

# Pengantar Fortran 90



**Imam Fachruddin**

Departemen Fisika, Universitas Indonesia

Untuk dipakai dalam kuliah Analisis Numerik

Dapat diunduh dari <http://staff.fisika.ui.ac.id/imamf/>



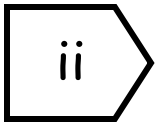
# Pengantar Fortran 90

Imam Fachruddin

Departemen Fisika, Universitas Indonesia

## Daftar Pustaka:

- M. Metcalf & J. Reid, **Fortran 90/95 Explained**, Oxford University Press, New York, 1998
- **Fortran 90 Tutorial** (<http://wwwasdoc.web.cern.ch/wwwasdoc/f90.html>)

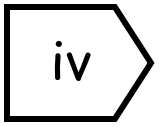


# Isi

---

• struktur program Fortran 90	1
• tipe data	5
• konstanta & variabel	7
• operator	11
• cabang	17
• loop	23
• subprogram	27
• input/output (I/O)	33

---



# Struktur Program Fortran 90

**PROGRAM** nama\_program

[deklarasi: variabel, konstanta dll]

[isi program]

**END PROGRAM** nama\_program

[program utama]

**CONTAINS**

[subprogram:

- subroutine

- function]

2

PROGRAM coba

! last edited: August 23, 2005 by Imam

Komentar diawali oleh tanda "!".

IMPLICIT NONE

Dengan ini pemrogram bisa memberikan nama untuk variabel dan konstanta secara bebas (tentu saja asal berbeda dari nama-nama milik F90).

REAL :: a,b,c,d

! Beri nilai a dan b

a=3.0

b=0.7

Program Fortran 90 (F90) bersifat "case insensitive". Ada baiknya membedakan "case" nama-nama milik F90 (mis. uppercase) dari "case" nama-nama buatan si pemrogram (mis. lowercase).

! Hitung nilai c dan d

c=(a+b)\*\*2.0 ! c yaitu kuadrat dari jumlah a dan b

d=0.5\*c ! d yaitu setengah dari c

a= 10.0 &

+b &

-20.0\*c-d

Pernyataan bisa lebih dari satu baris, maximum 40 baris. Tanda "&" di akhir baris berarti pernyataan berlanjut ke baris berikutnya.

STOP

END PROGRAM coba





```
PROGRAM coba
```

```
IMPLICIT NONE
```

```
REAL :: a,b,c,d
```

```
a=3.0
```

```
b=0.7
```

```
CALL tambah2(a,b,c)
```

```
d=paruh(c)
```

```
STOP
```

```
CONTAINS
```



```
SUBROUTINE tambah2(x,y,z)
```

```
IMPLICIT NONE
```

```
REAL, INTENT(IN) :: x,y
```

```
REAL, INTENT(OUT) :: z
```

```
z=(x+y)**2.0
```

```
RETURN
```

```
END SUBROUTINE tambah2
```



```
FUNCTION paruh(x) RESULT(z)
```

```
IMPLICIT NONE
```

```
REAL, INTENT(IN) :: x
```

```
REAL :: z
```

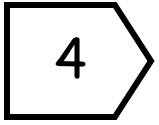
```
z=0.5*x
```

```
RETURN
```

```
END FUNCTION paruh
```

```
END PROGRAM coba
```

Nama yang dideklarasikan di subprogram berlaku hanya di subprogram itu. Nama yang dideklarasikan di program utama berlaku di seluruh program, kecuali jika nama itu dideklarasikan ulang di subprogram. Dalam hal ini, deklarasi di program utama untuk nama itu tidak berlaku dalam subprogram tersebut.



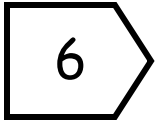
# Tipe Data

## Intrinsic:

- integer : 0, 10, 3, -8, -300
- real : 0.0, -0.4, 34.28, -1.2
- complex : (2.0,4.5), (-2.1,0.3)
- character : "a", 'QED', 'Huruf "A"', nih'
- logical : `.TRUE.`, `.FALSE.`, `2 > 3`

## Derived

Di dalam program orang mendefinisikan konstanta dan variabel untuk menyimpan data tipe tertentu.



# Konstanta & Variabel

Konstanta & variabel didefinisikan untuk menyimpan data tipe tertentu. Nilai yang diberikan pada konstanta (nilai konstanta) **tetap**, sedangkan nilai variabel **dapat berubah**.

Jenis konstanta & variabel:

Contoh analogi:

- scalar ————— konstanta / variabel scalar  
berisi **hanya satu nilai** massa
- array ————— konstanta / variabel array  
berisi **beberapa elemen data** kecepatan  
(3 komponen  
/ dimensi)
- (variabel) pointer

## IMPLICIT NONE

INTEGER, PARAMETER :: kilo=1000, mega=1000\*kilo

REAL, PARAMETER :: gravitasi=9.8, smallest=1.0e-4

COMPLEX, PARAMETER :: bili=CMPLX(0.0,1.0)

LOGICAL, PARAMETER :: flag1=.TRUE., flag2=.FALSE.

CHARACTER(\*), PARAMETER :: alamat='Kampus UI, Depok 16424'

INTEGER :: i,j

REAL :: massa, speed

COMPLEX :: indeksbias

LOGICAL :: benaratausalah

CHARACTER(9) :: nip, npm, nik

INTEGER, DIMENSION(3,5) :: gigi, gaga

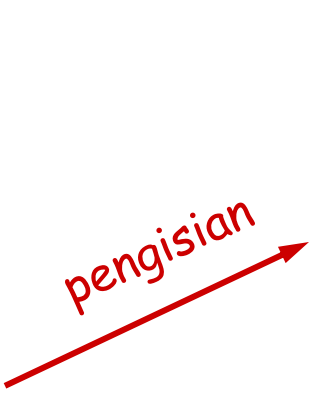
REAL, DIMENSION(10) :: panjang, lebar

COMPLEX, DIMENSION(3,5,4) :: realita

LOGICAL, DIMENSION(1,2,1,4) :: flag0

CHARACTER(80), DIMENSION(60) :: halaman

*pengisian*



A red arrow labeled 'pengisian' points from the text 'INTEGER, DIMENSION(3,5) :: gigi, gaga' to the table below. The table is a 3x5 grid with columns separated by vertical green lines. The first column contains the numbers 1, 2, 3. The second column contains 1, 2, 3, 4, 5. The third column contains 1, 2, 3. The fourth and fifth columns are empty. A red arrow points downwards along the first column, and another red arrow points to the right below the table.

