F. Zsongor GOBESZ

C. Mihai MOCAN

Programare în Fortran G95 pentru începători

(Teorie pentru lucrări)

U.T.PRESS
Cluj-Napoca, 2025
ISBN 978-606-737-803-0

F. Zsongor GOBESZ C. Mihai MOCAN

Programare în Fortran G95 pentru începători

Teorie pentru lucrări

U.T.PRESS
Cluj-Napoca, 2025
ISBN 978-606-737-803-0

Editura U.T.PRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.: 0264-401.999
e-mail: utpress@biblio.utcluj.ro
www.utcluj.ro/editura

Recenzia: Conf.dr.ing. Ciprian Pavel Oprișa
Conf.dr.ing. Adrian Colești

Pregătire format electronic on-line: Gabriela Groza

ISBN 978-606-737-803-0

CUPRINS

Preambul

Realizarea unui program pe calculator are sens doar în situația în care volumul de calcule ar depăși posibilitățile manuale. Asemenea situații se pot ivi în cazul necesității rezolvării multor probleme similare, a unor calcule complexe și laborioase, sau în cazul unor mulțimi mari de date care trebuie prelucrate. Pentru crearea unui program de calcul sunt necesare câteva cunoștințe specifice. Eficiența și performanțele programului nu vor depinde doar de platforma pe care va rula, ci și de cunoștințele și de experiența celor care colaborează pentru realizarea lui. În cele ce urmează ne vom referi la realizarea unor aplicații simple, utilizând un limbaj de programare de nivel înalt (apropiat de limbajul natural), și anume Fortran. Etapele de programare sunt în general sintetizate prin 3 faze: concepţia (nivelul logic de rezolvare a problemei cu elaborarea sau alegerea algoritmului corespunzător, în funcție de formularea problemei), codificarea (transcrierea algoritmului într-un limbaj de programare, accesibil calculatorului, pentru realizarea programului), testarea şi implementarea (verificarea corectitudinii cu date de test şi punerea la punct a programului). Aceste faze pot fi parcurse prin următorii pași:

- recunoaşterea şi definirea problemei (pentru a cunoaște datele inițiale);
- alegerea şi descrierea metodei propuse (pentru modul în care se va ajunge la rezultat);
- transpunerea descrierii metodei într-un limbaj de programare (crearea fișierului sursă);
- realizarea programului prin compilare (traducerea în cod maşină a fișierului sursă, generând imaginea obiect) și editarea legăturilor (completarea imaginii obiect cu părți din biblioteca limbajului, generând fișierul executabil);
- rularea şi testarea programului creat.

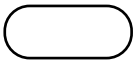
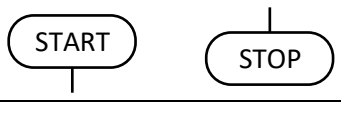
Conținutul acestui manual a fost alcătuit pentru a ghida studenții din primul an la Inginerie Civilă, la disciplina de ”Programarea calculatoarelor și limbaje de programare”, însă poate fi util și începătorilor care doresc să se inițieze în limbajul Fortran. În prima parte sunt ilustrate câteva concepte privind modul de utilizare a schemelor logice structurate pentru a descrie anumite metode, urmate de o scurtă prezentare a limbajului Fortran și menționarea câtorva medii de dezvoltare cu acces liber. Informațiile referitoare la redactarea fișierelor sursă este urmată de prezentarea mai detaliată a aspectelor fundamentale din sintaxa limbajului Fortran 95 (conform compilatorului G95) pentru scrierea unor programe simple. La final sunt câteva exerciții exemplificatoare (cu transcrierea schemelor logice și fișiere sursă).

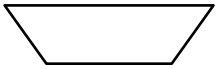
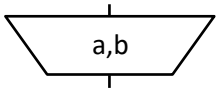
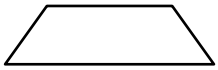
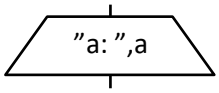
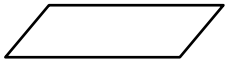
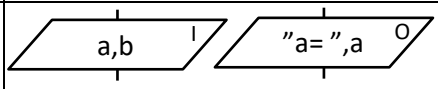
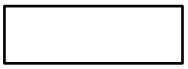
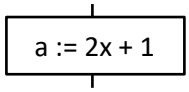
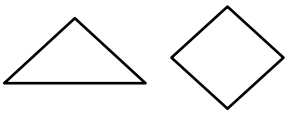
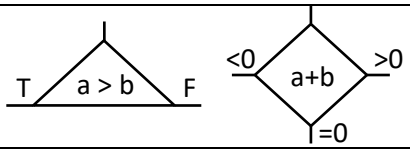
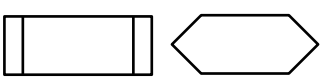
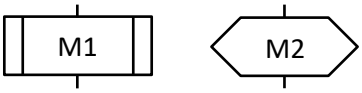
Scheme logice structurate

Schema logică este un instrument grafic prin care se pot reprezenta paşii dintr-un algoritm, sub formă de blocuri (simboluri) legate prin linii. Pentru a folosi acest instrument într-o manieră structurată, trebuie cunoscute câteva principii, cum ar fi:

- Schemele se alcătuiesc şi se citesc de sus în jos (excepţiile se marchează cu săgeți).
- Blocurile pot avea un singur punct de intrare (exceptând cel de pornire, care are doar o ieșire), iar numărul punctelor de ieșire depinde de tipul blocului: modulele și cele care reprezintă operații de intrare / ieșire sau de atribuire au doar o ieșire, cele decizionale în funcție de tipul expresiei (cele logice au 2, pentru adevărat sau fals; cele aritmetice au 3, pentru negativ, nul sau pozitiv), iar blocul final nu are punct de ieșire.
- În alcătuirea schemelor logice structurate se folosesc pe cât posibil părți asimilabile cu module (care au un singur punct de intrare și un singur punct de ieșire) înlănțuite secvențial.

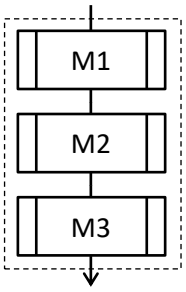
Un modul poate conține orice, cu condiția ca să aibă doar o intrare și o ieșire. Conținutul unui modul trebuie detaliat separat, atunci când se impune. Simbolurile folosite la alcătuirea schemelor logice sunt prezentate în tabelul următor:

Utilizare	Simbol	Variante (exemple)
Bloc de pornire (se marchează cu START) sau de final (se notează cu STOP)		

Bloc de intrare (se marchează cu elementele care se citesc)		
Bloc de ieșire (se marchează cu elementele care se scriu)		
Alternativă la bloc de intrare / ieșire (se notează dacă este intrare sau ieșire)		
Bloc de atribuire (conține o singură expresie, a cărei valoare se atribuie variabilei din stânga)		
Blocuri pentru decizie (la ieșire se ramifică în funcție de tipul expresiei evaluate, se marchează și condițiile)		
Bloc de procedură sau modul (se marchează cu numele modulului)		
Conector intern (pentru întrerupere sau continuare în cadrul aceleiaşi pagini, cu marcaj corespunzător)		
Conector extern (pentru întrerupere sau continuare între pagini diferite, cu marcaj corespunzător)		

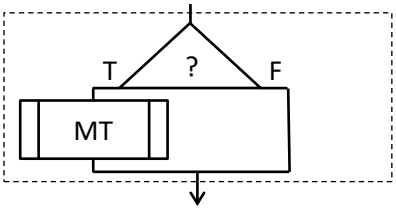
Principiile programării structurate au fost publicate de către Edsger W. Dijkstra[1], exemplificând următoarele 3 niveluri în ordinea complexității:

1. Înlănțuirea secvențială (ieșirea dintr-un modul va fi intrarea în următorul modul):



2. Structuri decizionale (la parcurgere se trece doar pe o singură ramură):

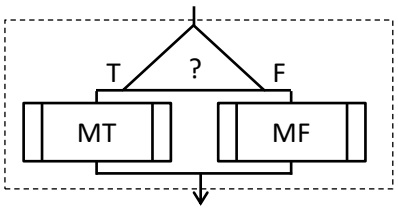
   2.1. Decizie logică simplă cu ramură vidă, ramura cazului fals fiind goală.



”if ? do MT”

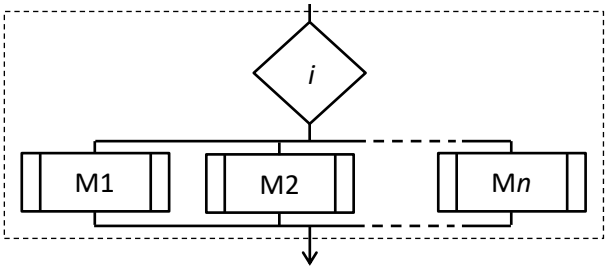
[1] E. W. Dijkstra: “Notes on Structured Programming” (Report) 70-WSK-03, Technical University of Eindhoven, The Netherlands, 1970. via E.W. Dijkstra Archive. Center for American History, University of Texas at Austin, USA. https://www.cs.utexas.edu/~EWD/ewd02xx/EWD249.PDF

2.2. Decizie logică uzuală.



”if ? then MT else MF”

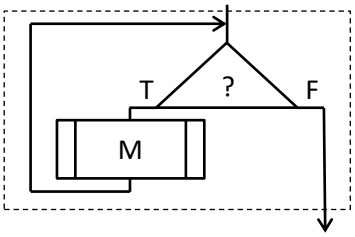
2.3. Decizie generalizată sau alegere.



”case *i* of (M1; M2; …; M*n*)”

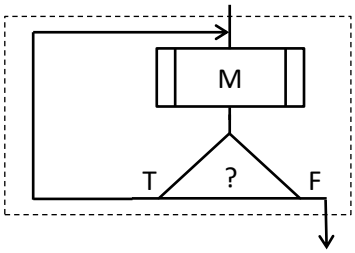
3. Structuri repetitive (cicluri):

3.1. Cilclu precondiționat.



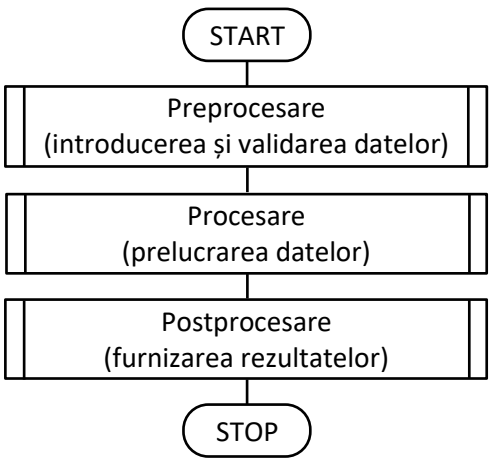
”while ? do M”

3.2. Ciclu postcondiționat.

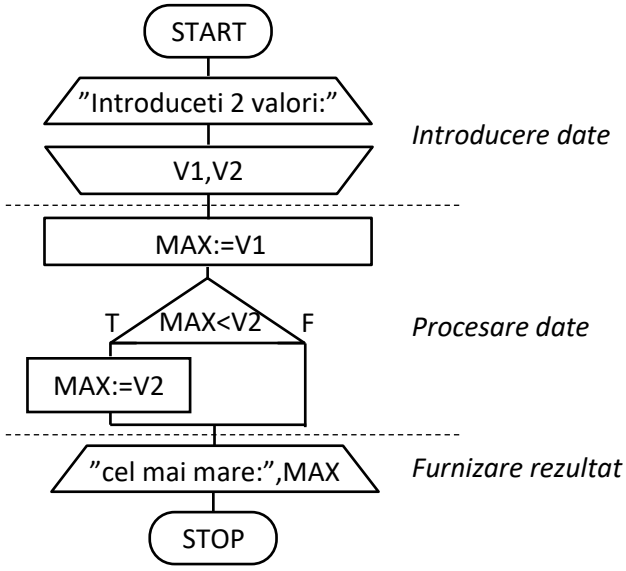


”repeat M until ?”

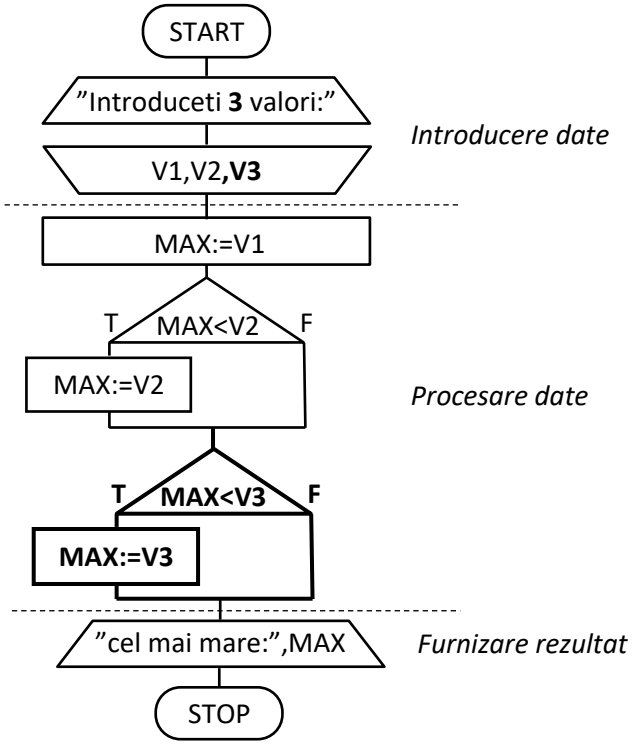
În majoritatea limbajelor de programare există specificații (instrucțiuni) corespunzătoare acestor structuri elementare. Pentru a ușura înțelegerea (verificarea) conținutului fișierelor sursă și pentru a reduce timpul de rulare a instrucțiunilor, se recomandă utilizarea structurilor pornind de la cele simple către cele mai complicate. Aplicând principiul structurării, algoritmul unui program poate fi descris printr-o schemă logică generică alcătuită din 3 module:



Astfel, divizarea părților și verificarea aplicațiilor devine mai simplă în cadrul echipelor de programatori, iar în cazul unor actualizări sau modificări ale fișierelor sursă, abordarea separată a modulelor generează sarcini mai simple. Iată un exemplu pentru afișarea valorii maxime din conținutul a două variabile:



După cum se poate remarca, în cazul în care se dorește modificarea acestui algoritm pentru a compara mai multe valori, partea de introducere a datelor se va modifica puțin, partea de procesare se va completa prin replicarea structurii decizionale cu ramură vidă, iar partea de furnizare a rezultatelor va rămâne nemodificată. Iată o variantă pentru 3 valori (modificările fiind marcate mai gros):



Despre Fortran, pe scurt

Prima versiune a limbajului a fost creat de către o echipă de la IBM sub conducerea lui John Backus, fiind lansată în anul 1957 sub denumirea ”IBM Mathematical Formula Translating System” (pe scurt: FORTRAN, din combinarea cuvintelor FORmula TRANslation), acesta fiind primul limbaj de programare de nivel înalt (apropiat de limbajul natural). În anul 1958 IBM a publicat o verisune revizuită, numită FORTRAN II, care oferea suport pentru programare procedurală, introducând specificații pentru subprograme și funcții. Din cauza popularității, IBM a decis să elimine caracteristicile care limitau utilizarea limbajului la sistemele IBM și, în anul 1964, a lansat o variantă numită FORTRAN IV, care putea rula pe orice calculator. Varianta FORTRAN 66 a apărut în anul 1966, ca urmare a standardizării realizate de către Asociația Americană de Standarde (American Standards Association, precursoarea ANSI), fiind primul limbaj de programare definit prin standard. Comitetul ”ANSI FORTRAN” (cunoscut ca ”X3J3”) a început să dezvolte o variantă nouă în 1969 și, ca rezultat, a apărut FORTRAN 77, cea mai utilizată variantă a limbajului.
Următoarea versiune se aștepta să fie lansată în anii ’80 (FORTRAN 8X), dar a apărut doar în 1991, introducând forma liberă și a devenit cunoscut ca Fortran 90, deschizând calea pentru HPF (High Performance Fortran). În 1997 a fost publicat standardul pentru Fortran 95, prima variantă orientată pe obiecte.
În comparație cu C++ (limbaj orientat pe obiecte care suportă polimorfism și moșteniri), Fortran a introdus câteva caracteristici similare (prin module și tipuri derivate), însă fără moșteniri automate. Pe de altă parte, Fortran este mai ușor de învățat și de utilizat pentru calcule științifice decât C++, având suport nativ pentru valori complexe, tablouri multidimensionale etc., care lipsesc din C++. Fortran 2003 reprezintă o cotitură semnificativă în privința caracteristicilor orientate pe obiecte, asigurând și interoperabilitate cu C/C++, iar în 2010 a fost lansat Fortran 2008 cu noi facilități (sub-module, co-tablouri, atributul contiguu etc.) și având implementată procesarea paralelă cu memorie distribuită. După Fortran 2018, care a fost o revizuire a versiunii anterioare prin facilități adiționale pentru procesare paralelă, Fortran 2023 este cea mai recentă versiune standardizată, cu și mai multe facilități.
Iată un fragment citat (tradus) din secțiunea ”FAQ” (întrebări frecvente) de pe pagina https://fortran-lang.org/, pentru cei interesați de utilitatea limbajului: ”*Pentru ce este utilizat Fortran? Fortran este utilizat mai ales în domeniile de pionierat în calcul–știință și inginerie. Acestea includ prognoza vremii și a oceanelor, mecanica fluidelor, matematică aplicată, statistica și finanțele. Fortran este limbajul dominant în calculul de înaltă performanță (HPC) și este utilizat pentru a etalona cele mai rapide supercomputere din lume*.”

Crearea programelor

Pentru crearea unui program Fortran, aveți nevoie de un editor de text (preferabil ASCII) și de un pachet corespunzător cu compilator și editor de legături. Compilatorul G95 conține câteva facilități din Fortran 2003 alături de Fortran 95 și se poate instala pe Windows cu MinGW (lansând fișierul g95-MinGW.exe), chiar dacă dezvoltarea ei a încetat în 2013. După instalare, puteți crea un fișier sursă în mapa C:\MinGW (cum ar fi test.f95), în care să scrieți următoarele două rânduri:

```
print *,”OK”
end
```

Aveți grijă să salvați fișierul cu extensia ”.f95”. Pentru a testa funcționalitatea lui G95, deschideți o fereastră consolă și tastați:

```
cd C:\MinGW
g95 -c test.f95
```

Dacă nu s-a afișat nimic, atunci s-a creat imaginea obiect (fișierul test.o) fără erori în urma compilării, din care se poate crea fișierul executabil prin comanda următoare:

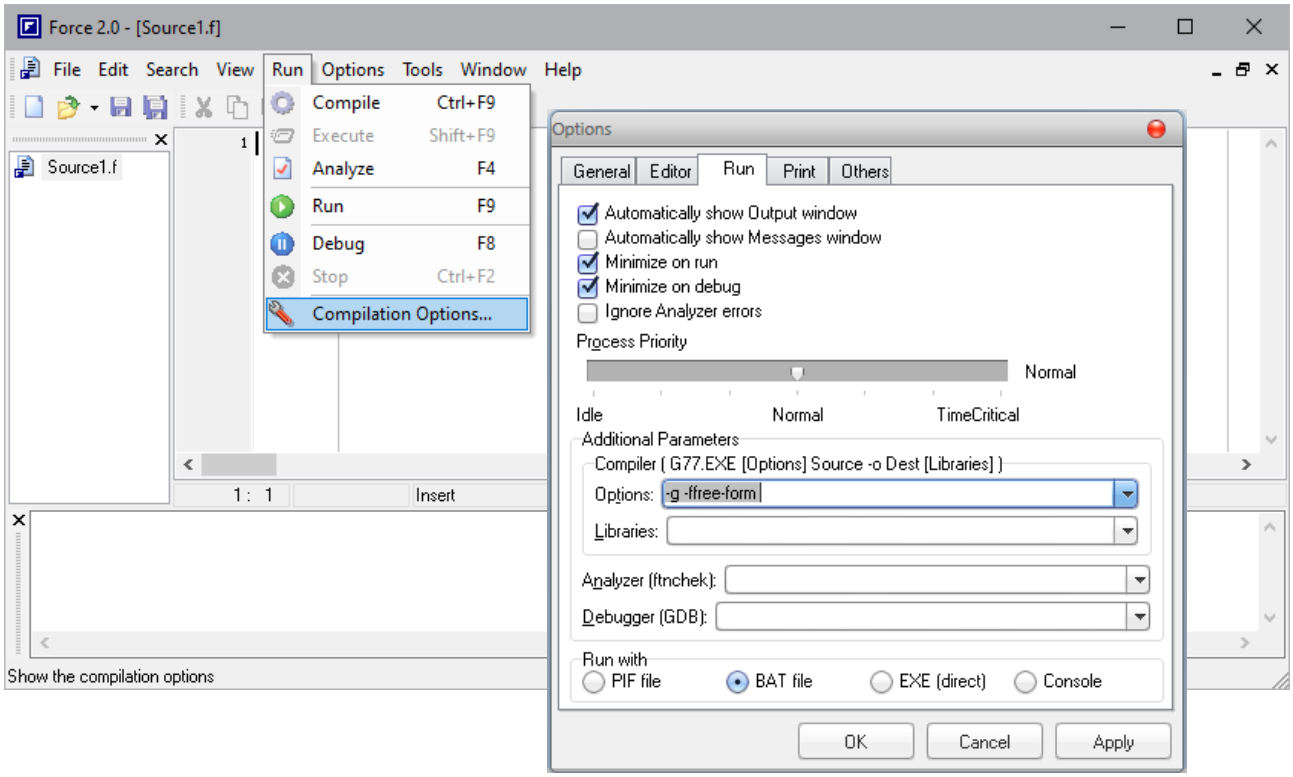
```
g95 test.f95 -o test.exe
```

Pentru a rula programul creat (fișierul test.exe) tastați `test` și literele `OK` ar trebui să apară pe ecran.
Dacă doriți să aflați mai multe despre opțiunile compilatorului G95 și facilitățile incluse, consultați fișierul G95Manual.pdf din mapa C:\MinGW\doc.
În cazul altor sisteme de operare, vă puteți orienta către GNU Fortran (GFortran 95) inclus în pachetul GCC.

Dar, este mai convenabil să folosiți un mediu de dezvoltare integrat (IDE) și pentru Windows opțiunea noastră este pachetul Force cu compilatorul G95 inclus. Pentru a-l obține, accesați pagina web a proiectului[2] și din secțiunea ”Downloads” alegeți varianta marcată în ilustrația de mai jos.

Filename	Size	Description
Force209GFortranSetup.exe	10.6 MB	Force 2.0.9 plus GNU Fortran (GFortran)
Force209G95Setup.exe	3.55 MB	Force 2.0.9 plus G95 Fortran (G95)
Force209G77Setup.exe	2.03 MB	Force 2.0.9 plus GNU Fortran 77 (G77)
Force3beta3Setup.exe	2.19 MB	Force 3.0 b3 plus GNU Fortran 77 (G77)

După instalare, fișierul G95Manual.pdf va fi în mapa C:\Program Files (x86)\Force 2.0\doc. Înainte de prima rulare sub Windows 10 și 11 va trebui să modificați proprietățile mapei C:\Program Files (x86)\Force 2.0, acordând drepturi depline administratorului (eventual și utilizatorului curent), altfel setările inițiale nu se vor salva. Pentru a putea utiliza forma liberă a fișierului sursă în Force, după instalare va trebui să completați opțiunile pentru compilare cu ”`-ffree-form`” (după cum este ilustrat) și să experimentați varianta convenabilă de rulare (prin fișier de comandă, rulare executabil direct, sau prin fereastră de consolă).



Desigur, se poate opta pentru alte variante, cum ar fi Code::Blocks cu Fortran[3], Geany[4] (nu are compilator Fortran inclus, dar se poate instala separat G95 pe Windows, sau GFortran pe alte sisteme de operare), etc. Există chiar și variante care rulează online prin program de navigare (cum ar fi: GDB online[5], myCompiler[6], Ideone[7], Jdoodle[8] și multe altele). Pentru alte posibilități sau resurse vizitați și paginile de la https://fortranwiki.org/ și https://fortran-lang.org/.

