پیاده سازی زیربرنامه ها

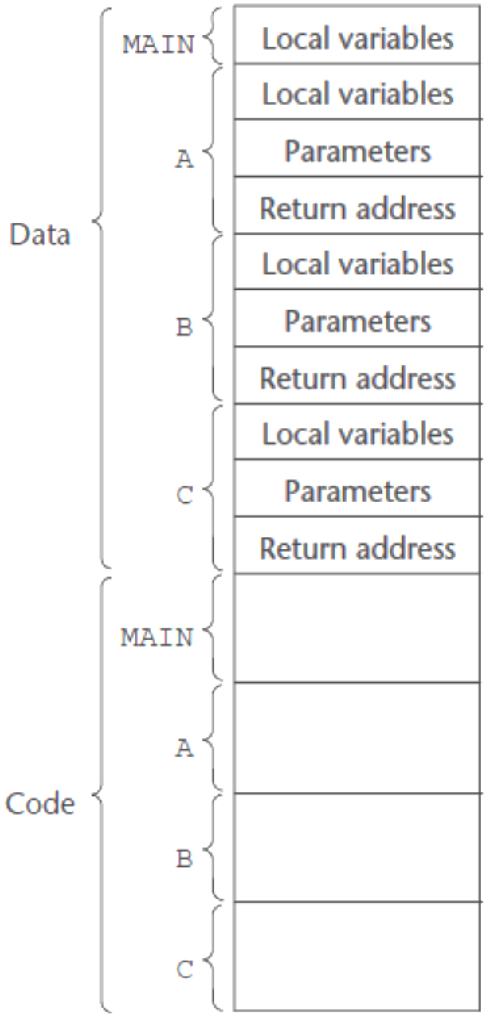
پیاده سازی زیر برنامه های ساده

منظور از زیر برنامه ساده، تودرتو نبودن آن و ایستا بودن متغیرهای محلی آن است.
زیر برنامه‌های ساده از بازگشتی پشتیبانی نمی‌کنند.
در زیر برنامه‌های ساده، اندازه کد زیر برنامه و رکورد فعالیت ثابت است.

رکورد فعالیت زیر برنامه‌های ساده به صورت زیر است:

متغیرهای محلی
پارامترها
آدرس برگشت

هر زمان فقط یک سابقه فعالیت از زیر برنامه ساده وجود دارد، در نتیجه فقط یک رکورد فعالیت از زیر برنامه موجود است.



رکورد فعالیت و کد زیربرنامه های ساده

برنامه ای شامل برنامه اصلی main و سه زیر برنامه A,B,C.
رکورد فعالیت و کد برنامه ها در محل های جداگانه ای قرار دارند.

با فراخوانی پیوند دهنده (Linker) برای MAIN ، کد ماشین و رکورد فعالیت برنامه های A,B,C به همراه کد MAIN در حافظه قرار گرفته و سپس آدرس مقصد فراخوانی های A,B,C و زیربرنامه های کتابخانه ای تنظیم می شود.
به پیوند دهنده بارکننده نیز می گویند.

وقتی پیوند دهنده برای برنامه اصلی فراخوانی می‌شود، اولین وظیفه‌اش یافتن فایلی است که حاوی زیر برنامه‌هایی است که در برنامه به آن‌ها مراجعه شده است.
آن‌ها را به همراه رکورد فعالیت به حافظه بار می‌کند.
سپس پیوند دهنده باید آدرس مقصد تمام فراخوانی‌های آن زیر برنامه را در برنامه اصلی، به آدرس ورود به آن زیر برنامه‌ها را تنظیم کند.

پیاده سازی زیر برنامه شامل متغیر پویای پشته‌ای

زبان‌هایی که از متغیرهای پویای پشته‌ای استفاده می‌کنند، از بازگشتی پشتیبانی می کنند.
بازگشتی موجب می‌شود همزمان چند سابقه فعالیت از زیر برنامه وجود داشته باشد.
به ازای هر سابقه فعالیت زیر برنامه، به یک رکورد فعالیت نیاز است.
هر رکورد فعالیت، کپی خاصی از پارامترهای مجازی، متغیرهای پویای محلی و آدرس برگشت را دارد.

