

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



اصول طراحی کامپایلر

درس ۲۰

تحلیل معنایی (۵)

سازمان حافظه‌ی زمان اجرا

Semantic Analysis (5)

Run-Time Storage Organization

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

<http://courses.fouladi.ir/compiler>

اصول طراحی کامپیوتر

سازمان حافظه‌ی زمان اجرا



مقدمه

ماژول پشتیبانی زمان اجرا

در طول اجرای یک برنامه، یک نام واحد در کد منبع می‌تواند به اشیای داده‌ای متفاوتی در کامپیوتر اشاره کند.

ماژول پشتیبانی زمان اجرا *Run-Time Support Module*

مدیریت وظیفه‌ی تخصیص و آزادسازی اشیای داده‌ای

ماژول پشتیبانی زمان اجرا

محیط، حالت، انقیاد

نگاشت یک نام به یک فضای حافظه

 $name \rightarrow storage\ space$

محیط

Environment

مقدار فعلی یک فضای حافظه

 $storage\ space \rightarrow state$

حالت

State

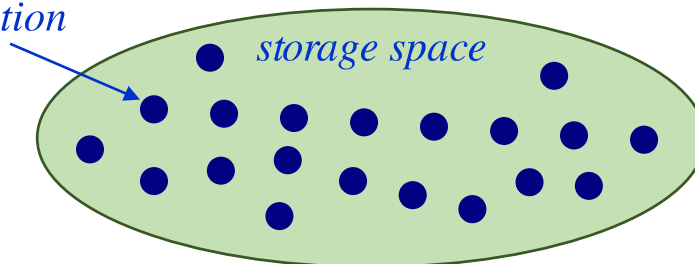
وابسته کردن یک نام به یک مکان حافظه

 $name \rightarrow storage\ location$

انقیاد

Binding

storage location



ماژول پشتیبانی زمان اجرا

«فعالیت» و «طول عمر» یک روال

فعالیت

Activation

هر اجرای یک روال، یک **فعالیت** نام دارد.

اگر یک روال بازگشتی باشد: تعداد زیادی از فعالیت آن می‌تواند در یک زمان موجود باشد.

* یک روال می‌تواند خود را به صورت غیرمستقیم نیز فراخوانی کند.

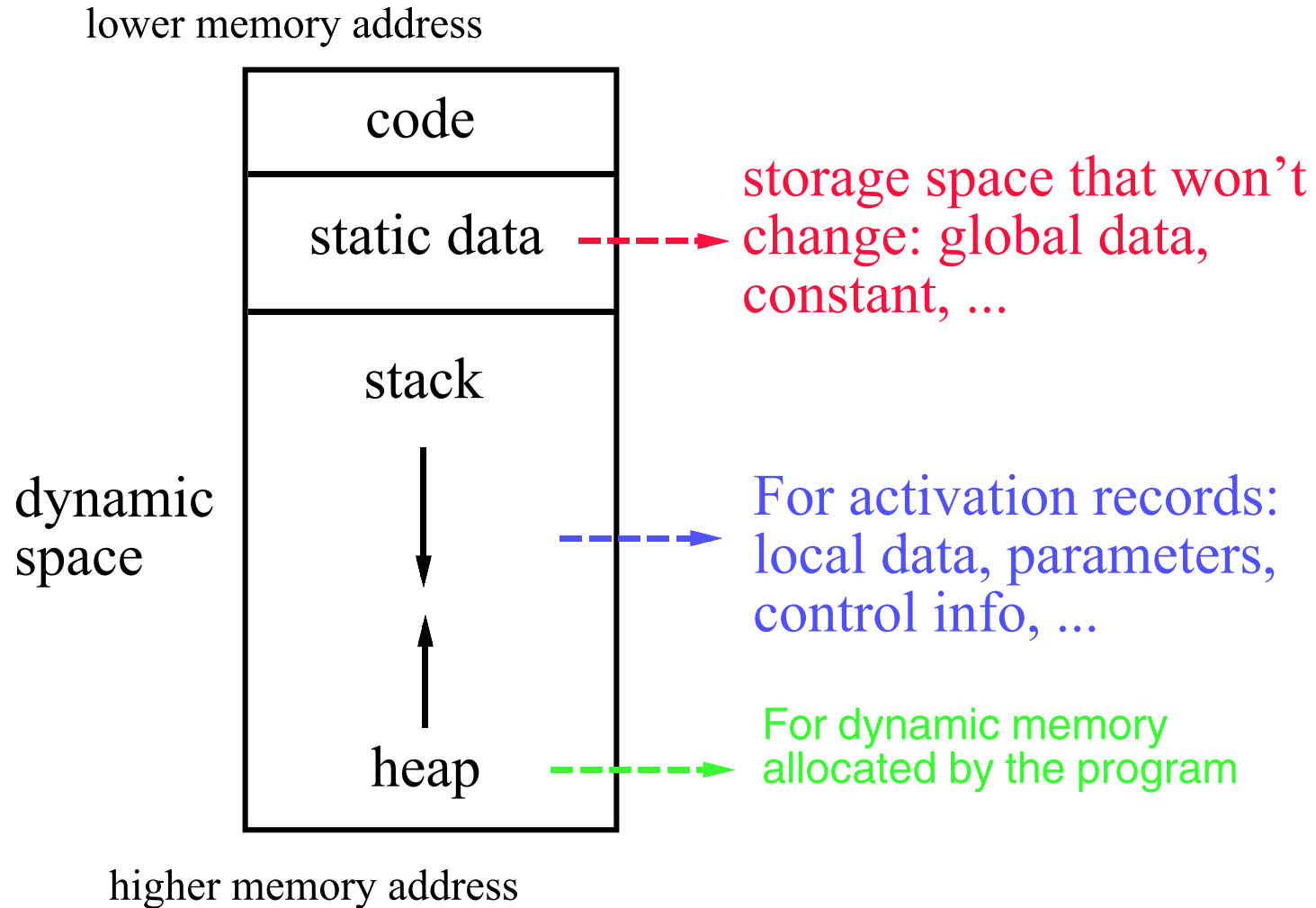
طول عمر

Life-Time

طول بازه‌ی زمانی میان نخستین گام و آخرین گام در یک روال

جانمایی فضای حافظه در زمان اجرا

Run-Time Storage Layout



رکورد فعالیت

محل نگهداری داده‌های مربوط به یک فعالیت
(داده‌های مربوط به اجرای یک روال)

