

Toolchain and flags

```
gfortran -std=f2023 -Wall -Wextra -pedantic \
  -fimplicit-none -fcheck=all -g -fbacktrace \
  -ffpe-trap=invalid,zero,overflow x.f90 -o x
flang -std=f2018 -Wall -g x.f90 # LLVM Flang
ifx -stand f23 -warn all -check all -traceback x.f90
-O2 -march=native -fopenmp -fcoarray=lib
mpifort / mpirun -n 4 ./x caf / cafrun -n 4 ./x
fpm new|build|run|test|install cmake -S . -B build
.f90 free form · F90 preprocessed · .mod = module file

# fpm.toml
name = "heatflow"
[library]
[[executable]]
name = "heatflow"
source-dir = "app"
main = "main.f90"
[dependencies]
[dev-dependencies]
test-drive.git = "https://github.com/fortran-lang/test-drive"
[fortran]
implicit-typing = false
implicit-external = false
```

Program skeleton and free form

```
module m_mod
  use, intrinsic :: iso_fortran_env, only: dp => real64
  implicit none (type, external)
  private; public :: area
  contains
    pure function area(r) result(a)
      real(dp), intent(in) :: r
      real(dp) :: a
      a = acos(-1.0_dp) * r**2
    end function
  end module
program p
  use m_mod, only: area
  implicit none (type, external)
  print '(f0.3)', area(2.0_dp)
end program
! submodule (m_mod) m_impl ... end submodule

Free form: ! comment, & continues, | separates, 132 chars/line
(F2023: 10 000), 255 continuation lines (F2023: no limit), case-
insensitive, names up to 63 chars. Statement order: use, import,
implicit, declarations, executable, contains.
```

Types and kinds

type	example	kind
integer	integer(int32) :: n = 3	int8/16/32/64
real	real(dp) :: x = 1.0_dp	real32/64/128
complex	complex(dp) :: z = (1,2)	as real
logical	logical :: ok = .true.	logical8..64
character	character(len=8) :: s	default, ucs4

selected_int_kind(9), selected_real_kind(15, 307),
selected_char_kind('ISO_10646'),
selected_logical_kind(8). Literals: 1.0_dp, 42_int64,
b'101' o'7' z'FF' (BOZ), 1.5e3_dp. Default real is single: 0.1
is not 0.1_dp. parameter = named constant.
Model: kind epsilon huge tiny digits precision range
radix minexponent maxexponent bit_size storage_size;
nearest spacing rrspaceing exponent fraction scale
set_exponent.

Operators and precedence

```
** (right assoc) > * / > unary + - > binary + -
// > == /= <= >= (eq. .ne. .lt. .le. .gt. .ge.)
.not. > .and. > .or. > .eqv. .neqv. (defined:
lowest/highest)
7/2 = 3 (integer division) -7/2 = -3 mod(-7,2) = -1
modulo = 1
x = ( cond ? a : b ) ! F2023 conditional
expression
a = b ! intrinsic assignment,
auto-realloc
.and./..or. need not short-circuit: guard with nested if
Mixed mode promotes to the wider kind; real(1, dp), int(x),
nint(x), floor, ceiling convert. interface operator(+),
operator(.cross.), assignment(=) define your own.
```

Control flow

```
if (a > b) then ... else if (...) then ... else ... end if
select case (k); case (1,3); case (5:9); case default;
end select
select type (p); type is (real(dp)); class is (t);
class default
select rank (a); rank (1); rank (*); rank default; end
select
outer: do j = 1, n ! named; exit outer /
cycle outer
do i = 1, m, 2; end do ! start, stop, step
end do outer
do while (err > tol); end do do ... if (x) exit
... end do
do concurrent (i=1:n, j=1:m, mask) local(t)
  local_init(c) &
  shared(a) reduce(+:s) ! F2018 locality,
F2023 reduce
block; integer :: t; end block associate (u =>
  gwu(2:nx-1,:))
stop 0 / error stop 'msg', quiet=.true. (exit code)
```

