حوزه ها (ایستا ، پویا)

زیر برنامه

زیر برنامه‌ها به دو دسته رویه و تابع تقسیم می‌شوند.

تابع فقط یک مقدار را بر می‌گرداند، ولی رویه می‌تواند چند مقدار را برگردانند.

در C و C++ ، زیر برنامه‌ها فقط به صورت تابع‌اند، ولی رفتار رویه‌ها را نیز از خود نشان می‌دهند.

متدها در c++ ، c# و Java از نظر نحوی شبیه توابع در c هستند.

محیط ارجاع

Refrencing Enviroment

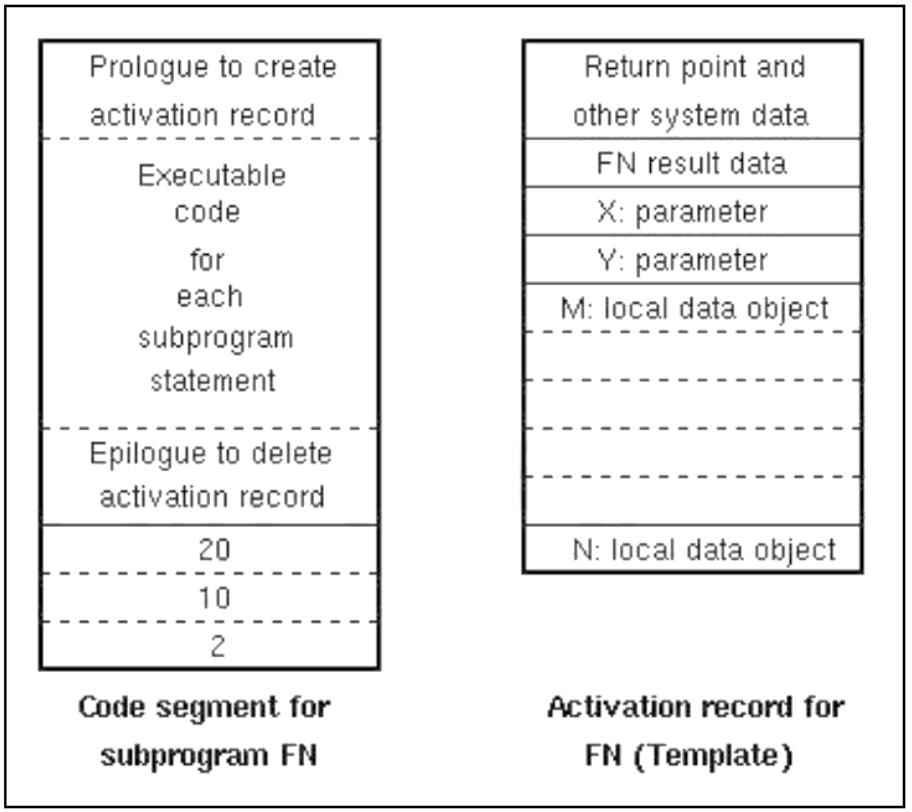
هر برنامه یا زیر برنامه، مجموعه‌ای از وابستگی‌های شناسه دارد که در حین اجرا به آن‌ها مراجعه می‌شود. این مجموعه از وابستگی‌ها را محیط ارجـاع زیـر برنامه یا برنامه می‌گویند.

محیط ارجاع زیر برنامه معمولاً در حین اجرا تغییر نمی‌کند.

محیط ارجاع در حین ایجاد رکورد فعالیت **زیر برنامه ایجاد و تنظیم می گردد و با از بین رفتن رکورد فعالیت از بین می رود.**

رکورد فعالیت

```
float  FN( float  X ,  int  Y )
  const  initval = 2;
  #define  finalval  10
  float  M(10);
  int  N;
  N = initval;
  if ( N < finalval )  { ... }
   return ( 20 * X + M(N) );
}
```



بخش های محیط ارجاع زیر برنامه

۱- محلی (local)

۲- غیر محلی

۳- عمومی(global)

۴- از قبل تعریف شده

کلمات کلیدی یا رزروی، می‌توانند محیط ارجاع از قبل تعریف شده را مشخص کنند.

اگر یک وابستگی برای یک زیربرنامه بخشی از محیط ارجاع آن زیر برنامه باشد گوییم آن وابستگی در زیربرنامه قابل رویت است.

قوانین حوزه زبان

محیط ارجاع: مجموعه‌ای از وابستگی‌های شناسه است که توسط قوانین حوزه (scope rules) تعیین می‌شود.

قوانین حوزه یک زبان مشخص می‌کند که وقوع هر شناسه‌ای در برنامه، به کدام اعلان بایند شود.

