Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 1. Structura computerelor

- ***Ce este un computer?***
- ***Computere cu arhitectură Von Neumann***
- ***Memoria***
- ***Microprocesorul***
- ***Setul de caractere***

Ce este un computer?

Calculatorul, cunoscut și sub denumirea de sistem de calcul, este perceput de marea majoritate a utilizatorilor ca fiind o ***mașină*** ce execută un ***set de instrucțiuni***, reunite într-un ***program***, cu scopul de a prelucra datele furnizate (***date de intrare***) și de a obține rezultate (***date de ieșire***).

Un ***calculator numeric*** (*computer*) este un sistem (ansamblu) definit prin componentele sale (echipamente fizice, programe) conectate astfel încât să formeze o entitate coerentă cu o funcţie bine definită.

Orice calculator cuprinde două componente principale: e*chipamentele fizice* și *programe*.

Ce este un computer?

Echipamentele fizice sunt reprezentate de totalitatea componentelor electronice, electrice, electromecanice care realizează împreună funcţiile sistemului de calcul, fiind cunoscute sub denumirea de ***hardware***.

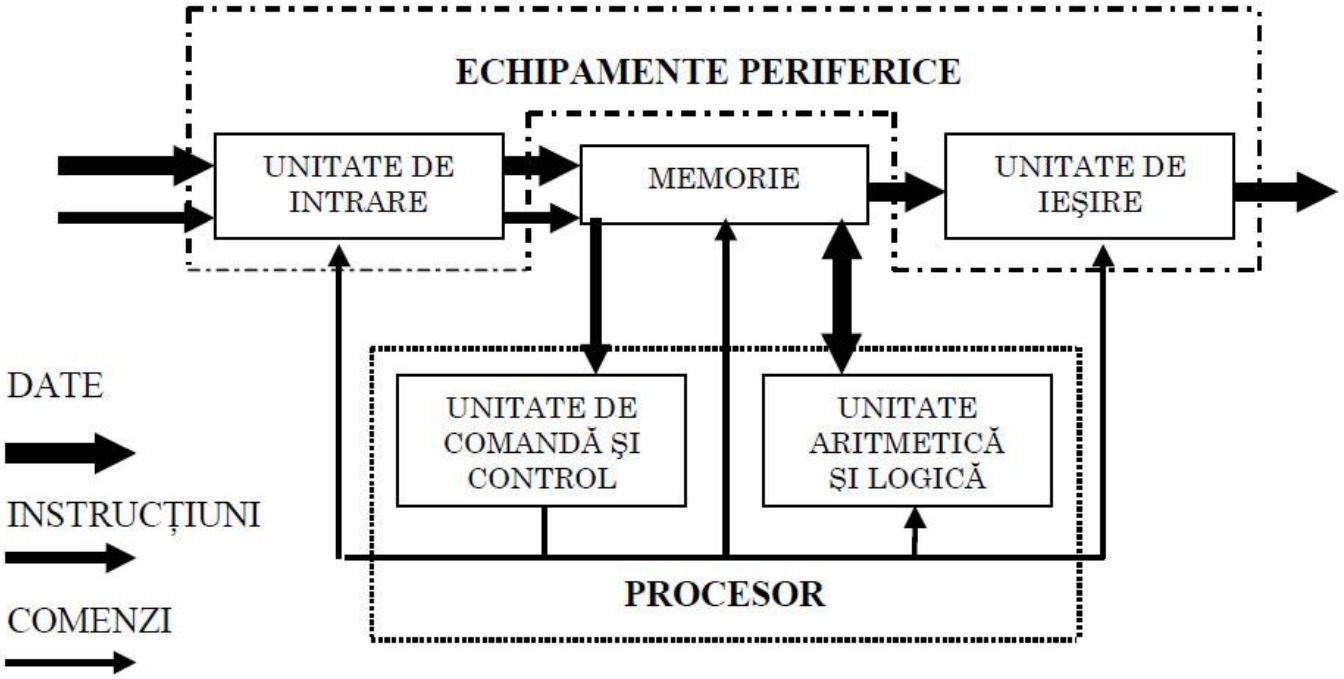
Totalitatea programelor ce permit utilizatorului accesul la toate funcţiile sistemului de calcul sunt cunoscute sub denumirea de ***software.***

Persoanele ce utilizează un computer sunt fie ***programatori***, fie ***utilizatori.*** Prin programator se înțelege o persoană ce creează un program sau corectează unul creat anterior. Persoana ce foloseşte, pentru un anumit scop, un program creat anterior se numește utilizator.

Computere cu arhitectură Von Neumann

Arhitectura unui computer poate fi definită ca o schemă *funcțională generală* utilizată pentru realizarea constructivă a oricărui calculator.

Structura funcțională clasică a unui computer este cea ***serială***, numită și *arhitectură von Neumann*.



Computere cu arhitectură Von Neumann

Principiile funcționale fundamentale ale unui calculator von Neumann sunt:

- Calculatorul are în componență o ***unitate de intrare*** şi o ***unitate de ieșire***, prin intermediul cărora se realizează comunicarea cu mediul extern (utilizator, proces industrial, calculator);
- Calculatorul are un dispozitiv numit ***memorie*** unde sunt păstrate, atât timp cât este necesar, instrucțiunile, datele supuse prelucrării şi rezultatele obținute;
- Calculatorul posedă o ***unitate de calcul aritmetic şi logic***, care execută operaţiile codificate în instrucţiunile programului;

Computere cu arhitectură Von Neumann

- Calculatorul posedă o ***unitate de comandă şi control***, care determină execuția instrucțiunilor în mod secvențial, câte una la un moment dat, în ordinea stabilită prin program; de asemenea, unitatea de comandă şi control determină transferul datelor ce urmează a fi prelucrate, între memorie şi unitatea aritmetică şi logică.

Unitatea aritmetică şi logică împreună cu unitatea de comandă şi control alcătuiesc ***unitatea centrală de prelucrare*** (*CPU-Central Processing Unit*), cunoscută şi sub denumirea de ***procesor***.

Echipamente fizice: Memoria

Memoria unui calculator se numește ***memorie internă*** deoarece în componența sa mai există şi alte dispozitive de memorare, dar cu alte caracteristici şi funcţii, care alcătuiesc împreună așa-numita ***memorie externă***.

În memoria computerului se stochează ***obiecte***, adică informații codificate. Aceste obiecte stocate în memorie le vom mai numi ***date***. Cu obiectele stocate în memorie procesorul poate efectua diferite operații.

Memoria este formată dintr-o mulțime de ***locații*** identificabile fiecare printr-o ***adresă***, adică un număr natural ce arată poziția în memorie a locației. Locațiile sunt formate din elemente binare de memorie, adică dispozitive fizice ce prezintă doar 2 stări stabile diferite, stări pe care le notăm simbolic cu 0 și 1.

Echipamente fizice: Memoria

Locaţiile de memorie se grupează în ***celule de memorie***. Identificarea unei celule de memorie se face prin ***adresa*** şi ***lungimea*** ei. *Adresa* celulei de memorie este dată de *adresa primului octet*, iar *lungimea* de numărul de elemente binare grupate într-o celulă de memorie.

Cuvântul (*word*) este o unitate logică de informaţie, formată din 8, 16, 24, 32 sau 64 de biți și denotă numărul de cifre pe care un CPU le poate prelucra la un moment dat. Lungimea cuvântului depinde de tipul constructiv al calculatorului.

Echipamente fizice: Memoria

Unitățile de măsură a memoriei sunt:

- ***Bitul*** (***0***, ***1***), notat ***b***;
- ***Byte*** (***octet***), format din *8* biți succesivi, notat ***B***;
- ***Kilobyte***, notat ***KB***, 1 KB = 2^{10} B = 1024 B;
- ***Megabyte***, notat ***MB***, 1 MB = 2^{10} KB = 1 048 576 B;
- ***Gigabyte***, notat ***GB***, 1 GB =2^{10} MB = 2^{20} KB = 2^{30} B.

Echipamente fizice: Memoria

Memoria internă este caracterizată de doi parametri: (1) ***capacitatea de memorare***, (2) ***timpul mediu de acces*** sau ***frecvența de lucru***.

Capacitatea memoriei interne, caracteristică importantă pentru viteza şi eficiența cu care va lucra calculatorul, depinde de tipul microprocesorului şi se măsoară în KB sau MB. Practic, memoria internă este formată din plăcuțe de memorie care actual au capacități de 512 MB, 1, 2, 4, 8, 16 GB.

Timpul mediu de acces se măsoară în *ns* și se referă la intervalul de timp care este necesar memoriei interne pentru a fi citită sau scrisă, iar *frecvența* în *MHz*.

Echipamente fizice: Memoria

Memoria internă este formată din două tipuri de cipuri de memorie cu caracteristici diferite şi anume:

- ***Memorie RAM*** (***R**andom **A**ccess **M**emory* – memorie cu acces aleatoriu: poate fi *citită* sau *scrisă*, este *volatilă*, adică datele memorate se păstrează atât timp cât calculatorul este conectat la rețeaua electrică, iar fiecare locație de memorie poate fi accesată imediat;
- ***Memorie ROM*** (***R**ead **O**nly **M**emory* – memorie doar citită): poate fi numai *citită*, *nu poate fi scrisă* sau modificată, este *nevolatilă* (*permanentă*).

Memoria RAM este folosită pentru a memora date sau instrucţiuni ale programelor care se execută la cererea utilizatorului, iar memoria ROM conține date necesare la pornirea şi în timpul funcționării calculatorului.

Echipamente fizice: Microprocesorul

O unitate centrală de prelucrare (**CPU**) a informației are 2 funcții: (a) transferul şi prelucrarea datelor și (b) controlul activităţii întregului sistem de calcul şi care, fizic, se prezintă sub forma unui *cip* ce se numeşte microprocesor.

Există mai multe clasificări ale noțiunii de microprocesor. După tipul de sarcini ce pot fi realizate eficient, se disting:

- Microprocesoare de uz general, nespecializate;
- Microprocesoare specializate, ca de exemplu procesoare de intrare/ieşire, utilizate la transferul datelor dintre calculator şi mediul extern, coprocesoare matematice, specializate în operaţii aritmetice de utilitate generală, coprocesoare grafice, destinate îmbunătățirii vitezei și calității afișării imaginilor etc.

Echipamente fizice: Microprocesorul

Microprocesorul este un cip format dintr-un circuit integrat complex, care conține milioane de tranzistoare, ce prelucrează date prin executarea, în mod secvențial, a unor operații logice şi/sau aritmetice în conformitate cu instrucțiunile furnizate de utilizator.

În interiorul CPU am putea regăsi subunități precum: *magistrale interne* (conductori pentru transmiterea semnalelor electrice), *registre* (memorii de capacitate mică folosite pentru memorarea unor stări sau valori intermediare), *unitatea aritmetică şi logică* (*ALU-**A**rithmetic and **L**ogic **U**nit*), *unitate de calcul aritmetic cu numere reale*, unități specializate (*buffer* de instrucțiuni etc).

Echipamente fizice: Microprocesorul

Indiferent de tipul de microprocesor există două unități comune şi anume:

- *Unitatea de execuție*, care realizează în mod efectiv operațiile folosind zonele de memorie incluse microprocesorului, cunoscute sub denumirea de *registre*; scopul registrelor este de a memora date, adrese de memorie, adresa următoarei instrucțiuni ce trebuie executată, precum şi indicatori de stare care arată cum s-au terminat instrucțiunile anterior executate;
- *Unitatea de interfaţă cu magistrala externă*, care realizează transferul datelor şi instrucțiunilor de la şi spre microprocesor.

Echipamente fizice: Microprocesorul

Diferențierea microprocesoarelor, din punctul de vedere al performanțelor, se face în funcție de *cantitatea de memorie ce poate fi adresată*, *setul de instrucțiuni executabile* (numărul şi tipul), *viteza de lucru*.

Valoarea maximă a memoriei adresabile este importantă deoarece un program nu poate fi executat decât dacă el se află în memorie, astfel încât, o memorie adresabilă de dimensiune mai mare facilitează o execuție mai rapidă a programului.

Cu cât setul de instrucțiuni specific unui microprocesor este mai mare cu atât se pot rezolva probleme din ce în ce mai complexe.

Viteza de lucru a unui microprocesor este determinată, în principal, de următorii parametri: *frecvența ceasului intern*, *mărimea şi numărul registrelor interne*, *mărimea magistralei de date*.

Echipamente fizice: Microprocesorul

Ceasul intern este un oscilator ce trimite pulsuri la intervale egale de timp, bine determinate. Executarea unei instrucțiuni de către microprocesor se face într-un număr de etape realizate în intervalul delimitat de două semnale succesive ale ceasului intern. Frecvența cu care se generează aceste semnale se numește frecvența ceasului intern.

Prin creșterea *numărului registrelor* şi a capacității de memorare a acestora se micșorează numărul de operații de transfer de date cu memoria internă, îmbunătățind astfel viteza de lucru a microprocesorului. Volumul de date ce este transferat la un singur puls al ceasului intern este mai mare cu cât mărimea magistralei de date este mai mare, prin aceasta micşorându-se numărul de operații de transfer de date cu memoria.

Echipamente fizice: Microprocesorul

Pentru a mări viteza de lucru a microprocesoarelor, odată cu avansul tehnologic, au fost introduse microprocesoarele multi-core.

Aceste tipuri de procesoare execută în paralel instrucțiuni și fiecare nucleu are propriul L1 și L2 cache. Aceste tipuri de memorii au viteze foarte mari de scriere/citire însă capacitatea lor de stocare este redusă.

Datorită spațiului mic de stocare al acestor memorii s-a făcut următorul pas și anume introducerea L3 cache care poate avea până la 40 MB (AMD Ryzen Threadripper 1950X) însă este mai lentă. Acest tip de memorie comună tuturor nucleelor.

Setul de caractere

Pentru a descrie datele, programul, precum și transmisia datelor între diferite dispozitive hardware, programatorul folosește un ***set de semne*** numite ***caractere***. Caracterele sunt, fie caracterele folosite de obicei în scrierea unor texte (litere, cifre, semne de punctuație, semne speciale), fie caractere de comanda folosite pentru transmisia datelor.

Pentru a descrie datele, programul, precum și transmisia datelor pentru uzul său intern, computerul folosește un set de ***semne binare***.

Între cele două seturi de semne caracter și semne binare există o lege de corespondență ce se numește *cod* .

În industria informațională cel mai răspândit cod este ***codul ASCII*** *(***A**merican **S**tandard **C**ode for **I**nformation **I**nterchange). Codul ASCII standard are 128 de caractere. Fiecărui caracter i se asociază o anumită combinație de 8 biți.

Setul de caractere

Tastatura este principalul dispozitiv periferic al unui calculator prin intermediul căruia se transmit informaţii (comenzi, date) către unitatea centrală. De la tastatură datele circulă unidirecţional către unitatea centrală, fiind un periferic de intrare.

Din punctul de vedere al funcţionalităţii, o tastatură standard cuprinde un număr de 101 taste:

- *Taste caracter*, ce cuprinde tastele corespunzătoare literelor alfabetului latin, mari şi mici, cifrele arabe şi caracterele speciale (!,@,”,#,$,%,^,&,*,(,),_, -, =,+,/,\,|,;,:,’,~,`,);
- *Taste numerice*, care cuprind taste numerice propriu-zise, cu două caractere înscrise şi taste cu operatorii aritmetici;
- *Taste de alegere a modului de lucru,* **Caps Lock**, **Num Lock**, **Insert** (schimbă modul de lucru al tastelor caracter din inserare în suprapunere şi invers);

Setul de caractere

- *Taste de modificare a funcției unor taste sau grupe de taste (Shift)*, prin menținerea în stare apăsată permite tastarea caracterului superior de pe tastele cu două caractere sau inversează acțiunea tastei Caps Lock; tastele **Ctrl** şi **Alt**, separat sau împreună, sunt utilizate în combinații cu alte taste pentru generarea de comenzi;
- *Taste de deplasare/poziţionare*, prin apăsare determină deplasarea cursorului în direcția indicată de fiecare tastă; tastele de deplasare sunt tastele *săgeți,* **Page Up, Page Down, Home** şi **End.**
- *Taste funcționale*, care determină lansarea imediată a unei comenzi predefinite, în funcție de programul ce se execută de către calculator; aceste taste sunt codificate **F1, F2, …, F12**;

Setul de caractere

- *Taste cu acțiune bine definită:*

✓ **Enter** (execuție comandă sau trecere la un rând nou);

✓ **RETURN** (execuție comandă sau trecere la un rând nou);

✓ **Esc** (anulează comanda, respectiv programul în curs de execuție);

✓ **Back Space** (șterge un caracter în stânga cursorului) **;**

✓ **Delete** (șterge un caracter de pe poziția curentă a cursorului);

✓ **Tab** (determină trecerea la rubrica următoarea sau afișează un spațiu liber);

✓ **Print Screen** (determină tipărirea (copierea) ecranului curent);

✓ **Scroll Lock** (oprește defilarea pe ecran a datelor determinate de execuția unei comenzi);

✓ **Pause/Break** (determină oprirea (temporară sau definitivă) a anumitor procese - programe).

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 2. Computere Personale

- ***Structura hardware a PC-urilor***
- ***Sisteme de operare. Microsoft Windows***

Structura hardware a PC-urilor

Structura hardware a unui computer se referă la toate componentele fizice ce alcătuiesc un calculator. Există multe tipuri de echipamente hardware ce se pot regăsi în interiorul unui calculatorului. Acestea pot comunica între ele precum și cu alte dispozitive din afara calculatorului.

Cele mai uzuale echipamente hardware ce se pot regăsi într-un calculator sunt:

- Placa de bază,
- CPU (unitatea centrală de procesare),
- RAM
- Sursă de tensiunea,
- Placă video,
- Hard disk
- Unitate optică,
- Unitate de citire carduri.

Structura hardware a PC-urilor

Pe lângă aceste componente se mai regăsesc și altele, ce se regăsesc în afara calculatorului. Printre acestea reamintim:

- Monitor,
- Tastatură,
- Mouse,
- UPS (baterie de back-up),
- Stick de memorie,
- Imprimantă,
- Boxe,
- Hard disk extern.

Un alt grup de echipamente mai puțin comune, fie datorită faptului că acestea au fost integrate în alte echipamente sau au fost înlocuite de noi echipamente, sunt următoarele: Placă de sunet, Placă de rețea, Modem, Scaner, Videoproiector, Floppy disk, Joystick, Cameră web, Microfon.

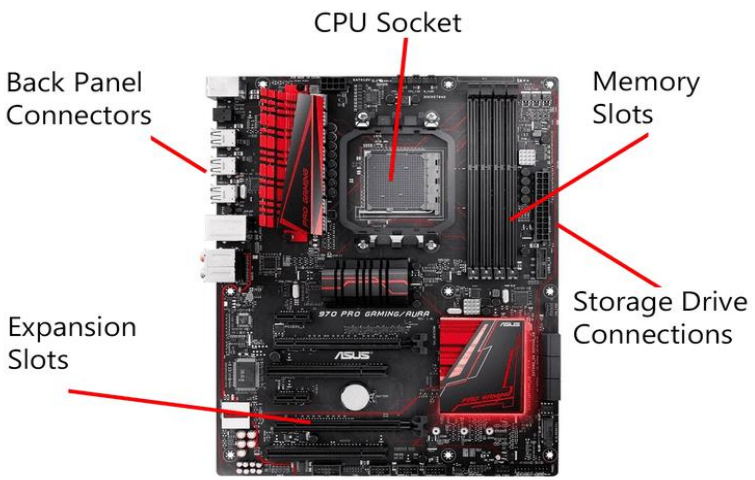
Structura hardware a PC-urilor

Placa de bază se regăsește în carcasa calculatorului și este fixată de aceasta prin șuruburi.

Toate componentele calculatorului sunt conectate, într-un fel sau altul, la placa de bază.

Pe placa de bază regăsim următoarele:

- mai multe conexiuni pentru componente cum ar fi placa video și placa de sunet,
- conectori dedicați pentru memoria RAM și CPU,
- un panou situat în partea din spate a carcasei, cu conexiuni pentru echipamentele periferice,
- conectori pentru dispozitivele de stocare (Flopy, Cd-Rom, Hard disk).