Procedures

```
function f(x, n, opt) result(r)
  real(dp), intent(in) :: x(:) ! assumed shape
  integer, intent(in) :: n
  real(dp), intent(in), optional :: opt
  real(dp) :: r
  if (present(opt)) r = opt
end function
subroutine s(a, b); real(dp), intent(inout) :: a; end
subroutine
call s(b=2.0_dp, a=x) ! keyword arguments
pure | elemental | simple | recursive | non_recursive |
impure elemental
interface; subroutine cb(x); real(dp), intent(in) :: x;
end subroutine; end interface
abstract interface; ...; end interface
procedure(cb), pointer :: p => null()
interface gen; module procedure f_sp, f_dp; end
interface ! generic
procedure(),... value | intent(in/out/inout) |
contiguous | asynchronous
Internal procedures follow contains and see the host. Explicit
interfaces come free from modules; implicit none (external)
forbids implicit ones. Dummy intent(out) allocatables are
deallocated on entry.
```

Arrays

```
real(dp) :: a(3,4), b(0:9) ! explicit
shape, 1-based default
real(dp), allocatable :: u(:,:)
allocate(u(nx,ny), stat=s, errmsg=m)
allocate(v, source=u) allocate(w, mold=u)
deallocate(u)
real(dp) :: d(:), e(:,), f(*), g(..) ! assumed shape
/ size / rank
a(1:3:2, 2:4) a(:), j a(i, :) a(n:1:-1) a([1,3]) !
sections, vector subscript
x = [1, 2, 3] x = [(i,i, i=1,5)] x = [real(dp) ::
1, 2] reshape(x,[2,3])
where (a > 0) b = sqrt(a) elsewhere b = 0 end where
sum prod maxval minval count any all (a, dim=, mask=)
norm2 reduce
maxloc minloc findloc(a, v, back=) pack unpack merge
spread matmul transpose
size shape lbound ubound rank is_contiguous cshift
eoshift dot_product
call move_alloc(from, to) u = f(u) ! auto
(re)allocation on assignment
a([3,2]) ! F2023 list of subscripts real ::
x(...); rank(1) clause
Column-major: first index varies fastest, so loop do j; do i with
i innermost. Whole-array operations are elemental; conformable
shapes only.
```

Strings

```
character(len=:), allocatable :: s character(*),
intent(in) :: t
s = 'It's' // "ok" s = repeat('-', 20)
len(s) len_trim(s) trim(s) adjustl adjustr index(s,'x')
scan verify
lge lgt lle llt s(2:4) achar(65) iachar('A') ichar
char
call split(s, ' ', pos) call tokenize(s, ' ', tokens)
! F2023
write(buf, '(i0') n read(buf, *) n ! internal
file conversion
new_line('a') trim(adjustl(s)) upper/lower: via
achar/iachar arithmetic
Blank-padded assignment and comparison;
character(kind=selected_char_kind('ISO_10646')) for UCS-
4.
```

Input / output

```
open(newunit=u, file='out.csv', status='replace',
action='write', &
access='sequential', form='formatted', iostat=ios,
iomsg=msg)
read(u, *, iostat=ios) x write(u, '(a,f8.3)') 'T=',
t print *, x
read(u, '(a)', iostat=ios) line if
(is_iostat_end(ios)) exit
open(..., access='stream', form='unformatted')
write(u) a inquire(file=f, exist=e)
namelist /cfg/ nx, ny, alpha; read(u, nml=cfg)
close(u)

i0 int f8.3 fixed es12.4 sci g0 auto
a char l1 logical x blank / new rec
at tab 3(i4) repeat *(f6.2) unlimited advance='no'
```

Derived types and OOP

```
type :: grid_t
  integer :: nx = 0, ny = 0
  real(dp), allocatable :: u(:,:)
contains
  procedure :: max_temp ! type-bound
  procedure, pass(g) :: init
  generic :: write(formatted) => wf
  final :: cleanup
end type
type, abstract :: solver_t
contains
  procedure(step_i), deferred :: step
end type
type, extends(solver_t) :: jacobi_t
type(grid_t) :: g = grid_t(nx=4, ny=4) !
structure constructor
class(solver_t), allocatable :: s; class(*) :: any
type :: vec_T(k, n); integer, kind :: k = dp; integer,
len :: n; end type
enumeration type :: bc; enumerator :: dirichlet,
neumann; end enumeration type ! F2023
enum, bind(c); enumerator :: a = 1, b; end enum !
works today
```