1	1	1		
2	2	2		
3	3	3		
	4	4		
	5			

IMPLICIT NONE

INTEGER :: i

REAL :: r

COMPLEX :: c

CHARACTER(4) :: kode

CHARACTER(8) :: nama

r=7/2

i=7/2

i=-7/2

i=3/(2+3)

r = 3.5

i = 3

i = -3

i = 0

c=CMPLX(2.5)

c=CMPLX(2,3)

c = (2.5,0.0)

c = (2.0,3.0)

kode='AB'

kode="ABCDEFGH"

kode = "AB. . "

kode = "ABCD"

Pada contoh ini  
tempat kosong (blank)  
ditandai oleh ".".

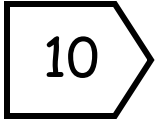
nama="ABI&

&MANYU"

nama = "ABIMANYU"

kode=nama(2:5)

kode = 'BIMA'

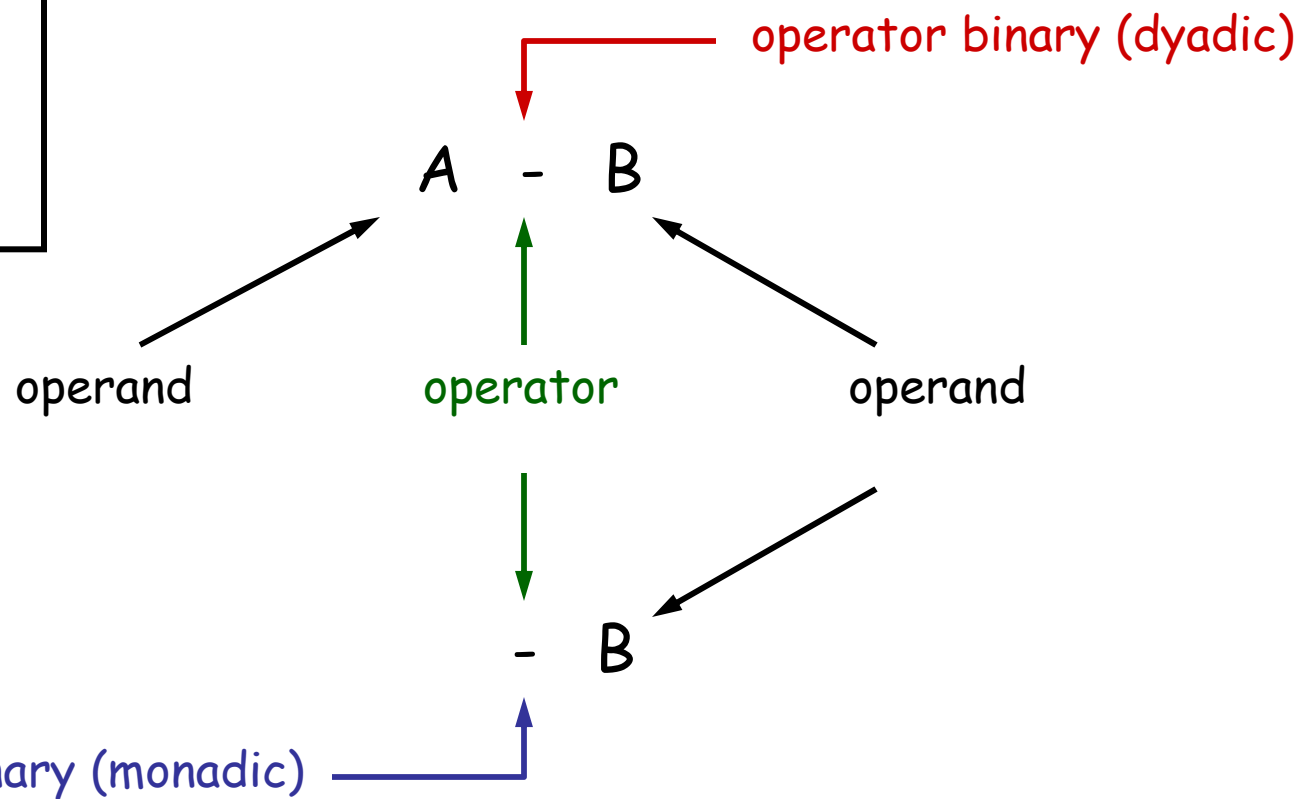




# Operator

operator:

- intrinsic
- defined



## Operator Intrinsic

	operator	sifat	tipe operasi	tipe operand	tipe hasil
<div>tinggi</div> <div>↑</div> <div>hirarki</div> <div>↓</div> <div>rendah</div>	**	binary	numerik	integer, real, complex	integer, real, complex
	*, /	binary	numerik	integer, real, complex	integer, real, complex
	+, -	u- & bi-nary	numerik	integer, real, complex	integer, real, complex
	//	binary	character	character	character
	.EQ. (==), .NE. (/=), .LT. (<), .LE. (<=), .GT. (>), .GE. (>=)	binary	relasi	integer, real, complex (==, /=), character (==)	logical
	.NOT.	unary	logical	logical	logical
	.AND.	binary	logical	logical	logical
	.OR.	binary	logical	logical	logical
	.EQV., .NEQV.	binary	logical	logical	logical

Keterangan: • \*\* : pangkat

• // : penjumlahan/penggabungan/penyambungan karakter

• == : sama, /= : tidak sama, <= : kurang atau sama, >= lebih atau sama

• .EQV. : logical equivalence, .NEQV. : logical non-equivalence

# Tipe Data Operasi Numerik

$$a (**, *, /, +, -) b$$

tipe a            tipe b            tipe hasil

I	I	I
I	R	R
I	C	C
R	I	R
R	R	R
R	C	C
C	I	C
C	R	C
C	C	C

I : integer  
R : real  
C : complex

IMPLICIT NONE

INTEGER :: i,j,k

REAL :: r,s

COMPLEX :: c,d

CHARACTER(4) :: nama1,nama2

CHARACTER(8) :: nama

LOGICAL :: bs,cek,flag

...

i=(-k+2)\*j/10

r=(s\*\*(2-3\*i)-2.0)\*1.0e-12

c=CMPLX(1.2,-0.3)\*i-20.0\*d

i=i\*2

s=2.0\*\*s

$i_{\text{baru}} = i_{\text{lama}} * 2$   
 $s_{\text{baru}} = 2.0 ** s_{\text{lama}}$

nama1='SENO'

nama2="PATI"

nama=nama1//nama2

nama = 'SENOPATI'

nama2=nama(1:3)//nama(6:6)

nama=nama(5:8)//nama(1:4)

nama2 = 'SENA'  
 nama = 'PATISENO'

bs=10 .NEQ. (2\*5)

bs=0.4 >= 0.0

cek=nama == 'SENOPATI'

flag=d .EQ. c

flag=i > r

bs = .FALSE.  
 bs = .TRUE.  
 cek = .FALSE.  
 nilai flag bergantung  
 pada d, c, i, r

bs= .NOT. cek

bs=cek .OR. bs

cek=bs .AND. cek

cek= .NOT. (cek .OR. .NOT. bs)

bs = .TRUE.  
 bs = .TRUE.  
 cek = .FALSE.  
 cek = .TRUE.

