Noțiuni de teorie începând cu LUCRAREA nr. 2

Scurt istoric Fortran

Prima versiune a acestui limbaj de programare a fost creat de către o echipă de la IBM sub conducerea lui John W. Backus, fiind lansată în anul 1957 sub denumirea ”IBM Mathematical Formula Translating System” (pe scurt: FORTRAN, din combinarea părților 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. Versiunea 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 ar fi trebuit să fie lansată în anii ’80 (Fortran 8X), dar a apărut doar în 1991 introducând formatul liber și a devenit cunoscut ca Fortran 90, deschizând o cale pentru HPF (High Performance Fortran). În 1997 a fost publicat standardul pentru Fortran 95, prima variantă orientată pe obiecte. Prezentul document se referă în principal la această versiune (Fortran 95) a limbajului, prezentând însă doar noțiuni pentru începători și având ca scop furnizarea informațiilor necesare lucrărilor la disciplina de ”Programarea calculatoarelor și limbaje de programare” pentru studenții de la Inginerie Civilă.
În comparație cu C++ (limbaj orientat pe obiecte care suportă polimorfism și mosteniri), Fortran a introdus câteva caracteristici similare (prin module și tipuri derivate), însă nu are 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 implementat 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 (de pe pagina accesibilă la adresa https://fortran-lang.org/) pentru cei care se interesează de utilitatea acestui limbaj: ”Fortran este utilizat în principal în domenii care au fost pionieri în domeniul calculului digital, adică în știință și inginerie. De exemplu, prognoza vremii și oceanelor, mecanica fluidelor, matematică aplicată, statistică și finanțe. Fortran domină în calculul de înaltă performanță (HPC) și este folosit pentru a testa capacitățile celor mai puternice supercomputere din lume.”
O parte semnificativă din cele ce urmează constituie o adaptare a conținutului cărții ”Inițiere în programare și în limbajul Fortran” (F.-Zs. Gobesz, C. Bacoțiu, Editura UTPres, Cluj-Napoca, 2003, ISBN 973-662-005-0).

Alcătuirea fișierului sursă

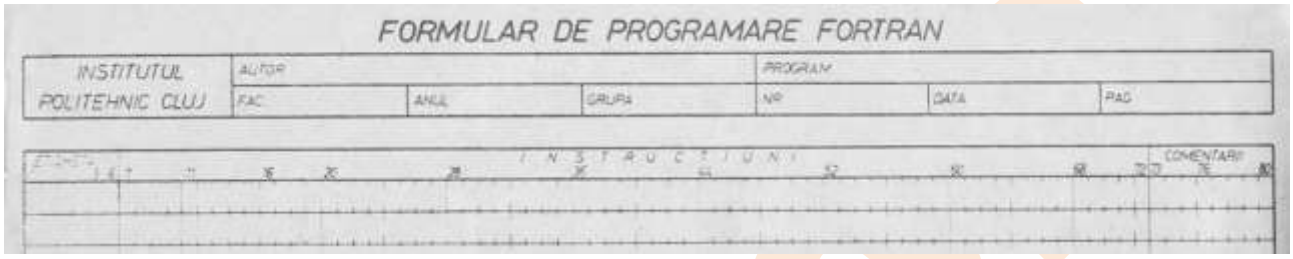
Un fișier sursă poate conține una sau mai multe unități de program (acestea vor fi prezentate mai târziu), sau fragmente ale acestora (sub formă de secțiuni). Redactarea fișierului sursă se poate face cu orice editor de text, cu condiția ca să rezulte un conținut alcătuit din caractere (format ASCII).
Setul de caractere utilizabile in limbajul Fortran conține caracterele alfanumerice (cele 26 litere mici sau mari ale alfabetului englez: `a`–`z`, `A`–`Z`, şi cifrele: `0`–`9`), plus 4 simboluri pentru operaţii aritmetice (`+`, `-`, `*`, `/`) şi un set de caractere speciale (blank sau spaţiu, tabulator orizontal, virgulă, punct, apostrof, paranteze rotunde deschise şi închise, precum şi următoarele caractere: `=`, `$`, `&`). Limbajul Fortran 90 a extins această listă cu următoarele caractere speciale: `_`, `!`, `:`, `;`, `"`, `%`, `<`, `>`, `?`, `^` şi `#`. Conform convenţiilor anglo-saxone virgula are rolul de a separa elementele în cadrul unei liste, iar punctul este separatorul zecimal.