قوانین حوزه زبان‌ها به دو دسته تقسیم می‌شوند:

۱- حوزه پویا (dynamic scope)

۲- حوزه ایستا (static scope)

در زبانی که از قانون حوزه پویا استفاده می‌کند، وابستگی اسامی به طور پویا در زمان اجرا تعیین می‌شود.

ولی در قانون حوزه ایستا، وابستگی شناسه‌ها (بایند اسامی به اعلان‌ها) در زمان ترجمه قابل تعیین است.

حوزه پویا و ایستا

قاعده حوزه ایستا:

در صورت وجود نداشتن یک اسم به صورت محلی در یک بلاک، به بلاکهای بیرونی تر آن مراجعه می شود.

قاعده حوزه پویا:

در صورت وجود نداشتن یک اسم به صورت محلی در یک بلاک، به سراغ بلاکی می رویم که زیر برنامه را فراخوانی کرده است (قاعده تازه ترین وابستگی)

مثال

تعیین خروجی برنامه در زبانی با قانون حوزه پویا و ایستا :

```
procedure   main
   x : integer;
    procedure   sub1
    begin
          write(x);
    end;
     procedure  sub2
        x : integer;
     begin
        x :=1;
        sub1;
     end;
begin
   x:=2;
   sub2;
end;
```

حوزه پویا : 1

مراجعه به x در دستور write ، به اعلان آن در sub2 مقید می‌شود. (زیر برنامه فراخوان)

حوزه ایستا: 2

مراجعه به x در دستور write ، به اعلان x در main مقید می‌شود. (روال در برگیرنده)

مثال

```
Procedure A( )
      Var    x: integer;
      Procedure    B ( )
      Begin
            X: =x+1;
            Write(x);
      End;
      Procedure C ( )
            Var x: integer;
      Begin
            X: =30;
            B ( );
      End;
Begin
      X:=7;
      B ( );
      C ( );
End;
```

تعیین خروجی برنامه در زبانی با قانون حوزه پویا و ایستا :

حوزه پویا :

8

31

حوزه ایستا:

8

9

مشکلات حوزه پویا

قانون حوزه پویا دارای مشکلاتی است از جمله:

۱- در حین اجرای زیر برنامه، شناسه‌های محلی زیر برنامه‌ها برای تمام زیر برنامه‌های در حال اجرا قابل رویت هستند.

۲- قوانین حوزه پویا از میزان قابلیت خوانایی برنامه می‌کاهد.

۳- دستیابی به شناسه‌های غیر محلی، طولانی‌تر است.

۴- کنترل نوع ایستا در ارجاع به شناسه‌های غیر محلی صورت نمی‌گیرد. بنابراین کارایی برنامه پایین است.

قانون حوزه ایستا، در بسیاری از موارد کارآمد است، ولی خالی از عیب نیست.

قانون حوزه ایستا علاوه بر بالا بردن قابلیت اعتماد برنامه، بر قابلیت خوانایی برنامه نیز می‌افزاید.
خواننده برنامه، بدون دنبال کردن اجرای برنامه می‌تواند مراجعه به شناسه‌ای را به اعلان های آن مقید کند.
بنابراین، قانون حوزه ایستا، درک برنامه را آسان می‌سازد.

اگر زیر برنامه‌ها بتوانند تودرتو باشند، در حوزه ایستا، اگر به شناسه‌ای در یک زیر برنامه، مراجعه شود ولی اعلانی برای آن پیدا نشود، در زیر برنامه در برگیرنده آن جست و جو انجام می شود. (این روند در صورت نبودن اعلان، ادامه می‌یابد.)

در زبان‌های مبتنی بر C ، زیر برنامه‌ها نمی‌توانند تو در تو باشند.

مثال

در c++ می‌توان با عملگر : : به متغیر عمومی دست یافت.

```
int   x=3;
void  main( ){
  int  x=1;
  x =::x - x;
 cout  << x;
}
```

در این برنامه برای تشخیص متغیر x سراسری از x محلی، قبل از x سراسری از عملگر :: استفاده شده است.

خروجی : 2

متغیرهای محلی

متغیرهایی که در داخل زیر برنامه‌ها تعریف می‌شوند، متغیرهای محلی نام دارند.

انواع متغیرهای محلی :

۱- محلی ایستا (در C , C++ شناسه‌هایی که با واژه static تعریف می‌شوند.)

۲- محلی پویای پشته‌ای (وقتی به حافظه بایند می‌شوند که اجرای زیر برنامه شروع شود و وقتی بایند آن‌ها به محل حافظه از بین می‌رود که زیر برنامه خاتمه یابد.)

زیر برنامه‌های Ada ، متدهای C++ , C# , Java فقط متغیرهای محلی پویای پشته‌ای دارند.

کارایی متغیرهای محلی ایستا بیشتر است و دستیابی به آن‌ها سریع و مستقیم است.

