فصل هفتم
کنترل داده ها

by: Mahdi Sadeghizade
Email: Mahdi_sadeghe@yahoo.com
Site: www.Sadeghizadeh.ir

در این بخش چگونگی قرار دادن اطلاعات جهت عملیات مختلف و مکانیزم های ذخیره نتایج میانی و بازیابی آن جهت عملیات بعدی تعیین می شود .

شیوه های مختلف قرار دادن اطلاعات جهت عملیات

1. **انتقال مستقیم :**

در هر نقطه از زیربرنامه نتیجه یک عمل به طور مستقیم برای عمل بعدی استفاده می شود . مثلا در عبارت x:=y+2*z نتیجه ی 2*z برای عمل جمع به عنوان یک عملوند در محاسبات به کار گرفته می شود (به صورت مستقیم)

2. **مراجعه به DO از طریق نام آن (انتقال غرمستقیم)** : یک DO ممکن است به هنگام ایجاد ، نامگذاری شود و نام آن به عنوان یک عملوند جهت عملیات به کار برده شود . مانند x,y,z در عبارت بالا .

☑ انتقال مستقیم بیشتر برای کنترل داده ها در داخل عبارت استفاده می شود و اکثر کنترل داده های خارج از عبارات شامل استفاده از نامها هستند .

محیط ارجاع RE: Reference Environment

هر زیربرنامه دارای یک مجموعه شناسه می باشد که در طول اجرا از آنها استفاده می کند . به این مجموعه ، محیط ارجاع گفته می شود .

انواع محیط ارجاع

1. **Local RE**

مجموعه ای از شناسه هایی که در داخل زیربرنامه تعریف شده اند می باشند . مانند پارامترهای ورودی به آن و متغیر های محلی آن .

2. **NoneLocal RE**

مجموعه ای از شناسه هایی که می توانند در زیربرنامه استفاده شوند ولی در ورود به آن ایجاد نشده اند . (هر چیزی که محلی نباشد می شود غیرمحلی)

3. **Global RE**

تمامی شناسه هایی که در ابتدای برنامه اصلی تعریف می شوند به عنوان GRE محسوب می شوند.

4. **Predefiend RE**

تمامی شناسه هایی که از قبل توسط سیستم تعریف شده اند از این دسته اند مثل ثابت MAXINT در Pascal .

☑ Global همیشه زیرمجموعه ای از NoneLocal است .

○ **قابل دید بودن Visibility**

یک شناسه در داخل زیربرنامه قابل دید است اگر قسمتی از RE آن زیربرنامه را تشکیل دهد .

○ **غیر قابل دید بودن Hidden**

مفهومی کاملا برعکس قابل دید بودن می باشد .

○ **عملیات ارجاع Refrence Operation**

یک RO ، یک شناسه و یک RE را گرفته و آن شناسه را در محیط پیاده کرده و یک DO یا زیربرنامه را برمی گرداند .

Ref-Op : identifier*R.E → DO or Subprogram

ارجاع محلی ، غیرمحلی و سراسری

یک ارجاع به یک شناسه می تواند به صورت محلی ، غیرمحلی و سراسری انجام شود .
اگر عملیات ارجاع شناسه را در Local RE پیدا کند ، ارجاع محلی می شود و در غیر این صورت اگر عملیات ارجاع ، شناسه را در محیط غیرمحلی و یا سراسری پیدا کند ، یک ارجاع غیرمحلی و یا سراسری خواهد بود.

نام مستعار برای DO

طی LifeTime یک DO ، آن DO ممکن است پیش از یک نام داشته باشد که هرکدام از این نامها یک نام متغار برای آن DO هستند . برای مثال پارامترهای با ارجاع یک نوع Alias برای DO می باشند.

☑ نام مستعار فهم برنامه را مشکل می کند و خوانایی آن را پایین می آورد .

