FORTRAN 95 A 2003

Normy a překladače Fortranu

66 neboli IV (historická), 77 (klasická), 90 (moderní), 95 (soudobá), 2003 (objektová), 2008 (ve vývoji)
volné překladače f95 s prvky 2003: Intel Fortran (ifort, ifc) pro Linux, gfortran a g95 pro Linux, MS Windows aj.

Struktura programu

program se skládá ze sekvenčně řazených programových jednotek (na pořadí nezáleží) uložených
v jednom nebo více zdrojových souborech
programové jednotky – (jeden) hlavní program PROGRAM
procedury, tj. podprogramy SUBROUTINE a funkce FUNCTION
moduly MODULE s modulovými daty a procedurami
koncové příkazy END [PROGRAM | SUBROUTINE | FUNCTION | MODULE [name]]
moduly se připojují k jiným programovým jednotkám (USE)
procedury lze vnořovat do hostitelských programových jednotek (CONTAINS)
formátování zdrojového kódu: délka řádku 132 znaků, pokračovací řádky &, oddělovač příkazů ;, komentáře !

př. MODULE mName
specifikace modulových dat, rozhraní procedur, definice operátorů aj.
CONTAINS
modulové procedury
END MODULE

PROGRAM pName nebo SUBROUTINE sName(arguments) nebo FUNCTION fName(arguments)
USE mName
specifikace lokálních dat, argumentů procedur aj.
příkazy
CONTAINS
vnořené procedury
END PROGRAM nebo END SUBROUTINE nebo END FUNCTION

Příkazy (control statements and constructs)

prázdný příkaz:		CONTINUE
přiřazení:		= ; => (target and pointer assignments; včetně polí a struktur)
podmínky a větvení:	podmínka	IF (condition) THEN ; … ; ELSEIF (condition) THEN ; … ; ELSE ; … ; END IF
	větvení podle výrazu	SELECT CASE (expression) ; CASE (list) ; … ; CASE DEFAULT ; … ; END SELECT
	větvení podle masky	WHERE (mask) ; … ; ELSEWHERE (mask) ; … ; ELSEWHERE ; … ; END WHERE
cykly:	s řídicí proměnnou	DO i=imin,imax,istride ; … ; ENDDO
	s podmínkou	DO WHILE (condition) ; … ; ENDDO
	nekonečný	DO ; … ; ENDDO
	skoky v DO cyklech	CYCLE [name] a EXIT [name]
	nesekvenční	FORALL (i=imin:imax:istride, j=…, scalar-mask) ; … ; END FORALL (od f95)
řádkové varianty:		IF (condition) command ; WHERE (mask) command ; FORALL (…) command
pojmenované konstrukce		name : DO ; … ; ENDDO name, též IF, SELECT CASE, WHERE a FORALL
alokace dynamických proměnných:		ALLOCATE (arrays | pointers); DEALLOCATE (arrays | pointers)
nulování ukazatelů:		NULLIFY (pointers)
volání podprogramu:		CALL subroutine(arguments)
(předčasný) návrat z procedury:		RETURN
ukončení a pozastavení programu:		STOP string ; PAUSE string (zastaralé)
příkazy vstupu a výstupu (I/O):		READ, PRINT, WRITE, OPEN, CLOSE, INQUIRE, BACKSPACE, REWIND, ENDFILE
vložení zdrojového kódu ze souboru:		INCLUDE 'filename'
2003:		ASSOCIATE pro synonyma, SELECT TYPE pro větvení, FLUSH a WAIT pro I/O
zastaralé příkazy:		různá GOTO, ASSIGN, aritmetický IF, DO s návěštím

Specifikace (specification statements; popisy, deklarace)

