زیر برنامه (Sub-Programs)

- در الگوریتم های پیچیده، نوشتن همه مساله در قالب یک برنامه یکپارچه، مشکل و گاها غیر ممکن است. در این موارد الگوریتم برنامه را به زیر برنامه های کوچکتری تقسیم نموده و پس از طراحی الگوریتم زیر برنامه، آنها را در الگوریتم اصلی فراخوانی می کنیم. زیر برنامه ها در زبان فرترن به دو صورت تابع (FUNCTION) و زیر روال (SUBROUTINE) می باشند.

- هر زیر برنامه دارای دو جنبه می باشد:
  - جنبه تعریف: که شامل مجموعه ای از دستورات و عملکردهاست.
  - جنبه فراخوان: که شامل دستوری برای ارتباط دادن زیر برنامه به برنامه اصلی است.

زیر برنامه (تابع–FUNCTION)

- در فرترن ۹۰ یک تابع(FUNCTION) دارای ساختار زیر است:

```
FUNCTION function-name (arg1, arg2, ..., argn)
IMPLICIT NONE
[specification part]
[execution part]
[subprogram part]
END FUNCTION function-name
```

- type در فرترن۹۰ همان دستور تعیین نوع می باشد (یعنی ...,REAL, INTEGER)
- function-name از قوانین تعیین معرف (identifier) پیروی می کند.
- arg1, arg2, ..., argn آرگومان های تابع نام دارند.

زیر برنامه (تابع–FUNCTION)

- زیر برنامه تابع(FUNCTION) یک واحد مجزای برنامه است که تعدادی ورودی را از جهان خارج، از طریق آرگومان هایش می گیرد، محاسبات را انجام داده و نتایج را از طریق نام تابع(function-name) برمی گرداند.
- در جایی از یک زیر برنامه تابع(FUNCTION) می بایست یک عبارت انتسابی به صورت زیر وجود داشته باشد:

function-name = *expression*

که در آن نتیجه *expression* *در نام تابع ذخیره می شود.*

- تذکر: نام تابع(function-name)نمی تواند در سمت راست هیچ عبارت (*expression*) ظاهر شود.

زیر برنامه (تابع–FUNCTION)

- در قسمت تعیین نوع، آرگومان ها باید دارای ویژگی INTENT(IN) باشند.
- منظور از ویژگی INTENT(IN) این است که زیر برنامه تابع، تنها می تواند مقادیر آرگومان ها را دریافت نموده و نمی تواند هیچگونه تغییری در آن ایجاد نماید. به عبارت دیگر آرگومان تنها می تواند داده های ورودی را برای زیر برنامه تابع فراهم نماید.
- زیر برنامه تابع، یک واحد مجزای برنامه است و تمامی دستوراتی که در برنامه اصلی به کار می رود در زیر برنامه تابع نیز می تواند به کار رود.
- زیر برنامه تابع می تواند فاقد آرگومان باشد. با این وجود علامت ( ) در زمان تعریف نام زیر روال تابع، ضروری می باشد.

زیر برنامه (تابع–FUNCTION)

- مثال: زیر برنامه تابع محاسبه فاکتوریل عدد n

```
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
```

زیر برنامه (تابع–FUNCTION)

- اشتباهات متداول

```
REAL FUNCTION DoSomething(a, b)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: a, b
        IF (a > b) THEN
          a = a - b
        ELSE
          a = a + b
    END IF
    DoSomthing = SQRT(a*a+b*b)
END FUNCTION DoSomething
```

change INTENT(IN) argument

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

forget to return a value

زیر برنامه (تابع–FUNCTION)

- اشتباهات متداول

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

forget function type

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

forget INTENT(IN)

– not an error

زیر برنامه (تابع–محل قرار گرفتن زیر برنامه تابع)

- زیر برنامه تابع در فرترن ۹۰ به دو صورت داخلی و خارجی در برنامه اصلی قرار می گیرد.
- در زیر برنامه داخلی، تابع درون برنامه اصلی درست پیش از دستور END PROGRAM قرار دارد.
  ساختار آن به صورت زیر است:

```
PROGRAM program-name
    IMPLICIT NONE
    [specification part]
    [execution part]
CONTAINS
    [functions]
END PROGRAM program-name
```

- در حالت کلی هر زیر برنامه تابع می تواند شامل تعداد دلخواهی زیر برنامه تابع دیگر باشد، اما در زیر برنامه تابع داخلی این امر امکان پذیر نمی باشد.