نمونه‌ای از رکورد فعالیت برای زبان‌هایی با متغیرهای محلی پویای پشته‌ای :

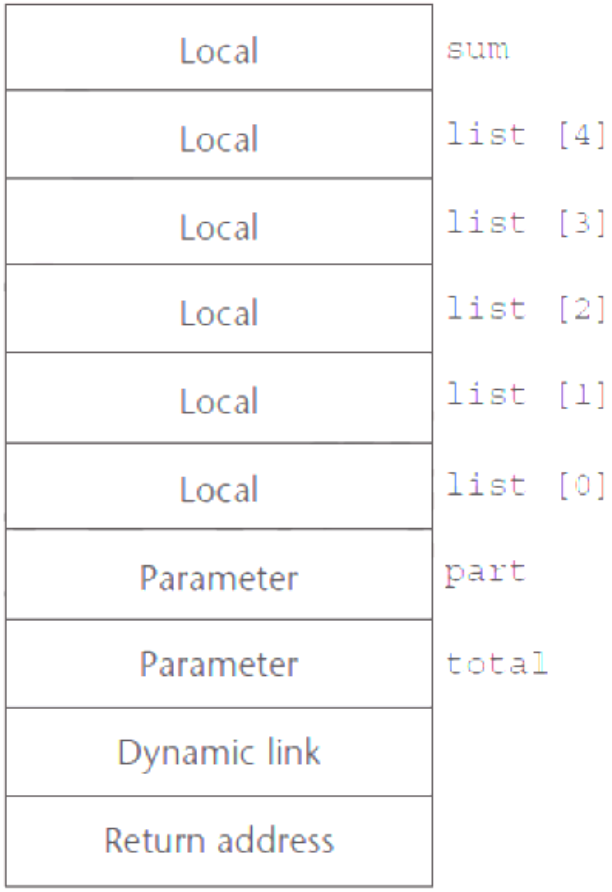
متغیرهای محلی
پارامترها
پیوند پویا
آدرس برگشت

آدرس برگشت: شامل اشاره‌گری به سگمنت کد برنامه فراخوان
پیوند پویا: اشاره‌گر به بالای رکورد فعالیت فراخوان.
پارامترها: مقادیر یا آدرس‌هایی که توسط فراخوان آماده می‌شود.
چون متغیرهای محلی در داخل زیر برنامه تخصیص می‌یابد، در بالای پشته قرار می‌گیرند.

قالب رکورد فعالیت در زمان ترجمه ثابت است، ولی اندازه آن در بعضی از زبان‌ها ممکن است تغییر کند.

رکورد فعالیت تابع sub

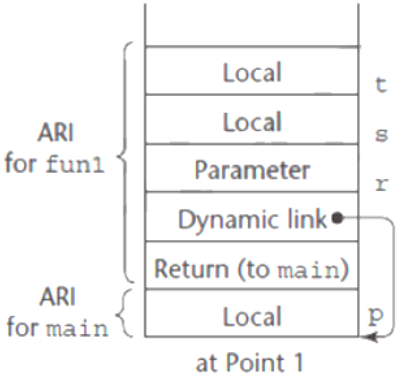
```
void   sub ( float   total , int  part) {
   int   list[5];
    float  sum;
    …
}
```