عیب متغیرهای محلی ایستا :

۱- عدم پشتیبانی از بازگشتی

۲- قابل استفاده نبودن حافظه آن‌ها، توسط شناسه‌های محلی زیر برنامه‌های غیر فعال.

مثال

```
f( ){
   static   int  x=1;
   cout << x;
   x++;
}
main( ){
  f( ) ;
  f( ) ;
}
```

خروجی: 12

مثال

به برنامه زیر که به زبان Ada نوشته شده، توجه کنید:

```
procedure   p1   is
      . . .
end;
procedure    p2    is
     a : integer = 1;
begin
         write (a);
          p1;
          a=a+2;
          write (a);
end;
procedure    p3    is
      . . .
      p2;
end;
```

وقتی p3 در حال اجرا است، a قابل رویت نیست.
وقتی p3 زیربرنامه p2 را فراخوانی می‌کند، a با مقدار 1 در نظر گرفته شده و توسط write(a) ، مقدار 1 چاپ می‌شود.
وقتی p2 ، زیر برنامه p1 را فراخوانی می‌کند، وابستگی a مخفی می‌شود ولی در اثنای اجرای p1 نگهداری می‌شود.
وقتی p1 کنترل را به p2 برمی‌گرداند، وابستگی a دوباره قابل رویت می‌شود.
P2 اجرای خود را از سر گرفته و دو واحد به a اضافه می‌کند و سپس مقدار 3 چاپ می‌شود.
وقتی p2 کنترل را به p3 برمی‌گرداند، وابستگی a دوباره مخفی می‌شود.

برای این وابستگی دو معنای مختلف ممکن است در نظر گرفته شود:

۱- **نگهداری** : وابستگی a ممکن است نگهداری شود تا p2 دوباره فراخوانی شود.
زبان‌هایی مثل C با واژه static شناسه‌هایی را اعلان می‌کنند که نگهداری می‌شوند.

۲- **حذف**: وابستگی a ممکن است از بین برود.

روش نگهداری به برنامه نویس اجازه می دهد تا زیربرنامه هایی بنویسد که نسبت به گذشته حساس باشد، به طوری که بخشی از نتایج آن ها در هر فراخوانی توسط ورودی و بخش دیگر توسط داده های محلی تعیین می شود که در حین سابقه فعالیت قبلی محاسبه شده اند. مثل متغیرهای static در C .

در روش نگهداری، جدول های محیط محلی برای تمام زیربرنامه ها در حین اجرا وجود دارد.

در روش حذف ، داده های محلی نمی توانند از یک فراخوانی به فراخوانی دیگر منتقل شوند. بنابراین متغیری که باید بین فراخوانی های مختلف نگهداری شود، باید برای آن زیربرنامه به صورت غیر محلی اعلان گردد.

در زیربرنامه های بازگشتی روش حذف متداول تر است.

روش حذف موجب صرفه جویی در حافظه می شود، به طوری که جدول های محلی فقط برای زیربرنامه هایی وجود دارند که در حال اجرا هستند یا اجرای آنها به تعویق افتاده است.

پیاده سازی محیط ارجاع محلی

برای پیاده سازی محیط ارجاع محلی، آن را به صورت جدول محیط محلی نشان می دهیم.

```
Procedure sub (x:integer) is
      y:real;
      z:array (1..3) of real;
      procedure sub2 is
      begin
            ...
      end {sub2};
begin
      ...
end {sub};
```

جدول محیط محلی sub :

نام	نوع	محتویات Lvalue
x	Integer	پارامتر با مقدار
y	Real	متغیر محلی
z	Real	آرایه
		توصیفگر : [1..3]
Sub2	Procedure	اشاره گر به سگمنت کد

نگهداری :

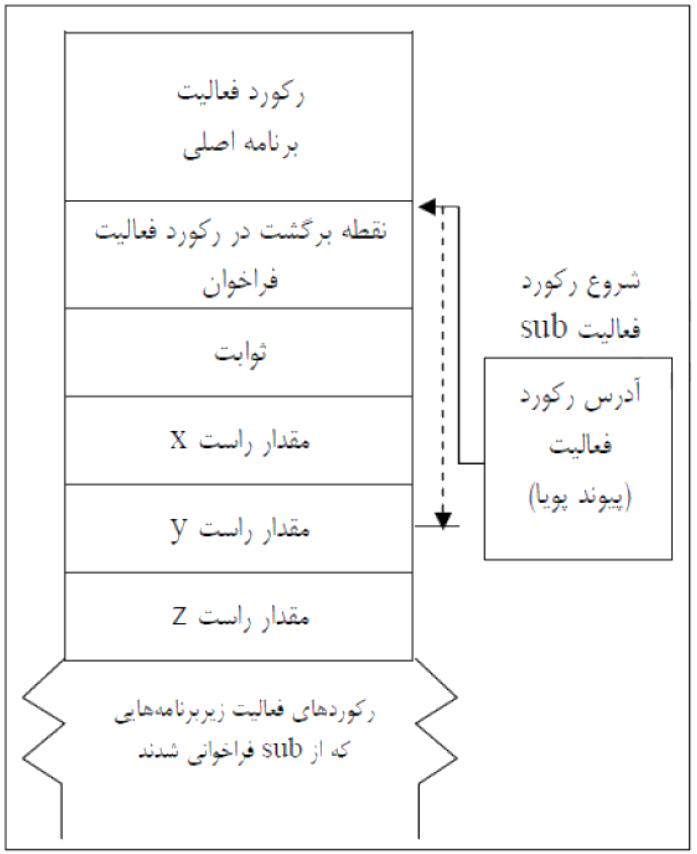
اگر قرار باشد محیط محلی زیربرنامه sub ، بین فراخوانی های مختلف باقی بماند، جدول حاوی متغیرهایی که باید نگهداری شوند، به عنوان بخشی از سگمنت کد تخصیص می یابد.