data:	podle přístupu	konstanty (literály, pojmenované konstanty) a proměnné
	podle složitosti	skaláry, pole (atribut dat, nikoliv samostatný typ) a struktury (záznamy)
	podle dostupnosti:	data globální, modulová, data hostitele, společné bloky, lokální data
připojení modulu:		USE module, ONLY: list ; USE module, local-name=>module-name
zrušení implicitní typové konvence:		IMPLICIT NONE (v každé programové jednotce n. volbou překladače)
standardní typy:		INTEGER | REAL | COMPLEX | LOGICAL | CHARACTER
odvozené (derived) typy: popis typu		TYPE name ; specifikace položek ; END TYPE
	kde	položky jsou libovolného typu, i pole (alokovatelná až od 2003), i ukazatele
podtyp (KIND) standardních typů:		1, 2, 4, i 8 (ifc/g95) pro INTEGER a LOGICAL 4, 8 pro REAL a COMPLEX, někdy 10 (g95) a 16 (ifc/g95) pro REAL 1 i 2 pro CHARACTER
délka (LEN) pro CHARACTER:		default 1, až 2**31−1 pro 32bit. překladače
specifikace:	standardních typů	type-name(kind), attributes :: list
	řetězců	CHARACTER(len), attributes :: list
	odvozeného typu	TYPE(type-name), attributes :: list
	kde	list (seznam proměnných) může obsahovat inicializační výrazy
inicializační výraz: konstantní výraz s výjimkami (mj. nelze vnitřní funkce s REAL a COMPLEX argumenty)		
	2003/ifc/g95:	i vnitřní funkce, jsou-li jejich argumenty inicializačními výrazy př. REAL,PARAMETER :: pi=ATAN(1.)*4.
atributy dat:	pojmenované konstanty	PARAMETER
	pole	DIMENSION(:,…), ALLOCATABLE
	ukazatele a cíle	POINTER | TARGET
	uchování hodnoty	SAVE
	formální parametry	INTENT(IN | OUT | INOUT), OPTIONAL
	viditelnost mimo modul	PUBLIC | PRIVATE, od 2003: PROTECTED
	procedury	EXTERNAL | INTRINSIC
2003:	atributy pro interoperabilitu s C: BIND(C) a VALUE, pro parametrizaci odvozených typů: KIND a LEN, pro objektové programování: EXTENDS (k TYPE) a PASS | NOPASS (k PROCEDURE), pro paralelní programování: VOLATILE	
	interoperabilní výčty	př. ENUM,BIND(C) ; ENUMERATOR :: a=0,b,c ; END ENUM
	import jmen do INTERFACE bloku: IMPORT	
literály:	INTEGER	př. 0, 1, 1_4, +1_I4, −1_I4
	REAL	př. 0., .0, 1._4, 1._8, 1E0, 1E0_8, −1.0E+0_DP
	COMPLEX	př. (0,1), (0.,1._4), (0._DP,1._DP)
	LOGICAL	př. .TRUE., .FALSE., .TRUE._1
	CHARACTER	př. 'abc', "ABC"
	pole	př. (/1.,2.,3./), od 2003: [1.,2.,3.]
	struktura	př. MyDerivedType(1,2.,'c',.TRUE.)

zastaralé specifikace: společné bloky COMMON, překrývání v paměti EQUIVALENCE, předefinování implicitní typové konvence – př. IMPLICIT REAL*8 (A-H,O-Z), řádkové varianty atributů – PARAMETER, DIMENSION, ALLOCATABLE ad.

Výrazy a operátory (expressions and operators)

výrazy se vyhodnocují v pořadí priorit podvýrazů, v případě rovnosti priorit zleva doprava (s výjimkou umocnění)

operátory podle priority:

(aritmetické) umocnění ** (nejvyšší priorita), multiplikativní *, / (nižší priorita), adiční +, −

(znakové) řetězení //, pozn. podřetězec př. CH(2:2), CH(:5), CH(2:), CH(:)

(relační) ==, /=, <, <=, >, >= (starší ekvivalenty .EQ., .NE., .LT., .LE., .GT., .GE.)

(logické) .NOT. (nejvyšší priorita), .AND. (nižší), .OR. (nižší), .EQV., .NEQV. (nejnižší)

automatická typová konverze operandů různého typu a podtypu (s výj. umocňování na integer):

IR a RI -> RR, IC a CI -> CC, RC a CR -> CC

na místech operandů mohou být skaláry, ale i pole nebo sekce polí, viz prvkové (elemental) výrazy níže

na místech operandů mohou být ukazatele, za které se automaticky dosazuje obsah jejich cíle, viz o ukazatelích níže

lze vytvářet definované operátory, nové pro operandy standardních i odvozených typů a standardní pro odvozené typy, viz INTERFACE OPERATOR(operator) níže

lze vytvářet definovaná přiřazení pro nestandardní kombinace typů a změnit standardní přiřazení pro odvozené typy, viz INTERFACE ASSIGNMENT(=) níže

Pole (arrays)

terminologie: počet dimenzí (rank), velikost (size), tvar (shape) = rank+size, dolní a horní mez (lbound a ubound)
podobná pole (conformable arrays) = pole téhož tvaru
maska = LOGICAL pole podobné zpracovávanému výrazu
příkaz nebo funkce zpracují pouze prvky odpovídající .TRUE. prvkům masky
v příkazu WHERE, FORALL (skalární zápis masky) a v některých vnitřních funkcích
konstruktor pole = seznam prvků téhož typu, podtypu a délky v závorkách (/…/)
součástí může být cyklický seznam (expressions,i=imin,imax,istride)
vždy vektor (1D pole), konstrukce vícerozměrných polí pomocí funkce RESHAPE(source,shape)
př. (/0,1,4,9/), (/(i**2,i=0,3,1)/), (/0.,1./), (/'ABC','DE_'/), nikoliv však (/0,1./)
2003/ifc/g95: závorky […] a specifikace typu, př. [COMPLEX :: 1,1.,(0,1)], [CHARACTER(10) :: 'A','B']

specifikace: atribut DIMENSION a meze buď u atributu DIMENSION(bounds) nebo za jménem pole name(bounds)
kde bounds obsahuje ubound, lbound:ubound nebo : pro každou dimenzi, default lbound=1
př. REAL,DIMENSION(10) :: arr1,arr2(0:10,0:10),arr3=0.