زیر برنامه (FUNCTION–تابع)

- مثال: برنامه ای بنویسید که دو عدد را از کاربر دریافت نماید و میانگین ریاضی و میانگین هندسی دو عدد را محاسبه نماید؟

– این برنامه با استفاده از زیر برنامه تابع داخلی نوشته شده است

```
PROGRAM twofunctions
IMPLICIT NONE
REAL :: a, b, A_Mean, G_Mean
WRITE(*,*) "Please enter two numbers"
READ(*,*) a, b
A_Mean = ArithMean(a, b)
G_Mean = GeoMean(a,b)
WRITE(*,*) a, b, A_Mean, G_Mean
CONTAINS
   REAL FUNCTION ArithMean(x, y)
   IMPLICIT NONE
   REAL, INTENT(IN) :: x, y
   ArithMean = (x+y)/2.0
   END FUNCTION ArithMean
   REAL FUNCTION GeoMean(z, k)
   IMPLICIT NONE
   REAL, INTENT(IN) :: z, k
   GeoMean = SQRT(z*k)
   END FUNCTION GeoMean
END PROGRAM twofunctions
```

زیر برنامه (تابع–محل قرار گرفتن زیر برنامه تابع)

- زیر برنامه تابع در فرترن ۹۰ به دو صورت داخلی و خارجی در برنامه اصلی قرار می گیرد.
- در زیر برنامه خارجی، تابع خارج برنامه اصلی درست پس از دستور END PROGRAM قرار دارد.

ساختار آن به صورت زیر است:

```
PROGRAM program-name
    IMPLICIT NONE
    [specification part]
    [execution part]
END PROGRAM program-name
    [functions]
```

زیر برنامه (FUNCTION–تابع)

- مثال: برنامه ای بنویسید که دو عدد را از کاربر دریافت نماید و میانگین ریاضی و میانگین هندسی دو عدد را محاسبه نماید؟

– این برنامه با استفاده از زیر برنامه تابع خارجی نوشته شده است

```
PROGRAM twofunctions
IMPLICIT NONE
REAL :: a, b, A_Mean, G_Mean
WRITE(*,*) "Please enter two numbers"
READ(*,*) a, b
A_Mean = ArithMean(a, b)
G_Mean = GeoMean(a,b)
WRITE(*,*) a, b, A_Mean, G_Mean
END PROGRAM twofunctions
```

```
REAL FUNCTION ArithMean(x, y)
IMPLICIT NONE
REAL, INTENT(IN) :: x, y
ArithMean = (x+y)/2.0
END FUNCTION ArithMean
```

```
REAL FUNCTION GeoMean(z, k)
IMPLICIT NONE
REAL, INTENT(IN) :: z, k
GeoMean = SQRT(z*k)
END FUNCTION GeoMean
```

زیر برنامه (تابع–FUNCTION)

- در زیر برنامه آرگومان ها به دو صورت تعریف می شوند:
  - آرگومان رسمی (formal argument) که به آرگومان بدلی (dummy argument) نیز نامیده می شود. عموما آرگومان های به کار رفته در زیر برنامه را گویند.
  - آرگومان اصلی (actual argument)، آرگومان تعریف شده در زیر برنامه ای که در برنامه اصلی به کار می رود، است.
  - روشی که در آن مقادیر آرگومان های اصلی به آرگومان های بدلی انتقال می یابد **Argument Association** نام دارد.
  - تعداد و جنس آرگومان های بدلی با آرگومان های اصلی می بایست یکسان باشد.

