اصول طراحی کامپایلر

درس ۲۰

تحلیل معنایی (۵)

سازمان حافظه‌ی زمان اجرا

Semantic Analysis (5)

Run-Time Storage Organization

کاظم فولادی قلعه

دانشکده مهندسی، پردیس فارابی

دانشگاه تهران

http://courses.fouladi.ir/compiler

سازمان حافظه‌ی زمان اجرا

۱

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

محیط
Environment

نگاشت یک نام به یک فضای حافظه

name → storage space

حالت
State

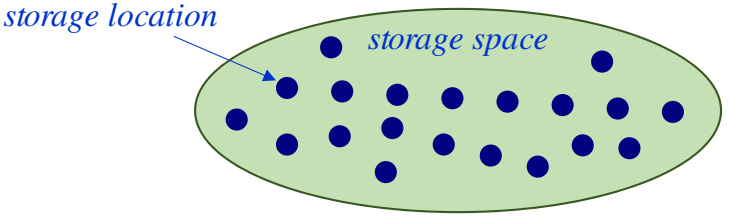
مقدار فعلی یک فضای حافظه

storage space → state

انقیاد
Binding

وابسته کردن یک نام به یک مکان حافظه

name → storage location



ماژول پشتیبانی زمان اجرا

«فعالیت» و «طول عمر» یک روال

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

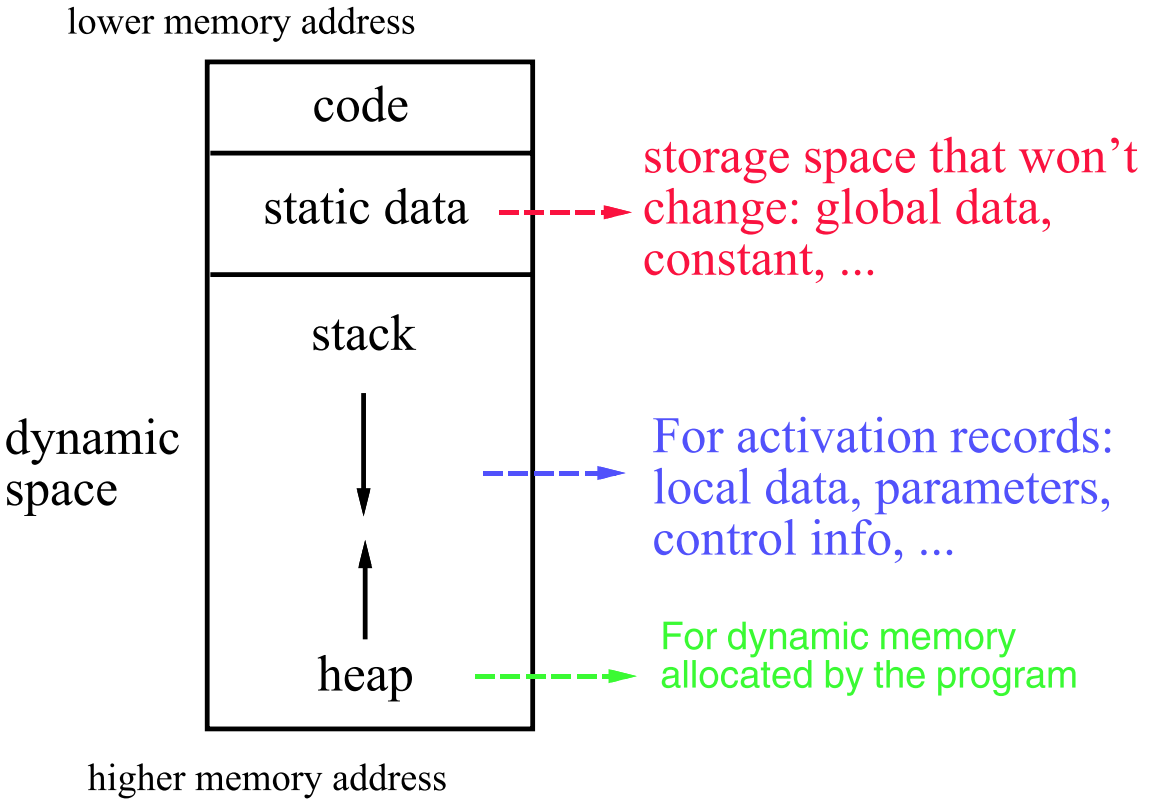
اگر یک روال بازگشتی باشد: تعداد زیادی از فعالیت آن می‌تواند در یک زمان موجود باشد.