druhy polí: statická pole – specifikace obsahuje celý tvar včetně (konstantních) mezí
př. INTEGER,PARAMETER :: n1,n2
REAL :: a(n1,n2)
dynamická pole – specifikace vyjadřuje pomocí : počet dimenzí, meze až za běhu příkazem ALLOCATE
alokovatelná pole – spojitý úsek v paměti, optimalizovatelnost, snadná automatická dealokace
př. REAL,ALLOCATABLE :: a(:,:) ; ALLOCATE (a(n1,n2)) ; ... ; DEALLOCATE (a)
ukazatelová pole – obecnější rozložení v paměti (+ i –), problematická automatická dealokace
př. REAL,POINTER :: a(:,:) ; ALLOCATE (a(n1,n2)) ; ... ; DEALLOCATE (a)
automatická pole – jen v procedurách, pole samo není argumentem procedury, ale jeho meze jsou
vyjádřeny pomocí argumentů n. modulových proměnných
automaticky alokováno při vstupu procedury a dealokováno při jejím ukončení
př. SUBROUTINE s(n1) ; USE m,ONLY : n2 ; REAL :: a(n1,n2)
formální pole – jsou argumentem procedur, předává se adresa nebo hodnoty z místa volání
pole předpokládaného tvaru – : na místě mezí, pole přejímá tvar předávaného pole
pole předpokládané velikosti – * na místě poslední dimenze, pole nemá velikost, nesmí přetéci
meze předávaného pole
př. SUBROUTINE s(a,b) ; REAL :: a(:,:) ; REAL :: b(n1,*)

přístup k poli: prvek pole – a(index,index,…)
sekce pole (řez) – a(subscript-triplet | subscript-dublet | vector-subscript,…)
indexový triplet (subscript triplet) imin:imax:istride, kde istride je kladné nebo záporné
lze také :imax:istride, imin::istride, ::istride
indexový dublet (subscript dublet) imin:imax, kde krok roven implicitně 1
lze také :imax, imin:, :
vektorový index (vector-subscript) INTEGER vektor, např. konstruktor pole
dolní meze (LBOUND) sekce rovny 1, pole nulové velikosti jsou přípustná
př. i-tý řádek a(i,:), j-tý sloupec a(:,j), celá matice a(:,:), sudé prvky vektoru v(2::2)
př. iv=(/2,4,6,8/) ; print *,a((/1,3,5,7/),iv)

funkce vracející pole (array-valued functions):
tvar i meze výsledku musejí být specifikovány na vstupu do funkce
př. FUNCTION f(a) ; REAL :: a(:),f(SIZE(a)) ; f=a*a ; END FUNCTION

prvkové (elemental) funkce a výrazy: většina vnitřních funkcí a standardní i definované operátory mohou jako
argumenty/operandy zpracovat podobná pole (sekce pole) nebo pole a skaláry
výsledkem je pole podobné argumentům/operandům po operaci provedené prvek po prvku

příkazy: prvkové (elemental) přiřazení: prvkům pole (sekce pole) na levé straně se přiřadí skalár nebo
odpovídající prvek podobného pole (sekce pole) na pravé straně
WHERE (mask); array-assignments-true ; [ELSEWHERE ; array-assignments-false ;] END WHERE
pole v přiřazeních a maska musí být podobná
ELSEWHERE může mít též masku a větve mohou obsahovat vnořené příkazy WHERE
př. WHERE (a/=0.) ; a=1./a ; ELSEWHERE ; a=HUGE(0.); END WHERE
FORALL (i=imin:imax:istride, j=…, scalar-mask) ; assignments ; END FORALL (od f95)
indexované nesekvenční přiřazení do pole, včetně přiřazení ukazatelů
příkazy FORALL lze vnořovat a mohou obsahovat příkazy WHERE a volání čistých (PURE) funkcí
př. FORALL (i=imin:imax) ; a(i,i)=i ; END FORALL
FORALL (i=imin:imax, j=jmin:jmax, i+j/=1) ; a(i,j)=1./(i+j-1) ; END FORALL