# IMPLICIT NONE

REAL :: s

REAL, DIMENSION(5) :: u,v

REAL, DIMENSION(10,20) :: p,q,r

CHARACTER(4), DIMENSION(2) :: nama

LOGICAL, DIMENSION(5) :: bs

i = 1,...,10  
j = 1,...,20  
k = 1,...,5

...

p=q\*r+q\*\*r-q/r

u=30.0\*v-10.0\*\*v

q=s

bs=u <= v

$p(i,j) = q(i,j)*r(i,j)+q(i,j)**r(i,j)-q(i,j)/r(i,j)$   
 $u(k) = 30.0*v(k)-10.0**v(k)$   
 $q(i,j) = s$   
 $bs(k) = u(k) <= v(k)$

u=p(1:5,1)/v

r(1,13:17)=v+q(3,10:14)

q(2,:)=r(10,:)

$u(k) = p(k,1)/v(k)$   
 $r(1,k+12) = v(k)+q(3,k+9)$   
 $q(2,j) = r(10,j)$

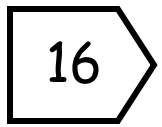
u(3:5)=u(1:3)

$u(3)_{\text{baru}} = u(1)_{\text{lama}}, u(4)_{\text{baru}} = u(2)_{\text{lama}}, u(5)_{\text{baru}} = u(3)_{\text{lama}}$

nama(1)='SANI'

nama(2)=nama(1)(3:4)//nama(1)(1:2)

nama(2) = 'NISA'



# Cabang

## GOTO

Perintah **GOTO** dipakai untuk melompat ke bagian tertentu dari sebuah program. Bagian yang menjadi target perintah **GOTO** harus diberi label.

Perintah:

**GOTO** label

ekspresi bilangan, panjangnya 1 sampai maksimum 5 digit, angka 0 di depan (leading zeros) tidak berarti (0100 sama dengan 100)

Contoh:

```
x=2.0**i
GOTO 110
z=100.*x
110 z=0.1*x
    y=5.0+z
```

← baris ini diloncati, tidak dijalankan

Label hanya berlaku dalam program / subprogram tempat label itu dibuat.

# IF

Perintah **IF** dipakai untuk mengontrol cabang seperti di bawah.



Bentuk:

**IF** (syarat) tindakan

ekspresi logical  
scalar

**IF** (syarat) **THEN**  
    blok tindakan  
**END IF**

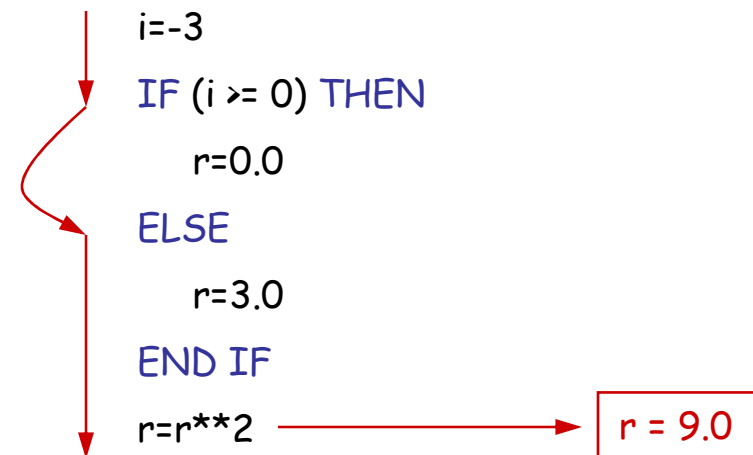
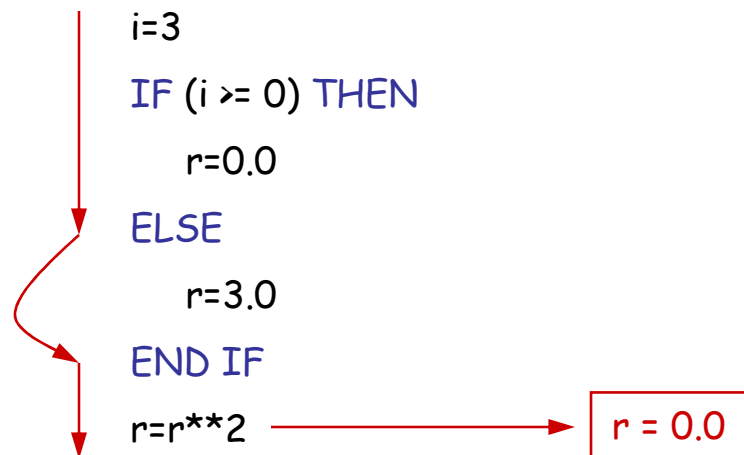
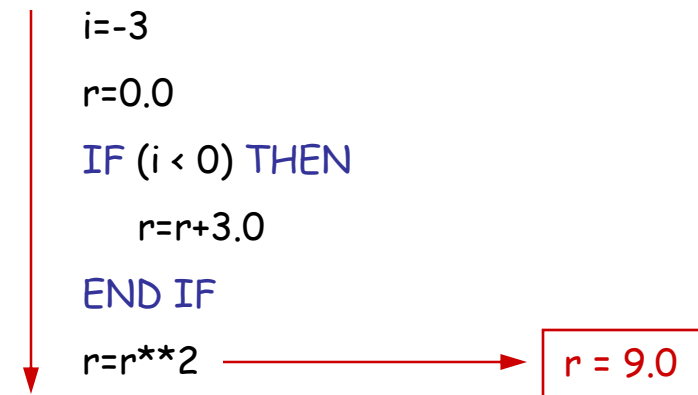
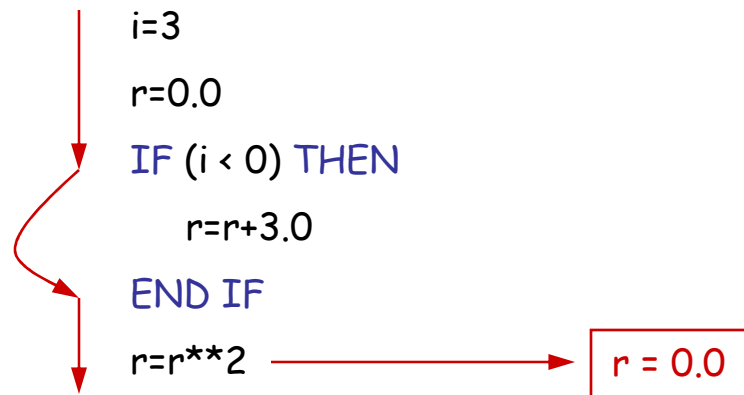
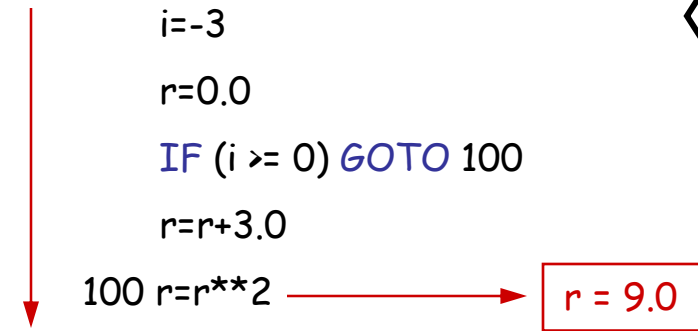
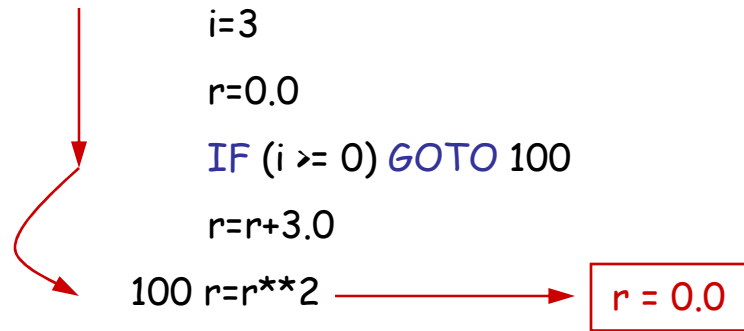
bisa terdiri  
atas beberapa  
pernyataan