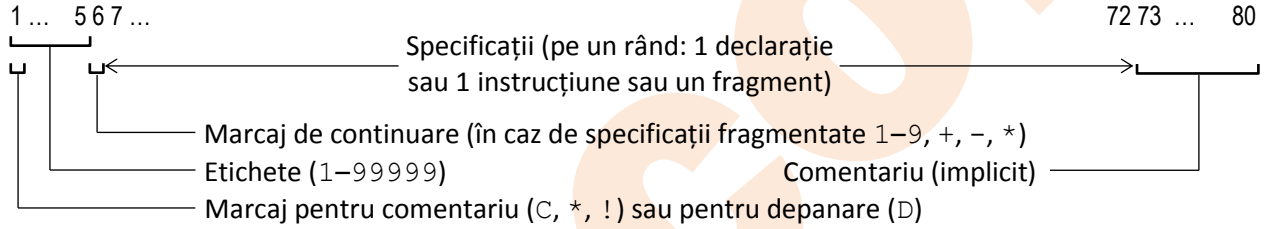
Pentru denumirea variabilelor, diferitelor secţiuni de program precum şi pentru identificarea funcţiilor, se folosesc nume simbolice. Dacă convenţiile versiunilor mai vechi ale limbajului au permis utilizarea a doar 8 caractere (alcătuite din caractere alfanumerice şi caracterul special `$`), Fortran 95 permite utilizarea a 31 de

caractere (alcătuite din caractere alfanumerice, caracterul special `$` şi caracterul special `_`). Primul caracter trebuie să fie întotdeauna o literă. Numele unităților de program și ale secţiunilor sunt considerate globale şi trebuie să fie unice în întreaga sursă, iar numele entităților trebuie să fie unice în cadrul aceleiași unități de program. Limbajul Fortran nu face diferențe între majuscule sau litere mici în cazul numelor simbolice.

Modul de redactare al fișierului sursă poate fi în *formă fixă* (Fortran 77), *formă tabulară* sau *formă liberă* (ultimele fiind introduse prin Fortran 90 şi admise de versiunile ulterioare ale limbajului).

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 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 dorește marcarea unui rând ca şi comentariu, în prima coloană se va scrie litera `C` sau caracterul `*` (respectiv `!` începând cu Fortran 90). În acest caz structura şi conţinutul rândului fiind ignorate 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 se scria doar o singură specificație pe un rând, dar începând cu Fortran 90 se admite scrierea mai multor specificații pe un rând, în cazul acesta caracterul ; fiind separatorul dintre ele.
Dacă instrucțiunea este mai lungă decât spaţiul dintre coloanele 7 şi 72, se poate continua pe rândurile următoare marcând în coloana 6 (printr-o cifră sau cu unul din simbolurile: `+`, `–`, `*`) că fragmentele sunt continuarea celor precedente. Începând cu Fortran 90 se poate folosi orice caracter în afară de cifra `0` pentru acest marcaj de continuare. Numărul liniilor de continuare admise depinde şi de compilatorul ales. Fortran 77 permitea 99 de fragmente (1 rând inițial și 98 rânduri în continuare), însă Fortran 90 standard acceptă doar 19 fragmente în forma fixă, respectiv 39 de fragmente în formatul liber. 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 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 <*Tab*> este primul caracter de pe un rând, atunci rândul conține o specificație (declarație sau instrucțiune, sau eventual un marcaj). În cazul în care acest prim caracter este urmat de o cifră nenulă, cifra va marca un fragment de continuare a rândului anterior și va trebui să fie urmată de un spațiu pentru a-l separa de conținutul continuării. În fața caracterului <*Tab*> poate fi doar un marcaj de comentariu sau o etichetă. Lungimea rândurilor nu poate depăși coloana 72 în cazul formatului fix și coloana 132 în cazul formatului liber.