alokovatelná pole: příkaz ALLOCATE (allocation-list [,STAT=stat]), kde po neúspěšné alokaci stat>0
př. ALLOCATE (a(n,0:n+1),STAT=istat) ; IF (istat>0) …
alokované alokovatelné pole alokovat nelze (2003: MOVE_ALLOC), ukazatelové pole lze
příkaz DEALLOCATE (allocated-object-list [,STAT=stat])
lokální alokovaná pole bez atributu SAVE dealokována automaticky
funkce ALLOCATED(array) vrací alokační status pole
2003/ifc/g95: přenos alokace pomocí podprogramu MOVE_ALLOC(from,to), pole from bude nealokované
př. ALLOCATE (temp(n)) ; temp(1:SIZE(a))=a ; CALL MOVE_ALLOC(temp,a)
2003/ifc/g95: alokovatelné skaláry (ne však CHARACTER) a typová alokace,
tj. zatím nelze: CHARACTER(:),ALLOCATABLE :: x ; ALLOCATE (CHARACTER(n) :: x)
2003/ifc/–: automatická (re)alokace přiřazením ("ifort -assume realloc_lhs")
př. REAL,ALLOCATABLE :: x(:) ; … ; x=PACK(x,x/=0)
2003/ifc/–: chybová zpráva ERRMSG=errmsg v ALLOCATE/DEALLOCATE
př. ALLOCATE (x,STAT=istat,ERRMSG=errmsg)
TR 15581/1998 (Technical Report ISO/IEC): běžně implementované rozšíření f95, součást 2003/ifc/g95
vlastnosti už k dispozici pomocí ukazatelových polí, pomocí alokovatelných polí realizovatelné efektivněji
alokovatelné formální argumenty – odpovídající skutečné argumenty alokovatelná pole téhož typu a ranku
formální argument přejímá i vrací alokační status
má-li INTENT(IN), nemění alokační status, má-li INTENT(OUT), dealokuje se na vstupu
alokovatelné výsledky funkce – alokační status na vstupu „not allocated", pro výstup musí být „allocated"
a v poli musí být přiřazeny hodnoty, výsledek automaticky dealokován
př. FUNCTION f(a) ; REAL,ALLOCATABLE :: a(:),f(:) ; ALLOCATE (f(SIZE(a))) ; f=a*a ; END
alokovatelné položky struktur – počáteční alokační status položky „not allocated"
alokace pomocí ALLOCATE, přiřazením nebo předáním skutečného argumentu
konstruktor struktury na příslušné pozici může obsahovat alokovatelné pole nebo NULL()
při dealokaci struktury alokované položky dealokovány automaticky
př. TYPE t ; REAL,ALLOCATABLE,DIMENSION(:) :: c ; END TYPE ;
TYPE(t),ALLOCATABLE :: a(:) ;
ALLOCATE (a(n)) ; DO i=1,n ; ALLOCATE (a(i)%c(n)) ; ENDDO ;
PRINT *,a(1)%c(1:n) ; ! PRINT *,a(1:n)%c(1) nelze, sekce příliš nepravidelně uložena
DEALLOCATE (a) ! dealokuje i alokované položky, na rozdíl od POINTER položek
ukládání polí: do zásobníku (stack) obvykle lokální proměnné (automatická pole) a pomocná pole
na hromadu (heap) alokovatelná pole
(default) limity (překladačů, shellů) pro zásobník obvykle nevelké (chyba Segmentation fault)
bash: ulimit -s unlimited
tcsh: limit stacksize unlimited
ifc: ifort -heap-arrays [n] pro vytváření polí min. velikosti n KB na hromadě

Vnitřní procedury (intrinsic procedures)
funkce a podprogramy definované standardem
dotazovací funkce (array inquiry functions) – dotaz po vlastnostech, nikoliv hodnotách pole
SHAPE(source)
SIZE(array[,dim])
LBOUND(array[,dim]), UBOUND(array[,dim])
ALLOCATED(array)
prvkové funkce – argumentem může být skalár i pole, výsledkem je skalár nebo podobné pole vyhodnocené
prvek po prvku
ABS, AIMAG, AINT, ANINT, CEILING, CMPLX, FLOOR, INT, NINT, REAL
CONJG, DIM, MAX, MIN, MOD, MODULO, SIGN
ACOS, ASIN, ATAN, ATAN2, COS, COSH, EXP, LOG, LOG10, SIN, SINH, SQRT, TAN, TANH
ACHAR, CHAR, IACHAR, ICHAR
LGE, LGT, LLE, LLT
ADJUSTL, ADJUSTR, INDEX, LEN_TRIM, SCAN, VERIFY
LOGICAL
EXPONENT, FRACTION, NEAREST, RRSPACING, SCALE, SET_EXPONENT, SPACING
BTEST, IAND, IBCLR, IBITS, IBSET, IEOR, IOR, ISHFT, ISHFTC, NOT, subr. MVBITS
subr. RANDOM_NUMBER
multiplikační funkce – skalární součin a maticové násobení

DOT_PRODUCT, MATMUL

transformační funkce – argumentem pole, volitelně pořadí dimenze a maska, výsledkem různé souhrny
ALL(mask), ANY(mask), COUNT(mask), MAXVAL(array), MINVAL(array), PRODUCT(array), SUM(array)
MAXLOC(array), MINLOC(array)
volitelné argumenty: dim, mask

manipulační funkce
MERGE(tsource,fsource,mask)
PACK(array,mask[,vector]), UNPACK(vector,mask,field)
RESHAPE(source,shape[,pad][,order])
SPREAD(source,dim,ncopies)
CSHIFT(array,shift[,dim]), EOSHIFT(array,shift[,boundary][,dim])
TRANSPOSE(matrix)