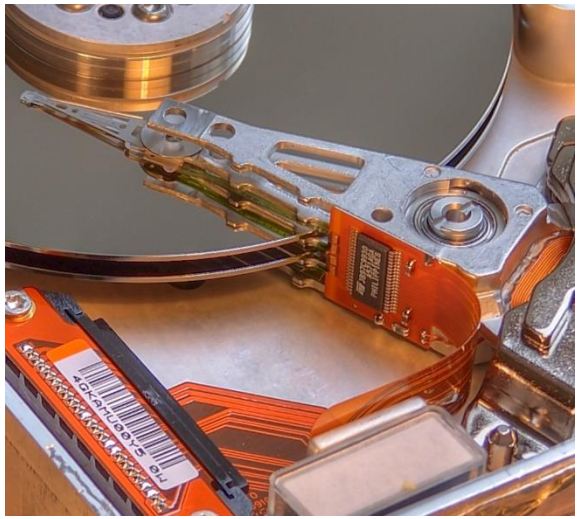


Structura hardware a PC-urilor

Hard disk-ul este un dispozitiv electro-mecanic folosit pentru stocarea sau memorarea nevolatilă a datelor şi reprezintă principala memorie externă a unui computer.

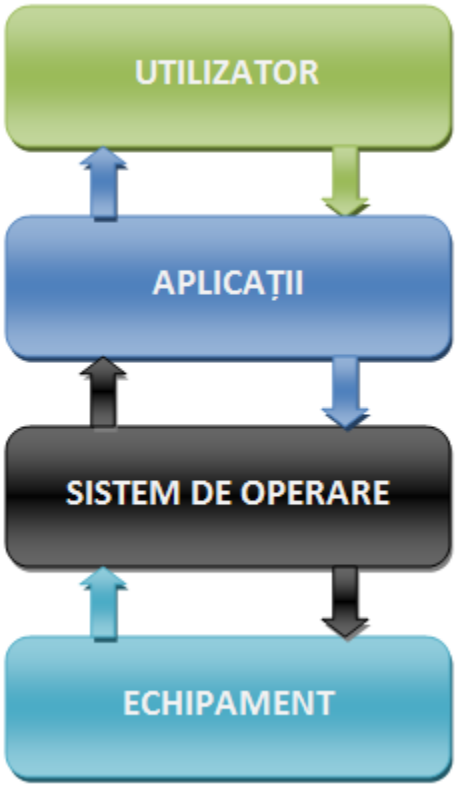
Stocarea datelor se face pe o suprafaţă magnetică dispusă pe platane rotunde metalice rigide (dure).

O alternativă la folosirea discurilor în mişcare pentru memorarea datelor au devenit memoriile pur electronice de tip *Solid-state drive* (SSD), care neavând piese în mişcare sunt mult mai rapide, dar şi mai scumpe. Una din formele de implementare sunt cardurile de memorie de ex. de tip CF, MD, MMC, SD, SDHC, microSDHC, SM, USB stick şi altele.



Sisteme de operare. Microsoft Windows

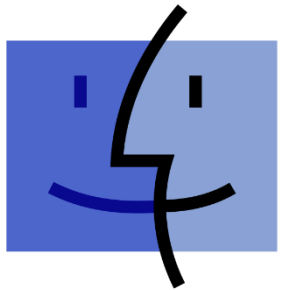
Un **sistem de operare**, prescurtat **SO** (engleză *Operating system*, prescurtat ***OS***), reprezintă un produs de tip software care este parte componentă a unui sistem, echipament sau aparat computerizat, și care se ocupă de gestionarea și coordonarea activităților acestuia. Sistemul computerizat poate fi un computer, o stație de lucru (*workstation*), un server, un PC, un *notebook*, un *netbook*, un *smartphone*, un aparat de navigație rutieră, un *e-book reader* sau unele aparate de uz casnic, precum și playerele multimedia. Sistemul de operare joacă și rolul de gazdă pentru aplicațiile care rulează pe echipamentul (hardware-ul) respectiv.



Sisteme de operare. Microsoft Windows

Exemple de sisteme de operare:

- Android-x86
- iOS (Apple)
- BlackBerry OS
- Windows mobile
- MS-DOS
- Linux
- Windows
- Mac OS



Mac OS

Sisteme de operare. Microsoft Windows

Microsoft Windows este numele unei serii de sisteme de operare create de compania Microsoft. Microsoft a introdus Windows pe piață pentru prima dată în noiembrie 1985, ca un supliment la MS-DOS, deoarece interfețele grafice erau din ce in ce mai apreciate.

Sitemele de operare Windows prezintă o serie de caracteristici pe care MS-DOS nu le are: (i) interfaţă grafică cu ulizatorul (GUI -graphical user interface), interfaţă ce prezintă icoane, ferestre, meniuri, boxe de dialog şi de control al aplicațiilor, (ii) input ce așteaptă (o serie de procese ce așteaptă să fie transferate in RAM), (iii) capacități multitasking, (iv) interschimb de date între aplicații. Interfața grafică permite utilizatorului să poată vedea pe ecran unelte necesare îndeplinii diferitelor sarcini.

Sisteme de operare. Microsoft Windows

Într-un sistem de operare multitasking este important ca tuturor aplicațiilor să li se aloce o porțiune de ecran a monitorului astfel încât utilizatorul să poată interacționa cu mai multe aplicații. În Windows orice aplicație are acces la o anumită parte a ecranului monitorului cu ajutorul unei *ferestre*. O fereastră este un dreptunghi ce poate avea dispozitive vizuale utile (meniuri, controale, bare de derulare) cu care utilizatorul poate controla o aplicație.

Sistemele de operare Windows sunt proiectate pentru a folosi mouse-ul. Deși de cele mai multe ori putem să folosim doar tastatura, folosirea mouse-ului este mult mai simplă. Poziția pe ecran a mouse-ului este indicată printr-un *mouse pointer numit și cursor* sau *cursor grafic*. De obicei cursorul este o săgeata, însă poate avea și altă formă în funcție de acțiunea curentă.

Sisteme de operare. Microsoft Windows

Mouse-ul se folosește pentru:

- a deschide şi a închide ferestre;
- a deschide şi a închide meniuri;
- a selecta texte, obiecte, comenzi ale unor meniuri, opțiuni din boxele de dialog;
- a parcurge documente prin defilarea textului pe ecran;
- a muta obiecte pe ecran.

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 3. Elemente de programare

- ***Algoritmi***
- ***Limbaj cod-mașină, limbaj de asamblare***
- ***Limbaje de programare. Etapele dezvoltării unui program***
- ***Limbajul Fortran 90***
- ***Convenții utilizate***

Algoritmi

Procesul elaborării de către om a programelor în scopul comunicării cu computerul pentru rezolvarea unor probleme se numeşte ***programare***. Programarea începe întotdeauna cu o *definiție* clară a problemei de rezolvat, adică un enunț cât mai precis, cu specificarea datelor de intrare, datele pe care le cunoaștem, și a datelor de ieșire, a rezultatelor pe care dorim să le aflăm. După ce problema a fost definită, pentru rezolvarea ei, alegem un algoritm.

Mai precis, un ***algoritm*** este o rețetă, formată dintr-un număr finit de pași de prelucrare, rețetă ce descrie procesul de transformare a datelor de intrare pentru a obține datele de ieșire.

Algoritmi

Orice algoritm trebuie să posede următoarele trei proprietăți :

- În primul rând, ***algoritmul trebuie să fie definit****;* fiecare pas de prelucrare este descris foarte precis și riguros, astfel încât pentru cel ce execută algoritmul să nu existe nimic de neînțeles și să nu apară nici un fel de ambiguitate în interpretare.
- În al doilea rând, ***algoritmul trebuie să fie eficace****;* în toate cazurile pentru care a fost creat, algoritmul conduce la rezultat după un număr finit de pași de prelucrare relativ simpli. Acesta proprietate se mai numește *realizabilitate potențială*.
- În al treilea rând, ***algoritmul trebuie să aibă caracter de masă****;* să permită rezolvarea oricărei probleme ce aparține unei anumite clase de probleme.

Limbaj cod-mașină, limbaj de asamblare

Limbajul algoritmic înțeles de către un computer se numește ***limbaj cod-maşină***, ***limbaj obiect*** sau ***limbaj intern***. Este un limbaj binar în care propozițiile limbajului se scriu numai cu simbolurile 0 și 1. În funcție de tipul procesorului fiecare computer, posedă un număr finit de instrucțiuni simple, *comenzi,* pe care le poate executa. Oricât de diferite ar fi comenzile de la un computer la altul, ele prezintă unele caracteristici generale. Orice comandă conține un *cod al operației* ce trebuie executată şi de obicei o *adresă*.

Codul operației poate indica: adunare, scădere, înmulțire, împărțire, comparație, citire din memorie, depunere în memorie, salt condiționat sau necondiționat, etc.

Adresa indică locul de unde se ia informația ce trebuie prelucrată sau locul unde trebuie depus rezultatul (memorie sau registre).

Limbaj cod-mașină, limbaj de asamblare

Pentru a uşura munca de scriere a unor programe s-au inventat *limbajele de asamblare*, în care codul operației se reprezintă printr-o mnemonică (*se referă la memorie; care ajută memoria*), iar adresele binare se înlocuiesc cu adrese simbolice. Programul sursă scris în limbajul de asamblare este tradus în mod automat de computer în limbajul obiect (limbajul binar) cu un program special numit *assembler;* Procesul de traducere poartă numele de ***asamblare***.

Până la sfârșitul anilor '50 programele se scriau doar în limbajele de asamblare sau în limbajele cod-maşină. Această activitate se numește de obicei *codificare,* iar persoana ce o practică se numește *codificator*.

Cu apariția unor computere mai rapide și cu memorie mai mare, dificultățile de codificare au crescut și a devenit evident că nu este rațional ca această muncă obositoare de codificare să o facă omul.

Limbaje de programare

Tendințele de a înlătura insuficiențele limbajelor de asamblare şi ale limbajelor cod-maşină au condus la crearea *limbajelor de programare,* sau cum se mai spune, a *limbajelor de programare de nivel înalt*.

Cu ajutorul acestor limbaje de programare, *procesul dezvoltării unui program* a fost redus la următorii patru pași:

- *Pasul* 1 - ***Crearea și editarea***. Programatorul lansează în execuție la computer un program numit *editor de texte* și redactează de la tastatură un *fișier sursă* ce conține textul programului scris în limbajul de programare. Crearea și editarea se pot face separat; mai întâi programatorul își concepe programul folosind hârtie și creion și apoi, cu editorul de texte, îl rescrie de la tastatură creând fișierul sursă. Sau, programatorul, pur și simplu, concepe programul și în același timp îl și editează.

Limbaje de programare

- *Pasul 2*. La computer se lansează în execuție ***compilatorul***. La intrare, compilatorul are fișierul sursă și la ieșire se obține fișierul *obiect* cu rezultatul traducerii, adică programul în limbaj binar. Fișierul obiect se obține doar în cazul în care toate instrucțiunile fișierului sursă au respectat întocmai regulile sintactice ale limbajului de programare. Dacă în fișierul sursă există greșeli de sintaxă, compilatorul generează doar un fișier cu o listă de erori de compilare.
- *Pasul* 3. La computer se lansează în execuție ***editorul de legături*** (linker). Linker-ul combină unul sau mai multe fișiere obiect cu fișiere obiect din bibliotecile computerului și generează *programul executabil*.
- *Pasul* 4. La computer se lansează în execuție programul executabil. Dacă nu există erori programul generează fișierul (fișierele) *cu* ***rezultate*** .

Limbajul Fortran 90

Primul limbaj de programare care a fost standardizat este limbajul Fortran. Cuvântul *FORTRAN* este un acronim de la cuvintele englezești "**FOR**mula **TRAN**slation" ceea ce înseamnă "traducător de formule". Limbajul Fortran a fost conceput astfel încât să fie convenabil rezolvării problemelor științifice inginerești, probleme în care apar *formule matematice*. **Fortran** este un limbaj de programare născut în anul 1950 și care este încă folosit după mai bine de jumătate de secol de existență.

De-a lungul timpului, acest limbaj de programare a fost în continuu îmbunătățit ajungându-se la varianta **Fortran 2015** ce va fi lansat la jumătatea anului 2018.

Convenţii utilizate

Pentru a descrie în mod economic şi precis sintaxa limbajelor de programare se foloseşte de obicei un ***metalimbaj***, un limbaj de descriere a limbajelor. În metalimbaj propoziţii ale limbajelor naturale se înlocuiesc cu nişte notaţii, numite *metasimboluri*. Cele mai des folosite sunt metasimbolurile introduse de Backus şi Naur. Pentru prezentarea limbajului Fortran 90 vom folosi notaţii Backus-Naur modificate.

Facem următoarele convenţii:

1. Cuvintele scrise cu caractere mari cu fontul ARIAL desemnează cuvinte cheie cu o semnificaţie specială pentru limbajul Fortran. Ele sunt o parte componentă a instrucţiunilor în afară de cazul în care aceste cuvinte sunt închise între paranteze pătrate [ ].

Convenții utilizate

2. Literele mici şi cuvintele scrise cu caractere mici *italice* (adesea prescurtat) reprezintă clase sintactice pentru care entitățile sintactice specifice trebuie substituite cu informația dată de utilizator.

3. Metasimbolurile folosite sunt următoarele:

[ ]	- un element opțional.
[ ]...	- include un element opțional repetat; poate să apară de zero sau mai multe ori.
este	- introduce o definiție a unei clase sintactice.
{a1|a2}	- indică o alegere între elementele a1 și a2
▪	- continuă o regulă sintactică

4. Caracterul blanc (spațiu liber) se reprezintă cu caracterul *b* (italic).

Convenții utilizate

Exemplu

constantă_întreagă este *cif* [*cif*] ...

unde *cif* este o cifră. De aici rezultă că o constantă întreagă se poate scrie:

cif

cif cif

cif cif cif cif

cif cif cif cif cif

Dacă înlocuim *cif* cu constante concrete, atunci exemplu va arăta așa:

3

29

12345

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 4. Tipuri de date intrinseci

- ***Date scalare***
- ***Date întregi***
- ***Date logice***
- ***Date reale***
- ***Necesități de stocare***

Date scalare

În limbajul Fortran pentru a descrie datele ce urmează a fi prelucrate se folosește terminologia ***obiect - date*** (data object). Datele cele mai simple sunt ***datele scalare***, care sunt date fără structură.

O dată scalară prezintă o ***notație*** şi o ***valoare***. În timpul execuției programului fiecare dată scalară este stocată într-un anumit mod într-o celulă de memorie. În limbajele de programare există şi ***date compuse*** ce prezintă o altă structură. De exemplu, în Fortran, datele compuse sunt tablourile.

Fiecărei date i se asociază un ***tip***; tipul datei este determinat de natura ei, de notaţia folosită pentru indentificare, de modul de stocare în memorie şi de modul de efectuare a operaţiilor.

Date scalare

Majoritatea limbajelor de programare posedă anumite ***tipuri de date intrinseci***, adică anumite tipuri de date predefinite: date întregi, date reale, date logice şi date caracter. Limbajul Fortran acceptă 5 tipuri de date intrinseci: întregi, reale, complexe, logice şi caracter.

Într-un program cu datele de un anumit tip putem să efectuăm diferite sarcini:

- pentru a calcula, folosim date reale sau complexe
- pentru a număra, folosim date întregi
- pentru a lua decizii, folosim date logice
- pentru a explica, folosim date caracter.

Date întregi

Pentru un computer dat ce lucrează cu un anumit compilator ***datele de tip întreg***, ***datele întregi***, sau prescurtat, ***întregii***, sunt elemente ale sistemului numerelor întregi. ***Sistemul numerelor întregi*** este o submulţime de *n* numere cuprinse într-un anumit interval al mulțimii numerelor întregi binecunoscute din matematică:

$$min \leq n \leq max$$

Pentru a descrie reprezentarea și comportarea datelor de un anumit tip se folosește un ***model de reprezentare***. În Fortran se folosește următorul ***model de reprezentare a datelor întregi***:

$$s \times \sum_{k=0}^{q-1} x_k \cdot B^k$$

unde *B* este *radix* (baza), un număr mai mare decât 1, *q* este un întreg pozitiv numit *digits* care reprezintă *numărul de cifre semnificative ale bazei*, *s* reprezintă semnul (+ sau -).

Date logice

Datele de *tip logic,* sau de *tip Boolean* definesc registrul mărimilor logice ce nu pot avea decât doua valori **TRUE** (adevărat) şi **FALSE** (fals). Numele "Boolean" provine de la numele lui George Boole (1815-1864) care a pus bazele algebrei logice. Cu datele logice se pot efectua ***operaţiile logice*** :

∨ disjuncţie, OR, sau	∧ conjuncţie, AND, şi	¬ negaţie, NOT, nu

Valorile expresiilor $p \vee q$, $p \wedge q$ şi $\neg p$ sunt definite în continuare unde p şi q sunt operanzi logici.

p	q	$p \vee q$	$p \wedge q$	$\neg p$
False	False	False	False	True
True	False	True	False	False
False	True	True	False	True
True	True	True	True	False

Date logice

Din definițiile operațiilor logice prezentate anterior, putem demonstra următoarele legi:

1. Legile de comutativitate

$$p \vee q = q \vee p$$
$$p \wedge q = q \wedge p$$

2. Legile de asociativitate

$$(p \vee q) \vee r = p \vee (q \vee r)$$
$$(p \wedge q) \wedge r = p \wedge (q \wedge r)$$

3. *Legile de distributivitate*

$$(p \wedge q) \vee r = (p \vee r) \wedge (q \vee r)$$
$$(p \vee q) \wedge r = (p \wedge r) \vee (q \wedge r)$$

4. *Legile lui De Morgan*

$$\neg (p \vee q) = \neg p \wedge \neg q$$
$$\neg (p \wedge q) = \neg p \vee \neg q$$

Date reale

La baza analizei matematice se află multimea ***R*** a numerelor reale. Computerele nu pot lucra cu numere reale, ele pot lucra numai cu aproximații finite ale acestora. Există mai multe metode de aproximație a numerelor reale cu reprezentări finite. Metoda cea mai folosită este aproximația numerelor reale prin *sistemul numerelor reale* numit şi ***sistemul numerelor cu virgulă flotantă*** (floating-point system). În programare *datele reale* sunt elemente ale sistemului $\hat{R}$ al numerelor cu virgulă flotantă. În sistemul numerelor cu virgulă flotantă un număr real x se aproximează cu un număr $\hat{x}$ ce se poate scrie cu ajutorul a două numere întregi M şi E , fiecare din ele conținând un număr finit de cifre, astfel:

$$\hat{x} = M \times B^E$$

B se numeşte ***baza*** reprezentării sistemului numerelor cu virgulă flotantă, M ***mantisă*** iar E se numeşte ***exponent***.

Date reale

Denumirea de "virgulă flotantă" provine de la faptul că același număr se poate scrie mutând poziția virgulei şi ajustând corespunzător exponentul. De exemplu, dacă $B = 10$, avem $10{,}56 \times 10^3 = 1{,}056 \times 10^4 = 0{,}1056 \times 10^5 = 1056 \times 10^1$.

Mulțimea $\hat{R}$ nu este o mulțime infinită. Numere din această mulțime nu sunt dispuse în mod uniform pe axa numerelor reale. Cu datele reale se pot efectua operațiile aritmetice, însă aritmetica numerelor cu virgulă flotantă, diferă de aritmetica numerelor reale. Nu este posibil să descriem în mod absolut exact operațiile aritmetice efectuate cu date reale.