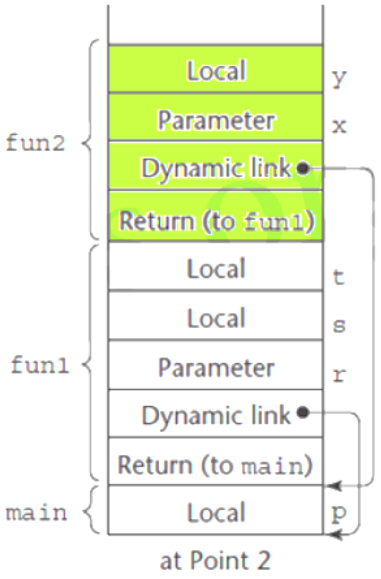


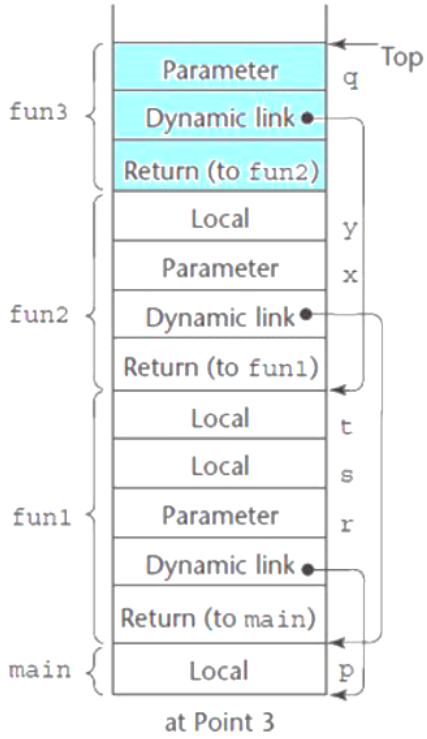
```
void  fun2(int  x) {
    int   y;
    ...          ← 2
    fun3(y);
    ...
}
void  fun1(float  r) {
    int   s,t;
    ...          ← 1
    fun2(s);
    ...
}
void  fun3(int  q) {
    ...          ← 3
}
void  main( ) {
    float  p;
    ...
    fun1 (p);
    ...
}
```

مثال

ترتیب فراخوانی‌ها : main → fun1 → fun2 → fun3







پیاده سازی زیر برنامه‌های بازگشتی

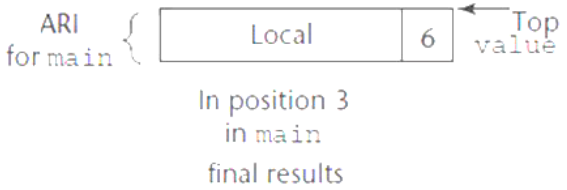
برای درک پیاده سازی، تابع فاکتوریل در c رادر نظر بگیرید:

قالب رکورد فعالیت برای این تابع :