رکورد فعالیت
Activation Record

returned value	مقدار برگشتی
actual parameters	پارامترهای واقعی
optional control link	پیوند کنترلی اختیاری (پویا)
optional access link	پیوند دسترسی اختیاری (ایستا)
saved machine status	وضعیت ذخیره‌شده‌ی ماشین
local data	داده‌های محلی
temporaries	داده‌های موقتی

رکورد فعالیت

پارامترهای واقعی

محل نگهداری داده‌های مربوط به یک فعالیت
(داده‌های مربوط به اجرای یک روال)

رکورد فعالیت
Activation Record



returned value
actual parameters
optional control link
optional access link
saved machine status
local data
temporaries

پارامترهای روال (parameters)

واقعی (actual)

مقدار پارامترها برای یک فعال‌سازی

صوری (formal)

اعلان پارامترها

رکورد فعالیت

پیوندها

محل نگهداری داده‌های مربوط به یک فعالیت
(داده‌های مربوط به اجرای یک روال)

رکورد فعالیت
Activation Record

returned value
actual parameters
optional control link
optional access link
saved machine status
local data
temporaries



پیوندها (links)

پیوندهای کنترلی
(control link)

پیوند پویا (dynamic)

اشاره‌گر به رکورد فعالیت
روال فراخوانی‌کننده‌ی این روال

پیوندهای دسترسی
(access link)

پیوند ایستا (static)

اشاره‌گر به داده‌های غیرمحل
نسبت به این روال

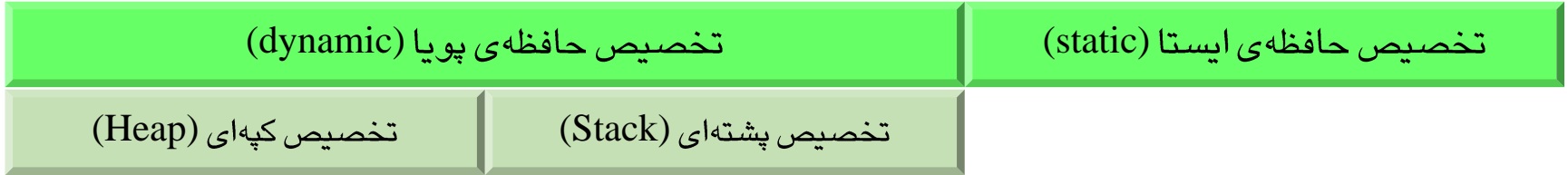
سازمان حافظه‌ی زمان اجرا

۲

تخصیص
حافظه

رویکردهای تخصیص حافظه

Memory Allocation Approaches



روی کردهای تخصیص حافظه

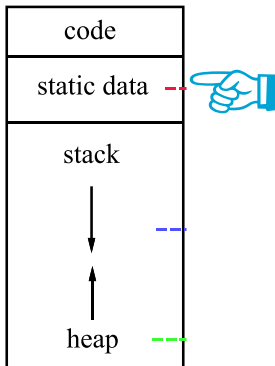
تخصیص حافظه ایستا

تخصیص حافظه‌ی پویا (dynamic)

تخصیص حافظه‌ی ایستا (static)

تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)



استفاده از ناحیه‌ی داده‌های ایستا در حافظه

رکورد فعالیت روال در ناحیه‌ی داده‌های ایستا قرار می‌گیرد.

برای هر روال، یک رکورد فعالیت

انقیاد نام‌ها به مکان‌های حافظه در زمان کامپایل

انتساب نام روال به فضای از پیش تعیین شده در حافظه هنگام فراخوانی آن

مزایا

دسترسی سریع‌تر به متغیرها
(عدم دستکاری پشته یا دسترسی غیر مستقیم به متغیرها)

باقی ماندن مقدار رکورد فعالیت از یک فراخوانی تا بعدی
(مانند متغیرهای استاتیک زبان C)

معایب

عدم امکان بازگشتی بودن روال‌ها

هدر رفتن مقدار زیادی حافظه هنگام فعال نبودن روال

عدم امکان تخصیص حافظه پویا (در زمان اجرا)

روی کردهای تخصیص حافظه

تخصیص حافظه‌ی ایستا: فرآیند فراخوانی روال‌ها

تخصیص حافظه‌ی ایستا (static)

تخصیص حافظه‌ی پویا (dynamic)

تخصیص پشته‌ای (Stack)

تخصیص کپه‌ای (Heap)

فرآیند فراخوانی روال‌ها

در روال فراخوانی شونده (called)

۱) کپی آدرس برگشت از ثبات RA در فیلد آدرس برگشت رکود فعالیت روال فراخوانی شونده

۲) ذخیره‌سازی مقدار برخی ثبات‌ها (در صورت لزوم)

۳) مقداردهی اولیه‌ی داده‌های محلی (در صورت لزوم)

در روال فراخوانی کننده (calling)

۱) ارزیابی آرگومان‌ها

۲) کپی آرگومان‌ها در فضای پارامتر رکورد فعالیت روال فراخوانی کننده

۳) ذخیره‌سازی مقدار برخی ثبات‌ها در رکورد فعالیت روال فراخوانی کننده (در صورت لزوم)

۴) پرش و پیونددهی:
پرش به نخستین دستورالعمل روال فراخوانی شونده و قرار دادن آدرس برگشت (آدرس دستورالعمل بعدی) در ثبات آدرس برگشت (RA)

قرار داد: آنچه به یک روال پاس می‌شود، از نگاه فراخوانی کننده آرگومان و از نگاه فراخوانی شونده پارامتر نام دارد.

