Mr.Bean
```

Some Character Functions

The following table shows some commonly used character functions along with the description:

Function	Description
len(string)	It returns the length of a character string
index(string,sustring)	It finds the location of a substring in another string, returns 0 if not found.
achar(int)	It converts an integer into a character
iachar(c)	It converts a character into an integer
trim(string)	It returns the string with the trailing blanks removed.
scan(string, chars)	It searches the "string" from left to right (unless back=.true.) for the first occurrence of any character contained in "chars". It returns an integer giving the position of that character, or zero if none of the characters in "chars" have been found.
verify(string, chars)	It scans the "string" from left to right (unless back=.true.) for the first occurrence of any character not contained in "chars". It returns an integer giving the position of that character, or zero if only the characters in "chars" have been found
adjustl(string)	It left justifies characters contained in the "string"
adjustr(string)	It right justifies characters contained in the "string"
len_trim(string)	It returns an integer equal to the length of "string" (len(string)) minus the number of trailing blanks
repeat(string,ncopy)	It returns a string with length equal to "ncopy" times the length of "string", and containing "ncopy" concatenated copies of "string"

Example 1

This example shows the use of the **index** function:

```
program testingChars
implicit none

   character (80) :: text
   integer :: i

   text = 'The intrinsic data type character stores characters and   strings.'
   i=index(text,'character')

   if (i /= 0) then
      print *, ' The word character found at position ',i
      print *, ' in text: ', text
   end if

end program testingChars
```

When you compile and execute the above program it produces the following result:

```
The word character found at position 25
in text : The intrinsic data type character stores characters and strings.
```

Example 2

This example demonstrates the use of the **trim** function:

```
program hello
implicit none

   character(len=15) :: surname, firstname
   character(len=6) :: title
   character(len=25)::greetings

   title = 'Mr.'
   firstname = 'Rowan'
   surname = 'Atkinson'
   print *, 'Here is', title, firstname, surname
   print *, 'Here is', trim(title),' ',trim(firstname),' ', trim(surname)

end program hello
```

When you compile and execute the above program, it produces the following result:

```
Here is Mr. Rowan Atkinson
Here is Mr. Rowan Atkinson
```

Example 3

This example demonstrates the use of **achar** function

```
program testingChars
implicit none

   character:: ch
   integer:: i

   do i=65, 90
      ch = achar(i)
      print*, i, ' ', ch
   end do

end program testingChars
```

When you compile and execute the above program it produces the following result:

```
65  A
66  B
67  C
68  D
69  E
70  F
71  G
72  H
73  I
74  J
75  K
76  L
```

```
77  M
78  N
79  O
80  P
81  Q
82  R
83  S
84  T
85  U
86  V
87  W
88  X
89  Y
90  Z
```

Checking Lexical Order of Characters

The following functions determine the lexical sequence of characters:

Function	Description
lle(char, char)	Compares whether the first character is lexically less than or equal to the second
lge(char, char)	Compares whether the first character is lexically greater than or equal to the second
lgt(char, char)	Compares whether the first character is lexically greater than the second
llt(char, char)	Compares whether the first character is lexically less than the second

Example 4

The following function demonstrates the use:

```
program testingChars
implicit none
   character:: a, b, c
   a = 'A'
   b = 'a'
   c = 'B'

   if(lgt(a,b)) then
      print *, 'A is lexically greater than a'
   else
      print *, 'a is lexically greater than A'
   end if

   if(lgt(a,c)) then
      print *, 'A is lexically greater than B'
   else
      print *, 'B is lexically greater than A'
   end if

   if(llt(a,b)) then
      print *, 'A is lexically less than a'
   end if

   if(llt(a,c)) then
      print *, 'A is lexically less than B'
   end if
end program testingChars
```

When you compile and execute the above program it produces the following result:

```
a is lexically greater than A
B is lexically greater than A
A is lexically less than a
A is lexically less than B
```

12. Fortran – Strings

The Fortran language can treat characters as single character or contiguous strings.

A character string may be only one character in length, or it could even be of zero length. In Fortran, character constants are given between a pair of double or single quotes.

The intrinsic data type **character** stores characters and strings. The length of the string can be specified by **len specifier**. If no length is specified, it is 1. You can refer individual characters within a string referring by position; the left most character is at position 1.

String Declaration

Declaring a string is same as other variables:

```
type-specifier :: variable_name
```

For example,

```
Character(len=20) :: firstname, surname
```

you can assign a value like,

```
character (len=40) :: name
name = “Zara Ali”
```

The following example demonstrates declaration and use of character data type:

```
program hello
implicit none

   character(len=15) :: surname, firstname
   character(len=6) :: title
   character(len=25)::greetings

   title = 'Mr.'
   firstname = 'Rowan'
   surname = 'Atkinson'
   greetings = 'A big hello from Mr. Beans'

   print *, 'Here is', title, firstname, surname
   print *, greetings

end program hello
```

When you compile and execute the above program it produces the following result:

```
Here is Mr. Rowan Atkinson
A big hello from Mr. Bean
```

String Concatenation

The concatenation operator //, concatenates strings.

The following example demonstrates this:

```
program hello
implicit none

   character(len=15) :: surname, firstname
   character(len=6) :: title
   character(len=40):: name
   character(len=25)::greetings

   title = 'Mr.'
   firstname = 'Rowan'
   surname = 'Atkinson'

   name = title//firstname//surname
   greetings = 'A big hello from Mr. Beans'

   print *, 'Here is', name
   print *, greetings

end program hello
```

When you compile and execute the above program it produces the following result:

```
Here is Mr. Rowan Atkinson
A big hello from Mr. Bean
```

Extracting Substrings

In Fortran, you can extract a substring from a string by indexing the string, giving the start and the end index of the substring in a pair of brackets. This is called extent specifier.

The following example shows how to extract the substring ‘world’ from the string ‘hello world’:

```
program subString


   character(len=11)::hello
   hello = "Hello World"
   print*, hello(7:11)


end program subString
```

When you compile and execute the above program it produces the following result:

```
World
```

Example

The following example uses the **date_and_time** function to give the date and time string. We use extent specifiers to extract the year, date, month, hour, minutes and second information separately.

```
program  datetime
implicit none


   character(len = 8) :: dateinfo ! ccyymmdd
   character(len = 4) :: year, month*2, day*2


   character(len = 10) :: timeinfo ! hhmmss.sss
   character(len = 2)  :: hour, minute, second*6


   call  date_and_time(dateinfo, timeinfo)


   !  let’s break dateinfo into year, month and day.
   !  dateinfo has a form of ccyymmdd, where cc = century, yy = year
   !  mm = month and dd = day

   year  = dateinfo(1:4)
   month = dateinfo(5:6)
   day   = dateinfo(7:8)

   print*, 'Date String:', dateinfo
   print*, 'Year:', year
   print *,'Month:', month
   print *,'Day:', day

   !  let’s break timeinfo into hour, minute and second.
   !  timeinfo has a form of hhmmss.sss, where h = hour, m = minute
   !  and s = second

   hour   = timeinfo(1:2)
   minute = timeinfo(3:4)
   second = timeinfo(5:10)

   print*, 'Time String:', timeinfo
   print*, 'Hour:', hour
   print*, 'Minute:', minute
   print*, 'Second:', second

end program  datetime
```

When you compile and execute the above program, it gives the detailed date and time information:

```
Date String: 20140803
   Year: 2014
   Month: 08
   Day: 03
   Time String: 075835.466
   Hour: 07
   Minute: 58
   Second: 35.466
```

Trimming Strings

The **trim** function takes a string, and returns the input string after removing all trailing blanks.

Example

```
program trimString
implicit none

   character (len=*), parameter :: fname="Susanne", sname="Rizwan"
   character (len=20) :: fullname

   fullname=fname//" "//sname !concatenating the strings

   print*,fullname,", the beautiful dancer from the east!"
   print*,trim(fullname),", the beautiful dancer from the east!"

end program trimString
```

When you compile and execute the above program it produces the following result:

```
Susanne Rizwan, the beautiful dancer from the east!
Susanne Rizwan, the beautiful dancer from the east!
```

Left and Right Adjustment of Strings

The function **adjustl** takes a string and returns it by removing the leading blanks and appending them as trailing blanks.

The function **adjustr** takes a string and returns it by removing the trailing blanks and appending them as leading blanks.

Example

```
program hello
implicit none

   character(len=15) :: surname, firstname
   character(len=6) :: title
   character(len=40):: name
   character(len=25):: greetings
   title = 'Mr. '
   firstname = 'Rowan'
   surname = 'Atkinson'
   greetings = 'A big hello from Mr. Beans'

   name = adjustl(title)//adjustl(firstname)//adjustl(surname)
   print *, 'Here is', name
   print *, greetings

   name = adjustr(title)//adjustr(firstname)//adjustr(surname)
   print *, 'Here is', name
   print *, greetings

   name = trim(title)//trim(firstname)//trim(surname)
   print *, 'Here is', name
   print *, greetings

end program hello
```

When you compile and execute the above program it produces the following result:

```
Here is Mr. Rowan  Atkinson
A big hello from Mr. Bean
Here is Mr. Rowan Atkinson
A big hello from Mr. Bean
Here is Mr.RowanAtkinson
A big hello from Mr. Bean
```

Searching for a Substring in a String

The index function takes two strings and checks if the second string is a substring of the first string. If the second argument is a substring of the first argument, then it returns an integer which is the starting index of the second string in the first string, else it returns zero.

Example

```
program hello
implicit none

   character(len=30) :: myString
   character(len=10) :: testString

   myString = 'This is a test'
   testString = 'test'

   if(index(myString, testString) == 0)then
      print *, 'test is not found'
   else
      print *, 'test is found at index: ', index(myString, testString)
   end if

end program hello
```

When you compile and execute the above program, it produces the following result:

```
test is found at index: 11
```