مقدار تابع
پارامترها
پیوند پویا
آدرس برگشت

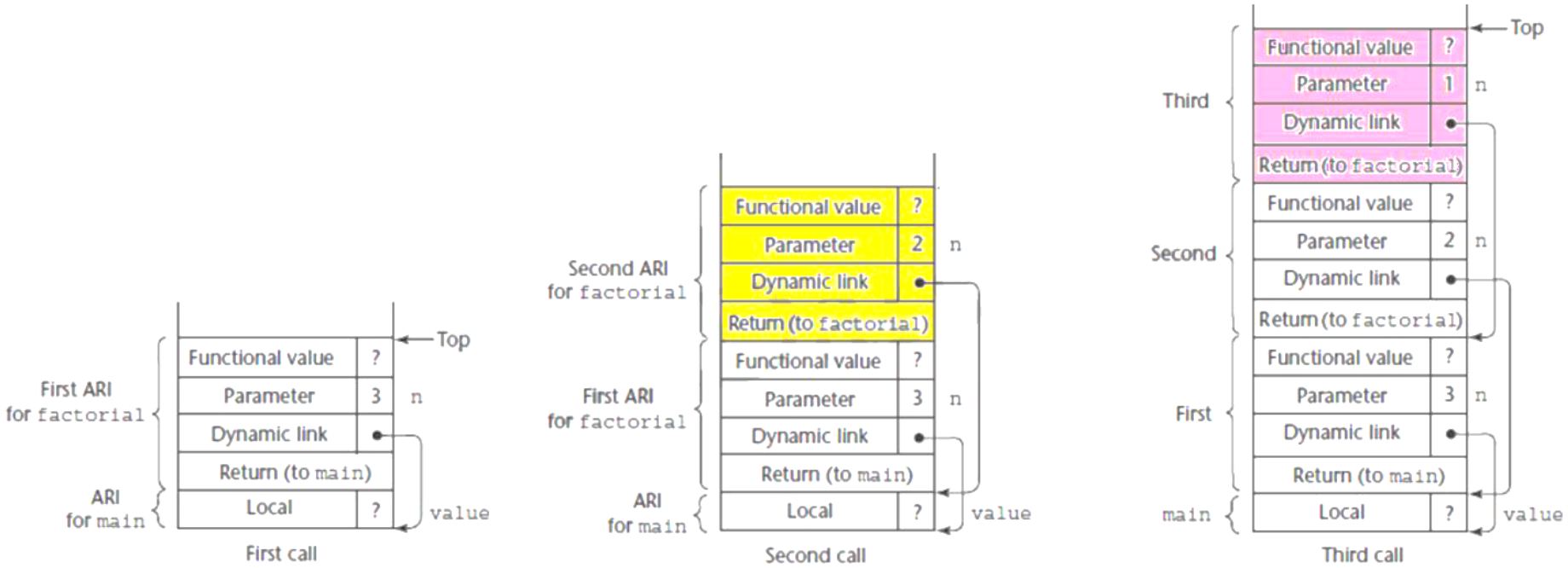
```
int    factorial (int   n) {
    ← 1
     if ( n <= 1 )
         return 1;
     else
         return ( n * factorial (n-1)  );
      ←2
}

void   main ( ) {
    int   a;
    a = factorial (3);
    ← 3
}
```



محتویات پشته در موقعیت ۱ تابع factorial

شکل زیر ، محتویات پشته را برای سه بار که کنترل اجرا به نقطه 1 می‌رسد، نشان می‌دهد.
این موقعیت زمان اجرای دستور return و قبل از حذف شدن رکورد فعالیت از پشته را نشان می‌دهد.

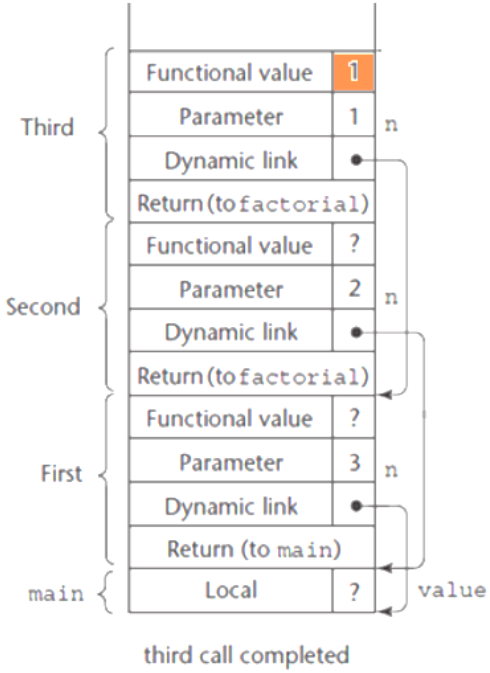


محتویات پشته در موقعیت ۲ تابع factorial

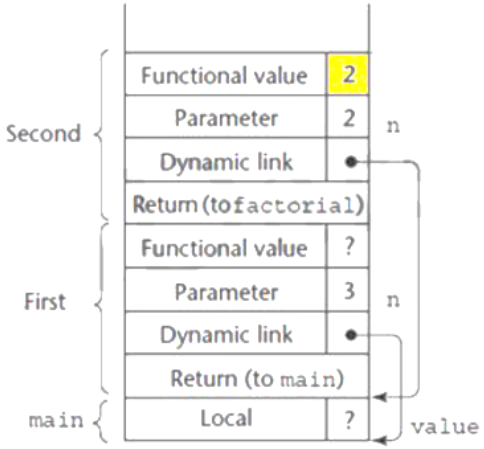
اولین برگشت از تابع فاکتوریل، مقدار 1 را برمی‌گرداند

نتیجه ضرب، یعنی 1 به سابقه فعالیت دوم فاکتوریل برگردانده می‌شود، تا در پارامتر n ضرب شود که برابر با 2 است.