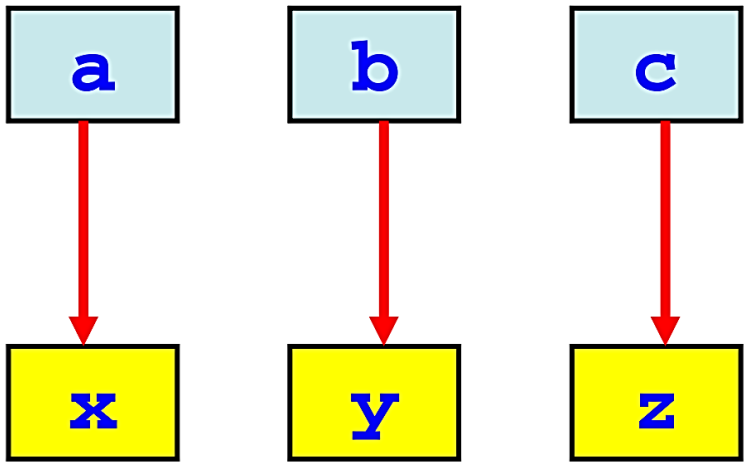
زیر برنامه (تابع– Argument Association)

- اگر آرگومان های اصلی متغیر باشند:

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



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



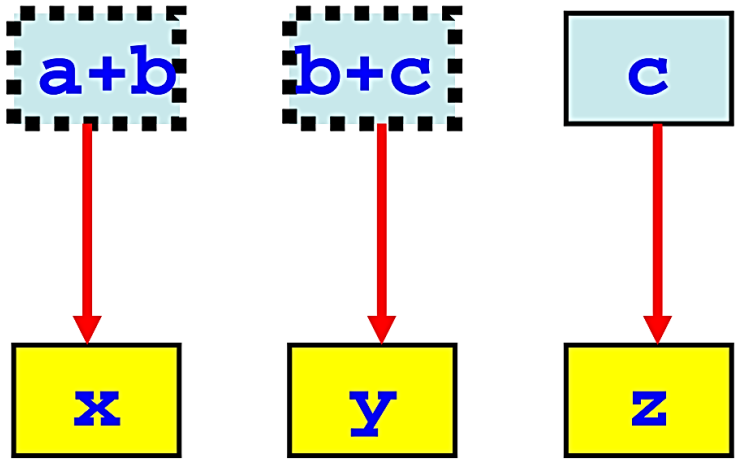
زیر برنامه (تابع– Argument Association)

- اگر آرگومان های اصلی شامل عبارت باشند:
  - در این حالت ابتدا مقدار عبارت محاسبه شده و در یک مکان موقت ذخیره شده، سپس این مقدار از آرگومان اصلی به آرگومان بدلی انتقال می یابد. (مکان های دارای خط تیره نشان دهنده مکان های موقت می باشد)

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



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



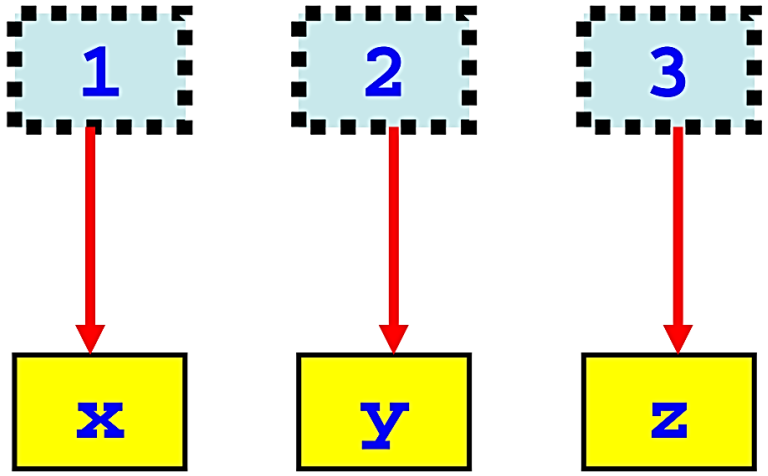
زیر برنامه (تابع– Argument Association)

