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
endo

end function poly

main program:

... integer :: n
real(8) :: a(0:nmax),x
real(8), external :: poly
...
pick*,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**Common blocks**  (outdated f77, but some times useful)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

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

**Modules**

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

```fortran
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 arguments (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)
Bit manipulations

Operate on the bits of integers (0,...,31 for 4-byte integer)

**Single-bit functions (b=bit#):**
btest(i,b) - .true. or .false.
ibset(i,b), ibclr(i,b) - integer

**All-bit functions (pair-wise on two integers):**
iand(i,j), ior(i,j), ieor(i,j) - integer

```fortran
function bits(int)
integer :: i,int
character(32) :: bits
bits='00000000000000000000000000000000'
do i=0,31
   if (btest(int,i)) bits(32-i:32-i)='1'
endo
dend function bits
```
Processor time subroutine

cpu_time(t) - t = seconds after start of execution

```fortran
integer :: i,nloop
real(8) :: sum
real    :: time0,time1

print*,'Number of operations in each loop'
read*,nloop

sum=0.0d0; call cpu_time(time0)
do i=1,nloop
   sum=sum+dfloat(i)*dfloat(i)
enddo
call cpu_time(time1)
print*,'Time used for s=s+i*i: ',time1-time0
```
Files

- A file has a name on disk, associated unit number in program
- File “connected” by `open` statement

```fortran
open(unit=10, file='a.dat')
  associates unit 10 with file a.dat
open(10, file='a.dat')
  “unit” does not have to be written out
open(10, file='a.dat', status='old')
  ‘old’ file already exists (‘new’, ‘replace’)
open(10, file='a.dat', status='old', access='append')
  to append existing file with new data
```

Reading and writing files:

```fortran
read(10, *) a
write(10, *) b
```
Output formatting

aa(1)=1; aa(2)=10; aa(3)=100; aa(4)=1000
bb(1)=1.d0; bb(2)=1.d1; bb(3)=1.d2; bb(4)=1.d3
print'(4i5)', aa
write(*,'(4i5)')aa
write(*,10)aa
10 format(4i5)
print'(4i3)', aa
print'(a,i1,a,i2,a,i3)',' one:', aa(1), ' ten:', aa(2)
print'(4f12.6)', bb

1  10  100  1000
1  10  100  1000
1  10  100  1000
1  10  100  1000
1 10100***
one:1  ten:10
  1.000000  10.000000  100.000000  1000.000000
Allocatable arrays

Mechanism to assign the size of an array when running the program (i.e., not fixed when compiling)

```fortran
integer :: m,n
real(8), allocatable :: matr(:,:)

write(*,*)'Give matrix dimensions m,n: '; read*,m,n
allocate(matr(m,n))
...
dallocate(matr)
```

To change the size of an already allocated array, it first has to be de-allocated, then allocated again.
Variable-sized arrays, interfaces, assumed-shape, and automatic

```fortran
integer :: m,n
real(8), allocatable :: matr(:,,:)

Interface
  subroutine checkmatr(matr) ! Declaring the interface
    real(8) :: matr(:,,:)
  end subroutine checkmatr
end interface

write(*,*)'Give matrix dimensions m,n: '; read*,m,n
allocate(matr(m,n))
call checkmatr(matr)
end

subroutine checkmatr(matr) ! of a procedure (include in all procedures that need it, e.g., when using “assumed shape”
  real(8) :: matr(:,,:)
  real(8) :: localmatr(size(matr,1),size(matr,2))
  print*,size(localmatr)
  print*,shape(localmatr)
end subroutine checkmatr
```