* 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

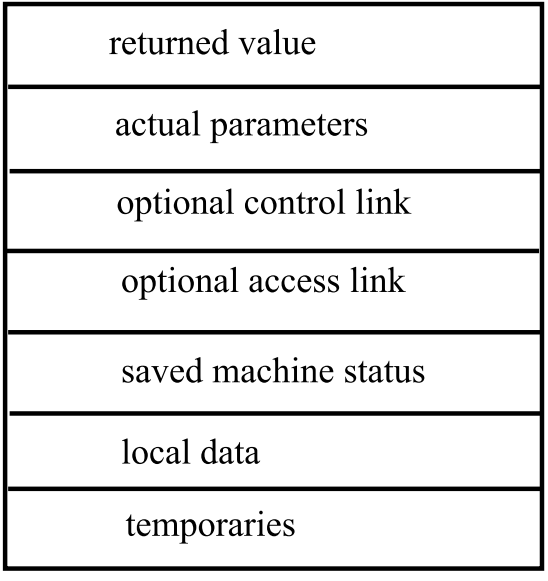
RUN-TIME STORAGE LAYOUT



رکورد فعالیت

رکورد فعالیت
Activation Record

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



مقدار برگشتی
پارامترهای واقعی
پیوند کنترلی اختیاری (پویا)
پیوند دسترسی اختیاری (ایستا)
وضعیت ذخیره‌شده‌ی ماشین
داده‌های محلی
داده‌های موقتی

رکورد فعالیت

پارامترهای واقعی

رکورد فعالیت
Activation Record

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

پارامترهای روال (parameters)	
صوری (formal)	واقعی (actual)
اعلان پارامترها	مقدار پارامترها برای یک فعال‌سازی

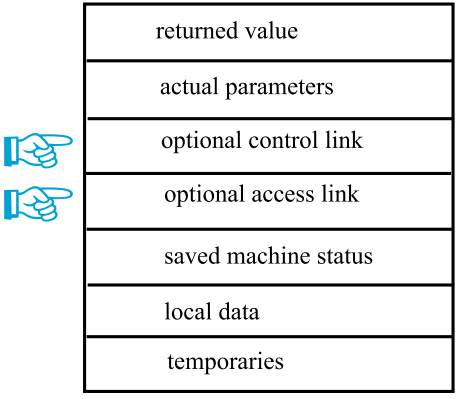
returned value
actual parameters
optional control link
optional access link
saved machine status
local data
temporaries

رکورد فعالیت

پیوندها

رکورد فعالیت
Activation Record

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



پیوندها (links)

پیوندهای دسترسی (access link)	پیوندهای کنترلی (control link)
پیوند ایستا (static)	پیوند پویا (dynamic)
اشاره‌گر به داده‌های غیرمحلی نسبت به این روال	اشاره‌گر به رکورد فعالیت روال فراخوانی‌کننده‌ی این روال

اصول طراحی کامپایلر

سازمان حافظه‌ی زمان اجرا

۲

تخصیص حافظه

روی‌کردهای تخصیص حافظه

MEMORY ALLOCATION APPROACHES

تخصیص حافظه‌ی ایستا (static)	تخصیص حافظه‌ی پویا (dynamic)	
	تخصیص پشته‌ای (Stack)	تخصیص کپه‌ای (Heap)