روی‌کردهای تخصیص حافظه

تخصیص حافظه‌ی ایستا: فرآیند برگشت از روال‌ها

تخصیص حافظه‌ی پویا (dynamic)

تخصیص حافظه‌ی ایستا (static)

تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)

فرآیند برگشت از روال‌ها

در روال فراخوانی‌کننده (calling)

- ۱) بازیابی مقادیر برخی ثبات‌ها (در صورت لزوم)
- ۲) قرار دادن مقدار برگشتی در یک مکان مناسب (اگر روال فراخوانی‌شونده، یک تابع باشد)

در روال فراخوانی‌شونده (called)

- ۱) بازیابی مقادیر ثبات‌های ذخیره‌شده
- ۳) پرش به آدرس موجود در فیلد آدرس برگشت

روی کردهای تخصیص حافظه

تخصیص حافظه‌ی پویا

تخصیص حافظه‌ی پویا (dynamic)

تخصیص حافظه‌ی ایستا (static)

تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)

روی کردهای تخصیص حافظه

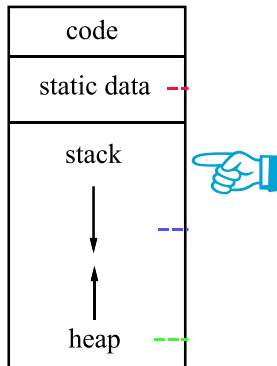
تخصیص حافظه‌ی پویا: تخصیص پشته‌ای

تخصیص حافظه‌ی پویا (dynamic)

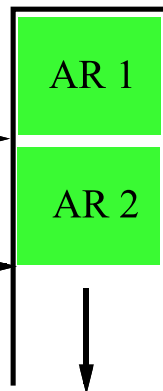
تخصیص حافظه‌ی ایستا (static)

تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)



stack

FP
frame pointerSP
stack pointer

استفاده از ناحیه‌ی پشته در حافظه

رکورد فعالیت جدید در بالای ناحیه‌ی پشته درج می‌شود (هنگام فراخوانی روال)

رکورد فعالیت موجود از بالای پشته برداشته می‌شود (هنگام برگشت از روال)

ثبات اشاره‌گر پشته (SP): به بالای پشته اشاره می‌کند (بزرگترین آدرس)

ثبات اشاره‌گر قاب (FP): به شروع رکورد فعالیت جاری اشاره می‌کند (بزرگترین آدرس)

رویکردهای تخصیص حافظه

تخصیص حافظه‌ی پویا: تخصیص پشته‌ای: فرآیند فراخوانی روال‌ها

تخصیص حافظه‌ی پویا (dynamic)

تخصیص حافظه‌ی ایستا (static)

تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)

فرآیند فراخوانی روال‌ها

در روال فراخوانی شونده (called)

۱) درج آدرس برگشت در ثبات آدرس برگشت (RA)

۲) درج FP قدیمی (پیوند کنترل پویا) در پشته (اشاره‌گر به رکورد فعالیت روال فراخوانی‌کننده)

۳) مقداردهی FP جدید با SP قدیمی

۴) مقداردهی FP جدید با SP قدیمی + (اندازه‌ی پارامترها) + (اندازه‌ی RA) + (اندازه‌ی FP)
[این اندازه‌ها در زمان کامپایل محاسبه می‌شوند.]

۵) ذخیره‌سازی مقدار برخی ثبات‌ها (در صورت لزوم)

۶) درج کردن داده‌های محلی در پشته

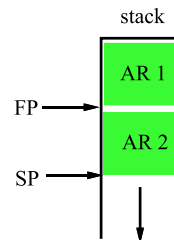
در روال فراخوانی‌کننده (calling)

۱) ذخیره‌سازی مقدار برخی ثبات‌ها در رکورد فعالیت روال فراخوانی‌کننده (در صورت لزوم)

۲) تنظیم پیوند دسترسی ایستا (درج در پشته) (در صورت لزوم)

۳) درج پارامترها در پشته

۴) پرش و پیونددهی:
پرش به نخستین دستورالعمل روال فراخوانی‌شونده و قرار دادن آدرس برگشت (آدرس دستورالعمل بعدی) در ثبات آدرس برگشت (RA)



روی‌کردهای تخصیص حافظه

تخصیص حافظه‌ی پویا: تخصیص پشته‌ای: فرآیند برگشت از روال‌ها

تخصیص حافظه‌ی پویا (dynamic)

تخصیص حافظه‌ی ایستا (static)

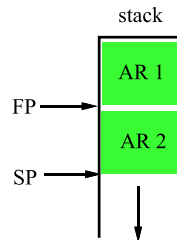
تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)

فرآیند برگشت از روال‌ها

در روال فراخوانی‌کننده (calling)

- ۱) بازیابی مقادیر برخی ثبات‌ها (در صورت لزوم)
- ۲) قرار دادن مقدار برگشتی در یک مکان مناسب (اگر روال فراخوانی‌شونده، یک تابع باشد)