Indiferent de structura pe orizontală (formă fixă, liberă sau tabulară), structura pe verticală a uni 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. În cazul în care 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 (prin specificația `INCLUDE`) într-un alt fișier sursă.

Atenție: 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 inclus, iar punctele succesive (…) se referă la elemente repetabile. Secvențele cursive marchează elemente care țin locul conținutului din pozițiile în care apar.

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 perechi de numere cu parte zecimală), `LOGICAL` (valori logice, există doar două, constantele `.TRUE.` și `.FALSE.`), `CHARACTER` (caracter sau șir de caractere), respectiv `BYTE` (valoare pe 8 biți, utilizat 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*`)`, unde *nume* 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=`, sau `LEN=` în cazul tipului `CHARACTER`). Această valoare depinde de tipul entităților, existând și variante implicite în funcție de compilator (de regulă 4 octeți pentru entități de tipul `REAL` și 2 sau 4 octeți pentru entități de tipul `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`. Caracterele individuale și entitățile de tipul `BYTE` se stochează pe câte 1 byte, deci lungimea lor de stocare nu se poate modifica explicit (dacă la tipul `CHARACTER` se specifică *fel*-ul, acesta va marca implicit numărul de caractere din șir). Entitățile de tipul `INTEGER(1)` și `LOGICAL(1)` se vor stoca tot pe câte 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 are doar rolul de a delimita lista cuvintelor cheie aflate la stânga, de *lista_entități* aflate la dreapta specificației).

Pentru entități numerice există o regulă implicită referitoare la tipul lor, care (desigur) se poate modifica sau anula cu următoarea sintaxă a declarației `IMPLICIT`,:

`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.

Definirea unui tip derivat se face conform sintaxei:

`TYPE` *nume*
specificații
`END TYPE` [*nume*]

După ce au fost definite, asemenea tipuri derivate se pot utiliza la specificarea tipului entităților, înlocuind cuvântul cheie *tip* cu `TYPE(`*nume*`)` în declararea explicită a tipului. 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* (listă delimitată cu caracterele `/`), în ordinea succesiunii de la stânga la dreapta.

Exemple:	Explicații:

`REAL(KIND=8) Di,e33`
`! Echivalent cu:`
`REAL(8) dI,E33`
	Entitățile (variabilele) numite DI și E33 sunt 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 dacă numele entităților e scris cu litere mici sau cu majuscule.

`COMPLEX(KIND=8) xC,Y1`
`! Echivalent cu:`
`COMPLEX(8) Xc,y1`
	Entitățile (variabilele) numite XC și Y1 sunt 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 este de tipul `INTEGER`, memorat pe 2 bytes și folosit doar pentru primire 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 doar un singur element.
`REAL,PARAMETER :: pi=3.14159`	Entitatea numită PI este de tipul `REAL` și are valoarea constantă (nemodificabilă) de 3,14159.
`EXTERNAL :: SIN`	Entitatea numită SIN este declarată ca 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 și de tipul `REAL`, accesibile doar în unitatea de program curentă.
`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).

`TYPE comp`
`  CHARACTER(LEN=24) nume`
`  INTEGER zi`
`  CHARACTER(3) luna`
`  INTEGER :: an=2023`
`END TYPE`
`...`
`TYPE(comp) r23,r24`
`CHARACTER at,stea*3`
`INTEGER m1,m2,m3`
 	
`DATA at,m1,m2/"@",2*1/,m3/5/`
`DATA stea/"***"/,r24%an/2024/`
`DATA r24%luna,r24%zi/"AUG",12/`
	Tipul derivat numit COMP este definit ca fiind alcătuit din două șiruri de caractere (NUME având 24 de poziții iar LUNA 3) și două numere întregi (ZI și AN, cel din urmă fiind și inițializat cu valoare 2023). Se poate observa opționalitatea cuvântului cheie `LEN=`, acesta nefiind utilizat la șirul LUNA.
	