[2] Force Fortran - The Force Project. https://force.lepsch.com/
[3] Code::Blocks IDE for Fortran | CBFortran. https://cbfortran.sourceforge.io/
[4] Geany - The Flyweight IDE. https://www.geany.org/
[5] GDB Online Fortran Compiler. https://www.onlinegdb.com/online_fortran_compiler
[6] myCompiler – Create a new Fortran program. https://www.mycompiler.io/new/fortran
[7] Online Compiler and IDE. https://ideone.com/
[8] Online Fortran Compiler. https://www.jdoodle.com/execute-fortran-online/

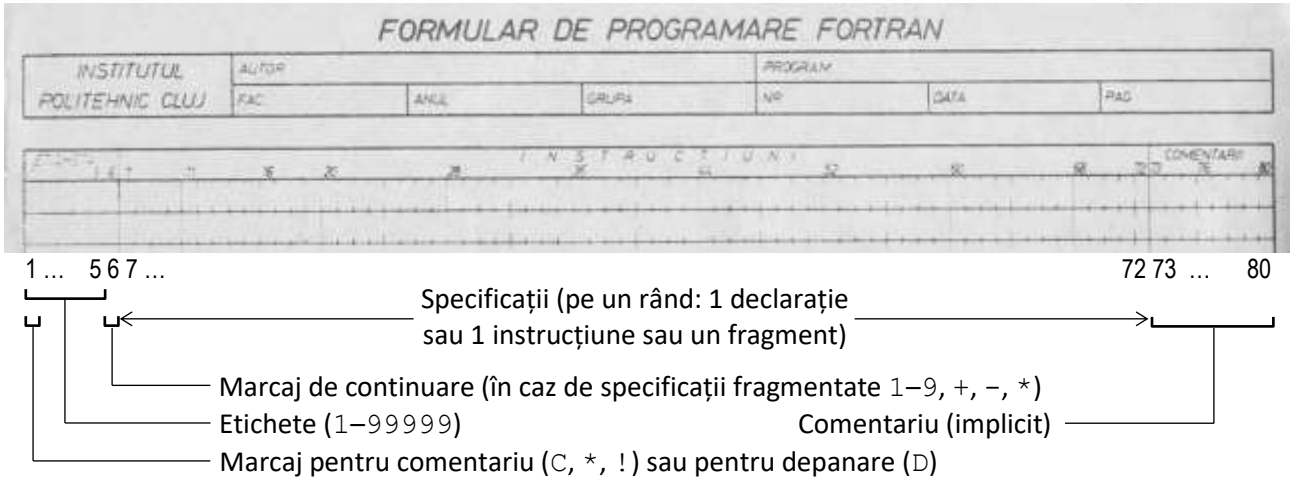
Alcătuirea fișierului sursă

Un fișier sursă poate conține una sau mai multe unități de program (acestea vor fi prezentate mai târziu), sau fragmente (sub formă de secțiuni). Redactarea fișierului sursă se poate face cu orice editor de text, cu condiția ca să rezulte un conținut alcătuit din caractere.
Setul de caractere utilizabile in limbajul Fortran constă din caractere alfanumerice (cele 26 litere mici sau mari ale alfabetului englez: `a–z`, `A–Z`; şi cifrele: `0–9`), 4 simboluri pentru operaţii aritmetice (`+`, `–`, `*`, `/`), precum şi dintr-un set determinat de caractere speciale (blank sau spaţiu, tabulator orizontal, virgulă, punct, apostrof, paranteze rotunde deschise şi închise, precum şi următoarele caractere: `=`, `$`, `&`). Limbajul Fortran 90 a mai extins această listă cu următoarele caractere speciale admise: `_`, `!`, `:`, `;`, `"`, `%`, `<`, `>`, `?`, `^` şi `#`. Conform convenţiilor anglo-saxone virgula are rolul de a separa elementele în cadrul unei liste, iar punctul este separatorul zecimal.
Pentru denumirea variabilelor, diferitelor secţiuni de program precum şi pentru identificarea funcţiilor, se folosesc nume simbolice. Dacă convenţiile versiunilor mai vechi ale limbajului au permis utilizarea a doar 8 caractere (alcătuite din caractere alfanumerice şi caracterul special "`$`"), Fortran 95 permite utilizarea a 31 de caractere (alcătuite din caractere alfanumerice, caracterul special "`$`" şi caracterul special "`_`"), dar primul caracter trebuie să fie întotdeauna o literă. Numele secţiunilor de program sunt considerate globale şi trebuie să fie unice în întreaga sursă, iar numele entităților trebuie să fie unice în cadrul aceleiași unități de program. În cazul numelor simbolice limbajul Fortran nu face diferențe între majuscule sau litere mici. Modul de redactare al fișierului sursă poate fi în *formă fixă* (Fortran 77), *formă tabulară* sau *formă liberă* (începând cu Fortran 90).

Indiferent de structura pe orizontală (formă fixă, liberă sau tabulară), structura pe verticală a unui fișier sursă trebuie să respecte următoarea succesiune de specificații: declarații (referitoare la unitatea de program, la entitățile utilizate), corp (conținând instrucțiunile care se vor efectua la rulare) și marcaj final. Dacă fișierul sursă conține doar un segment dintr-o unitate de program, oricare din cele 3 părți menționate (declarații, corp, marcaj final) poate lipsi, însă ordinea succesiunii trebuie respectată. În asemenea cazuri, conținutul unui asemenea fișier sursă se va include înaintea compilării (folosind specificația `INCLUDE`) în fișierul sursă corespunzător programului care se va crea.

Forma fixă

Respectă structura de redactare bazată pe cartele perforate și formularele tipizate vechi (ca cea din imagine), considerând lungimea unui rând (articol) de maximum 80 de caractere, având următoarea structură (sub imaginea formularului sunt marcate numerele coloanelor semnificative și conținutul admis pe zone):



Etichetele sunt numere întregi din cel mult 5 cifre, cu rol de referinţă în cadrul secţiunii de program, ele marcând instrucţiunile în faţa cărora apar (în rândul respectiv). Folosirea lor este opţională şi supusă unor restricţii (doar rândurile cu instrucţiuni executabile pot purta etichetă și etichetele nu pot depăși intervalul

coloanelor 1–5). Pentru ca o etichetă să fie validă, valoarea ei trebuie să fie cuprinsă în intervalul 1—99999. Dacă se doreşte marcarea unui rând ca şi comentariu, în prima coloană se va scrie litera "C" sau caracterul "*" (respectiv "!" începând cu Fortran 90), ceea ce va determina ignorarea rândului respectiv la compilare. Unele compilatoare permit şi folosirea caracterului "D" pentru marcarea în prima coloană a rândului curent ca şi comentariu, această facilitate permiţând compilarea (interpretarea) opţională a acestor rânduri în caz de depanare a sursei (debugging).
În Fortran 77 și anterior se scria doar o singură instrucţiune pe un rând, dar începând cu Fortran 90 se pot scrie mai multe instrucțiuni pe un rând (folosind caracterul ";" pentru a le separa).
Dacă spaţiul dintre coloanele 7 şi 72 din rândul curent nu este suficient pentru a scrie instrucţiunea dorită, ea poate fi extinsă marcând în coloana 6 pe următoarele rânduri continuarea celor precedente, prin cifre (doar din intervalul 1–9), litere sau prin unul din caracterele: +, −, * (începând cu Fortran 90 se poate folosi orice caracter în afară de cifra 0). Numărul liniilor de continuare admise depinde şi de compilatorul ales. Fortran 77 permitea 99 de fragmente (1 linie inițială cu 98 linii în continuare), dar standardul Fortran 90 permite doar 19 fragmente în forma fixă (39 fragmente în forma liberă), iar Fortran 95 permite până la 90 de linii de continuare în formatul fix (şi doar 31 de rânduri de continuare în formatul liber). Unele compilatoare permit extinderea zonei de interpretare a rândurilor până la coloana 80 (chiar 132, începând cu Fortran 90), dar în mod standard orice conţinut din intervalul coloanelor 72—80 este considerat implicit comentariu şi ca atare va fi ignorat la compilare.

Forma liberă

Nu conţine constrângerile descrise mai sus, instrucţiunile nu se limitează la o anumită încadrare pe coloanele liniilor, orice linie putând conţine până la 132 de caractere. În schimb spaţiile sunt semnificative, având rol separator în anumite cazuri (pentru a distinge nume, constante, cuvinte cheie sau etichete de instrucțiuni). Acest format a fost introdus doar începând cu Fortran 90 (acesta acceptă însă şi formatul fix şi tabular). În formatul liber comentariul este indicat prin caracterul "!" (începând din orice coloană) sau litera "C" scrisă în prima coloană (atenție la specificațiile și denumirile care încep cu acestă literă, să nu fie scise din prima coloană), iar prin caracterul "&" se marchează întreruperea unei specificații (la capăt) care se va continua pe rândul următor. Se pot scrie mai multe instrucţiuni pe un rând dacă sunt separate prin caracterul "**;**" (la sfârşitul unui rând acest caracter se ignoră în mod firesc).

Forma tabulară

Este de fapt o variantă atât a formei fixe, cât și a formei libere, fiind denumit așa din cauza utilizării caracterului de tabulare orizontală la începutul rândurilor. Dacă acest caracter <*Tab*> este primul, atunci rândul conține o specificație (declarație sau instrucțiune, sau eventual un marcaj). Dacă acest prim caracter este urmat de o cifră nenulă, cifra va marca un fragment de continuare a rândului anterior și trebuie să fie urmată de un spațiu pentru a-l separa de conținutul continuării. Caracterul <*Tab*> poate fi precedat doar de un marcaj de comentariu sau o etichetă. Lungimea rândurilor nu poate depăși coloana 72 în forma fixă și coloana 132 în forma liberă.

> Mențiune: În capitolele următoare vor fi prezentate sintaxe (reguli de scriere) și exemple în care se vor folosi și alte caractere. Parantezele drepte nu fac parte din sintaxă, ci marchează opționalitatea conținutului delimitat, iar punctele se referă la elemente repetabile. Secvențele cursive marchează conținutul corespunzător locului în care apar. Având în vedere că la orele de lucrări se va utiliza compilatorul G95 Fortran, specificațiile și instrucșiunile prezentate vor fi pentru această variantă și cele vechi, nefăcând parte din standard, vor fi marcate gri.

Tipul entităților

În Fortran orice entitate are tip, fie implicit, fie declarat în mod explicit. Există tipuri intrinseci și tipuri derivate (definite de programator, utilizând tipuri intrinseci sau tipuri derivate anterior definite). Tipurile intrtinseci sunt: `INTEGER` (numere întregi), `REAL` (numere reale, cu parte zecimală), `COMPLEX` (numere complexe, privite ca o pereche de numere reale), `LOGICAL` (valori logice, există doar două: `.TRUE.` și `.FALSE.`), `CHARACTER` (caracter sau șir alfanumeric), `BYTE` (valoare pe 8 biți, în variantele mai vechi ale limbajului). Declararea explicită a tipului unor entități se poate face conform sintaxei:

tip[(*fel*)][[, *atribut*]… ::] *lista_entități*

Cuvintele cheie pentru *tip* sunt: `INTEGER`, `REAL`, `COMPLEX`, `LOGICAL` și `CHARACTER` (în unele versiuni de Fortran există și `BYTE`), sau `TYPE(`*nume_tip*`)`, unde *nume_tip* se referă la un tip definit anterior de către programator. Ca și *fel* se poate specifica numărul de octeți (bytes) utilizați pentru memorare, precedat opțional de cuvântul cheie `KIND=`, în cazul tipului `CHARACTER` `LEN=` (sau `*` în sintaxa veche, fără paranteze). Această valoare depinde de tipul entităților, existând și variante implicite în funcție de compilator (de regulă 4 bytes pentru valori `REAL` și 2 sau 4 bytes pentru valori `INTEGER`). Valorile explicite pot fi: `1`, `2` sau `4`, eventual `8` la tipurile `INTEGER` și `LOGICAL`; `4` sau `8`, eventual `16` la tipurile `REAL` și `COMPLEX`. Entitățile de tipul `BYTE` se stochează pe câte 1 byte, la fel și caracterele individuale, deci lungimea nu se poate modifica explicit (dacă la `CHARACTER` se specifică lungimea, implicit va marca numărul de caractere din șir). Entitățile de tipul `INTEGER(1)` și `LOGICAL(1)` se vor stoca tot pe 1 byte.
Pe post de *atribut* se pot specifica următoarele:

- `ALLOCATABLE` pentru tablouri cu memoria alocată dinamic sau `DIMENSION(`*limite*`)` pentru tablouri cu memoria alocată static (vor fi prezentate la tablouri),
- `EXTERNAL` pentru entități redefinite de programator sau `INTRINSIC` pentru entități predefinite în Fortran,
- `INTENT(`*direcție*`)` pentru scopul de intrare/ieșire (unde *direcție* poate fi `IN` pentru intrare, `OUT` pentru ieșire, `INOUT` implicit),
- `PARAMETER` pentru valori constante,
- `PUBLIC` pentru entități vizibile, `PRIVATE` pentru entități locale (accesibile doar în unitatea curentă de program),
- `POINTER` pentru indicatori sau `TARGET` pentru ținte,
- `OPTIONAL` pentru entități temporare, `SAVE` pentru entități memorate.

În cazul în care nu se specifică niciun atribut se poate omite separatorul `::` (acesta având rolul de a delimita lista cuvintelor cheie din stânga de *lista_entități* din dreapta).

Pentru entități numerice există o regulă implicită, care (desigur) se poate modifica sau anula prin specificația `IMPLICIT` cu următoarea sintaxă:

`IMPLICIT` *tip* `(`*c*[`,`*c*]…`)`[`,` *tip* `(`*c*[`,`*c*]…`)`]…

unde *tip* trebuie să fie un specificator de tip intrinsec (sau tip derivat anterior definit), iar *c* reprezintă o literă sau un interval de litere în ordine alfabetică. Pentru anularea oricărei reguli implicite se scrie:

`IMPLICIT NONE`

În cazul anulării regulii implicite, trebuie declarate explicit tipurile tuturor entităților. Conform regulii implicite predefinite în Fortran, entitățile ale căror nume începe cu una dintre literele I, J, K, L, M sau N vor avea tipul `INTEGER`, iar restul vor avea tipul `REAL`. În consecință, dacă nu se modifică sau anulează această regulă, se pot omite declarațiile de tip ținând cont de respectarea regulii.

Exemple:	Explicații:
`IMPLICIT INTEGER(B,f-H,k)`	Toate entitățile a căror nume începe cu una din literele B, F, G, H sau K vor fi de tipul `INTEGER` (indiferent dacă se scriu cu majuscule sau cu litere mici).
`IMPLICIT REAL(n),COMPLEX(A-C)`	Toate entitățile a căror nume începe cu litera N vor avea tipul REAL, iar cele ale căror nume începe cu una din literele A, B, sau C vor fi de tipul `COMPLEX`.

`IMPLICIT NONE` `INTEGER I,j,K` `REAL X,Y`	S-a anulat regula implicită și trebuie definite explicit tipurile tuturor entităților. Cel e numite I, J și K vor fi de tipul `INTEGER`, iar cele numite X și Y vor fi de tipul `REAL`. Nefiind specificate atribute, separatorul `::` a fost omis (doar în dreapta este listă)
`REAL(KIND=8) Di,e33` *! Echivalent cu:* `REAL(8) dI,E33`	Entitățile (variabilele) numite DI și E33 vor fi de tip `REAL` și memorate pe câte 8 bytes (în variante mai vechi de Fortran se folosea tipul `DOUBLE PRECISION` în asemenea caz). După cum se poate observa, nu contează amestecarea / schimbarea literelor mici cu majuscule.
`COMPLEX(KIND=8) xC,Y1` *! Echivalent cu:* `COMPLEX(8) Xc,y1`	Entitățile (variabilele) numite XC și Y1 vor fi de tip `COMPLEX` și memorate pe câte 8 bytes (în variante mai vechi de Fortran se folosea tipul `DOUBLE COMPLEX` în asemenea caz). Fiind vorba de valori complexe care constau din perechi de valori (partea ”reală” și cea ”imaginară”), se vor folosi de fapt 16 bytes pentru fiecare entitate.
`INTEGER(2),INTENT(IN) :: Q`	Entitatea Q va fi de tipul `INTEGER` memorat pe 2 bytes și folosit doar pentru preluare de valori. Fiind specificat și un atribut (`INTENT`), este obligatorie utilizarea caracterelor `::` pentru delimitarea listei din stânga de cea din dreapta, chiar dacă în dreapta este un singur element.
`REAL,PARAMETER :: pi=3.14159`	Entitatea numită PI va fi de tipul `REAL` și cu valoarea constantă (nemodificabilă) de 3,14159.
`EXTERNAL :: SIN`	Entitatea numită SIN este declarată ca o variabilă, având tipul `REAL` implicit (pentru că numele începe cu litera S). În această situație numele SIN nu se va putea folosi pentru funcția trigonometrică intrinsecă din Fortran.
`REAL,POINTER,PRIVATE :: p,Q1`	Entitățile numite P și Q1 vor fi indicatori de tipul `REAL`, accesibile doar în unitatea de program curentă.

Definirea unui tip derivat se face conform sintaxei:

`TYPE` *nume_tip*
specificații
`END TYPE`[*nume_tip*]

După ce au fost definite, asemenea tipuri derivate se pot utiliza la specificarea tipului entităților, scriind `TYPE(`*nume_tip*`)` în locul cuvântului cheie care marchează tipul. Referirea la o componentă dintr-un asemenea tip derivat se poate face cu ajutorul selectorului %, sub forma *părinte*`%`*componentă*[`%`*subcomponentă*…], după cum se va ilustra într-un exempu mai jos.

În momentul în care se declară explicit tipul entităților, se pot atribui și valori inițiale. Atribuirea se poate realiza în cadrul *lista_entități* sau separat, prin specificația `DATA`. Sintaxa acestei specificații este următoarea:

`DATA` *lista_variabile*`/`*listă_valori*`/`[[`,`]*lista_variabile*`/`*lista_valori*`/` …]

unde pentru fiecare entitate din *lista_variabile* trebuie să corespundă o valoare din *listă_valori* (încadrată între caracterele ”/”), în ordinea succesiunii de la stânga la dreapta.

Exemple:	Explicații:
`TYPE comp` `  CHARACTER(LEN=24) nume` `  INTEGER zi` `  CHARACTER(3) luna` `  INTEGER :: an=2023` `END TYPE`	Tipul derivat numit COMP este definit ca fiind alcătuit din două șiruri alfanumerice (NUME având 24 de poziții pentru caractere iar LUNA 3) și două numere întregi (ZI și AN, cel din urmă fiind și inițializat cu valoarea 2023). Se poate observa opționalitatea cuvântului cheie `LEN=`, acesta nefiind utilizat la șirul alfanumeric LUNA.

`TYPE(comp) r23,r24`	Entitățile numite R23 și R24 vor avea tipul derivat definit anterior.
`CHARACTER at,stea*3` `INTEGER m1,m2,m3`	Declararea unor entități de tip `CHARACTER`: AT va conține 1 caracter, iar STEA va conține 3 caractere (în loc de `LEN=3` s-a folosit o sintaxă veche), urmată de declararea entităților M1, M2 și M3 de tip `INTEGER`.
`DATA at,m1,m2,m3/"@",2*1,5/` `DATA stea/"***"/,r24%an/2024/` `DATA r24%luna,r24%zi/"AUG",12/`	Variabila botezată AT va conține caracterul `@`, variabilele M1 și M2 valoarea 1 (2 bucăți, pentru cele 2 entități), iar M3 valoarea 5, după care se inițializează și șirul STEA cu caracterele `***`. Componentele AN, LUNĂ și ZI din entitatea R24 vor primi valorile 2024, AUG și 12.

Expresii

Expresiile sunt alcătuite din operatori, operanzi şi paranteze. Un operand este o valoare reprezentată printr-o constantă, variabilă, element de tablou sau tablou, sau rezultată din evaluarea unei funcţii. Operatorii pot fi intrinseci (recunoscuţi implicit de compilator şi cu caracter global, deci disponibili întotdeauna tuturor secvenţelor de program) sau definiţi de utilizator (în cazul în care un operator e descris explicit de programator ca funcţie). După modul de operare, putem vorbi de operatori unari (ce operează asupra unui singur operand) şi operatorii binari (ce operează asupra unei perechi de operanzi). Operatorii unari au prioritate față de cei binari. Evaluarea unei expresii are întotdeauna un singur rezultat, ce poate fi folosit pentru atribuire sau ca referinţă. Tipul valorii rezultate în urma evaluării unei expresii numerice depinde de tipul operanzilor şi de rangul acestora. Dacă operanzii din cadrul expresiei au ranguri diferite, valoarea rezultată va fi de tipul operandului cu cel mai mare rang (cu excepţia cazului în care o operaţie implică o valoare complexă şi una în dublă precizie, rezultatul în asemenea situaţii fiind de tip complex dublu). La verificarea corectitudinii unei expresii numerice compuse se recomandă să se ţină cont şi de tipul valorilor parţiale rezultate în cursul evaluării. Expresiile pot fi aritmetice (numerice), de șir (caractere), logice, respectiv de inițializare și specificare (începând cu Fortran 90).
Există expresii omogene (unde operatorii și operanzii sunt de același fel) și neomogene (unde felul operatorilor și operanzilor diferă). Prioritatea de evaluare a operatorilor din cadrul expresiilor neomogene este după cum urmează:

- operatori unari definiţi (funcţii);
- operatori numerici (în următoarea ordine: `**`, `*` sau `/`, `+` sau `−`);
- operatorul de concatenare pentru şiruri (caractere);
- operatori relaţionali (cu prioritate egală)
- operatori logici (în ordinea: `.NOT.`, `.AND.`, `.OR.`, `.EQV.`sau `.NEQV.` sau `.XOR.`).

Expresii aritmetice

Aşa cum sugerează denumirea lor, exprimă calcule numerice, fiind formaţi din operatori şi operanzi aritmetici, având rezultat numeric ce trebuie să fie definit matematic (împărţirea la zero, ridicarea unei baze de valoare zero la putere nulă sau negativă, sau ridicarea unei baze de valoare negativă la putere reală constituie operaţii invalide). Termenul de operand numeric poate include şi valori logice, deoarece acestea pot fi tratate ca întregi într-un context numeric (valoarea logică `.FALSE.` corespunde cu valoarea `0` de tip `INTEGER`). Operatorii numerici sunt: `**` (ridicarea la putere), `*` (înmulțire), `/` (divizare), `+` (adunare), `−` (scădere). Într-o expresie aritmetică cu mai mulţi operatori, prima dată se vor evalua întotdeauna părţile incluse în paranteze (dinspre interior spre exterior) şi funcţiile, prioritatea de evaluare a operatorilor intrinseci fiind după cum urmează: ridicarea la putere, înmulţirea şi împărţirea, plusul şi minusul unar, adunarea şi scăderea. În cazul operatorilor cu aceeaşi prioritate operaţiile vor fi efectuate de la stânga spre dreapta. Prin efect local, operatorii unari pot influenţa această regulă, generând excepţii în cazul unor compilatoare care acceptă asemenea expresii.

Expresie în Fortran	Formula matemică
`(3*X**2+1)/(2*Y)-1` `(3*X**2+1)/2/Y-1`	$\frac{3x^2+1}{2y}-1$
`X/(-5)*Y`	$\frac{x}{-5}y$
`X**(-Y)*3`	$x^{-y}3$

Expresie în Fortran	Formula matemică
`(3*X**2+1)/2*Y-1`	$\frac{3x^2+1}{2}y-1$
`X/(-5*Y)` `X/(-5)/Y`	$\frac{x}{-5y}$
`X**(-Y*3)`	x^{-y3}

Expresii şir (caractere)

Se pot alcătui cu operatorul de concatenare `//` (în variantele mai vechi de Fortran cu operatorul intrinsec `+`) sau cu funcţii create de programator, aplicate asupra unor constante sau variabile de tip caracter. Evaluarea unei asemenea expresii produce o singură valoare de tip caracter (șir). Concatenarea se realizează unind conţinuturile de tip caracter de la stânga spre dreapta fără ca eventualele paranteze să influenţeze rezultatul. Spaţiile conţinute de operanzi se vor regăsi şi în rezultat.

Expresii logice

Constau din operanzi logici sau numerici combinaţi cu operatori logici şi/sau relaţionali. Rezultatul unei expresii logice este în mod normal o valoare logică (echivalentă cu una din constantele literale logice `.TRUE.` sau `.FALSE.`), însă operaţiile logice aplicate valorilor numerice întregi vor avea ca rezultat tot valori de tip întreg, ele fiind efectuate bit cu bit în ordinea corespondenţei cu reprezentarea internă a acestor valori. Nu se pot efectua operaţii logice asupra valorilor numerice de tip `REAL`, `COMPLEX` sau `CHARACTER` în mod direct, însă asemenea tipuri de valori pot fi tratate cu ajutorul unor operanzi relaţionali în cadrul expresiilor logice. Operatorii de relație și cei logici sunt următoarele:

Operatori de relație		
Sintaxa	Semnificația	Sintaxa veche*
<	Mai mic	`.LT.`
<=	Mai mic sau egal	`.LE.`
==	Egal	`.EQ.`
/=	Diferit	`.NE.`
>	Mai mare	`.GT.`
>=	Mai mare sau egal	`.GE.`

* Se acceptă și variantele vechi (cele delimitate prin puncte din ultima coloană).

Operatori logici	
Sintaxa	Semnificația
`.NOT.`	Negaţia logică (complement logic) rezultă `.TRUE.` dacă operandul are valoarea `.FALSE.` şi rezultă `.FALSE.` dacă operandul are valoarea `.TRUE.`.
`.AND.`	Conjuncţia logică rezultă `.TRUE.` doar dacă ambii operanzi au valoarea `.TRUE.`, în caz contrar rezultă `.FALSE.`.
`.OR.`	Disjuncţia logică rezultă `.TRUE.` dacă unul din operanzi are valoarea `.TRUE.`, în caz contrar rezultă `.FALSE.`.
`.EQV.`	Echivalenţa logică rezultă `.TRUE.` dacă ambii operanzi au aceaşi valoare, dacă au valori diferite atunci rezultă `.FALSE.`.
`.NEQV.`	Inechivalenţa logică rezultă `.TRUE.` dacă operanzii sunt diferiţi, şi `.FALSE.` dacă sunt la fel.
`.XOR.`	Disjuncţia logică exclusivă (SAU exclusiv), efect similar cu inechivalenţa logică (`.NEQV.`).