Vnější procedury (external procedures)
funkce a podprogramy programované člověkem
platnost a sdílení dat: globální data – jména programu a vnějších (nevnořených) procedur, jména společných bloků
modulová data (use association) – PUBLIC data modulu dostupná jeho připojením USE
data hostitele (host association) – data hostitele dostupná vnořeným procedurám
společné bloky (common blocks) – data ve společných blocích dostupná specifikací COMMON
argumenty procedur – přístup k datům volající jednotky pomocí formálních (dummy) argumentů
– přednostní předávání odkazem, nelze-li, hodnotou (2003: atribut VALUE)
– předávání poziční nebo pomocí formálních jmen (keyword argument)
– volitelné argumenty (atribut OPTIONAL)
lokální data procedury – nejsou formálními argumenty, nemají atribut SAVE
proměnné FORALL a cyklického seznamu v konstruktoru pole – nesplývají s lokálními daty procedury

rekurzivní procedury: procedury volající samy sebe (přímo nebo nepřímo, tj. přes jinou proceduru)
př. RECURSIVE SUBROUTINE name(arguments)
IF ... CALL name(other-arguments) ; ...
END SUBROUTINE
RECURSIVE FUNCTION name(arguments) RESULT (result)
IF ... result=name(other-arguments) ; ...
END FUNCTION

čisté procedury: procedury bez vedlejších efektů (od f95)
účel – volání ve FORALL a definovaných přiřazeních
požadavky – nemění formální argumenty (podprogram může), nemění data modulu ani hostitele,
lokální proměnné nemají atribut SAVE, nepracují se soubory, neobsahují STOP (a PAUSE),
formální argumenty funkce musí mít INTENT(IN), volané procedury musí být PURE aj.
všechny vnitřní funkce jsou čisté
př. PURE FUNCTION distance(x,y)
REAL,INTENT(IN) :: x,y
distance= …
END FUNCTION

prvkové procedury: procedury specifikované pro skalární argumenty, použitelné i pro pole (od f95)
účel – automatické přetížení, snadná paralelizace
požadavky – tytéž jako pro čisté procedury, navíc formální argumenty jsou skalární bez POINTER,
ve volající jednotce INTERFACE blok s popisem rozhraní
prvkové procedury jsou (automaticky) čisté, nemohou být rekurzivní a skutečnými argumenty procedur
př. ELEMENTAL SUBROUTINE swap(x,y)
REAL,INTENT(INOUT) :: x,y ; REAL t
t=a; a=b; b=t
END SUBROUTINE
CALL swap(x,y) ; CALL swap(xarr1d,yarr1d); CALL swap(xarr3D,yarr3D)

explicitní rozhraní (explicit interface) volané procedury má volající jednotka k dispozici, když
– volající jednotka obsahuje INTERFACE blok volané procedury
– volaná procedura je v ní vnořená
– volaná procedura je v připojeném modulu
vhodné vždy (pro kontrolu shody typu a počtu skutečných a formálních argumentů)
nutné při předávání argumentů pomocí formálních jmen, při volitelných argumentech aj.

INTERFACE blok: pro popis rozhraní (argumentů a výstupních hodnot funkcí) vnějších procedur
```
    př. INTERFACE
        FUNCTION f(a,b) ; REAL,OPTIONAL :: a,b ; REAL f; END FUNCTION
        END INTERFACE
        PRINT *,f(),f(1.),f(b=2.),f(1.,2.),f(b=2.,a=1.)
```
pro přetěžování procedur (overloading, generic interfaces) – volání specifických procedur (rozlišených různým typem, podtypem, rankem nebo počtem argumentů) generickým jménem
```
    INTERFACE generic-name
    MODULE PROCEDURE specific-names          (v případě procedur vnořených v témže modulu)
    nebo popis rozhraní procedur se specific-names (nejde-li o modulové procedury)
    END INTERFACE
    př. INTERFACE f
        FUNCTION fr(x) ;    REAL(4) x,fr ;   END FUNCTION
        FUNCTION fd(x) ;    REAL(8) x,fd ;   END FUNCTION
        FUNCTION f2(x,y);   REAL(4) x,y,f2; END FUNCTION
        END INTERFACE
        PRINT *,f(1.),f(1._8),f(1.,2.)
```
pro definované operátory a rozšíření standardních operátorů pro struktury
```
    INTERFACE OPERATOR(operator)
    MODULE PROCEDURE fname          nebo popis rozhraní funkce fname
    END INTERFACE
    kde operator je standardní (+–*/ ** .NOT. .AND. ...) nebo vlastní (.NAME.)
    a fname je vnější funkce s jedním/dvěma argumenty INTENT(IN) pro unární/binární operátor
    nelze předefinovat standardní operace pro operandy standardních typů
    unární definované operátory mají nejvyšší prioritu (vyšší než **), binární nejnižší (nižší než .EQV.)
```
pro definovaná přiřazení a změnu standardního přiřazení pro struktury
```
    INTERFACE ASSIGNMENT(=)
    MODULE PROCEDURE sname          nebo popis rozhraní podprogramu sname
    END INTERFACE
    kde sname je podprogram o dvou argumentech, prvním INTENT(OUT|INOUT), druhým INTENT(IN)
    nelze předefinovat přiřazení do proměnné standardního typu
```
zastaralé možnosti: datové podprogramy BLOCK DATA, jednopříkazové funkce, vstupní body procedur ENTRY, alternativní návratové body