مثال :

```
Program main (output);
        Var A,B,C:Real
        Procedure SUB1(A:Real);
                Var D:Real;
                Procedure SUB2(C:Real);
                        Var D:Real;
                        Begin
                                -statements;
                                C:=C+B;
                                -statements;
                        end;
                Begin
                        -statements;
                        SUB2(B);
                        -statements;
                end;
```

RE For SUB2
Local : C,D
Nonlocal : A,SUB2 in SUB1,B,SUB1,OUTPUT in MAIN
Global : B,SUB1,OUTPUT in MAIN

RE For SUB1
Local : A,D,SUB2
Nonlocal = Global : B,C,SUB1,OUTPUT in MAIN

```
Begin                                   RE For MAIN
        -statements;                    Local : A,B,D,SUB1,OUTPUT
        SUB1(A);
        -statements;
End.                                    SUB2 برای Min قابل دید نیست در حالی که
                                        برای SUB1 قابل دید است .
```

مثال :

```
Program main(Output);
        Procedure SUB1(Var j:Real);
                Begin                   J is visible here. I is not
                                        J is alias for i
                end;
        Procedure SUB2;
                Var i:integer ;
                Begin                   I is visible here . j is not
                        SUB1(i);
                end;
        Begin
                                        Neither I nor j is visible here
                SUB2;
        End.
```

قواعد حوزه ایستا و پویا

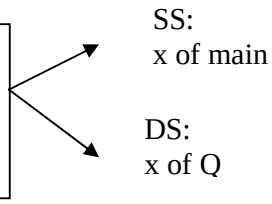
فرض کنید یک شناسه در داخل زیربرنامه ای وجود دارد که هیچ تعریفی برای آن در داخل آن زیربرنامه وجود ندارد . در این صورت این سئوال مطرح است که این شناسه از کجا آمده است (یا متعلق به کدام بخش از برنامه می باشد) برای پاسخگویی به این سئوال دو قاعده زیر معرفی شده است :

- **قاعده ایستا (SS):**

هر اعلان یا تعریفی از یک شناسه در داخل متن برنامه قلمرویی دارد که حوزه ایستا نامیده می شود. کامپایلر در زمان ترجمه برای برنامه اصلی و هر زیربرنامه یک جدول از تعاریف محلی را تشکیل می دهد و ارجاع را از آنها می گیرد . جهت Static TC باید از فلمرو ایستا استفاده کرد و متغیر و نوع آن را مورد دستیابی قرار داده و سپس عمل چک کردن را انجام دهیم .

به عبارت ساده تر هر شناسه در داخل یک زیربرنامه که به صورت محلی در آن اعلان نشده باشد آن شناسه از nonelocalRE و GoalRE آن در نظر گرفته می شود .

```
Program main ;
    Var x,y:integer ;
    Procedure R ;
            Var y:Real ;
            Begin
                    x:=x+1 ;
            end ;
    Procedure Q ;
            Var x:Real ;
            Begin
                    R ;
            end ;
    Procedure P
            Var x:Boolean ;
            Begin
                    Q ;
            end ;
    Begin
            P ;
    End .
```



♦ تعریف حوزه پویا (Dynamic Scope)

در زمان اجرا با استفاده از AR ها زنجیره ای از فراخوانی ها ساخته می شود . (main→P→Q→R) در هر یک از این AR ها اطلاعات مربوط به شناسه های محلی آن زیربرنامه وجود دارد . حال اگر شناسه ای در یک زیر برنامه وجود داشته باشد که در خود آن زیربرنامه تعریف شده باشد (یعنی در AR آن نباشد) برای مشخص کردن این که شناسه مربوط به کدام زیربرنامه دیگر است در زنجیره فراخوانی ها حرکت می کنیم . (زیربرنامه های فراخوانی کننده)تا به زیربرنامه برسیم که آن شناسه در آن تعریف شده است .

برای سازگاری این دو قاعده به این صورت عمل می شود که اول قاعده ایستا چک می شود و در صورت نا مشخص بودن به قاعده پویا مراجعه می شود .

☑ قوانین محدوده ایستا ، ارجاعات را به اعلان نامها در متن برنامه مرتبط می سازند و قوانین محدوده پویا ارجاعات را به وابستگی هایی برای نامها در ضمن اجرای برنامه مرتبط می سازند .

زبانهای برنامه سازی با ساختار بلاکی (Block Stracture)

در یک زبان برنامه سازی بر اساس ساختار بلاکی ، هر برنامه یا زیربرنامه به صورت مجموعه ای از بلاک های تودرتو سازماندهی می شوند مانند زبانهای C,Pascal,PL1,Ada

زبانهای بلاکی معمولا از قوانین حوزه ایستا استفاده می کنند .