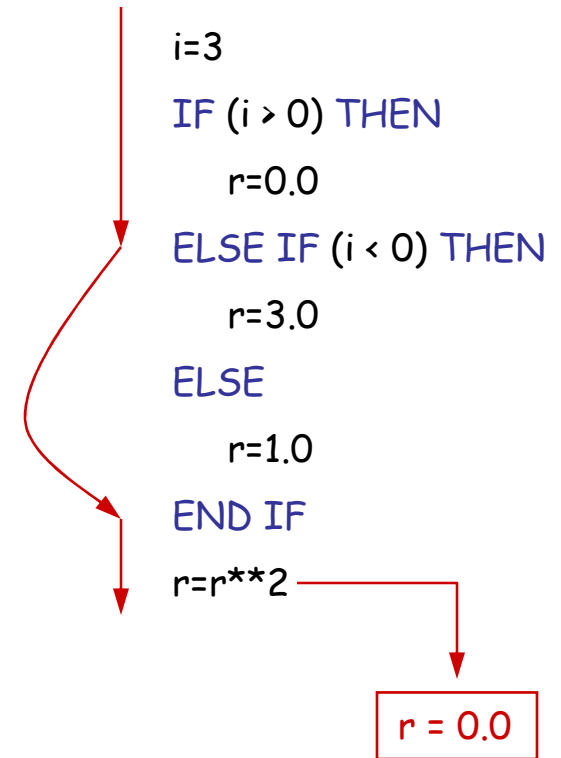
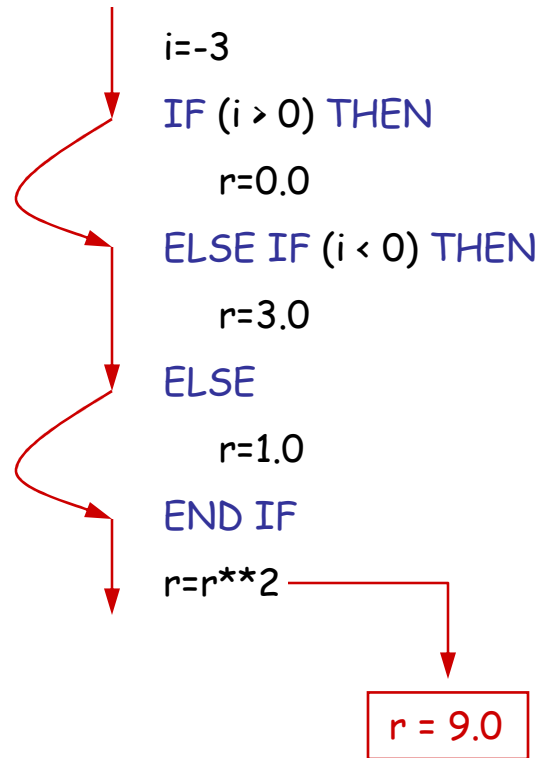
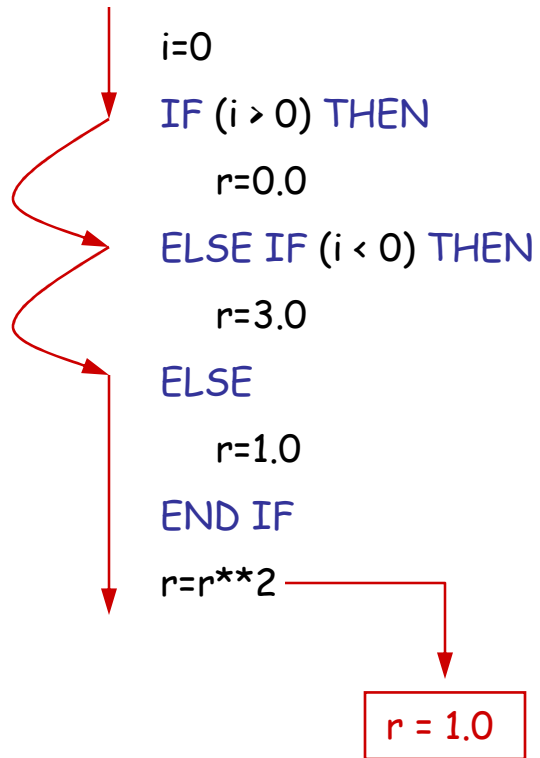
**IF** (syarat) **THEN**  
    blok tindakan ini  
**ELSE**  
    blok tindakan itu  
**END IF**

```

IF (syarat_1) THEN
    blok tindakan 1
ELSE IF (syarat_2) THEN
    blok tindakan 2
ELSE IF (syarat_3) THEN
    blok tindakan 3
...
ELSE
    blok tindakan lain
END IF
  