Operatorii relaţionali au nivel egal de prioritate (se execută de la stânga la dreapta, dar înaintea celor logici şi după cei numerici), iar operatorii logici sunt daţi în ordinea priorităţii lor la evaluare. Operatorii relaţionali sunt binari (se aplică pe doi operanzi), la fel şi operatorii logici, cu excepţia operatorului de negaţie logică (`.NOT.`) care este unar. Operatorii relaţionali şi operatorii logici sunt prezentați în tabelele următoare.

Expresii de specificare şi iniţializare

Acestea pot fi considerate cele care conţin operaţii intrinseci şi părţi constante, respectiv o expresie scalară întreagă. Aşa cum sugerează şi denumirea lor, ele servesc la iniţializarea unor valori (de exemplu indicele pentru controlul unui ciclu implicit) sau la specificarea unor caracteristici (de exemplu declararea limitelor de tablouri sau a lungimilor din şiruri de caractere).

Constante folosite în expresii

Operanzii pot fi variabile (doar entitățile denumite pot avea valori variabile) sau constante. Valorile constante se specifică în funcție de felul lor, după cum se exemplifică în tabelul următor:

Tip constantă	Exemple:	Explicații:
Șir de caractere	`"Bla 3-1a"` `"anii '80"` `'anii ''80'`	Caracterele imprimabile se citează. În cazul în care în cadrul unui șir de caractere există apostrof sau ghilimele, fie se poate dubla apostroful interior (ca la al treilea șir), fie se va utiliza celălalt caracter pentru delimitare.
Număr decimal	`231` `50.66` `-.13e2` `256.`	Separatorul zecimal este punctul, la valori negative se marchează semnul. Cifrele nesemnificative se pot omite (prima valoare este întreagă, iar ultimele trei valori sunt reale). A treia valoare este -13.0 (`e2` înseamnă $\times 10^2$)
Număr binar	`B"1001"` `b"1011"` `B'1100'`	Se pot folosi doar cifrele 0 sau 1 (max. 256 de poziții) citând valoarea după marcajul cu litera `B`. Semnul minus în fața marcajului `B` nu are efect, iar în conținutul citat nu este acceptat. Citarea se poate face fie cu ghilimele, fie cu apostrof (fără a le combina).
Număr octal	`O"152"` `O'223'` `o"107"`	Se pot folosi doar cifrele de la 0 la 7 (max. 86 de poziții) citând valoarea după marcajul cu litera `O`. Ca și anterior, semnul minus nu are efect în față și nu se acceptă în conținut.
Număr hexa	`Z"15F"  X"15f"` `Z'1B0'  x'1B0'` `z"A28"  x"a28"`	Se pot folosi cifrele de la 0 la 9 și literele de la A la F (max. 64 de poziții) citând valoarea după marcajul `Z` sau `X`. Ca și anterior, semnul minus nu are efect în față și nu se acceptă în conținut.
Hollerith	`1H&` `3H123` `12Hla "Taverna"` `12Hab"1 x'+#.%@`	Sunt constante ce pot conține orice caractere imprimabile. Sintaxa este: *n*`H`*șir*, unde *n* este numărul de caratere (numărul pozițiilor din șir), `H` – marcajul Hollerith, iar *șir* conținutul. Deși inițial aceste constante au fost definite cu un conținut de până la 2000 de caractere, numărul de caractere poate fi între 1 și 32767 (2^{15}-1) pe sistemele cu arhitectura pe 32 de biți, respectiv între 1 și 2147483647 (2^{31}-1) pe platformele cu 64 de biți. Nu se pot folosi ca descriptor de format începând cu Fortran 90.

Funcţii intrinseci (pentru expresii)

Funcţiile intrinseci sunt specifice bibliotecilor utilizate, având nume simbolice prestabilite (rezervate). Printre ele există unele care nu fac parte din echiparea standard a mediului de programare, neregăsindu-se în toate variantele limbajului Fortran. Faptul că numele acestor funcţii sunt rezervate înseamnă că nu ar trebui să existe entități care să aibe nume coincidente cu cele ale funcţiilor intrinseci. De asemenea, numele acestor funcţii nu se recomandă să apară într-o listă a unei instrucţiuni `EXTERNAL`, acest fapt ducând la anularea definiţiei lor intrinseci. În asemenea cazuri, prin includerea numelor lor în liste ale instrucţiunii

declarative INTRINSIC, ele vor putea fi utilizate în proceduri definite ca unități (subprograme sau funcţii definite de utilizator). Sintaxa generală a funcțiilor este următoarea:

nume_funcție(*a*,[*a*]…)

unde *nume_funcție* este numele simbolic al funcției, iar *a* reprezintă argumentul. Câteva funcții intrinseci sunt prezentate în tabelul următor:

Funcția:	*nume_funcție*:	Rezultatul:
|x|	ABS(*x*)	Valoarea absolută (modulul) argumentului X specificat.
A×B	MATMUL(*A*,*B*)	Matricea rezultată din înmulțirea matricelor A și B.
A^T	TRANSPOSE(*A*)	Returnează transpusa matricei A.
A_{max}	MAXVAL(*A*)	Returnează valoarea maximă din tabloul A.
A_{max}(poz)	MAXLOC(*A*)	Returnează prima poziție a valorii maxime din tabloul A.
A_{min}	MINVAL(*A*)	Returnează valoarea minimă din tabloul A.
A_{min}(poz)	MINLOC(*A*)	Returnează prima poziție a valorii minime din tabloul A.
arccos(x)	ACOS(*x*)	Arccosinusul argumentului X exprimat în radiani.
arcsin(x)	ASIN(*x*)	Arcsinusul argumentului X exprimat în radiani.
arctg(x)	ATAN(*x*)	Arctangenta argumentului X exprimat în radiani.
caracter	ACHAR(*x*)	Returnează caracterul de pe poziția X din tabela de coduri.
complex-i	AIMAG(*x*)	Partea imaginară dintr-un număr complex X.
complex-r	REAL(*x*)	Partea reală dintr-un număr complex X.
cos(x)	COS(*x*)	Valoarea cosinusului argumentului X exprimat în radiani.
cosh(x)	COSH(*x*)	Cosinusul hiperbolic al argumentului X.
e^x	EXP(*x*)	Valoarea exponenţială a constantei Euler (e=2,71828…).
ln(x)	LOG(*x*)	Valoarea logaritmului natural al argumentului X.
log(x)	LOG10(*x*)	Logaritmul în baza 10 al argumentului X.
Lung. șir	LEN(*șir*)	Numărul de caractere din şirul considerat argument.
max(x,y,...)	MAX(*listă_valori*)	Valoarea maximă dintre elementele cuprinse în lista de argumente.
mărime	SIZE(*tablou*[,*ri*])	Returnează mărimea tabloului (după rangul RI, dacă s-a specificat).
min(x,y,...)	MIN(*listă_valori*)	Valoarea minimă dintre elementele cuprinse în lista de argumente.
nr. aleator	RAN(*x*)	Returnează un număr pseudoaleator între 0 şi 1.
$\sqrt{x}$	SQRT(*x*)	Rădăcina pătrată (radicalul) argumentului X.
rest div.	MOD(*x1*,*x2*)	Restul împărţirii argumentelor (X1/X2, cu semnul primului argument).
rotunjiri	NINT(*x*) ANINT(*x*)	Valoarea rotunjită a argumentului X la cel mai apropiat întreg. Valoarea rotunjită a argumentului X cu zero zecimale.
sin(x)	SIN(*x*)	Valoarea sinusului argumentului X exprimat în radiani.
sinh(x)	SINH(*x*)	Sinusul hiperbolic al argumentului X.
subșir	INDEX(*șir*,*subșir*)	Poziţia de început a subşirului în şirul specificat ca primul argument.
ΣA	SUM(*tablou*[,*ni*])	Returnează suma valorilor din tablou (după rangul RI, opțional).
tg(x)	TAN(*x*)	Tangenta argumentului X exprimat în radiani.
tgh(x)	TANH(*x*)	Tangenta hiperbolică a argumentului X.
trunchieri	INT(*x*) AINT(*x*)	Valoarea trunchiată a argumentului X la cel mai apropiat întreg. Valoarea trunchiată a argumentului X cu zero zecimale.

Instrucțiuni de intrare și de ieșire (I/E)

Operațiunile de citire se numesc intrări (I), iar cele de scriere sau afișare ieșiri (E). Pentru intrări secvențiale se poate folosi instrucțiunea READ, cu următoarele variante de sintaxă:

READ *f*[,*listă_intrare*]

în cazul citirii de la unitatea logică implicită (de regulă consola, deci tastatura), unde *f* este specificația de format (va fi prezentată mai târziu). O variantă mai generală are sintaxa:

READ([UNIT=]*u*[,[FMT=]*f*][,ERR=e_1][,END=e_2][,IOSTAT=*var*][,ADVANCE=*opt*]) [*listă_intrare*]

unde cuvântul cheie UNIT= poate fi omis dacă este primul parametru și *u* reprezintă numărul unității logice (valoarea fiind * pentru unitatea logică implicită, adică consola), cuvântul cheie FMT= poate fi omis dacă este al doilea parametru sau dacă nu se dorește utilizarea unei specificații de format *f* (cazul citirii fără format), e_1 este eticheta unei instrucţiuni executabile la care s-ar sări în cazul întâlnirii marcajului final în fișier (EOF) sau în lipsa valorilor de citit, e_2 este eticheta unei instrucţiuni executabile la care s-ar sări în cazul apariției unei erori la citire, iar *var* este numele unei variabile de tip INTEGER în care s-ar înregistra succesul / eșecul operațiunii de citire (în cazul citirii reușite rezultă 0, în cazul nereușitei rezultă valori mai mari care marchează coduri de eroare, -1 înseamnă EOF, iar -2 EOR). La ADVANCE=, *opt* poate fi "YES" (avans la rândul următor după citire, implicit) sau "NO" (fără avans la rândul următor). Entitățile în care se dorește memorarea valorilor citite vor constitui *listă_intrare*. Dacă *listă_intrare* nu există, singurul efect al instrucțiunii va fi oprirea temporară (până la apăsarea tastei <*Enter*>) a rulării programului.
Există și alte variante, cum ar fi citirea internă (pentru conversia caracterelor în numere întregi corespunzătoare pozițiilor din tabela de coduri), citirea directă (sărind la numărul de ordine a unei înregistrări dintr-o unitate logică formatată cu structură fixă), sau citirea pe bază de câmpuri cheie (în cazul fișierelor indexate).

Pentru operații de ieșire secvențiale se pot utiliza următoarele instrucțiuni:

PRINT *f*[**,** *listă_ieșire*]

în cazul scrierii pe unitatea logică implicită (de regulă consola, deci afișajul monitorului), unde *f* este specificația de format, sau

WRITE([UNIT=]*u*[**,**[FMT=]*f*][**,** ERR=e_1][**,** IOSTAT=*var*][**,** ADVANCE=*opt*]) [*listă_ieșire*]

unde notațiile sunt aceleași cu cele de la citire (fără END=e_2, deoarece la scriere nu are sens). Entitățile ale căror valori se doresc a fi scrise vor constitui *listă_ieșire*, în lipsa acestora se va scrie un rând gol.
Există și variante diferite, cum ar fi scrierea internă (pentru conversia din numere întregi în caractere, conform pozițiilor din tabela de coduri), scrierea directă (sărind la numărul de ordine a unei înregistrări dintr-o unitate logică formatată cu structură fixă) sau rescrierea unei înregistrări. Pentru scrierea în fișiere indexate se utilizează scrierea secvențială cu specificator de format, printre entitățile din *listă_ieșire* figurând și câmpurile de cheie.
În cazul utilizării simbolului * la specificația de format (marcând format implicit), se va lua în considerare de regulă tipul valorii din lista entităților. În cazul valorilor lungi, cum ar fi REAL(8) sau DOUBLE PRECISION, REAL(16), COMPLEX(8) sau DOUBLE COMPLEX, COMPLEX(16), nu se poate utiliza format implicit, ci trebuie utilizat o specificație de format corespunzător tipului.

Exemple:	Explicații:
READ * *! Echivalent cu:* READ(*,*)	Citire aparentă (fără intrare). Se va aștepta apăsarea tastei <*Enter*> (retur de car) pentru a continua rularea.
READ *,I,j *! Echivalent cu:* READ(*,*)i,J	Se vor citi două valori numerice (de tip INTEGER) introduse de la tastatură și vor fi memorate în variabilele I, respectiv J. Cele două valori se pot introduce separat (programul nu va avansa până când nu s-au introdus ambele valori) sau pe aceeași rând separate prin virgulă (sau prin spațiu).
PRINT * *! Echivalent cu:* WRITE(*,*)	Se va afișa un rând gol pe ecran (similar cu efectul caracterului <*LF*>).
PRINT *,"Max= ",max *! Echivalent cu:* WRITE(*,*)"Max= ",MAX	Se va afișa șirul de caractere citat (fără semnele de citare), urmat de conținutul (valoarea) variabilei MAX.

Specificația de format și descriptori

Descriptorii de format sunt ca niște șabloane aplicate pe datele de intrare sau de ieșire. Utilizarea lor se face de regulă prin specificația de format, a cărei sintaxă este următoare:

etichetă FORMAT(*listă_descriptori*)

însă, descriptorii pot să apară și sub formă citată în cadrul instrucțiunilor de citire sau de scriere. Există două categorii de descriptori: pentru editare date, respectiv pentru controlul formatării. Vor fi prezentate în cele ce urmează în tabele separate, cu exemple, folosind următoarele notații:

n – numărul de bucăți;
w – lungimea descriptorului (numărul total de poziții din câmpul respectiv);
m – numărul minim de poziții solicitate (din numărul total), are efect doar la ieșire;
d – numărul pozițiilor pentru partea zecimală (din numărul total);
e – numărul pozițiilor pentru exponent (din numărul total);
c – caracter, respectiv [*c*...] alte caractere opționale;
□ – spațiu (caracterul blank) afișat în exemple.

Tabel cu descriptorii pentru editarea datelor (în ordine alfabetică):

Sintaxa:	Destinația:	Exemple și explicații:
[*n*]A[*w*]	Date alfanumerice (CHARACTER)	Intrare: Format: Tip entitate: Valoare: ABC_D A5 CHARACTER(1): D ABC_D A5 CHARACTER(3): C_D ABC_D A5 CHARACTER(6): ABC_D□ Valoare: Format: Ieșire (5 poziții): ABC A5 □□ABC ABCDE A5 ABCDE ABCDEFG A5 ABCDE
[*n*]B*w*[.*m*]	Date numerice binare	Intrare: Format: Valoare (în forma decimală): 1001 B4 9 (toate cele 4 poziții citite) 1001 B2 2 (doar primele 2 poziții citite) 1001 2B2 2 și 1 (2 valori distincte) Valoare: Format: Ieșire: 13 B5 □1101 0 B2 □0 0 B2.2 00 Dacă *w*=0, la ieșire se vor utiliza atâtea poziții, câte sunt necesare afișării valorii (la intrare nu se admite *w*=0).
[*n*]D*w*.*d*	Date numerice în dublă precizie: REAL(8) adică DOUBLE PRECISION, sau COMPLEX(8), adică DOUBLE COMPLEX	Intrare: Format: Valoare (dublă precizie): 123.456E3 D9.3 123456.0D+0 12345678 D6.2 1234.56D+0 123.45678 D7.3 123.456D+0 După cum se poate observa, se citesc *w* poziții din intrare, dintre acestea *d* poziții pentru partea zecimală (de la separatorul zecimal spre dreapta – dacă la intrare nu este separator zecimal, atunci partea zecimală va rezulta considerând *d* poziții din capătul celor *w* citite). Marcajul ”D+0” din capăt indică doar că valorile se vor obține în dublă precizie. Valoare: Format: Ieșire: 123456.789 D11.2 □□□0.12D+06 0.0363 D10.3 □0.363D-01 -0.5555 D10.3 -0.556D+00 Afișarea va rezulta pe *w* poziții, din care *d* poziții pentru partea zecimală, însă trebuie avut în vedere că 1 poziție se va consuma pentru semnul valorii, încă 1 pentru separatorul zecimal (punctul), 1 poziție pentru litera descriptorului (D), ultimele 3 poziții pentru semnul și valoare exponentului. Dacă luăm în considerare că prima cifră semnificativă va fi prima zecimală, rezultă că este recomandabil ca *w*−*d* > 6. Dacă nu se

<table>
<tr><td></td><td></td><td>respectă această condiție, rezultă depășire de format (pe cele w poziții se vor afișa asteriscuri).</td></tr>
<tr><td>[n]Ew.d[Ee]</td><td>Date numerice în format explonențial (REAL sau COMPLEX)</td><td>Intrare: Format: Valoare:
□□123.45□□ E10.2 123.45
123456789 E9.3 123456.789
123.456D3 E9.3 123456.0 (simplă precizie!)
Ca și în cazul descriptorului precedent, se citesc w poziții din intrare, dintre acestea d poziții pentru partea zecimală (de la separatorul zecimal spre dreapta – dacă la intrare nu este separator zecimal, partea zecimală va rezulta considerând d poziții din capătul celor w citite). În cazul citirii unor valori tip dublă precizie cu acest descriptor (sau cu altele utilizabile, exceptând D) se va obține o valoare convertită în simplă precizie.</td></tr>
<tr><td></td><td></td><td>Valoare: Format: Ieșire:
123456.789 E11.5 0.12345E+06
-0.5555 E12.3E3 □-0.556E+000
0.0363 E5.2 ***** (depășire de format!)
Afișarea va rezulta pe w poziții, din care d poziții pentru partea zecimală, însă trebuie avut în vedere că 1 poziție se va consuma pentru semnul valorii, încă 1 pentru separatorul zecimal (punctul), 1 poziție pentru litera descriptorului (E), ultimele 3 poziții pentru semnul și valoare exponentului.
Dacă luăm în considerare că prima cifră semnificativă va fi prima zecimală, reiese că w−d > 6 [+(e−2)] (unde e este numărul cifrelor exponentului). Dacă nu se respectă această condiție, rezultă depășire de format (pe cele w poziții se vor afișa asteriscuri).</td></tr>
<tr><td>[n]ENw.d[Ee]</td><td>Date numerice în format explonențial ”ingineresc” (REAL sau COMPLEX)</td><td>Intrare: Format: Valoare:
123.45E+03 EN10.2 12345.0
-12345678 EN9.3 -12345.678
123.456D3 EN9.3 123456.0 (simplă precizie!)</td></tr>
<tr><td></td><td></td><td>Valoare: Format: Ieșire:
123456.789 EN11.2 1234.57E+02
-0.5555 EN7.1 ******* (depășire de format!)
0.0363 EN12.3 □363.000E-04
La afișare punctul zecimal va fi după primele 3 cifră.</td></tr>
<tr><td>[n]ESw.d[Ee]</td><td>Date numerice în format explonențial ”științific” (REAL sau COMPLEX)</td><td>Intrare: Format: Valoare:
□□□1.234E+03 ES12.3 1234.0
-10.234E-03 ES11.3 -0.010234</td></tr>
<tr><td></td><td></td><td>Valoare: Format: Ieșire:
123456.789 ES11.2 □□□1.23E+05
-0.5555 ES10.3 -5.555E-01
0.0363 ES12.3 □□□3.630E-02
La afișare punctul zecimal va fi după prima cifră semnificativă.</td></tr>
<tr><td>[n]Fw.d</td><td>Date numerice (REAL, F vine de la ”Float”)</td><td>Intrare: Format: Valoare:
12345678 F8.5 123.45678
-12345678 F8.2 -1234.56
24.77E+2 F8.2 2477.0</td></tr>
<tr><td></td><td></td><td>Valoare: Format: Ieșire:
2.3547188 F8.5 □2.35472
325.03 F5.2 ***** (depășire de format!)
-0.2 F5.2 -0.20</td></tr>
<tr><td>[n]Gw.d[Ee]</td><td>Date de tip intrinsec</td><td>Intrare: Format: Valoare:
-0.05566 G10.3 -0.05566</td></tr>
</table>

	(G vine de la "Generic")	123456	G10.3	123.456
		123456.789	G10.3	123456.79
		Valoare:	Format:	Ieșire:
		-45.66	G11.3	□-4.566E+01
		123456	G10.3	□□□□123456
		123456.78	G10.3	□0.123E+06
		Observații: Se poate utiliza pentru orice valoare de tip intrinsec. Dacă se specifică 0 pentru *w*, valoarea efectivă va fi aleasă de către procesor (în aceste cazuri doar variantele G0 sau G0.*d* se pot utiliza). Dacă *w* este diferit de 0, atunci și valoarea pentru *d* trebuie specificată. În cazul valorilor de tip INTEGER, CHARACTER și LOGICAL se va ignora valoarea specificată prin *d*, descriptorul comportându-se ca cel corespunzător acestor valori (I, A și L).		
[*n*]I*w*[.*m*]	Date numerice întregi (INTEGER)	Intrare:	Format:	Valoare:
		-1234	I4	-123
		□□□123	I6	123
		1234.6	I6	Eroare! (nu este INTEGER)
		Valoare:	Format:	Ieșire:
		0	I3	□□0
		0	I3.0	□□□
		1	I3.2	□01
		-123	I3	*** (depășire de format!)
		1.2	I4	Eroare! (nu este INTEGER)
[*n*]L*w*	Date logice	Intrare – se acceptă valorile logice scrise sub următoarele forme, inclusiv cu caractere mici (nu doar cu majuscule): .TRUE. sau .T sau T, sau dacă primele caractere citite sunt .T sau T (pentru adevărat); respectiv .FALSE. sau .F sau F, sau dacă primele caractere din intrare sunt .F sau F, sau conținutul este din spațiu/spații (pentru fals).		
		Valoare:	Format:	Ieșire:
		.TRUE.	L7	□□□□□□T
		.FALSE.	L1	F
		□□□	L3	□□F
		La ieșire se va obține doar 1 caracter (T sau F) indiferent de lungimea *w* specificată.		
[*n*]O*w*[.*m*]	Date numerice octale întregi (cu baza în 8)	Intrare:	Format:	Valoare (decimală):
		1111	O2	9
		1111	O4	585
		□11□	O4	9
		191	O3	Eroare! (9 nu este cifră octală)
		12	O0	Eroare! (*w* trebuie să fie pozitiv)
		Valoare (decimală):	Format:	Ieșire:
		11	O6.4	□□0013
		-11	O6	****** (depășire de format!)
		-11	O12	□37777777765
		1.5	O11	□7760000000
		81	O0	121
		Dacă *w*=0, la ieșire se vor utiliza atâtea poziții, câte sunt necesare afișării valorii (la intrare nu se admite *w*=0).		
[*n*]Z*w*[.*m*]	Date numerice hexa întregi (cu baza în 16)	Intrare:	Format:	Valoare (decimală):
		A2F	Z3	2607
		-A2F□	Z5	-2607

		3.A2F	Z5	Eroare! (punct zecimal invalid)
		Valoare (decimală):	Format:	Ieșire:
		3033	Z5	□□BD9
		16	Z5.4	□0010
		-10	Z8	FFFFFFF6
		1.1	Z8	3F8CCCCD
		2.5	Z0	40200000
		Dacă *w*=0, la ieșire se vor utiliza atâtea poziții, câte sunt necesare afișării valorii (la intrare nu se admite *w*=0).		
'c[c...]' sau "c[c...]"	Constante alfanumerice citate (CHARACTER)	Intrare: – (nu se pot folosi pentru citiri).		
		Format:		Ieșire:
		'aBc''DD:'	(apostrof dublat în interior)	aBc'DD:
		'aBc''DD:'	(ghilimele dublate în interior)	aBc"DD:
		'aBc"DD:'	(ghilimele în conținut)	aBc"DD:
		"abcD'#"	(apostrof în interior)	abcD'#
		"ab'cD'#"	(citat în citat!)	Eroare!

Tabel cu descriptorii de control:

Sintaxa:	Semnificația:	Explicații și exemple:
BN BZ	BLANK NULL BLANK ZERO	BN va avea ca efect ignorarea spațiilor din câmpurile numerice. BZ va avea ca efect "înlocuirea" spațiilor din câmpurile numerice cu cifre 0. Intrare: Format: Valoare: □1□□ BN,I4 1 □1□□ BZ,I4 100 1□23 BZ,I4 1023
*k*P	Power *k* este un factor de scalare, cu valoare din intervalul [-128, +127]	Permite interpretarea valorilor numerice cu zecimale, cu factorul de scară *k* în cazul descriptorilor D, E, F și G, dacă aceste valori nu conțin în mod explicit exponent. La intrări, o valoare pozitivă *k* va avea efectul mutării separatorului zecimal spre stânga, iar o valoare negativă spre dreapta (la ieșiri efectul va fi invers). Descriptorul P nu trebuie separat neapărat cu virgulă de descriptorul la care se referă, însă trebuie să-l preceadă. De exemplu, următoarele specificații vor avea același efect, factorul de scară fiind asociat cu primul descriptor pentru numere reale care îl urmează în listă (E10.3): (2P,I4,E10.3,F8.2) (I4,2P,E10.3,F8.2) (I4,2PE10.3,F8.2) Intrare: Format: Valoare: □□□37.614□ 3PE10.5 0.037614 □□□37.614□ -3PE10.5 37614.0 123.45 2PF8.3 1.2345 123.45 -2P,F8.3 12345.0 Valoare: Format: Ieșire: -170.139 1P,E10.3 -1.701E+02 -170.139 -1PE10.3 □-0.02E+04
S SP SS	Sign Sign Positive Sign Suppress	SP va determina afișarea semnului + în fața valorilor pozitive, iar SS va inhiba afișarea acestuia. S funcționează ca un comutator între SP și SS.
T*n* TLn	Tab Tab Left	Prin descriptorul T se precizează poziția *n* dintr-un rând, de la care se dorește citirea (sau la care se dorește scrierea).