	Entitățile numite R23 și R24 vor avea tipul definit anterior. 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`.
	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 pot fi aritmetice (numerice), de șir (caractere), logice, respectiv de inițializare și specificare (începând cu Fortran 90), fiind 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 faţă 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 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 întreg). 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	Formulă
`(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	Formulă
`(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}

Expresiile de 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 `CHARACTER`. Evaluarea unei asemenea expresii produce o singură valoare de tip şir. 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.

Expresiile 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 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 relaţionali şi operatorii 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.`	Negaţie logică, rezultă adevărată dacă operandul are valoarea falsă şi falsă dacă operandul are valoarea adevărată.
`.AND.`	Conjuncţie logică, rezultă adevărată doar dacă ambii operanzi au valoarea adevărată, în caz contrar rezultă falsă.
`.OR.`	Disjuncţie logică, rezultă adevărată dacă unul din operanzi are valoarea adevărată, în caz contrar rezultă falsă.
`.EQV.`	Echivalenţă logică, rezultă adevărată dacă ambii operanzi au aceaşi valoare, dacă au valori diferite atunci rezultă falsă.
`.NEQV.`	Inechivalenţă logică, rezultă adevărată dacă operanzii sunt diferiţi, şi falsă dacă sunt la fel.
`.XOR.`	Disjuncţie logică exclusivă (SAU exclusiv), efect similar cu inechivalenţa logică.

Cei 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.

Expresiile de iniţializare şi specificare 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).

Există expresii omogene (dacă operatorii și operanzii sunt de aceeași tip) și neomogene (dacă tipul operatorilor și operanzilor este de mai multe feluri). Prioritatea de evaluare a operatorilor din cadrul expresiilor neomogene este următoarea (în ordine descrecătoare):

- operatori unari definiţi și 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.`, `.XOR.` sau `.EQV.`sau `.NEQV.`).

Operanzii pot fi valori variabile (doar entitățile denumite pot avea valori variabile) sau valori constante. Valorile constante se specifică în funcție de felul lor, după cum se eximplifică 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`
	`-.13`
	`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).

Număr binar	`B”1001”`
	`b”1011”`
	`B’1100’`
		Se pot folosi doar cifrele 0 sau 1 (max. 256 de poziții) citând valoarea după marcajul `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 `O`. Ca și anterior, semnul minus nu are efect în față și nu se acceptă în interior.

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 interior.

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 pe 64 de biți.

Funcții intrinseci

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, în ordinea alfabetică a rolului:

Rol:	Funcția:	Rezultat:
|x|	`ABS` (*x*)	Valoarea absolută (modulul) argumentului X specificat.
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.
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.
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), sau

`READ` (`[UNIT=]`*u*`[,` `[FMT=]`*f*`][,` `ERR=`e_1`][,` `END=`e_2`][,` `IOSTAT=`*var*`]`) [*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 (EOF) într-un fișier sau în cazul lipsei valorilor de citit, e_2 este eticheta unei instrucţiuni executabile la care s-ar sări în cazul apariției unei erori la citire, iar *var* reprezintă numele unei variabile de tipul INTEGER în care s-ar înregistra succesul / eșecul operațiunii de citire (în cazul citirii reușite valoarea rezultă 0, în cazul nereușitei rezultă valori mai mari care marchează coduri de eroare). 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 variante diferite, cum ar fi citirea internă (pentru conversia din caractere în numerele 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*]) [*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 (similar cu efectul caracterului <*LF*>, însemnând *Line_Feed*).
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 în care se utilizează * la specificația de format (semnificând 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 *,"n= "`
`! Echivalent cu:`
`WRITE(*,*)"n= "`
	Se va afișa șirul de caractere citat (fără semnele de citare).

`PRINT *,"Max= ",max`
`! Echivalent cu:`
`WRITE(*,*)"Max= ",MAX`
	Se va afișa șirul de caractere citat, urmat de conținutul (valoarea) variabilei MAX.