- اگر آرگومان های اصلی شامل ثابت ها باشند:
  - (مکان های دارای خط تیره نشان دهنده مکان های موقت می باشد)

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



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



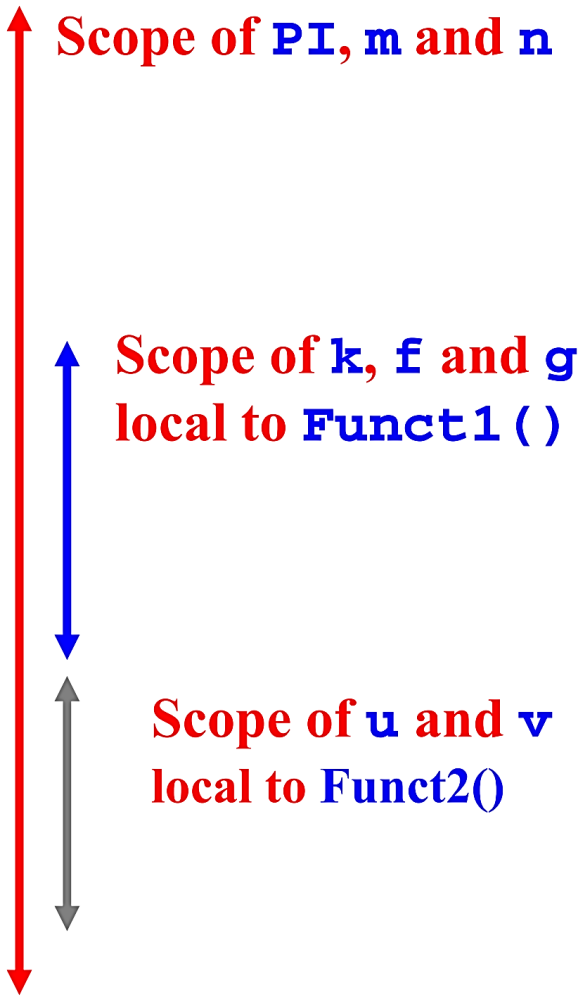
زیر برنامه (حوزه تاثیر)

- در برنامه ای که شامل زیر برنامه است ممکن است معرف ها (مانند متغیرها، ثابت ها و...) در چندین نقطه از برنامه اصلی، زیر برنامه ها و یا مدول ها (ماژول ها) تعریف شوند. قسمتی از برنامه که در آن معرف های یاد شده قابل دسترس و تاثیر گذار هستند، حوزه تاثیر آن معرف نامیده می شود.
- معرف ها بر اساس حوزه تاثیر به دو دسته تقسیم می شوند:
  - معرف ها (یا کمیت های) محلی(local): معرف هایی که داخل زیر برنامه تعریف شود کمیت محلی نام دارد.
  - معرف ها (یا کمیت های) عمومی(global): معرف هایی که در برنامه اصلی تعریف می شود کمیت عمومی نام دارد.

زیر برنامه (حوزه تاثیر)

❑ قانون اصلی: حوزه تاثیر هر معرف(یا کمیت)، برنامه اصلی یا زیر برنامه ای است که در آن تعریف شده است.

```
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
```



زیر برنامه (حوزه تاثیر)

- **قانون اول**: کمیت های محلی در خارج از زیر برنامه قابل دسترسی نیست. اما کمیت عمومی در تمام برنامه قابل دسترسی خواهد بود مگر اینکه در زیر برنامه کمیت محلی به همان نام تعریف شده باشد.

- **نکته**: برنامه اصلی (MAIN PROGRAM) و زیر برنامه FUNCTION Sum(n) هر دو دارای متغیر صحیح i می باشند، اما آنها دو معرف (کمیت) کاملا متفاوت هستند. از این رو هر تغییری در i برنامه اصلی تاثیری بر i موجود در زیر برنامه نخواهد داشت.

```
PROGRAM Scope_3
   IMPLICIT NONE
   INTEGER :: i, Max = 5
   DO i = 1, Max
     Write(*,*) Sum(i)
   END DO
CONTAINS
   INTEGER FUNCTION Sum(n)
     IMPLICIT NONE
     INTEGER, INTENT(IN) :: n
     INTEGER :: i, s
     s = 0
     …… other computation ……
     Sum = s
   END FUNCTION Sum
END PROGRAM Scope_3
```