- ☑ هد هر بلاک ، مجموعه شناسه های محلی آن بلاک را تشکیل می دهد .
- ☑ مجموعه شناسه های محلی هر بلاک از دید بلاکهای بیرونی آن مخفی است .
- ☑ در صورتی که یک شناسه مورد مراجعه قرار گیرد و تعریف برای آن در آن بلاک وجود نداشته باشد، تعریف آن شناسه از بلاک بیرونی تر گرفته می شود .
- ☑ بلاک می تواند ارای نام باشد (زیربرنامه)که در این صورت نام آن بلاک بخشی از محیط ارجاع محلی بلاک بیرونی تر از آن است .

انواع روشهای ارجاع محلی شناسه ها در داخل زیربرنامه

1. **حذفی (Deletion)**

در این روش در هنگام ورود به زیر برنامه ، متغیرهای محلی آن ایجاد و به هنگام بازگشت از آن متغیرهای آن از بین می روند . مانندزبانهای Pascal,Ada,Lisp,Apl,Snobol4

2. **نگهداری (Retention)**

در این روش در اولین ورود به زیربرنامه متغیر از بین نمی رود و در فراخوانی های بعدی آن زیربرنامه از آخرین مقدار مرحله قبل آن استفاده می گردد . مثل Cobol,Fortran

- ☑ زبانهای Algol, PL1,C,Java هر دو روش را ارائه کرده اند . مثلا در زبان C کلمه کلیدی Static روش نگهداری را و Automatic (پیش فرض) روش حذفی را ارائه می دهد .

پیاده سازی

برای پیاده سازی دو روش بالا به این صورت عمل می شود که در حالت نگهداری ، CS به همراه محیط ارجاع محلی (AR) در یک مکان ذخیره می شوند ، بنابراین چون کد همواره وجود دارد با ایجاد بازگشت از یک فراخوانی یک زیربرنامه محیط ارجاع محلی آن از بین نمی رود .
و برای روش حذفی دو قسمت کد و داده ها جدا از هم ذخیره می شوند (کد و AR) که بخش کد همواره وجود دارد ولی بخش AR به ازای هر فراخوانی ایجاد و به ازای بازگشت از آن از بین می رود .

- ☑ در زیربرنامه های بازگشتی روش deletion مناسب است و روش retention فضای زیادی را اشغال می کند . دلیل آن ، این است که در روش حذفی برای زیربرنامه های بازگشتی فقط AR تکرار می شود ولی در روش نگهداری هم AR و هم کد با هم تکرار می شوند .

اطلاعات مشترک

اگر چندین برنامه بخواهند داده های مشترکی داشته باشند ، لازم است که داده های مشترک در یک جدول نگهداری شود و در دسترس زیربرنامه ها قرار گیرد . این داده های مشترک در یک محیط غیرمحلی و اشتراکی به نام NoneLocal Environment نگهداری می شوند .

انواع معرفی های محیط اشتراکی

1. محیط اشتراکی صریح
2. محیط اشتراکی ضمنی

♦ محیط اشتراکی صریح

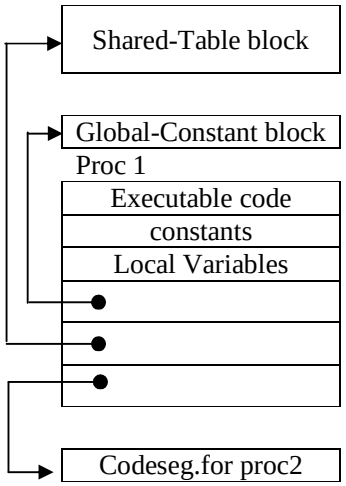
در این روش هر زیر برنامه باید یک اعلان صریح برای استفاده از داده های اشتراکی داشته باشند .