<table>
<tr><td>TRn</td><td>Tab Right
n – poziția de tabulare</td><td>Considerând că se va introduce de la tastatură șirul:
123456789ABC
care se va citi cu secvența:
<pre>   CHARACTER(3) C1,C2
      READ(*,5) NR,C1,C2
 5   FORMAT(T7,I3,T1,A3,T10,A3)</pre>vor rezulta valorile: NR=789; C1=”123” și C2=”ABC”.
TRn permite specificarea poziției a n-a spre dreapta, de la poziția curentă, iar TLn spre stânga (n fiind un număr pozitiv). În cazul utilizării TL, dacă n este mai mare sau egal cu poziția curentă, atunci poziționarea se va face pe primul caracter din rând.</td></tr>
<tr><td>[n]X</td><td>Determină saltul peste n poziții din rândul curent</td><td>La intrare va determina ignorarea a n poziții, iar la ieșire va avea ca efect tipărirea de n spații (dacă apare la sfârșitul listei de descriptori, atunci nu are efect). În exemplu efectul este evidențiat prin marcarea cu □ la afișare):
<u>Cod sursă</u>:
<pre>   PRINT 4
      READ 3,nr
      PRINT 4,nr
 3   FORMAT(2X,I2)
 4   FORMAT(”numar:”,1X,I2)</pre><u>Afișaj</u>:
numar:
<u>Input</u>:
1234
<u>Afișaj</u>:
numar:□34</td></tr>
<tr><td>$
\</td><td>Suprimă saltul la rând nou (suprimă <<i>LF</i>>).</td><td>Va determina rămânerea cursorului pe ultima poziție curentă (<<i>LF</i>> este prescurtarea de la <i>Line_Feed</i>).
Varianta $ este mai nou introdusă, însă nu face parte din standard, iar varianta \ este cea veche (compilatorul G95 acceptă ambele). Oricare variantă s-ar utiliza, descriptorul trebuie să fie ultima din lista din care face parte.
<u>Cod sursă</u>:
<pre>   PRINT 5,”nr:”
      READ *,nr
      PRINT 4,nr
 5   FORMAT(A,$)
 4   FORMAT(”numar:”,1X,I2)</pre><u>Afișaj</u>+<u>Input (12)</u>:
nr:12
<u>Afișaj</u>:
numar:□12</td></tr>
<tr><td>[<i>n</i>]/</td><td>Induce <i>n</i> salturi la rând nou (induce <i>n</i> bucăți <<i>LF</i>>)</td><td>Se poate utiliza și fără <i>n</i>, de exemplu (3/) este echivalent cu (///), fără a necesita virgulă de separare. În următorul exemplu va introduce un salt la rând nou înainte de a afișa ”numar”, după care se vor introduce încă 2 salturi la rând nou:
<u>Cod sursă</u>:
<pre>   PRINT 5,”nr:”
   READ *,nr
   PRINT 4,nr
 5 FORMAT(A,$)
 4 FORMAT(/”numar:”,2/,3X,I2)</pre><u>Afișaj</u>+<u>Input (12)</u>:
nr:12
<u>Afișaj</u>:
□
numar:
□
□□□12</td></tr>
<tr><td>:</td><td>Termină controlul descriptorilor în lipsa elementelor din lista de intrare / ieșire</td><td>În următorul exemplu, în lipsa elementelor de afișat, descriptorul va determina ignorarea părții ”j2”:
<u>Cod sursă</u>:
<pre>   PRINT 1,3
   PRINT 2,14
 1 FORMAT(”i”,I2,1X,”i2”,I2)
 2 FORMAT(”j”,I2,:,1X,”j2”,I2)</pre><u>Afișaj</u>:
i□3i2□□
j14</td></tr>
</table>

Specificația de format poate să fie compusă și din expresii de șir (caracter). În următorul exemplu se arată cum s-ar putea aplica pentru N perechi de descriptori de forma (I2,1X), considerând 1< N <9:

Exemplu:	Explicații:
`CHARACTER fm(10)` `INTEGER j(9)` `PRINT *,"nr. (1-9): "` `READ(*,*)n` `k=48+n` `fm="("//ACHAR(k)//"(i2,1x))"` `PRINT *,"cele ",n," valori: "` `READ *,(j(i),i=1,n)` `WRITE(*,fm)(j(i),i=1,n)` `END`	Se declară șirul FM cu 10 poziții (se va utiliza ca specificație de format). Variabila J va avea 9 poziții (va fi un vector) și va conține valorile care se vor afișa cu descriptori de tipul `I2`. Se afișează șirul citat și în urma citirii se memorează numărul introdus (de bucăți dorite) în variabila N. Se compune numărul poziției din tabela de coduri a cifrei corespunzătoare numărului de bucăți (din variabila N), obținând caracterul cifrei care reprezintă acea valoare. Se construiește prin concatenare și folosind funcția intrinsecă `ACHAR` (ce returnează caracterul de pe poziția K din tabela de coduri) un șir alfanumeric, ce se atribuie variabilei FM, care va fi descriptorul de format cu N perechi de câmpuri de tip `I2` (întreg pe două poziții) și `1X` (un spațiu) pentru cele N valori. Se afișează șirul de caractere citat (inclusiv valoarea lui N), după care se citesc valorile corespunzătoare celor N poziții ale vectorului J (prin ciclu implicit). Se afișează cele N poziții din vectorul J (tot prin ciclu implicit) folosind specificația de format memorată în variabila FM ca șir alfanumeric.

Tablouri

Declararea tablourilor (în engleză ARRAYs) se poate realiza prin specificația de tip, sau prin specificațiile: `DIMENSION`, `COMMON` (deși eliminat începând cu Fortran 90, se poate utiliza în contiunare), `ALLOCATABLE`, respectiv `POINTER` sau `TARGET` (doar începând cu Fortran 95, în Fortran 90 există posibilitatea de a le defini ca tip "derivat"). Caracteristicile oricărui tablou sunt:

- Tip (orice tip intrinsec sau derivat),
- Rang (numărul de "dimensiuni", de ex. un vector are rangul 1, o matrice are rangul 2 etc. – numărul maxim fiind 7 în Fortran 77, respectiv 15 începând cu Fortran 90),
- Limite (de "jos" și de "sus" pentru fiecare "dimensiune" în parte, cea de "jos" înseamnă valoarea inițială a indicelui respectiv, iar cea de "sus" înseamnă valoare finală a indicelui respectiv),
- Mărime (rezultă din numărul total de elemente),
- Formă (rezultă din rang și din limite).

Tablourile cu forma identică sunt conforme (semnificând că se pot efectua anumite operații pe elementele lor, fără a apela explicit la indicii de poziție ai elementelor). Un scalar este conform cu orice formă de tablou.

Declararea unui tablou presupune fie alocarea zonelor de memorie pentru fiecare element al tabloului în momentul creării programului (metoda alocării statice), fie alocarea zonelor de memorie doar pentru rangul tablourilor în momentul creării programului, urmând ca alocarea efectivă a memoriei pentru elementele tablourilor să se realizeze pe parcursul rulării programului (metoda alocării dinamice).

Pentru optimizarea utilizării memoriei (spațiul redus înseamnă adrese mai puține, rezultând viteză mărită la rulare), atunci ar fi de dorit ca să nu se aloce spații nefolosite tablourilor. Asta se poate obține alocând dinamic memoria în cursul rulării, specificând doar mărimea strict necesară pentru tablouri.

Alocarea statică de memorie

Utilizând specificația de tip, sau `DIMENSION`, prin precizarea limitelor corespunzătoare fiecărei dimensiuni (rang) al unui tablou, se va aloca o mărime cunoscută (și nemodificabilă în cursul programului) acelui tablou în memoria calculatorului. Această mărime va fi una maximă, nefiind obligatorie utilizarea completă (se pot

utiliza mai puține poziții din tablou). Sintaxa declarării prin DIMENSION (alocarea statică a memoriei pentru tablouri):

[*Tip*,]DIMENSION(*limite*)[,*atribut*] :: *lista_tablouri*

sau

Tip[,*atribut*] :: *nume_tablou_1*(*limite*)[,*nume_tablou_i*(*limite*)…]

Exemple:	Explicații:
DIMENSION A(10,2,3),L(8)	Tabloul numit A are rangul 3 (3 ”dimensiuni”, în total 10x2x3=60 de poziții pentru elemente) și va fi implicit de tip REAL, iar cel numit L rangul 1 (8 poziții) și implicit de tip INTEGER (din cauza literei cu care începe numele).
REAL,DIMENSION(3,3) :: D,E	Tablourile D și E vor fi de tip REAL având rangul 2 și conforme (având froma identică).
INTEGER MAT(2:11,3)	Tabloul MAT este de tip INTEGER, cu rangul 2, având în total 30 de poziții pentru elemente. La primul rang limita inferioară este 2, iar cea superioara 11 (indicii de poziție fiind incrementați de la 2 până la 11), iar la al doilea rang limita inferioară este 1 (implicit), iar cea superioară 3.

Stocarea tablourilor în memorie se realizează prin înșiruirea elementelor, incrementând succesiv indicile de poziție, în ordinea lor. Iată un exemplu pentru tabloul D (menționat mai sus, cu rangul 2 și mărimea 3x3=9 poziții):

D								
(1,1)	(2,1)	(3,1)	(1,2)	(2,2)	(3,2)	(1,3)	(2,3)	(3,3)

După cum se poate observa, indicele de pe prima poziție va fi incrementat (pornind de la valoarea inițială, ce a limitei inferioare, până la limita superioară), după aceea indicele următor, și așa mai departe…

Exemplificând cu o matrice ca: $\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{vmatrix}$, am putea spune că stocarea în memorie a elementelor se face pe coloane.

Inițializarea elementelor unui tablou, utilizând specificația DATA:

Exemple:	Explicații:
DIMENSION A10(10,10)	Declararea unui tablou cu numele *A10* (implicit de tip REAL), având în total 10x10=100 de poziții pentru elemente.
DATA A10/100*1.0/	Inițializare prin nume: toate cele 100 de elemente din tabloul *A10* vor primi valoarea 1,0.
DATA A(1,1),A(10,2),A(5,5)/2*3.3,2.0/	Inițializare prin elemente: elementele de pe pozițiile (1,1) și (10,2) vor primi valoarea 3,3, iar elementul de pe poziția (5,5) va primi valoarea 2,0.
DATA ((A(i,j), i=1,5,2),j=1,3)/9*3.5/	Inițializare prin ciclu: elementele de pe pozițiile (1,1), (3,1), (5,1), (1,2), (3,2), (5,2), (1,3), (3,3) și (5,3) vor primi câte o valoare de 3,5. Indicele *i* pornește de la valoarea 1 și ajunge la valoarea finală 5 prin incrementare cu câte 2.

Mențiune: specificația DATA este declarație, deci trebuie să fie trecută înaintea oricărei instrucțiuni executabile. În continuare urmează câteva exemple cu instrucțiuni executabile (instrucțiunea de atribuire).

Exemple:	Explicații:
L=10 *! Echivalent cu:* L(1)=10;L(2)=10;L(3)=10;L(4)=10 L(5)=10;L(6)=10;L(7)=10;L(8)=10	L este tabloul cu 8 poziții, iar numărul 10 este o valoare scalară. Având în vedere că un scalar este conform cu orice tablou, toate cele 8 poziții din tabloul L vor primi prin atribuire valoarea 10.
L=L*2 *! Echivalent cu:* L(1)=L(1)*2;L(2)=L(2)*2;L(3)=L(3)*2 L(4)=L(4)*2;L(5)=L(5)*2;L(6)=L(6)*2 L(7)=L(7)*2;L(8)=L(8)*2	Toate cele 8 elemente din tabloul L vor avea valoarea înmulțită cu 2 (având în vedere că scalarul 2 este ”conform” cu tabloul L).
D=-1.2 E=2.*D	Fiecare element din tabloul D va primi valoarea -1,2 prin atribuire. Tablourile D și E sunt ”conforme” (au forma identică 2x3), în consecință fiecare element din tabloul E va primi valoarea -2,4 (rezultând din înmulțirea cu 2 a valorii -1,2).

Subșiruri

Sintaxa referirii la un subșir (adică o parte dintr-un tablou cu rangul 1):

nume_șir(\[*start*\]:\[*stop*\]\[:*pas*\])

Exemple:	Explicații:
REAL,DIMENSION(6) :: VA INTEGER,DIMENSION(0:5) :: VB	Tablourile VA și VB declarate cu câte 6 poziții (indicele de poziție în cazul tabloului VA poate lua valori de la 1 până la 6 inclusiv, cu pasul de incrementare +1, iar în cazul tabloului VB de la 0 până la 5, tot 6 poziții).
VA(3:5)=1.0	Elementele de pe pozițiile 3, 4 și 5 din vectorul VA vor primi valoarea 1,0 prin atribuire.
VB(1:5:2)=1	Elementele de pe pozițiile 1, 3 și 5 (indicele pornește cu valoarea 1 și va ajunge până la 5 cu pasul 2) din vectorul VB vor primi valoarea 1 prin atribuire.
CHARACTER(LEN=8) :: TIT=”ALanDALa”	Șirul de caractere numit TIT va avea 8 poziții și va fi inițializat cu caracterele citate (câte un caracter pe fiecare poziție).

Referirile următoare la părți ale entității denumite TIT (din exemplul anterior) înseamnă caracterele (citate) din coloana dreaptă:

TIT(2:4)	”Lan” - caracterele de pe pozițiile 2-4 (inclusiv),
TIT(5:5)	”D” - caracterul de pe poziția 5,
TIT(:5)	”ALanD” - caracterele până la poziția a 5-a,
TIT(5:)	”DALa” - caracterele de la poziția a 5-a,
TIT(:)	”ALanDALa” - echivalent cu referirea TIT,
TIT(10:)	Șir cu lungime nulă (nu există caractere de la poziția 10),
TIT(5:10)	Ultima poziție din șir este 7 (LEN=7), iar 10 > LEN. Nu este permisă o asemenea referire, va genera eroare!

Câteva funcții intrinseci pentru șiruri de caractere:

LEN(*șir*) - returnează lungimea (numărul de caractere) al șir-ului specificat.

INDEX(*subșir*,*șir*) - returnează poziția (de început a) subșir-ului în *șir*, sau 0 dacă nu este.

TRIM(*șir*) - returnează *șir*-ul fără caracterele de spațiu (blank) finale.

Alocarea dinamică de memorie

Prin metoda alocării dinamice, în momentul creării programului se va rezerva spațiu în memorie doar pentru numărul de dimensiuni sau rangul tabloului (creând posibilitatea generării de adrese pentru pozițiile posibile), iar alocarea spațiului de memorie pentru tablou se va realiza în mod efectiv doar atunci când se ajunge la instrucțiunile care o impun. Desigur, programatorul va trebui să aibă în vedere, că în acest mod nu sistemul de operare va gestiona zona de memorie alocată tabloului, ci programul, ca urmare și eliberarea acestor zone de memorie va trebui controlată prin instrucțiuni. Dacă se ignoră acest aspect, atunci după fiecare rulare a programului vor rămâne zone de memorie ocupate și rămase fără control (fenomenul fiind numit ”scurgeri de memorie”), ceea ce în urma unor rulări repetate poate duce la umplerea memoriei de lucru, îngreunând sau chiar blocând funcționarea calculatorului. Alocarea dinamică a memoriei se poate realiza într-una din următoarele trei variante, prin:

- Tablouri alocabile (prin specificația ALLOCATABLE),
- Tablouri indicatoare/țintă (prin specificația POINTER sau TARGET – începând cu Fortran 95),
- Tablouri alocate automat (prin transfer de date către proceduri).

Tablouri alocabile

În cazul utilizării specificației ALLOCATABLE trebuie rezervat corespunzător rangul (numărul dimensiunilor) tabloului, iar limitele inferioare și superioare ale tabloului se pot fixa în orice moment în cadrul programului (cu condiția, ca să nu fie deja fixate). Specificația ALLOCATABLE nu se poate combina cu specificațiile COMMON, DATA, EQUIVALENCE sau NAMELIST. Tablourile alocabile pot fi utilizate între proceduri numai dacă în prealabil s-a alocat memoria pentru ele (s-au fixat limitele pentru fiecare dimensiune), însă pentru evitarea ”scurgerilor de memorie” spațiul alocat pentru ele trebuie eliberat (dealocat) înainte de sfârșitul procedurii în care s-a alocat memoria. Nu se admite alocarea multiplă simultană de memorie pentru un tablou (pentru testarea stării de alocare se poate utiliza funcția intrinsecă ALLOCATED, care va returna valoarea logică .TRUE. în cazul în care tabloul are deja spațiu alocat). Pentru eliberarea spațiului de memorie alocat unui tablou se poate utiliza funcția intrinsecă DEALLOCATE, iar pentru alocarea spațiului de memorie funcția intrinsecă ALLOCATE. Sintaxa declarării prin ALLOCATABLE (alocarea dinamică a memoriei pentru tablouri):

[*Tip*,]ALLOCATABLE[,*atribut* … ::] *nume_tablou_1*(:[,:]…)[,*nume_tablou_i*(:[,:]…)…]

Mențiuni: pentru fiecare rang se rezervă câte o poziție marcată prin caracterul ":" în paranteza rotundă după *nume_tablou*. Pentru alocarea dinamică a memoriei se pot utiliza și atributele POINTER sau TARGET, sintaxa declarării prin POINTER sau TARGET fiind similară cu sintaxa pentru ALLOCATABLE, doar cuvântul cheie utilizat diferă (se va scrie POINTER sau TARGET în loc de ALLOCATABLE).

În cazul utilizării alocării dinamice a memoriei prin ALLOCATABLE, POINTER sau TARGET, pentru alocarea efectivă a spațiului necesar tablourilor se va utiliza în cadrul corpului fișierului sursă funcția:

ALLOCATE(*nume_tablou_1*(*limite*)[,*nume_tablou_i*(*limite_i*)…])

În asemenea situații trebuie avut în vedere că la terminarea rulării programului se va termina și controlul asupra zonei de memorii alocate tablourilor (în cadrul programului), așa că prin rulări succesive se poate ajunge în situația ocupării totale a memoriei de lucru cu zone alocate tablourilor care nu se mai pot controla. Pentru evitarea acestor situații, memoria alocată dinamic în cadrul unui program trebuie eliberată tot în cadrul aceluiași program (înainte de a pierde controlul asupra zonei de memorie) prin funcția:

DEALLOCATE(*lista_tablouri*)

Eliberarea zonelor de memorie alocată poate fi necesară și pentru alocarea unor zone diferite de memorie (cu alte dimensiuni) aceloraşi tablouri, starea alocării putând fi testată prin utilizarea funcției ALLOCATED(*lista_tablouri*), de exemplu, în cadrul unul IF logic simplu (sintaxa fiind prezentată la instrucțiunile de control) folosit pentru eliberarea memoriei alocate anumitor tablouri:

IF(ALLOCATED(*lista_tablouri*)) DEALLOCATE(*lista_tablouri*)

Exemple:	Explicații:
ALLOCATABLE X12(:,:),B(:)	Tabloul numit X12 are rangul 2 (rezervarea fiecărei ”dimensiuni” s-a marcat cu câte un caracter :), iar tabloul B va fi un vector, având rangul 1. Ambele

	tablouri vor fi implicit de tip REAL. Numărul efectiv de poziții din fiecare tablou se va specifica ulterior.
REAL,ALLOCATABLE,DIMENSION(:) :: n,m	Tablourile N și M vor acea același rang (1). În acest caz trebuie specificat și tipul entităților (REAL).
REAL,ALLOCATABLE :: v(:),m(:,:) … ALLOCATE(v(10),m(0:9,-2:7)) … DEALLOCATE(v,m) … IF(ALLOCATED(m)) THEN DEALLOCATE(m) ENDIF … ALLOCATE(m(3,3)) … DEALLOCATE(m) …	S-au declarat două tablouri alocabile, vectorul V cu rangul 1 (rezervat prin caracterul ” : ”) și matricea M cu rangul 2 (adică, 2 dimensiuni). S-au alocat 10 poziții pentru vectorul V, respectiv 10x10=100 poziții pentru matricea M (de la 0 la 9 inclusiv pentru prima dimensiune și de la -2 la 7 inclusiv pentru cea de a doua dimensiune). Atenție la închiderea parantezei pentru funcția ALLOCATE! Eliberarea spațiilor de memorie alocate celor 2 tablouri precedente. Testarea printr-o expresie logică a stării tabloului M pentru evitarea unei alocări duble (nepermise) de memorie. Eliberarea spațiului de memorie (prin DEALLOCATE) se va întâmpla doar dacă funcția intrinsecă ALLOCATED va indica (prin valoarea logică .TRUE. returnată), că există deja spațiu alocat în prealabil. Dacă funcția intrinsecă ALLOCATED va returna valoarea logică .FALSE., înseamnă că nu există spațiu de memorie alocat tabloului precizat, în consecință nu va fi nevoie de eliberarea memoriei (funcția intrinsecă DEALLOCATE va fi ignorată). Realocarea unui spațiu de memorie nou tabloului M, de data aceasta de 3x3=9 poziții. Eliberarea memoriei alocate tabloului M, înainte de sfârșitul unității de program.

<u>Tablouri indicatoare/țintă</u>

Un POINTER (indicator) nu conține date, ci indică spre un scalar sau spre un tablou în care se pot stoca date. Acel scalar sau tablou spre care indică un POINTER, trebuie să aibă atributul TARGET (țintă). Spre deosebire de tablourile alocabile, un tablou POINTER (sau TARGET) poate fi transmis unei proceduri chiar și fără alocarea prealabilă a spațiului de memorie. Spațiul de memorie pentru un asemenea tablou se va aloca efectiv doar la executarea programului. Sintaxa specificării acestor tablouri este similară cu cea a tablourilor alocabile, cu mențiunea, că tablourile POINTER cer de regulă o interfață explicită (la proceduri interne interfața este cunoscută). Întrucât specificarea tablourilor POINTER și TARGET este posibilă doar începând cu Fortran 95 (în mod asemănător cu utilizarea specificației ALLOCATABLE deja prezentate), în cazul Fortran 90 asemenea tablouri se pot crea prin specificația de tip derivat (exemplificat mai jos).

Exemple:	Explicații:
POINTER C(:,:,:) REAL,TARGET :: kt(:)	Tabloul numit C are rangul 3 (rezervarea fiecărei ”dimensiuni” s-a marcat cu câte un caracter ” : ”) și va fi implicit de tip REAL (implicit), POINTER. Tabloul KT are rangul 1 (ca un vector) și este declarat explicit de tip REAL (deoarece numele începe cu litera K). Numărul efectiv de poziții din aceste tablouri se va specifica ulterior.

TYPE tablou_P REAL,DIMENSION(:),POINTER :: tp END TYPE … TYPE(tablou_P),ALLOCATABLE :: vp(:) … READ(*,*)n,m … ALLOCATE(vp(n)) DO i=1,n ALLOCATE(vp%tp(m)) ENDDO … DEALLOCATE(vp) …	S-a specificat un tip derivat denumit TABLOU_P care conține o componentă de tip REAL cu atributul POINTER sub formă de vector (tablou cu rangul 1), denumit TP. S-a utilizat tipul derivat TABLOU_P pentru a declara un alt tablou alocabil VP, tot sub formă de vector (tablou cu rangul 1). Asta înseamnă, că fiecare element din VP va avea în componență câte un tablou de tip REAL cu atributul POINTER sub formă de vector (tablou cu rangul 1), denumit TP. Presupunând valorile entităților scalare de tip INTEGER N și M cunoscute (în exemplul alăturat prin citire), se poate aloca prin funcția intrinsecă ALLOCATE spațiul de memorie dorit (în exemplul alăturat N poziții) pentru tabloul VP, respectiv spațiul de memorie dorit (în exemplul alăturat M poziții) pentru fiecare componentă de tip TP din VP. Astfel, fiecare element din tabloul tip POINTER VP va avea câte M poziții, ceea ce înseamnă că tabloul VP va avea în total NxM poziții. Eliberarea spațiului alocat, ca la tablourile alocabile.

<u>Tablouri automate</u>

Tablourile alocate automat sunt variabile permise doar în cadrul procedurilor (subrograme și funcții), iar limitele inferioare și superioare pentru fiecare dimensiune rezervată (rang rezervat) în prealabil se vor stabili în momentul apelului procedurii. Aceste tablouri nu pot fi inițializate (elementele lor nu pot conține valori inițiale) și nu se pot transfera valori prin asemenea tablouri între proceduri.