Ukazatele (pointers)

proměnná s atributem POINTER je deskriptorem cíle, obsahuje popis (typ, podtyp, tvar) a adresu cíle

ukazatelové přiřazení (pointer assignment) => pro přiřazení cíle ukazateli

cíl: pojmenovaný – proměnná (týž typ, podtyp, rank) explicitně popsaná atributem TARGET nebo jiný ukazatel
```
    př. REAL,POINTER :: p,r ; REAL,TARGET :: x=1. ; p=>x ; r=>p          ! p i r ukazují na x
```
nepojmenovaný – paměťové místo alokované příkazem ALLOCATE (pointer)
```
    př. REAL,POINTER :: p ; ALLOCATE (p) ; p=1. ; DEALLOCATE (p)
```
automatická dereference: ukazatel ve výrazech i na levé straně přiřazení „=" je automaticky nahrazován cílem

funkce ASSOCIATED(pointer[,target]) pro zjištění stavu ukazatele: disassociated („null") nebo associated; počáteční stav ukazatele je nedefinován, je proto vhodné ukazatele co nejdříve inicializovat

příkaz NULLIFY (list) (od f90) a funkce NULL() (od f95) pro nulování ukazatelů
```
    př. REAL,POINTER :: p=>NULL(),r=>NULL() ; NULLIFY (p,r)
```
ukazatelové pole, ukazatel na pole (array pointer, pointer to array)
```
    př. REAL,POINTER :: pa(:) ; REAL,TARGET :: x(0:10) ; pa=>x ; pa=>x(:) ; pa=>x(0:10:2)
```
pole ukazatelů (array of pointers): pomocí odvozeného typu s ukazatelem jako položkou
```
    př. TYPE tp ; REAL,POINTER :: p ; END TYPE ; TYPE(tp) :: ap(10) ; ap(1)%p=>NULL()
```
ukazovat lze na sekce pole tvořené tripletem, nikoliv vektorovým indexem

problematické situace: neinicializovaný ukazatel (wild pointer)
```
    př. REAL,POINTER :: p ; IF (ASSOCIATED(p)) ...                       ! nedefinovaný stav
```
únik paměti (memory leak) – ukazatel na alokovaný cíl přesměrován bez dealokace cíle
```
    př. REAL,POINTER :: p ; ALLOCATE (p) ; p=>NULL()                     ! g95: „Remaining memory" warning
```
visící ukazatel (dangling pointer) – ukazatel míří na neexistující cíl
```
    př. REAL,POINTER :: p,r ; ALLOCATE (p) ; r=>p ; DEALLOCATE (p)       ! ASSOCIATED(r) je .TRUE., cíl není
```
dvojí dealokace (double free)
```
    př. REAL,POINTER :: p,r ; ALLOCATE (p) ; r=>p ; DEALLOCATE (p,r)     ! runtime error
```