در روال فراخوانی‌شونده (called)

- ۱) بازیابی مقادیر ثبات‌های ذخیره‌شده
- ۲) بارگذاری آدرس برگشت در ثبات خاص RA
- ۳) بازیابی اشاره‌گر پشته SP
 $SP := FP$
- ۴) بازیابی اشاره‌گر قاب FP
 $FP := \text{saved FP}$
- ۵) پرش به آدرس موجود در فیلد آدرس برگشت

درخت فعالیت

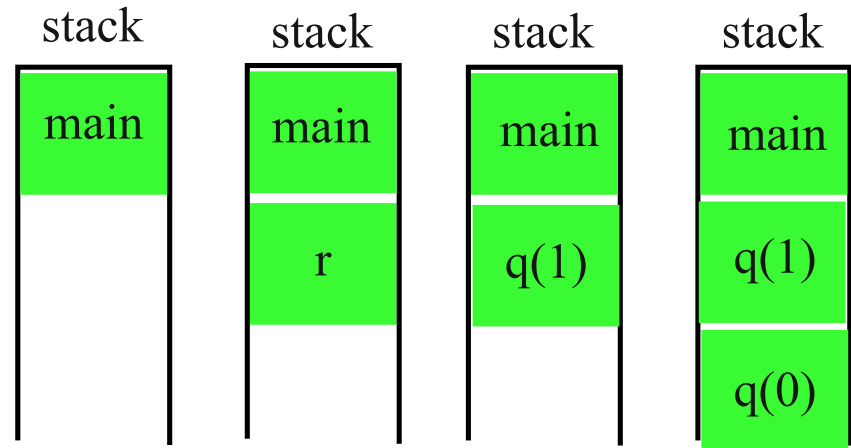
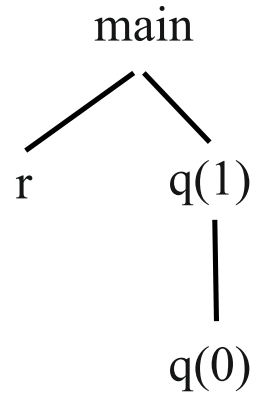
درخت فعالیت Activation Tree

ساختاری درختی برای ثبت ترتیب سلسله‌مراتبی و تغییرات رکوردهای فعالیت

```
main{
  r();
  q(1);
}

r{
  ...}

q(int i)
{
  if(i>0) then q(i-1)
}
```



روی کردهای تخصیص حافظه

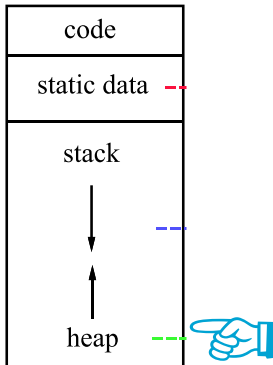
تخصیص حافظه‌ی پویا: تخصیص کپه‌ای

تخصیص حافظه‌ی پویا (dynamic)

تخصیص حافظه‌ی ایستا (static)

تخصیص کپه‌ای (Heap)

تخصیص پشته‌ای (Stack)



استفاده از ناحیه‌ی کپه در حافظه

برای حافظه‌هایی که از سوی برنامه‌نویس در زمان اجرا درخواست می‌شود.

مانند دستورات `new` و `delete` برای حافظه‌ی پویا در زبان `C++`

مسائل مرتبط با تخصیص حافظه‌ی کپه‌ای

ارجاعات سرگردان
Dangling References

قطعه‌بندی حافظه
Memory Segmentation

جمع‌آوری زباله‌ها
Garbage Collection

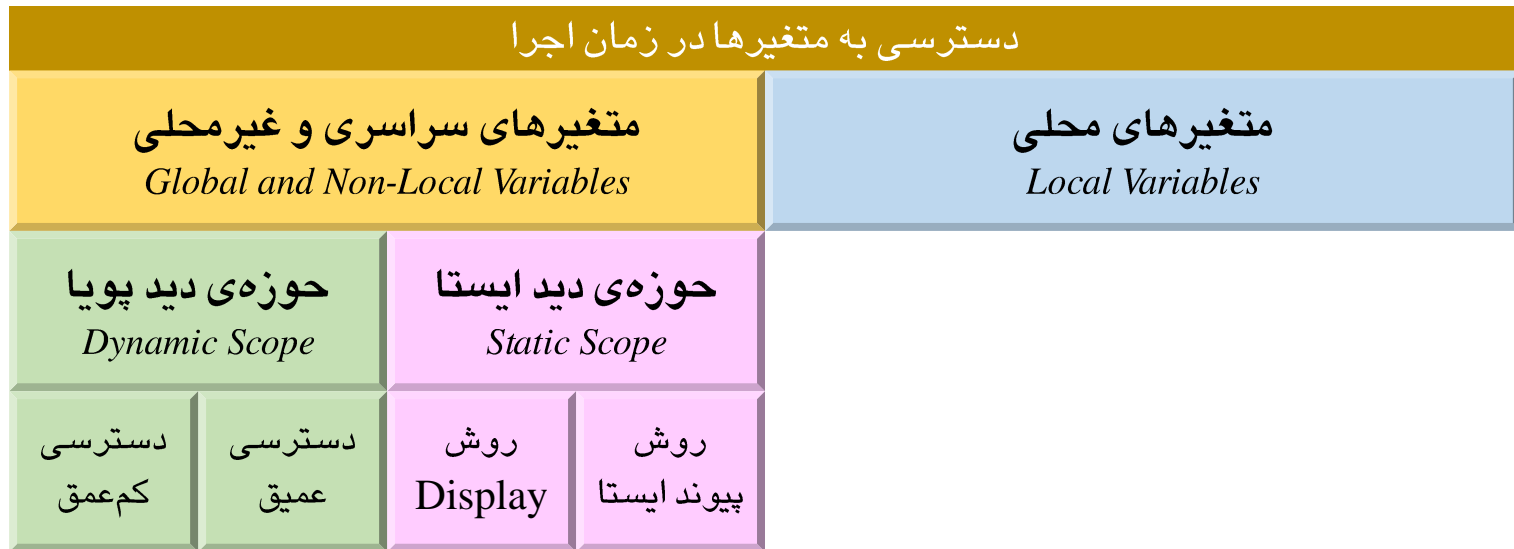
پیاده‌سازی این موارد کم و بیش به سیستم عامل وابسته است.