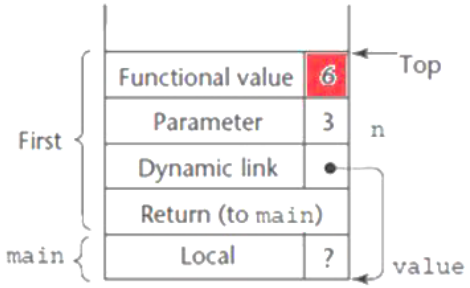
مقدار 2 به اولین سابقه فعالیت فاکتوریل برگردانده شده تا در 3 ضرب شود و نتیجه 6 به اولین فراخوانی در main برمی‌گردد.



third call completed



second call completed



first call completed

پیاده سازی زیر برنامه‌های تودرتو

در زبان‌هایی که از متغیرهای محلی پویای پشته‌ای استفاده می‌کنند، زیر برنامه‌ها می‌توانند تودرتو باشند.

مراحل دستیابی به متغیرهای غیر محلی در زبان‌هایی با حوزه ایستا که دارای زیر برنامه‌های تودرتو هستند، عبارتند از:

۱- یافتن رکورد فعالیتی در پشته که این متغیرها در آن‌ها وجود دارند.

۲- استفاده از آفست محلی متغیر در داخل رکورد فعالیت، برای دستیابی به متغیر .

روش‌های پیاده سازی حوزه ایستا در زبان‌هایی که از زیر برنامه‌های تودرتو پشتیبانی می‌کنند:

الف- زنجیر ایستا (static chain)

ب- نمایشگر (display)

زنجیر ایستا (static chain)

زنجیری از پیوندهای ایستا که چند رکورد فعالیت را در پشته به هم وصل می‌کند.

در هنگام اجرای یک رویه، پیوند ایستای رکورد فعالیت آن، به رکورد فعالیت رویه در برگیرنده آن اشاره می‌کند.

عمق ایستایی (static depth) : یک مقدار صحیح که عمق تودرتویی را نسبت به خارجی‌ترین حوزه مشخص می‌کند.

```
procedure   A   is
    procedure   B   is
        . . .
    end;
    . . .
end;
```

عمق ایستایی A : 0
عمق ایستایی B : 1

آفست زنجیر: تفاضل بین عمق تودرتویی رویه حاوی ارجاع به متغیر و عمق تودرتویی رویه‌ی حاوی اعلان متغیر است.

اگر رویه B به متغیری ارجاع کند که در A اعلان شده است، آفست زنجیر آن ارجاع برابر 1 است. (تفاضل عمق ایستایی B و A).

ارجاع واقعی را با زوج مرتب (آفست محلی و آفست زنجیر) نمایش می‌دهند.

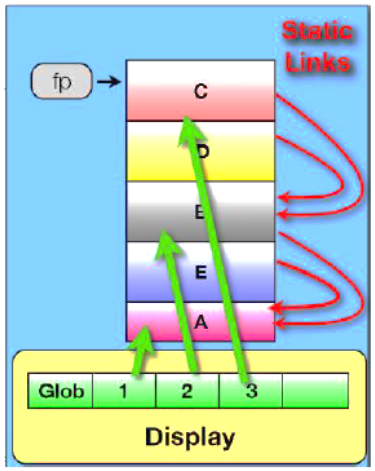
معایب زنجیر ایستا جهت دستیابی به متغیرهای غیر محلی

۱- هزینه مراجعات به حوزه‌های خارج از والد ایستا، بیشتر از مراجعات محلی است. زنجیر ایستا از مراجعه تا اعلان شیء مورد نظر باید ردیابی شود.