Rezultatele calculelor cu date reale, depind de tipul problemei şi de algoritmul ales. În cel mai bun caz rezultatele obținute vor fi aproximații cu erori obligatorii. Evaluarea acestor erori, ce sunt consecința înlocuirii conținutului real cu o mulţime finită de reprezentanți, este o problemă dificilă şi este obiect al *analizei numerice*

Necesități de stocare

Compilatorul FTN 90 (Nag, 1997) acceptă următoarele tipuri intrinseci de date scalare:

i) întreg: INTEGER, INTEGER ([K1ND =] 1), INTEGER([KIND =] 2) şi INTEGER ([KIND =] 3)

ii) real: REAL, REAL ([KIND =] 1) şi REAL ([KIND =] 2)

iii) complex: COMPLEX, COMPLEX ([KIND =] 1), COMPLEX ([KIND =] 2)

iv) logic: LOGICAL, LOGICAL ([KIND =] 1), LOGICAL ([KIND =] 2), LOGICAL ([KIND =] 3)

v) caracter (CHARACTER *n)

Compilatorul FTN 90 impune următoarele cerințele de stocare:

Necesități de stocare

Tip	Bytes
INTEGER	4
INTEGER (KIND = 1)	1
INTEGER (KIND = 2)	2
INTEGER (KIND = 3)	4
REAL	4
REAL (KIND = 1)	4
REAL (KIND = 2)	8
LOGICAL	4
LOGICAL (KIND = 1)	1
LOGICAL (KIND = 2)	2
LOGICAL (KIND = 3)	4
COMPLEX (KIND = 1)	8
COMPLEX (KIND = 2)	16
CHARACTER*n	n

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 5. Atomi lexicali. Expresii

- ***Setul de caractere Fortran***
- ***Nume***
- ***Constante***
- ***Constante întregi, reale, complexe, logice şi caracter***
- ***Constante numite. Instrucţiunea PARAMETER***
- ***Variabile. Declararea tipului***
- ***Expresii. Expresii aritmetice***
- ***Expresii caracter***
- ***Expresii de relaţie***
- ***Expresii logice***

Setul de caractere Fortran

Limbajul Fortran este un limbaj scris. Acest limbaj posedă un *alfabet* format din *caractere alfanumerice* şi *caractere speciale*.

Definiţii:

- *caracter este* { *caracter_alfanumeric*| *caracter_special}*
- *caracter_alfanumeric este* { *literă*| _ | cifră}
- *literă este* {A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|■ ■ U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|■ ■ v|w|x|y|z}
- cifră *este* {0|1|2|3|4|5|6|7|8|9}
- *caracter special este* {=|+|-|*|/|(|)|,|.|'|"|:|!|%|&|;|<|>|?|$}

Astfel, setul de caractere Fortran este format din cele 26 litere mari şi mici ale alfabetului englezesc, cele 10 cifre arabe 0, ... ,9, semnul subliniere şi un număr de caractere speciale. În orice context compilatorul Fortran interpretează literele mici ca fiind mari.

Atomi lexicali. Separatori

Atomii lexicali sunt cuvinte formate din caracterele alfabetului care, în timpul compilării, pot primi o anumită semnificaţie în funcţie de contextul în care apar. Astfel, atomii lexicali, sunt cele mai mici unităţi ale unei instrucţiuni Fortran care au o anumită semnificaţie. Atomii lexicali pot fi separatori, delimitatori, cuvinte cheie, nume, etichete, constante şi operatori.

Separatorii sunt:

/ ( ) (/ /) = => , : :: ; %

Separatori pot fi şi *blancurile* (spaţiile libere). O constantă sau o etichetă poate fi despărţită de un cuvânt cheie adiacent prin unul sau mai multe blancuri. Un şir de blancuri este echivalent cu un singur blanc. Aceasta ne permite să utilizăm blancurile cum dorim pentru a îmbunătăţi aspectul textului Fortran.

Nume

Un exemplu de atom lexical este ***numele***, sau, cum se mai numeşte uneori, ***nume simbolic***.

Definiţie

nume este *lit* [*char_alfanum*]...

- unde *lit* este literă iar *char_alfanum* este caracter alfanumeric. Numărul maxim de caractere este 31.
- *Exemplul 1*

ION, SUMA, X3X4, ARIA, W65, An_universitar

- *Exemplul* 2 Nume incorecte sunt:

a mic - conţine un blanc

2bcd - primul caracter nu este o literă

+beta - conţine + care nu este caracter alfanumeric

Constante

O constantă este o dată ce îşi păstrează aceeaşi valoare în decursul execuției unui program. Există ***constante literale*** şi ***constante numite***.

Constantele literale se marchează cu o notație ce indică valoarea constantei. În funcție de tipurile intrinseci constantele se împart în constante întregi, reale, complexe, logice şi caracter.

Constantele numite se notează cu un nume și se introduc cu instrucțiunea PARAMETER sau ca variabile cu atributul PARAMETER.

Constante întregi

Definiții

- *const_lit_int_s* *este* [*s*] *șir_cif* [*_kind*]
- *const_lit_int* *este* *șir_cif* [*_kind*]
- șir_cif *este* *cif* [*cif*]...
- *s* *este* {+|-}
- *kind* *este* {*șir_cif* | *nume_const_scal_înt*}

Aici *const-lit-int-s* înseamnă *constantă literală întreagă cu semn*, şir_cif înseamnă şir de cifre, *s* înseamnă semn, *cif* înseamnă cifră, iar *kind* este parametrul kind. Semnul este obligatoriu dacă este negativ și opțional dacă este pozitiv. Dacă parametrul kind lipsește (kind default), o constantă literală întreagă se va scrie ca un număr întreg cu sau fără semn.

Constante întregi

Constantele sunt interpretate ca fiind în baza 10. O constantă fără semn se presupune că este pozitivă.

- *Exemplu*

Constante întregi

0 1 5320 +25 -3000

Limbajul Fortran permite scrierea unor constante întregi pozitive şi în alte baze diferite de 10. Astfel, avem *constante binare* (baza 2), *constante octale* (baza 8) şi *constante hexazecimale* (baza 16).

Constante reale

Constantele reale în Fortran reprezintă date reale.

Definiții

- *const_lit_reală_s* *este* [*s*] *const-lit-reală*
- *s* *este* {+|-}
- *const_lit_reală* *este* {semnif [E expo] [_kind] | *șir_cif* E ■ ■ expo [_*kind*]}
- semnif *este* {șir_cif. [șir_cif] | .șir_cif}
- *expo* *este* *șir_cif_cu_semn*
- *kind* *este* {şir_cif|*nume_constantă_scalară_întreagă*}

Aici *const_lit_reală_s* înseamnă *constantă literală cu semn, semnif* înseamnă partea semnificativă, E este litera exponent, *expo* înseamnă exponent, şir_cif înseamnă şir de cifre, *kind* este parametrul kind. Semnul - trebuie să apară înaintea unei constante negative; semnul + este opțional în fața unei constante pozitive.

Constante reale

Definiția conține două forme de scriere a constantelor literale reale. Prima formă se numește *forma pozițională*, numită şi *forma în virgulă fixă*, pentru că valoarea fiecărei cifre este determinată de poziția relativă fată de punctul zecimal. În forma poziționalăfără parametru kind o constantă literală reală este:

const_reală_posit *este* {[*s*] *int*. [*frac*] | [*s*] .*frac*}

unde *int* este un șir de cifre ce reprezintă partea întreagă, iar *frac* este un șir de cifre ce reprezintă partea fracționară. Constanta poate să nu aibă cifre în stânga sau în dreapta punctului zecimal, însă partea fracționară și partea întreagă nu pot lipsi simultan.

- *Exemplu*

12. +12. +12.0 0.534 .53 + 0.534 + .534 -3.0 30.02

Constante reale

A doua formă pentru constantele literale reale este *forma exponențială* sau *forma cu virgulă mobilă*. În forma exponențială fără parametru kind o constantă literală reală are forma:

cons_real_exp este *mantisa E expo*

unde *mantisa este* { [s] şir_cifre. [şir_cifre] | [s] . șir_cifres | [s] şir_cifre}

ceea ce înseamnă că mantisa este partea semnificativă. Litera exponent E înseamnă “înmulțit cu 10 la puterea". Exponentul *expo* reprezintă o putere a lui 10 cu care trebuie înmulțită constanta precedentă, întreagă sau reală.

Termenul "virgulă mobilă" provine de la posibilitatea de a muta virgula la mantisă printr-o ajustare corespunzătoare a exponentului.

- *Exemplu:* Numărul 15400 se poate scrie astfel

$$1.54 \times 10^4 = 15.4 \times 10^3 = 0.154 \times 10^5 = 0.0154 \times 10^6 = 154 \times 10^2$$

Constante reale

în formă exponenţiala acest număr îl putem scrie ca o constantă literală reală astfel:

1.54 E04 15.4 E3 0.154 E5 .0154 E6 154. E2 154 E2

Constante complexe

O *constantă literală complexă* reprezintă un număr complex, adică o pereche ordonată de date reale.

Definiţie:

const_lit_complx.	*este*	*(real, imag)*
real	*este*	*{const_lit_int_s|const_lit_real_s}*
imag	*este*	*{const_lit_int_s|const_lit_real_s}*

unde *const_lit_complx* înseamnă *constantă literală complexă*, *const_lit_int_s* înseamnă constantă literală întreagă cu semn, *const_lit_real_s* înseamnă constantă literală reală cu semn, real reprezintă *partea reală a constantei complexe* iar *imag* reprezintă *partea imaginară a constantei complexe*.

- *Exemplu:* Numărul complex 2+3i se poate scrie ca o constantă literal complexă astfel:

(2, 3) (2.,3.) (2._1, 3._1) (2._2, 3.) (2._2, 3._2)

Constante logice

O *constantă logică* specifică una din valorile logice, adevărat sau fals.

Definiţie:

const_lit_logic este { .TRUE. [_*kind*] | .FALSE. [_*kind*] }

unde *const_lit_logic* înseamnă constantă literală logică, .TRUE. înseamnă valoarea "adevărat", iar .FALSE. înseamnă valoarea "fals". Parametrul kind este opțional.

- *Exemplu*

.TRUE. .TRUE._3 .TRUE._2 .TRUE._1

Constante caracter

O *constantă caracter* reprezintă un șir de caractere tipăribile ale setului de caractere ce aparține procesorului.

Definiție:

const_lit_caracter este {[knd_] ‘char [char]...’ | [knd_] “char [char]...”}

unde *const_lit_caracter* este *constantă literală caracter*, *char* este unul din caracterele tipăribile, *knd* este parametrul kind. Apostrofurile și ghilimelele din stânga și din dreapta constantei sunt *delimitatori* și nu sunt incluse în valoarea constantei.

Exemplul 1

Constantele 'casa' și "casa"

au valoarea casa, adică șirul de caractere cuprins între delimitatori.

Constante caracter

Exemplul 2

Constantele 'un om' şi 'unom' sunt diferite.

În constantele caracter există diferență între literele mari și mici.

Exemplul 3

Constanta "Ac" este diferită de constanta "ac".

Delimitatorii de un fel pot fi incluși într-un șir mărginit de delimitatori de celălalt fel.

Exemplul 4

Delimitatori incluși în șirul de caractere: "a spus 'salut'" 'a spus "salut" '

Constante numite. Instrucțiunea PARAMETER

În Fortran putem avea şi *constante numite,* constante pe care le notăm cu un nume. Pentru a specifica că este vorba de o constantă numită folosim instrucţiunea PARAMETER.

Definiţie:

instr_parameter este PARAMETER (*name* = *expr* [, *name* = *expr*]...)

unde *name* este un nume de constantă numită, iar *expr* este o expresie ce poate conține doar nume dintr-o instrucţiune PARAMETER anterioară.

Exemplul 1

Instrucțiunea PARAMETER (pi = 3.14159)

definește constanta numită pi cu valoarea 3.14159.

Constante numite. Instrucțiunea PARAMETER

Exemplul 2

Instrucțiunile PARAMETER (i = 10, j = 20)

PARAMETER (ip2 = i + 2)

definesc constantele numite i, j, ip2 ce au valorile 10, 20, 12.

Constantele numite pot fi definite şi cu instrucţiuni de declarare a tipului ce folosesc atributul PARAMETER.

Exemplul 3

Instrucțiunea din exemplul 1 este echivalentă cu instrucțiunea

REAL PARAMETER pi = 3.14159

Variabile. Declararea tipului

O variabilă scalară este un obiect scalar notat cu un ***nume***. Pentru procesor variabila scalară este o celulă de memorie a cărei adresă simbolică este numele variabilei, celulă în care se stochează valoarea variabilei. În decursul execuției programului variabila își poate schimba valoarea.

Majoritatea variabilelor nu au nici o valoare atunci când începe execuția programului. Se spune că aceste variabile sunt *nedefinite*. Excepție o fac variabilele care sunt inițializate cu instrucțiunea DATA sau cu instrucțiuni de declarare a tipului; se zice că aceste variabile sunt *definite*. O variabilă poate căpăta o valoare sau își poate schimba valoarea la execuţia unei instrucțiuni de atribuire sau a unei instrucţiuni de citire. Astfel, o variabilă poate căpăta diferite valori la momente diferite de timp şi în anumite circumstanţe poate deveni nedefinită.

Variabile. Declararea tipului

Pentru a specifica tipul şi unele atribute ce descriu modul de utilizare în program a variabilelor se folosesc instrucţiuni de declarare a tipului.

Şabloane *pentru declararea* ***tipului întreg****:*

- INTEGER var [,var]...
- INTEGER var = expr [, var = expr]...
- INTEGER, PARAMETER var = expr [, var = expr]...

Exemplu

- INTEGER i1
- INTEGER I, J, K
- INTEGER i1 =10
- INTEGER X1=1, X2=2, X3=3, X4=4
- INTEGER, PARAMETER masa = 2

Variabile. Declararea tipului

Şabloane *pentru declararea* ***tipului real****:*

- REAL var [, var] ...
- REAL var = expr [, var = expr]...
- REAL, PARAMETER var = expr [, var = expr]...

Exemplu

- REAL a, b, c, d
- REAL gama
- REAL beta = 3.0
- REAL, PARAMETER pi = 3.14159

Variabile. Declararea tipului

Șabloane *pentru declararea* ***tipului complex****:*

- COMPLEX var [, var]...
- COMPLEX var = expr [, var = expr]...
- COMPLEX, PARAMETER var = expr [, var = expr]...

Exemplu

- COMPLEX AX, BX
- COMPLEX, PARAMETER s = (1.,0.)

Variabile. Declararea tipului

Șabloane *pentru declararea* ***tipului logic****:*

- LOGICAL var [, var]...
- LOGICAL var = expr [, var = expr]...
- LOGICAL, PARAMETER var = expr [, var = expr]...

Exemplu

- LOGICAL test
- LOGICAL g1, g2, g3
- LOGICAL h1 = .TRUE. , h2 = .FALSE.

Variabile. Declararea tipului

Şabloane *pentru declararea* ***tipului caracter****:*

- CHARACTER [([LEN =] len)] var [,var]...
- CHARACTER * len [,] var [, var]...
- CHARACTER var [, var] ...
- CHARACTER var = expr (, var = expr]...

Exemplu

- CHARACTER (80) LINIE
- CHARACTER (LEN = 80) LINIE
- CHARACTER*80 LINIE
- CHARACTER A, S, C
- CHARACTER*1 K1='o', K2='p', K3='q'
- CHARACTER r1*2 = "xy", r2*2="xy"

Declararea implicită

Într-o unitate de program tipul variabilelor se poate descrie prin instrucțiuni de declarare a tipului. În lipsa instrucțiunilor de declarare a tipului (default), prin convenție, numele ce încep cu una dintre literele I, J, K, L, M, N, litere mici sau mari, reprezintă variabile de tip întreg, iar numele ce încep cu orice altă literă reprezintă variabile de tip real.

Pentru a schimba asociația între tip și primul caracter al numelui se folosește instrucțiunea IMPLICIT.

Exemplu

```
IMPLICIT REAL (D)
IMPLICIT REAL (A - C, S, T, U - Z)
IMPLICIT INTEGER I, J, K, O
IMPLICIT CHARACTER*2 (W)
IMPLICIT LOGICAL (L)
```

Declararea implicită

Instrucțiunile specifică că toate numele ce încep cu litera D sunt numere reale; toate numele ce încep cu una din literele A, B, C, S, T, U, V, W, X, Y, Z reprezintă variabile reale; toate numele ce încep cu una din literele I, J, K, O reprezintă variabile întregi, toate variabilele ce încep cu litera W sunt variabile caracter cu lungimea 2 și numele ce încep cu litera L reprezintă variabile logice.

Astfel, instrucțiunea IMPLICIT atribuie tipul specificat tuturor numelor ce încep cu litera specificată sau cu una din literele din domeniul de litere specificat. Dacă se folosește **IMPLICIT NONE**, atunci toate numele variabilelor trebuie să apară în mod explicit în instrucţiuni de declarare a tipului; omiterea unui nume de variabilă va cauza o eroare de compilare.

Expresii aritmetice

În Fortran o expresie indică a serie de calcule sau de manipulări ale datelor. O expresie este formată din operanzi (datele de prelucrat) şi operatori (operații de prelucrare).

O expresie scalară numerică are ca rezultat o valoare de tip întreg, real sau complex. Expresiile scalare numerice se construiesc cu operanzi numerici de tip real, întreg sau complex, paranteze şi operatori aritmetici.

Operatorii aritmetici sau *operatorii numerici* sunt:

- ** - ridicare la putere
- * - înmulțire
- / - împărțire
- - - scădere
- + - adunare

Expresii aritmetice

Exemplu

Operatori binari: a+b; a-2; c*d; *x/y*

Exemplul

În paranteze + şi - sunt operatori unari:

a + (-2.); c *(+D); y * (-3)

Precedența

Operator	Precedența	În context de precedență egală
**	Cea mai mare	dreapta la stânga
* sau /	-	stânga la dreapta
+ sau – unar	-	stânga la dreapta
+ sau – binar	-	stânga la dreapta

Expresii caracter

Expresiile scalare caracter conțin operanzi ce pot fi: constante caracter, variabile caracter. Există un singur operator caracter, *operatorul de concatenare //* ce are ca efect combinarea a doi operanzi caracter într-un singur rezultat caracter.

Operanzii trebuie să aibă același parametru kind. Parametrul lungime al concatenării reprezintă suma lungimilor operanzilor.

Exemplu
'AB' // 'CDE'
produce constanta caracter 'ABCDE'. Operanzii 'AB', 'CDE' au lungimile 2, respectiv 3 iar rezultatul va avea lungimea 5.

Expresii caracter

Parantezele nu afectează evaluarea unei expresii caracter.

Exemplu

Următoarele expresii caracter sunt echivalente:

("ABC" // "DE") // "F"

"ABC" // ("DE") // "F"

"ABC" // "DE" // "F'

Dacă un operand caracter dintr-o expresie caracter conține blancuri, blancurile sunt incluse în valoarea expresiei caracter.

Exemplu

"ABC*b*" //„DE*b*" //"F"

are valoarea:

"ABC*b*DE*b*F"

Aici cu *b* am notat caracterul blanc.

Expresii de relație

Expresiile de relație, numite și *expresii relaționale*, compară valorile a două expresii numerice sau caracter. *Operatorii de* relaţie, sau cum se mai numesc, *operatorii relaţionali* sunt:

Operator	Operația
.LT. sau <	Mai mic decât
.LE. sau <=	Mai mic decât sau egal
.EQ. sau ==	Egal
.NE. sau /=	Neegal
.GT. sau >	Mai mare decât
.GE. sau >=	Mai mare decât sau egal

Expresii logice

Operatorii logici sunt:

Operator	Operația
.NOT.	Negație
.AND.	Conjuncție
.OR.	Disjuncție
.EQV.	Echivalență
.NEQV.	Neechivalență

Rezultatul evaluării unei expresii logice este de tip logic, constanta logică .TRUE. sau constanta logică .FALSE. Operatorii .AND., .OR., .EQV., .NEQV. sunt operatori binari; ei se scriu între operatori de tip logic.

Expresii logice

Operatorul .NOT. este operator unar și precedă operandul. Operatorii .NOT., .AND. şi .OR au semnificația din logica matematică. Semnificația operațiilor .EQV. şi .NEQV. este arătată în continuare:

a	b	a.EQV.b	a.NEQV.b
TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE

Expresii logice

Precedența operatorilor logici este prezentată în continuare:

Operator	Precedența	În context de precedență egală
.NOT.	Cea mai mare	
.AND.	-	stânga la dreapta
.OR.	-	stânga la dreapta
.EQV. sau .NEQV.	-	stânga la dreapta

Când două operații consecutive sunt de precedență egală, operația din stânga se efectuează prima. Doi operatori .NOT. nu pot fi adiacenți. Operatorul .NOT. poate să apară lângă un alt operator logic.