روی‌کردهای تخصیص حافظه

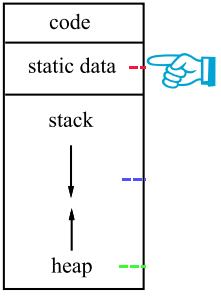
تخصیص حافظه‌ی ایستا

تخصیص حافظه‌ی ایستا (static)

تخصیص حافظه‌ی پویا (dynamic)

تخصیص پشته‌ای (Stack)

تخصیص کپه‌ای (Heap)



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 تعیین‌شده 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)

روی‌کردهای 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 فراخوانی‌کننده (calling)

۱) ارزیابی آرگومان‌ها

۲) کپی آرگومان‌ها در فضای پارامتر رکورد فعالیت روال فراخوانی‌کننده

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

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

در روال فراخوانی‌شونده (called)

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

۲) ذخیره‌سازی مقدار برخی ثبات‌ها (در صورت لزوم)

۳) مقداردهی اولیه‌ی داده‌های محلی (در صورت لزوم)

قرارداد: آنچه به یک روال پاس می‌شود، از نگاه فراخوانی‌کننده **آرگومان** و از نگاه فراخوانی‌شونده **پارامتر** نام دارد.

روی‌کردهای تخصیص حافظه

تخصیص حافظه‌ی ایستا : فرآیند برگشت از روال‌ها

تخصیص حافظه‌ی ایستا (static)

تخصیص حافظه‌ی پویا (dynamic)

تخصیص پشته‌ای (Stack)

تخصیص کپه‌ای (Heap)

فرآیند برگشت از روال‌ها

در روال فراخوانی‌شونده (called)

۱) بازیابی مقادیر ثبات‌های ذخیره‌شده

۳) پرش به آدرس موجود در فیلد آدرس برگشت

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)

روی‌کردهای تخصیص حافظه

تخصیص حافظه‌ی پویا

تخصیص حافظه‌ی ایستا (static)	تخصیص حافظه‌ی پویا (dynamic)	
	تخصیص پشته‌ای (Stack)	تخصیص کپه‌ای (Heap)

روی‌کردهای تخصیص حافظه

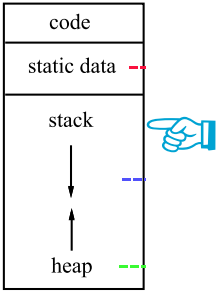
تخصیص حافظه‌ی پویا : تخصیص پشته‌ای

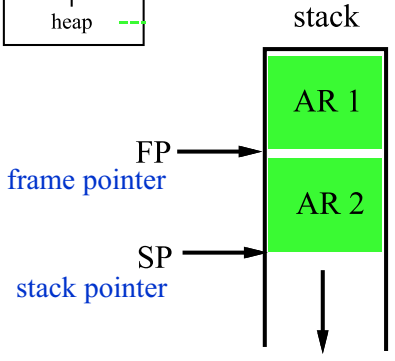
تخصیص حافظه‌ی ایستا (static)

تخصیص حافظه‌ی پویا (dynamic)

تخصیص پشته‌ای (Stack)

تخصیص کپه‌ای (Heap)





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 پشته‌ای (Stack)

فرآیند فراخوانی روال‌ها

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)

در روال فراخوانی‌شونده (called)

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

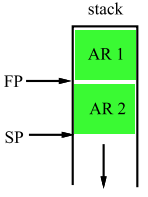
۲) درج FP قدیمی (پیوند کنترل پویا) در پشته (اشاره‌گر به رکورد فعالیت روال فراخوانی‌کننده)

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



روی‌کردهای تخصیص حافظه

تخصیص حافظه‌ی پویا: تخصیص پشته‌ای: فرآیند برگشت از روال‌ها

تخصیص حافظه‌ی ایستا (static)

تخصیص حافظه‌ی پویا (dynamic)

تخصیص پشته‌ای (Stack)

تخصیص کپه‌ای (Heap)

فرآیند برگشت از روال‌ها

در روال فراخوانی‌شونده (called)