۲- برنامه نویس در برنامه‌هایی که نیاز به اجرای سریع است، نمی‌تواند هزینه ارجاع غیر محلی را برآورد کند، زیرا هزینه هر ارجاع به عمق تودرتویی بین مراجعه و حوزه اعلان بستگی دارد.

۳- اصلاح کد ممکن است عمق تودرتویی را تغییر دهد و در نتیجه زمان‌بندی بعضی از مراجعات تغییر کند.

نمایشگر Display



می‌توان از روش نمایشگر به جای زنجیره‌های ایستا برای دستیابی به متغیرهای غیر محلی استفاده کرد.

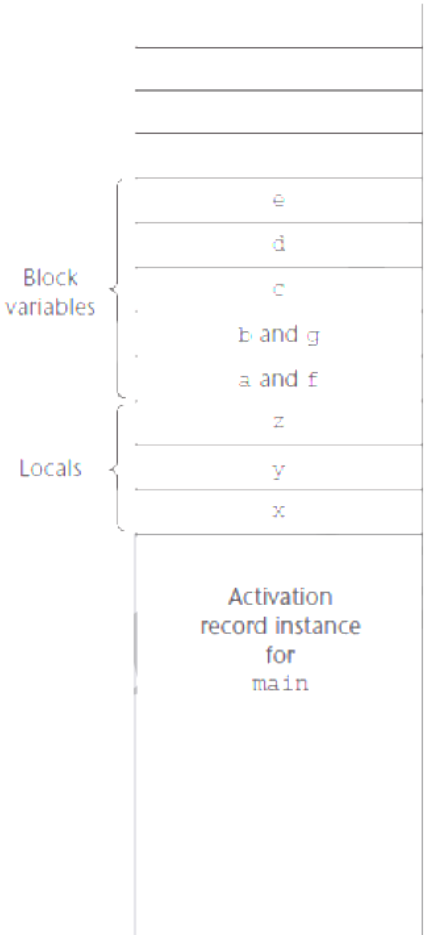
Display از static chain سریعتر است، اما نیاز به کار اضافی دارد.

در این روش، پیوندهای ایستا به جای ذخیره شدن در رکورد فعالیت در آرایه‌ای به نام نمایشگر جمع‌آوری می‌شوند.

محتویات نمایشگر در هر زمان، لیستی از آدرس‌های رکوردهای فعالیت موجود در پشته است. (یک اشاره‌گر برای هر حوزه)

مثال

```
void main( ) {
    int  x , y , z;
    while ( . . . ) {
        int a , b , c;
        . . .
        while ( . . . ) {
            int d , e;
            . . .
        }
    }
    while(...){
    int f , g;
      ...
    }
   . . .
}
```



طرح حافظه ایستا برای کد زیر در شکل آمده است:

f ، g محل‌های حافظه را با a ، b به طور اشتراکی استفاده می‌کنند، زیرا قبل از ایجاد شدن f ، g متغیرهای a ، b از پشته حذف می‌شوند.

پیاده سازی حوزه پویا

برای پیاده سازی متغیرهای محلی و مراجعات غیر محلی به آن‌ها در زبان‌هایی با حوزه پویا، می‌توان از روش‌های زیر استفاده کرد:

۱- دستیابی عمیق (deep)

۲- دستیابی سطحی (shallow)