Exemple de expresii logice

.NOT. a; a .OR. b .AND. .NOT. c;

((.NOT. d) .AND. . TRUE.) .NEQV. (a .OR. c)

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 6. Proceduri intrinseci

- ***Generalități. Notații***
- ***Exemple de funcții intrinseci***

Generalități. Notații

Într-un limbaj orientat către aplicații științifice este necesar ca funcțiile matematice, mai des utilizate, să fie înglobate în limbaj. În Fortran, acest lucru se realizează cu ajutorul ***procedurilor intrinseci***. Codurile binare ale procedurilor intrinseci sunt introduse în ***biblioteca sistemului Fortran***. Procedurile intrinseci se includ în mod automat în program în faza de editare a legăturilor.

În Fortran există patru clase de proceduri intrinseci: *funcții element, funcții de informare, funcții de transformare și subrutine.*

O *funcție element* este specificată pentru argumente scalare, însă se poate apela și pentru argumente tablou.

Generalități. Notații

O funcție de informare este o funcție al cărei rezultat depinde de statutul sau natura argumentului și nu de valoarea acestuia.

O funcție de transformare. Majoritatea funcțiilor de transformare au ca argument, sau argumente, de intrare un tablou, sau mai multe, iar argumentul de ieșire tot un tablou.

Standardul Fortran 90 prevede 113 proceduri intrinseci dintre care 108 funcții intrinseci și 5 subrutine intrinseci.

Utilizarea funcțiilor intrinseci este foarte simplă: în orice unitate de program în locul în care avem nevoie de valoarea unei funcții intrinseci vom scriem ***numele*** funcţiei şi în paranteze ***lista argumentelor actuale***, adică lista argumentelor pentru care dorim să evaluăm funcția.

Generalități. Notații

Argumentele actuale conform poziției lor în listă, trebuie să concorde ca tip și lungime cu argumentele din ***lista argumentelor formale*** a procedurii intrinseci.

Unele dintre funcțiile intrinseci sunt ***funcții generice*** ce acceptă argumente de mai multe tipuri, iar tipul rezultatului va coincide cu tipul argumentului.

În Fortran există 14 **funcții intrinseci element** ce *evaluează funcțiile matematice elementare*. Tipul și parametrul kind al rezultatului sunt cele ale primului argument al funcției, care, de obicei, este singurul argument.

Există 14 **funcții intrinseci element** ce *pot efectua diferite manipulări numerice simple*. Funcțiile numerice pot efectua și operații de conversie a tipului.

Generalități. Notații

Limbajul Fortran are mari posibilități de procesare a textelor datorită prezenței unui set puternic de proceduri intrinseci caracter. O porțiune de text scris într-un program se numește *string* (şir de caractere). În Fortran string poate fi valoarea unei constante caracter sau valoarea unei variabile caracter, adică un șir de caractere tipăribile ale procesorului.

Funcțiile caracter (element) se pot grupa în: funcții de conversie caracter-întreg, funcții de comparare lexicală și funcții de manipulare string. Există și funcții caracter non-element cum ar fi o funcție de informare şi funcții de transformare.

Mai există 7 **funcții de manipulare a datelor reale** care returnează valori legate de modelul de reprezentare a datelor reale.

De asemenea, există și o funcție ce convertește parametrii kind pentru datele logice.

Generalități. Notații

Următoarele tipuri de funcții sunt **funcțiile de informare numerică (9 funcții)** ce returnează valori legate de modelele de reprezentare a datelor întregi și a celor reale. Fiecare funcție are un singur argument ce poate fi un scalar sau un tablou. Rezultatul va fi un scalar iar valoarea argumentului nu este necesar să fie definită.

Limbajul Fortran permite să specificăm că un argument al unei proceduri posedă **atributul OPTIONAL**. Pot fi specificate ca opționale doar argumentele formale ale unei proceduri. Aceasta se poate face fie cu o instrucțiune de declarare ce conține atributul OPTIONAL, fie cu o instrucțiune OPTIONAL. În timpul execuției unei proceduri ce posedă un argument OPTIONAL este necesar să știm dacă s-a furnizat un argument actual pentru argumentul formal corespunzător.

Pentru aceasta putem folosi funcția intrinsecă *PRESENT* ce indică prezența argumentului opțional.

Generalități. Notații

Există și **proceduri de manipulare BIT**. Procedurile de manipulare bit sunt: funcția de informare bit BIT_SIZE, 10 funcții element de manipulare bit și subrutina MVBITS.

De asemenea, 3 **funcții kind**. Funcția KIND și două funcții de transformare, funcțiile SELECTED_INT_KIND și SELECTED_REAL_KIND.

O funcție de transfer ce permite ca datele de un tip să fie transferate altui tip fără a altera reprezentarea fizică.

În Fortran există **4 subrutine intrinseci non element**. Două dintre aceste subrutine sunt DATA_AND_TIME și SYSTEM_CLOCK ce returnează informație de la ceasul intern al PC-ului.

Exemple de funcții intrinseci

Un exemplu de funcție generice este funcția ***SQRT***, ce calculează rădăcina pătrată. SQRT (X) returnează valoarea funcției rădăcină pătrată $\sqrt{X}$ pentru X real sau complex. Dacă X este real rezultatul va fi ≥ 0.

Aceasta poate avea următoarele forme:

SQRT (3.) SQRT (3.D0) SQRT ((3. ,2))

Modul de scriere a acestor referinţe de funcție ne sugerează că ar fi vorba despre o singură funcție intrinsecă SQRT. În realitate, există 3 funcții intrinseci, una ce acceptă argument real, alta argument cu dublă precizie şi a treia argument complex.

Exemple de funcții intrinseci

Funcția exponențială și cea logaritmică:

- EXP (X) - returnează valoarea funcției exponențiale exp(X) pentru X real sau complex. Dacă X este complex, partea imaginară a rezultatului se va exprima în radiani.
- LOG(X) - returnează valoarea funcției logaritm natural ln(X) în baza e pentru X real sau complex. Dacă X este real rezultatul va fi pozitiv iar dacă X este complex atunci rezultatul va fi diferit de 0.
- LOG10(X) - returnează valoarea funcției logaritm în baza 10, log(X), pentru X real pozitiv rezultatul va fi pozitiv.

Exemple de funcții intrinseci

Funcții trigonometrice: SIN(X), COS(X), TAN(X), SINH(X), COSH(X), TANH(X), ASIN(X), ACOS(X), ATAN(X).

Funcțiile MIN și MAX:

- MIN(A1, A2 [,A3,]) - returnează minimul dintre 2 sau mai multe argumente întregi sau reale. Argumentele, toate de tip 1 sau toate de tip R, trebuie să aibă același parametru kind. Exemple:

  MIN(-8.0, 5.0, 1.0) = -8.0

  MIN(5, 0, -4, 2)=-4

- MAX(A1, A2 (,A3,]) - returnează maximul dintre 2 sau mai multe argumente întregi sau reale. Argumentele, toate de tip 1 sau toate de tip R, trebuie să aibă același parametru kind.

  Exemple

  MAX(-8.0, 2.0, 3.0,10.0) = 10. MAX(-1,2, -8, 5) = 5

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 7. Instrucțiuni de prelucrare

- ***Fișiere. Înregistrări***
- ***Instrucțiunea READ***
- ***Instrucțiunea WRITE***
- ***Instrucțiunea de atribuire***
- ***Scheme logice***
- ***Pseudocod***
- ***Teorema de structură. Construcții de control***

Fișiere. Înregistrări

Un fișier este o resursă a calculatorului folosită pentru înregistrarea discretă a datelor într-un dispozitiv de stocare. Există diferite tipuri de fișiere concepute pentru scopuri diferite.

Un fișier poate fi proiectat pentru a stoca o imagine, un mesaj scris, un videoclip, un program de calculator sau o mare varietate de alte tipuri de date. Unele tipuri de fișiere pot stoca mai multe tipuri de informații simultan.

Prin utilizarea unor programe, o persoană poate deschide, citi, schimba și închide un fișier. Fișierele computerului pot fi redeschise, modificate și copiate de nenumărate ori.

În mod tipic, fișierele sunt organizate într-un sistem de fișiere, care urmărește unde sunt acestea și le permite oamenilor să le acceseze.

Fișiere. Înregistrări

În sistemele de operare moderne, fișierele sunt organizate într-o rețea unidimensională de octeți. Formatul unui fișier este definit de conținutul său, deoarece un fișier este doar un recipient pentru date. Pe anumite platforme, formatul este de obicei indicat de extensia fișierului, fiind specificate regulile pentru modul în care octeții trebuie să fie organizați și interpretați.

De exemplu, octeții unui fișier text simplu (.txt în Windows) sunt asociați cu caractere ASCII sau UTF-8, în timp ce octeții fișierelor imagine, video și audio sunt interpretați altfel. Cele mai multe tipuri de fișiere alocă de asemenea câteva octeți pentru **metadate**, ceea ce permite unui fișier să transmită câteva informații de bază despre el însuși.

Instrucțiunea READ

Instrucțiunea de citire (READ) are forma

read vname [,vname]...

Și este alcătuită din cuvântul read ce înseamnă "citește" și o listă de variabile.

Exemplu: Instrucțiunea - read alfa – unde lui alfa îi este atribuită valoarea 204 se execută astfel: procesorul caută în memorie celula cu adresa simbolică alfa. Presupunem că nu găsește o astfel de celulă; atunci alege o celulă de memorie liberă, o numește alfa și în ea stochează valoarea 204. În urma execuției instrucțiunii read alfa variabila alfa a fost definită, astfel în memorie va există o celulă cu adresa simbolică alfa și în acea celulă este stocată valoarea variabilei, 204.

Instrucțiunea WRITE

Instrucțiunea de scriere (WRITE) are forma

write entitate [,entitate]...

Și este alcătuită din cuvântul write ce înseamnă „scrie" și o listă de entități. Fiecare entitate poate fi o variabilă sau o expresie.

Exemplu: Instrucțiunea - write kapa - se execută astfel: procesorul caută în memorie celula kapa, o consultă (află valoarea stocată acolo fără a o distruge și fără a-i altera conținutul) și realizează în fișierul de ieșire o înregistrare cu valoarea variabilei kapa; de exemplu, dacă valoarea variabilei kapa este 5, se creează o nouă înregistrare ce conține valoarea 5.

Evident instrucțiunea write nu are sens dacă în prealabil variabilele ce apar în entitățile din lista de ieșire nu au fost definite anterior.

Instrucțiunea de atribuire

Instrucțiunea de atribuire are forma

vname ← expr

unde vname este o variabilă, expr este o expresie iar ← reprezintă semnul de atribuire. Dacă expr este o expresie numerică, variabila vname este o variabilă numerică. Dacă expr este o expresie logică, vname este o variabilă logică. Dacă expr este o expresie caracter atunci vname este o variabilă caracter. Instrucțiunea se execută astfel: cu datele existente în memorie procesorul evaluează expr şi apoi rezultatul este stocat în celula cu numele vname.

Scheme logice

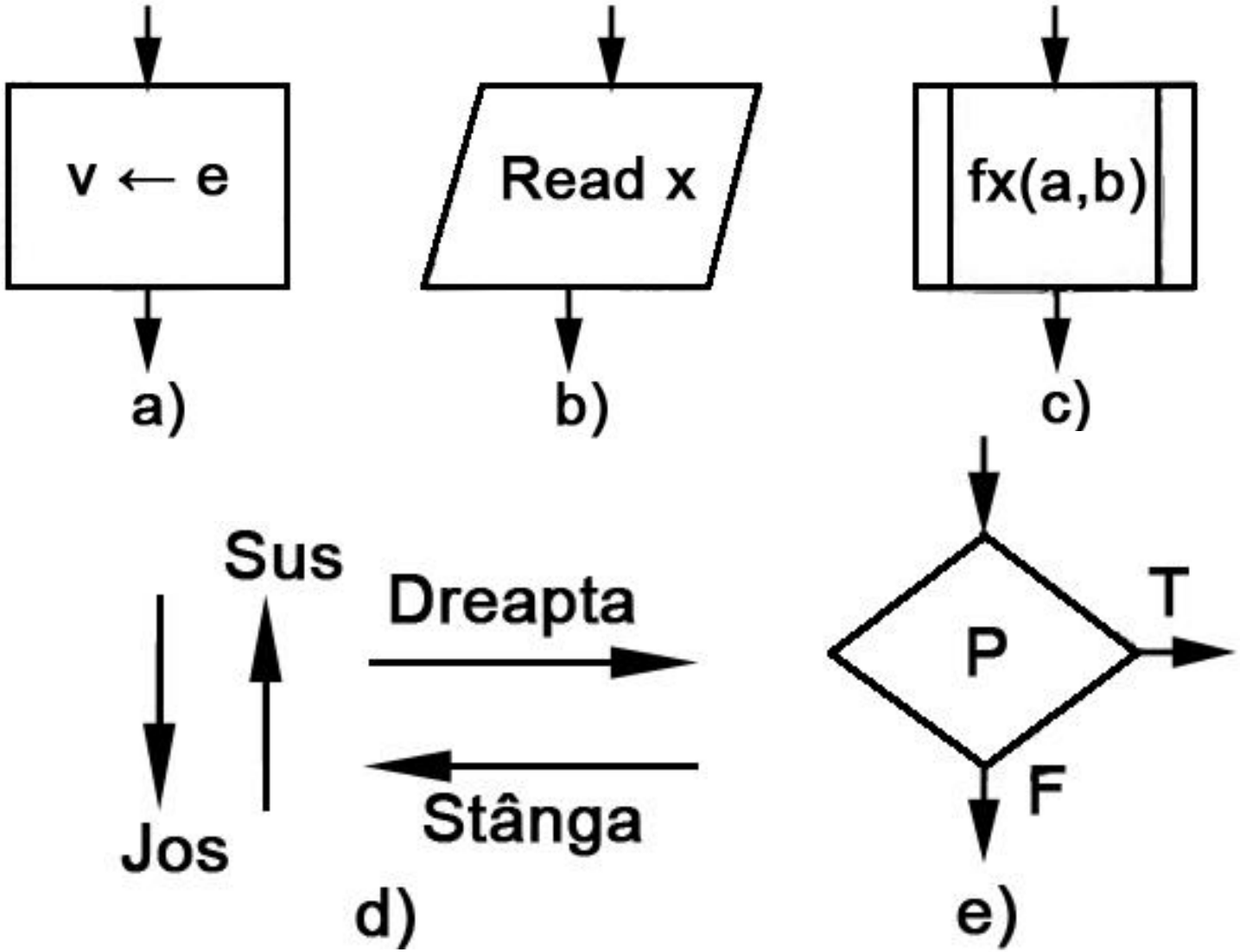
Descrierea logicii programului, adică instrucțiunile de prelucrare și ordinea lor de execuție se face de obicei cu ajutorul schemelor logice.

Schema logică este o reprezentare grafică bidimensională a algoritmului cu ajutorul unor simboluri speciale. Schema logică este mai intuitivă decât programul scris într-un limbaj de programare. De aceea erorile logice ale programului se descoperă mai repede pe schema logică.

Din punct de vedere matematic schema logică este un grafic ce poate avea noduri de trei feluri: ***nod funcțional, nod predicativ și nod de reunire***.

Nodul funcțional sau nodul de prelucrare se poate reprezenta ca în figura următoare (a-c). Acesta are o intrare și o ieșire și reprezintă o operație de prelucrare a informației.

Scheme logice



Scheme logice

Nodul predicativ are o intrare și două ieșiri (e). Predicatul P înscris în romb și reprezintă o condiție logică ce trebuie verificată. În funcție de rezultat, true (T) sau false (F) se alege una din ieșiri. **Nodul de reunire** se reprezintă printr-un cerculeț plin în care intră două linii și iese o singură linie. În nodul de reunire nu se efectuează nici o operație de prelucrare, nodul de reunire este o simplă reuniune cu două intrări şi o ieșire.

Simbolurile se pot grupa în simboluri de bază, simboluri adiționale și simboluri specializate de prelucrare.

Simbolul proces reprezintă procesul de execuție a unei anumite operații sau a unui grup de operații în urma căruia rezultă schimbarea formei sau valorii informației. Forma este dreptunghi cu raportul înălțime - lățime 2/3 (a).

Scheme logice

Simbolul intrare/ieşire reprezintă funcţia generală de input-output. Mediul suport de intrare/ieşire nu este specificat. Forma este paralelogram cu unghiul ascuțit de 75° și raportul înălțime-lăţime 2/3 (b).

Simbolul linie și vârf de săgeată arată direcția fluxului de informație în și succesiunea operațiilor. Se reprezintă prin orice segment de dreaptă ce trebuie să lege două simboluri. Direcția normală este sus-jos și dreapta-stânga (d).

Simbolul decizie se folosește pentru a reprezenta o decizie ce determină una din două alternative posibile. Forma este romb cu diagonala mare orizontală și cu raportul înălțime-lățime 2/3 (e).

Simbolul proces predefinit reprezintă una sau mai multe operații de prelucrare specificate în detaliu în altă parte (procedură, altă schemă logică). Forma este aceeași cu a simbolului proces la care se adaugă 2 linii verticale (c).

Pseudocod

În locul schemelor logice adesea se folosește pseudocodul. Pseudocodul este un limbaj asemănător limbajului natural care ne permite să descriem formal logica programului fără toate detaliile concrete ale unui limbaj de programare. Textul programului scris în pseudocod se aseamănă cu textul programului scris într-un limbaj de programare, dar este mai simplu. Pseudocodul se deosebește de limbajul de programare prin două caracteristici: i) nu are reguli sintactice formale, ii) poate exprima operații care nu sunt foarte precis definite.

Programul în pseudocod se scrie instrucțiune după instrucțiune, câte una pe un rând. Instrucțiunile le separăm prin punct și virgulă; aceasta înseamnă că instrucțiunea ce urmează se va executa după ce s-a terminat execuția instrucțiunii precedente.

Pseudocod

În afara de **linii-instrucțiuni** textul pseudocodului poate conține și **linii-comentariu**. În pseudocod liniile comentariu încep cu semnul exclamării "!". Comentariile nu sunt instrucțiuni, la execuția programului comentariile fiind ignorate. Comentariul este un text scris de programator pentru a face programul cât mai inteligibil.

Teorema de structură

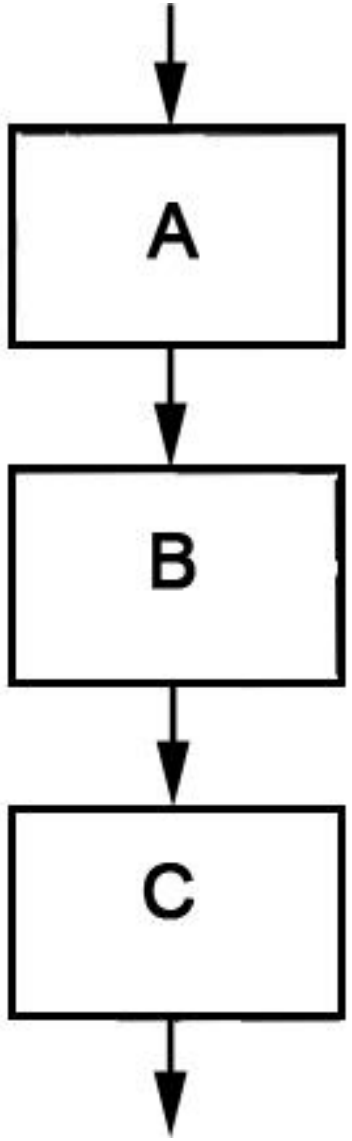
Instrucţiunile simple, citire, scriere și de atribuire, se execută în mod secvențial, una după alta, în ordinea în care sunt scrise în program. Programele conțin şi instrucțiuni de salt ce descriu o ordine de execuție diferită de cea secvențială: în funcție de îndeplinirea sau neîndeplinirea unor condiții se fac treceri în salt la instrucțiuni dinainte sau la instrucțiuni ce urmează. Prin urmare, în general, ordinea de execuție a instrucțiunilor nu coincide cu ordinea lor de scriere. În mod obișnuit fiecare programator tinde să-şi proiecteze programele folosind procedeul "încercare, eroare, corectare".