۱) بازیابی مقادیر ثبات‌های ذخیره‌شده

۲) بارگذاری آدرس برگشت در ثبات خاص RA

۳) بازیابی اشاره‌گر پشته SP
SP := FP

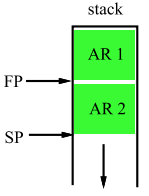
۴) بازیابی اشاره‌گر قاب FP
FP := saved FP

۵) پرش به آدرس موجود در فیلد آدرس برگشت

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)



درخت فعالیت

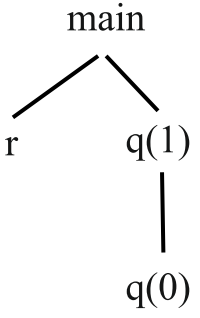
درخت فعالیت
Activation Tree

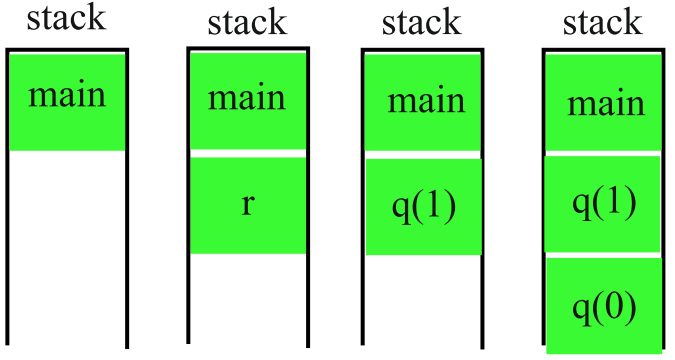
ساختاری درختی برای ثبت ترتیب سلسله‌مراتبی و تغییرات رکوردهای فعالیت

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

r{
...}

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





روی‌کردهای تخصیص حافظه

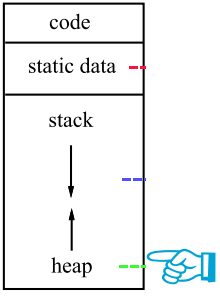
تخصیص حافظه‌ی پویا: تخصیص کپه‌ای

تخصیص حافظه‌ی ایستا (static)

تخصیص حافظه‌ی پویا (dynamic)

تخصیص پشته‌ای (Stack)

تخصیص کپه‌ای (Heap)



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

جمع‌آوری زباله‌ها	قطعه‌بندی حافظه	ارجاعات سرگردان
Garbage Collection	*Memory Segmentation*	*Dangling References*

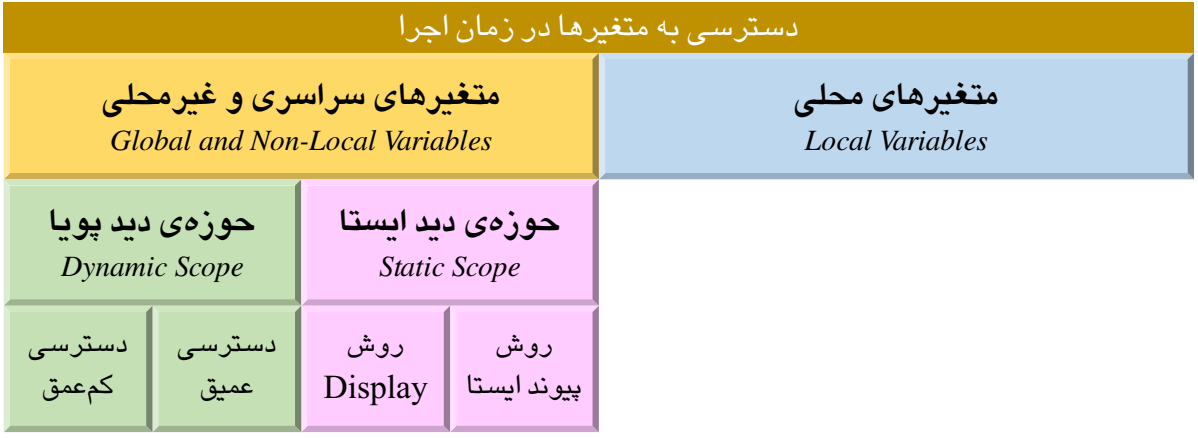
پیاده‌سازی این موارد کم و بیش به سیستم عامل وابسته است.