دستیابی عمیق

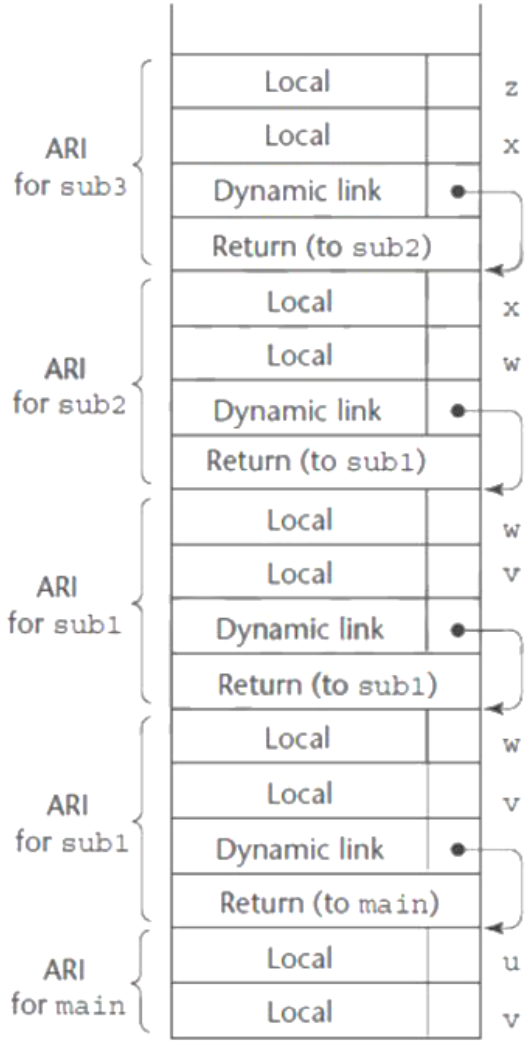
از آنجا که پیوندهای زنجیر پویا، رکوردهای فعالیت زیر برنامه‌ها را به ترتیب معکوس فعال شدن آن‌ها، به هم پیوند می‌دهند، زنجیر پویا ابزاری برای مراجعه به متغیرهای غیر محلی در زبان‌هایی با حوزه پویا است.

به این روش، دستیابی عمیق می‌گویند، زیرا مستلزم جست و جوی عمقی در پشته است.

محتویات پشته برای برنامه ای با حوزه پویا:
رکوردهای فعالیت فاقد پیوندهای ایستا هستند.

```
void   sub3( ){
   int  x , z;
   x=u+v;
   …
}
void   sub2( ){
   int  w , x;
   …
}
void   sub1( ){
   int  v , w;
   …
}
void   main( ){
  int  v , u;
  …
}
```

مثال



تفاوت‌های روش دستیابی عمیق و روش زنجیر ایستا

در زبانی با حوزه پویا، در زمان ترجمه نمی‌توان طول زنجیر جست و جو را تعیین کرد.

هر رکورد موجود در زنجیر ایستا باید جست و جو شود تا اولین وقوع متغیر مورد نظر پیدا شود.

بنابراین زبان‌هایی با حوزه پویا کندتر از زبان‌هایی با حوزه ایستا هستند.

رکوردهای فعالیت باید اسامی متغیرها را برای اهداف جستجو ذخیره کنند، در حالی که در پیاده سازی‌های زبان‌هایی با حوزه ایستا، فقط نیاز به مقادیر است.

چون متغیرها با جفت‌های (آفست محلی و آفست زنجیر) نمایش داده می‌شوند، نیاز به ذخیره نام آن‌ها نیست.

دستیابی سطحی

در این روش، متغیرهایی که در یک زیر برنامه اعلان می‌شوند، در رکوردهای فعالیت آن زیر برنامه ذخیره نمی‌گردند.

برای پیاده سازی دستیابی سطحی می‌توان برای هر نام متغیر در کل برنامه، پشته جداگانه‌ای در نظر گرفت.

همچنین می‌توان از یک جدول مرکزی استفاده کرد.

در روش دستیابی عمیق، پیوند زیر برنامه سریع انجام می‌گیرد، اما مراجعات غیر محلی دارای هزینه است.

در روش دستیابی سطحی، مراجعات غیر محلی سریع‌تر انجام می‌گیرد، ولی هزینه پیوند برنامه در آن زیاد است.