Exemple:	Explicații:
SUBROUTINE puncte(nr,poz) INTEGER,INTENT(IN) :: nr REAL,INTENT(OUT) :: poz REAL :: zona(nr),zona_2(2*nr) …	Un subprogram numit PUNCTE s-a specificat cu argumentele NR și POZ (a căror valoare se va afla în momentul intrării în subprogram). Argumentul NR este de tip INTEGER și se va utiliza doar ca transfer către subprogramul PUNCTE, iar POZ este de tip REAL și se va utiliza doar pentru transfer din subprogramul PUNCTE către unitatea de program care apelează subprogramul. Când subprogramul PUNCTE se activează (și valoarea NR cunoscută se transferă subprogramului), se alocă memoria (mărimea definită) tablourilor ZONA și ZONA_2 de tip REAL.
PROGRAM functie_tablou ALLOCATABLE X(:) PRINT *,"n: " READ *,n ALLOCATE(X(n)) PRINT *,"cele ",n," valori: " READ *,(X(i),i=1,n) PRINT *,func(n,X) DEALLOCATE(X) CONTAINS FUNCTION func(k,X) DIMENSION func(k),X(k) DO i=1,k func(i)=X(i)	Un exemplu mai complex cu o funcție definită ca procedură internă și ca tablou automat (prin extinderea unui exemplu anterior de la prezentarea funcțiilor). Tabloul X care se va transfera funcției FUNC (împreună cu dimensiunea N), beneficiază de alocare dinamică a memoriei. Memoria alocată pentru tabloul X va fi eliberată înainte de terminarea programului. În momentul apelării funcției se vor transfera argumentele, iar rezultatul se va obține prin numele funcției (în cazul de față N valori distincte). Funcția (ca tablou) va avea automat K poziții (corespunzător valorii N transferate în momentul apelării). Fiecare element din tabloul FUNC va primi

ENDDO END FUNCTION END	valoarea de pe poziția corespunzătoare din tabloul X.

Instrucțiuni de control

În această categorie intră instrucțiunile decizionale, cele pentru salt și pentru repetiții (cicluri), precum și cele pentru oprirea sau suspendarea rulării unui program.

Instrucțiuni decizionale

Există mai multe tipuri, unele având și variante structurate (introduse cu Fortran 90), sintaxa lor fiind următoare:

IF(*expresie_aritmetică*) e_1, e_2, e_3	Decizia aritmetică (IF aritmetic) presupune testarea valorii rezultatului expresiei aritmetice față de zero, fiind precizate 3 etichete (nu neapărat diferite). În cazul în care rezultatul din *expresie_aritmetică* este strict negativ, se va efectua un salt la eticheta e_1, dacă rezultatul este nul (valoarea 0) la eticheta e_2, iar în cazul unui rezultat strict pozitiv la eticheta e_3.
IF(*expresie_logică*) *instrucțiune*	Decizia logică nestructurată (IF logic simplu), cu ramură vidă, permite specificarea unei singure instrucțiuni. Această instrucțiune va fi executată doar în cazul în care *expresie_logică* rezultă adevărată (cu valoarea .TRUE.). În caz contrar (rezultând valoarea .FALSE. pentru *expresie_logică*) instrucțiunea va fi ignorată.
IF(*expresie_logică_1*) THEN *instrucțiuni_1* [ELSE IF(*expresie_logică_i*) THEN *instrucțiuni_i*] [ELSE *instrucțiuni_x*] ENDIF	Decizia logică structurată (IF logic structurat) poate fi cu ramură vidă (varianta în care apar doar cuvintele cheie IF, THEN și ENDIF), sau nu. Cuvintele cheie ELSE IF se pot scrie și împreună, ca ELSEIF, în multe variante ale limbajului Fortran. Dacă se specifică mai multe secvențe ELSE IF, *expresie_logică_i* trebuie să fie distinctă pentru fiecare secvență, fără posibilități de îndeplinire simultană a mai multor condiții exprimate (mențiunea fiind valabilă și pentru *expresie_logică_1*). În cazul în care *expresie_logică_1* rezultă cu valoarea .TRUE., se vor executa cele precizate în blocul *instrucțiuni_1*. Altfel, dacă *expresie_logică_1* rezultă cu valoarea .FALSE., se va considera prima *expresie_logică_i* (dacă s-a specificat) care, dacă rezultă cu valoarea .TRUE. va conduce la executarea celor precizate în blocul *instrucțiuni_i* corespunzător. Doar dacă toate expresiile logice precedente au rezultat cu valoarea .FALSE. vor fi executate cele precizate în blocul *instrucțiuni_x*.
SELECT CASE (*expresie*) [CASE(*set_criterii_i*) *instrucțiuni_i*] [CASE DEFAULT *instrucțiuni_x*] END SELECT	Decizia generalizată permite testarea valorii oricărei expresii. Trebuie avut grijă ca fiecare *set_criterii_i* specificat să fie clar, și fără suprapuneri între ele! Ramura CASE DEFAULT va fi considerată (executând *instrucțiuni_x*) doar în cazul neîndeplinirii condițiilor specificate în toate *set_criterii_i* anterioare.

Variantele structurate pot conține alte instrucțiuni structurate (structuri), însă **fără a intersecta acestea**. Instrucțiunile structurate conținute trebuie să înceapă și să se termine în cadrul aceluiași bloc (marcate în sintaxele precedente cu *instrucțiuni_1*, *instrucțiuni_i*, respectiv *instrucțiuni_x*).

Exemple:	Explicații:
`  CHARACTER r` `    …` `1 WRITE(*,*)'Introduceti valorile: '` `    …` `  WRITE(*,*)'Reluati? (D/N): '` `  READ(*,*)r` `  IF(r.EQ.'d'.OR.r.EQ.'D') GOTO 1` `…`	Declararea unei entități de tip caracter (1 poziție). Instrucțiune executabilă cu eticheta 1. Citirea unui caracter și memorarea acestuia în R, iar în urma testării valorii printr-un `IF` logic simplu și eventual un salt necondiționat la instrucțiunea cu eticheta 1.
`   CHARACTER r` `    …` `1  WRITE(*,*)'Introduceti valorile: '` `    …` `   WRITE(*,*)'Reluati? (D/N): '` `   READ(*,*)r` `   IF(r=='d'.OR.r=='D') THEN` `    GOTO 1` `   ENDIF`	Exemplul precedent, utilizând în locul unui `IF` logic simplu, un `IF` logic structurat (fără ramura `ELSE`) și înlocuind `.EQ.` cu `==`.
`   IF(x+1)3,1,6` `3  WRITE(*,*)"rezultat negativ"` `   GOTO 2` `1  WRITE(*,*)"rezultat nul"` `   GOTO 2` `6  WRITE(*,*)"rezultat pozitiv"` `2  CONTINUE`	Testarea rezultatului expresiei numerice `x+1`, printr-un `IF` aritmetic, urmând ca în funcție de rezultat să se afișeze dacă este negativ, nul, sau pozitiv.
`IF(x+1<0) THEN` `   WRITE(*,*)"rezultat negativ"` `   ELSE IF(x+1==0) THEN` `     WRITE(*,*)"rezultat nul"` `     ELSE` `       WRITE(*,*)"rezultat pozitiv"` `ENDIF`	Exemplul anterior, utilizând în locul unui `IF` aritmetic un `IF` logic structurat.
`    IF(x/2)3,3,6` `3   WRITE(*,*)"rezultat <=0"` `    GOTO 2` `6   WRITE(*,*)"rezultat >0"` `2   CONTINUE`	Testarea valorii rezultate în urma evaluării expresiei numerice `x/2`, printr-un `IF` aritmetic, urmând ca în funcție de rezultat să se afișeze dacă este mai mic sau egal cu zero, sau strict pozitiv.
`SELECT CASE(x/2)` `    CASE(:0)` `    WRITE(*,*)"rezultat <=0"` `    CASE DEFAULT` `    WRITE(*,*)"rezultat >0"` `END SELECT`	Exemplul anterior, dar în locul unui `IF` aritmetic folosind o structură `SELECT CASE` cu o expresie aritmetică. Criteriul specificat prin `(:0)` semnifică toate valorile numerice până la zero (inclusiv).
`SELECT CASE(x/2<=0)` `    CASE(.TRUE.)` `    WRITE(*,*)"rezultat <=0"` `    CASE(.FALSE.)` `    WRITE(*,*)"rezultat >0"` `END SELECT`	Exemplul anterior, folosind o structură `SELECT CASE` cu o expresie logică. Valoarea în urma evaluării unei expresii logice poate fi `.TRUE.` sau `.FALSE.` (doar una dintre cele două valori logice).

<u>Instrucțiuni de salt</u>

Variantele instrucțiunilor de salt (folosind cuvintele cheie `GO TO` sau ca `GOTO`) au sintaxa următoare:

`GOTO` ***e***
`GO TO` ***e***

Saltul necondiționat, *e* este eticheta la care se va sări în momentul executării acestei instrucțiuni. Eticheta *e* trebuie să marcheze o

instrucțiune executabilă (trebuie scrisă în fața instrucțiunii la care se va sări pe parcursul rulării).

`GOTO (`*lista_etichete*`) [,]`*expresie*
`GO TO (`*lista_etichete*`) [,]`*expresie*

Saltul calculat. Prin evaluarea *expresie*i va rezulta poziția etichetei din *lista_etichete*, care se va considera pentru efectuarea saltului. Se subînțelege că valoarea rezultată din *expresie* trebuie să fie un număr întreg, strict pozitiv. Dacă este negativ, zero, sau mai mare decât numărul elementelor din *lista_etichete*, saltul nu se va efectua.

Instrucțiuni pentru cicluri (repetări)

Instrucțiunile pentru realizarea ciclurilor sunt structurate în Fortran 95 (cele la care se marchează finalul structurii în loc de etichetă cu END_*nume* fiind introduse cu Fortran 90). Fiind vorba de instrucțiuni structurate (structuri), acestea pot conține alte structuri (de exemplu, ciclu în ciclu, sau decizie structurată în ciclu, sau ciclu în structură decizională etc.), dar nu pot fi intersectate. Instrucțiunile structurate trebuie să înceapă și să se termine în cadrul aceluiași bloc (marcat cu *instrucțiuni*) în sintaxele de mai jos. Variantele marcate cu etichetă la final sunt moștenite din versiunile anterioare de Fortran.

`DO` *e* [[`,`] *control_ciclu*]
instrucțiuni
e ultima_instrucțiune_executabilă

Ar corespunde cu un ciclu post-condiționat, cu mențiunea că, dacă s-a specificat *control_ciclu*, acesta va fi întâi evaluat (conform celor menționate mai jos). La această instrucțiune structurată, dacă se includ alte cicluri în ciclu având același corp, se admite utilizarea unei singure etichete pentru marcarea sfârșitului structurilor (nu se consideră intersectare). Dacă ultima specificație din corpul ciclului nu este o instrucțiune executabilă (de exemplu `ENDIF`, sau similar), se poate utiliza instrucțiunea neutră `CONTINUE` (prezentată mai jos).

`DO` [*control_ciclu*]
instrucțiuni
`ENDDO`

Diferența față de varianta anterioară constă prin marcajul final `ENDDO` (unele variante de Fortran acceptând și `END DO`).

Sintaxa pentru *control_ciclu* este următoare:

contor_ciclu=*valoare_inițială*`,` *valoare_finală*[`,` *pas_incrementare*]

Interpretarea acestuia se face atribuind *valoare_inițială* la *contor_ciclu* și verificând dacă acesta este sub *valoare_finală* (dacă nu este, ciclul va fi ignorat fără a se executa vre-o instrucțiune din corpul ciclului). După o primă parcurgere a instrucțiunilor din corpul ciclului, *contor_ciclu* se modifică cu valoarea specificată la *pas_incrementare*. În cazul în care nu s-a specificat *pas_incrementare*, acesta va fi considerat în mod implicit +1. Se verifică ca valoarea din *contor_ciclu* să nu fi depășit *valoare_finală*, pentru a relua din nou executarea instrucțiunilor din corpul ciclului. Ieșirea din ciclu se va realiza în momentul în care *contor_ciclu* va avea o valoare peste *valoare_finală*. Nu este permisă modificarea explicită (prin instrucțiuni) în corpul ciclului a vre-unei componente din *control_ciclu*.

În cazul în care nu se specifică *control_ciclu*, ieșirea se poate realiza cu instrucțiunea `EXIT` sau se poate realiza o buclă ”infinită” (se poate opri rularea prin apăsarea simultană a tastelor <*Ctrl*> și <*C*>, determinând ieșirea din program prin întrerupere forțată).

`DO` *e* [`,`] `WHILE (`*expresie_logică*`)`
instrucțiuni
e ultima_instrucțiune_executabilă

Ar corespunde cu un ciclu pre-condiționat. Instrucțiunile din corpul ciclului vor fi executate doar dacă *expresie_logică* rezultă cu valoarea `.TRUE.` (și ciclul se va realiza doar atâta vreme cât timp această valoare există). În momentul în care *expresie_logică* devine `.FALSE.`, se va ieși din ciclu. Dacă ultima specificație din corpul ciclului nu este o instrucțiune executabilă (de exemlplu, un marcaj `ENDIF`, sau similar), se poate utiliza instrucțiunea neutră `CONTINUE` (prezentată într-un exemplu mai jos).

`DO WHILE(`*expresie_logică*`)` *instrucțiuni* `ENDDO`	Diferența față de varianta anterioară constă prin marcajul final `ENDDO` (unele variante de Fortran acceptând și `END DO`).

Pe lângă aceste instrucțiuni, există și câteva instrucțiuni de control utilizabile pentru reluarea sau ieșirea din buclele descrise anterior.

`CYCLE`	Determină reluarea executării instrucțiunilor anterioare dintr-un ciclu, fără a parcurge toate instrucțiunile din corpul ciclului.
`EXIT`	Permite părăsirea corpului unui ciclu (ieșirea din ciclu).
[*e*] `CONTINUE`	Este o instrucțiune executabilă fără efect. Sensul utilizări este doar de a purta etichetă.

Exemple:	Explicații:
`   DO i=1,10` `     WRITE(*,*)i` `   ENDDO` `   WRITE(*,*)i` `! Echivalent cu:` `   DO 8 i=1,10` `8  WRITE(*,*)i` `   WRITE(*,*)i`	Ciclu pentru afișarea valorii *contor_ciclu* (*i*), în varianta cu `ENDDO`, sau, utilizând eticheta 8 ca marcaj final al corpului ciclului.
`   DO i=1,n` `     DO j=i+1,n` `     REZ(i,j)=1.0/(i+j)` `     ENDDO` `   ENDDO` `! Echivalent cu:` `   DO 20 i=1,n` `   DO 20 j=i+1,n` `20 REZ(i,j)=1.0/(i+j)` `! Echivalent cu:` `   DO 11 i=1,n` `   DO 20 j=i+1,n` `20 REZ(i,j)=1.0/(i+j)` `11 CONTINUE`	Ciclu în ciclu, în varianta cu `ENDDO` (primul `ENDDO` este pentru ciclul cu contorul *j*, considerat interior) și în varianta utilizării unei etichete (20) pentru marcajul finalului corpului ciclului. Se poate observa, că în a doua variantă s-a folosit o singură etichetă (nu se consideră intersectare de structură în asemenea situații). Se poate observa că se permite folosirea valorii contoarelor de ciclu, dar este interzisă modificarea explicită a valorii acestora. Desigur, se poate folosi și instrucțiunea `CONTINUE` (amintită mai sus) în asemenea situații. Ciclul interior va fi cel cu eticheta 20 (ultima structură deschisă trebuie să fie prima închisă).
`    DO` `      READ *,N` `      IF(N==0) EXIT` `    ENDDO`	O variantă de ciclu fără control, se va ieși din ciclu datorită instrucțiunii `EXIT` (în cazul în care s-a introdus pentru N valoare nulă).
`    DO i=1,4` `      PRINT *,i` `      IF(i>2) CYCLE` `      PRINT *,i` `    ENDDO` `    PRINT *,'gata...'`	Pe monitor se vor afișa următoarele: `1` `1` `2` `2` `3` `4` `gata...` Instrucțiunea `CYCLE` va determina reluarea ciclului (fără a executa instrucțiunile care îl urmează) din momentul în care valoarea lui *i* depășește 2.
`    CHARACTER*132 LINIE` `    READ ('A'),LINIE` `    i=1`	Se definește un șir de caractere (numit LINIE) cu 132 de poziții (s-a folosit sintaxa veche din Fortran 77) și se citesc caracterele printr-o singură instrucțiune (utilizînd descriptorul pentru valori alfanumerice).

`DO WHILE (LINIE(i:i)==" ")` `  i=i+1` `ENDDO`	Cât timp se vor întâlni spații (caractere blank) pornind de la începutul șirului, se va incrementa valoarea *i*, care este utilizată și pentru specificarea poziției caracterelor din șir (a se vedea subșiruri). La final, *i* va conține poziția primului caracter diferit de spațiu din șirul LINIE (numărul total al spațiilor +1).

În cazul operațiilor de intrare/ieșire se pot utiliza cicluri implicite (asemănător cu exemplificarea de la specificația `DATA`), ca în următorul exemplu:

Exemplu:	Explicații:
`DIMENSION A(10,10)` `READ *,"nr. de linii din matricea A: ",nl` `READ *,"nr. de coloane din matricea A: ",nc` `DO i=1,nl` `PRINT *,"elementele de pe rândul ",nl," :"` `READ *,(A(i,j), j=1,nc)` `ENDDO` `…` `PRINT *,"tabloul A:"` `PRINT *,((A(i,j),", ",j=1,nc),i=1,nl)`	Declararea unui tablou cu 10x10=100 poziții. Folosirea unui ciclu implicit în interiorul unuia explicit pentru citirea elementelor de pe un rând dintr-o matrice. Interpretare: citește A(*i*,*j*) cât timp indicele de poziție *j* pornește de la valoarea 1 și ajunge (incrementat la fiecare pas cu +1) la valoarea lui NC. Afișarea elementelor din matricea A, element cu element, urmate de caracterul ",". Ciclul cu contorul *j* este în interiorul ciclului cu contorul *i*. În acest exemplu afișarea valorilor va fi în linie.

Instrucțiuni pentru oprirea rulării

`STOP` [*cod_oprire*] — Termină executarea, oprind rularea programului. Dacă se specifică *cod_oprire*, atunci acesta va fi afișat (*cod_oprire* poate fi un număr întreg, sau un șir de caractere citate, se utilizează în cazul mai multor posibilități de oprire, pentru identificarea ramurii parcurse).

`PAUSE` [*cod_oprire*] — Se poate folosi până în Fortran 90, fiind eliminat din standardul Fortran 95 (se poate înlocui cu o instrucțiune goală de citire), însă este acceptat de G95. Oprește temporar rularea programului, afișând (dacă s-a precizat) și *cod_oprire* (poate fi un număr întreg, sau un șir de caractere citate). Pentru continuarea rulării va trebui apăsată tasta <*Enter*>. În toate cazurile va determina afișarea mesajului:
`PAUSE statement executed. Hit Return to continue.`
Dacă s-a specificat și un *cod_oprire*, acesta va fi afișat între cuvintele "`PAUSE`" și "`statement`" din mesajul de mai sus.

Utilizarea unităților logice (periferice și fișiere)

Părțile interne sau externe ale unui sistem de calcul folosite pentru operații de intrare (citiri) și de ieșire (scrieri), respectiv pentru stocarea datelor, se consideră unități fizice. Acestea sunt accesibile în limbajul Fortran ca unități logice (implicite sau explicite) corespunzătoare acelor unităților fizice. Unitatea logică implicită (marcată cu valoarea `*` în instrucțiunile care necesită specificarea) este consola, adică ansamblul alcătuit din tastatură și afișaj (ecranul monitorului) – la intrări se consideră tastatura, iar la ieșiri afișajul. Unitățile logice care trebuie specificate în mod explicit sunt fișierele, respectiv perifericele (imprimanta, unitate cu bandă magnetică etc.), acestora fiind alocată o valoare numerică întreagă. Indicarea unității

logice la care se referă o instrucțiune de intrare / ieșire se face prin această valoare numerică. Unele valori au și unități logice predefinite în variantele Fortran mai vechi, ca de exemplu 1, 2, 3 și 4 pentru fișiere numite FOR00*n*.DAT (unde în locul caracterului *n* va apare în denumirea fișierului cifra corespunzătoare de la 1 la 4), sau 5 pentru dispozitive de intrare (cititor de cartele, tastatură etc.), respectiv 6 pentru dispozitive de ieșire (imprimantă, afișaj etc.). Alocarea acestor referințe (numere) unităților logice se poate face în mod explicit prin instrucțiunea OPEN. Sintaxa acestei instrucțiuni executabile este următoarea:

OPEN(*parametru*[,*parametru*]…)

unde *parametru* poate fi un *cuvânt_cheie*, sau de forma *cuvânt_cheie*=*valoare* (fiecare *parametru* poate să fie specificat doar o dată în cadrul listei din paranteză).

Tabel cu parametri din instrucţiunea OPEN în ordine alfabetică Fortran:

cuvânt_cheie	*valoare*	Explicație	Valoare implicită
ACCESS=	”SEQUENTIAL” ”DIRECT” ”APPEND” ”STREAM”	Stabilirea modului de accesare a unității logice: - secvențial, - direct, - adăugare înregistrări, - acces prin poziția stocării.	”SEQUENTIAL” (secvențial, rând după rând).
ACTION=	”READ” ”WRITE” ”READWRITE”	Modul în care se poate folosi unitatea logică: - citire (fără scriere), - scriere (fără citire), - citire și scriere.	”READWRITE” (citire și scriere).
BLANK=	”NULL” ”ZERO”	Interpretarea spaţiilor: - nu se transformă (blank), - se transformă în cifre 0 în cazul numerelor.	”NULL” (nu se transformă).
CONVERT=	”NATIVE” ”SWAP” ”LITTLE_ENDIAN” ”BIG_ENDIAN”	Permite specificarea unui format numeric (de conversie / interpretare) pentru datele neformatate: - nativ (fără convertire), - comută (între formatele LITTLE_ENDIAN și BIG_ENDIAN), - ultimele două specifică explicit codificările.	”NATIVE” (nu se convertesc datele).
DECIMAL=	”COMMA” ”POINT”	Caracterul pentru separarea zecimalelor.	”POINT”.
DELIM=	”NONE” ”APOSTROPHE” ”QUOTE”	Specificarea caracterului delimitator (pentru constante tip CHARACTER) la operații de I/E: - fără delimitator, - caracterul apostrof, - caracterul ghilimele.	”NONE” (fără delimitator).
ERR=	*etichetă*	Eticheta instrucțiunii la care se va sări în caz de eroare la deschiderea unității logice.	Nu există, nu se face salt.
FILE=	*șir*	Specificarea fișierului care va fi utilizat ca unitate logică. Specificatorul de fișier este	Depinde de unitatea logică şi de sistemul de operare.

		un șir, deci se delimitează cu apostrof sau ghilimele în cazul în care se citează, iar în cazul în care este conținut de o entitate de tip CHARACTER, se specifică numele acelei entități.	
FORM=	”FORMATTED” ”UNFORMATTED”	Formatul unității logice (a fișierului) accesat: - cu format, - neformatat.	Depinde de valoarea cuvântului cheie ACCESS. Dacă este ”DIRECT” atunci va fi considerat ”FORMATTED”, altfel va fi considerat ”UNFORMATTED”.
IOSTAT=	*variabilă*	Returnează o valoare scalară de tip INTEGER în *variabilă*, prin care se poate vedea succesul (sau insuccesul) accesării unității logice. În cazul deschiderii unității logice valoarea va fi 0.	Nu este implicit.
PAD=	”YES” ”NO”	Specifică dacă înregistrările se completează cu spații (caractere blank), când formatul necesită mai multe poziții decât are valoarea introdusă, sau dacă nu se completează.	”YES” (se completează cu spații când e necesar).
POSITION=	”ASIS” ”REWIND” ”APPEND”	Specifică poziționarea într-un fișier: - așa cum este, - revenire la început, - adăugare la sfârșit.	”ASIS” (poziția curentă).
RECL=	*număr*	Lungimea înregistrării în bytes pentru unitatea logică în cazul accesului direct, sau lungimea maximă în cazul accesului secvențial (*număr* este un întreg pozitiv).	Depinde de valoarea specificată la cuvântul cheie ACCESS.
SHARE=	”COMPAT” ”DENYNONE” ”DENYWR” ”DENYRD” ”DENYRW”	Controlează cum pot accesa alte procese unitatea logică în mod simultan: - compatibil (nepartajat), - fără restricții, - fără scriere din alte programe, - fără citire din alte programe, - exclusiv, fără acces din alte programe.	”DENYNONE” (fără restricții).
STATUS=	”OLD”	Starea unității logice (a	”UNKNOWN” (se

	”NEW” ”REPLACE” ”SCRATCH” ”UNKNOWN”	fişierului) la deschidere: - existent (dacă nu există, se obține eroare), - nou (dacă există, va da eroare), - suprascierea unui fișier, - temporar (se șterge după închidere), nu se poate folosi cu FILE=, - necunoscut (dacă există se deschide, altfel se crează).	deschide dacă există, se crează dacă nu există).
UNIT=	*număr*	Numărul unităţii logice (asociat fişierului, sau dispozitivului dorit) care se accesează (*număr* este un întreg pozitiv). Numărul unității logice se poate specifica și fără cuvântul cheie UNIT=, dacă este primul parametru din paranteză.	Nu este implicit.