```

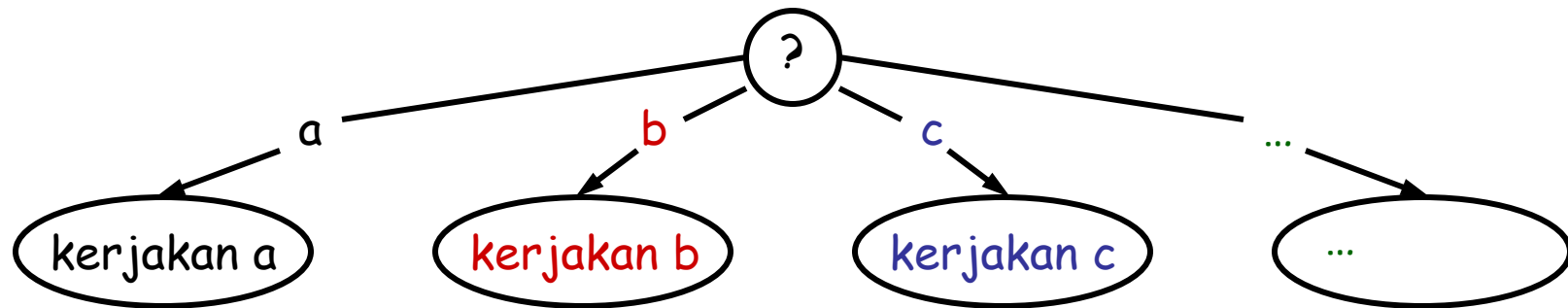






# CASE

Perintah **CASE** dipakai untuk mengontrol cabang seperti di bawah.



Bentuk:

```

SELECT CASE (selektor)
CASE (nilai_1)
    blok tindakan_1
...
CASE DEFAULT
    blok tindakan_lain
END SELECT
  
```

variabel integer / character  
/ logical scalar

nilai selektor untuk kasus 1

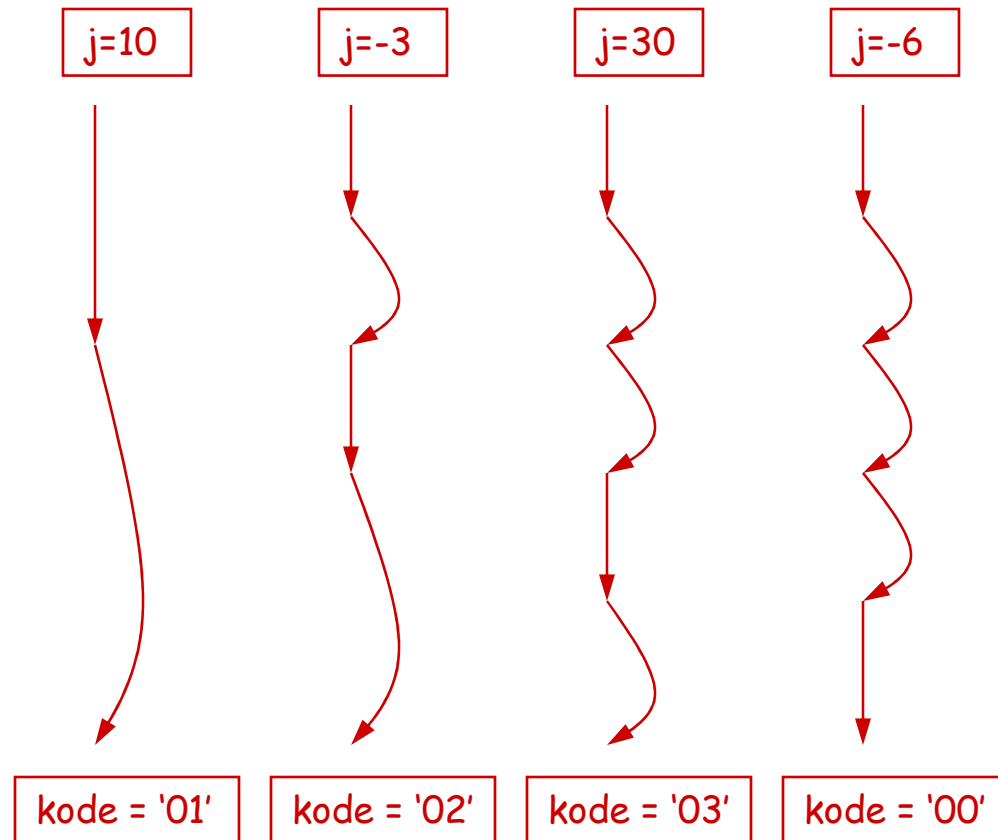
tidak harus ada;  
jika ada berarti kasus  
untuk nilai selektor yang  
lain yang mungkin



```

SELECT CASE (j)
  CASE (0,3,-7,10)
    kode='01'
  CASE (-5:-1,13)
    kode='02'
  CASE (: -10,20:)
    kode='03'
  CASE DEFAULT
    kode='00'
END SELECT

```



```

SELECT CASE (kode)
  CASE ('01','02','03')
    x=2.0*i
  CASE ('00')
    x=-2.0*i
END SELECT

```

```

SELECT CASE (cerita)
  CASE (.TRUE.)
    laporan='Ini kisah nyata.'
  CASE DEFAULT
    laporan='Ini bukan kisah nyata.'
END SELECT

```

# Loop

## DO

Pekerjaan yang berulang dapat diselesaikan dengan perintah DO.