Fortran → common C++,Smalltalk → class PL1→ external C → extern
Ada → package

از نظر پیاده سازی در صورتی که یک زیربرنامه از ثابت های عمومی و داده های اشتراکی استفاده کند و زیربرنامه دیگری را فراخوانی نماید ، اتصال از زیربرنامه به بلاک اشتراکی و زیربرنامه های دیگر به صورت شکل زیر است :

مثالی از محیط اشتراکی صریح :
داده های اشتراکی صریح به صورت import/export در زبان Modula2

```
Procedure P(…);
        defines x,y,z;  ──►  Export
        x,y,z : real ;
                Begin
                □
                End;
Procedure Q(…);
        Uses p.x , p.z  ──►  Import
        y:integer;
                Begin
                □
                End;
```



در این روش از جهت پیاده سازی باید آدرس پایه (Base Address) مربوط به .codeseg زیربرنامه P به همراه offset داده های اشتراکی استفاده شوند .

قلمرو پویا

در صورتی که شناسه ای در زیربرنامه ای استفاده شده باشد ولی در تعاریف محلی آن موجود نباشد و هیچ اعلان صریحی مانند import/export برای آن وجود نداشته باشد ، آن شناسه از طریق dynamic chain در زیربرنامه های فراخوانی شده پیدا می شود و بر این اساس این قاعده از آخرین AR زیربرنامه فراخوانی شده به سمت عقب جستجو انجام می شود . نام دیگر این قاعده Most Recent Association Rule می باشد .

همانطور که در شکل قبل مشاهده می شود ، از نظر پیاده سازی می توان برای هر داده ای flag ای در نظر گرفت که در آن flag اطلاعاتی راجع به نادیده گرفتن شناسه ها قرار گیرد .

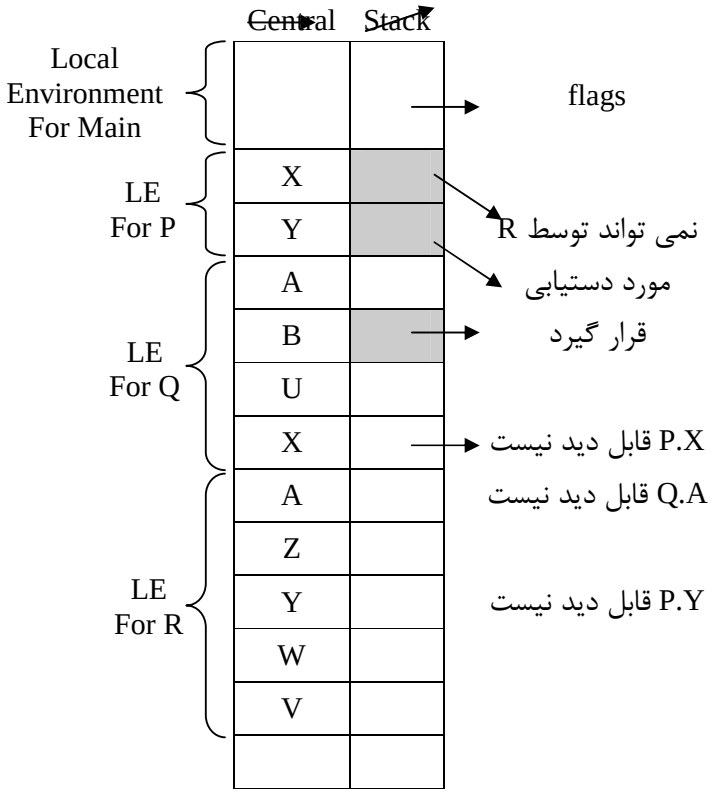
این مکانیزم در snobol4 استفاده می شود .

```
Main ;
        P;
        □
End.

Proc P ;
        X,Y;
        □
        Q;
end;

Proc Q ;
        A,B,U,X;
        □
        R;
end;

Proc R ;
        Z,Y,A,W,V;
        □
end;
```



قلمرو ایستا

در این روش در صورتی که به یک شناسه در داخل زیربرنامه برسیم که به صورت Local در داخل آن تعریف نشده باشد ، طبق قاعده ایستا با توجه به کد برنامه به سمت بلاکهای بیرونی تر می رویم و آن قدر این کار را ادامه می دهیم تا آن شناسه را پیدا نماییم .

- ☑ برای پیاده سازی قاعده پویا در داخل AR هر زیربرنامه یک اشاره گر "RP" به AR زیربرنامه فراخوانی کننده آن نگهداری می شود .
- ☑ برای پیاده سازی قاعده ایستا در داخل AR هر زیربرنامه ، یک اشاره گر به نام "SCP" به AR زیربرنامه بیرونی تر در CS آن باید نگهداری شود .