Astfel, s-a demonstrat teorema de structură potrivit căreia: *Orice program executabil este echivalent cu un program alcătuit doar cu trei tipuri de instrucțiuni simple (citire, scriere și atribuire) și trei tipuri de instrucțiuni compuse (secvența, selecția și iterația).*

Construcții de control

Instrucţiunile compuse sunt exemple de structuri de control, sau cum se mai numesc **construcții de control**.

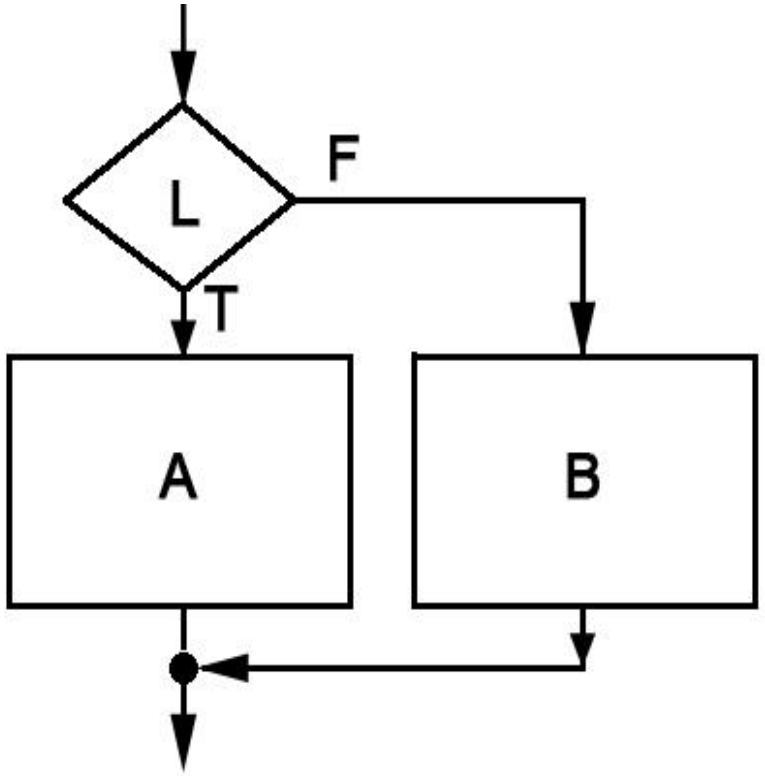
Secvența este o structură formată din două sau mai multe părți ce se execută fiecare o singură dată. Prin parte înțelegem fie o instrucțiune simplă, fie o instrucțiune compusă. Schema logică a unei secvențe este arătată mai jos. Secvența cu trei părți se execută astfel: se execută A, se execută B, se execută C și secvența ia sfârșit.



Construcții de control

Selecția este o structură de control cu două părți A și B din care se execută numai una în funcție de rezultatul unui test logic L:

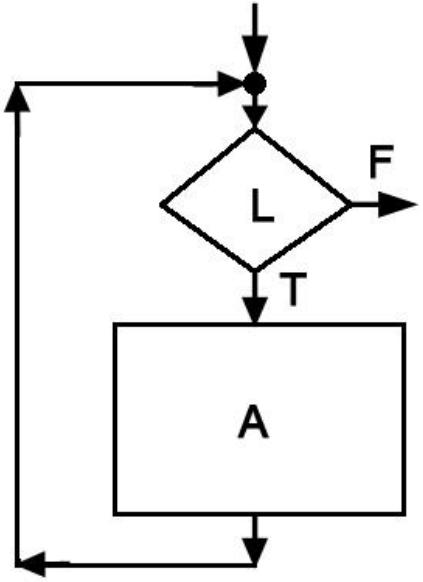
```
IF (L) THEN
        A
ELSE
        B
END IF
```



Modul de execuție al selecției este următorul: cu datele din memorie se calculează expresia logică L; dacă rezultatul este true se execută partea A și selecția ia sfârșit; dacă rezultatul este false se execută partea B şi selecția ia sfârşit.

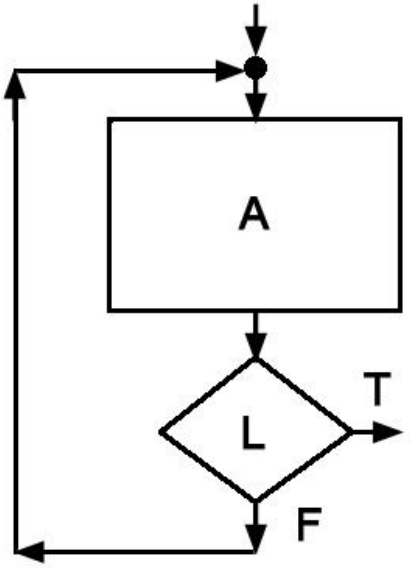
Construcții de control

Iterația este o structură de control cu o parte ce se execută de zero ori sau de mai multe ori în funcție de rezultatul unui test logic. Există două forme de iterație, iterație cu testul în capul iterației numită do while (stânga) şi iterație cu testul în coada iterației, iterație numită repeat until (dreapta).



do while (L)
A
enddo

repeat
A
until (L)



Construcții de control

Modul de execuție al iterației do while este următorul: cu datele din memorie se calculează expresia L. Dacă rezultatul este true atunci se execută A, se calculează L, dacă L este true se execută A, și așa mai departe, până când rezultatul devine false; atunci iterația ia sfârșit. Dacă la prima evaluare a expresiei logice rezultatul este false iterația ia sfârșit și partea A nu se execută niciodată. În programare iterația se mai numește buclă. Pentru ca iterația să se termine într-un număr finit de pași este necesar ca partea A să modifice valoarea unei variabile ce apare și în testul L. Dacă această condiție nu se îndeplinește avem de-a face cu o bucă infinită.

Construcții de control

Modul de execuție al iterației repeat until este următorul: se execută A, se evaluează expresia logică L, dacă rezultatul este false se execută din nou A, se evaluează L și așa mai departe până ce rezultatul devine true; atunci iterația ia sfârșit. Partea iterată A se execută în acest caz cel puțin o dată. Evident și buclele repeat until pot fi infinite, dacă la execuția după un număr finit de pași expresia logică L nu capătă valoarea true.

Pe plan mai general, programarea structurată acceptă și alte structuri de control, altele decât selecția, secvența și iterația. Un alt tip de structură de control este *Select Case* a cărei structură este prezentată în dreapta.

```
select case (e)
case (e1)
F1
case (e2)
F2
case (e3)
F3
case (en-1)
Fn-1
case default
Fn
end select
```

Construcții de control

Odată cu structura Case, deoarece rezultatul structurii nu corespunde modului natural de a gândi, trebuie să folosim instrucțiunea goto (sau go to), instrucțiune ce are sintaxa:

goto k

unde k este eticheta unei instrucțiuni. Eticheta este formată numai din cifre și se asociază unei instrucțiuni ce va fi referită de o altă instrucțiune.

Exemplu

```
10: A
    if (L) then
        B
        goto 10
    endif
```

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 8. Programe Fortran secvențe de instrucțiuni simple

- *Structura programelor Fortran*
- *Forma codului sursă*
- *Înregistrări formatate*
- *Input-Output cu formatare condusă de listă*
- *Compilarea, editarea legăturilor şi execuția*
- *Output formatat*

Structura programelor Fortran

Orice *program Fortran* este alcătuit din una sau mai multe *unități de program*. Unitate de program poate fi: *programul principal* (main), un *subprogram extern* (subrutină sau funcție), un *modul* sau un *bloc de date* (block data). Dacă programul este format dintr-o singură unitate de program, acea unitate, în mod obligatoriu, este programul principal.

Subprogramul extem se poate utiliza pentru a efectua prelucrări asupra unor entități care sunt disponibile subprogramului extern. Pentru a apela o subrutină externă se foloseşte instrucţiunea CALL. Atunci când este nevoie de valoarea unei funcții externe se va apela un subprogram funcție.

Modulul conține definiții de date, definiții de tip, definiții de procedură, numite *subprograme modul* ce pot fi făcute accesibile altor unităţi de program.

Structura programelor Fortran

Subprogramele modul pot fi subprograme subrutină sau funcție. Un subprogram modul poate fi apelat de orice alt subprogram modul sau de orice altă unitate de program ce are acces la modul.

Programul principal, subprogramele externe şi subprogramele modul pot conține *subprograme interne* care pot fi subrutine sau funcții. Subprogramele interne se introduc cu instrucțiunea **CONTAINS**.

Ex:

```
Program principal
...
CONTAINS
Subprogram
END
```

Forma codului sursă

Fișierul sursă în Fortran conține textul unui program ce se creează de obicei cu un editor de texte. Textul sursă este divizat în unități fizice numite *linii* (lines) sau înregistrări ale fișierului sursă și conține instrucțiuni, comentarii și linii INCLUDE.

Există două forme de scriere a codului sursă, *forma fixă* acceptată de standardele Fortran 77 și Fortran 90 și *forma liberă* acceptată de standardul Fortran 90.

În *forma fixă* înregistrările fișierului sursă au o lungime de 80 de coloane. În coloanele 1-5 se pot scrie doar etichetele instrucțiunilor urmând ca în coloanele 7-72 să fie scrise instrucțiunile.

Dacă poziția 6 conține orice caracter Fortran cu excepția caracterului blanc sau 0, atunci acea linie este o *continuare* a liniei precedente.

Forma codului sursă

De obicei, pe o linie se regăsește o singură instrucțiune însă aceasta se poate continua pe mai multe linii. Limita maximă de linii pe care se poate întinde o instrucțiune este de 20 de linii.

Coloanele 73-80 sunt ignorate de către compilator iar liniile mai scurte de 72 de caractere se completează cu blancuri.

Într-o instrucțiune blancurile nu sunt semnificative cu excepția prezenței lor într-o constantă caracter.

Exemplu:

Read*, a, b, c este echivalent cu Read*,a,b,c

Caracterul c sau * în coloana 1 indică faptul că întreaga linie este un *comentariu*. Semnul exclamării dacă nu face parte dintr-o constantă caracter, marchează începutul unui comentariu. Comentariul se termină cu sfârşitul liniei.

Forma codului sursă

Exemplu:

C Aceasta este o linie comentariu

* și aceasta este o linie comentariu

REAL alfa, beta, gama, delta, epsilon ! Începe un comentariu

! si aceasta este o linie comentariu

Comentariile sunt ignorate de către compilator. Ele se folosesc de către programator pentru a face programul mai inteligibil. De exemplu comentariile se folosesc pentru a arăta scopul fiecărei secțiuni din program.

În timpul procesării, într-un fișier sursă poate fi importat şi inclus un text sursă din alt fișier. Pentru aceasta se scrie o linie ce are sintaxa:

INCLUDE const_lit_char

Forma codului sursă

unde const_lit_char este o constantă literară caracter care reprezintă de obicei numele fișierului ce conține textul ce trebuie inclus. Linia INCLUDE se scrie în coloanele 7-72.

Exemplul:

O unitate de program ce conține o linie INCLUDE:

```
PROGRAM ANALIZA
IMPLICIT NONE
REAL A, B, C
...
INCLUDE 'ECUATIE.FOR'
...
END
```

Forma codului sursă

În *forma liberă,* instrucțiunile se scriu pe linii ce pot conține de la 0 la 132 caractere. În forma liberă dispare interdicția formei fixe de a scrie etichetele și instrucțiunile în anumite coloane.

O instrucțiune poate continua pe linia următoare dacă ultimul caracter al liniei, ce trebuie să fie continuată, este &. O instrucțiune poate continua pe cel mult 39 linii. În absența semnului de continuare &, sfârșitul liniei marchează sfârșitul instrucțiunii.

Apariția semnului exclamării în afara unei constante caracter marchează începutul unui comentariu.

Pe o linie se pot scrie două sau mai multe instrucțiuni cu condiția ca respectivele instrucțiuni să fie separate prin punct și virgulă.

Blancurile se pot folosi pentru a îmbunătăți aspectul textului.

Înregistrări formatate

În computer informațiile sunt stocate în diferite tipuri de fișiere. Fișierul este format dintr-o succesiune de înregistrări iar valorile dintr-o înregistrare se pot reprezenta în două moduri, *formatale* sau *neformatate*.

Dacă înregistrarea este scrisă cu caractere pe care le putem citi atunci se spune că înregistrarea este *formatată*.

Înregistrările neformatate constau din valori aşa cum sunt ele stocate în memoria computerului, adică în formă binară.

De exemplu, la execuția unei instrucțiuni de citire, atunci când o valoare este citită dintr-o înregistrare formatată, ea trebuie convertită de la forma externă, o înșiruire de caractere, la forma de reprezentare internă, adică cea binară.

Input-Output cu formatare condusă de listă

Instrucțiunea de citire (input) cu formatare condusă de listă are sintaxa

READ* [, var [, var]...]

unde *var* este o variabilă. Asteriscul înseamnă formatare implicită, adică este lăsat la alegerea celui ce pregătește fișierul de intrare.

Instrucțiunea de scriere (output) cu formatare condusă de listă are sintaxa

PRINT* [, ent [, ent]...]

unde *ent* este o entitate, poate fi o variabilă sau o expresie. Asteriscul în instrucțiunea PRINT înseamnă formatare implicită adică formatul datelor de ieșire este lăsat pe seama computerului.

Input-Output cu formatare condusă de listă

Separatorii de valori într-o înregistrare input sunt:

- o virgulă – este opțional precedată de blancuri unul după altul
- un slash – este opțional precedat blancuri unul după altul
- un blanc - ce se află între două caractere non blanc.

Compilarea, editarea legăturilor şi execuția

Înainte de a fi compilat, un program Fortran trebuie mai întâi convertit la o formă binară, proces ce are loc în două faze:

1) *compilarea*: compilatorul FTN90 traduce textul Fortran în cod binar care este conținut într-un fișier binar realocabil.

2) încărcarea. Folosind editorul de legături (linker) fișierul binar este încărcat în memorie împreună cu alte fișiere ce conțin cod binar relocabil sau fișiere de bibliotecă care au fost produse prin compilări anterioare.

În mod normal după compilare se vor produce:

• un fișier *binar realocabil* care are același nume cu fișierul sursă Fortran şi care are extensia "OBJ".

• un fișier *executabil* care are același nume cu fișierul sursă Fortran şi care are extensia „EXE".

Output formatat

Formatarea implicită este foarte comodă și simplă, însă uneori este rigidă. De exemplu, dacă la ieșire dorim să formăm un tabel cu date reale, nu întotdeauna reușim să aliniem datele pe coloane.

În Fortran programatorul poate introduce stilul său personal în forma outputului prin instrucțiuni de scriere ce utilizează informația dintr-o *specificație de format*.

Sintaxa unei instrucțiuni de afișare cu output formatat este:

PRINT *fmt* [, *entitate* [, *entitate*]...]

unde *fmt* este o constantă caracter a cărei valoare este specificația de format. *Specificația de format* este o *listă de descriptori de editare* separați prin virgule și închiși în paranteze.

Output formatat

Pentru a putea formata output-ul se poate utiliza următoarea instrucțiune:

PRINT "(A, I3, 2X, A, F6.2, 2X, A, E15.6)", ' n =', n, 'b = ', b, 'c = ', c

Dacă n = 100, b = 12.3 și c = 15.10^8 ea va produce outputul

n =100 b = -12.3 c = 0.150000E10

Exemplul arată 5 dintre cei mai des folosiți descriptori: A pentru date alfanumerice, I pentru date întregi, F şi E pentru date reale, X pentru spațiere.

Output formatat

Descriptorul A rezervă spațiu pentru outputul datelor caracter. Lungimea reală a constantei caracter de tipărit sau lungimea declarată a variabilei determină câte coloane sunt folosite pentru ea la ieșire. *Forma generală* a descriptorului A este

A [w]

unde w este lungimea câmpului măsurată în caractere.

Dacă w > len (numărul de caractere ce vor fi printate) atunci vor fi introduse blancuri pentru a compensa diferența (aliniate la dreapta).

Dacă w < len atunci vor fi afișate un număr de caractere, de la stânga la dreapta, ce va fi egal cu w.

Dacă w este omis se folosește lungimea existentă a valorii la ieșire.

Output formatat

Exemple

PRINT *, "Simple text" ⇒ Simple text

PRINT "(A)", "Simple text" ⇒ imple text

CHARACTER(LEN=5) :: FMT = "(A15)"

PRINT FMT, "Simple text" ⇒ Simple text

FMT = "(A10)"

PRINT FMT, "Simple text" ⇒ imple tex

CHARACTER(LEN=20) :: VAR = "Simple text"

PRINT '(A)', VAR ⇒ imple text + 9 spații

PRINT '(A)', VAR(8:11) ⇒ text

Output formatat

Descriptorul I se folosește la formatarea întregilor. Forma generală a acestui descriptor este

Iw

unde w este lungimea câmpului. Trebuiesc incluse toate cifrele inclusiv semnul, dacă este negativ.

ex. I4 -123

Print “(I4)”, 12345 - ****

Output formatat

*Descriptorul F*6.2 înseamnă că un total de 6 coloane sunt rezervate pentru a tipări valoarea variabilei reale b în forma poziţională, valoarea fiind rotunjită la două zecimale. Punctul zecimal ocupă o coloană, iar dacă semnul este negativ, semnul mai ocupă o coloană, prin urmare, în formatul F6.2, numărul cel mai mare tipăribil este 999.99, iar -99.99, cel mai mic.

Forma generală a acestui descriptor este

Fw.d

unde w este lungimea în caractere a câmpului, iar d este numărul de zecimale care se rotunjesc la tipărire.

Output formatat

Descriptorul E15.6 înseamnă că un total de 15 coloane sunt rezervate pentru a tipări valoarea variabilei reale c. Forma generală a acestui descriptor este

Ew.d

unde w este lungimea câmpului dată în coloane și d este numărul de zecimale.

Descriptorul 2X este un editor de editare a poziției ce lasă două spații. Forma generală este

nX

unde n înseamnă număr de caractere blanc.

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 9. Controlul execuției

- ***Blocuri de instrucțiuni executabile***
- ***Construcția IF***
- ***Instrucțiunile IF logic şi GO TO***
- ***Construcția DO***
- ***Bucle DO simple***
- ***Bucle DO WHILE***
- ***Instrucțiunile GO TO, CONTINUE şi STOP.***

Blocuri de instrucțiuni executabile

O construcție de control constă din unul sau mai multe blocuri de instrucțiuni şi construcții Fortran precum şi o logică a controlului ce cuprinde implicit sau explicit aceste blocuri.

La execuție, în funcție de o condiție de control, se selectează un anumit bloc de instrucțiuni şi construcții.

Un *bloc* este o secvență de zero sau mai multe instrucțiuni de acțiune şi construcții de control. Altfel spus, un bloc este o secvență de construcții executabile. Expresiile din logica controlului execuției determină dacă blocul va fi sau nu va fi executat. De exemplu, la începutul blocului poate exista o instrucțiune de salt ce face ca celelalte instrucțiuni din bloc să nu fie executate; şi în acest caz se consideră că a avut loc o execuție completă a blocului.

Blocuri de instrucțiuni executabile

De obicei construcția de control conține o instrucțiune inițială înaintea unui bloc şi o instrucțiune finală după un bloc. Există construcții ce au mai mult decât un bloc. Construcția include şi condiții ce determină dacă blocul se execută sau nu. Unele din construcții conțin instrucțiuni adiționale ce determină care bloc anume este ales pentru executie.

Pentru blocuri şi controlul blocurilor există următoarele *reguli şi restricții*:

- Când se execută un bloc execuția începe cu prima instrucțiune sau prima construcție a blocului. Instrucțiunile blocului se execută în ordine una după alta atâta timp cât în bloc nu există o construcție de control sau o instrucțiune de control care să modifice ordinea secvențială.