Bentuk:

```
DO counter=nilai_awal,nilai_akhir,step  
    blok pekerjaan  
END DO
```



Keterangan:

- counter merupakan variabel integer scalar
- nilai\_awal, nilai\_akhir dan step bisa berupa bilangan atau ekspresi integer scalar (mis.  $j*k$ ,  $k+j$ ,  $2*(k-j)$ )
- jika step tidak diberikan maka dianggap  $step = 1$

lakukan blok pekerjaan  
untuk nilai counter =  
nilai\_awal sampai nilai  
counter = nilai\_akhir,  
dengan perubahan nilai  
counter sebesar step

24

x=0.0

DO i=1,8,2

x=x+i

END DO

y=x\*\*2

y=256.0

i=1  
x=1.0

i=3  
x=4.0

i=5  
x=9.0

i=7  
x=16.0

x=0.0

DO i=7,0,-2

x=x+i

END DO

y=x\*\*2

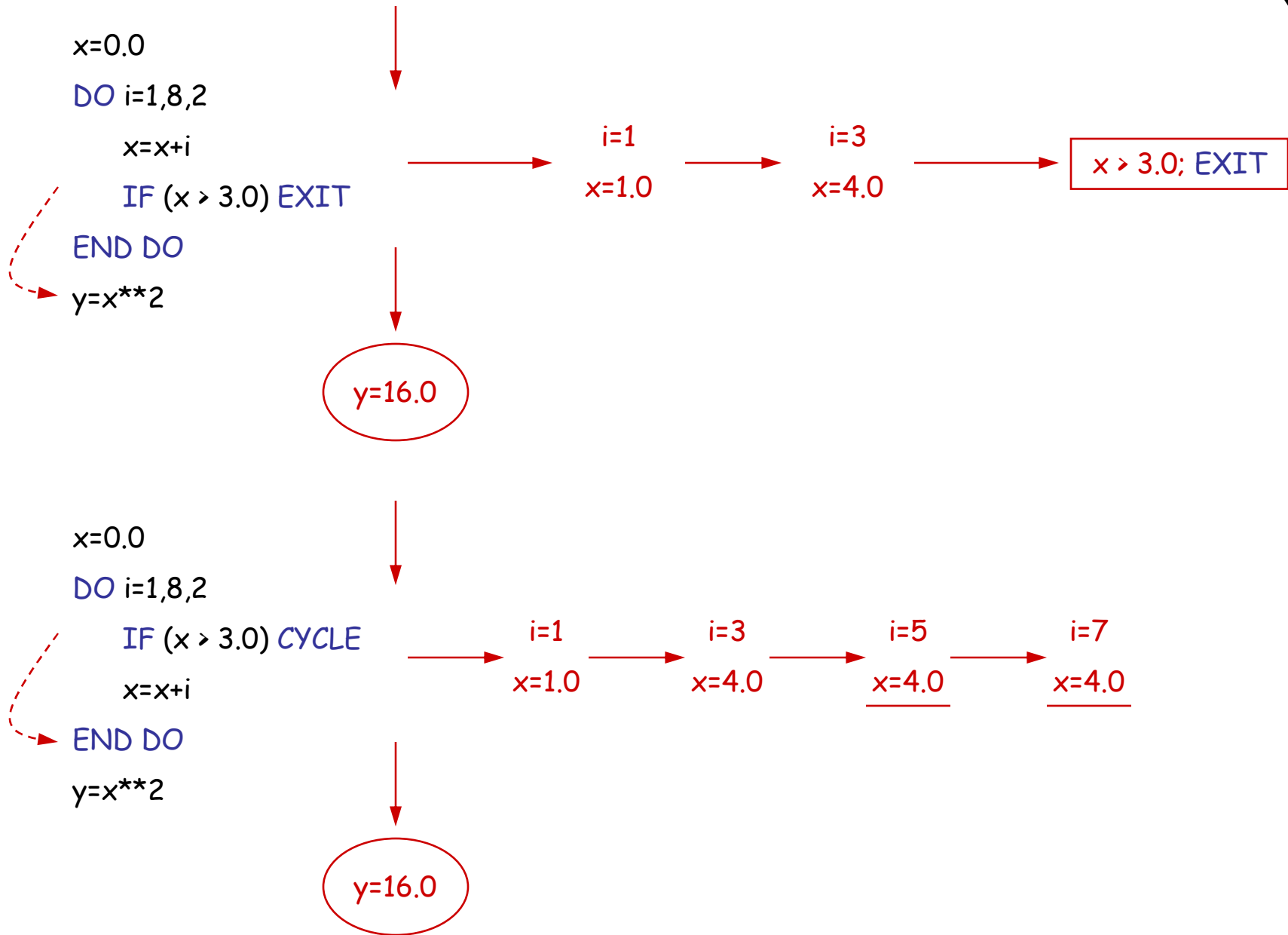
y=256.0

i=7  
x=7.0

i=5  
x=12.0

i=3  
x=15.0

i=1  
x=16.0



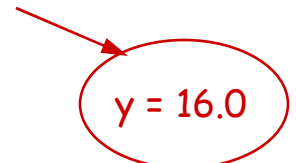
## DO Tak Berbatas

Jika counter tidak diberikan, maka loop **DO** akan terus berjalan tanpa henti.  
Agar loop itu berhenti, maka harus ada perintah **EXIT** yang menyebabkan keluar dari blok **DO ... END DO**.

```
DO
...
pekerjaan
IF (syarat_loop_berhenti) EXIT
...
END DO
```

Contoh:

```
i=-1
x=0.0
DO
    i=i+2
    x=x+i
    IF (x > 3.0) EXIT
END DO
y=x**2
```



y = 16.0



# Subprogram

Sebuah program bisa saja terdiri hanya dari program utama. Namun, **sebaiknya sebuah program disusun atas satu program utama dan beberapa subprogram**. Tiap subprogram menjalankan satu pekerjaan tertentu.

Salah satu keuntungan, penulisan program dan pencarian kesalahan lebih mudah dilakukan, pemrogram dapat berkonsentrasi pada satu bagian program (subprogram/program utama) dan tidak mengusik bagian program yang lain.

Contoh keuntungan lain, jika ada suatu pekerjaan yang dilakukan di beberapa tempat dalam sebuah program, maka sepotong program untuk pekerjaan itu cukup dibuat sekali dalam sebuah subprogram.

Dalam Fortran 90 juga telah tersedia banyak subprogram intrinsic.

Subprogram dipanggil (dijalankan) oleh program utama atau oleh sesama subprogram. Bentuknya berupa **SUBROUTINE** atau **FUNCTION**.