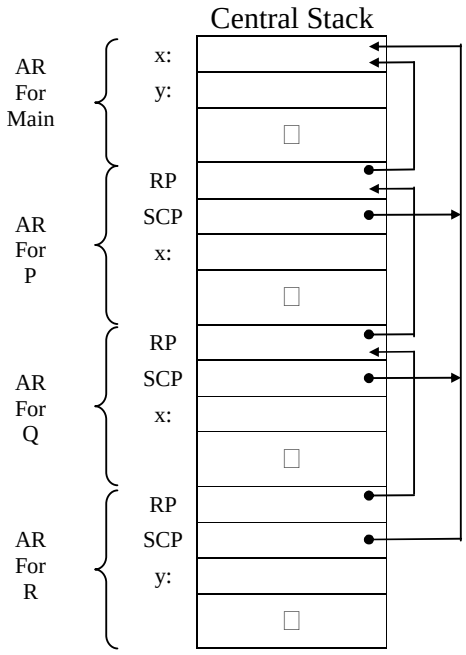
مثال

```
Program main ;
        Var x,y:integer ;
        Procedure R ;
                Var y:Real ;
                Begin
                        x:=x+1 ;
                end ;

        Procedure Q ;
                Var x:Real ;
                Begin
                        R ;
                end ;

        Procedure P
                Var x:Boolean ;
                Begin
                        Q ;
                end ;

        Begin
                P ;
        End .
```



- ☑ چون در زبان C امکان تعریف متداخل زیربرنامه ها وجود ندارد ، طول SC همواره برابر 1 است ولی در زبان پاسکال این چنین نیست .

پارامترها و انتقال آن

Argument : یک DO که به یک تابع یا پروسیجر فرستاده می شود .

Result : مقداری که توسط زیربرنامه فراخوانی شده ، برگشت داده می شود .

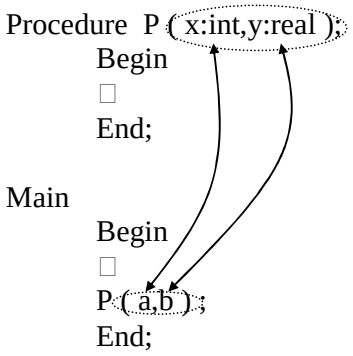
Formal Parameters : DO های تعریف شده در داخل زیربرنامه

Actual Parameters : DO های موجود در دستور فراخوانی ، زیربرنامه فراخوانی کننده .

روش های نسبت دادن پارامترهای Actual به Formal

1- تاظر موقعیت Position Correspondence

2- تناظر نام Correspondence By Explit Name



1: در این روش بر اساس قرارگیری پارامترهای اصلی و مجازی تناظر به صورت ترتیبی انجام می شود ، یعنی اولین پارامتر Actual در اولین پارامتر Formal ، دومی در دومی و ... اکثر زبانها از این روش استفاده می کنند : C,Pascal ...

2:در این روش تناظر بر اساس نام پارامترهای اصلی و مجازی انجام می شود یعنی در هنگام فراخوانی ، مشخص نماییم که کدام پارامتر Actual داخل کدام پارامتر Formal وارد شود .

مکانیزم های ارسال پارامترها به زیربرنامه ها :

1. **مکانیزم Pass By Value :**

در این روش در زمان فراخوانی یک زیربرنامه ، مقدار پارامترهای واقعی (Actual) در پارامتر مجازی کپی می شود و بنابراین هر تغییری در داخل زیربرنامه که روی پارامترهای مجازی انجام شود به پارامترهای اصلی برگردانده نمی شود (تاثیر ندارد) این روش تنها برای حالت input استفاده می شود . Ada95

2. **مکانیزم Pass By Result :**

در این روش در زمان فراخوانی زیربرنامه ، مقدار پارامترهای واقعی در پارامترهای مجازی کپی نمی شود و در طول اجرای زیربرنامه نیز تغییرات پارامترهای مجازی بر روی پارامترهای واقعی تاثیر نمی گذارد ، اما در موقع بازگشت از زیربرنامه ، مقدار پارامترهای فرمال در واقعی کپی خواهد شد . این روش برای متغیرهایی استفاده می شود که حالت output دارند . پارامترهای out در Ada