سازمان حافظه‌ی زمان اجرا

۳

دسترسی به متغیرها در زمان اجرا

دسترسی به متغیرها در زمان اجرا



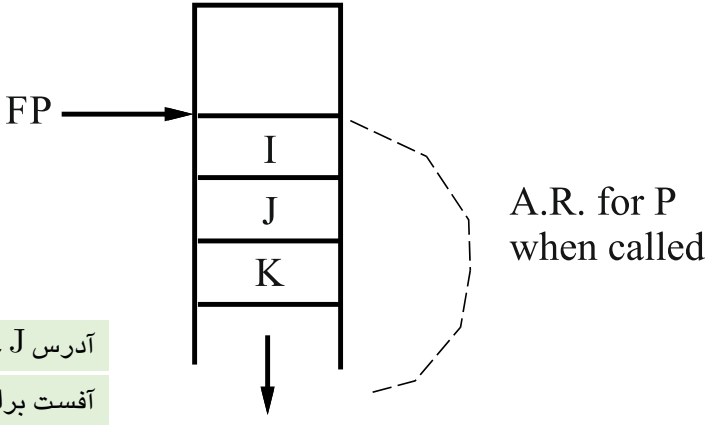
دسترسی به متغیرهای محلی در زمان اجرا

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: $FP + 1 * sizeof(int)$

آفست برابر است با: $1 * sizeof(int)$

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 ایستا (لغوی)
Static Scope (Lexical)

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

مانند زبان‌های C، JAVA

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 فراخوانی‌شده‌ی هم‌اکنون فعال»

مانند زبان LISP

حوزه‌ی دید ایستا (لغوی)

برای ساختار بلاک‌بندی شده

بلاک
Block

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

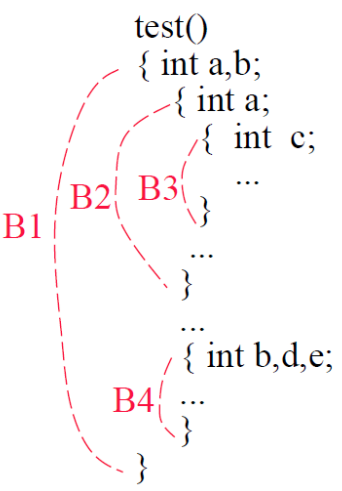
قاعده‌ی حوزه‌ی دید

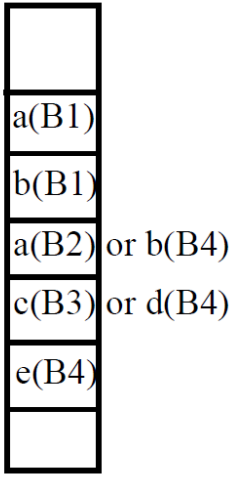
قاعده‌ی نزدیک‌ترین حوزه‌ی تودرتو
Most Closely Nested Rule

- 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.

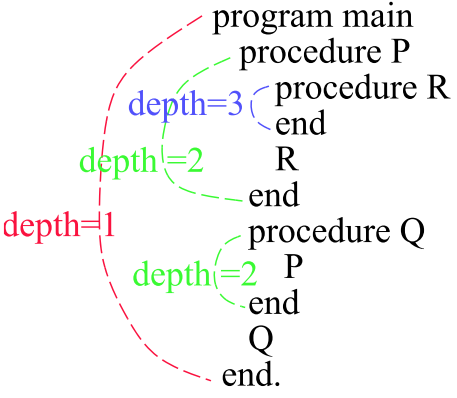
حوزه‌ی دید ایستا (لغوی)