Struktur **SUBROUTINE**:

```
SUBROUTINE nama_subroutine(argument)
[deklarasi]
[isi subroutine]
END SUBROUTINE nama_subroutine
```

Contoh:

```
SUBROUTINE tambah2(x,y,z)

IMPLICIT NONE
REAL, INTENT(IN) :: x,y
REAL, INTENT(OUT) :: z
REAL :: c

c=x+y
z=c**2.0

RETURN

END SUBROUTINE tambah2
```

**INTENT:**

- **IN** - variabel input, nilainya tidak boleh diubah
- **OUT** - variabel output
- **INOUT** - variabel input dan juga output, nilainya boleh diubah (nilainya ketika keluar dari subroutine bisa berbeda dari nilainya ketika masuk)

**SUBROUTINE** dipanggil dengan perintah **CALL**, contoh:

```
CALL tambah2(a,b,c)
```

Struktur **FUNCTION**:

```
FUNCTION nama_function(argument) RESULT(variabel_output)
[deklarasi]
[isi function]
END FUNCTION nama_function
```

Contoh:

```
FUNCTION paruh(x) RESULT(z)

IMPLICIT NONE
REAL, PARAMETER :: y=0.5
REAL, INTENT(IN) :: x
REAL :: z

z=x*y

RETURN

END FUNCTION paruh
```

argument **FUNCTION** hanya boleh ber-**INTENT IN** (argument harus variabel input)

variabel output tidak perlu diberi **INTENT OUT**

**FUNCTION** dipanggil tanpa perintah **CALL**, contoh:

paruh(c)

a\*paruh(c)+b



PROGRAM coba

IMPLICIT NONE

REAL :: a,b,c,d

a=3.0

b=0.7

CALL tambah2(a,b,c)

d=paruh(c)

STOP

CONTAINS



SUBROUTINE tambah2(x,y,z)

IMPLICIT NONE

REAL, INTENT(IN) :: x,y

REAL, INTENT(OUT) :: z

REAL :: c

c=x+y

z=c\*\*2.0

RETURN

END SUBROUTINE tambah2



FUNCTION paruh(x) RESULT(z)

IMPLICIT NONE

REAL, INTENT(IN) :: x

REAL :: z

z=0.5\*x

RETURN

END FUNCTION paruh

END PROGRAM coba

# Subprogram Recursive

Subprogram recursive yaitu, subprogram yang bisa dipanggil oleh dirinya sendiri.

Struktur:

```
RECURSIVE SUBROUTINE nama_subroutine(argument)
[deklarasi]
[isi subroutine]
END SUBROUTINE nama_subroutine
```

```
RECURSIVE FUNCTION nama_function(argument) RESULT(variabel_output)
[deklarasi]
[isi function]
END FUNCTION nama_function
```

Cara memanggil subprogram recursive sama dengan cara memanggil subprogram bukan recursive.

## RECURSIVE FUNCTION yang menghitung nilai factorial:

RECURSIVE FUNCTION factorial(n) RESULT(m)

IMPLICIT NONE

INTEGER, INTENT(IN) :: n

INTEGER :: m

IF (n == 1 .OR. n == 0) THEN

m=1

ELSE IF (n > 1) THEN

m=n\*factorial(n-1)

ELSE

STOP

END IF

RETURN

END FUNCTION factorial

$m = n! = n * (n-1) * (n-2) * \dots * 1$

Program berhenti jika n negatif.

# Input/Output (I/O)

Sebuah program mungkin memerlukan data yang dibaca dari, misal, sebuah file, keyboard. Juga, sebuah program mungkin perlu menuliskan data ke, misal, layar monitor, sebuah file.

Perintah I/O:

- `READ(unit, format) daftar_input`
- `WRITE(unit, format) daftar_output`

Keterangan:

- unit menentukan dari / ke device mana input / output dibaca / ditulis; contoh device: keyboard, monitor, file, printer dll.
- format menentukan format data yang dibaca / ditulis.
- `daftar_input` harus variabel, sedangkan `daftar_output` bisa variabel atau ekspresi character

# Unit

Contoh untuk device **file internal**, **file eksternal** dan **terminal**. Terminal secara praktis berarti keyboard (untuk **READ**) dan monitor (untuk **WRITE**).

**terminal**

---

```

READ(*, format) daftar_input
...
WRITE(*, format) daftar_output
  
```

---

```

unit = *;
membaca dari keyboard / menulis
ke monitor
  
```

**file  
internal**

---

```

CHARACTER(10) :: buffer
...
READ(buffer, format) daftar_input
...
WRITE(buffer, format) daftar_output
  
```

---

```

unit = buffer (variabel character);
membaca / menulis dari / ke
buffer sebagai file internal
  
```

**file  
eksternal**

---

```

OPEN(nunit, argument_lain)
...
READ(nunit, format) daftar_input
...
WRITE(nunit, format) daftar_output
...
CLOSE(nunit)
  
```

---

```

OPEN file eksternal, beri nomor
unit nunit;
membaca / menulis dari / ke unit
dengan nomor unit nunit (file
eksternal);
CLOSE file eksternal
  
```



# OPEN & CLOSE File Eksternal

Perintah:



- `OPEN(nunit, FILE='nama_file', STATUS='status_file', ACTION='tindakan')`
- `CLOSE(nunit)`

Keterangan:

- `nunit` yaitu nomor unit, biasanya bilangan 1 - 99.
- `FILE` berisi nama file eksternal yang dibuka sebagai unit nomor `nunit`.
- `STATUS` berisi `NEW` (buat file baru) / `OLD` (file sudah ada) / `REPLACE` (mengganti file yang sudah ada)
- `ACTION` berisi `READ` (file hanya untuk dibaca) / `WRITE` (file hanya untuk ditulis) / `READWRITE` (file untuk dibaca dan ditulis)

Contoh:

```
OPEN(5, FILE='hasil.out', STATUS='NEW', ACTION='WRITE')  
WRITE(5, format) daftar_output  
CLOSE(5)
```

# Format

Format merupakan ekspresi character "case insensitive" yang diawali dan diakhiri oleh tanda kurung, contoh: '(3F5.2)' atau "(3F5.2)".

Tiga cara memberikan keterangan format pada perintah **READ** dan **WRITE**:

---

```
READ(unit, "(3F5.2)") daftar_input
```

```
...
```

```
WRITE(unit, '(3F5.2)') daftar_output
```

---

Format dituliskan langsung pada perintah **READ** dan **WRITE**.

---

```
CHARACTER(*), PARAMETER :: fmt='(3F5.2)'
```

```
...
```

```
READ(unit, fmt) daftar_input
```

```
...
```

```
WRITE(unit, fmt) daftar_output
```

---

Format diberikan oleh konstanta character atau variabel character yang telah diberi nilai.