3. مکانیزم Pass By Value-Result

در این روش در هنگام فراخوانی زیربرنامه ، مقدار پارامترهای واقعی در پارامترهای مجازی کپی می شود . در طول اجرای زیربرنامه ، تغییرات پارامترهای ، تاثیری روی پارامترهای واقعی نمی گذارد اما در هنگام بازگشت از زیربرنامه ، مقدار پارامتهای مجازی در واقعی کپی خواهند شد . Algol w برای حالت input/output

4. مکانیزم Pass By Reference

در این روش (که متداول ترین روش ارسال پارامترها می باشد و در آن بحث Alias مطرح است)در زمان فراخوانی زیربرنامه یک اشاره گر از پارامترهای واقعی در پارامترهای مجازی ایجاد می شود و هر تغییری در داخل زیربرنامه بر روی پارامترهای مجازی مستقیما در پارامترهای واقعی تاثیر می گذارد . Fortran

5. مکانیزم Pass By Name

در این روش پارامترهای واقعی تا زمان استفاده در زیربرنامه فراخوانی شونده ارزشیابی نخواهند شد و زیربرنامه فراخوانی شونده ، زمان ارزشیابی آن را مشخص می کند . از دید برنامه نویس این مکانیزم معادل مکانیزم Pass By Reference می باشد . Algol

این روش چون هزینه و سربار زیادی (در زمان اجرا) دارد بنابراین به صورت روش مشهور مطرح نشده است.

- ☑ زبان C از روش ارسال با مقدار استفاده می کند ولی استفاده از اشاره گرها به برنامه نویس امکان ایجاد ارسال با ارجاع را نیز می دهد ولی در زبان Pascal هر دو روش با ارجاع و با مقدار وجود دارد .
- ☑ در زبان Ada روش متفاوتی برای ارسال وجود دارد :

In (value)
Out (result)
In out (value-result)

مثال :

```
Main
        M,N:integer ;
        M:=10;
        N:=20;
        Proc(M,N);
        
End.

Proc (x:integer,y:iteger)
        Begin
        X:=7;
        Y:=X+Y;
End;
```

: Pass by Value

قبل از Proc :	N=20	M=10
درداخل Proc :	Y=27	X=7
بعد از Proc :	N=20	M=10

: Pass by Result

قبل از Proc : N=20 M=10

درداخل Proc : Y=7 X=7

بعد از Proc : N=7 M=7

: Pass by Value-Result

قبل از Proc : N=20 M=10

درداخل Proc : Y=27 X=7

بعد از Proc : N=27 M=7

: Pass by Reference

قبل از Proc : N=20 M=10

درداخل Proc : Y=27 X=7

بعد از Proc : N=27 M=7

: Pass by Name

قبل از Proc : N=20 M=10

درداخل Proc : Y=27 X=7

بعد از Proc : N=27 M=7

+ تفاوت 3 و 4 با در نظر گرفتن M,N به صورت Global و تغییر کد زیر برنامه Proc به صورت زیر :

```
Proc (x:integer,y:iteger)
        Begin
        X:=7;
        Y:=M+X;
End;
```

: Pass by Value-Result

قبل از Proc : N=20 M=10

درداخل Proc : Y=17 X=7

بعد از Proc : N=17 M=7

: Pass by Reference

قبل از Proc : N=20 M=10

درداخل Proc : Y=14 X=7

بعد از Proc : N=14 M=7

+ مقایسه 4 و 5

پیاده سازی انتقال پارامترها :

برای پیاده سازی 5 روش ارسال پارامتر به توابع از دو مکانیزم زیر استفاده می شود .