سازمان حافظه‌ی زمان اجرا

۳

دسترسی
به
متغیرها
در
زمان اجرا

دسترسی به متغیرها در زمان اجرا



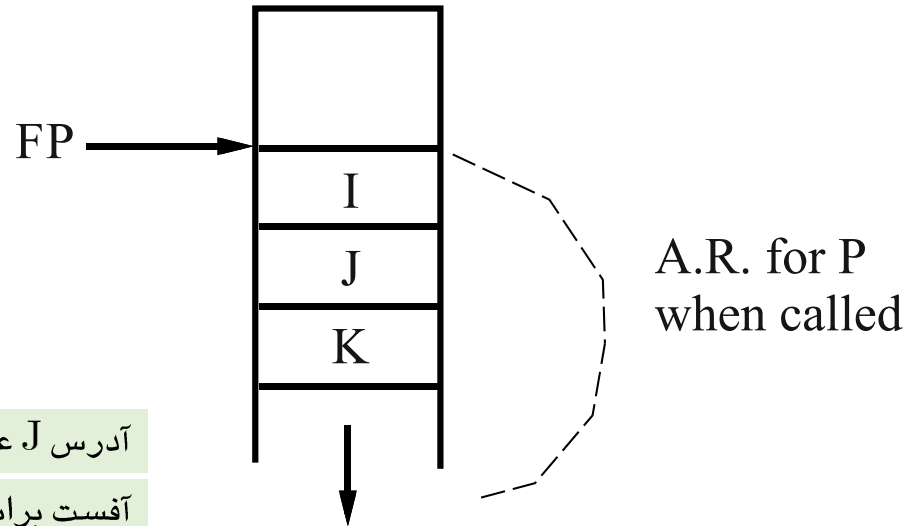
دسترسی به متغیرهای محلی در زمان اجرا

متغیرهای محلی
Local Variables

ذخیره‌سازی متغیرهای محلی در رکورد فعالیت روال اعلان‌کننده‌ی آنها

دسترسی به متغیرهای محلی از طریق یک آفست نسبت به اشاره‌گر قاب (FP)

```
P()
{
  int I,J,K;
  ...
}
```

آدرس J عبارت است از: $FP + 1 * \text{sizeof}(\text{int})$ آفست برابر است با: $1 * \text{sizeof}(\text{int})$

آفست برای هر متغیر محلی، در زمان کامپایل معلوم است.

آدرس مطلق، تنها در زمان اجرا قابل تعیین است.

دسترسی به متغیرهای سراسری و غیرمحلی در زمان اجرا

متغیرهای سراسری

Global Variables

متغیرهای سراسری در ناحیه‌ی داده‌های ایستا ذخیره می‌شوند:

- * دسترسی به آنها تنها از طریق نام آنهاست.
- * آدرس آنها در زمان کامپایل معلوم است.

متغیرهای غیرمحلی

Non-Local Variables

حوزه‌ی دید پویا

Dynamic Scope

پاسخ به پرسش «کدام متغیر غیرمحلی استفاده می‌شود؟» نمی‌تواند در زمان کامپایل مشخص شود و تنها در زمان اجرا قابل تعیین است.

استفاده از یک متغیر غیرمحلی متناظر با اعلان در «تازه‌ترین روال فراخوانی شده‌ی هم‌اکنون فعال»

مانند زبان LISP

حوزه‌ی دید ایستا (لغوی)

Static Scope (Lexical)

آدرس یک نام محلی می‌تواند در زمان کامپایل با تحلیل نحوی مشخص شود.

مانند زبان‌های C، JAVA

حوزهی دید ایستا (لغوی)

برای ساختار بلاک‌بندی شده

بلاک، در یک زبان برنامه‌سازی، مجموعه‌ای از دستورها است که حاوی اعلان داده‌های محلی خود می‌باشد.

بلاک
Block

قاعدهی حوزهی دید

قاعدهی نزدیک‌ترین حوزهی تودرتو *Most Closely Nested Rule*

- حوزهی دید یک اعلان در بلاک B شامل خود B است.
- اگر x در B استفاده شود، اما در B تعریف نشده باشد، آنگاه به x در بلاک B' مراجعه می‌کنیم که در آن:
 - B' حاوی اعلان x است.
 - B' نسبت به سایر بلاک‌های حاوی اعلان x ، نزدیک‌ترین بلاک پیرامون B است.