حذف:

اگر قرار باشد محیط محلی زیر برنامه، بین فراخوانی های مختلف از بین برود و دوباره ایجاد شود، آن گاه جدول محیط محلی حاوی متغیرهایی که باید حذف شوند، به عنوان بخشی از رکورد فعالیت زیر برنامه تخصیص می یابند.

چون با ورود به زیربرنامه، رکورد فعالیت آن در پشته قرار می گیرد و با خروج از آن ، رکورد فعالیت حذف می شود ، حذف محیط محلی به طور خودکار صورت می گیرد.

تخصیص و ارجاع به متغیرهای قابل حذف شدن



با فرض اینکه تمام متغیرهای محلی در ابتدای زیربرنامه اعلان شده اند، کامپایلر می تواند تعداد متغیرها و اندازه هر کدام را در جدول محیط محلی محاسبه کند و سپس آفست شروع هر شیء داده را از ابتدای رکورد فعالیت محاسبه کند.

اشاره گری به نام پیوند پویا در حین اجرا به ابتدای رکورد فعالیت زیربرنامه در پشته اشاره می کند.

اگر زیر برنامه در حین اجرا به متغیر y مراجعه کند، محل شیء داده وابسته به آن با افزودن اشاره گر پیوند پویا به آفست y پیدا می شود.

شکل مقابل تخصیص و ارجاع به متغیرهای محلی قابل حذف شدن را نشان می دهد.

پارامترهای مجازی و واقعی

زیر برنامه‌ها (غیر از متدهای کلاس) ، به دو طریق می‌توانند به داده‌ها دسترسی داشته باشند:

۱ - پارامترها

۲- متغیرهای غیر محلی

روش دسترسی به کمک پارامترها، قابلیت انعطاف بیشتری دارد.

دستیابی به متغیرهای غیر محلی از میزان قابلیت خوانایی و قابلیت اعتماد برنامه می‌کاهد.

پارامترها بر دو نوع می‌باشند:

۱- **مجازی** (formal) :

پارامترهایی که در سرآیند زیر برنامه ظاهر می‌شوند. (a)

۲- **واقعی**:

پارامترهایی که در دستور فراخوانی به کار می‌رود. (x)

```
int  f (int a ) {
    ....
}
main( ) {
   int  x=1 ;
   ...
   f(x);
}
```

تناظر بین پارامترها

وقتی زیر برنامه با لیستی از پارامترهای واقعی فراخوانی می‌شود، بین پارامترهای مجازی و واقعی دو نوع تناظر باید وجود داشته باشد:

۱- تناظر موقعیتی

۲- تناظر بر اساس نام (پارامترها به هر ترتیبی می‌توانند در لیست پارامترها ظاهر شوند.)

مثال

به فراخوانی رویه f در Ada توجه کنید:

```
f (a ,b){
   . . .
}
```

```
f (a => x , b => y);
```

این دستور فراخوانی مشخص می‌کند که زیر برنامه f دارای پارامترهای مجازی a و b است که به ترتیب با پارامترهای واقعی x و y متناظر شده‌اند.

مثال:

```
drawLetter( C => 'A';  Font  => TimesNewRoman ,  Size => 12 )
```

عیب روش تناظر بر اساس نام این است که کاربر باید اسامی پارامترهای مجازی را بداند.

در زبان Ada ترکیب دو روش نیز ممکن است.

در زبان‌هایی که پارامترهای مجازی نمی‌توانند مقدار اولیه بگیرند، در هنگام فراخوانی، تعداد پارامترهای واقعی باید با تعداد پارامترهای مجازی برابر باشند.

در C# متدها می‌توانند تعداد متغیری از پارامترها را دریافت کنند (نوع آن‌ها باید یکسان باشد).

در C# پارامترهای مجازی متدها با واژه params مشخص می‌شوند.