1- پارامترهای مجازی را به عنوان DO ها (متغیرهای) محلی از نوع همان پارامترهای وتقعی در نظر بگیریم در صورتی که ارزش اولیه پارامترهای مجازی ، یک کپی از ارزش پارامترهای واقعی باشد .
Value,Result,Value-Result

2- پارامترهای مجازی به عنوان یک DO مجلی از نوع اشاره گری به پارامترهای واقعی متناظر با آنها باشد ، در صورتی که ارزش اولیه این اشاره گرها آدرس های پارامترهای واقعی باشند .
Name,Reference

زیربرنامه به عنوان پارامتر

مثال زیر را در نظر بگیرید :

Procedure Q(x:integer,function R(y,z:integer) :integer);

برای فراخوانی Q می توان به صورت ;Q(27,FN) عمل کرد که در آن FN یک Function است و در داخل Q دستور ;z:=R(I,x) برای فراخوانی R استفاده می شود .

برای ارسال یک زیربرنامه به عنوان پارامتر دو مشکل زیر وجود دارد :

1-Static Type Chacking

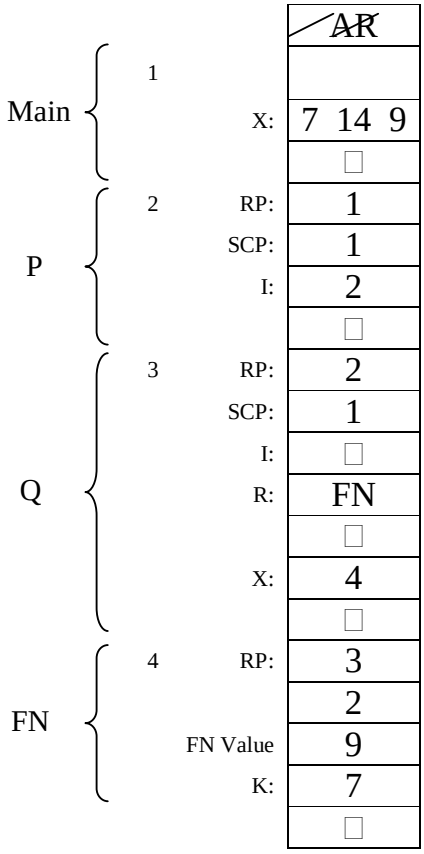
باید آرگومانهای زیربرنامه ای که به عنوان پارامتر فرستاده می شوند ، از نظر تعداد ، نوع و ترتیب با زیربرنامه اصلی تعریف شده مطابقت داشته باشند .

2- متغیرهای آزاد (Free Variable)

در صورتی که به هنگام مراجعه غیرمحلی (NoneLocal) هیچ مکانیزمی جهت Binding در تعریف زیربرنامه وجود نداشته باشد ، آن متغیر ، متغیر آزاد نامیده می شود . به عبارت دیگر ارجاعات غیرمحلی اعلان نشده داخل تابع را متغیر آزاد می نامیم .

```
Program  Main;
        Var x:integer;
        Procedure Q(var i:integer ; function R (j:integer):integer;
                Var x:integer;
                Begin
                        X:=4;
                        Write('in Q before call R.i=', I ,'x=',x);          (a) i=7 , x=4
                        I:=R(I);
                        Write('in Q after call R.i=', I , 'x=',x);          (c) i=9 , x=4
                end ;
        Procedure P;
                Var I:integer;
                Function FN(k:integer):integer;
                        Begin
                                X:=x+k;
                                FN:=I+k;
                                Write ('in FN.I=', I , 'k=',k , 'x=' , x);  (b) i=2 , k=7 , x=14
                        end;
```

```
        begin
                I:=2;
                Q(x,FN);
                Write('in P.i=', I, 'x=',x);                    (d) i=2 , x=9
        end;
begin
        x:=7;
        P;
        Writeln('in main.x=' , x);                              (e) x=9
End.
```



در مثال فوق دو متغیر x و i در داخل تابع FN به عنوان متغیر آزاد می باشند . که در اینجا این سئوال مطرح است که هنگام فراخوانی FN از طریق نام R در داخل زیر برنامه Q ، این متغیرها از کجای آزاد از کجا گرفته می شوند ؟ (در حالت ایستا x,i به ترتیب از main , P گرفته می شوند و در حالت پویا x,i به ترتیب از Q و متغیر با ارجاع Q گرفته می شود) که این مشکل به این صورت رفع می شود که ابتدا قاعده ی ایستا چک می شود و در صورت وجود نداشتن آنها به قاعده پویا مراجعه می شود .
طبق این قاعده ، همان حالت ایستا درست می باشد .