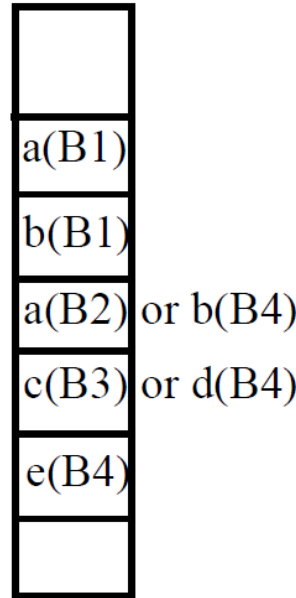
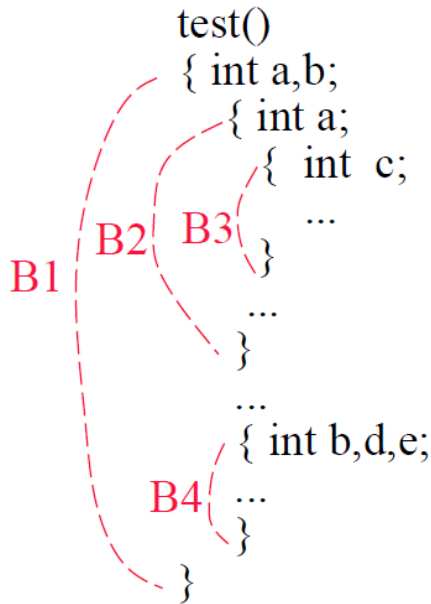
حوزهی دید ایستا (لغوی)

برای ساختار بلاکبندی شده، بدون روال‌های تودرتو

اگر یک زبان روال‌های تودرتو را مجاز نداند:

یک متغیر

- یا سراسری است
- یا نسبت به روال حاوی آن محلی است.



در زمان اجرا:

همه‌ی متغیرهای اعلان‌شده در یک روال (شامل آنهایی که در بلاک‌ها هستند) در رکورد فعالیت آن روال ذخیره می‌شوند (امکان همپوشانی وجود دارد).

در زمان کامپایل:

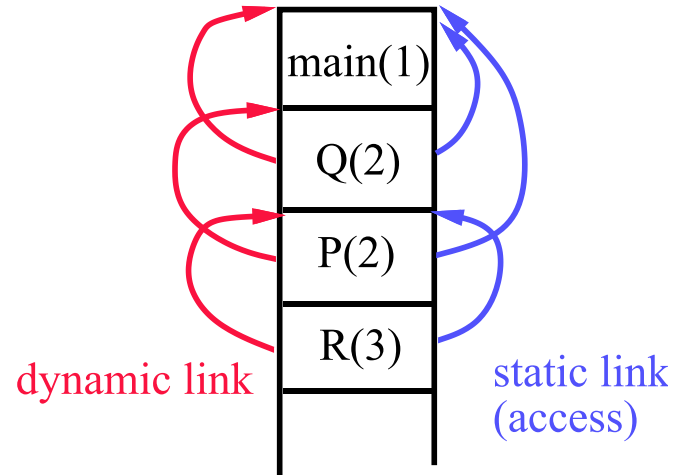
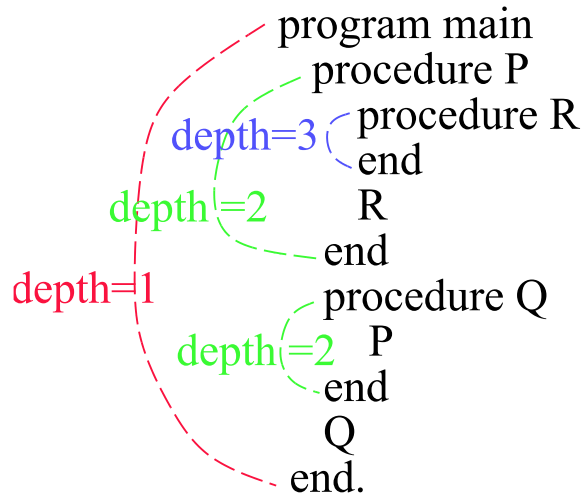
آفست صحیح برای هر داده‌ی محلی با استفاده از اطلاعات معلوم از ساختار بلاکی محاسبه می‌شود.

حوزهی دید ایستا (لغوی)

برای ساختار بلاک‌بندی شده، با روال‌های تودرتو

هر متغیر به یک عمق تودرتویی نسبت داده می‌شود.

- اگر یک روال با عمق h داشته باشیم و به متغیری غیرمحلی در عمق k دسترسی پیدا کنیم، آن‌گاه $h \geq k$
- پیوند دسترسی ایستا را $h - k$ مرتبه طی می‌کنیم؛
- سپس از اطلاعات آفست برای یافتن آدرس استفاده می‌کنیم.



- عمق برنامه‌ی اصلی برابر با ۱ است.
- هرگاه به یک روال تودرتو وارد می‌شویم، یک واحد به عمق اضافه می‌شود.
- هرگاه از یک روال تودرتو خارج می‌شویم، یک واحد از عمق اضافه می‌شود.

عمق
depth

حوزه‌ی دید ایستا (لغوی)

برای ساختار بلاک‌بندی شده، با روال‌های تودرتو: الگوریتم تنظیم پیوندها

تنظیم پیوندها

تنظیم پیوندهای ایستا

روال p (عمق n_p) روال x (عمق n_x)

اگر روال p ، روال x را فراخوانی کند

وقتی $n_p < n_x$ باشد، آن‌گاه

x در p محصور می‌شود و $n_p = n_x - 1$
(مشابه تنظیم پیوندپویا)

وقتی $n_p \geq n_x$ باشد، آن‌گاه

یک فراخوانی بازگشتی یا
فراخوانی یک روال از پیش اعلان‌شده را داریم.

تنظیم پیوندهای پویا

به رکورد فعالیت روال فراخوانی‌کننده
تنظیم می‌شود.

با یک مرتبه بالا رفتن از پیوند دسترسی،
عمق یک واحد کاهش می‌یابد.

پس:

پیوند دسترسی x ، همان پیوند دسترسی p است
که $n_p - n_x + 1$ مرتبه از آن بالا رفته‌ایم.

