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

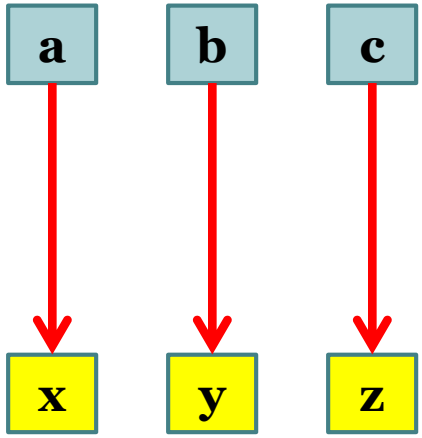
- **Asocierea argumentelor** este o modalitate de a transfera conținutul de la argumentele reale la argumentele formale.
- În cazul în care un argument real este o **expresie**, acesta se evaluează și se stochează într-o **locație temporară** după care valoarea este transferată la argumentul formal corespunzător.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

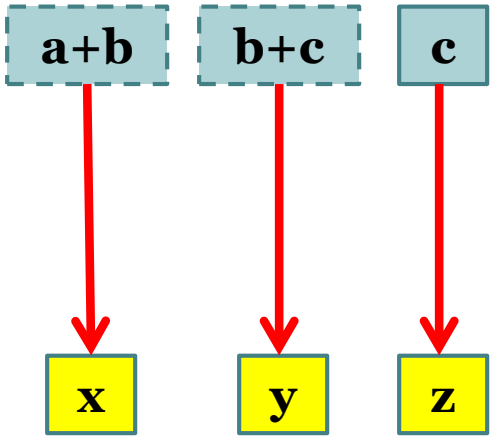


Asocierea argumentelor: 3/5

- 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
```

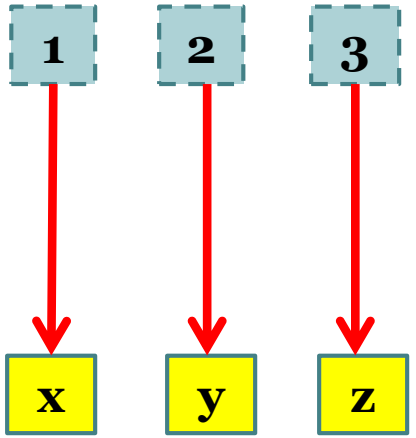


Asocierea argumentelor: 4/5

- 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
```

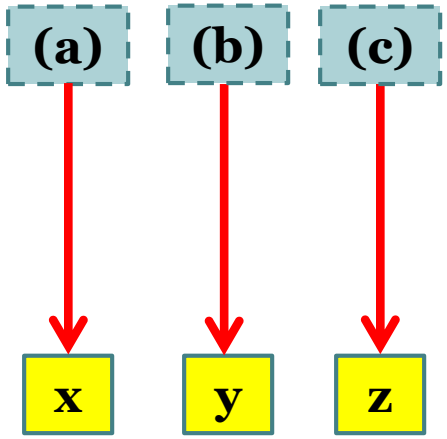


Asocierea argumentelor: 5/5

- 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**.