Deconectarea unității logice (în cazul fișierelor înseamnă închiderea lor) se poate specifica prin instrucțiu–nea executabilă CLOSE, a cărei sintaxă este următoarea:

CLOSE(*parametru*[,*parametru*]…)

unde *parametru* este de forma *cuvânt_cheie*=*valoare* (fiecare *parametru* poate să fie specificat doar o dată în cadrul listei din paranteză).

Instrucțiunea CLOSE va determina și înregistrarea (scrierea) marcajului de sfârșit al fișierului (<EOF>, adică ”End of File”) în momentul deconectării (închiderii fișierului).

Tabel cu parametri din instrucţiunea CLOSE (în ordine alfabetică):

cuvânt_cheie	*valoare*	Explicație	Valoare implicită
ERR=	*etichetă*	Eticheta instrucțiunii la care se va sări în caz de eroare la deconectarea unității logice (*etichetă* este un număr întreg pozitiv).	Nu există, nu se face salt.
IOSTAT=	*variabilă*	Returnează o valoare de tip INTEGER scalar în *variabilă*, prin care se poate vedea succesul (sau insuccesul) deconectării unității logice. În cazul închiderii corecte a unității logice, valoarea variabilei va fi 0.	Nu este implicit.
STATUS=	”KEEP” ”DELETE”	Opțiune pentru a păstra sau a șterge fișierul.	”KEEP” (salvează fișierul) dacă STATUS= nu era ”SCRATCH”.
UNIT=	*număr*	Numărul unităţii logice (asociat fişierului, sau dispozitivului dorit) care se deconectează (*număr* este un întreg pozitiv). Numărul unității logice se poate specifica și	Nu este implicit.

		fără cuvântul cheie UNIT=, dacă este primul parametru din paranteză.	

Atenție: Un fișier deschis cu specificarea STATUS="SCRATCH" nu poate fi salvat sau tipărit (afișat), o asemenea tentativă va genera eroare la rulare, iar dacă la deschidere s-a specificat ACTION="READ", atunci fișierul nu va putea fi șters la deconectare (închidere). Un fișier utilizat doar pentru citire nu trebuie neapărat închis, însă unul în care s-a modificat conținutul (s-a scris) trebuie deconectat (închis) folosind instrucțiunea CLOSE, altfel la terminarea rulării programului poate rămâne blocat și cu conținut inaccesibil. Scrierea efectuându-se printr-o memorie tampon, dacă aceasta nu a fost golită în mod expres (prin efectul instrucțiunii CLOSE), atunci nu e sigur că s-au transferat toate înregistrările, iar prin terminarea rulării programului nu va mi fi cine să gestioneze conținutul memoriei tampon (rezultând umplerea memoriei calculatorului cu date inutile).

Exemplu:	Explicații:
`OPEN(3,FILE="TEST.DAT",STATUS="OLD")` `READ(3,*)n,m` `CLOSE(3)`	Deschiderea fișierului existent TEST.DAT, asociat cu unitatea logică numărul 3, urmată de citirea valorilor variabilelor N și M din acest fișier și deconectarea unității logice (închiderea fișierului).
`DIMENSION A(10,10)` `  CHARACTER(12) nume` `PRINT *,"nume fisier de date: "` `3  READ(*,"(A)")nume` `OPEN(1,FILE=nume,STATUS="OLD",ERR=9)` `! se citeste numărul de linii pt. A` `  READ(1,*) nl` `! se citeste numărul de coloane pt. A` `  READ(1,*) nc` `! citire elementele, pe câte un rând:` `  DO i=1,nl` `  READ(1,*)(A(i,j),j=1,nc)` `  ENDDO` `  …` `OPEN(2,FILE="R.DAT",STATUS="UNKNOWN")` `! scriere titlu în fisierul R.DAT` `  WRITE(2,*)"tabloul A:"` `! scrierea elementelor, câte un rând:` `  DO i=1,nl` `  WRITE(2,*)(A(i,j),",",j=1,nc)` `  ENDDO` `CLOSE(2)` `  …` `STOP` `9  PRINT *,"nu s-a gasit fisierul!"` `GOTO 3` `  …`	Declararea unui tablou cu 10x10=100 poziții și a entității NUME cu 12 poziții (caractere). Atenție la CHARACTER, litera C în prima coloană ar marca comentariu! Citirea denumirii fișierului în variabila NUME. Deschiderea fișierului (existent) prin asociere cu unitatea logică numărul 1, din care se vor citi datele (a se vedea comentariile din coloana alăturată marcate cu semnul de exclamare). Dacă fișierul nu există, se va sări la instrucțiunea cu eticheta 9. Utilizarea unui ciclu implicit (J=1,NC) în cadrul unui ciclu explicit (I=1,NL), pentru citirea elementelor dintr-o matrice. Deschiderea fișierului R.DAT prin asociere cu unitatea logică numărul 2 (dacă fișierul nu există, se va crea, iar dacă există, se va deschide și se va suprascrie conținutul). Scrierea elementelor din matricea A, pe rânduri, cu virgulă după fiecare element. Închiderea fișierului R.DAT (deconectarea unității logice numărul 2). Unitatea logică numărul 1 nu a suferit modificări și va fi automat deconectată în momentul terminării rulării programului. În cazul negăsirii fișierului cu date, după afișarea mesajului specificat se va încerca recitirea denumirii acestuia (sărind la instrucțiunea cu eticheta 3).

Există și alte instrucțiuni suplimentare pentru tratarea, cum ar fi:

Sintaxa instrucțiunii:	Explicații:
`BACKSPACE([UNIT=]`*u*`[,ERR=`*label*`][,IOSTAT=`*var*`])` sau `BACKSPACE` *u*	În cazul accesului secvențial repoziționează fișierul pe înregistrarea anterioară.
`ENDFILE([UNIT=]`*u*`[,ERR=`*label*`][,IOSTAT=`*var*`])` sau `ENDFILE` *u*	Scrierea marcajului de sfârșit în fișierul asociat cu unitatea logică *u* (accesat printr-o instrucțiune `OPEN` prealabilă) în poziția curentă (trunchiază restul).
REWIND(`[UNIT=]`*u*`[,ERR=`*e*`])` sau `REWIND` *u*	Repoziționarea la începutul fișierului asociat (în prealabil prin instrucțiunea `OPEN`) cu unitatea logică numărul *u*.

Unități de program

Toate programele scrise în limbajul Fortran pot fi organizate în unități de program. O unitate de program este considerată o secvență de specificații și de instrucțiuni care poate fi scrisă într-un fișier sursă separat și poate fi compilat. Desigur, într-un fișier sursă pot fi scrise și mai multe unități de program, iar ordinea în care se succed nu are importanță, cu excepția modulelor (modulele trebuie să fie compilate înainte de unitățile de program care le utilizează, de aceea trebuie să apară înaintea lor în fișierul sursă, astfel încât în momentul în care se ajunge la compilarea unității care utilizează modulul, modulul să fie deja compilat).
De regulă fiecare unitate de program începe prin definire și se termină cu marcajul END urmat de specificarea tipului corespunzător unității de program. Numele unităților de program trebuie să fie unic și trebuie să respecte criteriile referitoare la numele simbolice (nu poate conține spații sau caractere nepermise, trebuie să înceapă cu o literă și nu poate avea o lungime mai mare de 32 de caractere, iar în cazul variantelor mai vechi ale limbajului Fortran se recomandă limitarea la 6 caractere).
Nu toate unitățile de program pot conține instrucțiuni executabile, există unități de program care pot conține doar specificații referitoare la entitățile utilizate de către alte unități de program. Există 4 tipuri de unități de program în Fortran:

- Program principal (obligatoriu în orice aplicație și poate conține instrucțiuni executabile),
- Proceduri externe (subprograme, funcții – pot conține instrucțiuni executabile),
- Module (nu pot conține instrucțiuni executabile, doar eventual în proceduri de modul incluse)
- Blocuri de date (nu pot conține instrucțiuni executabile, doar specificații).

Fiecare aplicație (program Fortran) trebuie să conțină un singur program principal (acesta va fi lansat la începutul rulării). Procedurile externe sunt subprograme și funcții care sunt definite separat. Există mai multe tipuri de proceduri, însă doar cele externe sunt considerate unități de program. Modulele sunt unități precompilate (trebuie să fie compilate înaintea unităților de program care le folosesc), conținând de regulă doar specificații referitoare la entități. Blocurile de date conțin specificații referitoare la entități și pot conține și inițializări de date. Diferența blocurilor de date față de fișierele de date constă în conținutul specificațiilor care necesită compilare (fișierele de date conțin doar valori, fără specificații în Fortran, deci nu necesită compilare). Programul principal și procedurile pot conține instrucțiuni executabile, dar blocurile de date și modulele pot conține doar specificații referitoare la entități (cu mențiunea, că și modulele pot conține instrucțiuni executabile dacă aceste instrucțiuni fac parte din proceduri de modul).

<u>Program principal</u>

Nu poate lipsi din nici o aplicație și nici o aplicație nu poate conține mai mult de 1 program principal. Este singura unitate de program la care specificarea tipului unității de program este opțională. Un program principal nu se poate autoreferi (direct sau indirect). Sintaxa unui program principal este (cu comentarii):

[`PROGRAM` *nume*] Dacă se folosește cuvântul cheie `PROGRAM`, atunci trebuie specificat și *nume* (care trebuie să fi unic și va fi considerat global – adică, se va

	”vedea” din toate unitățile de program). Fără cuvântul cheie `PROGRAM` nu se poate specifica *nume*, în asemenea situații se va considera implicit denumirea MAIN pentru programul principal. Orice unitate de program care începe cu specificații sau comentarii (sau directive de compilare prin cuvântul cheie `OPTIONS`) va fi considerată program principal.
[*specificații*]	Cuvintele cheie `INTENT`, `OPTIONAL`, `PUBLIC` și `PRIVATE` nu se pot utiliza în cadrul specificațiilor. Specificațiile referitoare la entități trebuie să preceadă instrucțiunile executabile în toate unitățile de program.
[*instrucțiuni executabile*]	Nu se pot utiliza cuvintele cheie `ENTRY` și `RETURN`.
[`CONTAINS` *proceduri interne*]	Pot fi definite mai multe proceduri interne (subprograme și funcții) succesiv.
`END` [`PROGRAM` [*nume*]]	Marcajul final trebuie să fie cel puțin cuvântul cheie `END`. Acesta poate fi urmat și de cuvântul cheie `PROGRAM`, însă *nume* poate fi precizat doar dacă s-a definit explicit la începutul unității de program.

Exemplu:	Explicații:
`END`	Un program principal gol cu numele implicit MAIN.
`PRINT *,”Salut!”` `END PROGRAM`	Program principal care va afișa doar textul `Salut!` pe monitor.
`PROGRAM test` `INTEGER C, D` `  …` `CALL sub1` `  …` `CONTAINS` `  SUBROUTINE sub1` `  …` `  PRINT *, func(X,Y)` `  …` `  END SUBROUTINE sub1` `  FUNCTION func(X,Y)` `  …` `  END FUNCTION func` `END PROGRAM test`	Program principal botezat TEST, care va apela subprogramul SUB1 (pe care îl conține ca procedură internă, alături de funcția FUNC). Apelul subprogramului numit SUB1. Marcarea procedurilor conținute. Definirea subprogramului SUB1 ca procedură internă. Tipărirea rezultatului funcției FUNC pentru valorile curente ale argumentelor X și Y. Marcajul de sfârșit pentru procedura internă SUB1. Definirea funcției FUNC ca procedură internă. Marcarea sfârșitului procedurii interne FUNC. Marcajul de sfârșit pentru programul principal TEST (cu specificarea numelui, deși era suficent doar `END`).

<u>Proceduri</u>

Pot să fie subprograme sau funcții, însă doar cele definite ca și proceduri externe reprezintă unități de program. Procedurile se pot autoreferi (direct sau indirect) și au interfețe implicite (dar se pot specifica și explicit interfețele, prin blocuri de interfețe). Tipurile de proceduri existente in Fortran sunt următoarele:

- Proceduri externe (subprograme și funcții care nu fac parte din altă unitate de program);
- Proceduri interne (subprograme și funcții care fac parte dintr-un program principal sau dintr-o altă procedură);
- Proceduri de modul (proceduri definite în cadrul unor module);
- Proceduri intrinseci (subprograme și funcții predefinite în limbajul Fortran);
- Proceduri DUMMY (de regulă un argument ”DUMMY” specificat ca procedură, sau figurând ca referință de procedură);
- Funcție instrucțiune (o procedură de calcul definită printr-o singură instrucțiune, care se poate referi prin numele ei simbolic).

Toate procedurile beneficiază de interfață, aceasta fiind de regulă definită în mod implicit. O interfață de procedură se referă la proprietățile unei proceduri cu care interacționează, sau la unitatea de program apelantă. Interfața poate fi definită și în mod explicit, prin blocuri de interfață. Cu excepția blocurilor de date, toate unitățile de program pot conține blocuri de interfețe.
Procedurile externe pot conține proceduri interne, însă procedurile interne și de modul nu pot conține la rândul lor proceduri interne. Procedurile interne sunt în secțiunea precedată de cuvântul cheie `CONTAINS` și au acces la toate entitățile din unitatea de program care le conține (HOST). Numele lor nu poate fi folosit ca argument către o altă procedură (există variante de Fortran care permit acest lucru, de exemplu Intel Visual Fortran) și nu pot conține puncte de intrare distincte (prin specificații `ENTRY`).
Subprogramele se invocă prin instrucțiunea `CALL` sau printr-o instrucțiune asignată definită. Subprog–ramele nu returnează o valoare în mod direct, dar se pot transfera valori prin argumente sau variabile cunoscute între unitatea de program apelantă și subprogram. Revenirea dintr-un subprogram în unitatea de program apelantă se face prin instrucțiunea `RETURN`, a cărei sintaxă este următoarea:

`RETURN` [*număr*] — Cuvântul cheie `RETURN` poate fi urmat de un *număr* sau de o expresie numerică a căror valoare trebuie să fie de tip `INTEGER` (semnificând poziția rezervată în lista argumentelor prin care se va reveni în unitatea de program apelantă).

Funcțiile se invocă prin nume sau printr-un operator definit. În mod normal ele returnează o singură valoare ca rezultat (prin numele funcției) în urma evaluării. Revenirea dintr-o funcție se va face în mod implicit în unitatea de program în care s-a folosit referirea la funcție, însă se poate utiliza și instrucțiunea `RETURN` (prezentată mai sus) pentru a specifica diferite puncte de revenire față de finalul funcției.

Intrarea într-o procedură (prin instrucțiunea `CALL` în cazul subprogramelor, sau prin nume în cazul unei funcții) se poate face și într-o altă poziție decât începutul procedurii, folosind specificația `ENTRY` (punct de intrare), a cărei sintaxă este:

`ENTRY` *nume* [`(` *argumente* `)`] — Declarația se poate specifica în conținutul procedurilor externe (nu poate fi utilizată în proceduri interne), făcând parte din corpul acesteia, iar *nume* este denumirea punctului de intrare în procedură (diferit de numele procedurii) prin care se va invoca acea parte din procedură. În asemenea situații instrucțiunile care preced specificația `ENTRY` în definirea procedurii vor fi ignorate în momentul activării acestuia (executarea instrucțiunilor din procedură va începe de la prima instrucțiune care urmează după punctul de intrare specificat).

În general se recomandă evitarea utilizării punctelor de intrare în proceduri, pentru claritatea fișierelor sursă. Argumentele care se specifică la definirea unei proceduri (sau a unui punct de intrare într-o procedură externă) sunt considerate fictive, în sensul în care în momentul definirii procedurii valorile lor nu sunt cunoscute, ci doar tipul lor. Argumentele care se precizează la invocarea unei proceduri sunt considerate efective, deoarece pe lângă cunoașterea tipului lor, de regulă se cunosc și valorile lor efective. Ordinea și tipul argumentelor efective (utilizate la apel) trebuie să coincidă cu ordinea și tipul argumentelor fictive (utilizate la definirea procedurii), însă numele argumentelor fictive poate să difere de numele argumentelor efective.

La definirea procedurilor, în fața cuvântului cheie care specifică tipul procedurii, se pot specifica și câteva caracteristici, cum ar fi:

`ELEMENTAL` — Când la o parcurgere se dorește aplicarea procedurii doar pe un element dintr-un tablou.

`PURE` — Pentru a evita eventuale efecte colaterale (asupra valorii entităților utilizate). În cazul funcțiilor declarate `PURE` nu se vor putea utiliza opțiunile `INTENT` pentru argumente și pentru numele funcției (la subprograme nu există această restricție). În plus, o procedură declarată `PURE` va putea utiliza doar alte proceduri `PURE`.

`RECURSIVE` — Așa cum s-a menționat, recursia (autoreferirea) directă sau indirectă este permisă funcțiilor și subprogramelor. În cazul specificării acestei caracteristici, la definirea procedurii se poate

completa linia prin care se declară tipul procedurii (după lista cu argumentele fictive) cu RESULT(*nume_r*) pentru a specifica un nume diferit (*nume_r*) față de numele inițial al procedurii, acest nume diferit fiind utilizat pentru recursie.

MODULE — Pentru a specifica o procedură de modul (se poate utiliza doar în cadrul modulelor).

Subprograme

Pe lângă subprogramele intrinseci existente în limbajul Fortran, pot fi definite și alte subprograme, după necesități. Sintaxa definirii unui subprogram este următoare:

SUBROUTINE *nume* [(*argumente*)] — Înaintea cuvântului cheie SUBROUTINE se poate specifica și o caracteristică a procedurii (ELEMENTAL, PURE, RECURSIVE), iar argumentele sunt opționale (acestea se precizează doar pentru transfer de valori între unitatea de program apelantă și subprogram). Argumentele sunt considerate fictive, în sensul în care în momentul definirii subprogramului valorile lor nu sunt cunoscute, ci doar tipul lor. Se pot utiliza și marcaje pentru poziții rezervate ca argumente (vedeți exemplele cu RETURN).

[*specificații*] — În toate unitățile de program specificațiile referitoare la entități trebuie să preceadă instrucțiunile executabile.

[*instrucțiuni executabile*] — Pot conține specificații ENTRY (pentru definirea unor puncte de intrare) și instrucțiuni RETURN (pentru revenirea în unitatea de program din care s-a apelat subprogramul).

[CONTAINS *proceduri interne*] — Pot fi definite mai multe proceduri interne (subprograme și funcții) succesiv, însă doar în cazul unui subprogram definit ca procedură externă.
În cazul procedurilor interne această secțiune nu poate apărea.

END [SUBROUTINE [*nume*]] — Marcajul final trebuie să fie cel puțin cuvântul cheie END în cazul subprogramelor definite ca procedură externă. Acesta poate fi urmat și de cuvântul cheie SUBROUTINE, eventual și de *nume*.
În cazul procedurilor interne marcajul final trebuie să conțină cel puțin ambele cuvinte cheie END SUBROUTINE.

Apelarea unui subprogram se face prin instrucțiunea CALL, a cărei sintaxă este următoarea:

CALL *nume* [(*argumente*)] — Argumentele se precizează dacă ele există în definirea subprogramului. La apel aceste argumente sunt considerate efective, în sensul în care în momentul apelării subprogramului alături de tipul lor și valorile lor sunt de regulă cunoscute. Ordinea argumentelor efective (de la apelarea subprogramului) trebuie să corespundă cu ordinea argumentelor fictive (de la definirea subprogramului) ca tip, dar se pot folosi denumiri diferite.

Exemple:	Explicații:
! program principal CALL salut END PROGRAM *! subprogram* SUBROUTINE salut PRINT *,”Salut!” END SUBROUTINE salut	Program principal care va apela doar subprogramul SALUT definit ca procedură externă, iar subprogramul va afișa doar textul Salut! pe monitor. În exemplul alăturat rularea se va opri în subprogram. De asemenea, se poate observa că la marcajul sfârșitului programului principal (END PROGRAM) nu s-a putut preciza numele acestuia, întrucât nu a fost definit.
! program principal CALL salut	Exemplul anterior modificat prin inserarea instrucțiunii RETURN în definirea subprogramului

```
END

! subprogram
SUBROUTINE salut
 PRINT *,"Salut!"
 RETURN
END
```

SALUT. În acest caz, după apelarea subprogramului și afișarea textului `Salut!` pe monitor, se va reveni în programul principal, iar rularea se va opri la sfârșitul programului principal.
De asemenea, se poate observa că este suficient marcajul `END` pentru subprogramul definit ca procedură externă.

```
! subprogram cu punct de intrare
SUBROUTINE semn
PRINT *,"valoare pozitiva sau nula"
 RETURN
ENTRY negativ
PRINT *,"valoare strict negativa"
 RETURN
END

! unitate de program apelantă
…
IF(N < 0) THEN
   CALL negativ
ELSE
   CALL semn
ENDIF
…
END
```

Exemplu cu un punct de intrare numit NEGATIV în subprogramul denumit SEMN.
Dacă valoarea scalarului N este negativă, atunci se va apela NEGATIV, ceea ce nu este un subprogram, ci un punct de intrare în subprogramul SEMN. Ca efect, se vor ignora instrucțiunile executabile care preced specificația punctului de intrare NEGATIV din subprogramul SEMN, tipărindu-se mesajul `valoare strict negativa` pe monitor, după care se revine în unitatea de program apelantă. Dacă valoarea scalarului N nu este negativă, atunci se va apela subprogramul SEMN, fiind executate instrucțiunile până la primul `RETURN` întâlnit (se va afișa mesajul `valoare pozitiva` după care se revine în unitatea de program apelantă). Desigur, specificarea unui punct de intrare condiționează doar începutul de la care se execută instrucțiunile, nu și finalul (dacă nu era specificat `RETURN` înainte de punctul de intrare NEGATIV, la apelul subprogramului SEMN după afișarea mesajului `valoare pozitiva` s-ar fi afișat și mesajul `valoare strict negativa`).

```
! unitate de program apelantă
     …
     CALL verif(A,B,*10,*20,C)
     PRINT *,"valoare negativa"
     GOTO 30
10   PRINT *,"valoare nula"
     GOTO 30
20   PRINT *,"valoare pozitiva"
30   CONTINUE
     …
     END
! subprogram ca procedură externă
     SUBROUTINE verif(X,Y,*,*,Z)
     …
     IF(X*Y-Z) 50,54,55
50   RETURN
54   RETURN 1
55   RETURN 2
     END
```

În unitatea de program din care se apelează subprogramul VERIF, în lista argumentelor efective apar entitățile scalare de tip `REAL` (datorită regulii implicite) A, B, apoi pozițiile rezervate (prin marcajul `*`) cu etichetele 10 și 20, respectiv entitatea scalară C (tot de tip `REAL` datorită regulii implicite). Acestea argumente corespund ca ordine (și tip) cu cele fictive care s-au specificat la definirea subprogramului: X și Y (de tip `REAL` implicit), apoi 2 poziții rezervate (fiecare marcată prin `*`) și Z (de tip `REAL` implicit). În momentul apelării subprogramului VERIF, valoarea din A se va transfera prin X, valoare din B prin Y, iar valoarea din C prin Z în subprogram. Când se va ajunge în subprogram la testarea valorii rezultate din expresia aritmetică, se va alege eticheta corespunzătoare din lista (în cazul unui rezultat strict negativ se va sări la eticheta 50, în cazul unui rezultat nul la eticheta 54, iar în cazul unui rezultat strict pozitiv la eticheta 55). Dacă se sare la instrucțiunea cu eticheta 50, revenirea în unitatea apelantă se va face la argumentele efective de la instrucțiunea `CALL` (valoarea din A se va actualiza din valoarea lui X, B din Y, iar C din Z) și se va executa prima instrucțiune care urmează (afișarea textului

	valoare negativa) după care se va sări la instrucțiunea cu eticheta 30. Deci, transferul de valori se va realiza și dinspre subprogram către unitatea de program apelantă (nefiind specificate prin INTENT alte opțiuni), iar RETURN înseamnă revenire ”normală”. În cazul în care din condiția aritmetică din subprogram rezultă saltul la instrucțiunea cu eticheta 54, revenirea în unitatea apelantă se va realiza prin activarea primei poziții rezervate din lista argumentelor fictive, ceea ce în lista argumentelor efective corespunde cu *10, în consecință prima instrucțiune executată după revenire va fi cea cu eticheta 10 (se va afișa textul valoare nula după care se va sări la instrucțiunea cu eticheta 30). Deci RETURN 1 înseamnă revenire prin prima poziție rezervată. În cazul în care din condiția aritmetică din subprogram rezultă saltul la instrucțiunea cu eticheta 55, revenirea în unitatea apelantă se va realiza prin activarea celei de a doua poziții rezervate (datorită valorii 2 specificate la RETURN) din lista argumentelor fictive, ceea ce în lista argumentelor efective corespunde cu *20, în consecință prima instrucțiune executată după revenire va fi cea cu eticheta 20 (se va afișa textul valoare pozitiva după care se va continua cu instrucțiunea cu eticheta 30). Deci RETURN 2 înseamnă revenire prin a doua poziție rezervată.