Blocuri de instrucțiuni executabile

- Orice bloc, conceput ca o singură entitate, trebuie să fie complet conținut, înglobat, în interiorul unei construcții.
- În interiorul unui bloc este permis un salt sau o construcție ce trimite controlul la o instrucțiune sau la o construcție din același bloc.
- Ieșirea din bloc se poate face oriunde din interiorul blocului.
- Este interzis saltul din afara blocului la o instrucțiune sau o construcție conținută într-un bloc.
- În interiorul unui bloc sunt permise apelurile unor proceduri.

Construcția IF

O construcție IF selectează pentru execuție cel mult un bloc de instrucțiuni şi construcții. Sintaxa generală a instrucțiunii IF este:

[*nume:*] IF *(expr_1)* THEN
bloc
[ELSE IF *(expr_2)* THEN [*nume*]
bloc] ...
[ELSE [*nume*]
bloc]
END IF [*nume*]

Aici *expr_1, expr_2* sunt expresii logice scalare, *bloc* este un bloc de instrucțiuni şi construcții iar *nume* este nume de construcție IF. Construcția IF conține în mod obligatoriu o instrucțiune IF THEN şi o instrucțiune END IF.

Construcția IF

Opțional ea poate conține instrucțiuni ELSE IF sau o instrucțiune ELSE.

Opțional instrucțiunea IF THEN se poate identifica, sau cum se mai spune, *se poate numi,* cu *nume,* un *nume de* construcţie IF. În acest caz instrucţiunea END IF trebuie să fie şi ea numită să specifice același nume. Dacă instrucțiunea IF THEN nu este numită, nu este numită nici instrucțiunea END IF. Dacă instrucțiunile IF THEN şi END IF sunt numite, este posibil să numim cu același nume instrucțiunea ELSE sau instrucțiunile ELSE IF.

Trebuie luate în considerație următoarele *reguli* şi *restricții:*

- Se execută cel mult unul din blocurile construcției; este posibil ca niciun bloc să nu se execute.

Construcția IF

- După instrucțiunea ELSE nu sunt permise instrucțiuni ELSE IF.
- Sunt interzise salturile la o instrucțiune ELSE IF sau la o instrucțiune ELSE.
- În interiorul unei construcții IF este permis saltul la END IF din oricare din blocurile construcției IF. Saltul din afară la END IF este permis, dar este considerat o caracteristică depășită pe care nu o recomandăm.

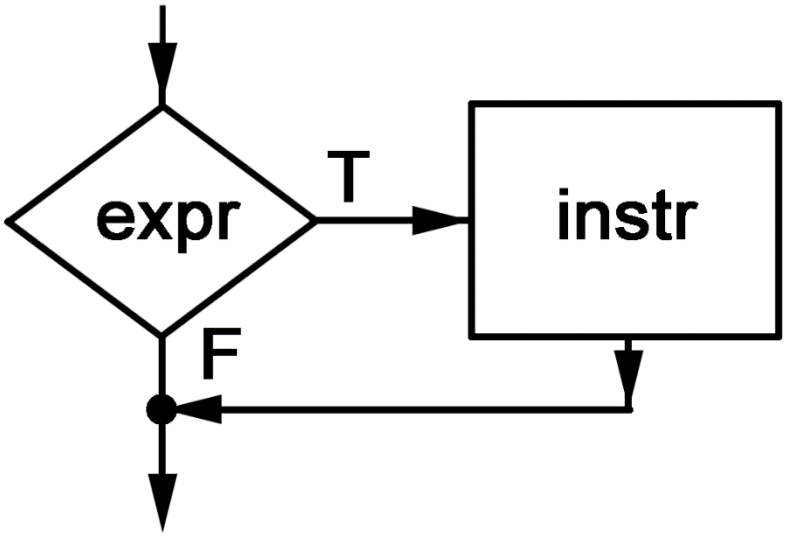
Execuția construcției IF se face astfel: expresiile logice sunt evaluate în ordine până ce găsește una adevărată; atunci se execută primul bloc ce urmează după prima expresie adevărată şi execuția construcției IF se termină. Expresiile logice adevărate care urmează nu mai au niciun efect.

Construcția IF

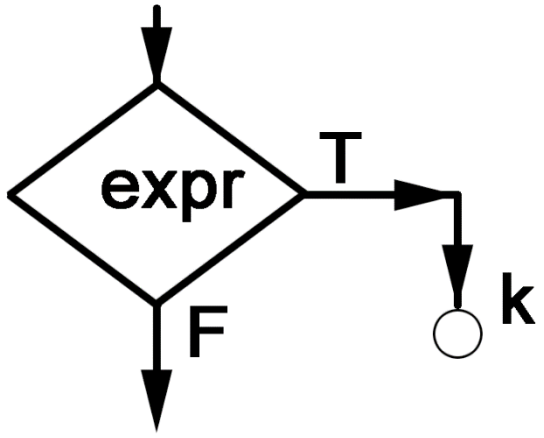
Se poate întâmpla ca niciuna din expresiile logice ale construcției să nu fie .adevărată. în acest caz se execută blocul ce urmează după instrucțiunea ELSE dacă există unul; în caz contrar nu se execută niciunul din blocurile construcției.

Instrucțiunile IF logic și GO TO

Cele mai simple selecții sunt arătate în schemele logice de mai jos. În acest caz nu mai există condițiile ELSE IF sau ELSE şi acțiunea ce trebuie luată dacă expresia logică *expr* este adevărată constă în execuția unei singure instrucțiuni de acțiune *instr* .



a)



b)

Instrucțiunile IF logic și GO TO

În limbajul schemelor logice cerculeţul din figura b este un simbol conector. Simbolul conector se foloseşte atunci când folosirea liniei continue este limitată de dimensiunea hârtiei sau din considerente de estetică a schemei logice. Pentru a identifica conectorii de intrare şi de ieşire se folosesc simboluri de identificare, litere sau cifre. Astfel, conectorii reprezintă un mijloc convenabil pentru a lega două puncte ale schemei logice fără a folosi liniile de flux. O altă funcţie a conectori lor este de a permite reprezentarea pe mai multe pagini a unei scheme logice mai complicate.

Pentru codificarea selecţiei în Fortran există instrucţiunea IF logic. Sintaxa instrucţiunii IF logic este:

IF (expr) *instr*

Instrucțiunile IF logic și GO TO

Aici expresie este o expresie logică scalară iar *instr* este o instrucțiune de acțiune. Instrucțiunea de acțiune nu poate fi o instrucțiune IF sau o instrucțiune END a unui program subrutină sau funcție.

Modul de execuție este următorul: se evaluează expresia logică scalară expr. Dacă valoarea expresiei este .TRUE. se execută instrucțiunea *instr*, după care, controlul trece la instrucțiunea următoare din program. Dacă valoarea expresiei este .FALSE., controlul trece la instrucțiunea ce urmează în program.

Exemplu

instrucțiunea IF logic

```
IF ( val .GT. 0. ) val = 0.0
```

este echivalentă cu următoarea construcție IF:

```
IF (val .GT. 0.) THEN
val = 0.0
ENDIF
```

Instrucțiunile IF logic și GO TO

Cazul din figura b corespunde situației în care instrucțiunea ce urmează a fi executată este instrucțiunea GO TO. Instrucțiunea GO TO este o instrucțiune de salt necondiționat ce afectează ordinea de execuție. Sintaxa instrucțiunii GO TO este următoarea:

GO TO *et*

unde *et* este o etichetă. Eticheta este un șir de 1 până la 5 cifre; zerourile din față se ignoră; aceasta înseamnă că 010, 0010 şi 10 reprezintă aceeași etichetă.

Execuția instrucțiunii GO TO se face astfel: când se execută instrucțiunea GO TO, următoarea instrucțiune ce se va executa este instrucțiunea țintă din aceeași unitate de program instrucțiune ce are eticheta *et*.

Construcția DO

Iterațiile, sau cum se mai numesc, *buclele,* se implementează prin construcții DO. În Fortran există trei forme diferite ale construcțiilor DO:

- construcție DO simplă
- construcție DO WHILE
- construcție DO cu control al iterației.

Există două forme de bază ale construcției DO, DO *cu bloc* și DO *fără bloc* însă practica programării moderne favorizează construcția DO *cu bloc*.

Sintaxa construcției DO este:

[*nume:*] DO [control_buclă]
bloc
END DO [*nume*]

Construcția DO

bloc conține zero sau mai multe instrucțiuni şi construcții a căror execuție repetată este comandată de controlul buclei.

Blocul de instrucțiuni *bloc* se mai numește *domeniul buclei* DO (range). Domeniul buclei DO poate conține şi construcții IF, construcții CASE sau chiar alte construcții DO; este necesar ca orice construcție interioară să fie înglobată complet în construcția exterioară. Dacă domeniul buclei DO conține o altă construcție DO se spune că buclele DO sunt *imbricate* (nested).

Există două instrucțiuni speciale ce pot să apară în domeniul unei construcții DO şi care pot să altereze într-un anumit mod special execuția secvențială a instrucțiunilor şi construcțiilor blocului. Una este instrucțiunea EXIT, cealaltă instrucțiunea CYCLE.

Construcția DO

Instrucțiunea EXIT are sintaxa:

EXIT [*nume*]

Daca în instrucțiunea EXIT este referit un *nume* (un nume de construcție DO), instrucțiunea EXIT trebuie să fie conținută în domeniul acelei bucle DO, altfel instrucțiunea EXIT trebuie să se găsească în domeniul cel puțin a unei bucle DO.

Execuția instrucțiunii EXIT înseamnă terminarea execuției domeniului buclei DO: instrucțiunile de acțiune ce urmează lui EXIT în domeniul buclei DO nu se mai execută şi controlul trece la prima instrucțiune executabilă ce urmează instrucțiunii END DO cu același nume. Dacă EXIT nu este numită, atunci instrucțiunea EXIT termină execuţia buclei DO celei mai interioare.

Construcția DO

Instrucțiunea CYCLE are sintaxa:

CYCLE [*name*]

Dacă în instrucțiunea CYCLE este referit un *nume* (un nume de construcție DO), instrucțiunea CYCLE trebuie să fie conținută în domeniul acelei bucle DO, altfel instrucțiunea CYCLE trebuie să se găsească în domeniul cel puțin a unei bucle DO.

Instrucțiunea CYCLE, spre deosebire de instrucțiunea EXIT, ce termină complet execuția unei construcții DO, întrerupe execuția domeniului şi cauzează începerea unui nou ciclu de execuţie al construcției DO. Cu alte cuvinte, execuţia se întrerupe şi se începe o nouă execuție a domeniului început cu prima instrucțiune a sa. În cazul unei construcții DO cu contor al iterației, după întrerupere se fac şi reajustările necesare la contorul iteraţiei şi la variabila DO.

Construcția DO simplă

În cazul construcției DO simple lipsește controlul buclei. De aceea, buclele DO simple se mai numesc “bucle fără sfârșit„. *Sintaxa* construcţiei DO *simple* este:

[*nume:*] DO
bloc
END DO [*nume*]

Construcția DO simplă, fară controlul buclei, permite să se repete execuția blocului (domeniul buclei) până ce construcția DO se termină în mod explicit la o instrucțiune din interiorul domeniului.

Există două instrucțiuni speciale ce pot să apară în domeniul unei construcții DO şi care pot să altereze execuția secvențială a instrucțiunilor şi construcțiilor blocului. Una este instrucțiunea EXIT iar cealaltă instrucțiunea CYCLE.

Construcția DO simplă

Instrucţiunea EXIT cauzează terminarea imediată a execuţiei construcţiei DO şi transmite controlul la prima instrucţiune executabilă dupa END DO. Sintaxa acestei instrucțiuni este:

EXIT

Instrucțiunea CYCLE, spre deosebire de instrucțiunea EXIT, ce termină complet execuția unei construcții DO, întrerupe execuția domeniului şi cauzează începerea unui nou ciclu de execuție al construcției DO. Sintaxa instrucțiunii CYCLE este:

CYCLE

La întreruperea execuției unei construcții DO de către o instrucțiune CYCLE, variabila de control a iterației este actualizată şi începe procesul de execuție al iterației următoare.

Bucle DO WHILE

Forma *DO WHILE* a construcţiei DO permite execuţia repetată a unui bloc atât timp cât valoarea unei expresii logice rămâne adevărată. Sintaxa instrucţiunii DO WHILE este:

do while (expr)
bloc
end do

Aici *expr* este o expresie logică scalară. Execuţia unei construcţii DO WHILE se face astfel: se evaluează expresia logică *expr*; dacă valoarea expresiei este .TRUE. se execută blocul.

După execuţia ultimei instrucţiuni a blocului, execuţia revine la instrucţiunea DO WHILE, expresia logică *expr* este din nou evaluată şi ciclul se repetă. Dacă valoarea expresiei logice *expr* este .FALSE. blocul nu se mai execută şi controlul trece la prima instrucţiune executabilă după END DO.

Construcția DO cu contor al iterației

În acest caz o variabilă întreagă ic, numită *contorul iterației*, controlează de câte ori se execută blocul buclei.

Sintaxa construcţiei DO cu contor al iterației este următoarea:

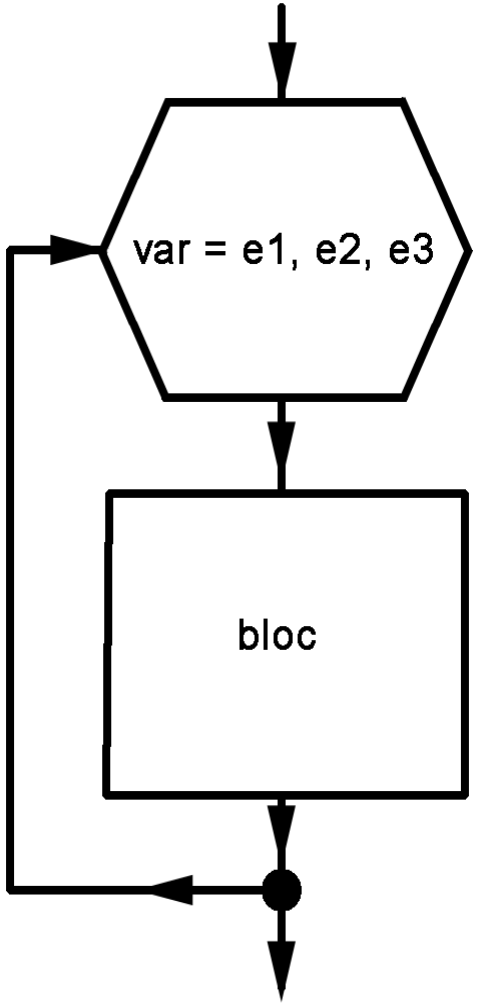
[*nume:*] DO i = e1, e2 [, e3]
bloc
END DO [*nume*]

Aici i = e1, e2 [,e3] se numeşte *controlul iteraţiei*. Mărimea i, numită *variabilă* DO, sau *variabila buclei*, este o variabilă întreagă sau reală, iar e1, e2 şi e3 sunt expresii numerice întregi sau reale. Dacă e3 este 1 se poate omite scrierea lui.

Construcția DO cu contor al iterației

Acest tip de construcție este des întâlnit în programare, motiv pentru care a fost introdus un simbol special, simbolul preparare, a cărui schemă este prezentată în dreapta.

După terminarea execuției construcției DO, variabila DO își păstrează ultima valoare, cea pe care a avut-o atunci când a fost testat contorul iterației ic. În decursul execuției domeniului buclei, variabila DO nu trebuie redefinită, nici nu trebuie să rămână nedefinită. Observăm că schimbarea în decursul execuției blocului a variabilelor folosite în expresiile e1, e2 sau e3 nu modifică contorul iteraţiei; acesta este fixat la intrarea în iteraţie.



Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 10. Tablouri

- *Tablouri şi elemente de tablou*
- *Declararea dimensiunilor şi a tipului*

Tablouri şi elemente de tablou

Adesea, în practica programării, întâlnim problema prelucrării unei mulţimi ordonate de date care se întind pe mal multe dimensiuni pentru a reprezenta coloane, linii, plane, etc.

În limbajul Fortran o mulțime ordonată de date scalare cu 1-7 dimensiuni, toate de același tip și cu aceeași parametri de tip (kind şi lungime) se poate organiza ca o entitate numită *tablou* (array). Tabloul se identifică printr-un *nume* iar oricare din elementele scalare ale tabloului se numește *element de tablou*.

Definiție

element de tablou este *nume* (*listă_de_indici*)

listă_de_indici este *indice* [, *indice*]

Tablouri şi elemente de tablou

Aici *nume* este numele tabloului iar *indice* este o expresie scalară întreagă. Prin urmare, pentru a referi în program un element de tablou, se scrie numele tabloului urmat de o listă *de indici* (subscript list) închisă între paranteze. Pentru fiecare dimensiune există un indice a cărui valoare indică poziția în acea dimensiune. Numărul de elemente într-o dimensiune se numește *întinderea* (extent) în acea dimensiune. *Rangul tabloului* (array rank) este numărul de dimensiuni ale tabloului. *Forma unui tablou* (array shape) este determinată de rangul său şi de întinderile sale. Pentru a specifica forma unui tablou de rang *n* folosim notația *(d1, d2,...,dn),* unde *di* este întinderea în dimensiunea *i*.

Numărul total de elemente ale tabloului, egal cu produsul întinderilor, se numește *mărimea tabloului (array* size).

Tablouri şi elemente de tablou

Exemplul 1. Vectorul viteză $\vec{v}$ ce are componentele $\vec{v}$ = (2,-3,5) se poate organiza ca un tablou real unidimensional cu forma (3), tablou ce are numele viteza. Cele 3 elemente de tablou se notează cu viteza(1), viteza(2) şi viteza(3) şi au valorile 2, -3 şi 5.

Exemplul 2. Matricea dreptunghiulară a cu 3 linii şi 4 coloane

$$a = \begin{pmatrix} 3 & 2 & 5 & -4 \\ -1 & 7 & 25 & 8 \\ 4 & 9 & -6 & 1 \end{pmatrix}$$

se poate organiza ca un tablou real bidimensional, cu forma (4,3) şi mărimea 4 x 3 = 12. Elementele tabloului sunt a(1, 1), a(2,1), a(3,1), a(4,1), a(1,2), a(2,2), a(3,2), a(4,2), a(1,3), a(2.3), a(3.3), a(4,3), a(1 ,4), a(2,4), a(3,4), a(4,4). Spre deosebire de notaţia matricială, aici primul indice numără coloanele iar al doilea numără liniile.

