`do` loops

Fortran `do` loop is the most basic loop statement with a basic counter running from an initial to a final value with a specified interval. No condition is checked inside the body of the loop and the loop is terminated when the counter reaches the final value (or greater than the final value with user specified interval)

```
do counter = initial, final, step
  [ loop body ]
end do
```

the last parameter `step` is optional and the default value of the interval is 1

```
do i=1,10
  [ loop body ]
end do
```

Example

```
program do_loop

  implicit none

  integer :: i, max_i, i_step

  max_i = 10
  i_step = 2

  do i=1, max_i, i_step
    print *,i
  end do

end program do_loop
```

will produce the following output

```
        1
        3
        5
        7
        9
```

`while` loop

A `while` loop has a significant difference from the `do` loop as a loop index is not present and the iterations of this loop are executed as long as an initial condition is not met. The condition is checked before the body of the loop is entered and in this case the `while` loop resembles multiple instances of `if` conditions with the same code following it

```
do while ( condition )
  [ loop body ]
end do
```

A good example of the `while` loop is calculation of the factorial of a given number where a counter is used and at the beginning of every iteration program checks if its value is larger than 0

```
program while_loop

  implicit none

  integer :: input, idx, factorial

  input = 5
  idx = input

  factorial = 1
  do while ( idx > 0 )

    factorial = factorial * idx
    print *, idx, factorial
    idx = idx - 1

  end do

  print *,'Value of factorial ', input, '! = ', factorial

end program while_loop
```

which given the output

```
       5           5
       4          20
       3          60
       2         120
       1         120
 Value of factorial            5 ! =          120
```

Infinite loop

Fortran allows omitting the counter parameters in the `do` loop which makes it an infinite loop.

```
do
  [ loop body ]
end do
```

will run forever, unless it is broken with a condition check inside the loop body and `exit` keyword.

Loop breaking

Keyword `exit` inside a loop will terminate the loop immediately without proceeding to the next iteration.

```
program loop_breaking

  implicit none

  integer :: i,total
  integer, parameter :: max_value = 50

  total = 0

  do i=1,10

    total = total + 10

    if ( total > max_value ) exit
    print *,i, total

  end do

  print *,'final total value : ', total

end program loop_breaking
```

will output

```
       1          10
       2          20
       3          30
       4          40
       5          50
final total value :           60
```

`until` loop

Fortran language does not have a concept of the `until` loop. Instead of that construct an infinite loop can be used together with loop breaking keyword `exit` to simulate that behavior

```
program until_loop

  implicit none

  integer :: i

  i = 0
  do
    print *,i
    i = i + 1

    ! condition check at the bottom of the loop
    if ( .not.(i < 10) ) exit
  end do

end program until_loop
```

which results in

```
           0
           1
           2
           3
           4
           5
           6
           7
           8
           9
```

Skiping a loop instance `cycle`

Keyword `cycle` will terminate current iteration of the loop and immediately proceed with the next iteration.

```
program loop_cycle

  implicit none

  integer :: i

  do i=1,10

    if ( i == 5 ) cycle
    print *,i

  end do

end program loop_cycle
```

will give an output

```
1
2
3
4
6
7
8
9
10
```

Loop labeling

Fortran allows for giving labels to major control structures like loops and conditionals. A label does not affect the body of the loop. For example

```
program loop_label

  implicit none

  integer :: i, max_i

  max_i = 3

  SimpleLoop: do i=1, max_i
    print *, i
  end do SimpleLoop

end program loop_label
```

which gives

```
           1
           2
           3
```

the label `SimpleLoop` at the beginning and the end of the loop is
particularly useful in case of many nested loops as it clearly defines beggining and the end of the loop body.

`forall` all independent loop

Fortran comes with an unusual loop which treats all iterations as independent by definition. All iterations can therefore be executed independently. The general form is

```
forall( var=initial_value:final_value[:stride])
    [ body ]
end forall
```

An example program

```
program forall_example

  implicit none

  real, allocatable, dimension(:,:) :: A
  integer :: n

  n = 100

  allocate( A(n,n) )

  forall( i=1:n ) A(i,i) = 1.0

  deallocate(A)

end program forall_example
```