Funcţii definite de utilizator

Pe lângă funcțiile intrinseci existente în limbajul Fortran, se pot defini și funcții diferite. Există mai multe categorii de funcții: definite ca proceduri externe (unități de program), definite ca proceduri interne sau de modul (conținute de către alte unități de program), definite ca instrucțiune (într-o singură expresie de specificare). Utilizarea funcțiilor se face prin specificare denumirii și a argumentelor (dacă o funcție nu are argumente, atunci numele va fi urmat de paranteze goale) în cadrul unor instrucțiuni. Se pot transfera valori către funcții prin argumente (ca în cazul subprogramelor, cu diferența că față de subprograme, în cazul funcțiilor parantezele care încadrează argumentele sunt obligatorii, chiar dacă nu sunt argumente), însă funcțiile vor returna un rezultat prin numele lor, nu prin argumente! Având în vedere acest aspect, în definirea unei funcții trebuie să apară obligatoriu o expresie prin care se calculează rezultatul funcției.

La definirea unei funcții, pe lângă cuvintele cheie care specifică caracteristici (ELEMENTAL, PURE, RECURSIVE, MODULE) se poate specifica și tipul funcției (în cazul celor definite ca proceduri externe se pot utiliza doar tipurile intrinseci). Sintaxa definirii unei funcții ca procedură este următoarea:

[*tip*] FUNCTION *nume* ([*argumente*])

Înaintea cuvântului cheie FUNCTION se poate specifica și o caracteristică a funcției (ELEMENTAL, PURE, RECURSIVE) și un *tip* (INTEGER, REAL, COMPLEX, LOGICAL, CHARACTER, BYTE), iar precizarea argumentelor este opțională (acestea se precizează doar dacă se dorește transfer de valori între unitatea de program apelantă și funcție), însă parantezele delimitatoare ale argumentelor sunt obligatorii . Argumentele sunt considerate fictive, în sensul în care în momentul definirii funcției valorile lor efective nu sunt cunoscute, ci doar tipul lor.

În cazul autoreferirii (RECURSIVE) se impune completare

definirii prin `RESULT(`*nume_r*`)` la capătul acestui rând, unde *nume_r* este entitatea prin care se va folosi rezultatul în funcție.

[*specificații*] — În toate unitățile de program specificațiile referitoare la entități trebuie să preceadă instrucțiunile executabile.

[*instrucțiuni executabile*] — Pot conține specificații `ENTRY` (pentru definirea unor puncte de intrare) și instrucțiuni `RETURN` (pentru revenirea în unitatea de program din care s-a apelat funcția).
Atenție: trebuie să conțină și o expresie din care să rezulte valoarea funcției!

[`CONTAINS`
proceduri interne] — Pot fi definite mai multe proceduri interne (subprograme și funcții) succesiv, însă doar în cazul unei funcții definite ca procedură externă.
În cazul procedurilor interne sau de modul această secțiune nu poate apărea.

`END` [`FUNCTION` [*nume*]] — Marcajul final trebuie să fie cel puțin cuvântul cheie `END` în cazul funcțiilor definite ca procedură externă. Acesta poate fi urmat și de cuvântul cheie `FUNCTION`, eventual și de *nume*.
În cazul procedurilor interne marcajul final trebuie să conțină cel puțin ambele cuvinte cheie `END FUNCTION`.

Sintaxa definirii unei funcții instrucțiune este următoarea:

[*tip*] *nume*([*argumente*])=*expresie* — Înaintea cuvântului cheie `FUNCTION` se poate specifica și un *tip* (intrinsec sau derivat), iar precizarea argumentelor este opțională (acestea se precizează doar dacă se dorește transfer de valori între unitatea de program apelantă și funcție), însă parantezele în care ar fi argumente sunt obligatorii . Argumentele sunt considerate fictive, în sensul în care în momentul definirii funcției valorile lor efective nu sunt cunoscute, ci doar tipul lor.

Apelarea unei funcții se face prin numele ei și precizând argumentele efective (dacă există) în cadrul unei instrucțiuni, sub forma:

… *nume* ([*argumente*]) — Argumentele se precizează dacă ele există în definirea funcției (dacă nu sunt, atunci parantezele vor fi goale). La apel aceste argumente sunt considerate efective, în sensul în care în momentul apelării funcției, alături de tipul lor și valorile lor sunt de regulă cunoscute. Ordinea argumentelor efective (de la apelarea funcției) trebuie să corespundă cu ordinea argumentelor fictive (de la definirea funcției) ca tip, dar se pot folosi denumiri diferite.

Exemple:	Explicații:
`! program principal` `  INTEGER pe2` `10 PRINT *,"numar: "` `  READ *,i` `  IF(i==0) STOP` `  PRINT *,pe2(i)` `  GOTO 10` `END` `! definirea functiei pe2` `INTEGER FUNCTION pe2(nr)` `pe2=nr/2` `END`	Program principal care va apela funcția PE2 definită ca procedură externă, și va afișa rezultatul acestei funcții pentru valoarea argumentului efectiv *i* (pe monitor). Programul se va opri doar dacă valoarea citită pentru *i* este nulă. În momentul invocării funcției (pentru tipărirea rezultatului) sa va transfera valoarea lui *i* în NR (din definirea funcției), iar rezultatul returnat se va obține prin numele funcției PE2. De asemenea, se poate observa că este suficient marcajul `END` pentru funcția definită ca procedură externă.

`` ! program principal INTEGER pe2 10 PRINT *,"numar: " READ *,i IF(i==0) STOP PRINT *,pe2(i) GOTO 10 CONTAINS ! definirea functiei pe2 FUNCTION pe2(nr) INTEGER pe2 pe2=nr/2 END FUNCTION pe2 END ```	Exemplul anterior modificat prin transformarea definirii funcției în procedură internă. Deși se putea definii funcția cu precizarea tipului `INTEGER` ca în cazul precedent, s-a optat pentru specificarea separată a tipului. În acest caz marcajul de final al funcției trebuie să conțină obligatoriu și cuvântul cheie `FUNCTION` (alături de `END`), menționarea numelui funcției fiind opțională acolo.
`` ! program principal INTEGER pe2,nr ! definirea functiei pe2 pe2(nr)=nr/2 ! instructiuni executabile 10 PRINT *,"numar: " READ *,i IF(i==0) STOP PRINT *,pe2(i) GOTO 10 END PROGRAM ```	Exemplul anterior modificat prin transformarea definirii funcției în instrucțiune. Se poate observa că la această variantă numele funcției este urmată de argumentul fictiv în linia de definire. Definirea funcției instrucțiune nu este instrucțiune executabilă, așa că trebuie să apară în zona specificațiilor.
`` PROGRAM factorial INTEGER f,i PRINT *,"i: " READ *,i PRINT *,"factorial ",i,": ",f(i) END ! definirea funcției recursive RECURSIVE FUNCTION f(i) RESULT(fa) INTEGER f, fa IF(i==1) THEN fa=1 ELSE fa=i*f(i-1) ENDIF END ```	Exemplu "clasic" de funcție definită ca procedură care se autoreferă (este recursivă), pentru calculul valorii factoriale al unui număr. Se observă că în acest caz fiind specificat `RECURSIVE`, și specificarea `RESULT(`*nume_r*`)` este obligatorie, *nume_r* fiind denumirea utilizată a funcției pentru autoreferire (recursie) în cadrul descrierii. Deși funcția este denumită F, în definirea ei se va folosi numele FA (cel specificat prin `RESULT`) pentru calculul rezultatului funcției.
`` PROGRAM functie_tablou PRINT *,'a,b,c: ' READ *,a,b,c PRINT *, func(a,b,c) CONTAINS ! procedura interna FUNCTION func(x1,x2,x3) DIMENSION func(3) func(1)=x1 func(2)=x2 func(3)=x3 END FUNCTION END ```	Scurt exemplu cu o funcție definită ca procedură internă și ca tablou. Deși în mod normal o funcție returnează un singur rezultat (o singură valoare scalară), în cazul definirii ca procedură internă se poate crea și o funcție tablou (care va returna un rezultat sub formă de tablou). În momentul apelării funcției se vor transfera argumentele în ordinea specificată (X1 va corespunde cu valoarea din A, X2 cu B, X3 cu *C*), iar rezultatul se va obține prin numele funcției (în cazul de față 3 valori distincte). Pentru fiecare poziție din funcția FUNC – tablou cu 3 poziții: FUNC(1), FUNC(2) ȘI FUNC(3) – se vor calcula rezultatele. Primul element din tabloul FUNC va primi valoarea din X1, al doilea valoarea din X2, iar al treilea valoarea din X3. Astfel, în momentul tipăririi rezultatului FUNC(A,B,C) se vor

	afișării 3 valori distincte consecutive pe monitor.

Module

Sunt unități de program care conțin de regulă specificații și definiții care se pot face accesibile altor unități de program. Pot conține și interfețe explicite (prin blocuri de interfețe) pentru o procedură externă sau DUMMY. Definirea unui modul se face în felul următor:

MODULE *nume*	Atribuirea unui *nume* este obligatoriu, acesta este global și unic!
[*specificații*]	Nu poate conține: AUTOMATIC, ENTRY, FORMAT, INTENT, OPTIONAL și nici funcții definite sau intrinseci.
[CONTAINS	Instrucțiunile executabile pot să apară doar în cadrul procedurilor de
proceduri de modul]	modul (interne).
END [MODULE [*nume*]]	Este suficient dacă se specifică doar cuvântul cheie END (dacă nu s-a botezat modulul, atunci nu există *nume* de specificat).

Utilizarea unui modul se poate face doar după ce modulul a fost compilat, specificând utilizarea acestuia în unitatea de program vizată prin:

USE *nume* (*nume* fiind numele cu care s-a definit modulul).

Exemple:	Explicații:
MODULE prim INTEGER,PARAMETER :: A,B REAL E22(5,5) END *! unitatile care-l utilizeaza* SUBROUTINE P21 USE prim … END FUNCTION FU33(A,X) USE prim … END	Un modul definit sub numele PRIM, conținând doar câteva specificații de date. În cazul în care se scrie în același fișier sursă cu unitatea de program care o va utiliza (în exemplu subprogramul P21 și funcția FU33), modulul trebuie să fie înaintea unității de program, ca la compilarea conținutului fișierului sursă, în momentul în care se ajunge la USE PRIM, modulul să fi fost deja compilat!
MODULE cal_M TYPE element PRIVATE INTEGER C,D END TYPE … INTERFACE FUNCTION calcul(R) REAL :: calcul REAL,INTENT(IN) :: R(:) END FUCTION END INTERFACE END MODULE cal_M	Un modul denumit CAL_M în care s-a definit un tip derivat denumit ELEMENT (implicit PUBLIC, deci vizibil din toate unitățile de program) cu componentele C și D declarate PRIVATE (doar din modul se pot vedea). După specificarea tipului derivat urmează un bloc de interfață pentru funcția CALCUL, în care argumentul R va fi un vector utilizat doar pentru intrare (transfer de valori către funcția numită CALCUL), atât funcția CALCUL (cu valoarea ce va rezulta dintr-o expresie specificată în definirea funcției undeva în altă parte) cât și R fiind de tip REAL.

Un exemplu mai vechi și mai complex, (adaptat după https://www.star.le.ac.uk/~cgp/f90course/f90.html#tth_sEc6) cu un modul care ar putea fi utilizat pentru simularea funcționării unei ferestre de consolă (VT100 sau fereastră X-TERM) dirijat prin secvențe ESC (ASCII), similar cu ANSI.SYS din DOS, conținând și proceduri de modul:

```
MODULE vt_mod
  IMPLICIT NONE
```

```
! specificarea codului pentru <ESC> ca valoare constanta numita ESC
  CHARACTER(1),PARAMETER :: esc=ACHAR(27)
! initializare variabile pentru 80 de coloane si 24 de randuri pe ecran
  INTEGER,SAVE :: nr_c=80,nr_r=24
  CONTAINS
! va sterge afisajul mutand cursorul la stanga sus
  SUBROUTINE sterge
   CALL afis(esc//"[H"//esc//"[2J")
  END SUBROUTINE sterge
! seteaza noua latime la 80 sau 132 coloane
  SUBROUTINE latime(col)
   INTEGER, INTENT(IN) :: col
   IF (col>80) THEN
   ! comutare pe 132 de coloane
      CALL afis(esc//"[?3h")
      nr_c=132
   ELSE
   ! comutare pe 80 de coloane
      CALL afis(esc//"[?3l")
      nr_c=80
   ENDIF
  END SUBROUTINE latime
! numarul actual de coloane
  SUBROUTINE nr_col(col)
   INTEGER,INTENT(OUT) :: col
   col=nr_c
  END SUBROUTINE nr_col
! doar pentru uzul intern al procedurilor de modul
  SUBROUTINE afis(sir)
   CHARACTER,INTENT(IN) :: sir
   WRITE(*,"(1X,A)",ADVANCE="NO")sir
  END SUBROUTINE afis
END MODULE vt_mod
```

Pentru a utiliza acest modul, se pot folosi următoarele variante de specificații (exempe):

`USE vt_mod`	Se va folosi întregul conținut al modulului.
`USE vt_mod,ONLY:sterge`	Se va folosi doar procedura de modul STERGE dintre proceduri.
`USE vt_mod,coloane=>nr_col`	Utilizarea întreg modulului, dar înlocuind temporar numele procedurii de modul NR_COL cu numele nou COLOANE.

Blocuri de date

Aceste unități de program au rostul de oferi posibilitatea inițializării entităților din blocuri comune (zone de memorie partajate), dar după Fortran77 sunt considerate depășite, întrucât specificația `COMMON` a fost eliminată începând cu Fortran 90. Conțin specificații referitoare la entități, eventual cu inițializarea unor date (nu și în caz de `POINTER` și `TARGET`), nu pot conține instrucțiuni executabile. Sintaxa definirii unui bloc de date este următoarea:

`BLOCK DATA` [*nume*]	Atribuirea unui *nume* este opțional, doar pentru claritatea fișierelor sursă. Dacă se definesc mai multe blocuri, doar una poate fi fără nume.
[*specificații*]	Poate conține: `COMMON` (nu și începând cu Fortran 90, deși compilatorul G95 permite), `INTRINSIC`, `STATIC`, `USE` (numai pentru constante botezate), `DATA` (pentru inițializări de date), `PARAMETER` (pentru

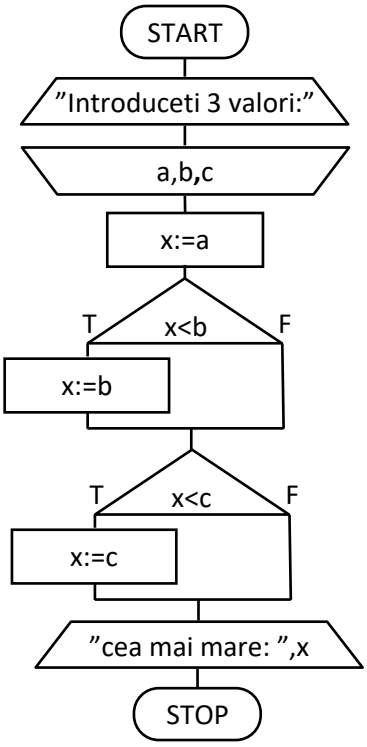
	constante), TARGET și POINTER (dar fără inițializări), DIMENSION (pentru tablouri), *tip* (cuvinte cheie pentru tipurile intrinseci de date), *tip derivat* (definiții și tipuri definite de către utilizator), RECORD și structură RECORD (de înregistrări), EQUIVALENCE, IMPLICIT, SAVE.
END [BLOCK DATA [*nume*]]	Este suficient dacă se specifică doar cuvântul cheie END (dacă nu s-a botezat blocul de date, atunci nu există *nume* de specificat).

Exemplu:	Explicații:
! program principal CHARACTER(6) Actor COMMON /zona1/a,b,c,d,Actor INTEGER :: s1=2 PRINT *,"s1:",s1 PRINT *, a,b,c,d PRINT 2,Actor … 2 format("Actor: ",A) … END	Începând cu Fortran 90 specificația COMMON a fost eliminată (se utiliza pentru desemnarea prin nume și componență a unei zone de memorie adresabilă din oricare unitate de program, prin specificarea numelui blocului comun). Sintaxa specificării unui bloc comun este: COMMON /*nume_bloc*/*listă_componente*[[,]…]
! blocul pt initializare valori BLOCK DATA DIMENSION x(4) COMMON /zona1/x,nume DATA x/3*1.,5/ CHARACTER(6) :: nume="Adrian" END	Definirea unui bloc de date prin care se specifică și se inițializează unele date. Prin specificația COMMON, făcând parte din blocul comun numit ZONA1, entitățile X (vector cu 4 poziții) și NUME vor ocupa aceași zonă de memorie cu A, B, C, D și ACTOR (având ca și conținut șirul Adrian) cu condiția ca spațiul de memorare să fie identic pentru entitățile corespunzătoare.

Exerciții

Transcrierea unor scheme logice în Fortran

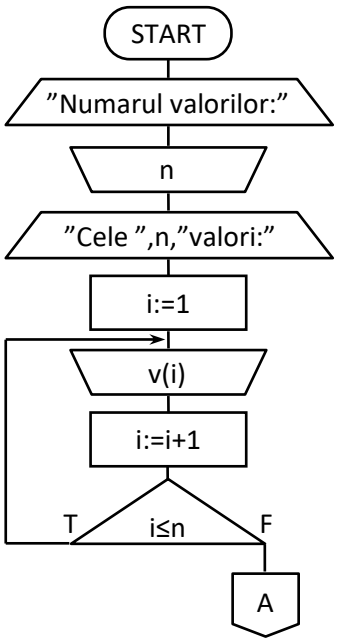
1. Cea mai mare valoare dintre a, b și c



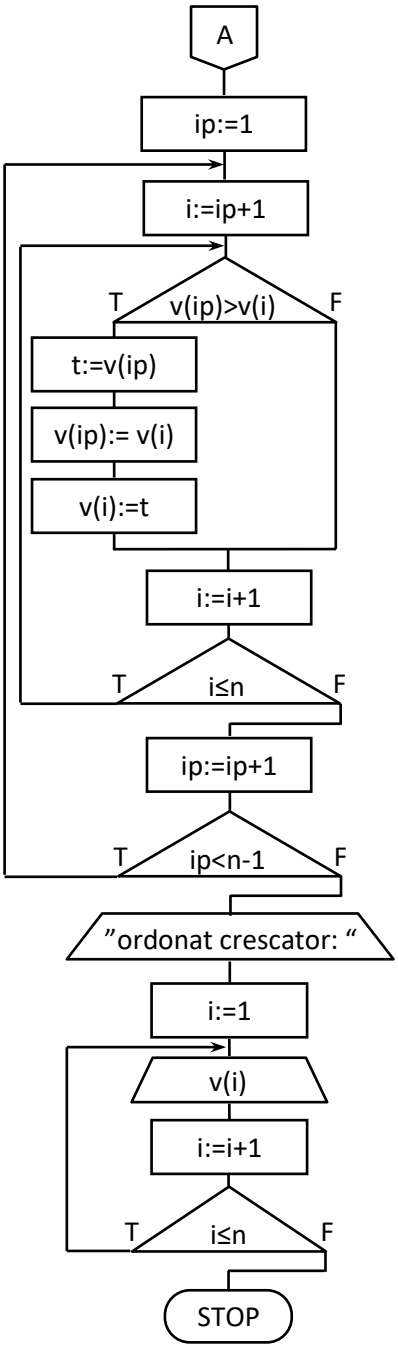

```
! varianta 1, tradusa elementar
      PRINT *,”Introduceti 3 valori:”

      READ *,a,b,c

      x=a


      IF(x<b) THEN
        x=b
      ENDIF


      IF(x<c) THEN
         x=c
      ENDIF

      PRINT *,”cea mai mare: ”,x
! marcajul final al unitatii de program
      END
```

```
! varianta 2, structurata
      PRINT *,”Introduceti 3 valori:”
      READ *,a,b,c
      x=a
      IF(x<b) x=b
      IF(x<c) x=c
      PRINT *,”cea mai mare: ”,x
      END
```

2. Ordonarea crescătoarea a valorilor
   a) Metoda pivotului

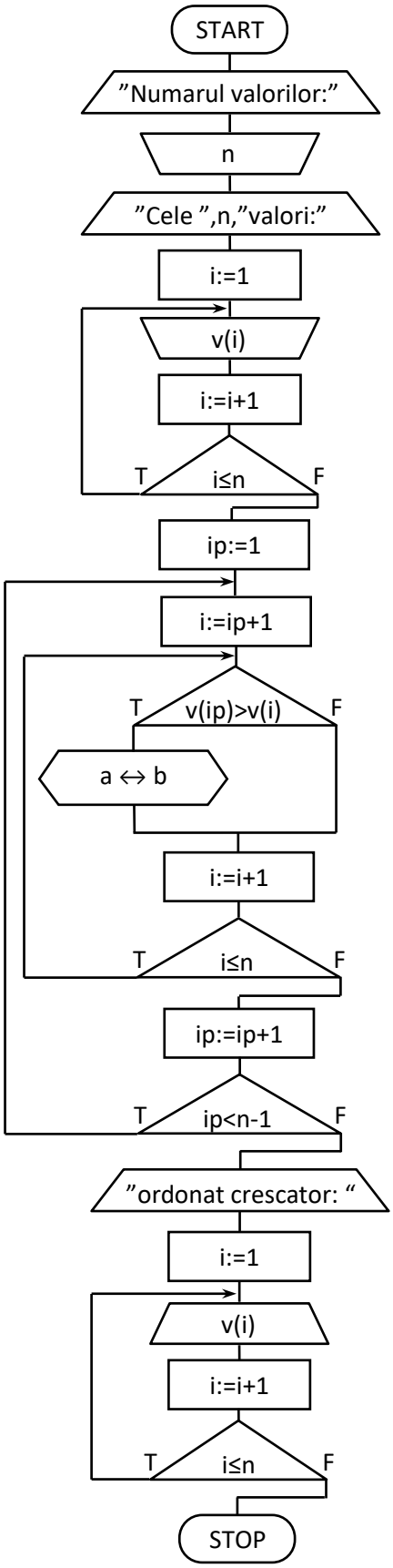



```
! varianta 1, cu alocare statica si salturi
      DIMENSION v(30)
! instructiunile din schema logica
      PRINT *,”Numarul valorilor:”

      READ *,n

      PRINT *,”Cele ”,n,”valori:”

      i=1


2     READ *,v(i)

      i=i+1


      IF(i<=n) GOTO 2

! continuare pe pagina urmatoare
```

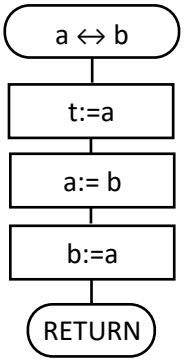



```
! continuare de pe pagina anterioara

      ip=1
21    i=ip+1

22    IF(v(ip)>v(i)) THEN
        t=v(ip)
        v(ip)=v(i)
        v(i)=t
      ENDIF
      i=i+1
      IF(i<=n) GOTO 22

      ip=ip+1

      IF(ip<=n-1) GOTO 21

      PRINT *,”ordonat crescator:”
      i=1
23    PRINT *,v(i)
      i=i+1
      IF(i<=n) GOTO 23
! marcajul final al unitatii de program
      END
-----------------------------------------------------------------
!varianta 2, cu alocare dinamica si cicluri
      ALLOCATABLE v(:)
      PRINT *,”Numarul valorilor:”
      READ *,n
      ALLOCATE(v(n))
      PRINT *,”Cele ”,n,”valori:”
      READ *,(v(i),i=1,n)
      DO ip=1,n-1
       DO i=ip+1,n
        IF(v(ip)>v(i)) THEN
            t=v(ip)
            v(ip)=v(i)
            v(i)=t
        ENDIF
       ENDDO
      ENDDO
      PRINT *,”ordonat crescator:”
      PRINT *,(v(i),i=1,n)
      DEALLOCATE(v)
      END
```