Tablouri şi elemente de tablou

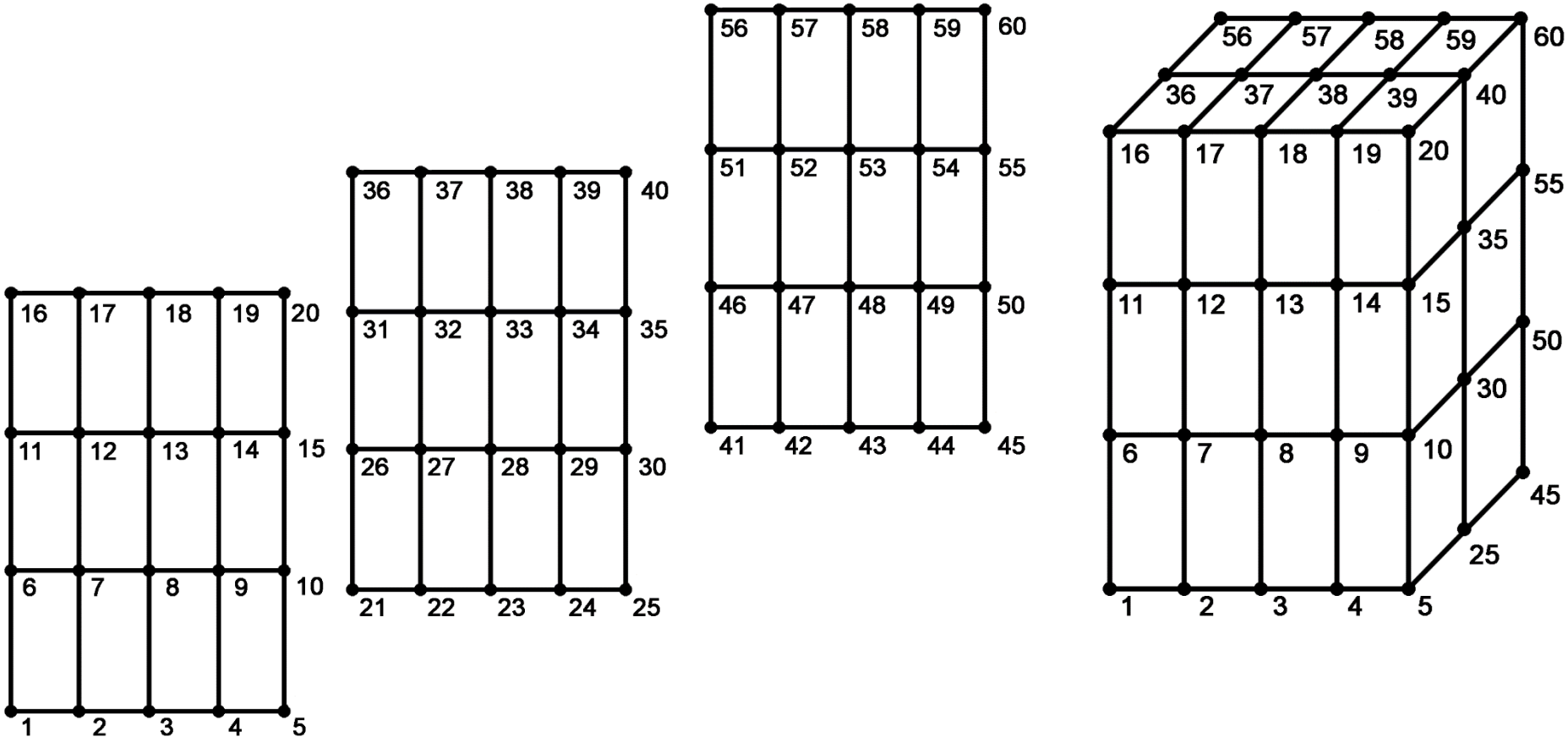
Exemplul 3. Lista cu numele studenţilor dintr-o grupă:

Adam
Barbu
Constantin
Diaconu

se poate organiza ca un tablou unidimensional, de tip caracter, de mărime 4 şi formă (4), tablou numit grupă. Elementele tabloului sunt grupa(1), grupa(2), grupa(3), grupa(4) toate având lungimea egală cu 10.

Tablouri şi elemente de tablou

Exemplul 4. Un corp omogen de forma unui paralelipiped dreptunghic este discretizat în elemente finite cu ajutorul unei grile 3D, prezentate mai jos.



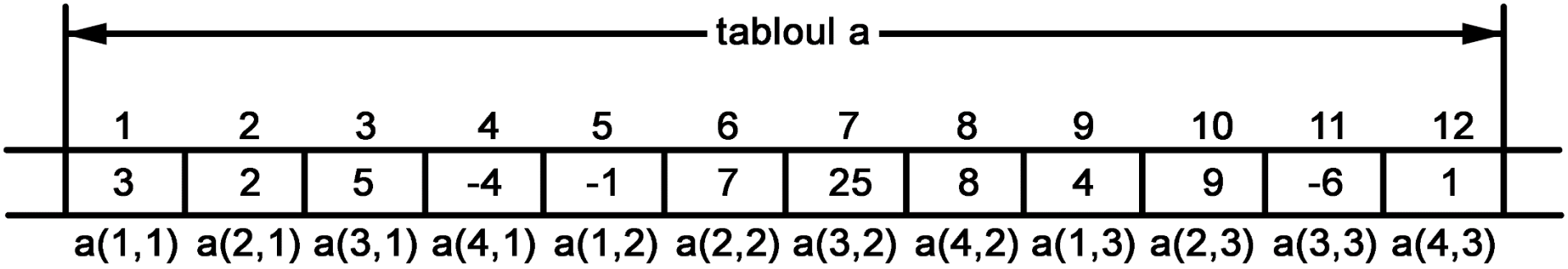
Tablouri şi elemente de tablou

Mulţimea nodurilor grilei, numerotate de la 1 la 60, mulțime ce alcătuieşte un patern ordonat cu linii, coloane şi planuri se poate organiza ca un tablou tridimensional numit nod. Elementele tabloului se notează nod(1,1,1), nod(2,1,1), nod(3,1,1), ... , nod(5,4,3). Aici primul indice numără coloanele, al doilea numără liniile şi al treilea numără planele. De exemplu, nod(2,3,1)=12, nod(4,3,2)=34. Tabloul nod are forma (5,4,3).

În memorie, elementele unui tablou unidimensional, se stochează unul după altul iar elementele tablourilor multidimensionale sunt stocate tot ca un șir liniar. Un tablou se stochează în memorie astfel ca cel mai din stânga indice să varieze cel mai rapid, iar cel mai din dreapta să varieze cel mai lent.

Tablouri şi elemente de tablou

Exemplul În figura de mai jos arătăm modul de stocare în memorie a tabloului a, care este un tablou 2D cu 12 elemente.



Declararea tipului

Într-un program Fortran entitățile folosite (variabile, tablouri, proceduri) posedă diferite *atribute*. Atributele determină modul în care entitatea va fi folosită în program.

Atributele pot fi specificate, fie cu o instrucțiune de declarare a tipului, fie cu instrucțiuni de declarare a unui anumit atribut (instrucțiunile DIMENSION, ALLOCATABLE, TARGET, DATA, PARAMETER, PUBLIC, PRIVATE, INTENT, OPTIONAL, SAVE, EXTERNAL, INTRINSIC). Asta înseamnă că în limbajul Fortran, specificațiile atributelor se pot face în două moduri: i) *orientate către entitate* și *orientate către atribut*.

Exemplu. În instrucţiunea de declarare a tipului

REAL, PARAMETER :: A = 2.

am colectat într-o singură instrucţiune toate atributele entității A. În acest exemplu avem de-a face cu o specificație orientată către entitate.

Declararea tipului

Putem face specificații orientate către atribut dacă în locul instrucțiunii anterioare scriem declarațiile:

```
REAL A
PARAMETER ( A = 2. )
```

Declararea dimensiunilor

În limbajul Fortran orice tablou posedă *atributul dimensiune*. Declararea dimensiunilor unui tablou este obligatorie şi se face cu ajutorul unor instrucţiuni ce conţine o specificaţie *de tablou*.

Există mai multe metode pentru a declara un tablou însă vom utiliza doar *declararea tablourilor cu dimensiuni fixe*.

Şabloanele recomandate sunt:

type, DIMENSION *(spec_de_tablou)* :: *nume_de_tablou*

type *nume_de_tablou (spec_de_tablou)*

Primul şablon îl folosim atunci când dorim să declarăm că mai multe tablouri au acelaşi tip şi aceleaşi dimensiuni. Al doilea şablon îl folosim atunci când dorim să declarăm că tablouri diferite, de dimensiuni diferite, au toate acelaşi tip.

Declararea dimensiunilor

Instrucțiunea DIMENSION are forma:

DIMENSION [::] *nume_de_tablou (spec_de_tablou)*

Exemplu. Tabloul întreg 1D numit indice are 25 elemente. Pentru a-i declara dimensiunile putem folosi o declarare orientată către entitate:

INTEGER, DIMENSION (25) :: INDICE

sau putem folosi declararea orientată către atribut:

INTEGER INDICE

DIMENSION INDICE (10)

Declararea dimensiunilor

Exemplu. Tablourile 2D, cu forma (3,4), **alfa**, **beta**, **gama**, **delta**, **epsilon** au fiecare 12 elemente. Declararea dimensiunilor o putem face cu instrucţiunea:

REAL, DIMENSION (3,4) :: alfa, beta, gama, delta, epsilon

sau cu instrucţiunea:

REAL alfa(3,4), beta(3,4), gama(3,4), delta(3,4), epsilon(3,4)

sau cu grupul de instrucţiuni:

REAL alfa, beta, gama, delta, epsilon

DIMENSION alfa(3,4), beta(3,4), gama(3,4), delta(3,4), epsilon(3,4)

Declararea dimensiunilor

Exemplu. Instrucţiunile

REAL a, b, c, d

DIMENSION a(3), b(-1 : 4), c(2,4), d(2,3,2)

declară că tabloul real cu numele a este unidimensional şi are trei elemente ce se stochează în memorie în ordinea a(1), a(2), a(3); tabloul real numit b este unidimensional şi are şase elemente ce se stochează în ordinea b(-1), b(0), b(1), b(2), b(3), b(4); tabloul real numit c este bidimensional şi că are 2 x 4 = 8 elemente stocate în ordinea c(1,1), c(2,1), c(1,2), c(2,2), c(1,3), c(2,3), c(1,4), c(2,4) şi tabloul real numit d are 2 x 3 x 2 = 12 elemente stocate în ordinea d(1,1,1), d(2,1,1), d(1 ,2,1), d(2,2,1), d(1,3,1), d(2,3,1), d(1 ,1,2), d(2,1 ,2), d(1,2,2), d(2,2,2), d(1,3,2), d(2,3,2).

Declararea dimensiunilor

Exemplu. Instrucțiunile

```
INTEGER, DIMENSION(4):: int
REAL alfa (2,3), b(0:5)
LOGICAL, DIMENSION(3) :: gama
CHARACTER*20 nume (10)
```

declară că tabloul int de tip INTEGER are patru elemente; tabloul alfa de tip REAL este bidimensional şi are 2 x 3 = 6 elemente iar b, tablou bidimensional, este de tip REAL şi are 6 elemente; tabloul unidimensional gama are 3 elemente şi este de tip LOGICAL, iar tabloul nume cu 10 elemente este de tip CHARACTER*20.

Utilizarea tablourilor

Exemplu. Pentru a citi element cu element tabloul real a cu 4 elemente putem folosi instrucțiunile:

```
READ*, a(1)
READ*, a(2)
READ*, a(3)
READ*, a(4)
```

Dacă tabloul a ar avea mai multe elemente, de exemplu 100, acest procedeu de citire este ineficient. Pentru citirea element cu element a tabloului putem folosi o secvență ce conține o iterație:

```
READ*,n
DO i=1,n
        READ*, a(i)
END DO
```

Utilizarea tablourilor

Aici n este numărul de elemente ale tabloului. Observăm că în corpul buclei DO figurează a(i) unde indicele i este o variabilă întreagă.

Pentru scrierea element cu element a tabloului a putem folosi secvența:

```
DO i=1,n
        PRINT*, a(i)
END DO
```

Utilizarea tablourilor

Exemplu. Citirea unei matrici pe linii, element cu element

```
INTEGER i, j, n
PARAMETER (n = 5)
REAL a(n, n)
DO i=1,n
        DO j=1,n
                READ*, a(i, j)
        END DO
END DO
```

imbricate).

Exemplu. Citirea unei matrici pe coloane, element cu element

```
INTEGER i, j, n
PARAMETER (n= 5)
REAL a(n, n)
DO j=1,n
        DO i=1,n
                READ*, a( j, i)
        END DO
END DO
```

în aceste exemplele avem de-a face cu bucle incluse unele în altele (bucle imbricate – suprapuse ca țiglele).

Utilizarea tablourilor

Exemplu. Programul următor citeşte două matrici, află produsul lor şi scrie rezultatul.

```
PROGRAM produs_matrici
IMPLICIT NONE
INTEGER i, j, k, n
PARAMETER (n = 3)
REAL a(n,n), b(n,n), c(n,n)
! Citirea pe linii a matricii a (element cu element)
DO i=1,n
        DO j=1,n
                READ*, a(i,j)
        END DO
END DO
! Citirea pe linii a matricii b (element cu element)
DO i=1,n
        DO j=1,n
                READ*, b(i,j)
        END DO
END DO
! Calculul matricii produs c = a * b
DO i=1,n
        DO j=1,n
            c(i,j) = 0.
              DO k=1,n
                 c(i,j) = c(i,j) + a(i,k) * b(k,j)
              END DO
        END DO
END DO
! Tiprirea matricii c
J= 1
DO i=1,n
        PRINT*, c(i,j), c(i,j+1), c(i,j+2)
END DO
END PROGRAM produs_matrici
```

Bibliografie

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org

Bogdan PRICOP
Gheorghe BĂDĂRĂU

Cap 11. Proceduri

- ***Proceduri Fortran***
- ***Funcții***
- ***Subrutine***
- ***Tehnici de testare a programelor***

Funcții și subrutine

- Limbajul Fortran are două tipuri de subprograme, funcția și subrutina.
- Funcția returnează un rezultat calculat prin numele acesteia.
- Dacă funcția nu trebuie să returneze un rezultat prin numele acesteia atunci se folosește subrutina.

Sintaxa funcției: 1/3

- O funcție în Fortran are următoarea sintaxă:

prefix FUNCTION numele_funcție (arg1, arg2, …, argn)
IMPLICIT NONE
[partea de specificație]
[partea de execuție]
[partea de subprograme]
END FUNCTION numele_funcție

- **Prefix** poate fi **INTEGER, REAL, LOGICAL**, etc. cu sau fără parametrul **KIND**.
- **numele_funcție** reprezintă un identificator Fortran.
- **arg1, arg2, …, argn** sunt argumente formale.

Sintaxa funcției: 2/3

- O funcție este o unitate de sine stătătoare care primește date de intrare din exterior, prin **argumentele sale formale**, face niște calcule și returnează rezultatul cu numele funcției.
- Undeva într-o funcție trebuie să existe unul sau mai multe declarații de atribuire ca aceasta:

  numele_funcției = **expresie**

  unde rezultatul **expresiei** este atribuit numelui funcției
- De reținut este că **numele_funcției** nu poate apărea în partea dreaptă a oricărei expresii.

Sintaxa funcției: 3/3

- Într-o specificație a tipului, argumente formale ar trebui să aibă o nou atribut **INTENT (IN)**.
- Semnificația atributului **INTENT (IN)** este că funcția ia numai valoarea de la un argument formal și nu schimbă conținutul său.
- Orice declarație care pot fi utilizată în **PROGRAM** poate fi, de asemenea, utilizată și într-o **FUNCȚIE**.

Exemple de Funcții

- Rețineți că funcțiile pot să nu aibă argumente formale.
- Însă, **( )** este în continuare necesară.

Calcul factorial

```
INTEGER FUNCTION Factorial (n)
    IMPLICIT NONE
    INTEGER, INTENT (IN) n
    INTEGER  i, Ans

    Ans = 1
    DO i = 1, n
        Ans = Ans * i
    END DO
    Factorial = Ans
END FUNCTION Factorial
```

Citeşte şi returnează un număr real pozitiv

```
REAL FUNCTION GetNumber ()
    IMPLICIT NONE
    REAL  Input_Value
    DO
        PRINT *,  “A positive number:”
        READ *,  Input_Value
        IF  (Input_Value .GT. 0.0) EXIT
        PRINT *, “ERROR. try again.”
    END DO
    GetNumber = Input_Value
END FUNCTION GetNumber
```

Probleme des întâlnite: 1/2

s-a uitat tipul funcției

```
FUNCTION DoSomething (a, b)
    IMPLICIT NONE
    INTEGER, INTENT (IN)  a, b
    DoSomthing = SQRT(a*a + b*b)
END FUNCTION DoSomething
```

s-a uitat INTENT(IN) – nu este greșit

```
REAL FUNCTION DoSomething (a, b)
    IMPLICIT NONE
    INTEGER a, b
    DoSomthing = SQRT(a*a + b*b)
END FUNCTION DoSomething
```

s-a schimbat argumentul INTENT(IN)

```
REAL FUNCTION DoSomething (a, b)
    IMPLICIT NONE
    INTEGER, INTENT(IN) a, b
      IF (a .GT. b) THEN
        a = a – b
      ELSE
        a = a + b
    END IF
    DoSomthing = SQRT(a*a+b*b)
END FUNCTION DoSomething
```

s-a uitat să se returneze rezultatul

```
REAL FUNCTION DoSomething (a, b)
  IMPLICIT NONE
  INTEGER, INTENT(IN) a, b
  INTEGER c
  c = SQRT(a*a + b*b)
END FUNCTION DoSomething
```

Probleme des întâlnite: 2/2

utilizare incorectă a numelui funcţiei

```
REAL FUNCTION DoSomething (a, b)
  IMPLICIT NONE
  INTEGER, INTENT(IN) a, b
  DoSomething = a*a + b*b
  DoSomething = SQRT(DoSomething)
END FUNCTION DoSomething
```

doar cea mai recentă valoare este returnată

```
REAL FUNCTION DoSomething (a, b)
   IMPLICIT NONE
   INTEGER, INTENT(IN) a, b
   DoSomething = a*a + b*b
   DoSomething = SQRT(a*a - b*b)
END FUNCTION DoSomething
```

Asocierea argumentelor: 1/5

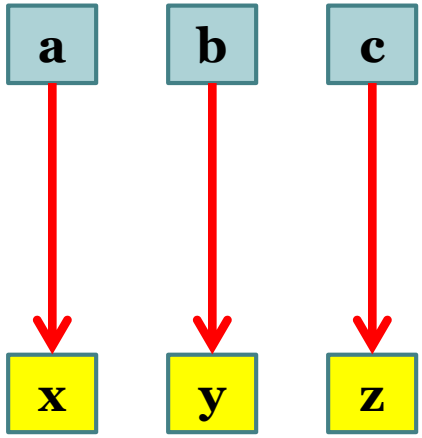
- **Asocierea argumentelor** este o modalitate de a transfera conținutul de la argumentele reale la argumentele formale.
- În cazul în care un argument real este o **expresie**, acesta se evaluează și se stochează într-o **locație temporară** după care valoarea este transferată la argumentul formal corespunzător.
- Dacă un argument real este o **variabilă**, valoarea acestuia este transferată argumentului formal corespunzător.

Asocierea argumentelor: 2/5

- Argumentele reale sunt variabile:

```
PRINT*,  Sum (a,b,c)

INTEGER FUNCTION Sum (x,y,z)
      IMPLICIT NONE
      INTEGER, INTENT (IN)  x,y,z
      ........
END FUNCTION Sum
```

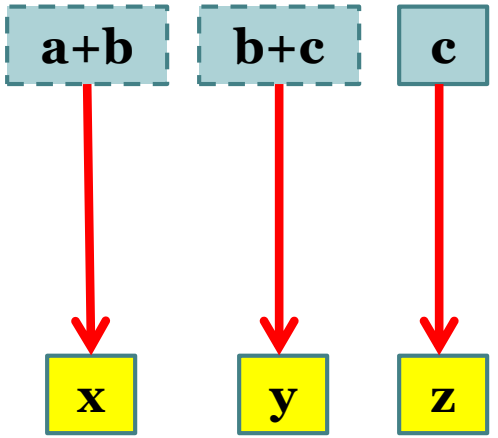


Asocierea argumentelor: 3/5

- Expresii pe post de argumente reale. Dreptunghiurile cu linie întreruptă sunt locații temporare.

```
PRINT*, Sum(a+b,b+c,c)

INTEGER FUNCTION Sum (x,y,z)
    IMPLICIT NONE
    INTEGER, INTENT (IN)  x,y,z
    ……..
END FUNCTION Sum
```

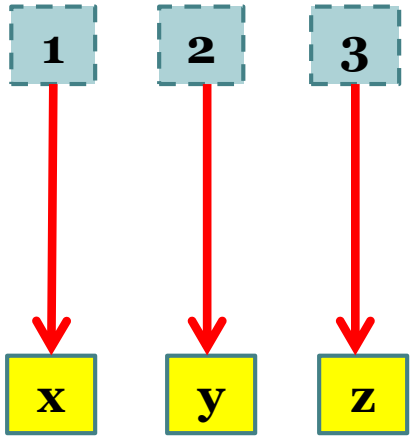


Asocierea argumentelor: 4/5

- Constante pe post de argumente reale. Dreptunghiurile cu linie întreruptă sunt locații temporare.

```
PRINT*, Sum (1, 2, 3)

INTEGER FUNCTION Sum (x,y,z)
    IMPLICIT NONE
    INTEGER, INTENT (IN)  x,y,z
    ……..
END FUNCTION Sum
```

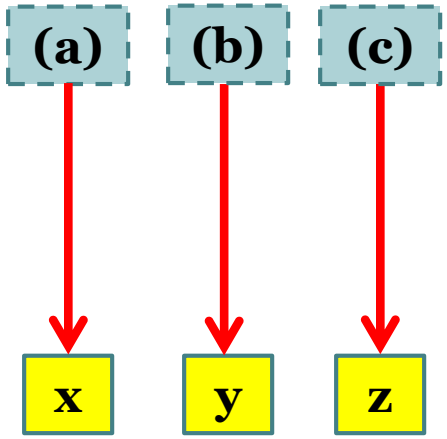


Asocierea argumentelor: 5/5

- Variabilele scrise între paranteze rotunde sunt considerate expresii. Dreptunghiurile cu linie întreruptă sunt locații temporare.

```
PRINT*, Sum ((a), (b),  (c))

INTEGER FUNCTION Sum (x,y,z)
      IMPLICIT NONE
      INTEGER, INTENT (IN)  x,y,z
      ........
END FUNCTION Sum
```



Unde sunt accesate funcțiile: 1/2

- Funcțiile în Fortran pot fi interne sau externe.
- **Funcțiile interne** sunt scrise în **PROGRAM**:

```
PROGRAM nume_program
    IMPLICIT NONE
    [partea de specificație]
    [partea de execuție]
CONTAINS
    [funcții]
END PROGRAM nume_program
```

- Cu toate că o funcție poate conține alte funcții, funcțiile interne **nu pot conține** alte funcții interne.