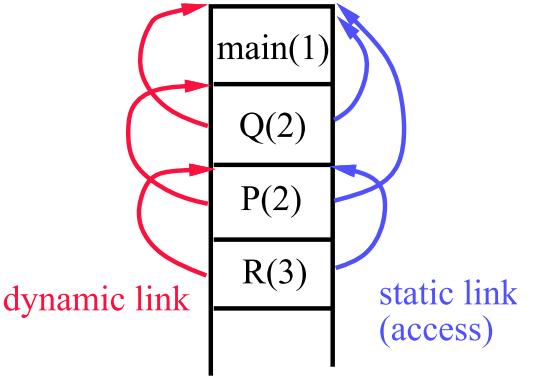
برای ساختار بلاک‌بندی شده، با روال‌های تودرتو

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 *depth*

- 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.

حوزه‌ی دید ایستا (لغوی)

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

تنظیم پیوندها

تنظیم پیوندهای پویا

به رکورد فعالیت روال فراخوانی‌کننده تنظیم می‌شود.

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.

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)

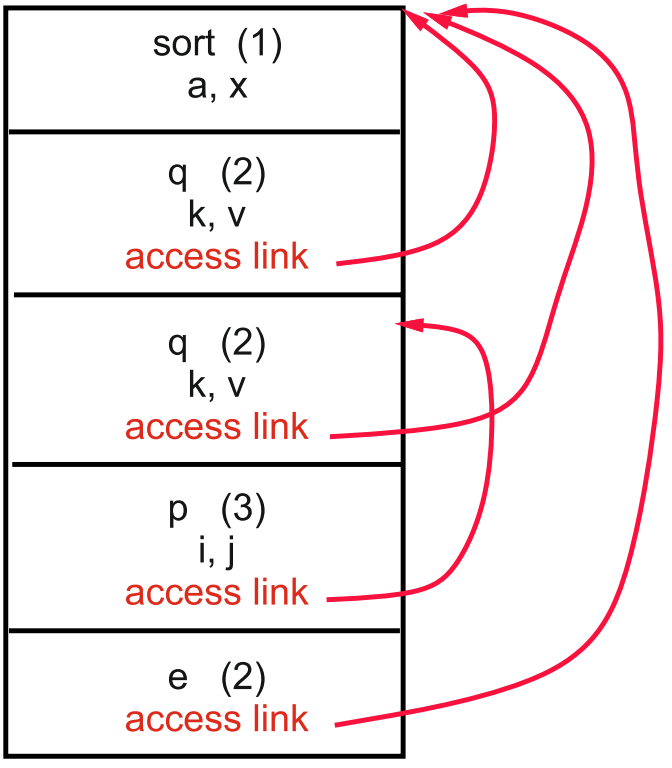
برای ساختار بلاک‌بندی شده، با روال‌های تودرتو: الگوریتم تنظیم پیوندها: مثال

```
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 ثبات‌ها	با استفاده از ناحیه‌ی داده‌های ایستا

وقتی روال P در عمق تودرتویی k فراخوانی شود:

$DISPLAY[1]$
$DISPLAY[2]$
⋮
$DISPLAY[k-1]$
$DISPLAY[k]$

$DISPLAY[1]$ تا $DISPLAY[k-1]$: حاوی اشاره‌گرهایی به رکورد فعالیت تازه‌ترین $k-1$ روالی که P را دربردارند.

$DISPLAY[k]$: حاوی اشاره‌گری به رکورد فعالیت P

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 غیرمحلی با استفاده از نمایشگرها (DISPLAY)

وقتی روال P در عمق تودرتویی k فراخوانی شود:	مقدار فعلی $DISPLAY$ در فیلد *save-display* رکورد فعالیت جدید ذخیره می‌شود.
	$DISPLAY[k]$ به‌گونه‌ای تنظیم می‌شود که به رکورد فعالیت جدید اشاره کند (مثلاً به فیلد *save-display* خود)
هنگام برگشت از روال	$DISPLAY[k]$ با استفاده از مقدار ذخیره‌شده در فیلد *save-display* بازیابی می‌شود.

حوزه‌ی دید ایستا (لغوی)

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

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

حوزه‌ی دید پویا

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.

مانند زبان LISP

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

لازمه: ذخیره‌سازی جدول نمادها برای استفاده در زمان اجرا

پیاده‌سازی

حوزه‌ی دید پویا
Dynamic Scope

دسترسی عمیق **(deep)**	دسترسی کم‌عمق **(shallow)**