زیر برنامه (حوزه تاثیر)

- قانون دوم: یک کمیت عمومی در تمام برنامه اصلی و زیر برنامه هایی که کمیت محلی با همان نام نداشته باشند، قابل دسترسی است.

```
PROGRAM Global
   IMPLICIT NONE
   INTEGER ::  a = 10, b = 20
   WRITE(*,*)  Add(a,b)
   WRITE(*,*)  b
   WRITE(*,*)  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
```

- The first `Add(a,b)` returns `30`
- It also changes `b` to `30`
- The 2nd `WRITE(*,*)` shows `30`
- The 2nd `Add(a,b)` returns `40`
- This is a bad side effect
- **Avoid using global entities!**

زیر برنامه (مثال کاربردی)

زیر برنامه (زیر روال–SUBROUTINE)

- زیر برنامه تابع به گونه ای طراحی شده است که مقادیر ورودی را از طریق آرگومان هایش گرفته و یک مقدار را از طریق نام تابع (*function-name*) بر می گرداند.
- زیر روال به گونه ای طراحی شده که مقادیر ورودی را از طریق آرگومان هایش گرفته و بیش از یک مقدار را بر می گرداند. البته زیر روال گاهی هیچ مقداری را بر نمی گرداند و تنها برخی اعمال نظیر نمایش دستورات کاربر را انجام می دهد.
- زیر برنامه تابع مقدار خود را از طریق نام تابع باز می گرداند، در صورتی که مقادیر بازگردانده شده به برنامه ای که زیر روال را صدا زده است از طریق آرگومان های به کار رفته در زیر روال تبادل می گردد.

زیر برنامه (SUBROUTINE–زیر روال)

- شکل کلی زیر روال در فرترن ۹۰ به صورت زیر است:

```
SUBROUTINE subroutine-name(arg1,arg2,...,argn)
    IMPLICIT NONE
    [specification part]
    [execution part]
    [subprogram part]
END SUBROUTINE subroutine-name
```

- اگر زیر روال نیاز به آرگومان های بدلی نداشته باشد arg1,arg2,...,argn را می توان حذف نمود. اما همچنان علامت ( ) می بایست پس از نام زیر روال وجود داشته باشد.
- زیر روال ها شبیه به تابع ها هستند.

زیر برنامه (زیر روال– ویژگی INTENT( ))

- زیر روال ها از آرگومان های بدلی جهت دریافت ورودی ها و بازگرداندن نتایج محاسبات استفاده می کند.
- ویژگی INTENT به دستور اعلام نوع مربوط شده و آرگومان های بدلی رامعرفی می کند. این ویژگی می تواند یکی از سه شکل زیر را دارا باشد:
- INTENT (IN): آرگومان بدلی فقط برای انتقال داده های ورودی به زیر روال استفاده می شود.
- INTENT (OUT): آرگومان بدلی فقط برای بازگرداندن نتایج به برنامه فراخوان استفاده می شود.
- INTENT (IN): آرگومان بدلی هم برای انتقال داده های ورودی به زیر روال و هم برای بازگرداندن نتایج به برنامه فراخوان استفاده می شود.

  - منظور از ویژگی INTENT این است که کامپایلر اطلاع دهد برنامه نویس از هر آرگومان ساختگی چگونه استفاده کند.

زیر برنامه (زیر روال– ویژگی INTENT( ))

- مثال:

```
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
```

Am, Gm and Hm are used to return the results

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

values of a and b are swapped

زیر برنامه (زیر روال– دستور CALL)

- به منظور فراخوانی زیر روال در برنامه اصلی از دستور CALL استفاده می شود. این دستور به ۳ صورت زیر روال را در برنامه اصلی فراخوانی می کند:

- CALL sub-name(arg1,arg2,…,argn)
- CALL sub-name( )
- CALL sub-name

- دو شکل آخر دستور CALL با هم معادل بوده و به منظور فراخوانی زیر روالی که فاقد آرگومان است در برنامه اصلی به کار می رود.