Unde sunt accesate funcțiile: 2/2

- În partea dreaptă sunt prezentate două funcții interne **ArithMean()** și **GeoMean()**.
- Acestea folosesc două argumente reale de tip **real** care după calcule vor returna valori **reale**.

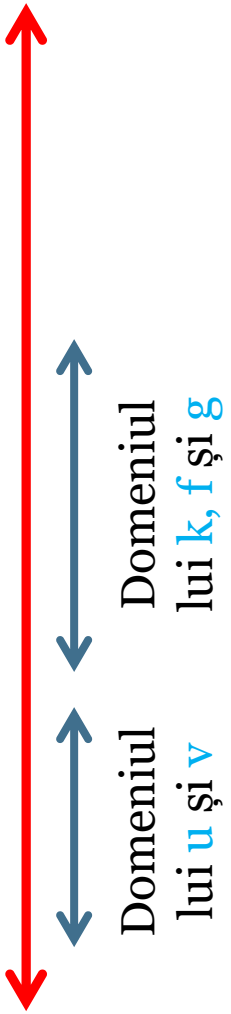
```
PROGRAM TwoFunctions
      IMPLICIT NONE
      REAL  a, b, A_Mean, G_Mean
      READ*, a, b
      A_Mean = ArithMean(a, b)
      G_Mean = GeoMean(a,b)
      PRINT*, a, b, A_Mean, G_Mean
CONTAINS
      REAL FUNCTION ArithMean(a, b)
            IMPLICIT NONE
            REAL, INTENT(IN)  a, b
            ArithMean = (a+b)/2.0
      END FUNCTION ArithMean
      REAL FUNCTION GeoMean(a, b)
            IMPLICIT NONE
            REAL, INTENT(IN)  a, b
            GeoMean = SQRT(a*b)
      END FUNCTION GeoMean
END PROGRAM TwoFunctions
```

Reguli pentru domeniul de aplicare a unor entități: 1/5

- Regulile pentru domeniul de aplicare a unor entități (de exemplu: variabile, parametrii și funcții) ne informează dacă acestea sunt vizibile sau accesibile în anumite locuri.
- Locurile unde pot fi accesate sau sunt vizibile unele entități reprezintă domeniul de aplicare a respectivelor entități.

Reguli pentru domeniul de aplicare a unor entități: 2/5

```
PROGRAM Scope_1
    IMPLICIT NONE
    REAL, PARAMETER  PI = 3.1415926
    INTEGER  m, n
    ..................................................
    CONTAINS
      INTEGER FUNCTION Funct1(k)
        IMPLICIT NONE
        INTEGER, INTENT(IN)  k
        REAL  f, g
        ..........................................
      END FUNCTION Funct1
      REAL FUNCTION Funct2(u, v)
        IMPLICIT NONE
        REAL, INTENT(IN)  u, v
        ..........................................
      END FUNCTION Funct2
END PROGRAM Scope_1
```



Regula nr. 1: Domeniul de aplicare al unei entități este programul sau funcția în care sunt declarate.

Reguli pentru domeniul de aplicare a unor entități: 3/5

```
PROGRAM Scope_2
    IMPLICIT NONE
    INTEGER  a = 1, b = 2, c = 3
    PRINT*, Add(a)
    c = 4
    PRINT*, Add(a)
    PRINT*, Mul(b,c)
CONTAINS
    INTEGER FUNCTION Add(q)
        IMPLICIT NONE
        INTEGER, INTENT(IN)  q
        Add = q + c
    END FUNCTION Add
    INTEGER FUNCTION Mul(x, y)
        IMPLICIT NONE
        INTEGER, INTENT(IN)  x, y
        Mul = x * y
    END FUNCTION Mul
END PROGRAM Scope_2
```

Regula nr. 2:
O **entitate globală** este **vizibilă** în toate funcțiile programului.

- a, b **şi** c **sunt globale**
- **Primul** Add(a) **returnează** 4
- **Al doilea** Add(a) **returnează** 5
- Mul(b, c) **returnează** 8

Cele două funcţii Add(a) returnează valori diferite cu toate că argumentele formale sunt aceleaşi! Acest lucru este numit efect secundar.

Evitaţi utilizarea de entităţi globale!

Reguli pentru domeniul de aplicare a unor entități: 4/5

```
PROGRAM Global
    IMPLICIT NONE
    INTEGER a = 10, b = 20
    PRINT*, Add(a,b)
    PRINT*, b
    PRINT*, Add(a,b)
CONTAINS
    INTEGER FUNCTION Add(x,y)
        IMPLICIT NONE
        INTEGER, INTENT(IN) x, y
        b = x+y
        Add = b
    END FUNCTION Add
END PROGRAM Global
```

- Primul Add (a, b) returnează 30
- De asemenea îl schimbă pe b în 30
- Cel de-al doilea PRINT ne arată 30
- Al doilea Add (a, b) returnează 40
- Acesta este un efect secundar foarte rău

Evitați utilizarea de entități globale!

Reguli pentru domeniul de aplicare a unor entități: 5/5

```
PROGRAM Scope_3
    IMPLICIT NONE
    INTEGER  i, Max = 5
    DO i = 1, Max
    PRINT*, Sum(i)
    END DO
CONTAINS
    INTEGER FUNCTION Sum(n)
        IMPLICIT NONE
        INTEGER, INTENT(IN)  n
        INTEGER  i, s
        s = 0
        ..... alte calcule.....
        Sum = s
    END FUNCTION Sum
END PROGRAM Scope_3
```

Regula nr. 3: O entitate declarată în cadrul altei entități este întotdeauna diferită chiar dacă numele celor două sunt identice.

Cu toate că atât programul cât şi funcţia Sum(n) au ca variabilă întreagă pe i, cele două variabile sunt entităţi diferite.

Astfel, orice modificare al variabilei i din Sum() nu va afecta variabila i din program.

Subrutina 1/2

- O funcție în Fortran primește prin argumentele formale și returnează **un singur rezultat** prin numele funcției.
- Subrutina primește date prin argumentele formale și returnează rezultate tot **prin intermediul argumentelor formale**.
- În Fortran subrutina nu returnează rezultate prin intermediul numelui acesteia.

Subrutina 2/2

- Sintaxa subrutinei în Fortran este:

```
SUBROUTINE nume_subrutină (arg1,arg2,...,argn)
    IMPLICIT NONE
    [partea de specificație]
    [partea de execuție]
    [partea de subprograme]
END SUBROUTINE nume_subrutină
```

- Dacă subrutina nu are argumente formale atunci "arg1, arg2,..., argn" pot fi scoase.

Atributul INTENT(): 1/2

- Din moment ce subrutina utilizează argumentele formale atât pentru a primi date cât și pentru a returna rezultate, pe lângă **INTENT(IN)** se va mai folosi **INTENT(OUT)** și **INTENT(INOUT)**.
- **INTENT(OUT)** înseamnă că argumentul formal nu primește o valoare, dar, va returna un rezultat argumentului real corespunzător.
- **INTENT(INOUT)** înseamnă că un argument formal primește date dar și returnează un rezultat argumentului real corespunzător.

Atributul INTENT(): 2/2

- Două exemple simple:

Am, Gm și Hm sunt utilizate pentru a returna rezultate

```
SUBROUTINE Means (a, b, c, Am, Gm, Hm)
    IMPLICIT NONE
    REAL, INTENT(IN)  a, b, c
    REAL, INTENT(OUT)  Am, Gm, Hm
    Am = (a+b+c)/3.0
    Gm = (a*b*c)**(1.0/3.0)
    Hm = 3.0/(1.0/a + 1.0/b + 1.0/c)
END SUBROUTINE Means
```

Valorile lui a și b sunt schimbate

```
SUBROUTINE Swap(a, b)
    IMPLICIT NONE
    INTEGER, INTENT(INOUT)  a, b
    INTEGER c
    c = a
    a = b
    b = c
END SUBROUTINE Swap
```

Instrucțiunea CALL: 1/2

- Spre deosebire de alte limbaje de programare, pentru a utiliza o subrutină trebuie folosită instrucțiunea CALL.
- Instrucțiunea CALL poate avea una dintre următoarele trei forme:

1. CALL nume_subrutină (arg1, arg2,...,argn)
2. CALL nume_subrutină ()
3. CALL nume_subrutină

- Formele 2 și 3 sunt echivalente și sunt utilizate pentru a chema o subrutină fără argumente reale.

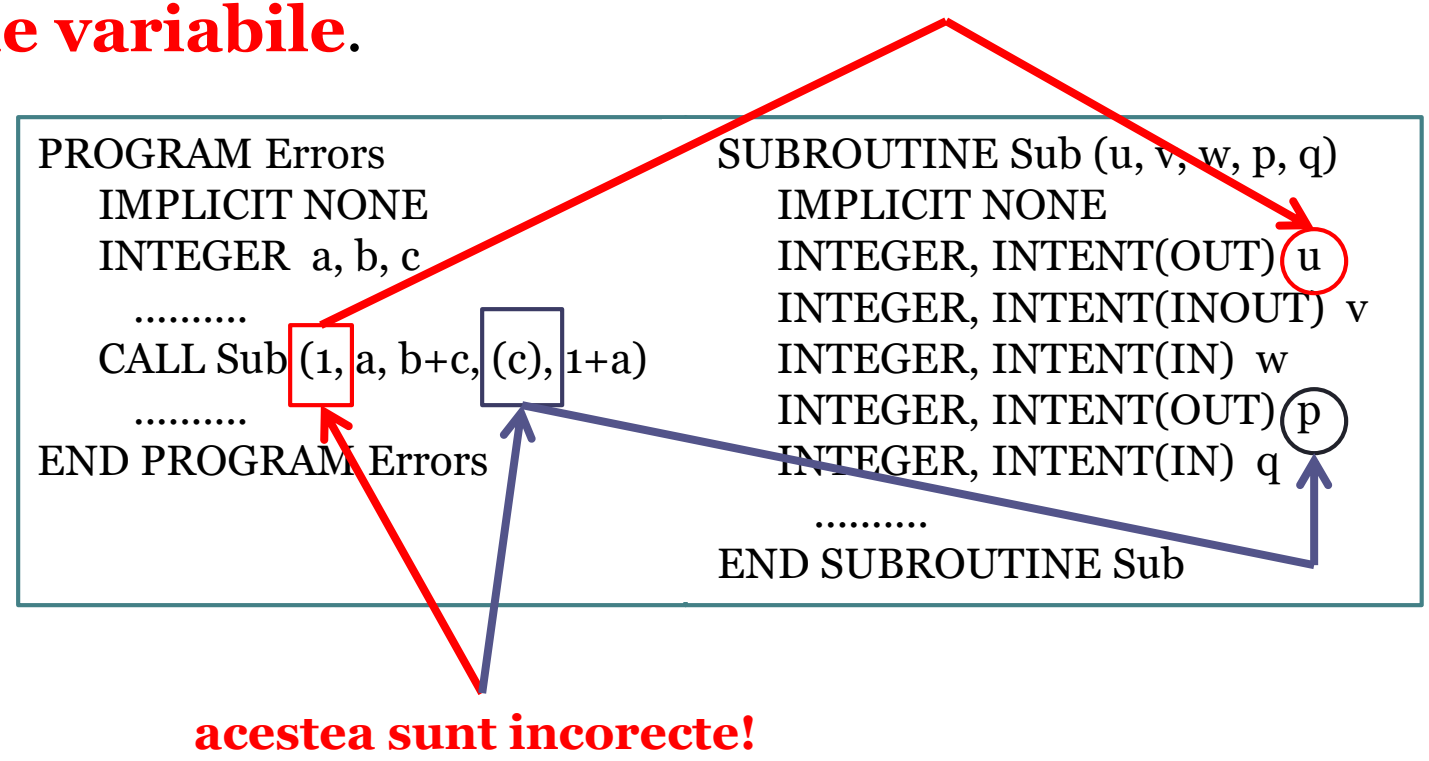
Instrucțiunea CALL: 2/2

```
PROGRAM Test
      IMPLICIT NONE
      REAL  a, b
      READ*, a, b
      CALL Swap (a,b)
      PRINT*,  a, b
CONTAINS
      SUBROUTINE Swap (x, y)
           IMPLICIT NONE
           REAL, INTENT(INOUT)  x,y
           REAL  z
           z = x
           x = y
           y = z
       END SUBROUTINE Swap
END PROGRAM Test
```

```
PROGRAM SecondDegree
      IMPLICIT NONE
      REAL  a, b, c, r1, r2
      LOGICAL  OK
      READ*, a, b, c
      CALL Solver (a,b,c,r1,r2,OK)
      IF (.NOT. OK) THEN
        PRINT*, "No root"
      ELSE
         PRINT*,  a, b, c, r1, r2
      END IF
CONTAINS
      SUBROUTINE Solver(a,b,c,x,y,L)
         IMPLICIT NONE
         REAL, INTENT (IN)  a, b, c
         REAL, INTENT(OUT)  x, y
         LOGICAL, INTENT(OUT)  L
         .....
      END SUBROUTINE Solver
END PROGRAM SecondDegree
```

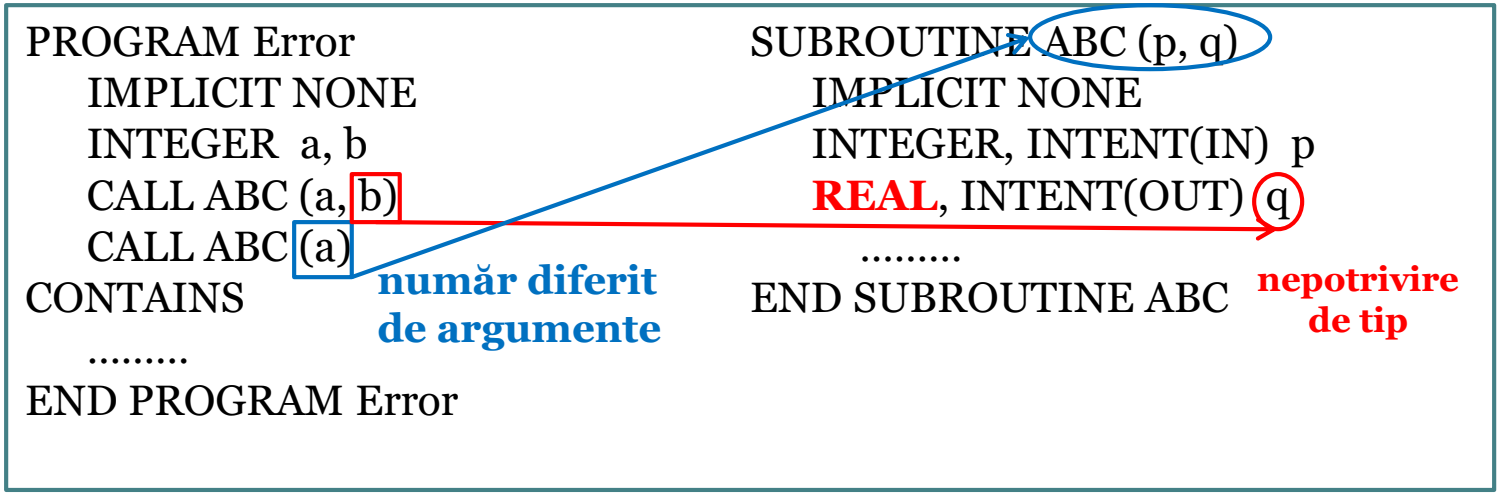
Asocierea argumentelor: 1/2

- Odată ce argumentele formale cu atributul INTENT(IN) sau INTENT(INOUT) transferă rezultate argumentelor reale corespunzătoare, **argumentele reale trebuie să fie variabile**.



Asocierea argumentelor: 2/2

- Numărul de argumente precum și tipul acestora trebuie să fie același.
- Nu există nici un fel de conversie a tipului între argumente!



Tehnici de testare a programelor

O dată ce un program a fost alcătuit, în mod natural se pune problema corectitudinii sale astfel că trebuie să ne asigurăm că el îndeplinește funcția pe care o dorim și nu alta.

Cea mai simplă tehnică de analiză a corectitudinii programelor este așa-numita *testare,* metodă de verificare prin exemple concrete.

Pentru a realiza testarea unui program vom utiliza *tabele trasoare* ce vor arată valorile succesive în memorie ale unui program. Exemplul utilizat va fi tabelarea funcției $y = x^2 + 2$ de la valoarea inițială xi, la valoarea finală xf cu pasul pas.

Tehnici de testare a programelor

Instrucțiuni	Memorie	Output

read xi, xf, pas;
x=xi;
do while x≤xf
y=x² + 2;
write x, y;
x=x + pas;
end do
end
	xi = 1
	xf = 1
	pas = 1
	x = 1 2 3 4 5 6
	y = 3 6 11 18 27
		1 3
		2 6
		3 11
		4 18
		5 27

Tehnici de testare a programelor

Tehnica verificării cu tabele trasoare este foarte utilă mai ales pentru programatorii începători. Tehnica este intuitivă şi foarte eficace.

În afară de tehnica testării cu programe trasoare mai există o formă de testare, aşa numita *testare* experimentă.

Tehnica testării experimentale este una din cele mai răspândite tehnici de testare a programelor tehnico-ştiinţifice. Programele performante în literatura de specialitate de regulă conţin în documentaţia lor o *colecţie de probleme test* alcătuită din:

- i) set de date de intrare şi rezultate corespunzătoare,
- ii) set de *test drivers*, programe de antrenare, pentru rezolvarea problemelor test.

Tehnici de testare a programelor

Trebuie să remarcăm că testarea experimentală a unui program nu este echivalentă cu demonstrația corectitudinii programului.

Testarea experimentală a programului poate servi ca demonstrație a prezentei erorilor și nicidecum ca o demonstrație a lipsei lor.

Pentru a demonstra corectitudinea programelor există metode analitice de exemplu tehnica lui Hoare, în genul demonstrațiilor matematice.

Bibliografie

Material realizat de Thomas E. Kurtz, Co-Designer of the BASIC language

- *Octavian PETRUȘ, Fortran 90/95, Limbaj și Tehnici de programare, Editura Universității Tehnice "Gheorghe Asachi" din Iași, 2001*
- Romeo CHELARIU, Sisteme de operare şi limbaje de programare (Îndrumar de laborator), http://www.sim.tuiasi.ro/wp-content/uploads/Chelariu-indrumar-solp.pdf, 2004
- https://ro.wikipedia.org