Pointers

```
real(dp), pointer :: p(:,:) => null()
real(dp), target :: t(10,10)
p => t(2:9, 2:9) associated(p) associated(p, t)
nullify(p) p => null() p(1:n) => vec ! bounds
remap
procedure(f), pointer :: fp => f_impl
Prefer allocatable; use pointer for aliasing, linked data and
procedure pointers. Undefined, disassociated, associated: test with
associated, never compare. Not allocatable: no automatic
cleanup.
```

Intrinsic modules

```
use, intrinsic :: iso_fortran_env, only: int32, real64,
&
input_unit, output_unit, error_unit,
compiler_version
use, intrinsic :: iso_c_binding, only: c_int, c_double,
c_ptr, c_loc
use, intrinsic :: ieee_arithmetic
ieee_is_nan(x) ieee_is_finite(x) ieee_value(x,
ieee_quiet_nan)
call ieee_get_flag(ieee_overflow, f)
ieee_support_halting(...)
use, intrinsic :: ieee_exceptions; use omp_lib (not
intrinsic)
Compare reals with a tolerance: abs(a-b) <=
tol*max(abs(a),abs(b)), never ==.
```

Parallel

```
real(dp) :: x(100)[*] ! coarray: one copy per
image
me = this_image() n = num_images() x(1)[2] = 0 !
coindexed
sync all sync images([1,2]) sync memory critical
... end critical
call co_sum(s) co_max co_min co_broadcast(a,
source_image=1) co_reduce
form team(1, t) change team (t) ... end team event
post(ev[2]) event wait(ev)
x(1)[2, notify=nt] = 0 notify wait(nt) ! F2023
!$omp parallel do reduction(+:s) collapse(2)
schedule(static)
!$acc parallel loop copyin(a) copyout(b)
use mpi_f08; call MPI_Init(); call
MPI_Comm_rank(MPI_COMM_WORLD, r)
call MPI_Allreduce(MPI_IN_PLACE, s, 1,
MPI_DOUBLE_PRECISION, MPI_SUM, MPI_COMM_WORLD)
```

Interoperability

```
function step(n, u) bind(c, name='heatflow_step')
result(r)
  integer(c_int), value :: n; real(c_double),
intent(inout) :: u(*)
  call c_f_pointer(cptr, f, [n]) c_loc(x)
  c_associated(p) f_c_string(s)
# C: extern int heatflow_step(int, double *); link -
lgfortran
python -m numpy.f2py -c k.f90 -m k # + .f2py_f2cmap
for dp, Meson backend
subroutine mexFunction(nlhs, plhs, nrhs, prhs) !
MATLAB: mex k.F90
mwPointer :: plhs(*), prhs(*); mwSize; mxGetDoubles
; mxCopyPtrToReal8
type(*), dimension(..) :: a ! C descriptor CFI_cdesc_t
(ISO_Fortran_binding.h)
```

Best practices and gotchas

Do: implicit none (type, external); modules everywhere,
private by default, use ..., only:: intent on every dummy;
kinds from one module; allocatable over pointer;
pure/elemental kernels; array syntax; -fcheck=all in debug.
Gotchas: initialised local = implicit save (integer :: n = 0);
0.1 is single precision, use 0.1_dp; integer division truncates; no
guaranteed short-circuit; loop order vs column-major; indexing is 1-
based; intent(out) deallocates; aliasing of dummies is illegal; ==
on reals; real*8 is non-standard; print * spacing is processor-
dependent; stat/iostat or the program aborts.

Legacy → modern (standard that made the change)

fixed form → free form (F90)	real*8 / double precision → real(dp) (F90)	common → modules (F90)	equivalence → transfer / pointers (F90)
statement functions → internal functions (F90; obsolescent)	arithmetic if (deleted F2018) / computed go to (obsolescent) → if / select case	forall → do concurrent (F2018 obsolescent)	implicit typing → implicit none (obsolescent in F2028)
ifort → ifx (ifort discontinued in oneAPI 2025.0)	flang-new → flang (LLVM 20)	f2py distutils → Meson (NumPy 2.x)	fixed-size work arrays → allocatable (F90/F2003)