**Integers**

A standard integer uses 4 bytes, holds numbers $-2^{31}$ to $2^{31}-1$

```fortran
integer :: i

i=-(2**30-1)*2-2
print*,i
i=i-1
print*,i
```

Output:

```
-2147483648
2147483647
```

Note how $-2^{31}$ has been written to stay within range!

- $i=-(2)**31$ also works
- $i=-2**31$ should not work

A “long” integer, integer(8), is 8 bytes, holds $-2^{63}$ to $2^{63}-1$

“Short” integers: integer(2), integer(1)

Integer division: $3/2=1$, but $3./2=1.5$

We discuss the bit representation of numbers (on the board)
Floating-point numbers

A simple program which assigns values to real variables (single- and double precision; use 4 and 8 bytes):

```fortran
implicit none

real :: a
real(8) :: b

a=3.14159265358979328
print*,a
b=3.14159265358979328
print*,b
b=3.14159265358979328d0
print*,b
end
```

Output:

```
3.14159274
3.1415927410125732
3.1415926535897931
```

3.14159265358979328_8 is another way to specify double precision (8 bytes)
Complex numbers

Assignment of real and imaginary parts: \( a = (a_r, a_i) \)

```plaintext
complex :: a

a=(1.,2.)
print*,a,real(a),aimag(a)
a=a*(0.,1.)
print*,a
```

Output:

\( \begin{align*}
(1.,2.), & 1., 2. \\
(-2.,1.) &
\end{align*} \)

real(a) and aimag(a) extract real and imaginary parts

Double precision: complex(8)
Characters and character strings

Example of characters, strings and operations with them:

```
integer       ::  n
character     ::  a,b
character(10) ::  c

a='A';  b='B';  c=a//b//b//a;  n=len_trim(c)
```

```
print*,'Number of characters in c',n
print*,c(1:n),' ',c(2:3),' ',c(1:1)
print*,iachar(a),iachar(b)
print*,char(65),char(66),char(67),char(68)
```

Output:
```
Number of characters in c: 4
ABBA  BB  A
65, 66
ABCD
```
Logical (boolean) variables

Values denoted as .true. and .false. in programs
In input/output; values are given as T and F

Examples of boolean operators:
and, or, neqv (same as exclusive-or), not:

```plaintext
logical :: a,b

print*,'Give values (T/F) for a and b'
read*,a,b
print*,a.or.b,a.and.b,a.eqv.b,a.neqv.b,.not.a
```

Running this program ⇒

Give values (T/F) for a and b
T F
T F F T F
T F F T F
Arrays

Can have up to 7 dimensions. Example with integer arrays:

```fortran
integer, dimension(2,2) :: a,b
integer :: c(2)

a(1,1)=1;  a(2,1)=2;  a(1,2)=3;  a(2,2)=4
b=2*a+1
print*,a
print*,b

c=a(1,1:2)
print*,c
```

Output:

```
1,  2,  3,  4
3,  5,  7,  9
1,  3
```

Lower bound declaration:

```
Integer :: a(-10:10)
```
Kind type parameter
For simplicity, a feature of type declarations was neglected
• “8” in real(8) does not actually refer to the number of bytes
  - it is a kind type parameter

With most compilers, the kind type parameter corresponds
to the number of bytes used, but it does not have to
• with some compilers 1=single and 2=double precision

More generic way to declare a real:
• real(selected_real_kind(m,n))
  m = number of significant digits, n = exponent \(10^{-n} - 10^n\)
  the type capable of representing at least this range and
  precision will be selected by the system (error if impossible)
• real(kind(1.d0))
  the function kind(a) extracts the kind type parameter of a

Analogous for integers: selected_integer_kind(n)

In this course we will for simplicity assume that the kind type
parameter corresponds to the number of bytes (4,8 used)
Program control constructs

- Branching using `if ... endif` and `select case`
- Loops (repeated execution of code segments); `do ... enddo`
- “Jumps” with `goto label#`
Branching with “if ... endif”

If (logical_a) then
    statements_a
elseif (logical_b) then
    statements_b
...
else
    statements_else
endif

Relational operators

== .eq.
/= .ne.
> .gt.
< .lt.
>= .ge.
<= .le.

- Expressions logical_i take the values .true. or .false.
- Only statements after first true expression executed
- The else branch optional

Simpler form: if (logical_expression) statement
Example program; if.f90

```fortran
integer :: int

print*, 'Give an integer between 1 and 99'; read*, int
if (int<1.or.int>99) then
    print*, 'Read the instructions more carefully! Good bye.'
elseif (int==8.or.int==88) then
    print*, 'A lucky number; Congratulations!'
elseif (int==4.or.int==13) then
    print*, 'Bad luck...not a good number; beware!'
else
    print*, 'Nothing special with this number, '
    if (mod(int,2)==0) then
        print*, 'but it is an even number'
    else
        print*, 'but it is an odd number'
    endif
endif
```
Loops
Repeated execution of a code segment. Examples:

**Standard loop** (also valid in f77)
```
do i=1,n
   print*,i**2
endo
```

**“Infinite” loop**
```
i=0
do
   i=i+1
   print*,i**2
   if (i==n) exit
endo
```

Loop with **do while**
```
i=0
do while (i<n)
   i=i+1
   print*,i**2
endo
```

**“Jump” with go to**
```
10 i=i+1
   i2=i**2
   if (i2<sqmax) then
      print*,i,i2
      goto 10
   endif
```
Procedures; subroutines and functions

- Program units that carry out specific tasks
- Fortran 90 has internal and external procedures

**Internal subroutine**

```fortran
program someprogram
  ...
  call asub(a1,a2,...)
  ...
contains
  subroutine asub(d1,d2,...)
  ...
  end subroutine asub
end program someprogram
```

- `asub` can access all variables of the main program
- `d1, d2` are “dummy” arguments
character(80) :: word

print*, 'Give a word'; read*, word
call reverse
print*, word

contains

subroutine reverse
implicit none

integer :: i, n
character(80) :: rword

rword = ''
n = len_trim(word)
do i = 1, n
   rword(i:i) = word(n-i+1:n-i+1)
end do
word = rword

end subroutine reverse

end
character(80) :: word1, word2

print*, 'Give two words'; read*, word1, word2
call reverse(word1)
call reverse(word2)
print*, trim(word2), ' ', trim(word1)

contains

subroutine reverse(word)

implicit none

integer :: i, n
character(80) :: word, rword

rword = ''
n = len_trim(word)
do i = 1, n
   rword(i:i) = word(n-i+1:n-i+1)
endo do
word = rword

end subroutine reverse

end
character(80) :: word1, word2

print*, 'Give two words'; read*, word1, word2
call reverse(word1(1:len_trim(word1)), len_trim(word1))
call reverse(word2(1:len_trim(word2)), len_trim(word2))
print*, trim(word2), ' ', trim(word1)
end

subroutine reverse(word, n)

implicit none

integer :: i, n
character(n) :: word, rword

rword = '

do i = 1, n
   rword(i:i) = word(n-i+1:n-i+1)
enddo
word = rword
end subroutine reverse

Program writerev3.f90

- External subroutine; cannot access variables of main program
- string word declared with variable length n passed from main