Looping in Fortran

Victor Eijkhout, Susan Lindsey

Fall 2022

last formatted: August 28, 2022

1. Indexed Do loops

```
integer :: i

do i=1,10
  ! code with i
end do
```

You can include a step size (which can be negative) as a third parameter:

By steps of 3:

```
do i=1,10,3
  ! code with i
end do
```

Counting down:

```
do i=10,1,-1
  ! code with i
end do
```

2. While loop

The while loop has a pre-test:

```
do while (i<1000)
  print *,i
  i = i*2
end do
```

3. Exit and cycle

Loop without counter or while test:

```
do
  call random_number(x)
  if (x>.9) exit
  print *,"Nine out of ten exes agree"
end do
```

Compare to `break` in C++.

Skip rest of current iteration:

```
do i=1,100
  if (isprime(i)) cycle
  ! do something with non-prime
end do
```

Compare to `continue` in C++.

4. Labeled loops

You can label loops
useful with `exit` statement:

```
outer: do i=1,10
   inner: do j=1,10
      test: if (i*j>42) then
         print *,i,j
         exit outer
      end if test
   end do inner
end do outer
```

5. Semantic fine points

- Fortran loops determine the iteration count before execution; a loop will run that many iterations, unless you `Exit`.
- You are not allowed to alter the iteration variable.

6. Non-integer loop variables

Used to be allowed-but-dangerous

Now deleted language feature:

Code:

```
real(4) :: r

do r=.1,1.,.1
   print *,r
end do
```

Output

```
[loopf] loopr:

loopr.F90:16:5:

   16 |   do
    r=.1,1.,.1
      |     1
Warning: Deleted
    feature: Loop
    variable at (1)
    must be integer
loopr.F90:16:7:

   16 |   do
    r=.1,1.,.1
      |       1
Warning: Deleted
    feature: Start
```

Exercise 1

Read an integer and set a boolean variable to determine whether it is prime by testing for the smaller numbers if they divide that number.

Print a final message

```
Your number is prime
```

or

```
Your number is not prime: it is divisible by ....
```

where you report just one found factor.

7. Implied do loops

If you loop over a print statement, each print statement is on a new line;
use an implied loop to print on one line.

```
Print *,(2*i,i=1,20)
```

You can iterate multiple expressions:

```
Print *,(2*i,2*i+1,i=1,20)
```

These loops can be nested:

```
Print *,( (i*j,i=1,20), j=1,20 )
```

Also useful for `Read`.

Exercise 2

Use the implied do-loop mechanism to print a triangle:

```
1
2 2
3 3 3
4 4 4 4
```

up to a number that is input.