زیر برنامه (زیر روال– دستور CALL)

```
PROGRAM Test
    IMPLICIT NONE
    REAL :: a, b
    READ(*,*) a, b
    CALL Swap(a,b)
    WRITE(*,*) 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
       WRITE(*,*) "No root"
    ELSE
       WRITE(*,*) 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
```

زیر برنامه (زیر روال- آرگومان ها)

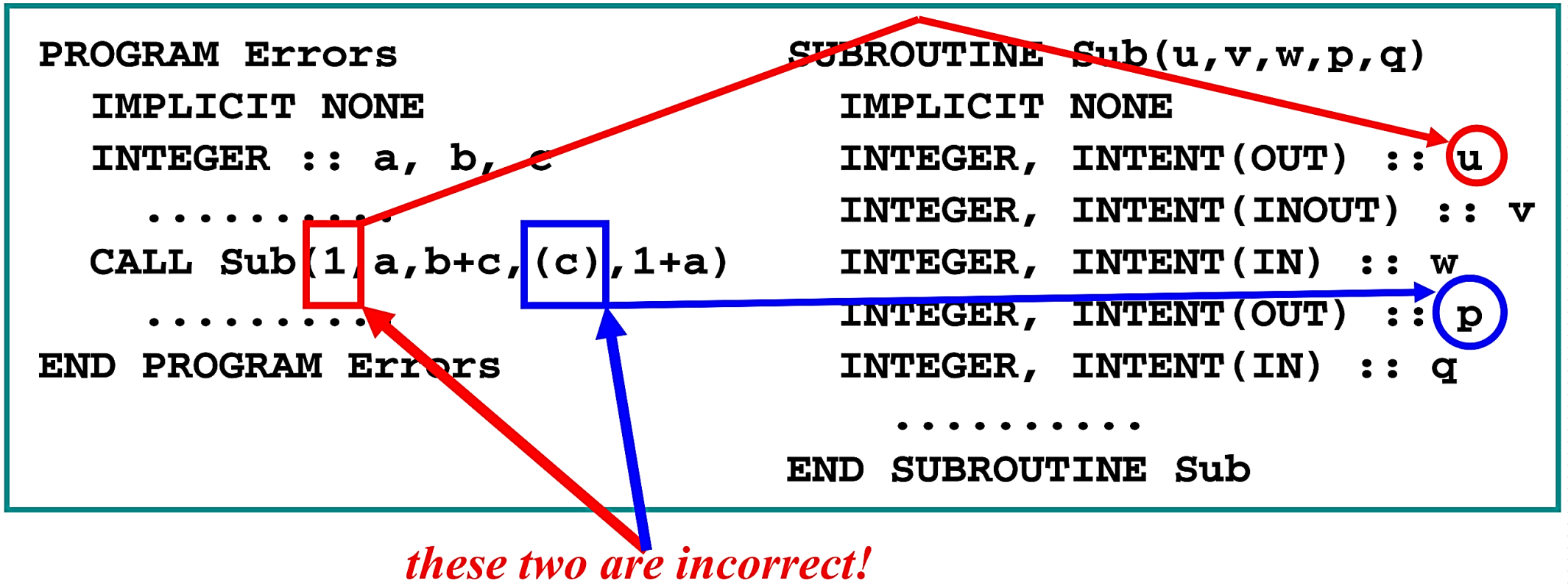
- هنگامی که آرگومان های بدلی همراه با INTENT(OUT) و INTENT(INOUT) مقادیر خروجی را به آرگومان های اصلی برمی گردانند، آرگومان های اصلی می بایست حتما از نوع متغیر باشند.

```
PROGRAM Errors
    IMPLICIT NONE
    INTEGER :: a, b, c
        .........
    CALL Sub(1,a,b+c,(c),1+a)
.........
END PROGRAM Errors
```

```
SUBROUTINE Sub(u,v,w,p,q)
    IMPLICIT NONE
    INTEGER, INTENT(OUT) :: u
    INTEGER, INTENT(INOUT) :: v
    INTEGER, INTENT(IN) :: w
    INTEGER, p INTENT(OUT) :: p
    INTEGER, INTENT(IN) :: q
        .........
END SUBROUTINE Sub
```