حوزه‌ی دید پویا

دسترسی عمیق

پیاده‌سازی

حوزه‌ی دید پویا
Dynamic Scope

دسترسی عمیق **(deep)**	دسترسی کم‌عمق **(shallow)**

در صورت استفاده از یک متغیر غیرمحلی:
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 می‌شود؟

در صورت استفاده از حوزه‌ی دید ایستا،
این دستور مجاز نخواهد بود؛
زیرا هیچ حوزه‌ی احاطه‌کننده‌ای، x را اعلان نکرده است!

حوزه‌ی دید پویا

دسترسی کم‌عمق

پیاده‌سازی

حوزه‌ی دید پویا
Dynamic Scope

دسترسی کم‌عمق **(shallow)**	دسترسی عمیق **(deep)**

لیستی از متغیرهای جاری نگهداری می‌شود.

به هر نام متغیر موجود در برنامه، یک فضا تخصیص داده می‌شود.
(در ثبات‌ها یا در ناحیه‌ی داده‌های ایستا)
* یعنی یک فضا برای هر x حتی اگر چندین اعلان مختلف برای x وجود داشته باشد.

برای هر ارجاع به x، کد تولیدشده به یک مکان ثابت خاص مراجعه می‌کند.

مقادیر فعلی همه‌ی متغیرها اعلان‌شده توسط روال فراخوانی‌شونده در رکورد فعالیت آن روال ذخیره می‌شود.

وقتی کار روال تمام شد، آن مقادیر بازیابی می‌شوند.

حوزه‌ی دید پویا

مقایسه‌ی «دسترسی عمیق» با «دسترسی کم‌عمق»

دسترسی عمیق	دسترسی کم‌عمق
دسترسی کندتر به داده‌های غیرمحلی	دسترسی سریع‌تر به داده‌های غیرمحلی
سربار کمتر در ورود و خروج از روال‌ها	سربار بیشتر در ورود و خروج از روال‌ها (متناسب با تعداد متغیرهای محلی)

گذر دادن پارامترها به روال‌ها

PARAMETER PASSING

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

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

$$x := y$$

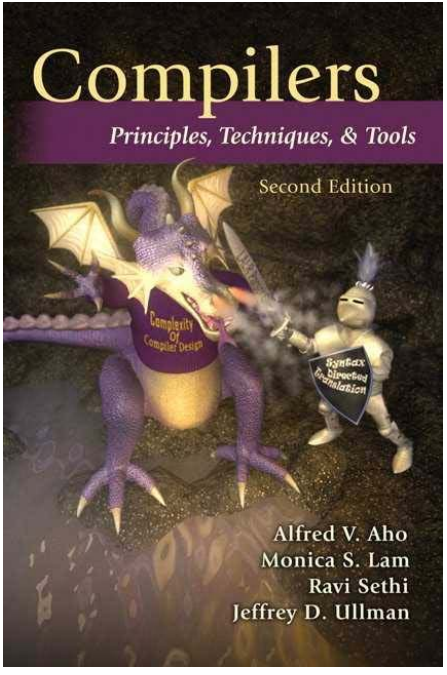
روش‌های گذر دادن پارامترها به روال‌ها			
فراخوانی با مقدار *Call by Value*	**فراخوانی با مرجع** *Call by Reference*	**فراخوانی با مقدار–نتیجه** *Call by Value-Result*	**فراخوانی با نام** *Call by Name*

سازمان حافظه‌ی زمان اجرا

۴

منابع

منبع اصلی



A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman,
Compilers: Principles, Techniques and Tools,
Second Edition, Addison-Wesley, 2007.

Chapter 7 (7.1-7.4)