حوزهی دید ایستا (لغوی)

برای ساختار بلاکبندی شده، با روالهای تودرتو: الگوریتم تنظیم پیوندها: مثال

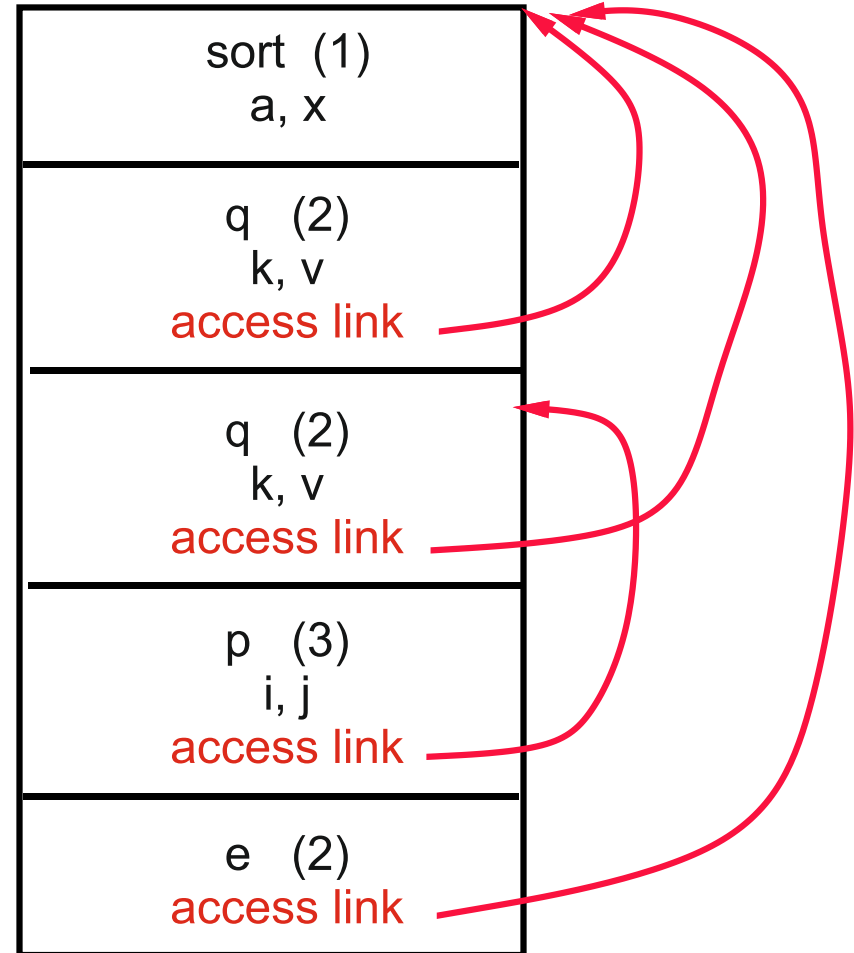
```

Program sort
  var a: array[0..10] of int;
      x: int;
  procedure r
  var i: int;
  begin ... r
  end

  procedure e(i,j)
  begin ... e
    a[i] <-> a[j]
  end

  procedure q
    var k,v: int;
    procedure p
    var i,j;
    begin ... p
      call e
    end
    begin ... q
      call q
    end

  begin ... sort
    call q
  end
  
```



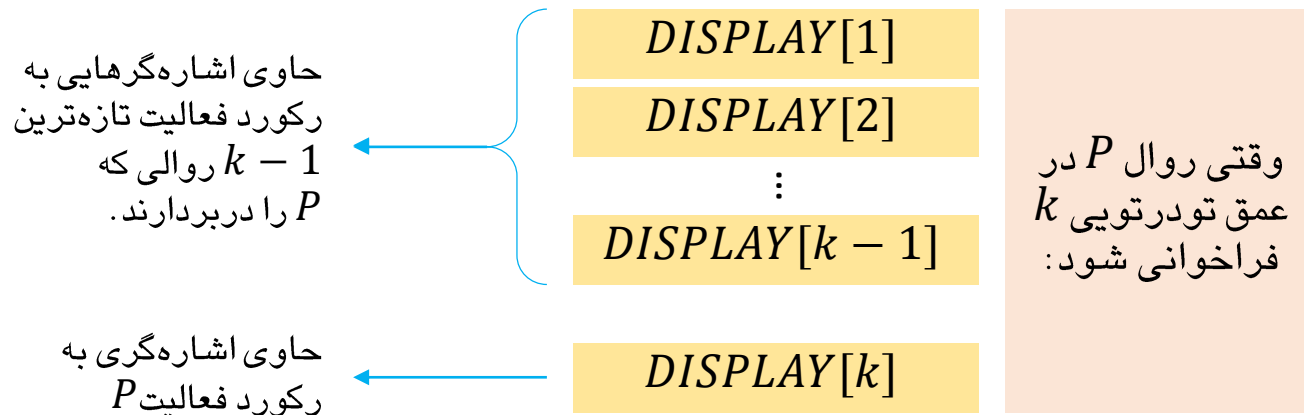
حوزهی دید ایستا (لغوی)

دسترسی به داده‌های غیرمحلی با استفاده از نمایشگرها (DISPLAY)

ایجاد یک آرایه‌ی سراسری با نام *DISPLAY*


با استفاده از ناحیه‌ی داده‌های ایستا

با استفاده از ثبات‌ها



- برای دسترسی به یک متغیر x که در عمق d اعلان شده است،
- از $DISPLAY[d]$ برای دسترسی به رکورد فعالیت دربردارنده‌ی x استفاده می‌شود،
- سپس، از آفست آن برای دسترسی به خود x استفاده می‌شود.

