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

```
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
enddo
```

“Infinite” loop

```
i=0
do
  i=i+1
  print*,i**2
  if (i==n) exit
enddo
```

Loop with `do while`

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

“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

```
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

Program writerev1.f90

```fortran
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
```

- Subroutine call without an argument list
- The string `word` can be accessed directly since `reverse` is an internal subroutine

`len_trim(string)` gives length of `string` without trailing blanks

Program writerev2.f90

```
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)
  enddo
  word=rword

  end subroutine reverse

end
```

- Subroutine calls with argument lists

- Strings `word1,word2` are passed through the dummy variable `word`

`trim(string)`
string obtained when trailing blanks removed from `string`

Program writerev3.f90

```
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
```

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

Functions (external)

```
function poly(n,a,x)

implicit none

integer :: i,n
real(8) :: poly,a(0:n),x

poly=0.0d0
do i=0,n
  poly=poly+a(i)*x**i
enddo

end function poly
```

main program:

```
...
integer :: n
real(8) :: a(0:nmax),x
real(8), external :: poly
...
print*,poly(n,a(0:n),x)
```

Accessing “global data”

Common blocks (outdated f77, but some times useful)

Global data accessible in any unit in which declarations and `common/blockname/v1,v2,...` appears

```
integer :: a,b
common/block_1/a,b
```

Modules

Global data accessible in any unit in which `use module_name` appears

```
module module_name
  integer :: a,b
end module module_name
```

Modules can also contain procedures, which are accessible only to program units using the module

Intrinsic procedures

- Many built-in functions (and some subroutines)
- In F90, many can take array argumens (not in F77)

Mathematical functions:

```
exp(x),sqrt(x),cos(x),...
```

Type conversion:

```
int(x),real(x),float(x)
```

Character and string functions:

`achar(i)` - ASCII character `i`

`iachar(c)` - # in ASCII sequence of character `c`

`len(string),len_trim(string),trim(string)`

Matrix and vector functions:

```
sum(a), matmul(m1,m2),dot_product(v1,v2)
```