زیر برنامه (زیر روال- آرگومان ها)

- هنگامی که آرگومان های بدلی همراه با INTENT(OUT) و INTENT(INOUT) مقادیر خروجی را به آرگومان های اصلی برمی گردانند، آرگومان های اصلی می بایست حتما از نوع متغیر باشند.



زیر برنامه (زیر روال- آرگومان ها)

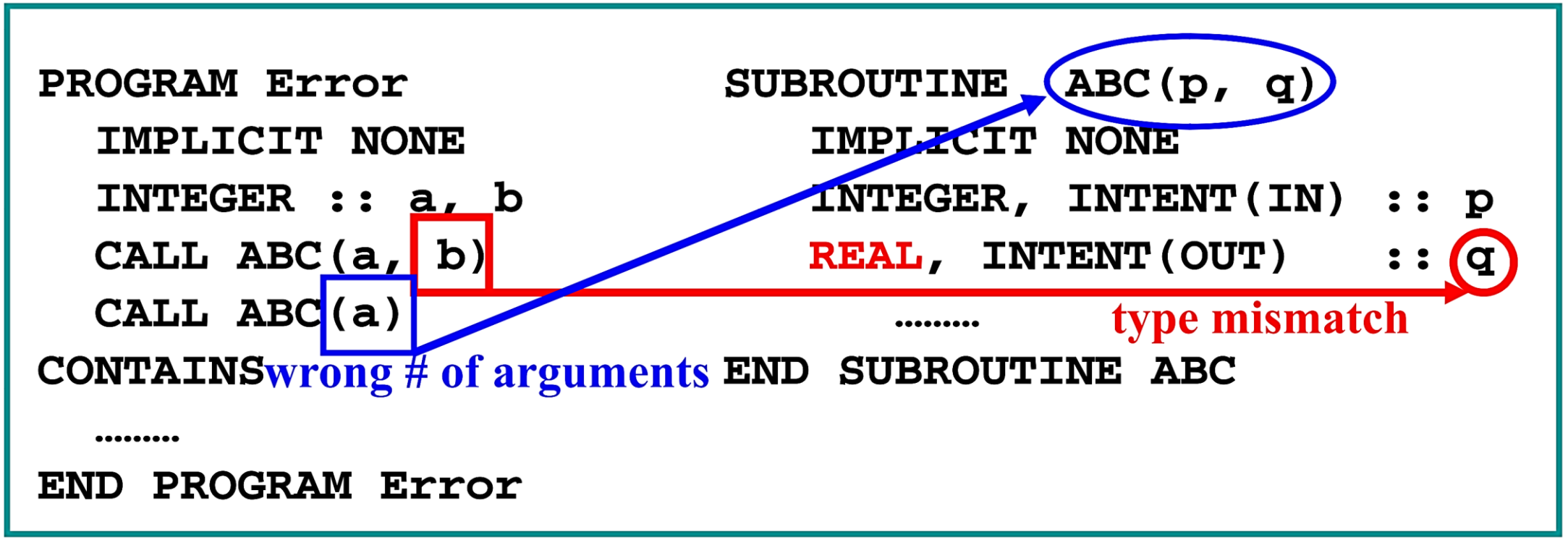
- تعداد و نوع آرگومان های اصلی و بدلی می بایست با هم یکسان باشند.
- هیچ تبدیل یا تغییر نوع بین آرگومان های متناظر اصلی و بدلی مجاز نیست.

```
PROGRAM Error
    IMPLICIT NONE
    INTEGER :: a, b
    CALL ABC(a, b)
    CALL ABC(a)
CONTAINS
    ……
END PROGRAM Error
```

```
SUBROUTINE ABC(p, q)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: p
    REAL, INTENT(OUT) :: q
    ……
END SUBROUTINE ABC
```

زیر برنامه (زیر روال- آرگومان ها)

- تعداد و نوع آرگومان های اصلی و بدلی می بایست با هم یکسان باشند.
- هیچ تبدیل یا تغییر نوع بین آرگومان های متناظر اصلی و بدلی مجاز نیست.