اندازه‌ی *DISPLAY* = ماکزیمم عمق تودرتویی روال‌ها

این تکنیک برای زبان‌هایی که بازگشت را مجاز می‌دانند، نامناسب است. 

حوزه‌ی دید ایستا (لغوی)

دسترسی به داده‌های غیرمحلّی با استفاده از نمایشگرها (DISPLAY)

مقدار فعلی $DISPLAY$ در فیلد $save-display$ رکورد فعالیت جدید ذخیره می‌شود.

$DISPLAY[k]$ به گونه‌ای تنظیم می‌شود که به رکورد فعالیت جدید اشاره کند (مثلاً به فیلد $save-display$ خود)

$DISPLAY[k]$ با استفاده از مقدار ذخیره‌شده در فیلد $save-display$ بازیابی می‌شود.

وقتی روال P در عمق
تودرتویی k
فراخوانی شود:

هنگام برگشت از روال

حوزه‌ی دید ایستا (لغوی)

مقایسه‌ی «پیوند دسترسی (access link)» با «نمایشگرها (DISPLAY)»

<i>DISPLAY</i>	پیوند دسترسی ایستا
نیاز به زمان کمتر برای دسترسی به داده‌های غیرمحلی در زمان اجرا	نیاز به زمان بیشتر برای دسترسی به داده‌های غیرمحلی در زمان اجرا (بخصوص زمانی که فاصله‌ی عمق داده‌های غیرمحلی از هم زیاد باشد.)
نیازمند فضای نسبتاً بیشتر (در زمان اجرا)	نیازمند فضای نسبتاً کمتر (در زمان اجرا)
کد تولیدشده‌ی ساده‌تر	کد تولید شده‌ی پیچیده‌تر

حوزهی دید پویا

حوزهی دید پویا

Dynamic Scope

پاسخ به پرسش «کدام متغیر غیرمحلی استفاده می شود؟»
نمی تواند در زمان کامپایل مشخص شود
و تنها در زمان اجرا قابل تعیین است.

مانند زبان LISP

استفاده از یک متغیر غیرمحلی متناظر با اعلان در
«تازه ترین روال فراخوانی شده ی هم اکنون فعال»

لازمه: ذخیره سازی جدول نمادها برای استفاده در زمان اجرا

پیاده سازی

حوزهی دید پویا

Dynamic Scope

دسترسی کم عمق (shallow)

دسترسی عمیق (deep)

حوزهی دید پویا

دسترسی عمیق

پیاده‌سازی

حوزهی دید پویا

Dynamic Scope

دسترسی کم عمق (shallow)

دسترسی عمیق (deep)

در صورت استفاده از یک متغیر غیر محلی:
از پیوندهای کنترل پویا استفاده می‌شود تا در پشته
تازه‌ترین رکورد فعالیتی که حاوی آن متغیر باشد، پیدا شود.

(باید مشخص باشد که چه متغیرهایی در هر رکورد فعالیت ذخیره شده است.)

استفاده از جدول نمادها در زمان اجرا

حوزهی دید پویا

دسترسی عمیق: مثال

```

program main
  procedure test
    var x : int;
  begin
    x := 30
    call DeclaresX
    call UsesX
  end
  procedure DeclaresX
    var x: int;
  begin
    x := 100
    call UsesX
  end
  procedure UsesX
  begin
    write(x)
  end
begin
  call test
end

```

کدام X در روال UsesX استفاده می شود؟

در صورت استفاده از حوزهی دید ایستا،
این دستور مجاز نخواهد بود؛
زیرا هیچ حوزهی احاطه کننده ای، X را اعلان نکرده است!

حوزهی دید پویا

دسترسی کم عمق

پیاده سازی

حوزهی دید پویا

Dynamic Scope

دسترسی کم عمق (shallow)

دسترسی عمیق (deep)

لیستی از متغیرهای جاری نگهداری می شود.

به هر نام متغیر موجود در برنامه، یک فضا تخصیص داده می شود.
(در ثبات ها یا در ناحیهی داده های ایستا)
* یعنی یک فضا برای هر x حتی اگر چندین اعلان مختلف برای x وجود داشته باشد.

برای هر ارجاع به x ، کد تولید شده به یک مکان ثابت خاص مراجعه می کند.

مقادیر فعلی همه ی متغیرها اعلان شده توسط روال فراخوانی شونده در رکورد فعالیت آن روال ذخیره می شود.

وقتی کار روال تمام شد، آن مقادیر بازیابی می شوند.

حوزه‌ی دید پویا

مقایسه‌ی «دسترسی عمیق» با «دسترسی کم عمق»

دسترسی کم عمق

دسترسی سریع‌تر به داده‌های غیرمحلی

سربار بیشتر در ورود و خروج از روال‌ها
(متناسب با تعداد متغیرهای محلی)

دسترسی عمیق

دسترسی کندتر به داده‌های غیرمحلی

سربار کمتر در ورود و خروج از روال‌ها

گذر دادن پارامترها به روالها

Parameter Passing

پارامترهای روال (parameters)	
پارامترهای واقعی (actual)	پارامترهای صوری (formal)
آرگومانها در فراخوانی یک روال	پارامترها در اعلان یک روال

مقادیر یک متغیر	
L-value	R-value
مکان/آدرس متغیر (مقدار سمت چپ عمل انتساب)	مقدار فعلی متغیر (مقدار سمت راست عمل انتساب)

$$x := y$$

روشهای گذر دادن پارامترها به روالها			
فراخوانی با نام <i>Call by Name</i>	فراخوانی با مقدار-نتیجه <i>Call by Value-Result</i>	فراخوانی با مرجع <i>Call by Reference</i>	فراخوانی با مقدار <i>Call by Value</i>