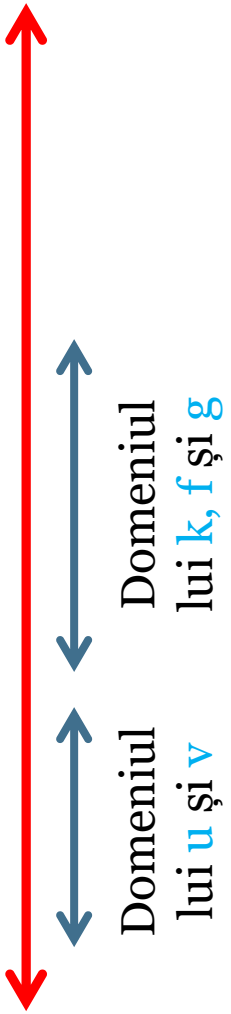
```
PROGRAM TwoFunctions
      IMPLICIT NONE
      REAL  a, b, A_Mean, G_Mean
      READ*, a, b
      A_Mean = ArithMean(a, b)
      G_Mean = GeoMean(a,b)
      PRINT*, a, b, A_Mean, G_Mean
CONTAINS
      REAL FUNCTION ArithMean(a, b)
            IMPLICIT NONE
            REAL, INTENT(IN)  a, b
            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.

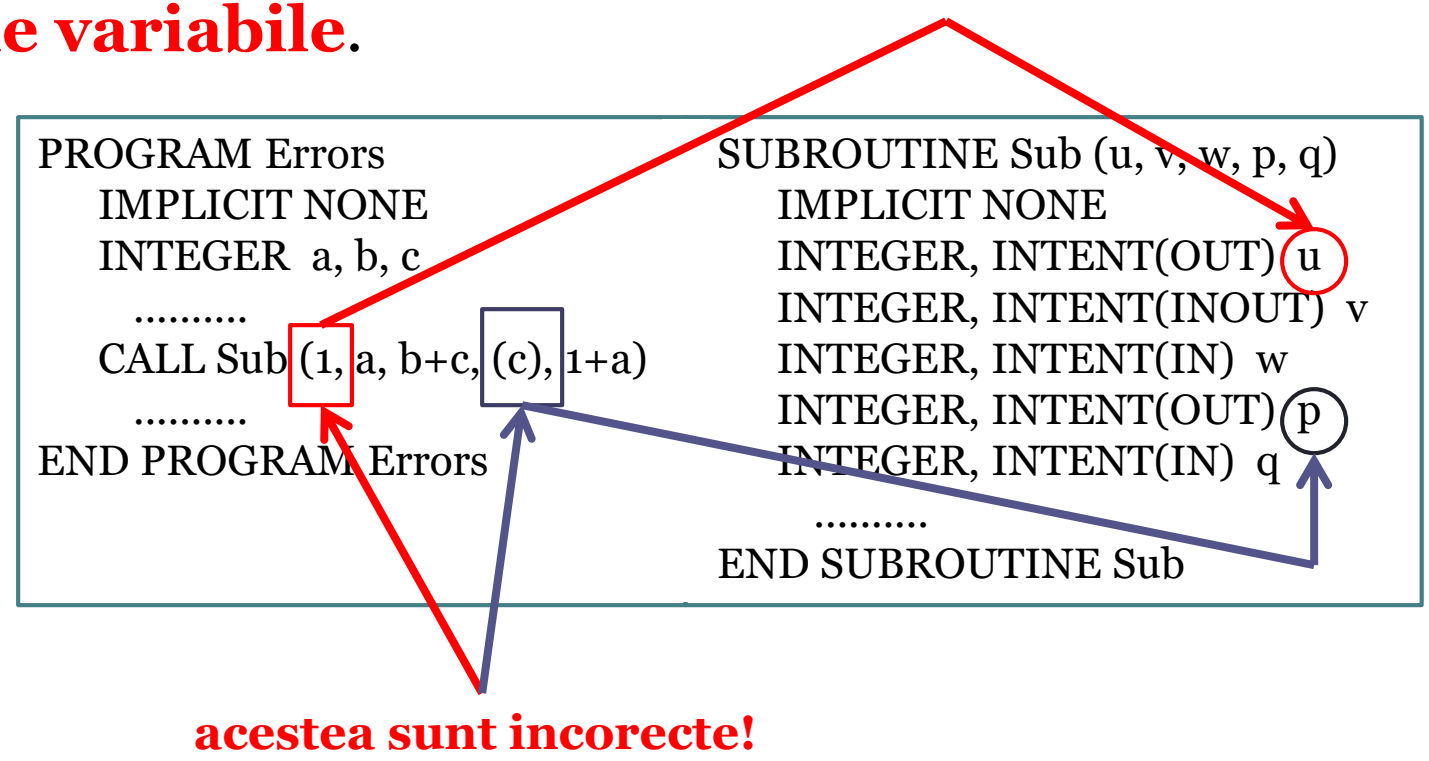
Instrucțiunea CALL: 2/2

```
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
```

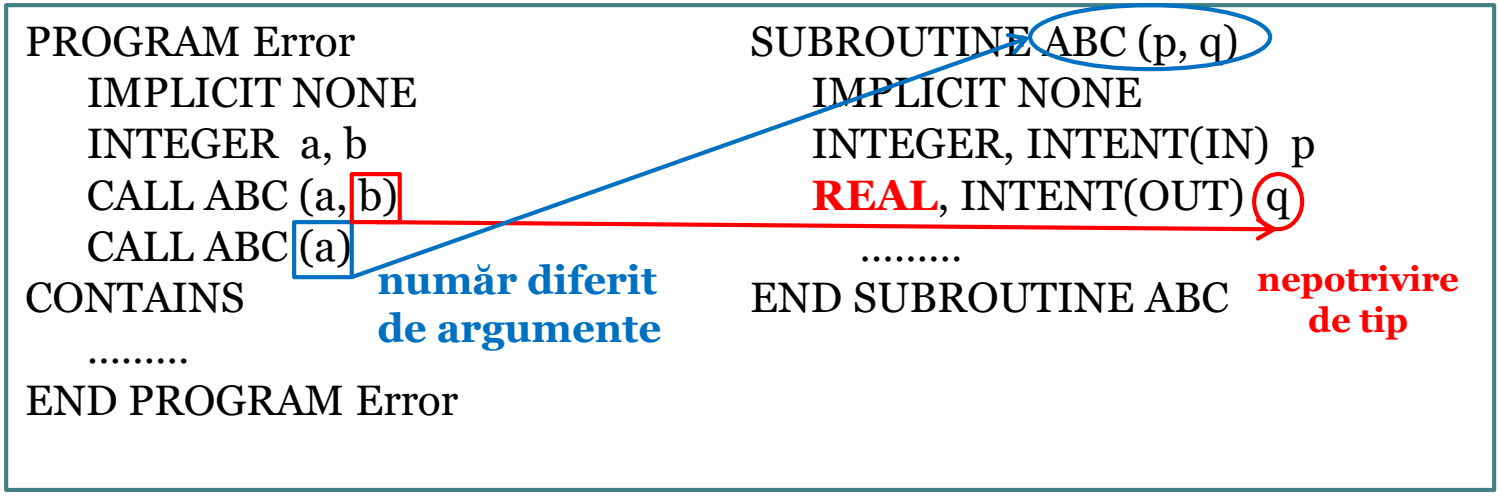
Asocierea argumentelor: 1/2

- 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