مدول (یا ماژول –MODULE)

- در بخش های قبل دیدیم که زیر برنامه ها واحدهای مجزای برنامه هستند که به منظور انجام محاسبات خاصی نوشته می شوند. بیشتر زیر برنامه ها قابل استفاده در سایر برنامه ها هستند. فرترن ۹۰ برای این منظور از ماژول ها استفاده می کند.
- یک ماژول واحد برنامه ای است که تعاریف جنس یا نوع داده و زیر برنامه ها را درون خود جای داده است. شکل کلی آن به صورت زیر است:

```
MODULE module-name
    IMPLICIT NONE
    [specification part]
CONTAINS
    [internal functions/subroutines]
END MODULE module-name
```

- بخش های تعریف (شامل تعریف ثابت ها م متغیر ها) و زیر برنامه(توابع و زیر روال ها) اختیاری هستند.

مدول (یا ماژول –MODULE)

- مثال:

Module SomeConstants does not have the subprogram part

```
MODULE SomeConstants
  IMPLICIT NONE
  REAL, PARAMETER :: PI=3.1415926
  REAL, PARAMETER :: g = 980
  INTEGER :: Counter
END MODULE SomeConstants
```

Module SumAverage does not have the specification part

```
MODULE SumAverage

CONTAINS
  REAL FUNCTION Sum(a, b, c)
    IMPLICIT NONE
    REAL, INTENT(IN) :: a, b, c
    Sum = a + b + c
  END FUNCTION Sum
  REAL FUNCTION Average(a, b, c)
    IMPLICIT NONE
    REAL, INTENT(IN) :: a, b, c
    Average = Sum(a,b,c)/2.0
  END FUNCTION Average
END MODULE SumAverage
```

مدول (یا ماژول –MODULE)

- مثال: ماژول رو به رو هم قسمت تعریف و هم بخش زیر برنامه را دارا می باشد.

```
MODULE DegreeRadianConversion
  IMPLICIT NONE
  REAL, PARAMETER :: PI = 3.1415926
  REAL, PARAMETER :: Degree180 = 180.0

CONTAINS
  REAL FUNCTION DegreeToRadian(Degree)
    IMPLICIT NONE
    REAL, INTENT(IN) :: Degree
    DegreeToRadian = Degree*PI/Degree180
  END FUNCTION DegreeToRadian

  REAL FUNCTION RadianToDegree(radian)
    IMPLICIT NONE
    REAL, INTENT(IN) :: Radian
    RadianToDegree = Radian*Degree180/PI
  END FUNCTION RadianToDegree

END MODULE DegreeRadianConversion
```

مدول (یا ماژول –MODULE)

❑ تفاوت ماژول و برنامه اصلی:

- ماژول تقریبا ساختاری مشابه با ساختار برنامه اصلی دارد.با این وجود یک ماژول با MODULE آغاز و با END MODULE پایان می یابد.
- ماژول ها دارای بخش تعریف و زیر برنامه هستند، اما هیچ دستوری نمی تواند بین بخش تعریف و کلمه CONTAIN قرار بگیرد. از این رو ماژول ها نمی توانند همانند برنامه های اصلی دارای قسمت دستورات اجرایی باشند.
- از این رو ماژول ها هرگز نمی توانند به تنهایی به کار روند و همواره توسط برنامه اصلی و همچنین ماژول دیگر استفاده می شوند.

مدول (PUBLIC/PRIVATE)

- در فرترن ۹۰ روشی مهیا شده که در آن برنامه ای که از ماژول استفاده می کند، تنها بتواند به موارد معینی در درون ماژول دسترسی داشته باشد. این نوع کنترل دسترسی به درون ماژول، توسط دستورهای PUBLIC/PRIVATE صورت می پذیرد.