---

```
READ(unit, 100) daftar_input
```

```
...
```

```
WRITE(unit, 100) daftar_output
```

```
...
```

```
100 FORMAT(3F5.2)
```

---

Format diberikan dengan merujuk ke sebuah LABEL (dalam contoh ini "100") yang berisi perintah **FORMAT**.

# Edit Descriptor

Ekspresi character yang membentuk format tersusun dari edit descriptor.

Pada perintah **READ**, edit descriptor menentukan bagaimana ekspresi data (susunan character) yang dibaca harus diartikan sebagai sebuah nilai.

109 dibaca sebagai nilai, simbol atau nama?

Pada perintah **WRITE**, edit descriptor menentukan bagaimana sebuah nilai harus ditampilkan sebagai suatu susunan character.

nilai 0.01306 ditulis sebagai 0.013, 0.0131 atau 0.13E-01?

Edit descriptor juga mempunyai fungsi kontrol, seperti mengatur posisi data yang dibaca / ditulis, menghentikan pembacaan / penulisan dll.

## Edit Descriptor untuk Data



integer

**I**w

w menunjukkan jumlah tempat termasuk tanda (- atau +)

real &  
complex

**F**w.d  
**E**w.d

w menunjukkan jumlah tempat termasuk tanda dan titik serta eksponen (untuk **E**w.d), d menunjukkan jumlah tempat di belakang titik;

untuk complex harus diberikan masing-masing untuk komponen real & imajineranya

logical

**L**w

w menunjukkan jumlah tempat;

output & input yaitu "**T**" (benar) atau "**F**" (salah), jika w cukup input juga dapat "**.TRUE.**" atau "**.FALSE.**"

character

**A**  
**A**w

w menunjukkan jumlah tempat;

tanpa w maka jumlah tempat bergantung pada data

Pada contoh berikut tempat kosong (blank) ditandai oleh “.”.

integer	I7: 1250027 ...7525	2I5: -2751..320 ...82.3018	I4,I2: ..-47.0 -37121
real & complex	F7.3: ..-0.030 ..75.125	E11.3: ..-0.430E-08 ..0.525E+12	2F7.3: ..-0.030-20.021 ..71.005..9.200
	2E10.3: -0.235E-04.0.525E+01 -0.371E+02-0.202E-00	E10.3,F4.1: -0.231E-0210.0 ..0.212E+03-0.2	
logical	L3: ..T ..F	L7: ..TRUE. ..FALSE.	(hanya untuk input)
character	CHARACTER(7) :: v v='GAGASAN' WRITE v dgn A: GAGASAN WRITE v dgn A9: ..GAGASAN WRITE v dgn A5: GAGAS	CHARACTER(7) :: v READ '..GAGASAN' dgn A: v='..GAGAS' READ 'GAGASAN' dgn A: v='GAGASAN' READ '..GAGASAN' dgn A9: v='GAGASAN' READ 'GAGASAN' dgn A9: v='GASAN.. READ '..GAGASAN' dgn A5: v='..GAG.. READ 'GAGASAN' dgn A5: v='GAGAS..'	

## Edit Descriptor untuk Kontrol

**nX**

menggeser posisi penulisan / pembacaan sejauh n ke kanan

**/**

**READ**: pindah ke awal baris berikutnya

**WRITE**: membuat baris baru

**:**

Jika jumlah data dalam daftar\_input / daftar\_output (daftar I/O) kurang dari yang seharusnya menurut format, maka tidak terjadi "error", karena pembacaan / penulisan dilakukan sesuai jumlah data yang ada dalam daftar I/O.

Pada contoh berikut tempat kosong (blank) ditandai oleh ".".

IMPLICIT NONE

INTEGER :: i,j,k,l,c

INTEGER, DIMENSION(3) :: m

REAL :: r,s,u,v

COMPLEX :: z

CHARACTER(9) :: g

LOGICAL :: b

READ(\*,'(I5,F7.3,E11.3,2I3)') j,r,s

READ(\*,'(2(I3,F5.2),/,2F4.1)') k,u,l,v,z

READ(\*,'(3I4,2X,L7)') (m(c), c=1,3),b

READ(\*,'(I3,1X,A)') i,g

•2065•-4.980••0.237E+01

625-2.21•10•0.03•3.1-0.2

•8.0-5.2

7500•750••75••.FALSE.

•20•DEWABRATA

j=2065, r=-4.980, s=0.237E+01

k=625, u=-2.21, l=10, v=0.03, z=(8.0,-5.2)

m(1)=7500, m(2)=750, m(3)=75, b=.FALSE.

i=20, g='DEWABRATA'

WRITE(\*,'(1X,4I4,A,2E10.3,A)') i,k,j,l,('z,')

WRITE(\*,'(1X,4F7.3,/,1X,3I5,2I5)') r,s,u,v, &

(m(c), c=1,3)

WRITE(\*,'(1X,L2,1X,A,/,1X,A5,3X,A11)') b,g,g,g

••20•6252065••10(•0.800E+01-0.520E+01)

•-4.980•2.370•-2.210••0.030

•7500••750•••75

•F•DEWABRATA

DEWAB•••••DEWABRATA

# Format Bebas

Jika format = \*, maka **READ** / **WRITE** membaca / menulis apa adanya sesuai ekspresi data dalam daftar I/O.

**READ**: data input dipisahkan oleh koma atau tempat kosong (blank), bilangan kompleks ditulis dalam kurung, tanda "/" menghentikan pembacaan

**WRITE**: format data output bergantung pada processor

**IMPLICIT NONE**

**INTEGER** :: j

**REAL** :: r

**COMPLEX** :: z

**CHARACTER(4)** :: g,h

**LOGICAL** :: b,d

**READ(\*,\*)** j,r,z,g,h,b,d

**WRITE(\*,\*)** g,'.',b,h,j,r,z,d

Pada contoh berikut tempat kosong (blank) ditandai oleh ".".

..2049.37.089...(-3.0.,5.1.)"BARU".AMAT.t/.FALSE.  
 2049,37.089,(-3.0,5.1),BARU,AMAT,.TRUE./,F  
 .2049,.,37.089,(-3.0,5.1),'BARU',"AMAT",.T/f

j=2049, r=37.089, z=(-3.0,5.1), g='BARU', h='AMAT',  
 b=.TRUE., d=.TRUE. (default)

BARU.TAMAT.....2049.....37.089000.....(-3.000000,5.100000).T