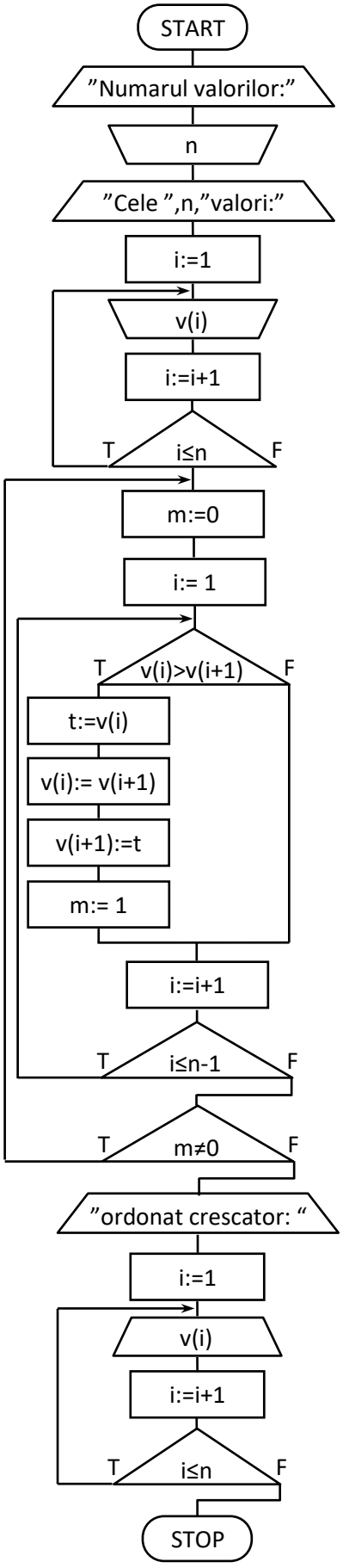
- variantă cu modul transcris ca procedură




```
! varianta 3, cu procedura interna
  DIMENSION v(30)
! instructiunile din schema logica
  PRINT *,”Numarul valorilor:”

  READ *,n

  PRINT *,”Cele ”,n,”valori:”



  READ *,(v(i),i=1,n)

  i=i+1



  DO ip=1,n-1

    DO i=ip,n

      IF(v(ip)>v(i)) THEN

        CALL schimb(v(ip),v(i))

      ENDIF


    ENDDO



  ENDDO

  PRINT *,”ordonat crescator:”



  PRINT *,(v(i),i=1,n)

  CONTAINS
! procedura interna ->
      SUBROUTINE schimb(a,b)
      t=a
      a=b
      b=t
      RETURN
      END SUBROUTINE

 END
```



b) Metoda marcajului




```
! varianta 1, cu alocare statica si salturi
      DIMENSION v(30)
! instructiunile din schema logica
      PRINT *,"Numarul valorilor:"
      READ *,n
      PRINT *,"Cele ",n,"valori:"
      i=1
2     READ *,v(i)
      i=i+1
      IF(i<=n) GOTO 2
23    m=0
      i=0
22    IF(v(ip)>v(i)) THEN
        t=v(ip)
        v(ip)=v(i)
        v(i)=t
        m=1
      ENDIF
      i=i+1
      IF(i<=n-1) GOTO 22
      IF(m/=n) GOTO 23
      PRINT *,"ordonat crescator:"
      i=1
23    PRINT *,v(i)
      i=i+1
      IF(i<=n) GOTO 23
! marcajul final al unitatii de program
      END
```

```
! varianta 2, cu alocare dinamica si cicluri
      ALLOCATABLE v(:)
      PRINT *,"Numarul valorilor:"
      READ *,n
      ALLOCATE(v(n))
      PRINT *,"Cele ",n,"valori:"
      READ *,(v(i),i=1,n)
23    m=0
      DO i=1,n-1
        IF(v(i)>v(i+1)) THEN
            t=v(i)
            v(i)=v(i+1)
            v(i+1)=t
        ENDIF
      ENDDO
      IF(m/=0) GOTO 23
      PRINT *,"ordonat crescator:"
      PRINT *,(v(i),i=1,n)
      DEALLOCATE(v)
      END
```

Exemple de fișiere sursă

1. Înmulțirea termenilor unei matrici cu o valoare scalară

```
      REAL  matrix(10,10)
      PRINT 1,"Numarul de linii si de coloane din matrice (max.10x10):"
      READ *,nl,nc
      PRINT 1,"Introduceti termenii din matrice:"
      READ *,((matrix(i,j),j=1,nc),i=1,nl)
      PRINT 1,"Valoarea scalara:"
      READ *,s
      matrix=matrix*s
      DO i=1,nl
        PRINT *,(matrix(i,j)," ",j=1,nc)
      ENDDO
1     FORMAT(A,$)
      END
```

Varianta cu citirea matricii din fișierul A.TXT:

```
      REAL  matrix(10,10)
      OPEN(1,FILE="A.TXT",STATUS="OLD",ERR=9)
      READ(1,*)nl,nc
      READ(1,*)((matrix(i,j),j=1,nc),i=1,nl)
      PRINT "(A,$)","Valoarea scalara:"
      READ *,s
      matrix=matrix*s
      DO i=1,nl
        PRINT *,(matrix(i,j)," ",j=1,nc)
      ENDDO
9     PRINT *,"fisierul A.TXT nu este gasit!"
      END
```

Structura fișierului A.TXT: *numărul_liniilor*, *numărul_coloanelor*
termenii_matricii

De exemplu, pentru matricea $\begin{vmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{vmatrix}$ se poate crea fișierul A.TXT cu următorul conținut:

```
3,3
1,1,1,2,2,2,3,3,3
```

sau:

```
3,3
1,1,1
2,2,2
3,3,3
```

2. Suma termenilor de pe o coloană aleasă dintr-o matrice

```
      REAL  matrix(10,10)
      CHARACTER r
      PRINT *,"Numarul de linii si de coloane din matrice (max.10x10):"
      READ *,nl,nc
      PRINT *,"Introduceti termenii din matrice:"
      READ *,((matrix(i,j),j=1,nc),i=1,nl)
1     PRINT *,"Numarul coloanei:"
      READ *,ncol
      sum=0.
      DO i=1,nl
        sum=sum+matrix(i,ncol)
      ENDDO
      PRINT *,"suma este: ",sum
      PRINT *,"Alegeti alta coloana? (D/N):"
      READ *,r
      IF(r=="D".OR.r=="d") GOTO 1
      END
```

Varianta cu alocare dinamică de memorie și cu citirea matricii din fișierul A.TXT:

```
      REAL,ALLOCATABLE :: matrix(:,:)
      CHARACTER r
      OPEN(1,FILE="A.TXT",STATUS="OLD",ERR=9)
      READ(1,*)nl,nc
      ALLOCATE(matrix(nl,nc))
      READ(1,*)((matrix(i,j),j=1,nc),i=1,nl)
1     PRINT *,"Numarul coloanei:"
      READ *,ncol
      sum=0.
      DO i=1,nl
        sum=sum+matrix(i,ncol)
      ENDDO
      PRINT *,"suma este: ",sum
      PRINT *,"Alegeti alta coloana? (D/N):"
      READ *,r
      IF(r=="D".OR.r=="d") GOTO 1
      DEALLOCATE(matrix)
9     PRINT *,"fisierul A.TXT nu este gasit!"
      END
```

Notă: structura fișierului A.TXT este ca în exemplul anterior.

3. Transpunerea unei matrici

```
      REAL,ALLOCATABLE :: matrix(:,:),mtranspus(:,:)
      CHARACTER r
1     PRINT *,”Numarul liniilor:”
      READ *,nl
      PRINT *,”Numarul coloanelor:”
      READ *,nc
      ALLOCATE(matrix(nl,nc),mtranspus(nc,nl))
      DO i=1,nl
      PRINT *,”Termenii de pe linia ”,i,”:”
      READ *,(matrix(i,j),j=1,nc)
      ENDDO
      DO i=1,nl
        DO j=1,nc
           mtranspus(j,i)=matrix(i,j)
        ENDDO
      ENDDO
      PRINT *,”matricea transpusa:”
      DO i=1,nc
        PRINT *,(mtranspus(i,j),” ”,j=1,nl)
      ENDDO
      DEALLOCATE(matrix,mtranspus)
      PRINT *,”reluati? (D/N):”
      READ *,r
      IF(r==”D”.OR.r==”d”) GOTO 1
      END
```

4. Înmulțirea a două matrici pătrate, folosind fișiere de date

```
      REAL,ALLOCATABLE :: mat1(:,:),mat2(:,:),matrez(:,:)
      CHARACTER r,numefis_in(12),numefis_out(12)
1     PRINT *,”Fisierul cu datele de intrare:”
      READ *,numefis_in
      OPEN(1,FILE=numefis_in,STATUS=’OLD’,ERR=9)
      READ(1,*)n
      ALLOCATE(mat1(n,n),mat2(n,n),matrez(n,n))
      READ(1,*)((mat1(i,j),j=1,n),i=1,n)
      READ(1,*)((mat2(i,j),j=1,n),i=1,n)
      matrez=0.
      DO i=1,n
        DO j=1,n
          DO k=1,n
            matrez(i,j)=matrez(i,j)+mat1(i,k)*mat2(k,j)
          ENDDO
        ENDDO
      ENDDO
      PRINT *,”Fisierul pentru rezultate:”
      READ *,numefis_out
      OPEN(2,FILE=numefis_out,STATUS=’UNKNOWN’,POSITION=’APPEND’)
      DO i=1,n
        WRITE(2,*)(matrez(i,j),” ”,j=1,n)
      ENDDO
      CLOSE(2)
      DEALLOCATE(mat1,mat2,matrez)
      STOP
9     PRINT *,”Nume de fisier inexistent! Reincercati? (D/N):”
      READ *,r ;IF(r==”D”.OR.r==”d”) GOTO 1
      END
```

Structura fișierului cu date: *n* (numărul liniilor= numărul coloanelor)
termenii_din_mat1 (n×n bucăți)
termenii_din_mat1 (n×n bucăți)

De exemplu, pentru matricile $\begin{vmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{vmatrix}$ și $\begin{vmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 3\end{vmatrix}$ se poate crea fișierul de date cu următorul conținut:

```
3
1,1,1,2,2,2,3,3,3
1,0,0,0,2,0,0,0,3
```

sau:

```
3
1,1,1
2,2,2
3,3,3
1,0,0
0,2,0
0,0,3
```

5. Rezolvarea unei ecuaţii pătratice de forma: $a \cdot x^2 + b \cdot x + c = 0$

```
      CHARACTER r
      INTEGER a,b,c
      PRINT *,"rezolvarea ecuatiei a.x2+b.x+c=0"
1     PRINT *,"introduceti coeficientii a, b, c: "
      READ *,a,b,c
      SELECT CASE(a)
       CASE(0)
        IF(b==0) THEN
         IF(c==0) THEN
          PRINT *,"x poate avea orice valoare"
         ELSE
          PRINT *,"greseala, c nu poate fi diferit de 0"
         ENDIF
        ELSE
         IF(c==0) THEN
          PRINT *,"ecuatia nu este patratica, solutia este x=",0.
         ELSE
          PRINT *,"ecuatia nu este patratica, solutia este x=",-c/b
         ENDIF
        ENDIF
       CASE DEFAULT
        delta=b**2-4*a*c
        IF(delta<0) THEN
         PRINT *,"radacini complexe"
         PRINT *,"x1=",-b/2/a,"+i(",SQRT(-delta)/2/a,")"
         PRINT *,"x2=",-b/2/a,"-i(",SQRT(-delta)/2/a,")"
        ELSEIF(delta==0.) THEN
         PRINT *,"radacini identice, x1=x2=",-b/2/a
        ELSE
         PRINT *,"x1=",(-b+SQRT(delta))/2/a
         PRINT *,"x2=",(-b-SQRT(delta))/2/a
        ENDIF
      END SELECT
      PRINT *,"Reluati? (D/N):"
      READ *,r
      IF(r=="D".OR.r=="d") GOTO 1
      END
```

6. Rezolvarea unui sistem liniar de 2 ecuaţii cu 2 necunoscute (x și y): $\begin{cases} a \cdot x + b \cdot y = c \\ d \cdot x + e \cdot y = f \end{cases}$

```
      CHARACTER r
1     PRINT 2,'Coeficientii a, b si c din ecuatia "a.x+b.y=c": '
      READ *,a,b,c
      PRINT 2,'Coeficientii d, e si f din ecuatia "d.x+e.y=f": '
      READ *,d,e,f
      delta=a*e-b*d
      IF(delta==0) THEN
       IF(b*f==c*e) THEN
         PRINT *,"sistem compatibil nedeterminat"
       ELSE
         PRINT *,"sistem incompatibil"
       ENDIF
      ELSE
       PRINT *,"x=",(c*e-b*f)/delta,"y=",(-c*d+a*f)/delta
      ENDIF
2     FORMAT(A,\)
      PRINT 2,"Reincercati? (D/N):"
      READ *,r
      IF(r=="D".OR.r=="d") GOTO 1
      END
```

7. Simularea unei extrageri loto cu numere pseudoaleatoare

```
      CHARACTER r
      ALLOCATABLE nr(:)
1     PRINT 2,"Cate numere se extrag: "
      READ *,n
      PRINT 2,"din cate: "
      READ *,nmax
      ALLOCATE(nr(nmax))
      DO i=1,nmax
       nr(i)=i
      ENDDO
      man=RAND(TIME())
      DO i=1,n
       id=INT(RAND(0)*(nmax-i+1))+i
       man=nr(i)
       nr(i)=nr(id)
       nr(id)=man
      ENDDO
      DO i=1,n
       DO j=i,n
        IF(nr(i)>nr(j)) THEN
         man=nr(i)
         nr(i)=nr(j)
         nr(j)=man
        ENDIF
       ENDDO
      ENDDO
      PRINT *,(nr(i),i=1,n)
      DEALLOCATE(nr)
2     FORMAT(A,$)
      PRINT 2,"Reincercati? (D/N):"
      READ *,r
      IF(r=="D".OR.r=="d") GOTO 1
      END
```

8. Simularea aruncării simultane a unei perechi de zaruri

```
      CHARACTER r
      DIMENSION n(2)
! initializarea generatorului de numere pseudoaleatoare
      man=RAND(TIME())
1     PRINT 2,”zarurile aruncate:”
      DO i=1,2
       n(i)=INT(6*RAND(0))+1
       WRITE(*,”(1X,I1)”,ADVANCE=”NO”)n(i)
      ENDDO
! avansare la un rand nou
      PRINT *
2     FORMAT(A,$)
      PRINT 2,”Reincercati? (D/N):”
      READ *,r
      IF(r==”D”.OR.r==”d”) GOTO 1
      END
```

9. Citirea unor șiruri de la tastatură și afișarea lor utilizând indicatori (pointeri)

```
! definirea unei entitati de tip nod (cu autoreferire)
      TYPE nod
        CHARACTER(60) rand
        TYPE(nod),POINTER :: urmator
      END TYPE
! definirea indicatorilor care vor fi utilizati
      TYPE(nod),POINTER :: fata,spate,pozitie
      CHARACTER(60)tampon
      CHARACTER r
1     NULLIFY(fata,spate)
      PRINT *,”apasati <Enter> pentru terminare”
! inregistrarea sirurilor tastate
      DO
        WRITE(*,”(A,$)”)”tastati orice: ”
        READ(*,”(A)”)tampon
! iesire din ciclu cand se apasa doar tasta <Enter>
        IF(tampon==””) EXIT
        IF(.NOT.ASSOCIATED(fata)) THEN
         ALLOCATE(fata)
         spate=>fata
        ELSE
         ALLOCATE(spate%urmator)
         spate=>spate%urmator
        ENDIF
        spate%rand=tampon
        NULLIFY(spate%urmator)
      ENDDO
! afisarea sirurilor tastate
      pozitie=>fata
      DO WHILE(ASSOCIATED(pozitie))
        WRITE(*,*)pozitie%rand
        pozitie=>pozitie%urmator
      ENDDO
      PRINT *,”Reluati? (D/N):”
      READ *,r
      IF(r==”D”.OR.r==”d”) GOTO 1
      STOP
      END
```

10. Calculul ariei și perimetrului unui dreptunghi, triunghi dreptunghic sau semicerc în urma selectării unei opțiuni (folosind constante Hollerith, subprograme, puncte de intrare, și reveniri la etichete)

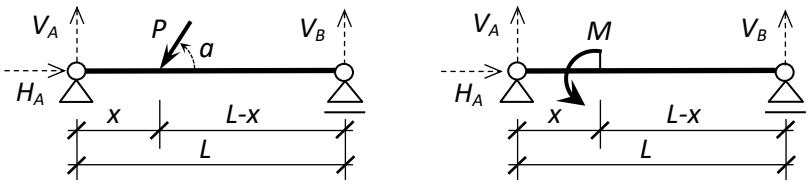
```
      CHARACTER op
1     PRINT *,24HCalcul arie si perimetru
      PRINT *,40Halegeti una dintre optiunile:
      PRINT *,40H D - pentru dreptunghi
      PRINT *,40H T - pentru triunghi dreptunghic
      PRINT *,40H S - pentru semicerc
      PRINT *,40H X - pentru iesire din program
2     PRINT 3,10H optiune:
3     FORMAT(A,\)
      READ *,op
      SELECT CASE(op)
      CASE(”D”,”d”) ; CALL d(*2)
      CASE(”T”,”t”) ; CALL t(*2)
      CASE(”S”,”s”) ; CALL s(*2)
      CASE(”X”,”x”)
       STOP
      CASE DEFAULT
       PRINT *,”optiune invalida”
       GOTO 2
      END SELECT
      GOTO 2
      END

      SUBROUTINE d(*)
      PARAMETER (pi= 3.14159)
3     FORMAT(A,\)
      PRINT 3,”lungimile celor doua laturi: ”
      READ *,a,b
      CALL test(a,b,*4)
      PRINT *,”Aria=”,a*b,”Perimetrul=”,(a+b)*2
      RETURN
      ENTRY t
       PRINT 3,”lungimile celor doua laturi perpendiculare: ”
       READ *,a,b
       CALL test(a,b,*4)
       PRINT *,”Aria=”,a*b/2,”Perimetrul=”,a+b+SQRT(a**2+b**2)
       RETURN
      ENTRY s
       PRINT 3,”lungimea bazei (diametrul): ”
       READ *,a
       CALL test(a,1.,*4)
       PRINT *,”Aria=”,pi*(a/4)**2,”Perimetrul=”,pi*a/2
       RETURN
4     RETURN 1
      CONTAINS
         SUBROUTINE test(a,b,*)
          IF(a==0.or.b==0) THEN
            PRINT *,”Nu se poate calcula cu valoare nula”
            RETURN 1
          ELSEIF(a<0.or.b<0) THEN
            PRINT *,”valoare negativa?!”
          ENDIF
          RETURN
         END SUBROUTINE test
      END
```

Varianta folosind 2 funcții cu puncte de intrare în loc de cele 3 subprograme:

```
      CHARACTER op
1     PRINT *,24HCalcul arie si perimetru
      PRINT *,40Halegeti una dintre optiunile:
      PRINT *,40H D - pentru dreptunghi
      PRINT *,40H T - pentru triunghi dreptunghic
      PRINT *,40H S - pentru semicerc
      PRINT *,40H X - pentru iesire din program
2     PRINT 3,10H optiune:
3     FORMAT(A,\)
      READ *,op
      SELECT CASE(op)
      CASE("D","d") ; PRINT 3,"lungimile celor doua laturi: "
       READ *,a,b ; CALL test(a,b,*2)
       PRINT *,"Aria=",da(a,b),"Perimetrul=",dp(a,b)
      CASE("T","t")
       PRINT 3,"lungimile celor doua laturi perpendiculare: "
       READ *,a,b ; CALL test(a,b,*2)
       PRINT *,"Aria=",ta(a,b),"Perimetrul=",tp(a,b)
      CASE("S","s") ; PRINT 3,"lungimea bazei (diametrul): "
       READ *,a ; CALL test(a,1.,*2)
       PRINT *,"Aria=",sa(a),"Perimetrul=",sp(a)
      CASE("X","x")
       STOP
      CASE DEFAULT
       PRINT *,"optiune invalida"
       GOTO 2
      END SELECT
      GOTO 2
      CONTAINS
         SUBROUTINE test(a,b,*)
          IF(a==0.or.b==0) THEN
            PRINT *,"Nu se poate calcula cu valoare nula"
            RETURN 1
          ELSEIF(a<0.or.b<0) THEN
            PRINT *,"valoare negativa?!"
          ENDIF
          RETURN
         END SUBROUTINE test
      END

      FUNCTION da(a,b)
      da=a*b ; RETURN
      ENTRY dp(a,b)
      dp=(a+b)*2 ; RETURN
      ENTRY ta(a,b)
      ta=a*b/2 ; RETURN
      ENTRY tp(a,b)
      tp=a+b+SQRT(a**2+b**2)
      END

      FUNCTION sa(a)
      PARAMETER (pi=3.14159)
      sa=pi*(a/4)**2 ; RETURN
      ENTRY sp(a)
      sp=pi*a/2
      END
```

11. Determinarea reacțiunilor la o grindă simplu rezemată supusă la o încărcare concentrată, și calcularea eforturilor dintr-o secțiune de pe axul grinzii:

Schițe cu notații:



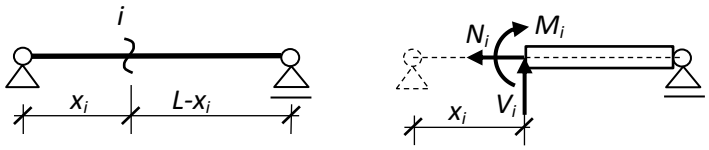
Relații utilizate: $V_A = P \cdot \sin(a) \cdot \frac{(L-x)}{L}$; $H_A = P \cdot \cos(a)$; $V_A = \frac{M}{L}$; $H_A = 0$; $V_B = -\frac{M}{L}$

$V_B = P \cdot \sin(a) \cdot \frac{x}{L}$

```
      INCLUDE ”f.txt”
1     PRINT *,”Calculul reactiunilor la capetele unei grinzi simplu”, &
       ”rezemate, care este inarcata cu o forta sau cu un moment”
      PRINT *
3     PRINT *,”pentru forta concentrata, tastati P”
      PRINT *,”pentru moment concentrat, tastati M”
      PRINT 2,”tipul incarcarii (P/M): ”
      READ *,t
      PRINT 2,”lungimea ””L”” a grinzii [m]: ”
      READ *,l
5     PRINT 2,”distanta ””x”” [m]: ”
      READ *,x
      IF(x<0.OR.x>l) THEN
       PRINT *,”incarcarea nu este pe grinda!”
       GOTO 5
      ENDIF
      SELECT CASE(t)
       CASE(”P”,”p”)
        PRINT 2,”intensitatea fortei ””P”” [kN]: ”
        READ *,p
        PRINT 2,”unghiul ””a”” fata de axul grinzii [grade]: ”
        READ *,a
        va=p*SIN(a*pi/180)*(l-x)/l
        ha=p*COS(a*pi/180)
        vb=p*SIN(a*pi/180)*x/l
        ra=SQRT(va**2+ha**2)
       CASE(”M”,”m”)
        PRINT 2,”intensitatea momentului ””M”” [kN.m]: ”
        READ *,m
        va=m/l ; ha=0. ; vb=-m/l
       CASE DEFAULT
        PRINT *,” optiune invalida! ”
        GOTO 3
      END SELECT
      PRINT 4,”VA= ”,va,”kN; HA= ”,ha,”kN; VB= ”,vb,”kN”
      IF(.NOT.(a==0.OR.a==90)) PRINT 4,”RA= ”,ra,”kN la”, &
       ATAN(va/ha)*180/pi,”grade”
      CALL FINT(l,x,t,va,ha,p,m,vb,a)
      PRINT 2,” Reluati? (D/N): ”
      READ *,r
      IF(r==”D”.OR.r==”d”) GOTO 1
      END
```

Subprogramul FINT pentru calculul eforturilor într-o secțiune aleasă:

Ilustrație:




```
SUBROUTINE FINT(l,x,t,va,ha,p,m,vb,a)
INCLUDE ”f.txt”
PRINT 2,’”xi” pentru sectiune:’
READ *,xi
SELECT CASE(t)
CASE(”P”,”p”)
  IF(xi<x) THEN
   fn=ha
   fv=va
   fm=va*xi
  ELSE
   fn=0.
   fv=va-p*SIN(a*pi/180)
   fm=va*xi-p*SIN(a*pi/180)*(xi-x)
  ENDIF
 CASE(”M”,”m”)
  IF(xi<x) THEN
   fn=ha
   fv=va
   fm=va*xi
  ELSE
   fn=0.
   fv=va
   fm=va*xi-m
  ENDIF
END SELECT
PRINT 4,”Ni= ”,fn,”kN; Vi= ”,fv,”kN; Mi= ”,fm,”kN.m”
RETURN
END
```

Conținutul fișierului F.TXT inclus în program (acest fișier trebuie să fie în aceeași mapă cu fișierele sursă):

```
 CHARACTER r,t
 REAL l,m
 DATA pi/3.14159/
2  FORMAT(1X,A,$)
4  FORMAT(1X,3(A,F8.2),A)
```

Resurse

Locul de origine al standardelor Fortran (JTC1/SC22/WG5): https://wg5-fortran.org/

Limbajul de programare Fortran: https://fortran-lang.org/

Fortran Wiki: https://fortranwiki.org/

Fortranplus | Fortran Information. https://www.fortranplus.co.uk/fortran-information/

Proiectul G95 / Rulare G95 (opțiuni, coduri de eroare): https://g95.sourceforge.net/docs.html

Câteva cărți și tutoriale accesibile online

Metcalf M., Reid J. K.: *Fortran 90/95 Explained*, Oxford University Press, 1996. https://archive.org/details/fortran9095expla0000metc

Sandu A.: *Lecture Notes. Introduction to Fortran 95 and Numerical Computing. A Jump-Start for Scientists and Engineers*. Michigan Technological University, 2001. https://www-eio.upc.edu/lceio/manuals/Fortran95-manual.pdf

van Mourik T.: *Fortran 90/95 Programming Manual*. University College London, 2005. https://www-eio.upc.edu/lceio/manuals/Fortran95-manual.pdf

Nicholson J. A.: *Introduction to Programming using FORTRAN 95*, 2011. https://www.fortrantutorial.com/documents/IntroductionToFTN95.pdf

Learn – Fortran Programming Language. https://fortran-lang.org/en/learn/

Fortran Tutorial – Free Guide to Programming Fortran 90/95. https://www.fortrantutorial.com/

TutorialsPoint - Fortran Tutorial: https://www.tutorialspoint.com/fortran/index.htm

The Irish Centre for High-End Computing (ICHEC): Fortran Tutorial. https://www.ichec.ie/academic/national-hpc/documentation/fortran-tutorial