Struktury (structures; proměnné odvozeného typu, záznamy)
specifikace typu: TYPE [,attributes ::] name ! attributes: PUBLIC | PRIVATE
[PRIVATE | SEQUENCE]
popisy položek
END TYPE [name]
položky lib. standardního i odvozeného typu, jejich jména mohou být totožná se jmény mimo typ
povolené atributy položek: DIMENSION, POINTER a (od 2003) ALLOCATABLE, PRIVATE a PUBLIC
položky mohou být inicializovány (od 2003)
položky s atributem POINTER mohou ukazovat na právě definovaný typ
atributy typu PUBLIC, PRIVATE a specifikaci položek PRIVATE lze použít jen v modulech
default atribut typu i položek je PUBLIC, tj. (vně modulu) přístupný typ i položky nebo přístupný typ
s nepřístupnými položkami nebo typ i položky nepřístupné
specifikace SEQUENCE vynutí sekvenční řazení položek v paměti
př. TYPE tir ; INTEGER :: i ; REAL :: x(3) ; END TYPE
TYPE tll ; INTEGER i ; TYPE(tll),POINTER :: p ; END TYPE ! spojový seznam (linked list)
TYPE tap ; INTEGER i ; REAL,POINTER :: x(:) ; END TYPE ! pole ve struktuře
TYPE taa ; INTEGER :: i=0 ; REAL,ALLOCATABLE :: x(:) ; END TYPE ! od 2003/ifc/g95
specifikace struktury: TYPE(name), attributes :: list
př. TYPE(tir) :: s ; TYPE(tll),POINTER :: node0
konstruktor struktury: type-name(list-of-components)
2003/ifc/g95: se jmény položek a nepovinným uváděním položek s default inicializací
př. s=tir(0,(/1.,2.,3./)) ; aa=taa(x=[1.,2.,3.])
2003/–/–: konstruktor lze přetížit
přístup ke struktuře
položka: structure%component
vstup a výstup: lze pro struktury (nemají-li ukazatelové položky)
př. print *,s%i,s%x ; print *,s ; print *,tir(0,(/1.,2.,3./))
funkce vracející strukturu: vyžaduje povinné explicitní rozhraní ve volající jednotce
př. FUNCTION f(n) ; TYPE tir ; ... ; TYPE(tir) f ; INTEGER n ; f=tir(n,(/1.,2.,3./)) ; END FUNCTION
alokovatelné struktury (2003/ifc/g95)
př. TYPE(tir),ALLOCATABLE :: s,sa(:)
přiřazení standardní přiřazení „=" mezi strukturami je po položkách, pro ukazatelové položky platí „=>"
př. TYPE(tll) :: p0 ; p0=tll(0,null()) ! totéž jako p0%i=0 ; p0%p=>null()
definované přiřazení: pro změnu standardního přiřazení mezi strukturami téhož typu nebo
pro definici přiřazení mezi strukturou a výrazem jiného typu
popis INTERFACE ASSIGNMENT(=) ; … ; END INTERFACE, viz výše
definované operátory: pro definici standardních nebo vlastních operací mezi strukturami
popis INTERFACE OPERATOR(operator) ; … ; END INTERFACE, viz výše
parametrizované odvozené typy: parametrizace pomocí INTEGER proměnných s atributy KIND, LEN (2003/–/–)
př. TYPE :: vector(kind,n)
INTEGER,KIND :: kind=4 ! parametry mohou mít default hodnoty
INTEGER,LEN :: n=10
REAL(kind) :: a(n)
END TYPE
TYPE(vector) :: v1 ; TYPE(vector(8,20)) :: v2 ! pro REAL(4) v1%a(1:10) a REAL(8) v2%a(1:20)
vstup a výstup struktur pomocí definovaných procedur (2003/–/–)
užitím editačního descriptoru DT(seznam-šířek-sloupců) a podprogramu uvedeného v bloku
INTERFACE WRITE(FORMATTED) ap.
další příklady:
pole struktur TYPE t ; REAL x ; END TYPE ; TYPE(t) :: s(3) ; s%x=0. ! lze sekce s(:)%x
struktura s poli TYPE t ; REAL x(3) ; END TYPE ; TYPE(t) :: s ; s%x=0. ! lze sekce s%x(:)
pole struktur s poli TYPE t ; REAL x(3) ; END TYPE ; TYPE(t) :: s(3) ! nelze zdvojená sekce s(:)%x(:) ani s%x

Na cestě k objektům (2003/–/–)

objektová terminologie
- třída (class) = odvozený typ s vlastnostmi (fields, variables) a metodami (methods, type-bound procedures)
- zapouzdření (encapsulation) = přístup k vlastnostem pouze pomocí metod
- dědičnost (inheritance) = přejímání vlastností a metod rodičovské třídy (superclass) při definici třídy (subclass)
- objekt, instance objektu = proměnná třídy
- zprávy (messages) = volání metody s objektem jako argumentem formou object%method(other-arguments)
- konstruktory a destruktory (constructors, finalizers) = metody pro inicializaci a likvidaci objektu

řízení přístupu k modulovým datům (access control) – zapouzdření
- atributy modulových proměnných PUBLIC (f90) | PRIVATE (f90) | PROTECTED (2003/ifc/g95)
- PROTECTED proměnná modifikovatelná pouze v modulu, vně dostupná jen pro čtení

rozšíření typu (type extension, ifc: ano) – dědičnost
- specifikace odvozeného typu děděním od rodičovského typu (atribut specifikace typu EXTENDS)
- zděděné položky mohou být souhrnně adresovány jako položka rodičovského typu
- př. TYPE t2d ; REAL x,y ; END TYPE
  TYPE,EXTENDS(t2d) :: t3d ; REAL z ; END TYPE
  TYPE(t3d) :: p=*t3d(1.,2.,3.)*,p2=t3d(t2d(1,2),3)
  PRINT *,p ; PRINT *,p%x,p%y,p%z ; PRINT *,p%t2d,p%z

metoda (type-bound procedure)
- procedura vázaná k odvozenému typu (TYPE … ; CONTAINS ; PROCEDURE … ; END TYPE)
- volání: var%procedure([arg1,]arg2,…)
- má-li procedura default atribut PASS, dosadí se var za arg1 (opak: NOPASS)
- př. MODULE mPoint ;
  TYPE t2d ; REAL x,y ; CONTAINS ; PROCEDURE,PASS :: add ; END TYPE
  CONTAINS
  TYPE(t2d) FUNCTION add(this,another_point) ; … ; END FUNCTION
  END MODULE
  … ; TYPE(t2d) a,b,c ; … ; c=a%add(b)

polymorfní proměnná
- struktura, která je kompatibilní se specifikovaným typem i jeho podtřídami (specifikace CLASS místo TYPE)
- struktura tak má jednak specifikovaný (declared) typ, jednak dynamický (dynamic) typ
- podle dynamického typu struktury lze větvit příkazem SELECT TYPE, lze jej testovat funkcemi EXTENDS_TYPE_OF, SAME_TYPE_AS
- struktura musí mít atribut ALLOCATABLE, POINTER nebo být formálním argumentem procedury
- př. CLASS(t2d),POINTER :: p ; TYPE(t2d) :: p2 ; TYPE(t3d) :: p3 ! viz specifikace typu výše
  p=>p2 ; p=>p3

… (abstract interfaces, finalizers)

Vstup a výstup (input and output; V/V, I/O)
příkazy pro standardní V/V: READ fmt ; READ (*,fmt) ; PRINT fmt ; WRITE (*,fmt)
formátování V/V: atribut FMT= * nebo návěští popisu FORMAT nebo řetězec s formátovou specifikací
odřádkování: atribut ADVANCE='YES' | 'NO' (default 'YES')
ošetření V/V chyb: atribut IOSTAT=ierr (ierr=0 bez chyby, <0 end-of-file, >0 jiná chyba)
soubory: OPEN (id,FILE='jmeno') ; READ (id,form) ; WRITE (id,form), CLOSE
formátové a bezformátové: atribut FORM='FORMATTED' | 'UNFORMATTED'
sekvenční a s přímým přístupem: atribut ACCESS='SEQUENTIAL' | 'DIRECT'
jmenný seznam (namelist)
formátová specifikace (edit descriptors):
datové deskriptory pro INTEGER Iw, Iw.m, a B|O|Z ekvivalenty pro 2-, 8-, 16-kovou soustavu
pro REAL a COMPLEX Fw.d, Ew.d, Ew.dEe, ENw.d, ESw.d
pro LOGICAL Lw
pro CHARACTER A, Aw
obecně Gw.d, Gw.dEe
jsou opakovatelné př. nIw
Iw, Fw připouštějí nulové w pro výstup v minimální potřebné šířce
řídicí deskriptory nX pro mezery, / nebo n/ pro odřádkování, nestd. $ pro ADVANCE='NO', aj.
vnořování n(…), př. (2147483647(2147483647E10.3))
2003/ifc/g95: stream I/O: OPEN a ACCESS='STREAM', ASYNCHRONOUS, OPEN a CONVERT (g95)
příkazy FLUSH a WAIT
specifikátory ASYNCHRONOUS, POS, BLANK, DELIM aj.
editační deskriptory DC, DP, RD aj.

Další rozšíření 2003
přístup k argumentům příkazového řádku a proměnným prostředí (ifc/g95)

integer funkce	COMMAND_ARGUMENT_COUNT()	vrací počet argumentů
podprogram	GET_COMMAND_ARGUMENT(number [,value] [,length] [,status])	vrací argumenty
podprogram	GET_COMMAND([command] [,length] [,status])	vrací celý řádek
podprogram	GET_ENVIRONMENT_VARIABLE(name [,value] [,length] [,status] [,trim_name])	vrací proměnné

asociační blok (2003/ifc/–) pro zavedení synonym (zkratek)
ASSOCIATE (association-list) ; block ; END ASSOCIATE
kde association-list tvoří (jakoby) ukazatelová přiřazení (avšak asociační jména se nespecifikají, cíle nemusejí mít
atribut TARGET, cíle mohou být obecné výrazy)
př. TYPE t ; REAL :: long_component_name ; END TYPE
TYPE(t) long_structure_name
ASSOCIATE (a=>long_structure_name%long_component_name,b=>1.) ; a=b ; END ASSOCIATE

ifc atributy BIND(C) (g95, včetně modulu ISO_C_BINDING), VALUE (g95), PASS a NOPASS, VOLATILE
ifc statements FLUSH, WAIT, IMPORT (g95)
ifc tři IEEE moduly, IEEE Inf/NaN na výstupu (g95: half IEEE support)
ifc USE a přejmenování operátorů, INTRINSIC a NON_INTRINSIC (g95)
ifc procedurové ukazatele (g95) a specifikace PROCEDURE, ABSTRACT INTERFACE
g95 přípony souboru .f, .for pro pevný formát, .f90, .f95, .f03 pro volný formát zdrojového textu
g95 corefile resume
g95 funkce BESJ0/1/N, BESY0/1/N, GAMMA, DLGAMMA, ERF/ERFC, DTIME, ISNAN, SIGNAL, SYSTEM
g95 volby: -o output file, -c compile only, -O[n] optimization (0-3, default -O0, -O je -O1), -Wall warnings
-march=prescott (včetně SSE3), -mcmodel=small/medium (2 GB limit pro data ano/ne)
-fimplicit-none, -Wunset-vars, -std=f95/f2003, -fbounds-check, -fzero
gfortran ...

www
http://en.wikipedia.org/wiki/Fortran
http://en.wikipedia.org/wiki/Fortran_language_features
http://www.root.cz/serialy/fortran-pro-vsechny
http://www.pbm.com/~lindahl/real.programmers.html

30. 6. 2009